Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicaciones

Desarrollo del portal web de gestión de licencias de RedBorder usando React y Node JS

Autor: David Señas Sanvicente
Tutor: Pablo Nebrera Herrera

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de las Telecomunicacones

Desarrollo del portal web de gestión de licencias de RedBorder usando React y Node JS

Autor:
David Señas Sanvicente

Tutor:
Pablo Nebrera Herrera
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Desarrollo del portal web de gestión de licencias de RedBorder usando React y Node JS

Autor: David Señas Sanvicente
Tutor: Pablo Nebrera Herrera

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia

Agradecimientos

En RedBorder he tenido mi primer acercamiento al mundo laboral. No puedo estar más satisfecho tras esta experiencia, la cual me ha ayudado a aprender, crecer y marcarme nuevos retos y objetivos en mi vida.

En primer lugar, me gustaría agradecer a mi familia el esfuerzo tan grande que ha realizado por mí, no solo durante el desarrollo de este proyecto, ni durante mi periodo académico, sino durante toda mi vida, ya que sin ellos no habría llegado a ser nada de lo que hoy soy, sin ellos nunca hubiera llegado hasta aquí, gracias a ellos hoy soy lo que soy.

También me gustaría darle las gracias a Claudia, mi pareja, por haber estado ahí siempre, haberme ayudado en mis días de agobio y desesperación cuando algo no salía y sobre todo haberme hecho ver siempre que yo podía.

Gracias a todos mis profesores, no solo a los que han estado durante mi etapa universitaria, sino a todas y cada una de esas personas que a lo largo de mi vida me han enseñado algo, por poco que sea, ya que de todo el mundo se puede aprender y todo el mundo nos ayuda a crecer en mayor o menor medida.

Gracias en especial a José Antonio Parra por haberme aportado todos sus conocimientos a la hora de realizar este proyecto, por su paciencia y su confianza en mí desde el primer minuto.

Gracias también a mi tutor, Pablo Nebrera Herrera, por haberme dado la oportunidad de pertenecer al equipo de RedBorder, sentir lo que es estar en una empresa real para tener esa oportunidad antes de enfrentarme a la vida laboral tras la vida académica.

¡GRACIAS A TODOS!

David Señas Sanvicente
Sevilla, 2017

Resumen

RedBorder® es un servicio encargado de la monitorización y gestión de eventos que suceden en una red, orientado principalmente a la seguridad. Los clientes de este servicio son, por lo general, organizaciones, las cuales disponen de una serie de licencias concedidas por RedBorder® para poder hacer uso de dicho servicio.
Durante el desarrollo de este proyecto se ha realizado un portal web, utilizando principalmente las tecnologías *React*, *Node JS*, *express* y *mysql*, para llevar a cabo la gestión licencias concedidas a los clientes de RedBorder®.
Dentro de este portal web cada organización tendrá una serie de usuarios que podrán conocer y descargar las licencias de las que dispone su organización, a la misma vez que podrán solicitar la creación de nuevas licencias.
La gestión de dichos usuarios y organizaciones será por parte de RedBorder® y se realizará mediante usuarios con privilegios de administradores.

Abstract

RedBorder® is a software for events management of a network, focused to security. RedBorder's clients are, generally, organizations, and they will have licenses for use this software.
In this project, it has been done a web portal with React technology where RedBorder's clients can management their licenses easily.
Inside of this web, each organization will have different users who can be able to see the different licenses of their organizations and request new licenses.
The management of different users and organizations will be done by the RedBorder's users.

Índice

1 INTRODUCCIÓN

RedBorder® es un servicio para la gestión de la seguridad, de código abierto, basado en tecnologías de Big Data desarrollado por la empresa de ENEO Tecnología.
Para que los clientes puedan hacer uso de dicho servicio han de tener una licencia activa concedida por la empresa. A lo largo de este proyecto se realizará un portal web de modo que estas licencias se puedan de un modo fácil e intuitivo.

1.1 Motivación y objetivo

Durante la asignatura de Seguridad de tercero de carrera conocí a Pablo Nebrera y me resultó bastante interesante su empresa y su forma de enseñar, por lo que hablé con él para ver si el año siguiente podría realizar tanto las prácticas de empresa, como mi TFG con él, y aceptó.

A comienzos de 2017 fui por primera vez a su empresa y estuvimos debatiendo sobre qué haría durante mis prácticas y mi TFG. Tras varias reuniones se comentó que actualmente las licencias de los clientes se solicitaban, o bien por email, o bien por teléfono, y tras ello se enviaban para que las pudiesen utilizar, algo bastante engorroso por lo que les propuse realizar un **portal web** que facilitara toda esa gestión tanto a los clientes como a la propia empresa, de modo que los clientes pudiesen solicitar licencias y RedBorder® las autorizase de un modo sencillo a través de esta web.

Es así como nació este proyecto, aunque he de decir que tras mi propuesta fue la propia empresa quien me aconsejó realizarlo en *React* ya que yo hasta el momento desconocía totalmente esa tecnología y también fue ella quien impuso los requisitos que finalmente deberá cumplir la aplicación con detalle.

1.2 Metodología de trabajo

Dado que realicé tanto las prácticas como el TFG en la propia empresa, durante las primeras semanas de prácticas estuve leyendo sobre *React*, informándome sobre esa tecnología y aprendiendo sobre ella a base de realizar pequeños ejemplos.

Tras estas primeras semanas de toma de contacto, comencé a trabajar siguiendo una metodología de trabajo conocida como *scrum.* Sin embargo, esta metodología está orientada para equipos de desarrollo de varias personas por lo que se tuvo que adaptar a mi trabajo de la siguiente forma:

- Todas las semanas, normalmente el martes, se realizaba una reunión con el jefe de desarrollo (*sprint planning*) en la que se marcaban los PBI[1] a cumplir antes del martes siguiente.
- Todos los días cuando llegaba a la empresa se realizaba lo que se conocen como *dailys*, que no son más que reuniones de no menos de 15 minutos en las que se debía responder a las siguientes preguntas ¿Qué hice ayer?, ¿Qué problemas tuve? y ¿Qué voy a hacer hoy?

Además, como apoyo de todo esto se hizo uso de una plataforma conocida como *Redmine* en la que se añadían todos los PBI semanales y según se iban realizando se marcaban como pendiente, en proceso, finalizado y verificado.

[1] PBI: Siglas en inglés de *product backlog item*

2 TECNOLOGÍAS EMPLEADAS

A lo largo de este capítulo se comentarán las diferentes tecnologías y plataformas que se han utilizado para llevar a cabo la realización del proyecto.

2.1 Tecnologías usadas para el desarrollo del código en el cliente (Front-end)

En este apartado, se hablará de las tecnologías que se han empleado para el desarrollo de la parte del lado del cliente de la aplicación (Interfaz de usuario, principalmente).

2.1.1 ReactJS

A lo largo de este proyecto se ha optado por basarse principalmente en una de las últimas tecnologías para el desarrollo de aplicaciones web: *ReactJS*.
ReactJS es una librería de código abierto creada por Facebook para la creación de las interfaces de usuario. Está basada en JavaScript y una de sus principales ventajas es que todo se basa en componentes creados por el desarrollador (u obtenidos de terceros), permitiendo asociar la vista de esos componentes a los datos, de modo que si se detecta algún cambio en los datos se cambiará la vista.

2.1.2 ¿Cómo funciona ReactJS?

ReactJS posee funciones que toman los cambios en los datos traduciéndolos en cambios de la vista, es decir, siempre que React detecte algún cambio de estado en alguno de sus datos volverá a ejecutar estas funciones para determinar una nueva representación virtual de la página que, a continuación, se traducirá en cambios en el DOM[2] para que aparezcan en el navegador del cliente.
A priori, esto suena más engorroso y lento que el enfoque JavaScript habitual de actualizar cada elemento según sea necesario. Sin embarro, ReactJS tiene un algoritmo muy eficiente para determinar las diferencias entre la representación virtual de la página actual y la nueva. Para ello hará uso de lo que se conoce como DOM virtual.

2.1.2.1 ¿Qué es el DOM virtual?

El DOM virtual de React no es más que un DOM almacenado en memoria, pero ¿Cómo lo usa ReactJS y qué ventajas tiene?
Como se ha comentado anteriormente, *ReactJS* analiza los cambios entre la vista actual y la siguiente vista cada vez que se detecten cambios en los datos, para ello hace uso del DOM virtual.
ReactJS creará en el DOM virtual la siguiente representación de la vista y la comparará con la vista actual del DOM del navegador, analizando las diferencias para poder así cambiar solo las partes de la interfaz que han cambiado.
Por ejemplo, supongamos que se tiene una lista con 2000 usuarios representados en el DOM del navegador, y para el usuario 1000 se realiza un cambio en su nombre. Bien, pues ReactJS lo que haría sería "mostrar" los 2000 usuarios en su DOM virtual y lo comparará con el DOM del navegador, apreciando que solo existe diferencia en el usuario 1000 y por tanto, en el DOM del navegador sólo se cambiaría ese usuario, haciéndolo de una forma más rápida y eficiente.

[2] DOM: Siglas en inglés de Document Object Model.

2.1.2.2 React y JSX

JSX es un pseudolenguaje que "mezcla" JavaScript y HTML, es decir, aparentan ser simples etiquetas HTML, pero con la posibilidad de ejecutar código dentro de ellas, definir componentes…
React ofrece la posibilidad de utilizar JSX, sin embargo, no es estrictamente necesario su uso, solo es muy recomendable ya que hace más fácil el desarrollo de aplicaciones con React.

2.1.3 React Router

React router es una librería que se utiliza con React en la parte del cliente para mostrar componentes distintos en función de la URL propiciándole al usuario la sensación de navegación por la web, mientras que solo se están mostrando unos u otros componentes.

2.1.4 React Bootstrap

React Bootstrap es un *framework* de desarrollo para el lado del cliente, mediante el cual podemos disponer de una serie de componentes ya definidos (con sus propios estilos de CSS), siendo así mucho más fácil y rápido el desarrollo de la interfaz de usuario en el lado del cliente, ya que simplemente hay que tomar el componente que necesitamos (importándolo de la librería y utilizándolo en la parte del código que se quiera).

2.1.5 Toastr

Toastr es una librería para JavaScript empleada para lanzar notificaciones en el navegador del cliente. Dichas notificaciones son configurables, pudiendo elegir notificaciones de error, de alerta, de éxito…

2.2 Tecnologías usadas para el desarrollo del código en el servidor (Back-end)

En este apartado, se hablará de las tecnologías que se han empleado para el desarrollo de la parte del lado del servidor de la aplicación (Servidor, rutas, base de datos…).

2.2.1 Node JS y Express

Para el desarrollo del lado del servidor se ha utilizado el entorno de ejecución de JavaScript *NodeJS* con el cual se ejecutará el servidor *express*.
Express está basado en *Connect*, el cual es un framework extensible para el manejo de servidores HTTP que provee funciones de alto rendimiento conocidos como *middleware*.
Las funciones middleware son funciones que tienen acceso al objeto de solicitud o petición (req), al objeto de respuesta (res) y a la siguiente función de middleware en el ciclo de solicitud/respuestas de la aplicación, es decir, una función middleware es una función "intermedia" por la que pasa la solicitud y la respuesta de la petición antes de llegar al fin del ciclo de solicitudes y respuestas.
Dentro de una función *middleware* se puede tanto ejecutar cualquier código, realizar cualquier modificación del objeto solicitud o respuesta, finalizar el ciclo de solicitudes/respuestas y/o invocar al siguiente *middleware* de la pila.

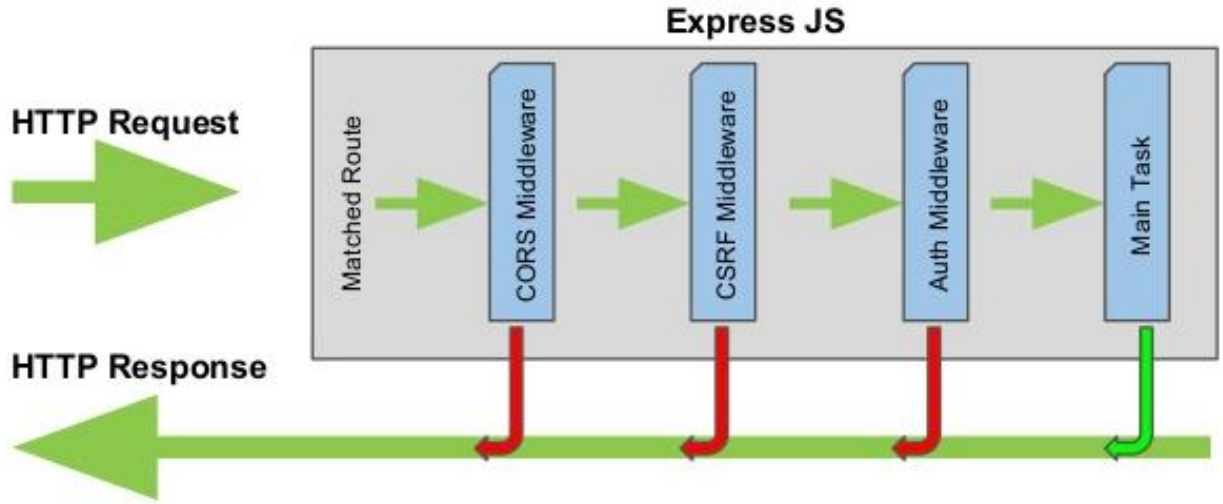


Figura 2-1. Funcionamiento de los *middlewares* de *Express JS*

Como se puede observar en la imagen anterior, en *express* cuando llega una petición *HTTP Request* pasa por todos los middleware que tengamos configurados hasta llegar al fin del ciclo. También podemos observar que cada *middleware* puede modificar el objeto petición (Request) o respuesta (Response).

2.2.2 Sequelize y Sequelize-fixtures

Sequelize es un módulo de *NodeJS* basado en ORM (Object-Relational mapping) que facilita el manejo de las bases de datos (MySql, Postgres, Mariadb, Sqlite y Mssq) permitiendo usar funciones en lugar de sentencias SQL anidadas y complejas.

Por otro lado, *Sequelize-fixtures* es un módulo que se utiliza para cargar datos en la base de datos a la hora de realizar tests para asegurarse así de que el estado inicial de la base de datos al realizar pruebas sea siempre el mismo.

2.2.3 MySql

MySql es la base de datos que se ha utilizado para almacenar los datos de la aplicación, sin embargo, se podría utilizar cualquier otra base de datos relacional compatible con Sequelize sin realizar muchos cambios.

2.2.4 Nodemailer

Nodemailer es un módulo de *node JS* que permite el envío de emails de una forma sencilla.
Permite el envío desde cualquier servidor ya que permite crear tu propio mecanismo de transporte (SMTP o similar), sin embargo, también dispone de una gran lista de servicios para el envío de correos electrónico ahorrando así al desarrollador el tener que crear un mecanismo de transporte para dichos servicios. En esta lista existen servicios como Gmail, Outlook, iCloud, SendCloud, SendPulse, SendGrip…

2.2.5 Nodemon

Nodemon es una utilidad para arrancar servidores (como por ejemplo *express*) de forma que automáticamente se reinicien tras detectar algún cambio en el código fuente.

Además, detecta errores sintáctico en dicho código (Por ejemplo errores de JavaScript) y por lo tanto es una herramienta indispensable para desarrolladores.

2.3 Tecnologías comunes al Front-end y Back-end

En este apartado, se hablará de las tecnologías que son comunes tanto a la parte del cliente como del servidor (test, gestor de librerías..).

2.3.1 Node package manager (npm)

NodeJS es un sistema fuertemente modular y cuando se instala, se instala prácticamente vacío. Para la adicción de módulos a *NodeJS* es para lo que se utiliza *npm*.
A lo largo del proyecto se ha utilizado npm para añadir las diferentes librerías, *frameworks* y servicios necesarios.
Otro uso que se le ha dado a npm es la ejecución de scripts mediante la configuración del fichero "package.json", como pueden ser los scripts de inicio del servidor, de inicio de WebPack, inicio de los test…

2.3.2 WebPack

WebPack es un empaquetador de módulos, es decir, nos permite tener el código fuente de una página web dividido en módulos, relacionados entre sí, los cuales se unirán en uno único gracias a su uso.

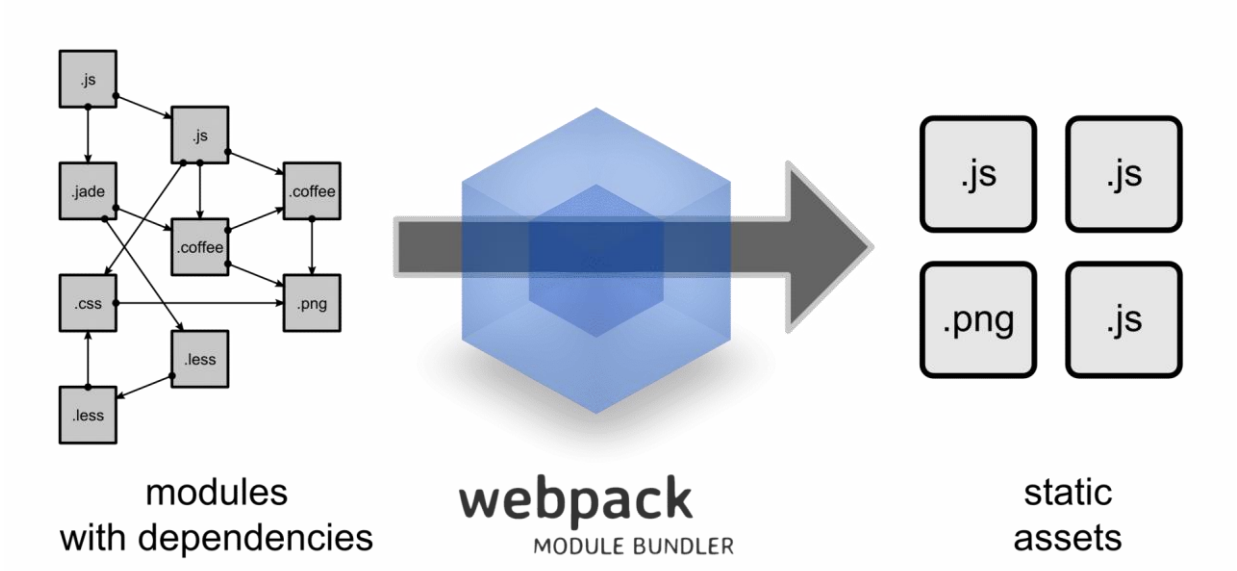


Figura 2-2. Funcionamiento de WebPack

WebPack no solo se encarga de empaquetar todos los módulos en uno solo, sino que también se encarga de cargar las partes necesarias en cada momento ya qué no siempre se van a necesitar todas y cada una de las partes de la aplicación, haciéndola así más eficiente.
Otro de los puntos fuertes de *WebPack*, y motivo por el cual se ha usado en este proyecto, es qué permite la transpilación de código EcmaScript qué todavía no es soportado por la mayoría de navegadores, como es el caso de ES6, a JavaScript puro mediante el loader: *babel*.
Babel no sólo se encarga de transpilar el código EcmaScript, sino que además también permite realizar la minificación del código, haciendo así que el fichero resultante ocupe menos memoria y sea menos legible por los humanos (dándole más privacidad, y por tanto seguridad, al código del cliente).

2.3.3 Mocha, Chai-http y Assert

Mocha es un framework de JavaScript qué puede ser implementado en *NodeJs* y permite la ejecución de pruebas unitarias en series.
Además, Mocha se ha utilizado a lo largo del proyecto junto con la librería de afirmaciones *assert* para la comprobar qué los test satisfacen determinadas condiciones.
Por otro lado, se ha hecho uso de la librería *chai-http* para las pruebas de las diferentes rutas del servidor.

2.3.4 JsonWebToken

Un Json Web Token es un contenedor de información codificado en base 64 que hace referencia a la autenticación de un usuario y posee 3 partes:

- La primera parte es la que se denomina "JOSE" (JavaScript Object Signing and Encryption) y se encarga de definir la tecnología utilizada para el cifrado de la información. No es más que un JSON que contiene el algoritmo usado y el tipo, por ejemplo: {"alg":"HS256","typ":"JWT"}
- La segunda parte es lo se denomina "información de negocio" y contiene datos relevantes sobre el usuario como puede ser su identificador, su nombre…
- La tercera parte es la más importante y se denomina "firma" la cual se encarga de dar validez y seguridad al token. Esta parte es un cifrado de las otras dos partes mediante el uso de un algoritmo de tipo HASH apoyado en una clave privada.

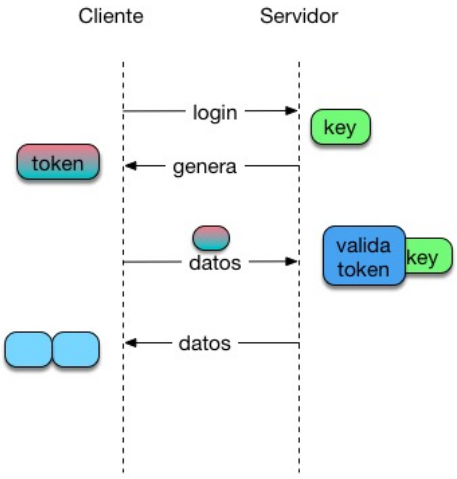


Figura 2-3. Uso de JSON Web Token

En la imagen anterior se puede observar cómo se genera y usa el token:

- En primer lugar, el cliente introduce su usuario y su contraseña (o se autentica de alguna manera similar: Facebook, GitHub…).
- Al recibirla el servidor, y comprobar que es correcta, generará el token en base64 haciendo uso de una "clave" para el firmado.
- El cliente recibe el token y lo almacena para próximas peticiones.
- Cuando el cliente realiza de nuevo una petición al servidor envía el token generado anteriormente.
- El servidor decodifica las partes del token de base 64 y usa su clave para verificar la firma del mismo.

2.3.5 Passport

Passport es un módulo externo de *NodeJS* orientado a la autenticación de usuarios y se basa en la utilización de *middlewares* permitiendo tener el código más ordenado y limpio a la vez que permite desarrollar tareas de autenticación de usuarios en la web de una forma rápida y sencilla.
Estas funciones *middlewares* se ejecutan cada vez que el cliente realiza la petición de inicio de sesión o creación de un usuario y dentro de ellas se definen lo que se conocen como *estrategias* encargadas, por ejemplo, de crear el token o almacenar el usuario en la base de datos.

2.3.6 Istanbul

Istanbul es una herramienta que permite medir qué cantidad del código está siendo testeada, algo muy útil para eliminar problemas futuros, ya que gracias a ella se podrá ver que todo el código funciona como es debido y no dejamos nada sin comprobar.
Esta herramienta es por tanto utilizada junto con aplicaciones de test (*mocha* en nuestro caso), ya que una vez pasados los test genera un directorio llamado *coverag*e que contendrá un HTML en el que se podrá visualizar todas y cada una de las partes del código que han sido testeada, qué líneas faltan por probar…

2.4 Servicios empleados

Además de las tecnologías descritas anteriormente también se han utilizado servicios en línea para un mejor desarrollo y metodología de trabajo.

2.4.1 GitHub

GitHub es una plataforma de desarrollo que permite alojar proyectos siguiendo el control de versiones de Git.
El control de versiones Git, como su propio nombre indica, permite llevar un control de los cambios producidos en un proyecto, esto es, permite anotar cada cambio, cada funcionalidad, deshacer uno o más cambios…
Además de esto, otra de las muchas funcionalidades de Git es la posibilidad de trabajar con *ramas*, lo cual es muy útil tanto para el trabajo en grupo (cada desarrollador trabaja en su propia *rama*), como para mantener una *rama* totalmente funcional (generalmente la *master*) al mismo tiempo que tienes otra en la que estás desarrollando, etc

2.4.2 Travis CI

Travis Ci es una plataforma que permite tener una integración continua del código ya que permite ejecutar los test definidos para nuestro proyecto en un momento determinado.
En general, funciona de forma conjunta con *GitHub*, es decir, asociamos un repositorio con *Travis* y se le dice dónde están alojados todos los test para dicho proyecto. Una vez hecho esto, cada vez que un usuario realiza cambios en el proyecto y los sube a *GitHub* se lanzarán todos los test en *Travis*, notificándose (por ejemplo, por email) en el caso de que no se hayan pasado los test, permitiendo así que cada actualización de código esté completamente testeada.

2.4.3 Codecov

Codecov es una plataforma que permite llevar un control de las partes del código que están siendo testeadas.
Por lo general se integra con *Travis* de tal forma que cada vez que se pasen los test se le envíe a *codecov* la información de qué partes de código han sido probadas y cuáles no, de tal forma que se pueda centralizar en una web y llevar un histórico.

3 REQUISITOS

A lo largo del siguiente capítulo se definirán todos los requisitos que debe cumplir la web para el manejo de las licencias de RedBorder®.
Todos estos requisitos fueron impuestos por parte de RedBorder® al inicio del proyecto.

3.1 Participantes en el sistema

3.1.1 Usuario

En el sistema existirán usuarios que dispondrán de la siguiente información:

- Nombre: Nombre del usuario completo
- Email: Email del usuario (único)
- Contraseña: Contraseña del usuario que debe tener entre 8 y 15 caracteres.
- Rol: Rol del usuario (Admin o Normal)

3.1.1.1 Roles de usuarios

En el sistema existirán dos roles distintos de usuarios, por un lado, estarán los usuarios con permisos de administrador, a los que a partir de ahora se les llamará "Usuario administrador", y los usuarios sin permisos a los que se les conocerá como "Usuario normal".

3.1.2 Organizaciones

En el sistema existirán organizaciones que dispondrán de la siguiente información:

- Nombre: Nombre de la organización.
- Email: Email del responsable de la organización para el envío de notificaciones (único).
- Cluster uuid: Identificador del cluster que posee dicha organización.
- Sensores: Conjunto de tipo de sensores que estarán disponibles para esa organización.

3.1.3 Licencias

En el sistema existirán licencias las cuales contendrán los siguientes campos:

- Licencia UUID: Identificador de una licencia
- Fecha de expiración: Fecha hasta la cual una licencia está disponible
- Límite de bytes: Número máximo de bytes que podrá cursar esa licencia al día
- Lista de sensores: Lista con los tipos de sensores y la cantidad de cada uno permitidos en esta licencia

3.1.4 Relaciones entre usuarios, organizaciones y licencias

Cada usuario podrá pertenecer a una organización, o no (puede haber usuarios administradores que no pertenezcan a ninguna organización), y por tanto una organización estará formada por usuarios.
Por otro lado, las licencias pertenecerán siempre a una única organización y deben tener la referencia del usuario que la solicitó.

3.2 Inicio de sesión

Un usuario registrado previamente en el sistema podrá acceder mediante su email y su contraseña para realizar las diferentes acciones que a continuación se describen.

3.3 Gestión de perfil

Un usuario, una vez que ha iniciado sesión, podrá gestionar su perfil, pudiendo:

- Cambiar su nombre
- Cambiar su email
- Cambiar su contraseña

Siempre que se haga alguna modificación en el perfil se solicitará, de nuevo, la contraseña actual por motivos de seguridad.

3.4 Renovación de contraseña

Un usuario que haya olvidado su contraseña podrá solicitar una renovación rellenando un campo con su email, al cual se le enviarán las instrucciones a seguir para crear una nueva contraseña.
Una vez solicitada la renovación de una contraseña, el usuario dispondrá de una hora para llevarla a cabo.
Finalmente, cuando se reestablezca la nueva contraseña, se le notificará al usuario por email.

3.5 Creación de usuarios

Un usuario **administrador** podrá crear usuarios (tanto administradores como normales) a través de un formulario de creación de usuario que solicitará los siguientes datos relativos al usuario a crear:

- Nombre completo
- Email
- Contraseña
- Confirmación de la contraseña
- Organización a la que pertenece, si procede

Tras la creación se le enviará un email del usuario que ha sido creado con su contraseña y pidiéndole que, por favor, la cambie por su seguridad.

3.6 Creación de organizaciones

Un usuario **administrador** podrá crear organizaciones a través de un formulario de creación de organizaciones que solicitará los siguientes datos relativos a la organización a crear:

- Nombre
- Email
- Cluster uuid
- Sensores disponibles para esa organización

3.7 Creación de licencias

Un usuario **normal** podrá crear licencias para **su** organización a través de un formulario que solicitará los siguientes datos:

- Tiempo de duración de la licencia
- Límite de bytes al día para dicha licencia
- Número de cada tipo de sensores disponibles para esa organización

Los usuarios **administradores** podrán crear licencias para **cualquier** organización.

3.8 Listado de usuarios

Un usuario **administrador** podrá conocer la lista de todos los usuarios existentes en el sistema, mostrando el nombre y el email de cada uno de ellos.

3.8.1 Usuarios pertenecientes a cada organización

Un usuario **administrador**, además, podrá conocer los usuarios de los que dispone cada organización si así lo desea.

3.9 Listado de organizaciones

Un usuario **administrador** podrá listar todas las organizaciones que existen en el sistema, mostrando el nombre, el email, el identificador del cluster y el número de usuarios de los que dispone cada organización

3.10 Listado de licencias

Un usuario **normal** podrá listar las licencias existentes para su organización, mostrando el identificador, el límite de bytes, el estado (activada o no), el tiempo de validez y la lista con la cantidad de sensores de los que dispone.
Un usuario **administrador**, además, podrá listar las licencias de cada organización.

3.11 Edición de un usuario

Un usuario **administrador** podrá editar el nombre, el email, la organización a la que pertenece y el rol de cualquier usuario.

3.12 Eliminación de un usuario

Un usuario **administrador** podrá eliminar a cualquier otro usuario del sistema. También se podrá eliminar a sí mismo, cerrándose así su sesión.

3.13 Edición de una organización

Un usuario **administrador** podrá editar el nombre, el email y el cluster uuid de cualquier organización del sistema.

3.14 Eliminación de una organización

Un usuario **administrador** podrá eliminar cualquier organización existente en el sistema. En dicho caso se borrarán también todas las licencias asociadas a esa organización y los usuarios que pertenecían a dicha organización ahora no pertenecerán a ninguna.

3.15 Activación de una licencia

Cuando un usuario (**normal** o **administrador**) crea una licencia, se creará como "desactivada", hasta que un usuario **administrador** la active.
Cuando se activa una licencia, la duración de la misma será desde el momento de la activación hasta pasado el tiempo de duración con el cual se creó la licencia.
Tras la activación de la licencia se le enviará un email al responsable de la organización (email de la organización) notificándole de que la licencia ha sido activada hasta el día de expiración que le corresponda.

3.16 Extensión de una licencia

Un usuario **normal** podrá extender licencias activas de su propia organización.
Un usuario **administrador** podrá extender licencias activas de cualquier organización.
Al extender una licencia se le sumará el tiempo de extensión solicitado a la fecha de expiración de la licencia que se está ampliando, es decir, si ampliamos una licencia que caduca el día 20 de abril de 2018 por un año, su fecha de expiración será **siempre** el día 20 de abril de 2019, indiferentemente de cuándo se solicite o active.
Al igual que para en la creación de licencias, no estará disponible hasta que un administrador la active.

3.17 Descarga de una licencia

Una licencia podrá ser descargada por los usuarios de la organización a la que pertenezca, siempre que haya sido activada previamente por un administrador.
Un usuario administrador podrá descargar cualquier licencia de cualquier organización que esté activa.
La licencia se descargará en un archivo con extensión *.lic*. La forma de generación de dicho fichero se explicará detalladamente más adelante.

4 IMPLEMENTACIÓN

A lo largo del siguiente capítulo se explicará detalladamente cómo se han resuelto todos y cada uno de los requisitos anteriores, cómo se ha organizado el código de la aplicación y cómo se han configurado cada una de las tecnologías empleadas para que funcionen correctamente en este proyecto.

4.1 Configuración inicial

Antes de comenzar con el desarrollo del proyecto se debe crear un directorio que contenga todo lo necesario para poder trabajar de una forma eficiente y cómoda.
Para ello se han seguido los siguientes pasos:

4.1.1 Configuración del repositorio en GitHub

En primer lugar, se ha creado un nuevo repositorio dentro de la organización de RedBorder en GitHub para llevar un seguimiento de todo el proyecto, llevar un control de cambios…
A lo largo del desarrollo de la aplicación se han ido creando diferentes ramas para cada funcionalidad distinta, se ha trabajado sobre ella, realizando *commits*[3] y, antes de pasar a la siguiente funcionalidad (una vez testeada la rama actual), se ha realizado un *pull request*[4] hacia la rama *master*[5], para finalmente hacer un *merge*[6] de la rama actual con la rama *master*.
Esto es lo que se conoce como método de trabajo *Git Flow* en el cual hay dos ramas principales: una rama master totalmente activa (se podría desplegarla en cualquier momento) y una rama en desarrollo.
El repositorio en el que se encuentra todo el código de la aplicación es el siguiente: https://github.com/redBorder/licensing-management

4.1.2 Creación del esqueleto de la aplicación

Una vez creado el repositorio en GitHub se ha creado un directorio en local que estará asociado a dicho repositorio.
Este directorio, inicialmente, parte con la siguiente estructura:

- Un directorio *client* en el cual existirá otro directorio llamado *src* con todo el código fuente para el desarrollo de la parte del cliente (Front-end).
- Un directorio *config* en el que se almacenarán los datos de configuración necesarios que se tomarán por defecto.
- Un directorio *server* que contendrá todo el código fuente necesario para el desarrollo de la parte del servidor (Back-end).
- Un directorio *test* que contendrá todos los test que ha de pasar la aplicación.
- Un fichero oculto llamado *gitignore* que contendrá el nombre de los directorios y/o ficheros que no se quieren controlar con Git.

Estos son los directorios iniciales con los que se inicia la aplicación, sin embargo, a lo largo del proyecto se irán creando nuevos.

---

[3] *Commit*: Término en inglés que hace referencia a la acción de asentar un cambio (hacer definitivo) en Git.
[4] *Pull request*: Término en inglés que hace referencia a la solicitud para que una rama se fusione con otra en Git.
[5] Rama *master*: Rama principal de Git, en la cual debe estar la última version funcional de la aplicación.
[6] *Merge*: Término en inglés que hace referencia a la acción de fusionar una rama con otra en Git.

A medida que vaya siendo necesario se irán explicando cada uno de los nuevos.

4.1.3 Inicialización del proyecto mediante NPM

Para iniciar el proyecto se ha ejecutado el comando `npm install`.
Este comando solicitará datos relativos al proyecto, tales como el nombre, el autor, la versión, el tipo de licencia, una descripción… y creará, en el directorio raíz, el fichero *package.json* que contendrá dicha información.
Desde este momento se dispondrá del fichero *package.json* en el que se irá añadiendo toda la información relativa a *npm.*
Todas las dependencias que hayan sido necesarias a lo largo del proyecto han sido instaladas mediante el comando npm install --save <dependencia> o bien npm install --save-dev <dependencia> si solo es necesaria para desarrollo. A lo largo de la memoria se irá comentando cuando ha hecho falta la instalación de alguna decencia.

4.1.4 Configuración de WebPack

En el desarrollo del código de la parte del cliente se ha optado por ayudarse de *WebPack*, que como se dijo en el apartado de tecnologías empleadas, se utilizará para poder tener el código divido en módulos y que WebPack se encargue de agrupar todo en un único fichero.
Para que *WebPack* realice esto de forma correcta es necesario, además de instalarlo para desarrollo con npm junto con *babel-loader*, crear el siguiente fichero de configuración:

Código 4-1. Fichero de configuración de WebPack. (.webpack.config.js)

```
const path = require('path');

process.noDeprecation = true;

module.exports = {
  // El punto de entrada estará en la siguiente ruta
  entry: path.join(__dirname, '/client/src/index.jsx'),

  // Nombre y directorio del fichero de salida
  output: {
    path: path.join(__dirname, '/client/dist/js'),
    filename: 'app.js',
  },

  module: {
    // Loaders utilizado para la transpilación
    loaders: [{
      test: /\.jsx?$/,
      include: path.join(__dirname, '/client/src'),
      loader: 'babel-loader',
      query: {
        presets: ["react", "es2015"]
```

```
      }
    }],
  },
};
```

En el fichero anterior se realizan las siguientes configuraciones:

1) Se le da un punto de entrada que en este caso será el fichero client/src/index.js.
2) Se le da un punto de salida donde guardar el fichero generado por WebPack y el nombre de dicho fichero: client/dist/js/app.js.
3) Se le dan una serie de *loaders* que se encargan de realizar la *transpilación* a JavaScript puro.

Una vez se ha configurado WebPack correctamente, para lanzarlo habría que ejecutar el comando `WebPack` desde el directorio raíz del proyecto. Además, WebPack tiene la opcion de lanzarlo de modo que si detecta algún cambio en algún fichero vuelva a ejecutarse, esta es la opción `-w`.
Para facilitar el arranque de WebPack tanto en uno u otro modo se ha creado entradas de script en el fichero package.json de tal modo que si se ejecuta `npm run build` se lanzará webpack una sola vez y si se ejecuta `npm run build:dev` se lanzará webpack con la opción `-w`.

4.2 Creación de modelos en la base de datos

Para la definición y creación de modelos en mysql se ha utilizado *Sequelize.* A continuación, se explicará cómo se ha definido cada modelo y las relaciones entre ellos para cumplir los requisitos impuestos por RedBorder® para los usuarios, las organizaciones y las licencias utilizando dicha tecnología.
Todo el código relativo a los modelos se halla en el directorio server/models, en el cual existen 4 ficheros, uno por cada modelo y el principal encargado de definir las relaciones entre ellos.

4.2.1 Modelo de Usuario

En el siguiente fichero, user.js, se define el modelo para un usuario:

Código 4-2. Fichero de definición del modelo para un usuario. (server/models/user.js)

```
const passwordHash = require('password-hash');
const DataTypes = require('sequelize/lib/data-types');

module.exports = function(sequelize) {
  const User = sequelize.define('User',
      {
        id:{
            type : DataTypes.UUID,
            primaryKey: true,
            defaultValue: DataTypes.UUIDV4,
            validate: {
```

```
                isUUID: 4
            }
        },
        name: {
            type: DataTypes.STRING,
            aallowNull: false,
            validate: {
                notEmpty: { msg: "Field name shouldn't be empty" }
            }
        },
        email: {
            type: DataTypes.STRING,
            allowNull: false,
            unique: true,
            validate: {
                isEmail: { msg: "Email should be a correct email" },
                notEmpty: {msg: "Field email shouldn't be empty'"},
            }
        },
        hashed_password: {
            type: DataTypes.STRING,
            validate: {
                notEmpty: {msg: "Field password shouldn't be empty"},
                not: {args: ["wrong_password"], msg: "Format password is
incorrect. Password should be between 8 and 15 alphanumeric characters"}
            }
        },
        role: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                isIn: {
                    args: [['normal', 'admin']],
                    msg: "Rol user must be normal or admin"
                    }
            }
        },
        resetPasswordToken: {
            type: DataTypes.STRING,
```

```
            allowNull: true
        },

        resetPasswordExpires:{
            type: DataTypes.DATE,
            allowNull: true
        }

    },
    //Expansion del modelo usuario
    {
        setterMethods   : {
            password: function(password){
                if(password.length > 15 || password.length < 8 || typeof
password != "string"){
                    this.setDataValue('hashed_password',
"wrong_password"); //Lanza error si la contraseña mide lo que no debe
                }
                else{
                    this.setDataValue('hashed_password',
passwordHash.generate(password));
                }
            },
            email: function(email){
                this.setDataValue('email', email.toLowerCase());
            }

        },
        getterMethods : {
            password: function(){
                return this.hashed_password;
            }
        },
        instanceMethods: {
            verifyPassword: function(password){
                return                      passwordHash.verify(password,
this.hashed_password);
            },
            changePassword: function(password, new_password){
```

```
                if(passwordHash.verify(password, this.hashed_password)){
                    this.setDataValue('hashed_password',
passwordHash.generate(new_password));
                    return true;
                }
                else
                    return false;
            }
        }
    });
    return User;
}
```

En el fichero anterior se definen, en primer lugar, cada uno de los campos que tendrá un usuario junto con el tipo y las restricciones de validación, las cuales lanzarán un mensaje de error si no se cumplen.
Se puede observar que existe el campo que identifica a un usuario, los campos que se requieren por parte de RedBorder (nombre, email, contraseña y rol) y dos campos más que será utilizado para recuperar la contraseña de un usuario (Se explicarán más adelante).
Por otro lado, se han definido métodos *setter* y *getter*. Estos métodos se utilizan de forma "transparente", es decir, cuando a través de *sequelize* se solicita el campo "password" se activará el método *getter* correspondiente a "password", devolviéndose así el campo "hashed_password". Del mismo modo sucede con los métodos *getter*, si intentamos modificar el campo "password", se llamará al método *setter* de "password" y se guardará la contraseña encriptada si tiene una longitud correcta. Para el campo "email" se guarda siempre todo en minúsculas gracias al método *setter* correspondiente.
Por último, se han definido dos métodos de instancia, el primero de ellos, *verifyPassword*, comprueba que la contraseña que se le ha pasado coincida con la almacenada, y el segundo de ellos, *changePassword*, cambia la contraseña del usuario siempre y cuando la antigua contraseña sea correcta.

4.2.2 Modelo de organización

En el siguiente fichero, organization.js, se define el modelo para una organización:

Código 4-3. Fichero de definición para el modelo de una organización. (server/models/organization.js)

```
const DataTypes = require('sequelize/lib/data-types');

module.exports = function(sequelize) {
  const Organization = sequelize.define('Organization',
      {
        id:{
            type : DataTypes.UUID,
            primaryKey: true,
            defaultValue: DataTypes.UUIDV4,
            validate: {
```

```
                isUUID: 4
            }
        },
        cluster_id: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                notEmpty: { msg: "Field cluster id shouldn't be empty" }
            }
        },
        name: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                notEmpty: { msg: "Field name shouldn't be empty" }
            }
        },
        email: {
            type: DataTypes.STRING,
            allowNull: false,
            unique: true,
            validate: {
                isEmail: { msg: "Email should be a correct email" },
                notEmpty: {msg: "Field email shouldn't be empty'"},
            }
        },
        sensors: {
            type: DataTypes.STRING, //Lista de sensores separados por ;
            allowNull: false,
            validate: {
                notEmpty: {msg: "Field sensors shouldn't be empty'"},
            }
        }

    },
    //Expansion of the model user
    {
        setterMethods   : {
```

```
            email: function(email){
                this.setDataValue('email', email.toLowerCase());
            }
        }
    });
    return Organization;
}
```

En este fichero se puede observar que se definen los campos para una organización de tal modo que cumplan los requisitos impuestos por RedBorder, además también se define el campo que identifica a una organización.
En este caso se define un único método *setter*, que al igual que para un usuario, se asegura de guardar siempre el email en minúsculas.

4.2.3 Modelo de licencia

En el siguiente fichero, license.js, se define el modelo para una licencia:

Código 4-4. Fichero de definición del modelo para una licencia. (server/models/license.js)

```
const DataTypes = require('sequelize/lib/data-types');

module.exports = function(sequelize) {
  const License = sequelize.define('Licenses',
      {
        id:{
            type : DataTypes.UUID,
            primaryKey: true,
            defaultValue: DataTypes.UUIDV4,
            validate: {
                isUUID: 4
            }
        },
        license_uuid : {
            type : DataTypes.UUID,
            allowNull: false,
            defaultValue: DataTypes.UUIDV4,
            validate : {
                notEmpty: {
                    msg: "Field licenses uuid shouldn't be empty"
                }
            }
        },
```

```
        duration: {
            type: DataTypes.INTEGER,
            allowNull: false,
            validate: {
                notEmpty: {
                    msg: "Field duration shouldn't be empty"
                }
            }
        },
        expires_at: {
            type: DataTypes.DATE,
        },
        sensors : {
            type: DataTypes.JSON,
            allowNull: false,
            validate:{
                notEmpty: {msg: "Field sensors shouldn't be empty"}
            }
        },
        limit_bytes : {
            type: DataTypes.INTEGER,
            allowNull: false,
            validate : {
                notEmpty: {
                    msg: "Field limit bytes shouldn't be empty"
                }
            }
        },
        enabled : {
            type: DataTypes.BOOLEAN,
            allowNull: false,
            defaultValue: false,
            validate : {
                notEmpty: {
                    msg: "Field enabled shouldn't be empty"
                }
            }
        }
```

```
    });
    return License;
}
```

En este fichero se puede observar que se definen los campos para una licencia, sin embargo, hay que hacer algunas aclaraciones ya que no solo existen los campos requeridos por RedBorder®:
Existen dos campos de "identificación", es decir, el campo "id" y el campo "license_uuid", lo cual es debido a que una licencia, cuando se amplía, creará una nueva entrada, pero el campo "license_uuid" será el mismo y por tanto han de diferenciarse por el identificador global (campo "id").
Por otro lado, se puede observar que existe un campo "duration" y un campo "expires_date", esto es así porque cuando se crea una nueva licencia lo que se hará será rellenar el campo "duration" con el número de meses para el cual haya sido creada y una vez se active la licencia se rellenará el campo "expires_date" con el valor de la fecha actual más el número de meses de duración de esa licencia (campo "duration").
Finalmente se ha creado un campo "enabled" que indica si una licencia está o no activada.

4.2.4 Relaciones entre los modelos

Por último, para crear las relaciones entre los diferentes modelos existe un fichero llamado index.js dentro del directorio server/models que realiza lo siguiente:

Código 4-5. Fichero con las relaciones entre los modelos. (server/models/index.js)

```
module.exports = function (sequelize){
    //Importamos todos los modelos, hacemos las relaciones entre ellos y
los devolvemos
    const User = require('./user')(sequelize);
    const Organization = require('./organization')(sequelize);
    const License = require('./license')(sequelize);
    //Un usuario pertenece a una organización
    User.belongsTo(Organization);
    //Una oganización tiene muchos usuarios o ninguno en caso de que sea
null la clave externa
    Organization.hasMany(User);
    //Una licencia es creada por un usuario ("pertenece" a un usuario).
    //Si el usuario se borra, se pierde la referencia al usuario
poniendose el campo UserId a null
    License.belongsTo(User);
    //Una licencia pertenece a una organizacion, y no puede no pertener a
ninguna (no puede ser null)
    //Si una organización se borra, se borrarán todas las licencias
asociadas a ella
    Organization.hasMany(License,   {foreignKey:   {allowNull:   false},
onDelete: 'Cascade'});
    return {
        User: User,
```

```
        Organization: Organization,
        License: License
    };
};
```

En este fichero lo que se realiza es una exportación de una función que recibe el parámetro "sequelize", el cual será el objeto de la inicialización de *Sequelize*, y realiza las distintas relaciones entre los distintos modelos impuestos por RedBorder.
Además, para la relación entre licencias y organizaciones se ha añadido en primer lugar la imposibilidad de que no pueda existir una licencia sin organización (no se permite que la clave externa sea *null*) y la cláusula "Cascade" para que en caso de eliminación de una organización se borren todas las licencias que pertenezcan a dicha organización.

4.2.5 Inicialización de la base de datos

La base de datos elegida para el proyecto ha sido *mysql*, sin embargo, gracias a *sequelize* esto se podría cambiar en cualquier momento ya que ofrece una forma genérica para conectarnos a una base de datos.
En este caso, se ha creado un fichero para la inicialización de la base de datos dentro del directorio server/db llamado index.js. En este fichero se configurará los datos necesarios para que *sequelize* se conecte con *mysql*:

Código 4-6. Fichero de conexión a la base de datos mediante *sequelize*. (server/db/index.js)

```
const Sequelize = require('sequelize')
const config = require('../../config/config.json')
const MODE_RUN = process.env.MODE_RUN || "development"
const DB_config = config[MODE_RUN]

const sequelize =
new Sequelize(process.env.DB_NAME || DB_config.database,
              process.env.DB_USER || DB_config.user,
              process.env.DB_PASSWORD || DB_config.password,
        //Options
        {
        dialect: process.env.DB_SERVER || DB_config.BD,
        host: process.env.DB_HOST || DB_config.host,
        port:   process.env.DB_PORT || DB_config.port,
        logging: (process.env.DB_LOG=="true" || DB_config.log =="true") ?
console.log : false
});

//Cargamos los diferentes modelos
const models = require('../models')(sequelize);
```

```
const connectDB = () => {
    if(process.env.MODE_RUN == "test"){
        sequelize.sync({force:true}).then( () => {
            console.log("Connected to DB");
        }, (err) => {
            console.log("Error connecting DB, retrying...");
            setTimeout(connectDB, 5000);
        })
    }
    else{
        sequelize.sync().then(() => {
            console.log("Connected to DB");
            //If there aren't users, create one admin user by default in
production mode...
            models.User.findAll({where: {
                role: "admin"
            }})
            .then((users) => {
                if(users.length == 0){
                    const NewUser = models.User.build({
                      name: "Admin",
                      email: "admin@redborder.com",
                      password: "adminadmin",
                      role: "admin"
                    })
                NewUser.save().then(() => {
                    console.log("New default admin user created.");
                    console.log("   Email: admin@redborder.com");
                    console.log("   Password: adminadmin");
                    console.log("Please, change this user profile");
                });
              }
            })
        }, (err) => {
            //Sequelize error
            console.log("Error connecting DB, retrying...")
            setTimeout(connectDB, 5000);
        });
```

```
    }
}


module.exports.sequelize=sequelize;
module.exports.connectDB=connectDB;
```

En este fichero se realizan varias acciones:

- En primer lugar, se importa la configuración que se encuentra en el fichero de configuración situado en el directorio config, en función de la variable MODE_RUN. Esta variable hace referencia a cómo está funcionando la aplicación, si en modo de producción, en modo de desarrollo o en modo test.
- En segundo lugar, se crea el objeto *sequelize* que gestionará la conexión a la base de datos. Para su configuración toma una serie de parámetros de las variables de entorno y si no están disponibles, las obtiene del fichero de configuración:
    1) El nombre de la base de datos: DB_NAME.
    2) El usuario con el que se conectará a la base de datos: DB_USER.
    3) La contraseña del usuario anterior: DB_PASSWORD.
    4) La base de datos a la que se conectará (mysql, postgree…): DB_SERVER.
    5) La dirección ip donde está la base de datos: DB_HOST.
    6) El puerto de escucha de la base de datos: DB_PORT.
    7) Si se quiere, o no, que muestre información sobre las distintas sentencias SQL que se están ejecutando: DB_LOG.
- Una vez creado el objeto *sequelize*, se llama a la función que se definió previamente para cargar los modelos (en el fichero server/models/index.js) pasándole el objeto *sequelize* como parámetro.
- Por último, existe una función que nos permite conectarnos a la base de datos mediante el uso del método `sync` de *sequelize*. En esta función se realiza lo siguiente:
    1) Se comprueba en qué modo se está ejecutando la aplicación mediante la variable de entorno MODE_RUN
    2) Si se está ejecutando en modo test, se conectará a la base de datos, pero con el parámetro `force=true`, es decir, se forzará a recrear los modelos de tal forma que la base de datos esté en dicho caso siempre vacía.
    3) Si no se está ejecutando en modo test, se comprueba si existe algún usuario administrador en el sistema, y si no existe, se creará uno por defecto (usuario: admin@redboder.com y contraseña: adminadmin) de tal forma que la primera vez que se lance la aplicación se pueda acceder con este usuario.
    4) Por último, esta función en el caso de no poder conectarse con éxito a la base de datos, pasado 5 segundos, se llama a sí misma para volver a intentarlo. De este modo si la base de datos tarda un tiempo en estar activa se seguirá intentando hasta que la base de datos esté disponible. Esto será útil para la *dockerización.*

Se puede observar, que al final del fichero se exporta tanto el objeto *sequelize* como la función *ConnetDB* para que puedan ser utilizados desde otras partes del proyecto mediante importación.

4.3 Inicio de sesión

Para llevar a cabo la implementación del inicio de sesión de un usuario en el portal web, se ha decidido optar por la utilización de la librería *passport*, la cual permitirá la creación de usuarios y la autenticación de los mismos, y la librería *jsonwebtoken*, que permitirá la creación de un *token* encargado de mantener la sesión del usuario.

4.3.1 Código de back-end

Para cumplir con el requisito para el inicio de sesión de un usuario, en primer lugar, se ha tenido que crear código de back-end (código interpretado en el lado del servidor) que registre usuarios y compruebe si la autenticación es correcta.
A continuación, se explicará detalladamente cómo se lleva a cabo el inicio de sesión en el lado del servidor utilizando *passport* y *jsonwebtoken*:

4.3.1.1 Estrategias de *passport*

Passport es un módulo de node js que permite a los clientes autenticarse, bien mediante proveedores de autenticación externos (Facebook, GitHub…), o bien mediante usuarios y contraseñas definidos localmente.
La forma de decirle a *passport* cómo se quiere que los usuarios se autentiquen es mediante el uso de lo que se conoce como *estrategias*.
Existen una gran cantidad de *estrategias* ya definidas por *passport* para la autenticación externa, sin embargo, en este proyecto no se quiere una autenticación externa, sino que lo que se desea es que existan usuarios y contraseñas en local mediante los cuales se pueda iniciar sesión.
Para hacer esto, existe un módulo complementario a *passport* que ofrece la posibilidad de crear *estrategias* locales. Este módulo es *passport-local* y a continuación se explicará cómo se ha incorporado al proyecto.
En primer lugar, se ha definido una estrategia encargada de la creación de un usuario, lo que generalmente se conoce como "registro de un usuario". Para ello, en el directorio server/passport, se ha creado el fichero "local-signup.js" que contiene lo siguiente:

Código 4-7. Fichero con la estrategia para la creación de un usuario. (server/passport/local-signup.js)

```
const PassportLocalStrategy = require('passport-local').Strategy;
//Incializamos sequelize
const sequelize = require('../db').sequelize;
//Cargamos los modelos
const models = require('../models')(sequelize);


module.exports = new PassportLocalStrategy({
  usernameField: 'email', //Por defecto busca username, pero mi usuario será el email
  passwordField: 'password', //La contraseña será el password
  session: false,
  passReqToCallback: true //Para poder leer el nombre del body
}, (req, email, password, done) => {
  const NewUser = models.User.build({
      name: req.body.name.trim(),
```

```
      email: email.trim(),
      password: password.trim(),

OrganizationId:(req.body.organization==''||req.body.organization=="No") ?
null : req.body.organization,
      role: req.body.role
    });
  NewUser.save().then(function(NewUser) {
      return done(null);
    }, function(err){
      return done(err);
  });
});
```

Como se puede observar, en el fichero anterior se realizan las siguientes acciones:

1) En primer lugar, se importa (de *passport-local*) la clase *"strategy"*.
2) Se importan también los modelos de la base de datos, a través de sequelize, para poder así almacenar un usuario en la tabla "users".
3) Por último, se crea una estrategia para el registro de un usuario nuevo.

El constructor de la clase *strategy* acepta dos parámetros, el primero de ellos son las opciones y el segundo es la función de *callback* que se ejecutará en esta estrategia. En el campo de opciones se puede configurar con qué campo el usuario iniciará sesión y qué campo será la contraseña, ya que por defecto sería "username" y "password".
En este caso se puede observar que el usuario iniciará sesión con su email y su contraseña. Los otros dos parámetros se utilizan para poder leer del cuerpo del mensaje los datos relativos al usuario a crear.
Por otro lado, en la función de *callback* se realiza el almacenado del usuario en la base de datos, con los datos que están en el cuerpo del mensaje (datos que el cliente ha de enviar en el formulario de creación de un usuario).
Una vez definida la estrategia para el registro de un nuevo usuario se ha creado la estrategia para la autenticación de una forma similar, pero en este caso, se hará uso de *jsonwebtoken* para crear un *token* y que así el usuario pueda mantener su sesión iniciada:

Código 4-8. Fichero con la estrategia para inicio de sesión. (server/passport/local-login.js)

```
const jwt = require('jsonwebtoken');
const PassportLocalStrategy = require('passport-local').Strategy;
const config_json = require('../../config/config.json');
const MODE_RUN = process.env.MODE_RUN || "development"
const config = config_json[MODE_RUN]

//Incializamos sequelize
const sequelize = require('../db').sequelize;
```

```
//Cargamos los modelos
const models = require('../models')(sequelize);

module.exports = new PassportLocalStrategy({
  usernameField: 'email',
  passwordField: 'password',
  session: false,
  passReqToCallback: true
}, (req, email, password, done) => {
  const userData = {
    email: email.trim().toLowerCase(),
    password: password.trim()
  };
  return models.User.findOne({where:{
    email: userData.email}})
  .then(function(Found_User, err){
        if(err || !Found_User){
            const error = new Error('Incorrect email or password');
            error.name = 'IncorrectCredentialsError';
            return done(error);
          }
        else{
            //Comprobamos si la contraseña es correcta
            if(Found_User.verifyPassword(password)){
              const payload ={
                sub: Found_User.id
              };
              // Creamos el token
              const  token  =  jwt.sign(payload,process.env.JWT_SECRET  ||
config.jwtSecret); //Algoritmo por defecto HS256
              const data = {
                name: Found_User.name,
                role: Found_User.role,
                id: Found_User.id,
                OrganizationId: Found_User.OrganizationId
              };
              return done(null, token, data);
            }
            else{
```

```
                const error = new Error('Incorrect email or password');
                error.name = 'IncorrectCredentialsError';
                return done(error);
              }
          }
        }, function(err){
              return done(err);
        });
});
```

Al igual que antes, en este código se crea una estrategia importando la clase *Strategy* y con las mismas opciones que para la estrategia de registro, lo único diferente es la función de *callback* ya que en este caso se realiza lo siguiente dentro de dicha función:

1) En primer lugar, se busca al usuario que está intentando autenticarse mediante su email en la base de datos, y en caso de no encontrarlo se devuelve un error notificando que las credenciales son erróneas.
2) Si el usuario existe, se comprueba si la contraseña para ese usuario es correcta (haciendo uso del método de instancia previamente definido para el modelo "user"), y en el caso de no ser correcta se devuelve el mismo error que antes para notificar que las credenciales son erróneas.
3) Por último, si el email y la contraseña son correctos se creará el *token* y se enviará a la siguiente función *middleware* para que se lo envíe al cliente y este pueda almacenarlo para sucesivas peticiones. Además, se puede observar que también se le envía al siguiente *middleware* información relativa al cliente como puede ser el nombre, la organización, el identificador o el email.

Dado que la creación del *token* es de vital importancia para mantener la seguridad de un usuario, ya que es su forma de identificarse, se detalla a continuación cómo se está creando dicho token:
Como se explicó en el apartado de "tecnologías empleadas" un token está formado por una cadena de caracteres codificados en base64[7] y contiene 3 partes: la primera contiene el algoritmo de cifrado y el tipo, la segunda contiene información relativa a la sesión (el identificador del usuario en esta implementación) y la tercera contiene la firma que valida el *token*, y por tanto la parte más importante.
En este caso, la creación del token se realiza haciendo uso de la función *sign* de *jsonwebtoken* (un módulo de *node js* que previamente se ha instalado con *npm*) y se utiliza el algoritmo HS256 (algoritmo por defecto) junto con una clave que la tomará, o bien de la variable de entorno JWT_SECRET, o bien del fichero de configuración por defecto (config/config.json).

[7] Base64: Sistema de codificación que usa 64 como base.

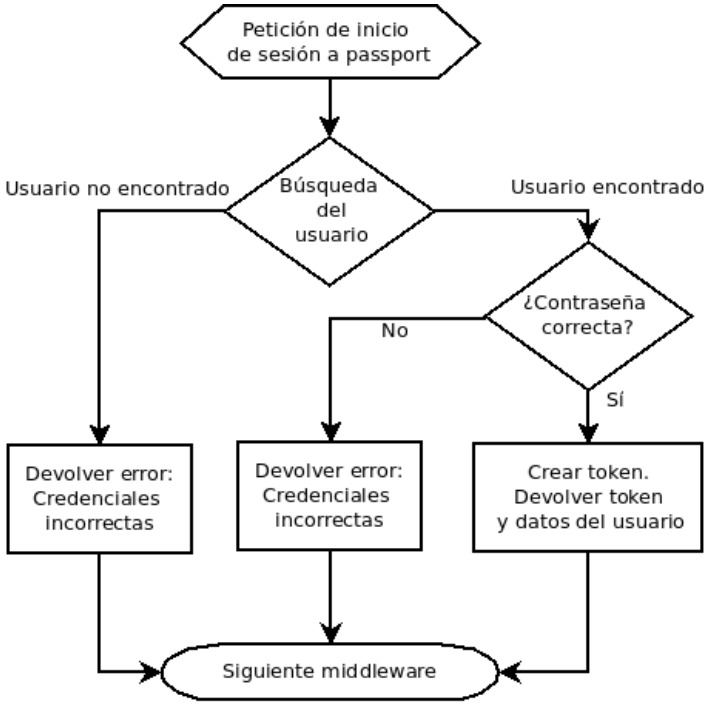


Figura 4-1. Diagrama de flujo de *estrategia* de inicio de sesión

4.3.1.2 *Middleware* de chequeo de sesión

Para que se mantenga la sesión iniciada y sólo los usuarios autenticados puedan acceder a determinadas rutas del servidor, se añadirá una función *middleware* que será "atravesada" por las peticiones que vayan dirigidas a rutas que requieran al usuario estar autenticados, la cual se encargará de verificar el *token*.
En esta función *middleware* se realizará lo siguiente:

Código 4-9. Función middleware para el chequeo de sesión. (server/middleware/auth-check.js)

```
const jwt = require('jsonwebtoken');
const config_json = require('../../config/config.json');

const MODE_RUN = process.env.MODE_RUN || "development"
const config = config_json[MODE_RUN]

//Incializamos sequelize
const sequelize = require('../db').sequelize;

//Cargamos los modelos
const models = require('../models')(sequelize);

module.exports = (req, res, next) => {
  if (!req.headers.authorization) {
    return res.status(404).end(); //Por seguridad enviamos el 404 para que el usuario no sepa que existe tal página
  }
```

```
  // Obtenemos el token de la cabecera del mensaje
  const token = req.headers.authorization.split(' ')[1];


  // Decodificamos el token haciendo uso de la clave secreta
  return  jwt.verify(token,  process.env.JWT_SECRET  || config.jwtSecret,
(err, decoded) => {
    // Si ha habido error, significa que el token no es correcto y se
devuelve un 401 (no autorizado)
    if (err) { return res.status(401).end(); }
    //En caso de no haber error se obtiene el identificador del usuario
del token
    const userId = decoded.sub;
    //Se añade el identificador del usuario a la petición (para usarse en
los siguientes "middleware")
    req.userId = userId;
    // Confirmamos si el usuario existe en la base de datos (puede ser un
token válido pero haberse borrado de la base de datos)
    return models.User.findOne({where: {id: userId}})
    .then(function(User){
      if(!User) //Si el usuario no existe, no le estará permitido su
acceso y pensará que no existe la página
        return res.status(404).end();
      else
        return next();
      }, function(err){
        return res.status(404).end();
      })
    });
}
```

En el fichero anterior, tras la importación de los modelos de la base de datos, el fichero de configuración necesario y *jsonwebtoken* para verificar el *token*, se crea la función *middleware* que realiza la verificación de la sesión de la siguiente forma:

1) Si la cabecera no contiene un token, se devuelve un mensaje HTTP 404 de tal forma que el cliente "ni se entere" de que la ruta a la que está intentando acceder existe, y ofreciendo de este modo una mayor seguridad al ocultar las rutas de acceso con privilegios.
2) Si existe el token, se obtiene de la cabecera y se validará haciendo uso de la función *verify* de la librería *jsonwebtoken.*
3) Si dicha función lanza algún error, significa que el token no es correcto y por tanto se responde con un mensaje HTTP 401 (no autorizado).
4) En el caso de que el token sea correcto, se obtiene el campo identificador que previamente se había incluido en el token y se busca en la base de datos si existe un usuario con dicho identificador. Además, el identificador se añadirá al mensaje de petición para que futuras funciones *middleware*

puedan conocer el identificador del usuario que está autenticado.

5) Si no existe un usuario con dicho identificador, lo que se realizará será devolver un mensaje HTTP 404 (por seguridad, al igual que antes). Este caso puede darse cuando, por ejemplo, se elimine un usuario de la base de datos, pero haya un cliente con una "sesión abierta", y por tanto con un token válido para un usuario que no existe.

6) En el caso de que todo sea correcto, se llamará a la siguiente función *middleware* de *express*, que por lo general será la ruta hacia la que iba dirigido el mensaje.

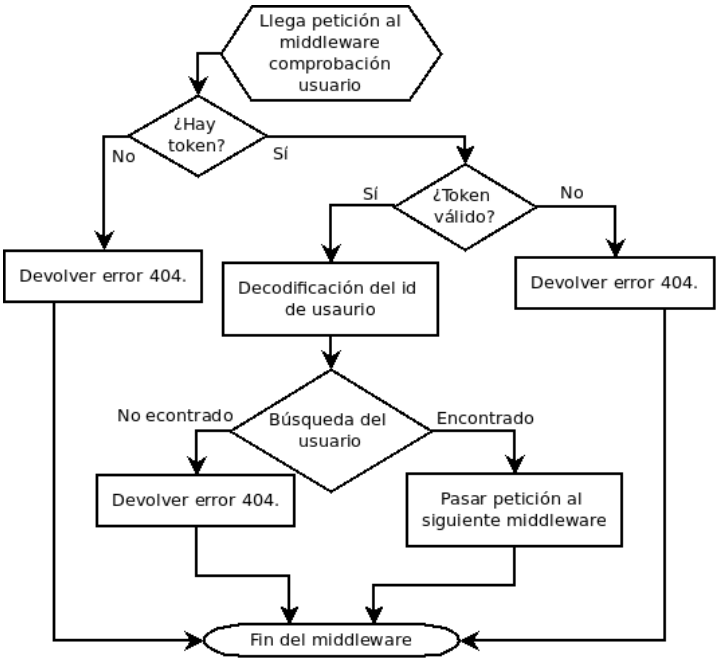


Figura 4-2. Diagrama de flujo del *middleware* que comprueba el token.

4.3.1.3 Configuración para el arranque del servidor *express*

Una vez creadas las estrategias locales para *passport* y definida la función *middleware* que comprueba si un usuario está autenticado, lo que se ha llevado a cabo es la configuración del servidor *express* para que utilice dichas estrategias y dicha función.
Para ello, en el fichero que hará de punto de entrada para el arranque del servidor se realiza lo siguiente:

Código 4-10. Fichero de entrada para el arranque del servidor express. (server/index.js)

```
//En primer lugar ocultamos los console.logs para producción.
process.env.NODE_ENV=="production" ? console.log = function  () {} : null

const express = require('express');
const passport = require('passport');
const bodyParser = require('body-parser');
const app = express();

//Inicializamos sequelize
const connectDB = require('./db').connectDB;

//Lanzamos la conexión a la base de datos mediante la función connectDB
```

```
connectDB();

// Decimos donde están los ficheros estáticos
app.use(express.static('./server/static/'));
app.use(express.static('./client/dist/'));
app.use(bodyParser.urlencoded({ extended: false }));

// Configuramos passport para la autenticación. Esto será un middleware
app.use(passport.initialize());
// Cargamos las estrategias que usará passport para el inicio de sesión y para la creación de usuarios
const localSignupStrategy = require('./passport/local-signup');
const localLoginStrategy = require('./passport/local-login');
passport.use('local-signup', localSignupStrategy);
passport.use('local-login', localLoginStrategy);

// Este middleware comprueba si un usuario está autenticado
const authCheckMiddleware = require('./middleware/auth-check');
app.use('/api', authCheckMiddleware);

// Importamos todas las rutas
const authRoutes = require('./routes/auth');
const apiRoutes = require('./routes/api');
app.use('/auth', authRoutes);
app.use('/api', apiRoutes);

// Finalmente, arrancamos express en el puerto 3000
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000 or http://127.0.0.1:3000');
});

module.exports = app;
```

En este fichero se realizan las siguientes acciones:

1) En primer lugar, se lanza la conexión a la base de datos haciendo uso de la función ConnectDB, descrita anteriormente.
2) Se definen las rutas donde se encuentran los ficheros estáticos (index.html, css, favicon.ico…)
3) Se inicia passport, creándose un *middleware* para cada una de las estrategias descritas anteriormente (*local-login* y *local-password*).

4) Se añade el *middleware*, situado en "server/middleware/auth-check.js", encargado de comprobar si un usuario está o no autenticado, descrito anteriormente.
5) Se cargan los dos ficheros que contendrán las rutas posibles del servidor: el fichero api.js y el fichero auth.js.
6) Se asocia el *middleware*, que comprueba si un usuario está autenticado, a las rutas definidas en el fichero api.js de tal modo que sólo puedan acceder a dichas rutas los usuarios que estén autenticados y envíen en la cabecera un token válido.
7) Finalmente, se lanza el servidor *express* en el puerto 3000

A partir de ahora se podrá iniciar sesión haciendo uso de *passport* y se podrá verificar si un usuario está autenticado "pasando" por la función *middleware* "auth-check".

4.3.1.4 Ruta para el inicio de sesión de un usuario

A continuación, se explicará detalladamente la ruta a la que se tendrá que realizar una petición para que un usuario pueda iniciar sesión. A esta ruta se accederá mediante una petición POST, la cual enviará en los parámetros las credenciales de inicio de sesión (email y contraseña). Además, esta ruta estará definida en el fichero "auth.js", y por tanto no pasará el *middleware* de chequeo de sesión:

Código 4-11. Ruta para el inicio de sesión auth/login (POST). (server/routes/auth.js)

```
{...}
function validateLoginForm(payload) {
  let isFormValid = true;
  let message = '';
  if (!payload || typeof payload.password !== 'string' ||
payload.password.trim().length == 0 || typeof payload.email !== "string"
|| payload.email.trim().length == 0)  {
    isFormValid = false;
    message = 'Please provide your credentials.';
  }
  return {
    success: isFormValid,
    message
  };
}
router.post('/login', (req, res, next) => {
  const validationResult = validateLoginForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  return passport.authenticate('local-login', (err, token, userData) => {
```

```
    if (err) {
      if (err.name === 'IncorrectCredentialsError') {
        return res.status(400).json({
          success: false,
          message: err.message
        });
      }
      return res.status(400).json({
        success: false,
        message: 'Could not process the form.'
      });
    }
    return res.json({
      success: true,
      message: 'You have successfully logged in!',
      token,
      user: {
        name: userData.name,
        role: userData.role,
        id: userData.id,
        OrganizationId: userData.OrganizationId,
        email: req.body.email.toLowerCase()
       }
    });
  })(req, res, next);
});
{...}
```

En la ruta anterior, lo primero que se realiza es llamar a la función validateLoginForm pasándole el cuerpo del mensaje HTTP para que compruebe que el mensaje contiene tanto el email como la contraseña del usuario que está iniciando sesión.

- Si no se pasa con éxito la validación se devuelve un mensaje HTTP 400 (Bad Request) con un campo "success" que será falso y un mensaje de error que contendrá el mensaje que ha devuelto la función de validación.
- Si pasa con éxito la función validación, se hace uso de la función *authenticate* de *passport* y se llama al *middleware* asociado a la estrategia *local-login*, descrita anteriormente.
- Si esta función devuelve error, se lanzará un mensaje HTTP 400.
- Si no ha habido error el mensaje será un 200 OK. Esta respuesta contendrá un mensaje de éxito, el *token* y la información del usuario (nombre, email, organización, rol e identificador).

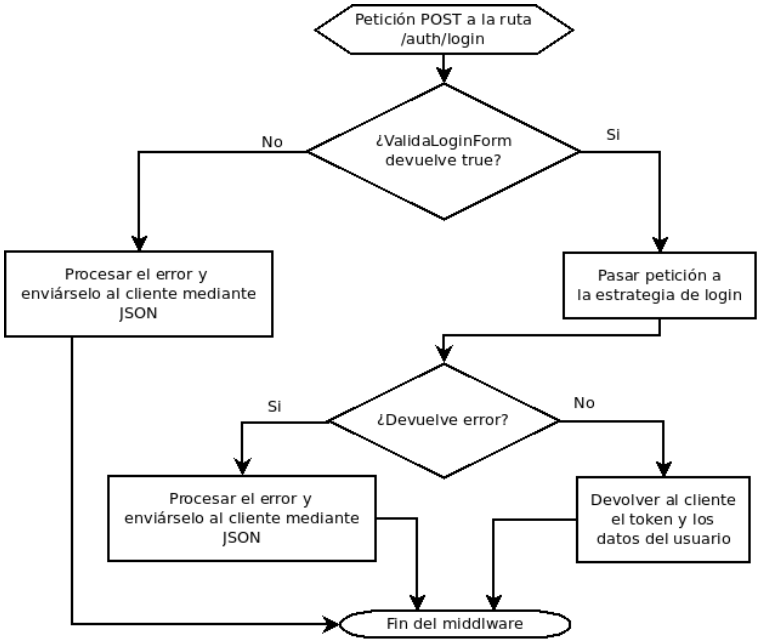


Figura 4-3. Diagrama de flujo del *middleware* de la ruta POST /auth/login

4.3.2 Código de Front-end

Una vez desarrollado todo el código de back-end encargado de comprobar si un usuario se ha autenticado correctamente, es el momento de llevar a cabo el desarrollo del código del lado del cliente para mostrar al usuario una interfaz en la cual pueda introducir sus datos e iniciar sesión.
Para llevar a cabo el desarrollo del código de Front-end se ha decidido optar, como ya se dijo, por *React JS.*

4.3.2.1 Código estático

Cuando el cliente realiza una primera petición al servidor, éste le responde con una página HTML completamente formada, con su cabecera, su cuerpo y los enlaces para las distintas librerías que se utilizarán (Bootstrap, Toastr…):

Código 4-12. Página HTML recibida por el cliente. (server/static/index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Licensing management of RedBorder</title>
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon"/>
    <link rel="stylesheet" href="css/style.css"/>
    <link                                                rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.min.
css">
    <link                                                rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap.min.
css">
    <link                                                rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap-
theme.min.css">
```

```
    <link   rel="stylesheet"   href="https://npmcdn.com/react-bootstrap-
table/dist/react-bootstrap-table-all.min.css">
  </head>
  <body>
    <div id="root"></div>
    <script src="/js/app.js"></script>
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"><
/script>
    <!-- Latest compiled and minified JavaScript -->
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.min.j
s"></script>
  </body>
</html>
```

Si se observa el código anterior, podemos ver que la página HTML contiene:

- El título de la página
- El icono de la página
- El enlace a un fichero de estilos personalizado
- Los enlaces a los distintos JavaScripts y CSS necesarios para *Bootstrap* y *Toastr*
- Una etiqueta HTML de tipo "div" con el identificador "root"
- Un script donde se importa el fichero *app.js*, que será el fichero generado mediante *WebPack* en el lado del cliente.

Una vez recibe el cliente esta página, será allí donde se "genere" la interfaz de usuario mediante *React JS*.

4.3.2.2 Fichero de entrada para el cliente

En el cliente existirá un fichero de entrada para WebPack (client/src/index.jsx), el cual se encargará de buscar, en la página web recibida del servidor, la etiqueta HTML cuyo identificador sea "root" para "introducir" dentro de ella al componente principal de la aplicación, el componente *App:*

Código 4-13. Fichero de entrada para el cliente. (client/src/index.jsx)

```
import { render } from 'react-dom';
import App from './app.jsx';
import injectTapEventPlugin from 'react-tap-event-plugin';
injectTapEventPlugin();
```

```
render(
  <App />,
  document.getElementById('root')
);
```

Como se puede observar, en el fichero anterior se importa la función *render* de *react-dom*, una función encargada de interpretar el componente principal App, previamente importado, e introducirlo en el elemento de la página HTML recibida cuyo identificador es "root".
También se importa la función *injectTapEventPlugin* que se llamará una vez en todo el código y se encarga de eliminar los retrasos en las pulsaciones que se introducen en algunos navegadores, principalmente en navegadores de teléfonos móviles.

4.3.2.3 Configuración de *React-Router*

Como *React* es una tecnología utilizada, principalmente, para llevar a cabo el desarrollo de interfaces de usuarios para páginas webs basadas en el modelo SPA (Single page application), se ha optado por utilizar una librería con la que el usuario tenga la sensación de estar navegando por las distintas opciones de la web sin tener que realizar en cada interacción una nueva petición solicitando una página web completa al servidor, como es el caso del modelo de desarrollo MPA (Multi page application). Esta tecnología es conocida como ***React Router***.
React se basa principalmente en la creación de componentes y de interpretarlos para mostrarlos en el navegador web. Pues bien, *React Router* se aprovecha de esto de forma que cuando el usuario accede a una URL concreta, se muestren unos componentes u otros en función de una serie de rutas definidas por el desarrollador.
Para configurarlo en este proyecto, lo primero que se necesita es introducir en el componente principal el componte *Router*, al cual se le pasará el fichero donde están definidas las diferentes rutas y cómo se navegará por la web. En este caso se ha optado por utilizar *hashHistory*, de forma que en la url haya una almohadilla (#) entre el dominio de la web y la ruta (www.dominio.com/#/ruta):

Código 4-14. Componente principal de React. (client/app.js)

```
import React from 'react';
import { hashHistory, Router } from 'react-router';
import routes from './routes.js';

class App extends React.Component{
    render(){
        return(
            <Router history={hashHistory} routes={routes} />
        );
    }
}
export default App;
```

Como se puede observar, en el fichero anterior se crea el componente principal, *App*, que se encarga de introducir el componente *Router* pasándole *hashHistory* y las rutas que se importarán del fichero routes.js.

4.3.2.4 Fichero de rutas para *React-Router*

Como se ha dicho anteriormente, existe un fichero de rutas para *React Router* encargado de definir las rutas disponibles y qué componentes mostrar en cada una de ellas.
A continuación, se mostrará un fragmento de dicho fichero con la ruta necesaria para el inicio de sesión:

Código 4-15. Fragmento del fichero de rutas. (client/routes.js)

```
import BasePage from './containers/BasePage.jsx';
import HomePage from './containers/HomePage.jsx';
import DashboardPage from './containers/DashboardPage.jsx';
import LoginPage from './containers/LoginPage.jsx';
{...}
import Auth from ‘./modules/Auth’;
const routes = {
  // Componente Base
  component: BasePage,
  childRoutes: [
    {
      path: '/',
      getComponent: (location, callback) => {
        if (Auth.isUserAuthenticated()) {
          callback(null, DashboardPage);
        } else {
          callback(null, HomePage);
        }
      }
    },
    {
      path: '/login',
      component: LoginPage
    },
    {
      path: '/logout',
      onEnter: (nextState, replace) => {
        Auth.deauthenticateUser();

        // Cambiamos la url por la base /
        replace('/');
      }
    }
```

```
  ]
};
export default routes;
```

En este fichero, en primer lugar, se ha llevado a cabo la importación de los componentes *BasePage*, *HomePage*, *DashboardPage* y *LoginPage* que se describirán más adelante.
Tras esto, se ha importado la clase auxiliar *Auth*, una clase que ofrecerá una serie de métodos útiles para el cliente, como son: el método para saber si un usuario está autenticado, para cerrar sesión, para saber si es administrador…
Esta clase está definida en el fichero /client/modules/auth.js.

Una vez se han importado todos los componentes necesarios, se crea el objeto *routes*, este objeto posee, en primer lugar, un componente que se usará como base (*BasePage*) y una serie de rutas que harán de "hijos" de ese componente, es decir, se le pasará a *BasePage* un componente u otro en función de la URL a la que estemos accediendo.
En este fragmento del fichero *routes.js* solo se muestran tres rutas:

- La ruta raíz (/), en cuyo caso, haciendo uso del método *isUserAuthenticated* de la clase auxiliar *auth*, utiliza bien el componente de inicio (*HomePage*), o bien el componente que corresponde a la página de bienvenida para un usuario que está autenticado (*DashboardPage*).
- La ruta correspondiente al inicio de sesión (/login), en cuyo caso se le pasará al componente *BasePage* el componente *LoginPage* para que sea mostrado en el navegador y el usuario pueda iniciar sesión.
- La ruta correspondiente a cerrar sesión (/logout). Esta ruta no tiene asociado ningún componente, sino que al ir a dicha ruta se llamará al método *deauthenticateUser* que cerrará la sesión de un usuario (Este método sólo elimina el *token*).

A continuación, se muestran los métodos *deauthenticateUser* y *isUserAuthenticated* de la clase *Auth*. En este último, lo único que se comprueba es si existe el ítem "token" almacenado en el navegador, en cuyo caso se supondrá que el usuario está autenticado.
Esto podría parecer que es muy inseguro, sin embargo, como medida de seguridad, en el caso de que haya un *token* almacenado, pero no sea válido, al lanzar alguna petición al servidor, como allí siempre se valida el token (*middleware* de chequeo de *token*), no se permitirá realizar nada.

Código 4-16. Métodos *isUserAuthenticated* y *deauthenticateUser* de Auth. (client/modules/Auth.js)

```
{...}
static isUserAuthenticated() {
    return localStorage.getItem('token') !== null;
 }
static deauthenticateUser() {
    localStorage.removeItem('token');
  }
{...}
```

4.3.2.5 Componente *BasePage*

A lo largo del desarrollo de este proyecto, se han creado 16 componentes propios que se irán describiendo a medida que se vayan utilizando.
Sin embargo, antes de pasar al detalle del componente *BasePage*, es importante aclarar cómo se han organizado los componentes a lo largo de la aplicación:

En la ruta client/src existen los directorios *components* y *containers*, en los cuales se almacenarán los ***componentes*** (conocidos en inglés *Presentational Components*) y los ***contenedores*** de dichos componentes, respectivamente.
La principal diferencia entre ellos radica en que los componentes interpretan directamente las etiquetas JSX que se encuentran definidos en ellos (Es decir, son el cómo se ven las cosas), mientras que los contenedores se encargan de tratar los datos, definir funciones y pasárselas a los componentes para que ellos los muestren y/o utilicen (Es decir, son el cómo se hacen las cosas).
En el proyecto, los componentes serán llamado con los nombres Base, Login… y a los contenedores se le añadirá el sufijo la "Page" ya que serán estos los que el navegador interprete para "crear la página".
A continuación, se mostrará la definición del componente *Base:*

Código 4-17. Componente *Base* para la creación de *BasePage.* (client/src/components/Base.jsx)

```
import React from 'react';
import { Link, IndexLink } from 'react-router';
import Auth from '../modules/Auth';
import PropTypes  from 'prop-types';


/* Componente Base encargado de crear la barra de navegación superior.
Hará uso de React router para la navegación entre las diferentes opciones del menú
Recibirá los siguientes parámetros:
  1) Children: Componente que se mostrará haciendo uso de React-Router bajo la barra de navegación
*/
const Base = ({children}) => (
  <div>
    <nav className="navbar navbar-inverse navbar-fixed-top">
      <div className="container">
        <div className="navbar-header">
          <button type="button" className="navbar-toggle collapsed" data-toggle="collapse"   data-target="#base-collapse"   aria-expanded="false" aria-controls="navbar">
            <span className="sr-only">Toggle navigation</span>
            <span className="icon-bar"></span>
            <span className="icon-bar"></span>
            <span className="icon-bar"></span>
          </button>
          <IndexLink   className="navbar-brand"   to="/"   style={{color: 'blue'}}><span className="glyphicon glyphicon-home"></span></IndexLink>
        </div>
        <div className="collapse navbar-collapse" id="base-collapse">
       {Auth.isUserAuthenticated() ? (
```

```
          Auth.isAdmin() ? (
          <div>
            <div>
              <ul className="nav navbar-nav" >
                <li>
                  <Link to="/listOrgs"> Organizations </Link>
                </li>
                <li>
                  <Link to="/listUsers/all/all"> Users </Link>
                </li>
              </ul>
            </div>
            <div>
              <ul className="nav navbar-nav navbar-right" >
                <li>
                  <Link    to="/user/edit"><span    className="glyphicon
glyphicon-user"></span> My Profile</Link>
                </li>
                <li>
                  <Link      to="/logout"><span      className="glyphicon
glyphicon-log-out"></span> Log out</Link>
                </li>
              </ul>
            </div>
          </div>
          ) : (
            <div>
              {Auth.hasOrganization() ? (
              <ul className="nav navbar-nav" >
                <li>
                  <Link              to={"/listLicenses/"              +
localStorage.getItem('userProfileOrg')  +  "/"  +  encodeURIComponent("my
organization")}> My licenses </Link>
                </li>
              </ul>
              ) : null }
              <ul className="nav navbar-nav navbar-right">
                <li>
                  <Link    to="/user/edit"><span    className="glyphicon
glyphicon-user"></span> My Profile</Link>
```

```
                </li>
                <li>
                  <Link      to="/logout"><span      className="glyphicon
glyphicon-log-out"></span> Log out</Link>
                </li>
              </ul>
            </div>
          )
        ) : (
            <ul className="nav navbar-nav navbar-right">
              <li>
                <Link  to="/login"><span  className="glyphicon  glyphicon-
log-in"></span> Log in user</Link>
              </li>
            </ul>
        )}
        </div>
      </div>
    </nav>
    <div className="container">
    {children}
    </div>
  </div>
);

//Haciendo  uso  de  propTypes  se  comprueba  que  existe  el  componente
'children', ya que es obligatorio
Base.propTypes = {
  children: PropTypes.object.isRequired,
};
export default Base;
```

En el fichero anterior, se puede observar que se define cómo el componente *BasePage* se mostrará en el navegador y para ello se crea una función (haciendo uso de la forma *arrows*[8] de ES6):

- Acepta como parámetro un objeto llamado *children*, el cual será obligatorio. Esto se valida por medio de propTypes.
- Devuelve una etiqueta html de tipo "div" haciendo uso de JSX.

[8] Función arrow: Traducido del inglés sería función "flecha" y permite definir funciones de una forma más compacta que de la forma tradicional en EcmaScript 6

La etiqueta de tipo “div” devuelta contendrá todos los componentes que *BasePage* mostrará en el navegador y son los siguientes:

- Una barra de navegación utilizando los estilos de *bootstrap* por medio de la identificación por el atributo className y una etiqueta *div* que hará de contenedor para el componente *children* que recibe por parámetro.
- Dentro de la barra de navegación, haciendo uso de la clase auxiliar *Auth*, se comprueba si el cliente está autenticado, o no, si es administrador, o no, y si pertenece, o no, a una organización. En función de estas comprobaciones se mostrarán unos enlaces en la barra de navegación u otros. Estos enlaces serán componentes de tipo *Link* importados de *react-router*, y al hacer click en alguno de ellos, por medio de react-router se cambiará la URL, enviándose a *BasePage* el nuevo componente que tendrá que mostrar en función de la URL.

Código 4-18. Contenedor *BasePage* para el componente *Base*. (client/src/containers/BasePage.jsx)

```
import React, { Component } from 'react';
import Base from '../components/Base.jsx'
/*
Clase BasePage encargada de crear un componente Base con un compenten
'children' recibido mediante reac-router
*/
class BasePage extends Component {
  constructor() {
    super();
  }
  render(){
    return <Base children={this.props.children}/>
  }
}
export default BasePage;
```

El contenedor para BasePage no hace más que crear el componente *BasePage* tomando el componente *Base* y pasándole la propiedad *children* que se le pasa a *BasePage.* En este caso no hubiera sido primordial la creación de un contenedor, ya que en él no se define ninguna función ni similar, sino que simplemente importa el componente *Base* y se usa. Sin embargo, para llevar un desarrollo ordenado, para todos los componentes del proyecto habrá un contenedor asociado.

4.3.2.6 Componente *HomePage*

A continuación, se describirá cómo se ha creado el componente *HomePage* que se mostrará cuando se accede a la ruta raíz sin estar autenticado.

Código 4-19. Componente *Home* para el componente *HomePage.* (client/src/components/Home.jsx)

```
import React from 'react';
import { Panel } from 'react-bootstrap';
/*
Componente encargado de mostrar la página de inicio
```

```
*/
const Home = () => (
    <div>
        <Panel header="Licenses management" bsStyle="info">
            This is the home page
        </Panel>
    </div>
);

export default Home;
```

Al igual que para el resto de componentes, aquí se define la forma de cómo se mostrará el componente HomePage, y para ello se crea una función que no acepta parámetros y devuelve un componente de tipo *Panel* importado de *react-bootstrap* entre etiquetas de tipo *div*.

Código 4-20. Contenedor *HomePage* para el componente *Home*. (client/src/containers/HomePage.jsx)

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import Home from '../components/Home.jsx';

class HomePage extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (<Home/>);
  }
}

export default HomePage;
```

Como sucede para el componente *BasePage*, al no ser muy complejo el componente *HomePage* no hubiera necesitado un contenedor, pero se ha creado para desarrollar de un modo ordenado.

4.3.2.7 Componente *DashboardPage*

A continuación, se describirá cómo se ha creado el componente DashboardPage que se motrará cuando un componente que esté autenticado acceda a la URL raíz.

Código 4-21. Componente *Dashboard* para *DashboardPage*. (client/src/components/Dashboard.jsx)

```
import React from 'react';
import { Panel } from 'react-bootstrap';
/*
```

```
Componente encargado de mostrar la pagina principal cuando un usuario se
ha autenticado
No recibe parámetros
*/
const Dashboard = () => (
    <div>
        <Panel header="Licenses management" bsStyle="info">
            This is the dashboard page, if you can see this, you are
autenticated
        </Panel>
    </div>
);
export default Dashboard;
```

Al igual que para el resto de componentes, aquí se define la forma de cómo se mostrará el componente DashboardPage, y para ello se crea una función que no acepta parámetros y devuelve un componente de tipo *Panel* importado de *react-bootstrap* entre etiquetas HTML de tipo *div.su*

Código 4-22. Contenedor *DashboardPage* para *Dashboard.* (client/src/containers/DashboardPage.jsx)

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import Dashboard from '../components/Dashboard.jsx';



class DashboardPage extends Component {


  /**
   * Clase constructora.
   */
  constructor(props) {
    super(props);
    //Obtenemos el nombre y el email almacenados de forma local
    const name = localStorage.getItem('userProfileName');
    const email = localStorage.getItem('userProfileEmail');
  }
  /**
   * Instanciamos un componente de tipo Dashboard.
   */
  render() {
    return (<Dashboard/>);
```

```
  }

}

export default DashboardPage;
```

Como sucede para el componente *BasePage* y *HomePage*, al no ser muy complejo, el componente *DashboardPage* no hubiera necesitado un contenedor, pero se ha creado para desarrollar de un modo ordenado.

4.3.2.8 Componente *LoginPage*

En este apartado se describirá el componente encargado de mostrar el formulario para el inicio de sesión en el lado del cliente.

Código 4-23. Fragmento componente *LoginForm.* (client/src/components/LoginForm.jsx)

```
import React from 'react';
{...}
const LoginForm = ({
  onSubmit,
  onChange,
  errors,
  user
}) => (
  <div className="container">
    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> Log in form
</h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="email" validationState={errors.email==""  ?
null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
        <Col sm={10}>
          <FormControl name ="email" type="email" placeholder="Email"
required="true" onChange={onChange} value={user.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>
```

```
      <FormGroup                                        controlId="password"
validationState={errors.password=="" ? null : errors.password}>
        <Col componentClass={ControlLabel} sm={2}>
          Password
        </Col>
        <Col sm={10}>
          <FormControl          name="password"          type="password"
placeholder="Password"           required="true"     onChange={onChange}
value={user.password}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Sign in
          </Button>
          <Link to={'/forgot'}> Forgot your password?</Link>
        </Col>
      </FormGroup>
    </Form>
  </div>
);
{...}
export default LoginForm;
```

Como se puede observar, en el código anterior se define el componente encargado de crear el formulario para el inicio de sesión haciendo uso de la librería *react-bootstrap* que acepta los siguientes parámetros obligatorios:

- onSubmit: Función que se ejecutará al pulsar el botón "Sign in"
- onChange: Función que se ejecutará cada vez que se produzca un cambio en algún campo de entrada del formulario
- errors: Objeto en el que se almacenará si algún campo tiene error de tal forma que se pueda mostrar visualmente haciendo uso de las funciones de los componentes *FormControl* y *Feedback*
- user: Objeto en el que se almacenarán los datos introducidos en el formulario de inicio de sesión

Código 4-24. Fragmento componente LoginPage. (client/src/containers/LoginPage.jsx)

```
{...}
 constructor(props, context) {
    super(props, context);
```

```
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    this.state = {
      errors: {
        email: '',
        password: ''
      },
      user: {
        email: '',
        password: ''
      }
    };
    /Necesario para poder llamar a estas funciones desde la propia clase
    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }
processForm(event) {
    // Previene que se envíen valores por defecto
    event.preventDefault();

    // Crea la cadena con los valores a enviar al servidor
    const email = encodeURIComponent(this.state.user.email);
    const password = encodeURIComponent(this.state.user.password);
    const formData = `email=${email}&password=${password}`;

    // Creación de la petición AJAX
    const xhr = new XMLHttpRequest();
    xhr.open('post', '/auth/login');
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Todo correcto
        this.setState({
```

```
          errors: {}
        });
        {xhr.response.message && toastr.success(xhr.response.message)}
        // Almacenamos en local los datos de un usuario
        localStorage.setItem('userProfileId', xhr.response.user.id);
        localStorage.setItem('userProfileOrg',
xhr.response.user.OrganizationId);
        localStorage.setItem('userProfileName', xhr.response.user.name);
        localStorage.setItem('userProfileEmail',
xhr.response.user.email);
        localStorage.setItem('userProfileRole', xhr.response.user.role);
        // Almacenamos el token
        Auth.authenticateUser(xhr.response.token);
        // Cambiamos la url a la página de inicio
        this.context.router.replace('/');
      } else {
        // Si hay error lo notificamos
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);
  }
changeUser(event) {
    const field = event.target.name;
    const user = this.state.user;
    user[field] = event.target.value;
    this.setState({
      user
    });
    //Esto es para validar el formulario visualmente
    if(event.target.name=="password"   &&   user[field].length   <   8   ||
user[field].length > 15)
      this.state.errors.password="error";
    else
      this.state.errors.password="success"
```

```
  }
{...}
```

En este fragmento del componente *LoginPage* se muestran sus dos funciones más importantes: *processForm* y *changeUser*, además de la función constructora en la que se inicia el estado de la clase (objetos *errors* y *user*). Ambas funciones se le pasarán al componente *LoginForm* junto con los dos objetos.
En la función *changeUser* cada vez que se llama se actualiza el campo del objeto *user* que ha cambiado (en función del nombre del evento) con el valor del evento. Una vez realizado esto se comprueba si la contraseña está rellena y con una longitud entre 8 y 16 caracteres, actualizándose el objeto *errors* para notificarlo de forma visual.
Por otro lado, en la función *processForm* se crea una cadena con los parámetros a enviarle al servidor, en este caso el email y la contraseña del usuario. Mediante AJAX, se llama al método POST de la ruta /auth/login del servidor. Tras configurar la cabecera y la respuesta se añade el código que se ejecutará tras recibir la respuesta mediante AJAX, en cuyo caso lo que se realizará será:

- Si todo ha ido bien (se recibe un 200 OK), se notifica al usuario haciendo uso de un *toast*, se almacenan los datos del usuario que ha iniciado sesión en local (para disponer de ellos más rápidamente), se almacena el token haciendo uso de la función *authenticateUser* del fichero *Atuh.js* y se redirecciona a la página principal.
- Si ha habido error, se notifica al cliente mediante un *toast* de dicho error y no se hace nada más.

4.4 Gestión de perfil

En el siguiente apartado, se explicará qué ha sido necesario realizar para poder implementar una solución para el requisito que permite la gestión del perfil a un usuario previamente autenticado.

4.4.1 Código de *back-end*

En el lado del servidor hay que verificar, en primer lugar, que el usuario que desea modificar el perfil está autenticado de forma correcta y en segundo lugar que los datos enviados, para modificar el perfil, son los necesarios y los correctos, antes de proceder a la modificación de los datos del usuario.
Para la validación de la autenticación, como se dijo en el apartado anterior, existe una función *middleware* que comprueba si un usuario está autenticado de forma correcta. Esta función solamente se aplica a rutas que estén dentro del fichero server/routes/api.js, y por tanto se ha definido la ruta encargada del cambio del perfil dentro de dicho fichero de tal modo que sólo pueden acceder a ellas usuarios que estén ya autenticados en el sistema.

A continuación, se muestra el fragmento de código que define la ruta:

Código 4-25. Ruta para cambiar el perfil de un usuario api/changeProfile (POST). (server/routes/api.js)

```
{...}
router.post('/changeProfile', (req, res) => {
  const validationResult = validateChangeProfileForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
```

```
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user.verifyPassword(req.body.password)){
        return res.status(401).json({
            success: false,
            message: "Current password is not correct.",
          });
      }
      if(req.body.new_password)
        user.password = req.body.new_password;
        user.email = req.body.email.trim().toLowerCase();
        user.name = req.body.name;
        user.save().then(function(){
          return res.status(200).json({
            success: true,
            message: "You have changed your profile correctly!",
            user
          });
        }, function(){
          return res.status(400).json({
            success: false,
            message: "Error changing user profile. This email alredy
exist"
          });
        })
      }, function(err){
        return res.status(400).json({
        success: false,
        message: "Error. User not found.",
        });
      })
});
{...}
```

Como se puede observar, en esta ruta lo primero que se hace es validar que se han recibido todos los parámetros necesarios y para ello hace uso de la función *validateChangeProfileForm* definida en ese mismo fichero. En el caso de que dicha función devuelva éxito:

- Se buscará al usuario mediante el identificador del mensaje de petición que ha sido decodificado y añadido en la función *middleware* que comprueba el *token* (*auth-check*), y si dicho usuario no existe se enviará un error en la respuesta (HTTP 400).
- En el caso de existir dicho usuario, se comprueba si la contraseña actual que se le ha solicitado al cliente es correcta ya que en el caso de no serla se devolverá un error HTTP 401.
- Si todo ha ido correctamente, se modifican los datos del usuario (la contraseña solo se modificará si el campo que contiene la nueva contraseña está relleno), se enviará la respuesta con un mensaje de éxito y el usuario modificado (*user*) con el código 200 OK.
- Si ha habido un error interno a la hora de modificar el usuario se notificará con un mensaje y el código 400.

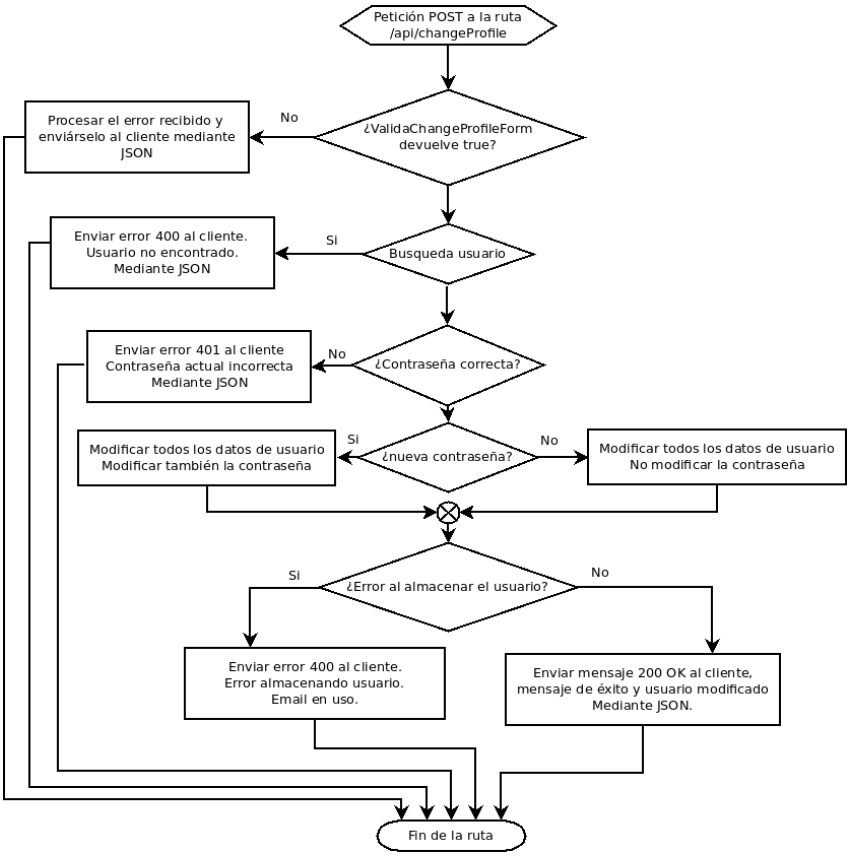


Figura 4-4. Diagrama de flujo del *middleware* de la ruta POST /api/changeProfile

4.4.2 Código de *Front-end*

Una vez se ha creado la ruta para la gestión del perfil en el lado del servidor, es el momento de crear la interfaz del lado del cliente, encargada de hacer uso de dicha ruta.

4.4.2.1 React router para gestión del perfil de un usuario

Para ello, se ha creado una nueva entrada en el fichero de rutas de *react router* (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/user/edit se muestre el formulario para introducir los nuevos datos del usuario que se está modificando.

Código 4-26. Fragmento con la ruta para mostrar el cambio de perfil en el cliente. (client/src/routes.js)

```
{...}
{
  path: '/user/edit',
  component: ProfilePage
},
{...}
```

Como se puede observar en el fragmento anterior, existe un componente llamado *ProfilePage* que será el que se muestre cuando accedamos a la ruta /user/edit. A esta ruta se podrá acceder de un modo sencillo mediante un *Link* (llamado "My profile") existente en el componente *BasePage* y está situado en la barra de navegación. Solo se mostrá si el usuario está autenticado.

4.4.2.2 Componente *ProfileForm*

Como se ha realizado con los componentes anteriores, existirá un componente de presentación (*presentational component*) en el cual se definirá qué se va a mostrar cuando se muestre el componente *ProfilePage*, ya que este será un contenedor que hará uso del componente *ProfileForm.*
El componente *ProfileForm*, como su propio nombre indica, no se encarga más que de mostrar el formulario donde introducir los datos para un usuario.
El código puede verse en el anexo A. Código A-16.

Este formulario contendrá:

- Un campo para el nombre del usuario.
- Un campo para el email del usuario.
- Un campo para la nueva contraseña.
- Un campo para la confirmación de la nueva contraseña.
- Un campo para la contraseña actual.
- Un botón para enviar el formulario.

Por otro lado, como sucederá con cualquier componente que crea un formulario definido a lo largo del proyecto, necesita:

- Una función de *callback* (*onSubmit*) que se llame cuando se pulse el botón para enviar el formulario.
- Una función de *callback* (*onChange*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario.
- Un objeto (*errors*) que contendrá si algún campo del formulario tiene error para que se muestre de forma visual en el navegador.
- Un objeto (*user*) que contendrá el valor recogido en los diferentes campos del formulario.

Todos estos parámetros serán obligatorios para este componente.

4.4.2.3 Componente *ProfilePage*

Una vez que se ha definido el componente de presentación (*presentational component*), se ha pasado a definir el contenedor encargado de usarlo. Este contenedor será el componente *ProfilePage* y en él se realiza lo siguiente:

En primer lugar, dentro del constructor, se inicializarán los estados de los objetos *errors* y *user*. Para el primero de ellos se inicializarán todos sus campos vacíos, pero, para el objeto *user* se inicializará con el nombre y el email rellenos con el valor de los ítems locales que se crearon al iniciar sesión. De esta forma, cuando se muestre el formulario, los campos correspondientes al nombre y al email ya estarán rellenos con los datos actuales del usuario:

Código 4-27. Inicialización de estados de ProfilePage. (client/src/containers/ProfilePage.jsx)

```
{...}
   const name = localStorage.getItem('userProfileName');
    const email = localStorage.getItem('userProfileEmail')

    // set the initial component state
    this.state = {
      errors: {
        name: '',
        new_password: '',
        confir_new_password: '',
        email: '',
        password: ''
      },
      user: {
        name: name,
        new_password: '',
        confir_new_password: '',
        email: email,
        password: ''
      }
    };
{...}
```

Una vez se inicializan los datos, se han tenido que definir las funciones correspondientes a *onChange* y *onSubmit*.

La función que se le pasará como *callback* para *onChange* es la siguiente:

Código 4-28. Función *changeUser* de de ProfilePage. (client/src/containers/ProfilePage.jsx)

```
{...}
changeUser(event) {
    const field = event.target.name;
    const user = this.state.user;
    user[field] = event.target.value;
```

```
    this.setState({
      user
    });

    //Esto es para validar el formulario visualmente, solo para el
usuario
    if(this.state.user.new_password                                    ===
this.state.user.confir_new_password
      && this.state.user.new_password.length > 8
      && this.state.user.new_password.length < 15
      && this.state.user.confir_new_password.length > 8
      && this.state.user.confir_new_password.length < 15
      ||            (this.state.user.new_password.length==0            &&
this.state.user.confir_new_password.length == 0)){
      this.state.errors.new_password="success";
      this.state.errors.confir_new_password="success";
    }
    else{
      this.state.errors.new_password="error";
      this.state.errors.confir_new_password="error";
    }
    if(this.state.user.password.length          <          8          ||
this.state.user.password.length > 15 || this.state.user.password.length==
0 )
      this.state.errors.password="error";
    else
      this.state.errors.password="success"

    if(this.state.user.name.length!=0)
      this.state.errors.name="success";
    else
      this.state.errors.name="error"
  }
{...}
```

Como se puede observar, en esta función lo que se realiza es en primer lugar almacenar, en función del nombre del evento que se le ha pasado, el valor del campo que se ha modificado.
Tras esto, se comprueba que todos los campos sean correctos, y si no es así cambiar el valor del objeto *errors* para que el componente *ProfileForm* pueda mostrarlo en el navegador.
Por otro lado, la función que se le pasará al método *onSubmit* encargada de procesar el formulario y enviarlo al servidor será:

Código 4-29. Función *processForm* de de ProfilePage. (client/src/containers/ProfilePage.jsx)

```
{...}
processForm(event) {
    // prevent default action. in this case, action is the form
submission event
    event.preventDefault();
    // create a string for an HTTP body message
    const name  = encodeURIComponent(this.state.user.name);
    const email = encodeURIComponent(this.state.user.email);
    const new_password =
 encodeURIComponent(this.state.user.new_password);
    const confir_new_password =
 encodeURIComponent(this.state.user.confir_new_password);
    const password = encodeURIComponent(this.state.user.password);
    const formData =
`email=${email}&password=${password}&name=${name}&new_password=${new_pass
word}&confir_new_password=${confir_new_password}`;


    // Creamos una petición AJAX
    const xhr = new XMLHttpRequest();
    xhr.open('post', '/api/changeProfile');
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Si ha ido bien borramos los errores
        this.setState({
          errors: {}
        });


        // Cambiamos el valor por el de los nuevos items recibidos.
        localStorage.setItem('userProfileName', xhr.response.user.name);
        localStorage.setItem('userProfileEmail',
xhr.response.user.email);
        //Notificamos el éxito con un toast
        {xhr.response.message && toastr.success(xhr.response.message)}
```

```
        // Cambiamos la url a la raiz /
        this.context.router.replace('/');

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      }
       else {

        // En caso de fallo lo notificamos con un toast y modificamos el
objeto errors
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);
  }
{...}
```

En la función anterior, lo primero que se realiza es obtener los campos del objeto *user* y crear una cadena para enviarla en el método POST.
Una vez creada la cadena, se crea la petición POST a la ruta /api/changeProfile, se modifica la cabecera incorporando el *token*, añadimos qué se ejecutará cuando se reciba la respuesta y es lo siguiente:

- Si se recibe un 200 OK, se eliminarán todos los errores, se almacenarán los nuevos valores de los campos nombre y email, se notificará con un *toast* y se cambiará a la URL raíz.
- Si se recibe un 404, significa que no se tienen permisos y se cierra la sesión del usuario.
- En otro caso, se muestra un *toast* con el error recibido.

4.5 Renovación de contraseña

En este apartado, se explicará cómo se ha implementado una solución para el recordatorio de contraseña de un usuario que la haya olvidado, de tal modo que pueda generar una nueva.

Antes de proceder a explicar el código hay que recordar que al definir el modelo de un usuario se añadieron dos campos extras: “resetPasswordToken” y “resetPasswordExpires”. Pues ahora es el momento de explicar para qué se usarán:

Cuando un usuario solicite recordar su contraseña introducirá su email, y se almacenará en los campos *resetPasswordToken* y *resetPasswordExpires* un *token* generado de forma aleatoria y una fecha de expiración (1 hora más desde que se solicitó la renovación), respectivamente.
Tras esto, se le enviará un email a la cuenta introducida con una url (contendrá el token aleatorio) a la cual podrá acceder antes de 1 hora y solicitándole una nueva contraseña.
Una vez cambiada la contraseña, los campos *resetPasswordToken* y *resetPasswordExpires* se pondrán a *null* de nuevo, y se le enviará una confirmación al email del usuario.

4.5.1 Código de *back-end*

Para llevar a cabo todo esto ha sido necesario crear dos rutas en el servidor, una para la generación del email con el *token* aleatorio y el tiempo de expiración, y la otra para el cambio de la contraseña. Ambas rutas se han creado en el fichero *Auth.js,* ya que como es obvio, no pueden estar dentro del fichero *Api.js* debido a que el usuario que solicita el cambio de contraseña no estará autenticado y por tanto no pasará el *middleware* asociado a las rutas del fichero *Api.js*

4.5.1.1 Ruta para solicitar renovación de contraseña

Código 4-30. Ruta para solicitar renovación de contraseña auth/forgot (POST). (server/routes/auth.js)

```
router.post('/forgot', function(req, res, next) {
  const validationResult = validateForgotForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  async.waterfall([
    function(done) {
      crypto.randomBytes(20, function(err, buf) {
        var token = buf.toString('hex');
        done(err, token);
      });
    },
    function(token, done) {
      models.User.findOne({
                    where: {
                        email: req.body.email
                    }
                })
        .then(function(user, err){
          if (!user) {
```

```
            return res.status(400).json({
            success: false,
            message: "No user registered with this email!"
            });
          }

          //Se ha creado anteriormente un token aleatorio
          user.resetPasswordToken = token;
          user.resetPasswordExpires = Date.now() + 3600000; // 1 hour

          user.save().then(function(user) {
            done(err, token, user);
            }, function(err){
               done(err, token, user);
          });
       });
    },
    function(token, user, done) {
      var mailOptions = {
        to: user.email.toLowerCase(),
        from: 'davsensan@gmail.com',
        subject: 'Redborder licensing management Password Reset',
        text: 'You are receiving this because you (or someone else) have
requested the reset of the password for your account in licensing
management of RedBorder.\n\n' +
          'Please click on the following link before one hour, or paste
this into your browser to complete the process:\n\n' +
          'https://' + req.headers.host + '/#/reset/' + token + '\n\n' +
          'If you did not request this, please ignore this email and your
password will remain unchanged.\n'
      };
      smtpTransport.sendMail(mailOptions,function(err) {
        res.status(200).json({
          success: true,
          message:    "An    e-mail    has    been    sent    to    "    +
user.email.toLowerCase() + " with further instructions."
        });
        done(err, 'done');
      });
```

```
    }
  ], function(err) {
    if (err) return next(err);
  });
});
```

Como se puede observar, en esta ruta lo primero que se realiza es validar que se han recibido los datos de forma correcta mediante la función *validateForgotForm* (En este caso solo se necesitará el campo email).
Una vez se ha validado correctamente, se hará uso de la función *waterfall* de la librería *async* (previamente instalada mediante *npm*). Esta función ayuda a ejecutar funciones asíncronas en JavaScript y en este caso se han ejecutado las siguientes funciones en orden:

- En primer lugar, mediante la función *randomBytes* de la librería *crypto* (previamente instalada) se crean 20 bytes hexadecimales de forma aleatoria que serán el *token* que se utilizará para esta solicitud de nueva contraseña.
- Tras esto se busca al usuario en la base de datos, (se enviará un error 400 en el caso de no encontrarlo con un mensaje de error), y se cambian el valor de los campos *resetPasswordToken* y *resetPasswordExpires.*
- Si todo lo anterior ha ido correctamente, se enviará un email al usuario que está solicitando la renovación de la contraseña. En este email se enviará la dirección URL a la que tendrá que acceder para ello con el token creado. Por qué se usa esta URL se explicará más adelante.

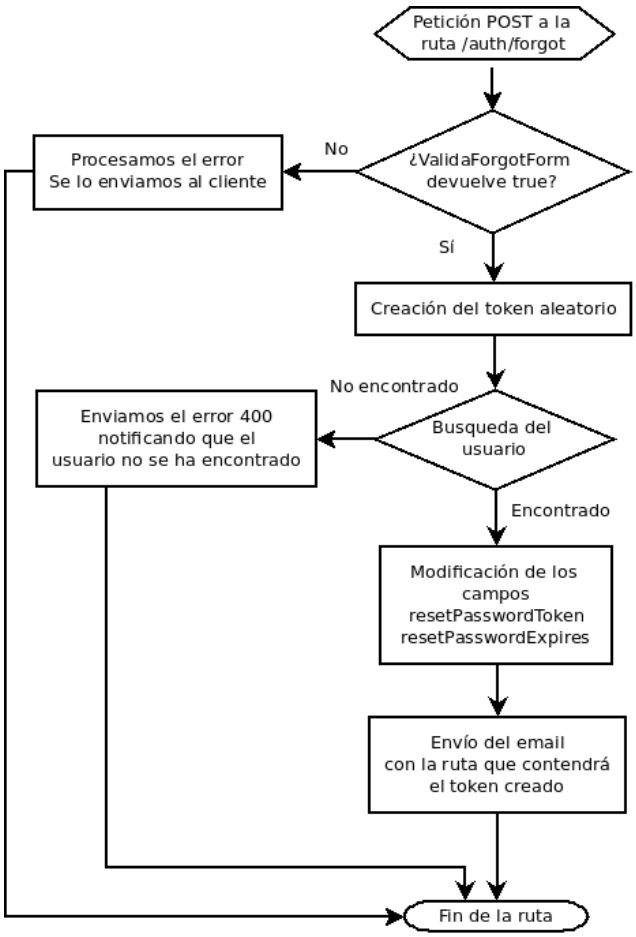


Figura 4-5. Diagrama de flujo del *middleware* de la ruta POST /auth/forgot

Para el envío de emails se ha utilizado la librería *nodemailer*, y para usarla se ha creado en primer lugar con qué servidor se enviará el email como se muestra a continuación:

Código 4-31. Configuración del objeto encargado del transporte de emails. (server/routes/auth.js)

```
{...}
//Configuramos el envío de emails
const transportMock = mockTransport({ //Configuramos el mock para el envío de correos
              service: process.env.EMAIL_SERVER || email.server,
              auth: {
                user: process.env.EMAIL_USER || email.email,
                pass: process.env.EMAIL_PASSWORD || email.password
              }
            });
const smtpTransport = MODE_RUN=="test" ? //Si estamos en modo test utilizamos el mock
            nodemailer.createTransport(transportMock)
            :
            nodemailer.createTransport({
              service: process.env.EMAIL_SERVER || email.server,
              auth: {
                user: process.env.EMAIL_USER || email.email,
                pass: process.env.EMAIL_PASSWORD || email.password
              }
            });
{...}
```

Este fragmento de código se ha extraído del inicio del fichero *Auth.js* y se puede observar que si la variable de entorno MODE_RUN es para test se crea un transporte usando la función *mockTransport* de la librería *nodemailer-mock-transport.* Esto se realiza para no enviar emails durante las pruebas.
Sin embargo, si la variable MODE_RUN no es para test, se crea un transporte de forma correcta, obteniendo los datos o bien de las variables de entorno, o bien del fichero de configuración por defecto (config/config.json).

4.5.1.2 Ruta para crear nueva contraseña

Una vez se ha solicitado la nueva contraseña se dispone de una hora hasta que se realice la creación de una nueva contraseña. La ruta a la que hay que mandar la petición para el creación de una nueva contraseña será la ruta *auth/reset/:token.*
Esta ruta es algo especial ya que se le envía un parámetro en la propia URL, este parámetro será el *token* que se ha generado de forma aleatoria anteriormente al solicitar el cambio de contraseña

y es por este motivo por el que en la URL que se envía en el email de recordatorio de contraseña contiene el *token* generado detrás.

Código 4-32. Ruta para creación de nueva contraseña auth/reset/:token (POST). (server/routes/auth.js)

```
{...}
router.post('/reset/:token', function(req, res) {
  const validationResult = validateNewPasswordForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  async.waterfall([
    function(done) {
      models.User.findOne({
                    where: {
                        resetPasswordToken: req.params.token,
                        resetPasswordExpires: {$gt: Date.now()}
                    }
                })
      .then(function(user, err){
        if (!user) {
          return res.status(400).json({
          success: false,
          message: "Password reset token is invalid or has expired."
          });
        }

        user.password = req.body.password;
        user.resetPasswordToken = null;
        user.resetPasswordExpires = null;

        user.save().then(function(user, err) {
            done(err, user);
          });
        });
    },
    function(user, done) {
```

```
      var mailOptions = {
        to: user.email.toLowerCase(),
        from: 'davsensan@gmail.com',
        subject: 'Your password has been changed in licensing management
of RedBorder',
        text: 'Hello,\n\n' +
          'This is a confirmation that the password for your account ' +
user.email.toLowerCase() + ' has just been changed.\n'
      };
      smtpTransport.sendMail(mailOptions,function(err) {
        res.status(200).json({
        success: true,
        message: "An e-mail has been sent to " + user.email.toLowerCase()
+ " with confirmation. The password has been changed"
        });
        done(err, 'done');
      });
    }
  ], function(err) {
    if (err) return next(err);
  });
});
{...}
```

En esta ruta se espera recibir, además del *token* en la URL, la contraseña nueva junto con la confirmación de la contraseña y por tanto en primer lugar, haciendo uso de la función *validateNewPasswordForm* (definida en el mismo fichero), se comprueba si se suministran la contraseña y la confirmación, y si ambas coinciden.
En caso de que se valide de forma correcta:

- Se busca el usuario cuyo *resetPasswordToken* coincida con el obtenido de la URL y además el campo resetPasswordExpires sea inferior a la fecha actual. Si no existe ningún usuario que cumpla esto, se devuelve un error 400 notificando que el token no existe o ha expirado.
- Si el token existe, se modifica la contraseña del usuario y los campos *resetPasswordTok* y *resetPasswordExpires* se borran (se ponen a null).
- Finalmente, se le envía un email de confirmación al usuario para que tenga constancia de que su contraseña ha sido cambiada.

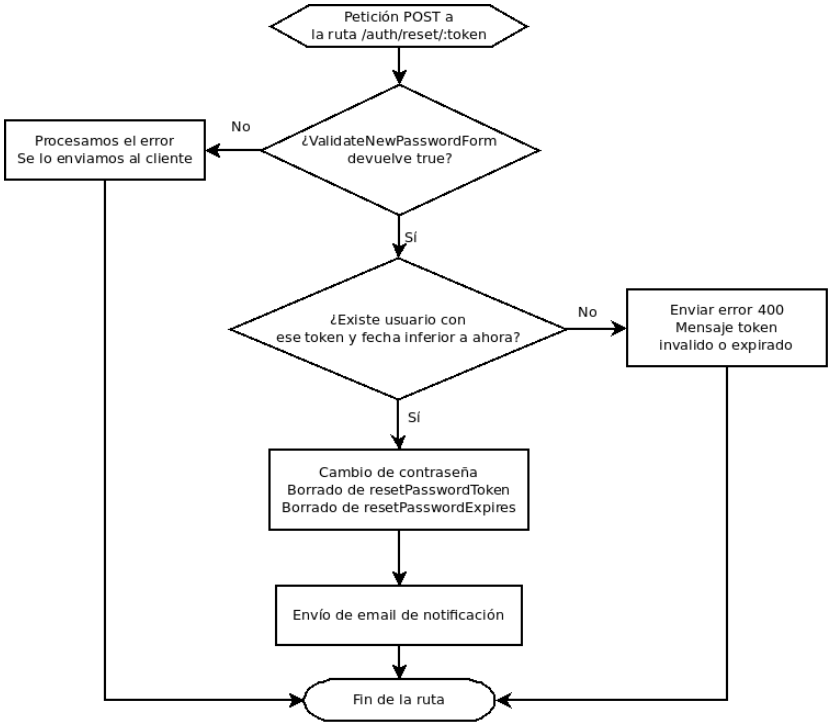


Figura 4-6. Diagrama de flujo del *middleware* de la ruta POST /auth/reset/:token

4.5.2 Código de *Front-end*

Una vez se han creado las dos rutas necesarias en el servidor para la restauración de la contraseña, es el momento de crear los componentes necesarios para las vistas en el cliente.
Estos componentes son *ForgotPage* y *NewPasswordPage*. Ambos componentes serán contenedores que harán uso de los componentes de presentación (*presentationals components*) *ForgotForm* y *NewPasswordForm*, respectivamente.

4.5.2.1 React router para la solicitud de nueva contraseña

Para poder usar dichos componentes se hará uso de *react router*, y para ello se ha creado una nueva entrada en el fichero de rutas (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/forgot se muestre el formulario donde introducir la dirección de correo electrónico para la cual se está solicitando el recordatorio de la contraseña, y cuando se acceda a la dirección http://host:port/reset/:token (donde :token se sustituya por el *token* aleatorio creado) aparezca el formulario para crear una nueva contraseña para el usuario que lo solicita.

Código 4-33. Fragmento con las rutas del formulario para recordar contraseña. (client/src/routes.js)

```
{...}
{
      path: '/forgot',
      component: ForgotPage
    } ,
    {
      path: '/reset/:token',
      component: NewPasswordPage
    },
```

```
{...}
```

Como se puede observar, en este fichero se hace uso de los compontes *ForgotPage* y *NewPasswordPage* antes mencionados.

Para acceder a la URL de recordatorio de contraseña, en el formulario de inicio de sesión, existe en la parte inferior un enlace de tipo *Link* de *react router*.
Sin embargo, para acceder a la URL de creación de una nueva contraseña se enviará un email en el que vendrá ya escrita directamente la URL, de tal forma que contenga el *token.*

4.5.2.2 Componente *ForgotForm*

El componente *ForgotForm* será el encargado de mostrar el formulario para solicitar la renovación de la contraseña olvidada y, al igual que todos los formularios que se han creado a lo largo del proyecto, hará uso de la librería *react-bootstrap,* importando de dicha librería los componentes necesarios para la creación del formulario.
En este caso solo se creará un campo en el cual el usuario introducirá su email y tendrá un botón para enviar el formulario. Además aceptará los siguientes parámetros obligatorios:

- Una función de *callback* (*onSubmit*) que se llame cuando se pulse el botón para enviar el formulario.
- Una función de *callback* (*onChange*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario.
- Un objeto (*errors*) que contendrá si algún campo del formulario tiene error para que se muestre de forma visual en el navegador.
- Un objeto (*user*) que contendrá el valor recogido en el campo del formulario.

Para ver el código detallado se puede dirigir a los anexos. Código A-9.

4.5.2.3 Componente *ForgotPage*

El componente *ForgotPage* será, por tanto, un contendor que hará uso del componente *ForgotForm* y donde se definan las funciones de *callback* y los estados iniciales de los objetos donde se guardará el email y los errores del formulario.
En este caso, tanto el constructor como la función *onChange* no tienen ninguna complejidad ya que los objetos inicialmente están vacíos y la función se encargará de rellenar el campo que haya cambiado en el formulario (en este caso siempre será el email).
Por lo tanto, de este componente la función más importante será la función *onSubmit* que se encargará, mediante AJAX, de invocar al método para solicitar la renovación de la contraseña:

Código 4-34. Función *onSubmit* del componente *ForgotPage*. (client/src/containers/ForgotPage.jsx)

```
{...}
  processForm(event) {
    // Prevención del envío por defecto
    event.preventDefault();

    // Cadena que contiene el email codificado en URI
    const email = encodeURIComponent(this.state.user.email);
    const formData = `email=${email}`;

    // Creación de la petición AJAX
```

```
    const xhr = new XMLHttpRequest();
    xhr.open('post', '/auth/forgot');
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
     if (xhr.status === 200) {
        // En caso de éxito se borra el estado
        this.setState({
          errors: {}
        });
        //Notificamos que se ha hecho correctamente
        {xhr.response.message && toastr.success(xhr.response.message)}
        // Volvemos a la url raiz /
        this.context.router.replace('/');
      } else {
        // Cambiamos los errores y los notificamos
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);
  }
{...}
```

Como se puede observar, en la función anterior se crea la cadena que contendrá el email codificado de forma que sea posible escribirlo en la URL y tras esto lanza una petición POST a la ruta /auth/forgot. Al recibir la respuesta realizamos lo siguiente:

- Si se recibe un 200 OK, es decir, si todo ha ido bien mostramos una notificación mediante un *toast*, eliminamos los posibles errores que pudieran existir y redirigimos a la página principal.
- Si no se ha recibido un 200 OK, es decir, si algo ha fallado, almacenamos el error y lo mostramos con un *toast*.

4.5.2.4 Componente *NewPasswordForm*

El componente *NewPasswordForm* será el encargado de mostrar el formulario para crear una nueva contraseña y, al igual que todos los formularios que se han creado a lo largo del proyecto, hará uso de la librería *react-bootstrap*, importando de dicha librería los componentes necesarios para la creación del formulario.

En este caso existirá lo siguiente:

- Un campo que contendrá la nueva contraseña.
- Un campo que contendrá la confirmación de la nueva contraseña.
- Un botón para confirmar la renovación de la contraseña.

Ademas aceptará los siguientes parámetros obligatorios:

- Una función de *callback* (*onSubmit*) que se llame cuando se pulse el botón para enviar el formulario.
- Una función de *callback* (*onChange*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario.
- Un objeto (*errors*) que contendrá si algún campo del formulario tiene error para que se muestre de forma visual en el navegador.
- Un objeto (*user*) que contendrá el valor recogido en los campos del formulario.

Para ver el código detallado se puede dirigir a los anexos. Código A-15.

4.5.2.5 Componente *NewPasswordPage*

El componente *NewPasswordPage* será, por tanto, un contendor que hará uso del componente *NewPasswordForm* y donde se definan las funciones de *callback* y los estados iniciales de los objetos donde se guardaran la nueva contraseña y la confirmación de la nueva contraseña, además de los errores del formulario.

En este caso, el constructor no tiene ninguna complejidad, ya que los objetos inicialmente están vacíos.
Sin embargo, en la función *onChange* de este componente se comprueba que las contraseñas coincidan, además de que ambas estén rellenas y tengan una longitud entre 8 y 16 caracteres. De esta forma, si no coinciden o alguna no cumplen la longitud, se cambiará el objeto *errors* para notificar visualmente al usuario.

Código 4-35. Función *onChange* de NewPassword*Page*. (client/src/containers/NewPaswordPage.jsx)

```
changeUser(event) {
  const field = event.target.name;
  const user = this.state.user;
  user[field] = event.target.value;
  this.setState({
    user
  });
  //Esto es para validar el formulario visualmente
  if(this.state.user.password !== this.state.user.confir_password){
    this.state.errors.password="error";
    this.state.errors.confir_password="error";
  }
  else{
    this.state.errors.password="success";
    this.state.errors.confir_password="success";
  }
```

```
    if(this.state.user.password.length<8 ||
this.state.user.password.length > 15)
      this.state.errors.password="error";
    if(this.state.user.confir_password.length<8 ||
this.state.user.confir_password.length > 15)
      this.state.errors.confir_password="error";
  }
```

Por otro lado, de este componente la función *onSubmit* se encargará, mediante AJAX, de invocar al método para solicitar la creación de la nueva contraseña:

Código 4-36. Función *onSubmit* de *NewPasswordPage*. (client/src/containers/NewPassword.jsx)

```
{...}
    processForm(event) {
    // Prevención del envío por defecto
    event.preventDefault();
    // Cadena que contiene el email codificado en URI
    const                          confir_password                          =
encodeURIComponent(this.state.user.confir_password);
    const password = encodeURIComponent(this.state.user.password);
    const                               formData                            =
`password=${password}&confir_password=${confir_password}`;
    // Creación de la petición
    const xhr = new XMLHttpRequest();
    //El token se le pasa como parametros en la URL
    xhr.open('post', '/auth/reset/' + this.props.params.token );
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        //Borramos los posibles errores
        this.setState({
          errors: {}
        });
        //Notificamos con un toast que todo ha ido bien
        {xhr.response.message && toastr.success(xhr.response.message)}
        //Volvemos a la página principal
        this.context.router.replace('/');
      } else {
```

```
        //En caso de fallo cambiamos el objeto errors y lo notificamos
con un toast
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);
  }
{...}
```

Como se puede observar, en la función anterior se crea la cadena que contendrá la contraseña y la confirmación de la contraseña codificadas de forma que sea posible escribirlas en la URL, y tras esto lanzará una petición POST a la ruta /auth/reset/:token (Se puede apreciar que el token se añade a la URL). Al recibir la respuesta realizaremos lo siguiente:

- Si se recibe un 200 OK, es decir, si todo ha ido bien mostramos una notificación mediante un *toast*, eliminamos los posibles errores que pudieran existir y redirigimos a la página principal.
- Si no se ha recibido un 200 OK, es decir, si algo ha fallado, se almacenará el error y se mostrará con un *toast.*

4.6 Creación de usuarios

En este apartado, se explicará cómo se ha implementado una solución para la creación de usuarios por parte de usuarios que sean administradores.

4.6.1 Código de *back-end*

Para llevar a cabo la creación de usuarios, en el lado del servidor ha sido necesario, en primer lugar, crear una estrategía de *passport.* Sin embargo, esta estrategia ya se explicó en el apartado 4.3.1 de "estrategías de *passport"* por lo que ahora se procederá a utilizar mediante la ruta para la creación de usuarios, de una forma similar a como se utilizó la estrategia de inicio de sesión en la ruta /auth/login.

Además, al ser los usuarios pertenecientes a una organización, será necesario conocer todas las organizaciones que existen en el sistema antes de proceder a la creación de un usuario, por lo que también hay que definir una ruta para obtener todas las licencias. Esta ruta será accedida por el método GET en la url /api/users/new

4.6.1.1 Ruta para la creación de usuario

Ya que en el requisito se pide que sólo los usuarios que sean administradores puedan crear otros usuarios, esta ruta ha de estar definida dentro del fichero *api.js* para que solo usuarios que estén autenticados en el sistema puedan acceder a ella. Además, en dicha ruta habrá que comprobar de alguna manera que el usuario que está solicitando la creación de un usuario es administrador.
A continuación, se explicará detalladamente cómo se ha desarrollado dicha ruta:

Código 4-37. Ruta para creación de un nuevo usuario api/users (POST). (server/routes/api.js)

```
{...}
router.post('/users', (req, res, next) => {
  const validationResult = validateCreateUserForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
        return passport.authenticate('local-signup', (err) => {
          if(err){
            if (err.message=="Validation error") {
                return res.status(409).json({
                  success: false,
                  message: "This email is already registered"
                });
              }
            else {
                  return res.status(409).json({
```

```
                  success: false,
                  message: err.message
                });
              }
          }
            const mailOptions = {
              to: req.body.email.toLowerCase(),
              from: 'davsensan@gmail.com',
              subject: 'Your email has been registered in RedBorder licenses',
              text: 'Hello,\n\n' +
                'You have been registered in RedBorder. Your email is ' + req.body.email.toLowerCase() + ' and your password is ' + req.body.password + '.\n'
                 + "Please, log in and change your password"
            };
            smtpTransport.sendMail(mailOptions,function(err) {
              res.status(200).json({
              success: true,
              message: 'User registered successfully. A email has been send to ' + req.body.email.trim() + ' with the password'
              });
            });


        })(req, res, next);
      }
    })
});
{...}
```

Como se puede observar, en esta ruta lo primero que se hace es validar si se han enviado todos los datos de forma correcta para la creación de un usuario, mediante la función validateCreateUserForm (definida en el mismo fichero api.js):

- Si la validación no fuera correcta se enviaría un error 400 con el mensaje devuelto por dicha función.
- Si la validación es correcta, se busca al usuario que está autenticado mediante el identificador que previamente se ha decodificado del token.
- Si dicho usuario no existe, se enviará un error 404 notificando que ese usuario no existe.
- Si el usuario existe, pero no es administrador, se enviará un error 401 notificando que no tiene permisos.
- Si el usuario existe y es administrador, se hace uso de la estrategia local-signup para registrar a un usuario.
- Si la estrategia devuelve error, se comprueba si es un error de validación y en ese caso significará que

el email ya existe y se devolverá un error 409 (Conflicto).

- Si la estrategia devuelve error, pero no es de validación, se devuelve un error 409 con el mensaje que contenga.
- En caso de que todo lo anterior haya ido bien, al igual que se hizo al renovar la contraseña, se le enviará un email al usuario creado con su contraseña y pidiéndole que, por favor, la cambie.

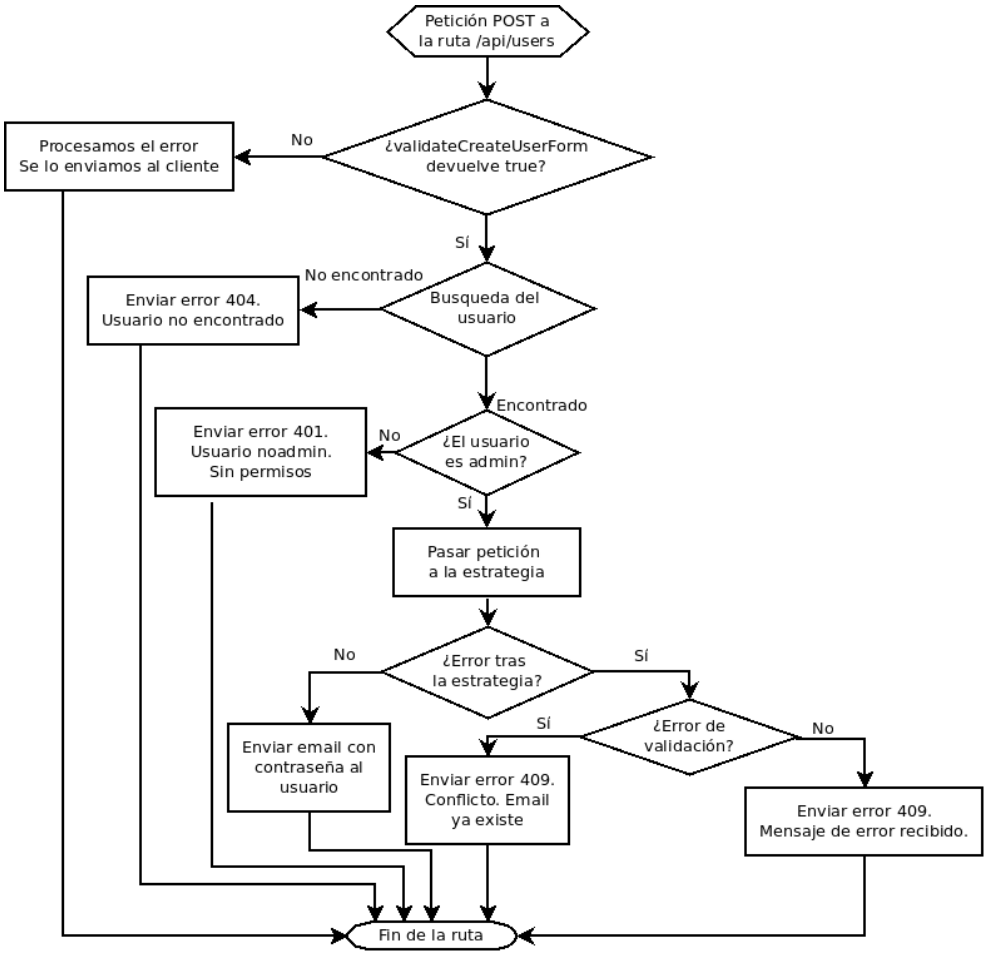


Figura 4-7. Diagrama de flujo del *middleware* de la ruta POST /api/user

4.6.1.2 Ruta para la obtención de todas las organizaciones

Además de la ruta que crea un usuario, como se dijo anteriormente, es necesario conocer todas las organizaciones que existen en el sistema para poder crear un usuario y que pertenezca a la organización que elijamos. Para eso se ha decidido por optar una ruta GET en la url /api/users/new de tal modo que al acceder a ella un usuario administrador devuelva la lista con todas las organizaciones.

Código 4-38. Ruta para obtención de las organizaciones api/users/new (GET). (server/routes/api.js)

```
{...}
router.get('/users/new', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
```

```
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.Organization.findAll({
          order: 'name'
        }).then(function(list_orgs){
          return res.status(200).json({
            success: true,
            orgs: list_orgs,
          })
        })
      }
    })
  });
{...}
```

Como se puede observar, en la ruta anterior lo único que se comprueba es si el usuario que está accediendo existe y es administrador, en cuyo caso se devuelven todas las organizaciones mediante una respuesta JSON

4.6.2 Código de *Front-end*

Una vez se han creado la ruta para el registro de un usuario por parte de un administrador en el lado del servidor y la ruta para la obtención de todas las organizaciones, es el momento de crear la interfaz, del lado del cliente, encargada de hacer uso de dicha ruta.

4.6.2.1 React router para la creación de un usuario

Para ello, se ha creado una nueva entrada en el fichero de rutas de *react router* (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/user/new se muestre el formulario para introducir los datos del usuario que se desea crear.

Código 4-39. Fragmento con la ruta para la creación de un usuario en el cliente. (client/src/routes.js)

```
{...}
    {
      path: '/user/new',
      component: CreateUserPage
    },
{...}
```

Como se puede observar en el fragmento anterior, existe un componente llamado *CreateUserPage* que será el que se muestre cuando accedamos a la ruta /user/new. A esta ruta se podrá acceder de un modo sencillo mediante un botón que existirá en la vista encargada de listar todos los usuarios (Se explicará más adelante esta vista).

4.6.2.2 Componente *CreateUserForm*

El componente *CreateUserForm* será el encargado de mostrar el formulario para la creación de un usuario y seguirá la metodología de todos los formularios que se han creado a lo largo del proyecto.
En este formulario existirán los siguientes campos:

- Un campo que contendrá el nombre del usuario a crear.
- Un campo que contendrá el email del usuario a crear.
- Un campo con la contraseña para el usuario
- Un campo con la confirmación de la contraseña para el usuario
- Una lista en la que seleccionar la organización a la que pertenece el usuario (por defecto no pertenece a ninguna)
- Un *checkbox* en el que marcar si el usuario es o no administrador.
- Un botón para confirmar la creación del usuario.

Además, aceptará los siguientes parámetros obligatorios:

- Una función de *callback* (*onSubmit*) que se llame cuando se pulse el botón para enviar el formulario.
- Una función de *callback* (*onChange*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario.
- Un objeto (*errors*) que contendrá si algún campo del formulario tiene error para que se muestre de forma visual en el navegador.
- Un objeto (*user*) que contendrá el valor recogido en los campos del formulario.
- Una tabla de objetos (organizations) que contendrá todas las organizaciones existentes en la base de datos para mostrarlas en el select.

Para ver el código detallado se puede dirigir a los anexos. Código A-4.

4.6.2.3 Componente *CreateUserPage*

El componente *CreateUserPage* será, por tanto, un contendor que hará uso del componente *CreateUserForm* y donde se definan las funciones de *callback* y los estados iniciales de los objetos que guardarán el usuario a crear y los errores del formulario.

En este caso, el constructor no tiene ninguna complejidad, ya que los objetos inicialmente están vacíos, excepto que por defecto el usuario no es administrador.

Sin embargo, para obtener la lista de organizaciones antes de pasársela al componente *CreateUserForm* hay que introducir código en un método que se ejecuta justo antes de mostrar el componente. Este es el método *componentWillMount* y dentro de él se realiza lo siguiente:

Código 4-40. *componentWillMount* de CreateUserPage. (client/src/containers/CreateUserPage.jsx)

```
{...}
//Justo antes de renderizar el componente se llamará a este método
  componentWillMount(){
    //Utilizando ajax, pedimos la lista de todas las organizaciones registradas
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión get
    xhr.open('get', '/api/users/new');
    // Configuramos el token para la autorización
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    // La respuesta se espera que sea un JSON
    xhr.responseType = 'json';
    // Añadimos el callback para cuando se reciba la respuesta de forma correcta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito
        // Cambiamos el estado, eliminando los errores y almancenando las organizaciones
        this.setState({
          error: "",
          organizations: xhr.response.orgs
        });

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de fallo, mediante un toast informamos del mensaje de error
          {xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
```

```
    //Enviamos la petición
    xhr.send();
  }
{...}
```

Se puede observar que dentro de este método lo único que se realiza es crear una petición al método GET /api/users/new mediante AJAX y en la respuesta, si no hay error, almacenar las organizaciones en la tabla de objetos *organizations* del estado del componente para después pasársela al componente *CreateUserForm.*

La función de *callback, onChange*, es muy similar a los vistos hasta ahora, en primer lugar, se almacena el valor del campo que ha cambiado en función del nombre del evento y después se comprueba si todos los campos son correctos (están rellenos, las contraseñas coinciden y miden mas entre 8 y 16…). Si se quiere ver en profundidad, estará recogido en los anexos (código A-20), no se introducirá aquí para disminuir la extensión de la memoria.

Por último, la función más importante de este componente será el *callback onSubmit:*

Código 4-41. Función *onSubmit* de *CreateUserPage*. (client/src/containers/CreateuserPage.jsx)

```
processForm(event) {
    //Prevenimos el envío del formulario vacío y por defecto
    event.preventDefault();
    //Creamos una cadena de carácteres par enviar en el método post los parámetros introducidos en el formulario
    const name = encodeURIComponent(this.state.user.name);
    const email = encodeURIComponent(this.state.user.email);
    const password = encodeURIComponent(this.state.user.password);
    const confir_password =
 encodeURIComponent(this.state.user.confir_password);
    const organization =
 encodeURIComponent(this.state.user.organization);
    const role = this.state.user.admin ? "admin" : "normal";
    const formData =
`role=${role}&name=${name}&organization=${organization}&email=${email}&password=${password}&confir_password=${confir_password}`;

    //Creación de la petición AJAX para la creación de un usuario
    const xhr = new XMLHttpRequest();
    //Abrimos la conexión post con el servidor
    xhr.open('post', '/api/users');
    //Modificamos la cabecera para indicar que será el envío de un formulario
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
```

```
urlencoded');
    //Configuramos el token que identifica al usuario que está realizando
la petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    //Esperaremos una respuesta JSON
    xhr.responseType = 'json';
    //Función que se ejecutará al recibir la respuesta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Si se ha recibido un 200 ok, notificamos con un mensaje que se
ha creado correctamente el usuario
        {xhr.response.message  &&  toastr.success(xhr.response.message)}
//Contendrá el mensaje recibido
        //  Cambiamos  los  valores  de  los  errores  en  el  estado  del
componente para indicar que no hay errores
        this.setState({
          errors: {}
        });
        //Redirigimos al listado de usuarios
        this.context.router.replace('/listUsers/all/all');
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de fallo mostramos el mensaje de error recibido del
servidor
        {xhr.response.message && toastr.error(xhr.response.message)}
        // Cambiamos los errores por los errores recibidos
        const errors = xhr.response.errors ? xhr.response.errors : {};
        //  El  mensaje  de  respuesta  recibido  será  un  resumen  de  los
errores que se han producido
        errors.summary = xhr.response.message;
        // Cambiamos el estado de los errores
        this.setState({
          errors
        });
      }
    });
    //Enviamos la petición
    xhr.send(formData);
```

```
  }
```

En esta función, se realiza lo siguiente:

- En primer lugar, se crea una cadena que contendrá todos los datos introducidos en el formulario codificados para que puedan enviarse en una URL.
- Se crea una petición AJAX a la ruta POST /api/users/ añadiendo el token a la cabecera.
- Cuando se recibe la respuesta, si todo ha ido bien se eliminan los errores, se notifica con un *toast* y se redirige al listado de usuarios de todos los usuarios (Se verá más adelante).
- En el caso de haber un error 404 en la respuesta, querrá decir que el usuario, aunque tenga el token correcto, no existe en la base de datos y por tanto se cerrará la sesión. En cualquier otro caso se notificará con un *toast* el error recibido y se almacenará el error.

4.7 Creación de organizaciones

En este apartado, se explicará como se ha implementado una solución para la creación de organizaciones por parte de usuarios que sean administradores.

4.7.1 Código de *back-end*

Para la creación de una organización sólo ha sido necesario definir una ruta en el lado del servidor, a la cual un usuario que esté autenticado y además sea administrador, podrá enviarle una petición para añadir una organización a la base de datos.
A esta ruta se accederá meditan el método POST y en la URL /api/organizations.

Código 4-42. Ruta para crear una organización api/organizations (POST). (server/routes/api.js)

```
{...}
router.post('/organizations', (req, res) => {
  const validationResult = validateCreateOrgForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
          return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
```

```
          });
      }
      else if (user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          const NewOrganization = models.Organization.build({
            cluster_id: req.body.cluster_id.trim(),
            name: req.body.name.trim(),
            email: req.body.email.trim(),
            sensors: req.body.sensors.trim()
          });
          NewOrganization.save().then(function(NewOrganization) {
            return res.status(200).json({
              success: true,
              message: 'Organization ' + NewOrganization.name + ' created
correctly'
            });
          }, function(err){
            return res.status(400).json({
              success: false,
              message: 'Error saving organization ' + req.body.name + '.
Email already exists.'
            });
        });
      }
    })
});
{...}
```

Como se puede observar, cuando llega una petición POST a dicha ruta se realiza lo siguiente:

- En primer lugar, se comprueba si los datos recibidos son los necesarios y los correctos haciéndose uso de la función *validaCreateOrgForm* (definida en el mismo fichero). En el caso de que esta función devuelva un error, se le enviaría al cliente mediante un mensaje HTTP 400.
- Si todos los campos son correctos, se comprobará si el usuario que está solicitando la creación existe y si además es administrador (Del mismo modo que se hizo para la ruta de creación de un usuario). En el caso de que el usuario no exista, como sucede en todas las rutas, se devolverá un error HTTP 404.

Por otro lado, si el usuario no es administrador se le devolverá un erro HTTP 401 indicándole que no tiene los permisos necesarios para realizar la operación.

- Si el usuario existe y, además, es administrador, se intentará almacenar una organización en la base de datos. Esto no se realizará si el email de dicha organización ya existe, y en dicho caso, se notificará al cliente con un mensaje de error HTTP 409 (conflicto).
- Sin embargo, si todo lo anterior ha ido bien se enviará un mensaje HTTP 200 indicándose que dicha organización se ha creado con éxito.

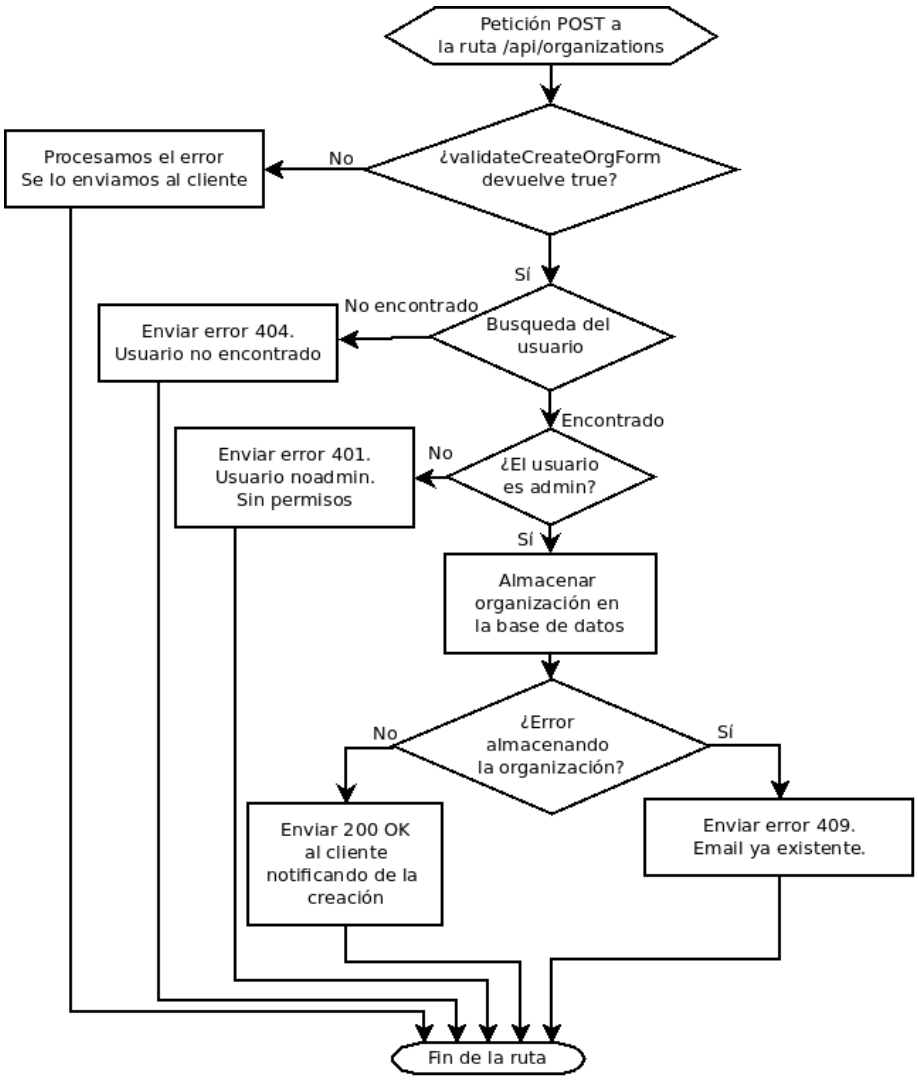


Figura 4-8. Diagrama de flujo del *middleware* de la ruta POST /api/organizations

4.7.2 Código de *Front-end*

Una vez se ha creado la ruta para almacenar una organización en la base de datos en el lado del servidor, es el momento de crear la interfaz, para el lado del cliente, encargada de hacer uso de dicha ruta.

4.7.2.1 React router para la creación de una organización

Para ello, se ha creado una nueva entrada en el fichero de rutas de *react router* (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/organizations/new se muestre el formulario para introducir los datos de la organización que se desea crear.

Código 4-43. Fragmento con la ruta para crear una organización en el cliente. (client/src/routes.js)

```
{...}
    {
      path: '/organization/new',
      component: CreateOrgPage
    },
{...}
```

Como se puede observar en el fragmento anterior, existe un componente llamado *CreateOrgPage* que será el que se muestre cuando accedamos a la ruta /organizations/new.
A esta ruta se podrá acceder de un modo sencillo mediante un botón que existirá en la vista encargada de listar todas las organizaciones (Se explicará más adelante esta vista).

4.7.2.2 Componente *CreateOrgForm*

El componente *CreateOrgForm* será el encargado de mostrar el formulario para la creación de una organización y seguirá la metodología de todos los formularios que se han creado a lo largo del proyecto.
En este formulario existirán los siguientes campos:

- Un campo que contendrá el nombre de la organización a crear.
- Un campo que contendrá el email de la organización a crear.
- Un campo con el identificador del cluster de la organización a crear.
- Una lista de sensores (y código) disponibles en esa organización.
- Un botón para confirmar la creación de la organización.

Ademas aceptará los siguientes parámetros obligatorios:

- Una función de *callback* (*onSubmit*) que se llame cuando se pulse el botón para enviar el formulario.
- Una función de *callback* (*onChange*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario.
- Un objeto (*errors*) que contendrá si algún campo del formulario tiene error para que se muestre de forma visual en el navegador.
- Un objeto (*org*) que contendrá el valor recogido en los campos del formulario.

Para ver el código detallado se puede dirigir a los anexos. Código A-3.

4.7.2.3 Componente *CreateOrgPage*

El componente *CreateOrgPage* será, por tanto, un contendor que hará uso del componente *CreateOrgForm* y donde se definan las funciones de *callback* y los estados iniciales de los objetos que guardarán la organización a crear y los errores del formulario.

En este caso, el constructor no tiene ninguna complejidad, ya que los objetos inicialmente están vacíos, excepto el campo para la lista de sensores que por defecto contendrá "IPS,event;Flow,flow;Social,social", de tal modo que toda organización, por defecto, tenga los sensores IPS, Flow y Social.

Por otro lado, tanto la función de *callback* que se llama cada vez que hay un cambio en las entradas del formulario, como la que envía la petición cuando se pulsa el botón de "crear

organización" son muy similares a las vistas anteriormente para la creación de un usuario y por ello no se mostrarán aquí. Puede verlas detalladamente en los anexos. Código A-19.

4.8 Creación de licencias

En los requisitos, se expuso que para la creación de una licencia el usuario debe pertenecer a la organización para la cual está creando la licencia, o bien, ser administrador. A lo largo de este apartado se explicará cómo se ha implementado una solución para dicho requisito.

4.8.1 Código de *back-end*

4.8.1.1 Ruta para obtener sensores disponibles para una organización

Debido a que la creación de licencias puede ser diferente para una organización o para otra, ya que cada una puede tener tipos de sensores diferentes, de algún modo será necesario conocer qué sensores tiene cada organización antes de crear la licencia.
Para ello se ha optado por crear una ruta a la que enviarle una petición GET y obtener la lista de sensores para la organización por la cual se le está preguntando:

- Esta ruta estará disponible en la URL /api/licenses/new
- Acepta como parámetro el identificador de la organización por la que se está preguntando.
- Comprueba si el usuario que la solicita pertenece a la organización que solicita o es administrador
- Devuelve, si no ha habido error, una cadena con la lista de sensores separados entre ellos por ";"

Para ver detalladamente la ruta, puede dirigirse a los anexos. Código B-13.

4.8.1.2 Ruta para solicitar la creación de una licencia

Por otro lado, y como sucede con la creación de usuarios y organizaciones, también ha sido necesaria la creación de una ruta a la cuál enviar una petición para que se cree y almacene en la base de datos una licencia. Esta ruta será accedida mediante el método POST en la URL /api/licenses.

Código 4-44. Ruta para crear una licencia api/licenses (POST). (server/routes/api.js)

```
{...}
router.post('/licenses', (req, res) => {
  const validationResult = validateCreateLicenseForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
```

```
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      //Si la licencia que se quiere crear no es para la organización a la que pertenecemos... no podemos
      else if(user.OrganizationId != req.body.OrganizationId && user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          //Si no se le ha pasado el license_uuid significa que estamos extendiendo una licencia, en ese caso el tiempo de expiración tambien existirá.
            const NewLicense = req.body.license_uuid ? models.License.build({
              expires_at: req.body.expires_at,
              license_uuid: req.body.license_uuid,
              duration: req.body.duration.trim(),
              limit_bytes: req.body.limit_bytes.trim(),
              OrganizationId: req.body.OrganizationId.trim(),
              UserId: user.id,
              sensors: req.body.sensors
            })
            :
             models.License.build({
              duration: req.body.duration.trim(),
              limit_bytes: req.body.limit_bytes.trim(),
              OrganizationId: req.body.OrganizationId.trim(),
              UserId: user.id,
              sensors: req.body.sensors
            })
```

```
          NewLicense.save().then(function(NewLicense) {
            return res.status(200).json({
              success: true,
              message: 'License created correctly'
            });
          }, function(err){
            return res.status(400).json({
              success: false,
              message: 'Error creating license.<br></br> The sensors and
limit bytes must be numbers'
            });
        });
      }
    })
  });
{...}
```

Cuando llega una petición a esta ruta se realiza lo siguiente:

- En primer lugar, se comprueba si los datos recibidos son los necesarios y los correctos haciéndose uso de la función *validaCreateLicenseForm* (definida en el mismo fichero). En el caso de que esta función devuelva un error, se le enviaría al cliente mediante un mensaje HTTP 400.
- Si todos los campos son correctos, se comprobará si el usuario que está solicitando la creación existe y si además es administrador o está solicitando la creación de la licencia para su organización. En el caso de que el usuario no exista, como sucede en todas las rutas, se devolverá un error HTTP 404. Por otro lado, si el usuario no es administrador y está intentado crear una licencia para una organización que no es la suya, se le devolverá un error HTTP 401 indicándole que no tiene los permisos necesarios para realizar la operación. (Para comprobar si está creándola para su organización se comprueba si el *OrganizationId* del usuario coincide con el que se le está pasando para crear la organización)
- Si no ha habido error, se intentará almacenar una licencia en la base de datos. En el caso de lanzar algún error al almacenarse en la base de datos la licencia se notifica al cliente con un mensaje HTTP 400.
- Sin embargo, si todo lo anterior ha ido bien se enviará un mensaje HTTP 200 indicándose que dicha licencia se ha creado con éxito.

Como se puede ver, antes de almacenar una licencia en la base de datos, se comprueba si el campo *license_uuid* está relleno, en cuyo caso se creará una licencia con dicho campo y el tiempo de expiración ya configurados.
Esto se verá detalladamente más adelante a la hora de extender una licencia, pero, en resumen, será lo que se use para diferenciar la extensión de una licencia de la creación, ya que en la creación, el campo *license_uuid* será nuevo y en la extensión el campo *license_uuid* será el de la licencia que se desea extender.

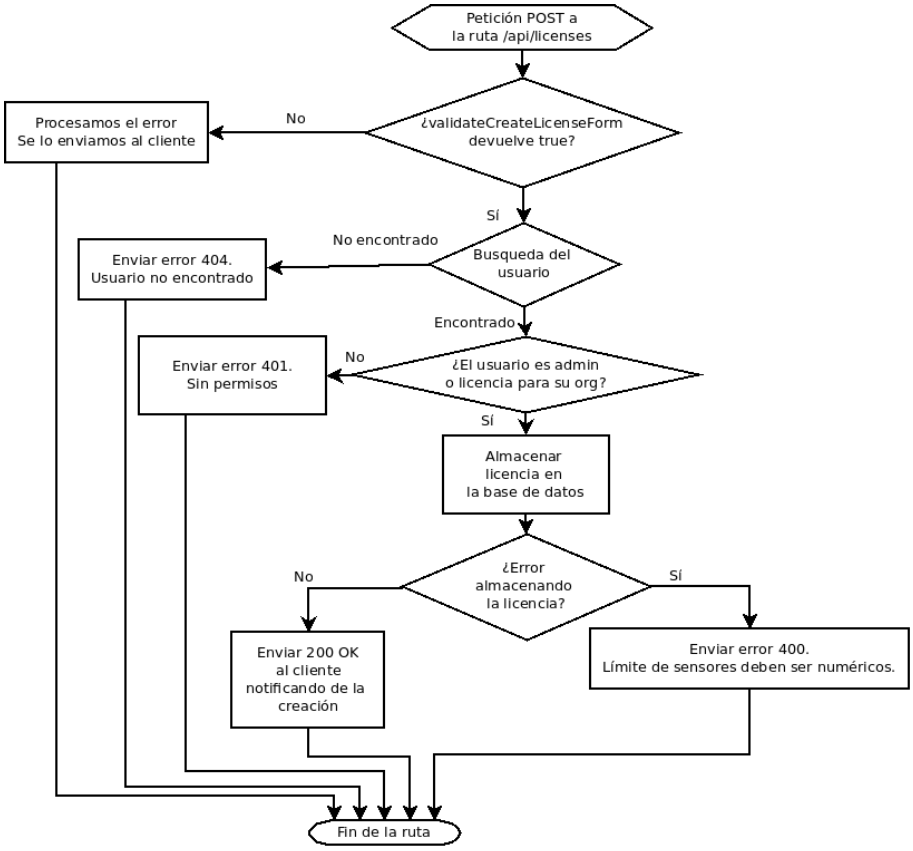


Figura 4-9. Diagrama de flujo del *middleware* de la ruta POST /api/licenses

4.8.2 Código de *Front-end*

Una vez se ha creado la ruta para almacenar una licencia en la base de datos en el lado del servidor y la ruta para obtener la lista de sensores disponibles para cada organización, es el momento de crear la interfaz, para el lado del cliente, encargada de hacer uso de dicha ruta.

4.8.2.1 React router para la creación de una licencia

Para ello, se ha creado una nueva entrada en el fichero de rutas de *react router* (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/license/new/:UserId/:OrgId se muestre el formulario para introducir los datos de la licencia que se desea crear.

Código 4-45. Fragmento con la ruta para crear una licencia en el cliente. (client/src/routes.js)

```
{...}
    {
      path: '/license/new/:UserId/:OrgId',
      component: CreateLicensePage
    },
{...}
```

Como se puede observar, la ruta anterior contendrá el identificador del usuario y de la organización que tendrá la licencia que se quiere crear en la URL. De este modo, cada vez que se quiera crear una licencia, se accederá mediante esta URL y se podrán obtener los campos *UserId* y *OrgId*. Además, existe un componente llamado *CreateLicensePage* que será el que se muestre cuando accedamos a la ruta y nos mostrará el formulario para la creación de la licencia.

Debido a que esta ruta no será "sencilla", ya que contendrá dos identificadores de tipo UUID en la URL, para acceder a ella existirá un botón en la vista encargada de listar todas las licencias para una organización que ya tendrá la URL formada con los datos del usuario que ha accedido a dicha vista y de la organización para la que se están listando las licencias. La creación de este botón se verá más adelante cuando se hable de la lista encargada de listar todas las licencias para una organización.

Resulta evidente pensar que, al pasar el identificador del usuario y de la organización por la URL, existirá un contra para la seguridad, por lo que tendrá que haber algún mecanismo que compruebe esto. Ese mecanismo no es más que validar el *token* en el servidor (mediante el *middleware* encargado de ello) y comprobar si el usuario es administrador o está creando la licencia para su propia organización, como se ha explicado en el apartado anterior para la ruta /api/licenses.

4.8.2.2 Componente *CreateLicenseForm*

El componente *CreateLicenseForm* será el encargado de mostrar el formulario para la creación de una licencia y seguirá la metodología de todos los formularios que se han creado a lo largo del proyecto.
En este formulario existirán los siguientes campos:

- Un campo de tipo selección que permitirá seleccionar 1, 3, 6 o 12 meses de duración de la licencia.
- Un campo de tipo numérico que permitirá introducir el número máximo de bytes al día para la licencia.
- Un campo numérico por cada tipo sensor para introducir el número de sensores de ese tipo.
- Un botón para confirmar la creación de la licencia.

Además, aceptará los siguientes parámetros obligatorios:

- Una función de *callback* (*onSubmit*) que se llame cuando se pulse el botón para enviar el formulario.
- Una función de *callback* (*onChange*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario.
- Una función de *callback* (*onChangeSensors*) que se ejecute cada vez que se produzca algún cambio en algún campo del formulario referente al número de sensores de cada tipo.
- Un objeto (*errors*) que contendrá si algún campo del formulario tiene error para que se muestre de forma visual en el navegador.
- Un objeto (*license*) que contendrá el valor recogido en los campos del formulario.
- Una tabla de cadenas (*sensors*) que contendrá el conjunto de los tipos de sensores diferentes que tendrá la organización para la que se está creando la licencia. Cada campo de esta tabla será el nombre del sensor separado por una coma (,) del número que identificará al sensor, por ejemplo, "Flow,300" podría ser un campo.

Debido a que en este formulario el número de campos no es estático, sino que depende del número de tipo sensores diferentes de los que dispone la organización para la que se está creando la licencia, a continuación, se mostrará cómo se crean los campos de forma dinámica:

Código 4-46. Campos sensores de forma dinámica. (client/src/components/CreateLicenseForm.jsx)

```
{...}
{
```

```
      sensors.map(function(sensor,key){
        return (<div key={key}>
          <FormGroup          controlId={sensor}          validationState=
{errors.sensors[sensor] == "" ? null : errors.sensors[sensor]}>
            <Col componentClass={ControlLabel} sm={2}>
              Sensors {sensor.split(",")[0]}
            </Col>
            <Col sm={10}>
              <input   className="col-sm-5"   name={sensor.split(",")[1]}
onChange={onChangeSensors} value={license[sensor]} type="number" min="0"
max="100" defaultValue="0"/>
            </Col>
          </FormGroup>
          </div>)
      })
}
{...}
```

En este fragmento lo que se realiza es la utilización de la función *map* para recorrer la tabla con los diferentes *sensores* que se le han pasado por parámetros al componente, y para cada campo de la tabla crear una entrada de formulario de tipo numérica con el nombre del tipo de sensor que se halle en dicha celda de la tabla.

4.8.2.3 Componente *CreateLicensePage*

El componente *CreateLicensePage* será un contendor que hará uso del componente *CreateLicenseForm* definirá las funciones de *callback* y los estados iniciales de los objetos que guardarán la organización a crear y los errores del formulario. Además, en este caso, obtendrá la tabla con los diferentes tipos de sensores disponibles para la organización que solicita una licencia, antes de mostrar el componente.

Este contenedor es más complejo que los vistos anteriormente y por tanto se explicará con más detalle cada una de sus partes nuevas. En primer lugar, hay que explicar los estados iniciales que existirán y cómo configurarán en el constructor del componente:

Código 4-47. Constructor de *CreateLicensePage*. (client/src/containers/CreateLicensePage.jsx)

```
{...}
constructor(props, context) {
{...}
// Configuramos los estados iniciales
    this.state = {
      errors: {
        duration: '',
        limit_bytes: '',
         sensors: {
```

```
        }
      },
      license: {
        duration: 1,
        limit_bytes: '0',
        sensors: {
        },
        OrganizationId: this.props.params.OrgId,
        UserId: this.props.params.UserId
      },
      sensors: ''
    };
{...}
```

Se puede observar, que en el constructor se crea:

- Un objeto (*errors*) que contendrá los errores en los campos del formulario (*duration*, *limit_bytes* y *sensors*)
- Un objeto (*license*) que tendrá inicialmente duración de un mes, y los campos referentes a la organización y el usuario se obtienen de la URL mediante *react router*.
- Una cadena (*sensors*) en la que se almacenará la lista de tipos de sensores disponibles para la organización para la que se está creando la licencia separadas por punto y coma (;).

Por otro lado, para rellenar el valor de la tabla de sensores se realizará lo siguiente en el método *componentWillMount* que se ejecuta justo antes de mostrar el componente:

Código 4-48. Método *componentWillMount* (client/src/containers/CreateLicensePage.jsx)

```
{...}
 //Justo antes de renderizar el componente se llamará a este método
  componentWillMount(){
    //Utilizando ajax, pedimos los tipos de sensores disponibles para la
organización que queremos crear
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión get
    xhr.open('get',         '/api/licenses/new?OrganizationId='         +
this.props.params.OrgId);
    // Configuramos el token para la autorización
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    // La respuesta se espera que sea un JSON
    xhr.responseType = 'json';
    // Añadimos el callback para cuando se reciba la respuesta de forma
correcta
    xhr.addEventListener('load', () => {
```

```
      if (xhr.status === 200) {
        // En caso de éxito
        // Cambiamos el estado, eliminando los errores y almancenamos los
sensores
        //Una vez sabemos qué sensores hay, configuramos sus valores
iniciales a 0
        xhr.response.sensors.split(';').map(sensor => {
          this.state.license.sensors[sensor.split(",")[1]] = '0';
        });
        this.setState({
          error: "",
          sensors: xhr.response.sensors
        });
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      }else {
        // En caso de fallo, mediante un toast informamos del mensaje de
error
          {xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    //Enviamos la petición
    xhr.send();
  }
{...}
```

En este método, se realiza una petición GET a la URL /api/licenses/new enviándole el identificador de la organización para la que se desea crear la licencia. Cuando se recibe la respuesta, si todo ha ido bien, se almacena en el estado la lista de sensores recibidas, y para el objeto *sensors* (perteneciente al objeto *licenses*), se inician a cero las cantidades de cada tipo de sensor utilizando la función *map*.

En cuanto a las funciones de *callback* que se ejecutan cuando cambia algún campo del formulario, serán muy similares a las vistas hasta ahora, solo que en este caso existirán dos:

- *changeSensors*: se llamará cuando se cambie el número de sensores en el formulario y pasará el valor obtenido del formulario a entero antes de almacenarlo (utilizando el método *parseInt*)
- *changeLicense*: se llamará cuando cambie algún otro campo del formulario (*duration* o *limit_bytes*).

Por último, se ha creado la funcion de *callback* que se ejecutará cuando se pulse el botón para enviar el formulario y crear la licencia. Esta función es prácticamente idéntica a las funciones que procesan los formularios de creación de una organización o un usuario, sólo que esta vez es a la ruta /api/licenses.

Cabe destacar, que el estado *sensors* del componente será una cadena con los sensores de una organización y seguirá esta forma "nombre_sensor1,id_sensor1;nombre_sensor2,id_sensor2". Por tanto, antes de enviarla al componente *CreateLicenseForm* habrá que hacer uso de la función *split* de modo que separe dicha cadena por puntos y comas (;) y creándose una tabla.

Dado a que estas últimas funciones descritas no son muy complejas, se ha decidido por no incluirlas aquí. Sin embargo, puede ver el código de este componente detalladamente en los anexos. Código A-18.

4.9 Listado de usuarios

A lo largo de este apartado, se explicará detalladamente como se ha llevado a cabo la posibilidad de listar los usuarios existentes en el sistema por parte de usuarios administradores, bien listar todos los usuarios del sistema, o bien listar los usuarios pertenecientes a una organización concreta.

4.9.1 Código de *back-end*

Dado que según los requisitos propuestos se debe poder listar tanto todos los usuarios del sistema, como sólo los que pertenecen a una organización, se han creado dos rutas diferentes para cada uno de estos requisitos y a continuación se detallarán.

4.9.1.1 Ruta para obtener todos los usuarios del sistema

Para poder llevar a cabo este requisito ha sido necesario crear una ruta en el servidor a la cual, tras realizarle una petición, devuelva la lista con todos los usuarios del sistema.

Sin embargo, devolver todos los usuarios del sistema no es muy "eficiente" ya que suponga que, por ejemplo, existen un millón de usuarios y se realiza una petición preguntando por todos los usuarios. ¡El servidor tendría que enviar un millón de usuarios en la respuesta! Algo no muy viable. Para solventar este problema se recurrió a lo que se conoce como "paginación", que no es más que solicitar los usuarios por "páginas", es decir, si, por ejemplo, se solicitase la primera página (y cada página es de 10 usuarios) se obtendrían los diez primeros usuarios, si solicitamos la segunda, los segundos diez usuarios, etc

A continuación, se detallará la ruta que realizará lo anterior al enviarle una petición GET a la URL /api/users y enviándole el número de página que se quiere obtener:

Código 4-49. Ruta para obtener los usuarios por página api/users (GET). (server/routes/api.js)

```
{...} router.get('/users', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
```

```
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.User.findAndCount({
            limit: 10,
            offset: 10*(req.query.page-1),
            order: 'name'
        }).then(function(result){
              return res.status(200).json({
                success: true,
                users: result.rows,
                number_users: result.count
              })
          })
      }
    })
  });
{...}
```

En esta ruta, cuando llega una petición se realiza lo siguiente:

- En primer lugar, se comprueba si el usuario que lo solicita existe y es administrador. En caso de no serlo se enviaría un error 404 o 401 respectivamente, del mismo modo que se ha hecho en todas las rutas anteriores.
- Si el usuario tiene permisos, se utilizará el método *findAndCount* de sequelize, pasándole de límite 10 usuarios a partir del *offset* (calculado a partir del número de página recibido en la petición). Además, se ordenará por nombre para que siempre devuelva los mismos en las mismas páginas. Este método devolverá los 10 usuarios solicitados además del número total de usuarios existentes (Se explicará para qué se usa esto en la vista de la lista de usuarios).

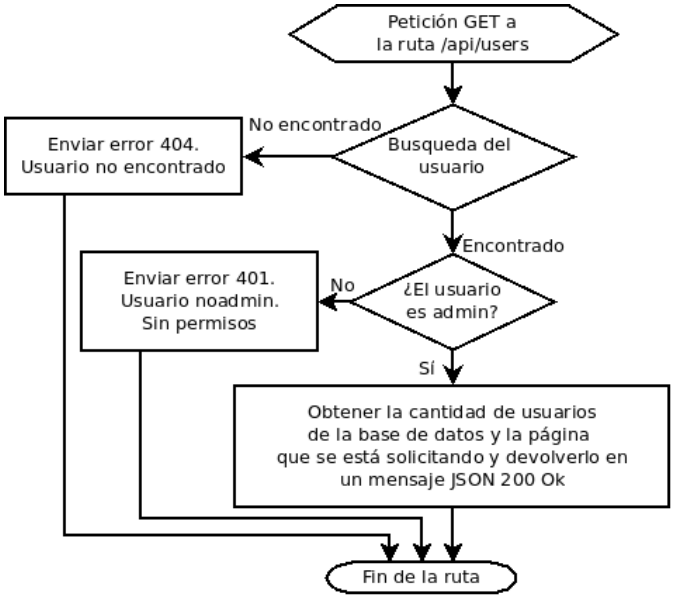


Figura 4-10. Diagrama de flujo del *middleware* de la ruta GET /api/users

4.9.1.2 Ruta para obtener los usuarios pertenecientes a una organización

Del mismo modo que se ha creado la petición para obtener todos los usuarios del sistema, en este caso se ha creado una ruta que, tras pasarle el identificador de la organización para la que queremos obtener todos los usuarios, devuelva una lista con todos los usuarios para esa organización. Al igual que antes, en este caso también se hace uso de la "paginación".

A continuación, se detallará la ruta que realizará lo anterior al enviarla una petición GET a la URL /api/organizations/:id/users y enviándole el número de página que se quiere obtener:

Código 4-50. Ruta para obtener usuarios org. api/organizations/:id/users (GET). (server/routes/api.js)

```
{...}
router.get('/organizations/:id/users', (req, res) => {
 models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
```

```
          });
      }
      else
      {
          models.User.findAndCount({
            where : {
              OrganizationId:    req.params.id=="null"    ?    null    :
req.params.id
            },
            limit: 10,
            offset: 10*(req.query.page-1),
            order: 'name'
        }).then(function(result){
              return res.status(200).json({
                success: true,
                users: result.rows,
                number_users: result.count
              })
          })
      }
    })
  });
{...}
```

Si se observa la ruta anterior, se puede ver que es muy similar a la ruta para obtener todos los usuarios descrita anteriormente y se realiza lo siguiente:

- En primer lugar, se comprueba si el usuario que lo solicita existe y es administrador. En caso de no serlo se enviaría un error 404 o 401 respectivamente, del mismo modo que se ha hecho en todas las rutas anteriores.
- Si el usuario está autorizado, se usará, al igual que en la ruta anterior, el método *findAndCount* de sequelize para obtener la página que se le esté solicitando por parámetro en la petición al método GET. Cabe destacar que, si el identificador de organización pasado es “null”, devolverá todos los usuarios que no pertenecen a ninguna organización

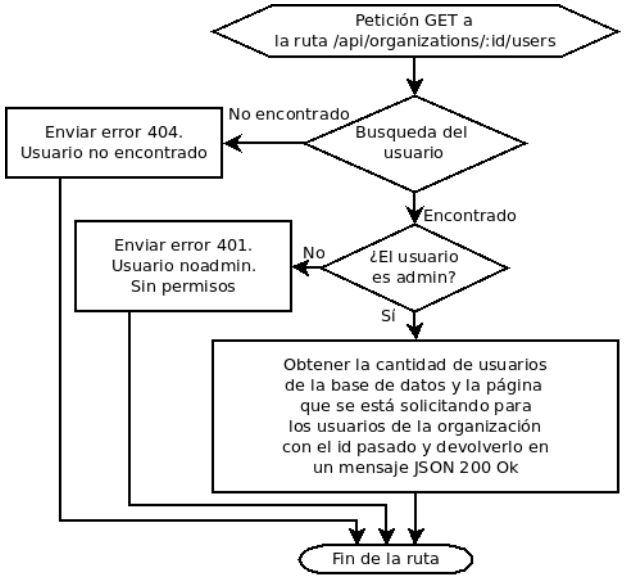


Figura 4-11. Diagrama de flujo del *middleware* de la ruta GET /api/organizations/:id/users

4.9.2 Código de *Front-end*

Una vez se han creado las rutas para listar, o bien todos los usuarios existentes en la base de datos, o bien los pertenecientes a una organización concreta, es el momento de crear la interfaz, para el lado del cliente, encargada de hacer uso de dicha ruta.

Esta interfaz será la vista que liste los usuarios, sin embargo, en dicha vista no solo se podrán listar los usuarios del sistema, sino que también se podrán eliminar y editar usuarios.

En este subapartado solo se explicarán las partes relacionadas con el listado de usuarios y más adelante, a medida que se vayan explicando los requisitos relacionados con ello, se explicará el eliminado y la edición de usuarios.

4.9.2.1 React router para el listado de los usuarios

Para acceder a esta vista, se ha creado una nueva entrada en el fichero de rutas de *react router* (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/listUsers/:id/:orgName se muestre la lista de usuarios

Código 4-51. Fragmento con la ruta para listar usuarios en el cliente. (client/src/routes.js)

```
{...}
    {
      path: '/listUsers/:id/:orgName',
      component: ListUsersPage
    },
{...}
```

Como se puede observar, en la ruta anterior se definen dos parámetros por la URL a los que se tendrá acceso mediante *react-router*, útiles en el caso de que se quieran listar los usuarios de una organización concreta:

- id: Parámetro que contendrá el identificador de la organización.
- orgName: Nombre de la organización para los que se están listando los usuarios.

Para acceder a esta URL de una forma sencilla se puede hacer mediante:

- El *Link* "Users" situado en la barra superior de navegación del componente *BasePage*, el cual ya se explicó al inicio del capítulo.
- Una serie de enlaces que existirán en la vista que muestra el listado de organizaciones. En cada entrada de la lista existirá un número que mostrará el número de usuarios para esa organización y al pulsarse aparecerán esos usuarios para esa organización. (Cuando se explique la vista del listado de organizaciones se explicará esto detalladamente).

4.9.2.2 Componente *ListUsers*

El componente *ListUsers* será el encargado de crear una tabla con cada uno de los campos de un usuario (nombre y email), además de los campos para editar el usuario o eliminar el usuario.
Para crear dicha tabla, este componente hará uso de un componente llamado *BootstrapTable*, importado de *bootstrap*, muy sencillo de usar:

Código 4-52. Tabla de usuarios creada por *ListUsers*. (client/src/components/ListUsers.jsx)

```
{...}
const ListUsers = ({
    users,
    removeUserFormat,
    editUserFormat,
    orgName }) => (
        <div>
        <div className="row">
            <div className="col-md-10">
                { //Si el nombre de la organización es "all" se mostrarán
todos los usuarios
                    orgName === "all" ?
                    <h1  className="text-left"  style={{color:"brown"}}  >
All users </h1>
                    :
                    <h1  className="text-left"  style={{color:"brown"}}  >
Users of {orgName} </h1>
                    }
                </div>
                <div className="col-md-2" >
                    <button  className="btn btn-primary text-right"><Link
style={{color:"white"}} to="/user/new">Create new user </Link></button>
                </div>
            </div>
            <div className="row">
                <br></br>
                <BootstrapTable data={users} >
                    <TableHeaderColumn       dataField="name">       Name
</TableHeaderColumn>
```

```
                    <TableHeaderColumn dataField="email" isKey> Email
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={editUserFormat}    dataAlign="center"    width="90">    Edit
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={removeUserFormat}  dataAlign="center"  width="90">  Remove
</TableHeaderColumn>
                </BootstrapTable>
            </div>
        </div>
);
{...}
```

Si se observa el código anterior, se puede ver que la forma de crear la tabla es muy sencilla. Además, este componente recibe por parámetros obligatorios:

- orgName: Cadena que será el nombre de la organización que se está mostrando. Si el nombre es "all", El nombre de la lista será "All Users", si es otro será "Users of orgName".
- users: Tabla que contendrá todos los usuarios que se quieren mostrar en esta lista.
- removeUserFormat: Función que se encargará de editar el formtato de la celda de eliminación de usuarios. (Se verá más adelante detalladamente).
- editUserFomat: Función que se encargará de editar el formtato de la celda para editar un usuario. (Se verá más adelante detalladamente).

4.9.2.3 Componente *ListUsersPage*

El componente *ListUsersPage* será un contendor que hará uso del componente *ListUsers*, definirá las funciones para editar el formato de las celdas de la tabla, los estados iniciales y gestionará la paginación.

A continuación, se explicarán las partes más importantes de este componente y que sean útiles para el listado de los usuarios, ya que las partes de edición y eliminación de usuarios se dejará para cuando se trate la implementación de dichos requisitos.

Código 4-53. Constructor de *ListUsersPage*. (client/src/containers/ListUsersPage.jsx)

```
{...}
constructor() {
{...}
    this.state={
      users: [
      ],
      activePage: 1,
      number_users: '',
      orgName: ''
```

```
    }
{...}
```

El constructor del componente no tiene mucha complejidad, sin embargo, es importante destacar como para llevar a cabo la paginación es necesario tener un estado en el que se marque la página que se está mostrando actualmente y en el constructor se marca la primera página para que se muestre esa por defecto.

Una vez se ha creado el componente, hay que hacer uso de la función *componentWillMount*, para llevar a cabo la carga de los usuarios que se desean listar:

Código 4-54. Método *componenteWillMount* (client/src/containers/ListUsersPage.jsx)

```
//Justo antes de renderizar el componente se llama a este método
  componentWillMount(){
    this.state.orgName=this.props.params.orgName;
    this.loadUsers(this.state.activePage, this.props.params.id);
  }
```

Se puede observar, que en este método no se realiza más que una llamada a la función *loadUsers* con el número de página que está activa y el identificador de la organización que se obtiene mediante *react router* de la URL. Por tanto, a continuación, se explicará qué realiza la función *loadUsers:*

Código 4-55. Función *loadUsers* de *ListUsersPage* (client/src/containers/ListUsersPage.jsx)

```
loadUsers(page, id){
    if(id=="all"){
     //Utilizando ajax, en el constructor pedimos la lista de usuarios
registrados
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/users?page=' + page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          users: xhr.response.users,
          number_users: xhr.response.number_users
        });
```

```
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // failure
        {
          xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    xhr.send();
    }
    {...}
    }else
    {
    //Utilizando ajax, en el constructor pedimos la lista de usuarios
registrados
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/organizations/' + id + '/users?page=' + page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          users: xhr.response.users,
          number_users: xhr.response.number_users,
        });

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // failure
        {
```

```
          xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    xhr.send();
    }
  }
```

Esta función, aunque parezca muy extensa no realiza nada complejo, sino que en función del identificador de la organización que se le ha pasado (id) se hará una petición para obtener todos los usuarios o solo los que pertenecen a una organización concreta:

- Si el identificador que recibe la función es "all", querrá decir que se quieren mostrar todos los usuarios y por tanto se creará una petición GET a la ruta /api/users, pasándole como parámetro el número de página que queramos mostrar. Si todo ha ido bien, almacenará los usuarios recibidos en el estado "users" y el número de usuarios totales que también devuelve esta ruta en el estado "number_users".. Si ha habido error, se tratará como se ha realizado con todas las peticiones anteriores: si es un 404 se cerrará la sesión y si es otro error se notificará con un *toast*.
- Si el identificador no es "all", querrá decir que se quieren mostrar todos los usuarios de una organización concreta y se realizará una petición GET a la url /api/organizations/:id/users, pasándole como parámetro el número de página que queramos mostrar. Si todo ha ido bien, almacenará los usuarios recibidos en el estado "users" y el número de usuarios totales que también devuelve esta ruta en el estado "number_users". Si ha habido error, se tratará como se ha realizado con todas las peticiones anteriores: si es un 404 se cerrará la sesión y si es otro error se notificará con un *toast*.

Una vez se ha descrito la función encargada de obtener los usuarios, es el momento de explicar cómo se muestra la paginación y para ello, en primer lugar, se ha importado el componente *Pagination* de la librería *react-bootstrap* que mostrará la barra de navegación entre las diferentes páginas.
Este componente se ha introducido en el método *render* del componente *ListUsersPage* del siguiente modo:

Código 4-56. Función *render* de *ListUsersPage* (client/src/containers/ListUsersPage.jsx)

```
{...}
render(){
    return (
      <div className="container">
        <div>
          <ListUsers                          orgName={this.state.orgName}
users={this.state.users}             editUserFormat={this.editUserFormat}
removeUserFormat={this.removeUserFormat}/>
        </div>
        {
        this.state.number_users > 10 ?
        <div className="text-center">
          <Pagination first last next prev ellipsis boundaryLinks
            bsSize="medium"
            //10 usuarios por página. Redondeamos para arriba
```

```
            items={Math.ceil(this.state.number_users/10)}
            maxButtons={5}
            activePage={this.state.activePage}
            onSelect={this.handleSelectPage} />
        </div>
        : null
      }
      </div>
        )
  }
}
{...}
```

Como se puede observar, en el método render del componente se incluye:

- Dentro de etiquetas *div* el componente *ListUsers* que se describió en el punto anterior.
- Dentro de otra etiqueta *div* y centrado, el componente *Pagination* que se ha importado previamente. A este componente se la han pasado las siguientes propiedades:
    - first, last: Permiten ir a la primera y la última página directamente.
    - next, prev: Aparecen flechas para pasar a la pagina anterior y a la siguiente.
    - ellipsis: Comprimen los botones de la barra de navegación de la paginación para que haya como máximo el número de páginas indicado por *maxButtons*. El resto se comprimirán con 3 puntos. (5 páginas como máximo en este caso).
    - boundaryLinks: Permite que aparezca el número de la primera página y la última cuando está activada la *ellipsis.*
    - activePage: Número de página activa. En este caso es el estado *activePage.*
    - items: Número de páginas existentes. Aquí es donde se usará el número total de usuario en el sistema, ya que el número de páginas será el numero total de usuarios dividido por 10 usuarios por página. (Redondeado al entero superior utilizando la función *ceil*)
    - onSelect: Función que se ejecutará cada vez que se haga pulse en otra página distinta. En este caso es la función *handleSelectPage* que se explicará a continuación:

Código 4-57. Función *handleSelectPage* de *ListUsersPage* (client/src/containers/ListUsersPage.jsx)

```
//Manejador para seleccionar la pagina a visualizar
 handleSelectPage(eventKey) {
   this.loadUsers(eventKey, this.props.params.id)
   this.setState({
     activePage: eventKey,
   });
 }
```

Esta función simplemente vuelve a llamar a la función *loadUsers* con el nuevo número de página y además actualiza el valor del estado *activePage* a la neuva página pulsada.

4.10 Listado de organizaciones

A lo largo de este apartado, se explicará detalladamente como se ha llevado a cabo la posibilidad de listar las organizaciones existentes en el sistema por parte de usuarios administradores.

4.10.1 Código de *back-end*

Para llevar a cabo este requisito se ha desarrollado, igual que para el caso de los usuarios, una ruta que devuelve las organizaciones del sistema y al igual que antes, también ha sido interesante utilizar la paginación, impidiendo de esta forma que todas las organizaciones se envíen en una única respuesta.

La ruta recibirá una petición GET con el número de página que se quiere obtener, y en el caso de que el usuario sea administrador, se le devolverá una lista con las 10 organizaciones desde la posición 0, 10, 20… en función del número de página que se le haya enviado en la petición, junto con el total de organizaciones en el sistema para conocer el número de páginas que existirán.
Cabe destacar que, a diferencia del caso para listar los usuarios, en este caso cada organización incluirá los usuarios de los que dispone, lo cual será usado para mostrar los usuarios de cada organización como se verá más adelante.

Esta ruta estará disponible en la URL /api/organizations y para ver el código en detalle se puede dirigir a los anexos. Código B-13.

4.10.2 Código de *Front-end*

Una vez se ha creado la ruta para listar las organizaciones, es el momento de crear la interfaz, para el lado del cliente, encargada de hacer uso de dicha ruta.

Esta interfaz será la vista que muestre la lista de organizaciones, sin embargo, en dicha vista también se podrán eliminar y editar cada organización, además de poder acceder a los usuarios y licencias de cada organización.

En este subapartado solo se explicarán las partes relacionadas con el listado de organizaciones y de los usuarios de cada organización. Más adelante, a medida que se vayan explicando los requisitos relacionados con ello, se explicará el eliminado y la edición de organizaciones.

4.10.2.1 React router para el listado de las organizaciones

Exactamente igual que se hizo para la vista del listado de los usuarios se ha creado una entrada en el fichero de rutas de tal forma que cuando se acceda a la URL http://host:port/listOrg se muestre el componente *ListOrgsPage.*
Para acceder de un modo sencillo a esta URL existirá un enlace en la barra de navegación del componente *BasePage* llamado “Organizations” que solamente será visible para usuarios administradores.

4.10.2.2 Componente *ListOrgs*

El componente *ListOrgs* será prácticamente igual que el componente *ListUsers*.

La única diferencia entre estos componentes está en los parámetros que acepta, ya que en este caso serán los siguientes parámetros obligatorios:

- organizations: Tabla que contendrá todas las organizaciones a mostrar en la tabla.

- removeOrgFormat: Función que se encargará de definir el formato para la celda de eliminación de una organización.
- listLicensesFormat: Función que se encargará de definir el formato para la celda que contenga el enlace con las licencias de cada organización.
- editOrgFormat: Función que se encargará de definir el formato para la celda de edición de una organización
- countUsersFormat: Función que se encargará de definir el formato para la celda que mostrará el número de usuarios de cada organización. Además, debe ser un enlace a la lista de usuarios de cada organización.

Puede ver este código detalladamente en el anexo A. Código A-12

4.10.2.3 Componente *ListOrgsPage*

El componente *ListOrgsPage* será un contendor que hará uso del componente *ListOrgs*, definirá las funciones para editar el formato de las celdas de la tabla, los estados iniciales y gestionará la paginación. Es muy similar al componente *ListUsersPage* y por ello solo se explicarán en detalle las partes diferentes.

Al igual que para *ListUsersPage,* en el constructor se inicializan los estados (la página activa será la primera…) y se hará uso del método *componentWillMount* para cargar las organizaciones llamando a una función llamada *loadOrgs*.

A la función *loadOrgs* se le pasará como parámetro el número de página que se está solicitando, y realizará una petición GET a la URL /api/organizations con dicho número de página. Una vez recibida la respuesta, dentro de esta función, se actualizará el estado que contiene el número total y la tabla de organizaciones.

Para realizar la paginación en este componente es exactamente igual que para el componente *ListUsersPage:* Se importa el objeto *Pagination,* se crea la función que se llamará cuando se seleccione otra página donde se llamará a *loadOrgs* con la nueva página…

Hay que destacar de este componente la definición de la función encargada de dar formato a la celda que mostrará el número de usuarios de cada organización:

Código 4-58. Función *countUsersFormat* de *ListOrgsPage* (client/src/containers/ListOrgsPage.jsx)

```
  countUsersFormat(cell,row){
    return (<Link style={{color:"blue"}} to={"/listUsers/" + row.id + "/"
+ encodeURIComponent(row.name) }>{row.Users.length}</Link>);
  }
```

Como se puede observar, en esta función se crea un enlace, mediante el componente *Link* importado de *react-router*, a la URL /listUsers seguida del identificador de la organización y el nombre (codificado de forma que se pueda enviar en la URL).
Al ir a este enlace se mostrará el componente *ListUsersPage* que se encargará de listar los usuarios para la organización cuyo identificador se le está pasando en la URL.

Además, el nombre que tendrá dicho enlace en la tabla será el número de usuarios existentes para esa organización (fila) que se obtiene contando la longitud de la tabla de usuarios (*Users*), es decir, si la organización A tiene 30 usuarios, el enlace para listar los 30 usuarios será un 30.

4.11 Listado de licencias

A lo largo de este apartado, se explicará detalladamente como se ha llevado a cabo la posibilidad de listar las licencias existentes para una organización.
Hay que tener en cuenta, que un usuario no administrador podrá listar las licencias para su organización y un usuario administrador podrá listar las licencias de cualquier organización.

4.11.1 Código de *back-end*

Para este requisito se ha creado una ruta muy similar a las anteriores para listar los usuarios o las organizaciones, solo que ahora podrán acceder a ellas usuarios normales. si desean listar las licencias de su organización, y administradores.

4.11.1.1 Ruta para obtener las licencias de una organización

La ruta que se ha creado, en primer lugar, comprobará que está accediendo a ella un usuario administrador o, si no es así, si está accediendo un usuario para listar las licencias de su propia organización.
En el caso de tener permisos para obtener la lista, devolverá las 10 licencias en función del número de página que se le haya pasado a la ruta del mismo modo que se hizo para los usuarios y las organizaciones.
En resumen, existirá una ruta en la URL /api/organizations/:id/licenses, a la cual se accederá mediante un método GET y se le enviará el parámetro *page* que indicará la página que estamos solicitando. El parámetro *id* de la URL será la organización para la que se quieren listar las licencias.
Dado que esta ruta es muy similar a las descritas anteriormente para obtener la lista de usuarios y organizaciones, no se incluirá en este apartado ni el código ni el diagrama de flujo. Si desea ver el código detalladamente, puede dirigirse a los anexos. Código B-13.

4.11.2 Código de *front-end*

Una vez se ha creado la ruta para listar las licencias de una organización, es el momento de crear la interfaz, para el lado del cliente, encargada de usarla.

Esta interfaz será la vista que muestre la lista de las licencias de una organización, sin embargo, en dicha vista también se podrá ampliar y descargar una licencia (por parte de usuarios administradores o normales para su propia organización) o activar una licencia por parte de administradores.

En este subapartado solo se explicarán las partes relacionadas con el listado de las licencias de cada organización. Más adelante, a medida que se vayan explicando los requisitos relacionados con ello, se explicará la ampliación, descarga y activación de una licencia.

4.11.2.1 React router para el listado de las licencias

Exactamente igual que se hizo para la vista del listado de los usuarios u organizaciones, se ha creado una entrada en el fichero de rutas de tal forma que cuando se acceda a la URL http://host:port/listLicenses/:id/:orgName se muestre el componente *ListLicensesPage.*

A esta URL se le pasa por parámetros mediante *react router* el identificador de la organización para la cual listar las licencias (*id*) y el nombre de la organización (*orgName*).

Para poder acceder de un modo sencillo a esta URL existirá un enlace en la barra de navegación del componente *BasePage* llamado "My licenses" que solamente será visible para usuarios normales y mostrará las licencias de su organización.
Por otro lado, para usuarios administradores, existirá en cada fila de la tabla del listado de organizaciones un enlace de tal modo que al hacer *click* en él se muestren todas las licencias para la organización que está en dicha fila. Aquí es donde se hará uso de la función *listLicensesFormat* del componente *ListOrgsPage*, que no es más que una función que crea un enlace para cada fila de la tabla del listado de organizaciones de la siguiente manera:

Código 4-59. Función *listLicensesFormat* de *ListOrgsPage* (client/src/containers/ListOrgsPage.jsx)

```
listLicensesFormat(cell, row){
    return (<div>
         <Link to={"/listLicenses/" +
        row.id + "/" + encodeURIComponent(row.name)}
        className="glyphicon glyphicon-folder-open"
        style={{color:"green"}} ></Link>
       </div>);
}
```

Como se puede observar, se crea un enlace mediante el componente *Link* de *react router* que será un enlace a la URL /listLicense + identificador de la organización (para esa fila) + el nombre de la organización codificado para poder enviarse en una URL.

4.11.2.2 Componente *ListLicenses*

El componente *ListLicenses* será prácticamente igual que el componente *ListUsers* o *ListOrgs.*

La única diferencia entre estos componentes estará en los parámetros que aceptan, ya que en este caso serán los siguientes parámetros obligatorios:

- licenses: Tabla que contendrá todas las licencias a mostrar en la tabla.
- orgId: Identificador de la organización para la que se están mostrando las licencias.
- orgName: Nombre de la organización para la que se están mostrando las licencias.
- expiresFormat: Función que se encargará de definir el formato para la celda que contenga el tiempo restante para la expiración de una licencia.
- sensorsFormat: Función que se encargará de definir el formato para la celda que contenga la lista de senosres para una licencia.
- extendFormat: Función que se encargará de definir el formato para la celda que contenga el enlace para ampliar dicha licencia.
- activateFormat: Función que se encargará de definir el formato para la celda que contendrá el estado de una licencia y que permitirá a los administradores activarla en caso de que no lo esté.

Puede ver este código detalladamente en el anexo A. Código A-11.

4.11.2.3 Componente *ListLicensesPage*

El componente *ListLicensesPage* será un contendor que hará uso del componente *ListLicenses*, definirá las funciones para editar el formato de las celdas de la tabla, los estados iniciales y

gestionará la paginación. Es muy similar a los componentes *ListUsersPage* o *ListOrgsPage* y por ello solo se explicarán en detalle las partes diferentes.

Al igual que para los otros dos componentes, en el constructor se inicializan los estados (la página activa será la primera…) y se hará uso del método *componentWillMount* para cargar las organizaciones llamando a una función llamada *loadLicenses*.

A la función *loadLicenses* se le pasará como parámetro el número de página que se está solicitando y el identificador de la organización para la cual se quieren listar las licencias.
Realizará una petición GET a la URL /api/organizations/:id/licenses con el número de página y sustituyendo *:id* por el identificador de la organización que se le ha pasado por parámetros.
Una vez recibida la respuesta, dentro de esta función, se actualizará el estado que contiene el número total y la tabla de licencias.

Para realizar la paginación en este componente será exactamente igual que para los otros dos componentes anteriores*:* Se importa el objeto *Pagination*, se crea la función que se llamará cuando se seleccione otra página donde se llamará a *loadLicenses* con la nueva página…

Hay que destacar de este componente la definición de las funciones encargada de dar formato a la celda que mostrará el tiempo restante para la expiración y la encargada de listar los sensores disponibles para esa licencia.:

Código 4-60. Función *sensorsFormat* de *ListLicensesPage* (client/src/containers/ListLicensesPage.jsx)

```
sensorsFormat(cell, row){
  const sensores = JSON.parse(JSON.parse(cell));
  let lista = [];

  for(const sensor in sensores){
        lista.push(<li key={sensor}>{sensor}: {sensores[sensor]}</li>);
  }
  return (<div>
      <ul>
      {lista}
      </ul>
  </div>);
}
```

Como se puede observar, en esta función lo único que se realiza es en primer lugar pasar el campo correspondiente a los sensores a JSON mediante la función *parse.*
Una vez se tiene el objeto *sensores,* se crea una tabla que contendrá etiquetas *li* de una lista y se devolverá para poder mostrarla en la celda.

Por otro lado, la función encargada de calcular el tiempo de expiración es la siguiente:

Código 4-61. Función *sensorsFormat* de *ListLicensesPage* (client/src/containers/ListLicensesPage.jsx)

```
expiresFormat(cell, row){
  if(row.enabled){
```

```
      const expires_period_days = Math.round((new Date(row.expires_at) -
new Date())/(24*60*60*1000)); //horas*minutos*segundos*milisegundos de un
dia

      if(expires_period_days<0)

        return           (           <div           style={{'color':"red",
'fontWeight':"bold"}}>¡¡Expires {-expires_period_days} days ago!!</div>)

      else if(expires_period_days<7)

        return  (  <div  style={{color:"red"}}>{expires_period_days}  days
remaining</div>)

      else if (expires_period_days<15)

        return ( <div style={{color:"orange"}}>{expires_period_days} days
remaining</div>)

      else

        return  (  <div  style={{color:"blue"}}>{expires_period_days}  days
remaining</div>)

    }

    else{

        if(row.duration<0){

          return (<div style={{color:"green"}}>{"Extension"}</div>)

        }

        else{

          return    (<div    style={{color:"green"}}>{"Duration:    "    +
row.duration + " months"}</div>)

        }

    }

  }
```

En esta función, se realiza lo siguiente:

- Se comprueba si la licencia está activada, y en el caso de que esté activada se calcula los días desde hoy hasta la fecha de expiración y
    - Si el número de días es negativo se imprimirá en rojo y negrita "¡¡Expires X days ago!!"
    - Si es menor que 7 se imprimirá en rojo "X days remaining"
    - Si es mayor que 7 y menor que 15 se imprimirá lo mismo en naranja
    - Si es mayor que 15 se imprimirá en verde
- Si la licencia no está activada se comprueba si la duración es negativa, en cuyo caso será una extensión (Se explicará por qué es negativo cuando se vea el apartado de extensión de una licencia) y se imprimirá "extension", si está desactivada y no es una extensión se mostrará el tiempo máximo de duración que podrá tener esa licencia una vez esté activada.

4.12 Edición de un usuario

A lo largo de este apartado, se explicará detalladamente como se ha llevado a cabo la posibilidad de que un usuario administrador pueda editar el nombre, el email, la organización y el rol de cualquier otro usuario.

4.12.1 Código de *back-end*

Para poder llevar a cabo este requisito ha sido necesaria la creación de dos rutas en el servidor: La primera de ella estará destinada a obtener los datos relativos al usuario que se quiere editar, y la segunda para enviar los nuevos datos al servidor del usuario a modificar.

4.12.1.1 Ruta para obtener datos de un usuario

Como se ha dicho antes, es necesario conocer los datos de un usuario antes de pasar a editarlo y para ello se ha creado una ruta a la que se le enviará una petición GET con el identificador del usuario a editar y devolverá sus datos.
Además de los datos del usuario, este método también devolverá la lista de las organizaciones existentes en el sistema, lo cual será útil para cambiar un usuario de organización. (Del mismo modo que se hizo para la creación de un usuario).
A este método solo podrán acceder usuarios administradores que estén autenticados mediante el método GET (al igual que se hizo para las rutas /api/users, /api/organizations…)

Código 4-62. Ruta para obtener datos de un usuario api/users/:id/edit (GET). (server/routes/api.js)

```
router.get('/users/:id/edit', (req, res) => {
{...}
{... si el usuario es administrador ...}
{
          models.Organization.findAll({
          order: 'name'
        }).then(function(list_orgs){
           models.User.findOne({
            where: {
                id: req.params.id
            }
          }).then(function(user_edit){
          return res.status(200).json({
            success: true,
            orgs: list_orgs,
            user: user_edit
          })
        })
      })
{...}
```

Como se puede observar, si la petición la ha realizado un usuario administrador se devolverá un mensaje 200 OK con la lista de las organizaciones (*orgs*) y los datos del usuario (*user*).

4.12.1.2 Ruta para editar los datos de un usuario

Por otro lado, ha sido necesaria la ruta a la cual se le enviarán los nuevos datos de un usuario por parte de un administrador y actualice dicho usuario en la base de datos. Esta ruta atendrá peticiones PUT en la URL /api/users/:id.

Código 4-63. Ruta para modificar datos de un usuario api/users/:id (PUT). (server/routes/api.js)

```
router.put('/users/:id', (req, res) => {
{... si el usuario es administrador ...}
      else
      {
          models.User.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(user_edit){
            if(!user_edit)
              return res.status(400).json({
                success: false,
                message: "User doesn't exists"
              })
            user_edit.name=req.body.name;
            user_edit.email=req.body.email;
            user_edit.role=req.body.role;
            user_edit.OrganizationId= (req.body.organization == "No" ||
req.body.organization == "" ) ? null: req.body.organization;
            user_edit.save()
            .then(function(user_save){
              return res.status(200).json({
              success: true,
              message: "User " + user_save.name + " edited correctly",
              user: user_save
              })
            }).catch(function (err) {
              return res.status(400).json({
              success: false,
              message: "Error editing user " + user_edit.name + '. Email
already exists.'
```

```
{...}
```

Como se puede observar en el código anterior se realiza lo siguiente:

- Si el usuario tiene permisos, es decir, es administrador, se busca el usuario que se quiere editar mediante el identificador que se le ha pasado en la URL. En caso de no encontrarse dicho usuario se devolverá un error 400 notificando que no existe.
- Si el usuario existe, se modificarán los datos. Cabe destacar que, si la organización a la que pertenece es "No", o el campo *organizationId* está vació, ese usuario no pertenecerá a ninguna organización y se almacenará *null* en la referencia a la organización a la que pertenece.
- Tras modificar los datos, se almacena el usuario y si hay error al guardarlo en la base de datos se enviará un mensaje 400 al cliente notificando que ese *email* ya existe en la base de datos para otra organización (error de validación)
- En el caso de almacenarlo correctamente se notificará al cliente con un mensaje 200 OK y un mensaje de éxito.

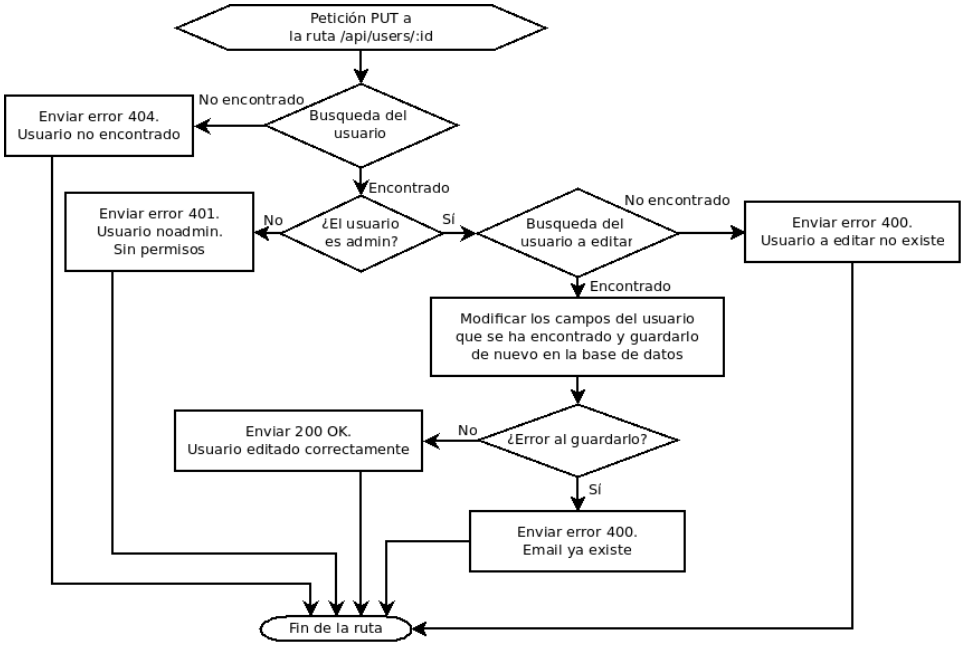


Figura 4-12. Diagrama de flujo del *middleware* de la ruta PUT /api/users/:id

4.12.2 Código de *front-end*

Una vez que se han creado las dos rutas anteriores en el lado del servidor, es el momento de crear la interfaz del cliente que haga uso de ellas.

4.12.2.1 React router para la edición de usuarios

Como se ha hecho para todos los requisitos anteriores, se ha creado una entrada en el fichero de rutas de tal forma que cuando se acceda a la URL http://host:port/users/edit/:id se muestre el componente *EditUserPage.*

En esta URL estará el identificador del usuario que se desea modificar y mediante *react router* se accederá a dicha identificador (*id*).

Para poder acceder de un modo sencillo a esta URL, en la tabla encargada de listar los usuarios, existe una columna para la edición del usuario correspondiente a cada fila.

Estas celdas se configuran mediante la función *editUserFormat* del componente *ListUsersPage*, la cual devuelve un enlace mediante el componente *Link* que contendrá en la URL el identificador del usuario correspondiente a esa fila.
Para ver detalladamente esta función puede dirigirse a los anexos. Código A-29.

4.12.2.2 Componente *EditUserForm*

Este componente es muy similar al componente *CreateUserForm*, y lo único que hace es mostrar el formulario para la edición de un usuario concreto. Para ello muestra los siguientes campos:

- Un campo de texto para el nombre del usuario
- Un campo de texto para el email del usuario
- Un campo de tipo *select* para elegir la organización a la que pertenece el usuario
- Un *checkbox* que indica si el usuario es o no administrador
- Un botón para confirmar la edición

Además, acepta los siguientes parámetros obligatorios:

- onSubmit: Función de *callback* a la que se llamará al pulsar el botón para enviar el formulario
- onChange: Función de *callback* que se ejecutará cuando algún campo cambie
- errors: Objeto que contendrá los posibles errores en el formulario
- user: Objeto que contiene los datos del usuario a editar
- organizations: Tabla de todas las organizaciones que existen en el sistema para editar a cuál pertenece el usuario

Para ver detalladamente el código de este componente puede dirigirse a los anexos. Código A-7.

4.12.2.3 Componente *EditUserPage*

El componente *EditUserPage* será un contenedor que hace uso del componente *EditUserForm* y en el se realiza lo siguiente:

- Se inician los estados en el constructor (objetos *users*, *errors* y tabla *organizations*)
- En el método *componentWillMount* se realiza una petición GET a la ruta /api/users/:id/edit para obtener los datos del usuario a editar y las organizaciones. Una vez se obtienen se almacenarán en el objeto *users* y la tabla *organizations* del estado del componente.
- Se define la función *changeUser* que será la que se le pase al componente *EditUserForm* y que se ejecute cada vez que haya un cambio en el formulario. Esta función cambiará el valor del objeto *users* y de *errors*, en el caso de haber algún error.
- Se define la función *processForm* que se le pasará a *EditUserForm* y se ejecutará cada vez que se envíe el formulario. En esta función se crea una petición PUT a la ruta /api/users/:id para así modificar el usuario. Se notificará con un *toast* si todo ha ido bien, o no, y se redirigirá al inicio.

Para ver detalladamente el código de este componente puede dirigirse a los anexos. Código A-23.

4.13 Eliminación de un usuario

A lo largo de este apartado, se explicará detalladamente como se ha llevado a cabo la posibilidad de que un usuario administrador pueda eliminar un usuario.

4.13.1 Código de *back-end*

Para poder llevar a cabo este requisito ha sido necesaria la creación de una la cual, tras recibir una petición, elimine el usuario de la base de datos cuyo identificador está en la URL, siempre y cuando el usuario que esté realizando la operación sea administrador y esté autenticado.
Esta ruta estará accesible mediante el método DELETE en la URL /api/users/:id.

Código 4-64. Ruta para eliminar un usuario api/users/:id (DELETE). (server/routes/api.js)

```
router.delete('/users/:id', (req, res) => {
{... si el usuario es administrador ...}
else
      {
          models.User.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(user_delete){
            if(!user_delete)
              return res.status(400).json({
                success: false,
                message: "User doesn't exists"
              })
            const name = user_delete.name;
            const email = user_delete.email;
            models.User.destroy({
            where: {id: req.params.id}
           }).then(function(affectedRows){
            if(affectedRows==1)
              return res.status(200).json({
              success: true,
              message: "User " + name + " (" + email + ") delete
correctly"
            })
            else
              return res.status(400).json({
              success: false,
              message: "Error removing user " + name
            })
{...}
```

Como se puede observar en el código anterior se realiza lo siguiente:

- Si el usuario tiene permisos, es decir, es administrador, se busca el usuario que se quiere eliminar

mediante el identificador que se le ha pasado en la URL. En caso de no encontrarse dicho usuario se devolverá un error 400 notificando que no existe.

- Si el usuario existe, se eliminará de la base de datos y si hay error al eliminarlo de la base de datos se enviará un mensaje 400 al cliente notificando que no se ha podido eliminar el usuario.
- En el caso de eliminarlo correctamente se notificará al cliente con un mensaje 200 OK y un mensaje de éxito.

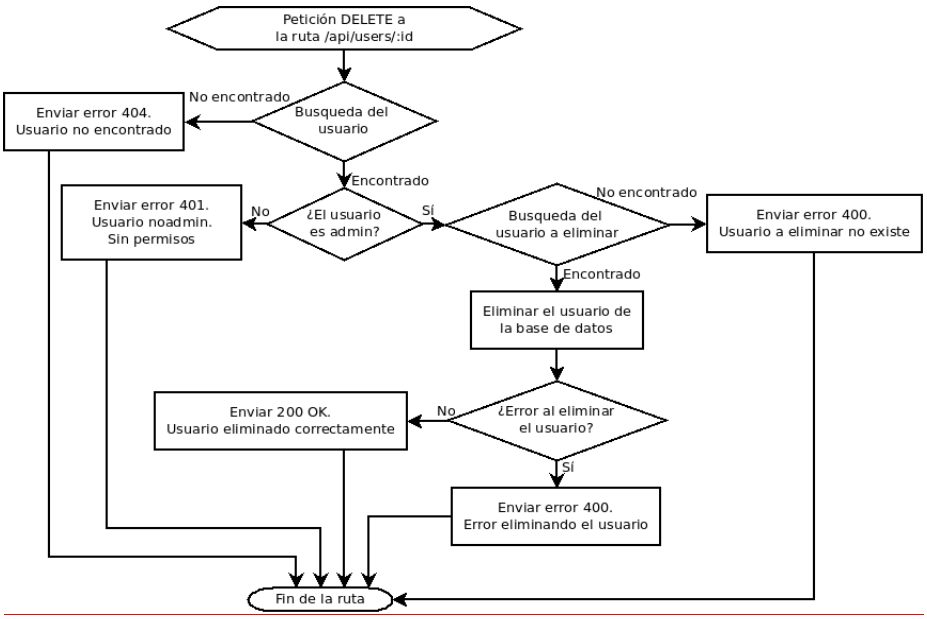


Figura 4-13. Diagrama de flujo del *middleware* de la ruta DELETE /api/users/:id

4.13.2 Código de *front-end*

Una vez que se ha creado la ruta anterior en el lado del servidor, es el momento de crear el código necesario en el lado del cliente que hace uso de dicha ruta.

Sin embargo, en este caso no ha sido necesario crear ninguna entrada para *react router* ya que la eliminación de un usuario no tiene asociada ninguna vista, sino que, se ha añadido en la tabla que se mostraba en la vista encargada de listar los usuarios (en el componente *ListUsersPage*) una columna en la que hay un botón por cada fila que enviará una petición a la ruta anterior para que elimine el usuario correspondiente a esa fila de la tabla.
Dado que la eliminación de un usuario es algo importante, antes de realizarse la petición se mostrará un cuadro de confirmación de la eliminación.
Todo esto se ha realizado en la función *removeUserFormat* definida en el componente *ListUserPage* que creará un botón por cada fila de la tabla de la siguiente forma:

Código 4-65. Función *removeUserFormat* de *ListUsersPage* (client/src/containers/ListUsersPage.jsx)

```
removeUserFormat(cell, row){
      return (<div>
            <Link style={{color:"red"}}
            onClick={() => {
              if(confirm('Are you sure to remove the user ' + row.name +
" (" + row.email + ")" )){
// create an AJAX request
                const xhr = new XMLHttpRequest();
```

```
                xhr.open('delete', '/api/users/' + row.id);
                // set the authorization HTTP header
                xhr.setRequestHeader('Authorization',            `bearer
${Auth.getToken()}`);
                xhr.responseType = 'json';
                xhr.addEventListener('load', () => {
                  if (xhr.status === 200) {
                    //Almacenamos el mensaje de respuesta
                    localStorage.setItem('successRemoveUser',
xhr.response.message);
                    //Recargamos la página para que recargue la lista de
usuarios
                    window.location.reload();
                  } else if(xhr.status === 404){
                    //No authorizated deauthenticateUser
                    this.context.router.replace('/logout');
                  } else {
                    // failure
                    {xhr.response.message                              &&
toastr.error(xhr.response.message)}
                    }
                });
                xhr.send();
              }
            }}
            className="glyphicon glyphicon-remove"></Link>
        </div>);
  }
```

Como se puede observar, esta función lo que realiza es la creación de un enlace en el que al hacer click (método *onClick*) suceda lo siguiente:

- Aparece un cuadro de confirmación. En caso de cancelarse no se realiza nada.
- Si se acepta el cuadro de confirmación, se envía una petición a la ruta DELETE definida anteriormente con el identificador del usuario de esa fila. Si se recibe error se notifica con un *toast*, y en su caso se cierra la sesión del usuario.
- Si se recibe éxito (mensaje 200), se almacena el mensaje de éxito y se recarga la página.
- Tras la recarga, debido a esta línea escrita en el constructor del componente *ListUserPage*: `localStorage.getItem('successRemoveUser') && toastr.success(localStorage.getItem('successRemoveUser')) && localStorage.removeItem('successRemoveUser')` se mostrará con un *toast* el mensaje de éxito y se eliminará el mensaje.

4.14 Edición de una organización

A lo largo de este apartado, se explicará cómo se ha llevado a cabo la posibilidad de que un usuario administrador pueda editar el nombre, el email, y el cluster id de cualquier organización.
Sin embargo, este requisito se ha implementado de una forma muy similar al de edición de un usuario y por ello se explicará de forma más breve.

4.14.1 Código de *back-end*

Al igual que para el usuario ha sido necesaria la creación de dos rutas, una para obtener los datos de una organización y otra para modificarlos.

4.14.1.1 Ruta para obtener datos de una organización

En este caso, se ha creado una ruta en el servidor accesible mediante el método GET en la URL api/organizations/:id/edit la cual devolverá los datos relativos a la organización cuyo identificador coincide con el de la URL, siempre y cuando el usuario que realice la petición esté autenticado y sea administrador.

4.14.1.2 Ruta para editar los datos de una organización

En este caso, se ha creado una ruta en el servidor accesible mediante el método PUT en la URL api/organizations/:id la cual modificará los datos que se le envíen para la organización cuyo identificador coincide con el de la URL, siempre y cuando el usuario que realice la petición esté autenticado y sea administrador.

4.14.2 Código de *front-end*

Una vez se han creado las rutas en el lado del servidor es el momento de implementar la interfaz en el lado del cliente, sin embargo, como se ha realizado prácticamente igual que para la edición de un usuario simplemente se enunciarán los componentes que se han utilizado.
Si se desea ver el código completo puede dirigirse a los anexos. Códigos A-6 y A-22

4.14.2.1 Componente *EditOrgForm*

En este componente, al igual que para *EditUserForm*, lo que se realiza es la creación de un formulario para la edición de una organización y recibe por parámetros dónde almacenar la organización, de dónde tomar los posibles errores y la funciones de *callback* necesarias, al igual que sucedía con el resto de formularios.

4.14.2.2 Componente *EditOrgPage*

Este componente hará de contenedor para el componente anterior y por tanto será donde se definan los objetos para almacenar la organización y los errores, además de las funciones de *callback.*
Al cargarse este componente, mediante el método *componentWillMount* se enviará una petición GET a la ruta /api/organizations/:id/edit para obtener los datos de la organización y rellenar el estado correspondiente al objeto que contiene la información de la organización a editar.
Y al igual que para todas las funciones que se ejecutan al enviar un formulario, se enviará una petición PUT a la URL /api/organizations/:id con los nuevos datos de la organización para que la modifique.

4.15 Eliminación de una organización

A lo largo de este apartado, se explicará cómo se ha llevado a cabo la posibilidad de que un usuario administrador pueda eliminar cualquier organización.
Sin embargo, este requisito se ha implementado de una forma muy similar al de eliminación de un usuario y por ello se explicará de forma más breve.

4.15.1 Código de *back-end*

Al igual que para el usuario, ha sido necesaria la creación de una ruta la cual, tras la recepción de una petición con el método DELETE a la URL /api/organizations/:id, elimine la organización cuyo identificador coincide con el de la URL de la base de datos, siempre y cuando el usuario que realice la petición sea administrador y esté autenticado.
Si se desea ver el código de esta ruta, puede dirigirse a los anexos. Código B-13.

4.15.2 Código de *front-end*

Para el código del lado del cliente, al igual que sucedió con la eliminación de un usuario, no ha sido necesario crear ninguna vista, sino que la propia vista encargada de listar las organizaciones tendrá una celda por cada fila que contenga un botón para eliminar la organización de esa fila.

Esta celda se ha creado mediante la función *removeOrgFormat* del componente *ListOrgsPage* del mismo modo que se hizo en el componente *ListUsersPage* con la función *removeUserFormat.*

Si se desea ver el código puede dirigirse a los anexos. Código A-28.

4.16 Activación de una licencia

A lo largo de este apartado, se explicará cómo se ha llevado a cabo la posibilidad de que un usuario administrador pueda activar una licencia para cualquier organización.

4.16.1 Código de *back-end*

Para llevar a cabo la implementación de este requisito se ha creado en el lado del servidor una ruta encargada de cambiar el estado de una licencia a activo.
Esta ruta será accesible mediante el método PUT en la URL /api/licenses/activate/:id y realizará lo siguiente:

Código 4-66. Ruta para activar una licencia api/licenses/activate/:id (PUT). (server/routes/api.js)

```
router.put('/licenses/activate/:id', (req, res) => {
{... si el usuario es administrador ...}
else
      {
          models.License.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(license_activate){
```

```
            if(!license_activate)
              return res.status(400).json({
                success: false,
                message: "License doesn't exists"
              })
            //Cambiamos el tiempo de expiración y la marcamos como
activada
            //Si la duración es negativa significa que es una extensión y
por tanto el tiempo de expiración ya está configurado
            if(license_activate.duration>0){
              const now = new Date();
              const expires_time = now.setMonth(now.getMonth() +
license_activate.duration);
              license_activate.expires_at=expires_time;
            }
            license_activate.enabled=true;
            license_activate.save()
            .then(function(license_saved){
              models.Organization.findOne({
                where:{
                  id: license_saved.OrganizationId
                }
              })
              .then(function(org){
                const mailOptions = {
                  to: org.email,
                  from: 'davsensan@gmail.com',
                  subject: "Your license has been activated",
                  text: 'Hello,\n\n' +
                    'Your license ' + license_saved.license_uuid + " has
been activated until " + license_saved.expires_at + ".\n You can use this
license since right now.\n Thank you!"
                };
                smtpTransport.sendMail(mailOptions,function(err) {
                  res.status(200).json({
                    success: true,
                    message: "License " + license_saved.id + " activated
correctly",
                    license: license_saved
                  })
```

```
                });
              })
            }).catch(function (err) {
              return res.status(400).json({
              success: false,
              message: "Error to activate license " + license_activate.id
{...}
```

En la ruta anterior se realiza lo siguiente:

- En primer lugar, se comprueba si el usuario existe y es administrado. En caso de no cumplirse se enviará un error 400 o 404, respectivamente, como se ha realizado en todas las rutas anteriores.
- Se comprueba si existe la licencia cuyo identificador se obtiene de la URL. En caso de no existir se envía un error 400.
- Si la licencia existe se comprueba si la duración es positiva, en cuyo caso se crearía la fecha de expiración sumándole esa duración a la fecha actual y se añadiría a la base de datos. Si la duración es positiva, significará que es una extensión de licencia y por tanto la fecha de expiración ya está configurada (se verá detalladamente por qué cuando se explique la ampliación de una licencia)
- Se cambia el campo *enabled* de la licencia y se almacena en la base de datos.
- Si hay error al almacenar la licencia, se envía un mensaje 400 con un mensaje de error al cliente.
- Si todo ha ido bien se envía un email al *email* de la organización para la que se ha creado la licencia notificando de la activación de la licencia hasta la fecha de expiración y se envía un mensaje 200 Ok con un mensaje de éxito al cliente.

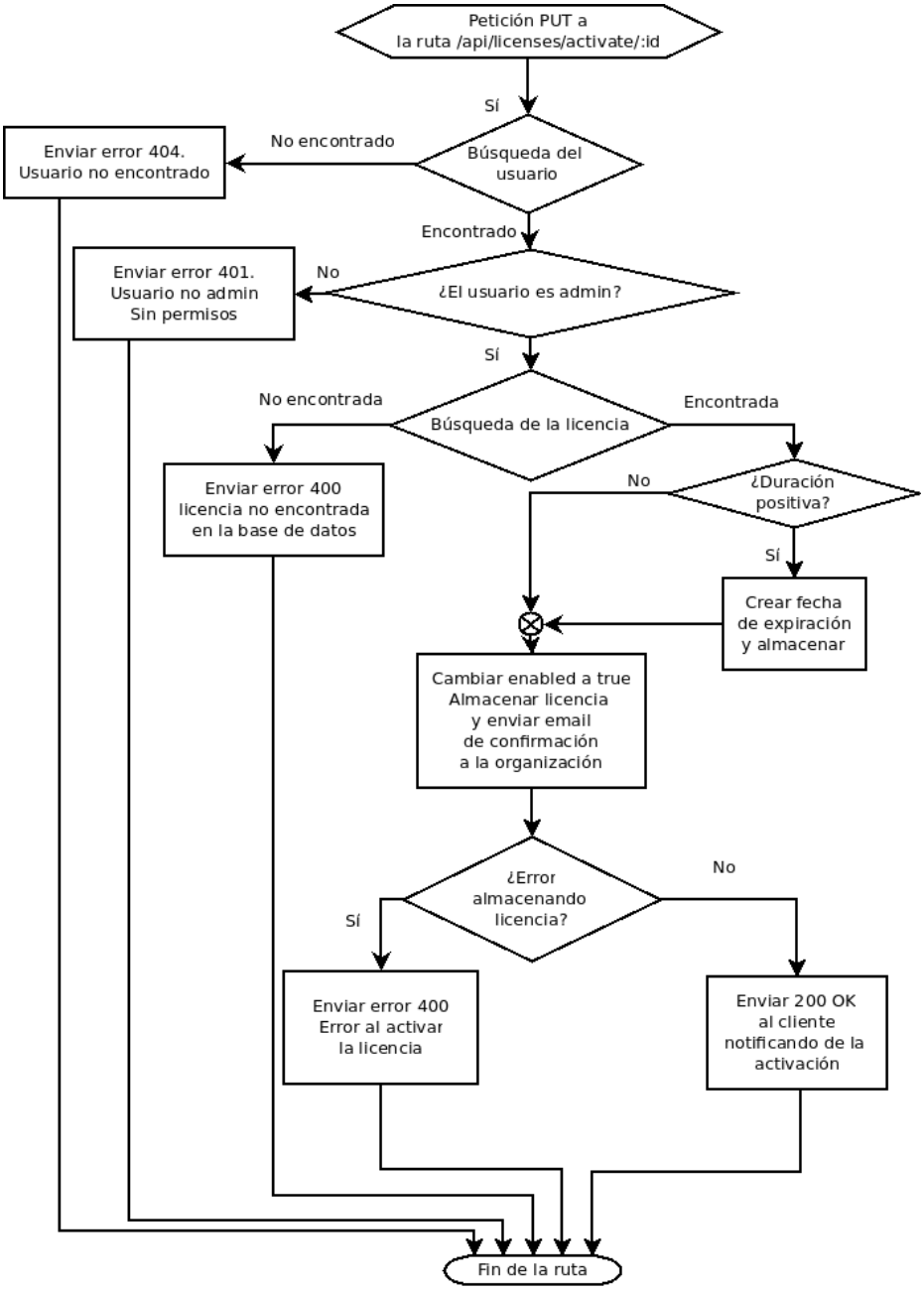


Figura 4-14. Diagrama de flujo del *middleware* de la ruta PUT /api/licenses/activate/:id

4.16.2 Código de *front-end*

Una vez que se ha creado la ruta anterior en el lado del servidor, es el momento de crear el código necesario en el lado del cliente que hace uso de dicha ruta.

Sin embargo, en este caso no ha sido necesario crear ninguna entrada para *react router* ya que la activación de una licencia no tiene asociada ninguna vista, sino que, se ha añadido en la tabla que se mostraba en la vista encargada de listar las licencias (en el componente *ListLicensesPage*) una columna en la que hay un botón por cada fila (solo se mostrará a los usuarios administradores) que enviará una petición a la ruta anterior para que active la licencia correspondiente a esa fila de la tabla.

Todo esto se ha realizado en la función *activateFormat* definida en el componente *ListLicensesPage* y creará lo siguiente por cada celda:

Código 4-67. Función *activateFormat* de *ListLicensesPage* (client/src/containers/ListLicensesPage.jsx)

```
activateFormat(cell, row){
    if(!row.enabled && Auth.isAdmin()){
      return (
        <div style={{color:"green"}}>
          Inactivated <br></br>
          <Link onClick={() => {
              const xhr = new XMLHttpRequest();
              xhr.open('PUT', "/api/licenses/activate/" + row.id);
              //Configuramos el token que identifica al usuario que está
realizando la petición
              xhr.setRequestHeader('Authorization',               `bearer
${Auth.getToken()}`);
              xhr.overrideMimeType('text/plain; charset=x-user-defined');
              xhr.addEventListener('load', () => {
                if (xhr.status === 200) {
                  // Si se ha recibido un 200 ok notificamos con un toast
                  toastr.success("License  "  +  row.id  +  "  activated
correctly")
                  //Redirigimos al inicio
                  this.context.router.replace("/");
                }else{
                  // En caso de fallo mostramos el mensaje de error
recibido del servidor
                  {xhr.response.message                                  &&
toastr.error(xhr.response.message)}
                }
              });
              xhr.send();
            }}
          className="glyphicon glyphicon-check"
          style={{color:"green"}}>
          </Link>
        </div>
        );
    }
    else{
      if(row.enabled)
        return (<div style={{color:"green"}} >Activated</div>)
```

```
      else
        return (<div style={{color:"red"}} >Inactivated</div>)
    }
 }
```

En la función anterior, se realiza lo siguiente:

- Si el usuario que muestra la tabla es administrador y la licencia está sin activar, se mostrará el texto "Inactivated" seguido de un enlace en el que al hacer *click* realizará una petición a la ruta anterior para activar la licencia correspondiente a esa fila. Notificará la respuesta con un *toast* y si todo ha ido bien se redirigirá al inicio.
- Si el usuario no es administrador y la licencia está sin activar se mostrará el texto "Inactivated" en color rojo.
- Si la licencia está activada se mostrará el texto "Activated" en color verde.

4.17 Extensión de una licencia

A lo largo de este apartado, se explicará cómo se ha llevado a cabo la posibilidad de que un usuario pueda extender una licencia para su propia organización o un administrador pueda extender licencias para cualquier organización.

La solución adoptada para extender una licencia consiste en crear una copia de la licencia que se está ampliando, pero con una fecha de expiración distinta. Aquí hay que recordar lo que se dijo en el requisito, cuando una licencia se amplía por X meses, la fecha de expiración será SIEMPRE la fecha de expiración de la licencia que se está ampliando más los meses por los que se amplíe, indiferentemente de cuándo se active.

4.17.1 Código de *back-end*

Para llevar a cabo la implementación de este requisito solo ha sido necesaria la creación de una ruta extra en el servidor: la ruta encargada de la obtención de los datos de una licencia, ya que al extender una licencia se creará una entrada nueva en la base de datos y para eso se hará uso de la ruta que se describió para la creación de una nueva licencia.

Esta ruta será accesible mediante el método GET en la URL /api/licenses/extend/ y recibirá el parámetro de la licencia de la que se desean obtener los datos:

Código 4-68. Ruta para extender una licencia api/licenses/extend (GET) (server/routes/api.js)

```
router.get('/licenses/extend', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      models.License.findOne({
        where: { id: req.query.LicenseId}
      }).then(function(license) {
```

```
        if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
        }
        //Si la licencia que se quiere extender no es para la organización a la que pertenecemos o no somos admin... no podemos
        else if(user.OrganizationId != license.OrganizationId && user.role != "admin")
        {
          return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
        }
        else if(!license.enabled){
          return res.status(401).json({
              success: false,
              message: "This license is pending of activation",
            });
        }
        else
        {
          return res.status(200).json({
            success: true,
            license: license
            });
        }
      }, function(err){
        return res.status(400).json({
          success: false,
          message: 'Error, this license not exists'
        });
      });
    })
  });
```

Como se puede observar en esta ruta se realiza lo siguiente:

- En primer lugar, se comprueba si el usuario tiene permisos para acceder al método (es administrado o

pertenece a la organización para la que se quiere extender la licencia). En caso de no tenerlo se devuelve un error 404 o 401, como en todas las rutas anteriores

- Se busca la licencia cuyo identificador está en la URL. Si la licencia no existe, se envía un error 400 notificando que esa licencia ya no existe.
- Si la licencia existe, pero no está activada se devuelve un error 401 notificando que esa licencia no está activada.
- Si todo lo anterior ha ido bien, se devuelve dicha licencia.

4.17.2 Código de front-*end*

Una vez se ha creado la ruta para obtener los datos de una licencia en el lado del servidor, es el momento de crear la interfaz, para el lado del cliente, encargada de hacer uso de dicha ruta.

4.17.2.1 React router para la creación de una licencia

Para ello, se ha creado una nueva entrada en el fichero de rutas de *react router* (client/src/routes.js) de tal modo que cuando se acceda a la ruta http://host:port/license/extendLicense/:LicenseId se muestre el componente *ExtendLicensePage*.

Para acceder a esta URL de una forma sencilla, se ha seguido una metodología muy similar a la utilizada para la edición de un usuario u organización, es decir, se ha creado una columna nueva en la tabla de la vista que lista las licencias, correspondiente a la extensión una licencia.
Estas celdas se rellenarán con el formato definido por la función *extendFormat* del componente *ListLicensesPage* en la cual se realiza lo siguiente:

Código 4-69. Función *extendFormate* de *ListLicensesPage* (client/src/containers/ListLicensesPage.jsx)

```
extendFormat(cell, row){
  if(row.enabled){
    return (
      <Link to={"/extendLicense/" + cell }
      className="glyphicon glyphicon-plus"
      style={{color:"green"}}>
      </Link>
      );
  }
  else{
    return (<div style={{color:"green"}} >Pending</div>)
  }
}
```

En esta función se comprueba si la licencia está activada (ya que en el caso de no estar activa se mostrará “Pending” y no se podrá ampliarla) y en el caso de estarlo, se mostrará un enlace a la ruta de *react router* anterior con el identificador de la licencia que será el que corresponda a esa celda.

4.17.2.2 Componente *ExtendLicenseForm*

En este componente lo que se realiza es la creación de un formulario para la extensión de una organización y recibirá por parámetros dónde almacenar la licencia ampliada y las funciones de *callback* necesarias, al igual que sucedía con el resto de formularios.
Este formulario sólo tendrá un campo de tipo *select* en el que seleccionar una ampliación de 1, 3, 6 o 12 meses.

4.17.2.3 Componente *ExtendLicensePage*

Este componente hará de contenedor para el componente anterior y por tanto será donde se definan el objeto para almacenar la licencia ampliada y las funciones de *callback*
.
Al cargarse este componente, mediante el método *componentWillMount* se enviará una petición GET a la ruta /api/licenses/extend para obtener los datos de la licencia a ampliar y rellenar el estado correspondiente al objeto que contiene la información de dicha licencia.

Por último, al igual que para todas las funciones que se ejecutan al enviar un formulario, se enviará una petición POST a la URL /api/licenses con todos los datos de la licencia a ampliar, incluido el campo *license_uuid* para crear una nueva extensión de licencia. Aquí hay destacar algunas diferencias con la creación de una licencia:

- En el caso de la extensión, el campo *License_uuid* es el mismo que el de la licencia extendida, mientras que en la creación este campo se creaba aleatoriamente al crear la nueva licencia en la base de datos
- La fecha de expiración de una licencia extendida se configura cuando se amplía, de tal modo que se le suma el número de meses de extensión a la fecha de expiración de la licencia que se está ampliando, mientras que para una licencia nueva ese campo se rellenaba sumando la duración a la fecha en la cual se activa.
- El campo de duración de una licencia extendida será siempre -1, ya que una licencia extendida, a diferencia de una nueva licencia, siempre expirará en la misma fecha indiferentemente de cuándo se active.

Todas estas “diferencias” se realizan dentro del componente *ExtendLicensePage* de tal modo que la licencia que se solicite crear ya tenga todos los campos configurados.
Puede ver el código de este componente detalladamente en los anexos. Código A-24.

4.18 Descarga de una licencia

Finalmente, pero muy importante, está el requisito de descarga de una licencia mediante el cual los clientes de RedBorder pueden descargarse sus licencias para utilizarlas.
Durante este apartado se explicará cómo se ha llevado a cabo una solución para este requisito.

4.18.1 Código de *back-end*

Para llevar a cabo este requisito se ha tenido que crear una ruta a la cual un usuario le pueda solicitar, mediante el método GET, una descarga de una licencia. Para poder descargarla, ésta debe estar en primer lugar activada y el usuario que la solicite debe, o bien pertenecer a la misma organización que la licencia, o bien ser administrador.

La ruta que se ha creado en el servidor está en la URL /api/licenses/download y es accesible mediante el método GET. Necesitará recibir un parámetro, *LicenseId,* que será el identificador de la licencia que se está solicitando descargar.

Código 4-70. Ruta para descargar una licencia api/licenses/download (GET) (server/routes/api.js)

```
router.get('/licenses/download', (req, res) => {
    models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      models.License.findOne({where: {
        id: req.query.LicenseId
      }}).then(function(license){
        if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
        }
        //Si la licencia que se quiere descargar no es para la organización a la que pertenecemos o no somos administradores... no podemos
        else if(user.OrganizationId != license.OrganizationId && user.role != "admin"){
          return res.status(401).json({
              success: false,
              message: "You don't have permissions",
            });
        }
        else if(!license.enabled){
          return res.status(401).json({
              success: false,
              message: "This license is pending of activation",
            });
        }
        else
        {
            models.Organization.findOne({
              where: {
                id: license.OrganizationId
              }
            }).then(function(organization){
```

```
              const license_json = {
                info: {
                  uuid: license.id,
                  cluster_uuid: organization.cluster_id,
                  expire_at: license.expires_at.getTime()/1000,
                  limit_bytes: license.limit_bytes,
                  sensors: JSON.parse(JSON.parse(license.sensors)),
                  createdAt: license.createdAt.toISOString()
                }
              }

              license_json.encoded_info                                    =
safeURLBase64Encode(JSON.stringify(license_json.info));
              //Firmado de la licencia
              license_json.signature                                       =
safeURLBase64Encode(key.sign(license_json.encoded_info));
              res.writeHead(200,   {'Content-Type':   'application/force-
download','Content-disposition':'attachment;         filename='         +
req.query.LicenseId + '.lic'});
              return
res.end(safeURLBase64Encode(JSON.stringify(license_json)));
            }, function(err){
              return res.status(404).json({
                success: false,
                message: "License not found!"
              })
          })
        }
      });
    });
  });
```

En esta ruta se realiza lo siguiente:

- En primer lugar, se comprueba si el usuario que solicita la descarga existe y es administrador o pertenece a la misma organización que la licencia que se quiere descargar. En el caso de no existir el usuario se devolverá un error 404, y si no tiene permisos (no es administrador o no pertenece a la misma organización) se devolverá un error 401 como se ha hecho en todas las rutas.
- Si el usuario tiene permisos, se busca la licencia y si no existe se devuelve un erro 404 con el mensaje "License not found!"
- Si la licencia existe, se comprueba si está activada ya que en el caso de no estarlo se devolverá un error 401, notificando que la licencia está pendiente de activación.
- Si la licencia existe y está activada se procede a la creación del objeto que se enviará al cliente:

- En primer lugar, se crea un objeto con los campos *uuid*, *cluster_uuid*, *expire_at*, *limit_bytes* y *sensors* con los datos de la licencia que se haya en la base de datos. Además, se le añade el campo *createdAt* con la fecha en la que se cree este objeto.
- Una vez creado este objeto, se le añade otro campo llamado *encoded_info* en el que se guardará el objeto anterior codificado en base 64.
- Finalmente se añade a este objeto el campo *signature* en el que irá la firma con la clave privada, del campo *encoded_info*, (obtenida, o bien de la variable de entorno PRIVATE_KEY, o bien del fichero server/private.key.json) utilizando el método *sign* de la librería *node-rsa*.

- Una vez creado este objeto, se configurará la cabecera de respuesta para notificar que irá un fichero adjunto cuyo nombre será el identificador de la licencia más la extensión .lic
- Finalmente se envía la respuesta al cliente con el objeto anterior codificado en base 64.

Hay que señalar que a la codificación en base64 se lleva a cabo haciendo uso de una pequeña función, definida en el mismo fichero api.js, que reemplaza los caracteres que darían problemas en una URL.

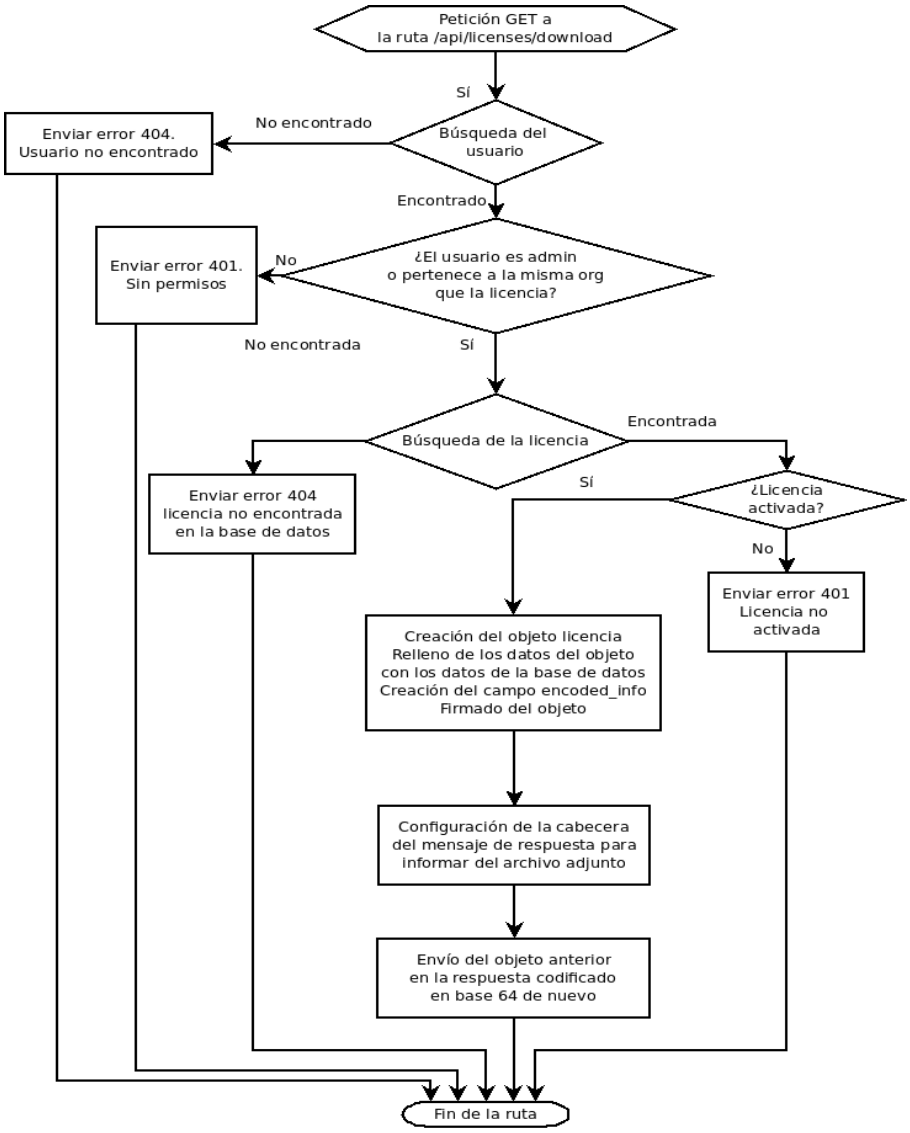


Figura 4-15. Diagrama de flujo del *middleware* de la ruta GET /api/licenses/download

4.18.2 Código de *front-end*

Una vez que se ha creado la ruta anterior para la descarga de una licencia, es el momento de crear el código en el lado del cliente que se encargue de solicitar la descarga de una licencia.
En este caso no ha sido necesario la creación de un componente, ya que para la descarga de una licencia no hay una vista asociada, sino que al igual que se ha realizado para la eliminación de usuarios y organizaciones, se ha creado una columna en la vista encargada de mostrar la lista de licencias.
En esa columna cada celda, asociada a una fila (licencia), dispondrá de un botón que al pulsar envíe la petición a la ruta anterior y comience la descarga del fichero de licencia.

Por tanto, para llevar a cabo esto se ha utilizado la función *downloadFormat*, definida en el componente *ListLicensesPage*, que se muestra a continuación:

Código 4-71. Función *downloadFormat* de *ListLicensesPage* (client/src/containers/ListLicensePage.js)

```
downloadFormat(cell, row){
    if(row.enabled){
      return (<div>
            <Link style={{color:"green"}}
            onClick={() => {
              const xhr = new XMLHttpRequest();
              xhr.open('GET',    "/api/licenses/download?LicenseId="    +
cell);
              //Configuramos el token que identifica al usuario que está
realizando la petición
              xhr.setRequestHeader('Authorization',               `bearer
${Auth.getToken()}`);
              xhr.overrideMimeType('text/plain; charset=x-user-defined');
              xhr.addEventListener('load', () => {
                if (xhr.status === 200) {
                  // Si se ha recibido un 200 ok, notificamos con un
mensaje que se está descargando la licencia
                  toastr.success("Download file...");
                  const   blob   =   new   Blob([xhr.response],   {type:
"text/plain;charset=utf-8"});
                  FileSaver.saveAs(blob, row.license_uuid + ".lic");

                } else if (xhr.status === 404){
                  //Si obtenemos un error 404 es que esa licencia no
existe, lo notificamos con un toast de error
                  toastr.error("Error. <br></br> License not found!")
                }else{
                  // En caso de fallo mostramos el mensaje de error
recibido del servidor
```

```
                    {xhr.response.message                                    &&
toastr.error(xhr.response.message)}
                  }
                });
                xhr.send();
              }}
              className="glyphicon glyphicon-download-alt"></Link>
          </div>);
    }
    else{
      return (<div style={{color:"green"}} >Pending</div>)
    }
  }
```

Como se puede observar en el fichero anterior:

- En primer lugar, se comprueba si la licencia está activa, ya que en el caso de no estarla se mostraría un mensaje con el texto "Pending".
- En el caso de estar activa, se crea un enlace y dentro de la función que se llama cuando se pulsa dicho enlace:
    - Se crea la petición a la ruta de descarga de una licencia
    - Si se recibe como respuesta un mensaje 200 OK, se guarda el fichero haciendo uso de la librería *file-saver* y se notifica con un *toast* que la licencia está siendo descargada.
    - Si se recibe un error 404, se notifica con un *toast* que la licencia no existe.
    - Si se recibe algún otro error, se notifica con un *toast* el mensaje de error recibido.

5 TESTEO DE LA APLICACIÓN

A lo largo del siguiente capítulo se explicará cómo se han llevado a cabo las pruebas a las diferentes funcionalidades de la aplicación, ya que resulta indispensable en un desarrollo software realizar pruebas para todas las funcionalidades.

En este proyecto, los test se han limitado a comprobar el funcionamiento del código de *back-end*, ya que debido a las limitaciones de tiempo se optó por testear sólo la parte del servidor.
Pero ¿Por qué se optó solo por probar la parte del servidor y no la del cliente? Pues supongan, por ejemplo, el caso en el que un cliente manipule el código de *front-end* (tarea que, en general, resulta sencilla) y consiga enviar una petición a la ruta para la eliminación de un usuario sin tener permisos. Si el servidor está testeado y funciona correctamente, impedirá dicha acción ya que el usuario no tiene los permisos necesarios.

Los test que se han realizado a la aplicación son test unitarios, encargados de probar solo partes de la aplicación por separado de tal forma que el resultado de un test no pueda interferir en ningún otro. Para dichos test, como se dijo en el apartado de "tecnologías empleadas" se ha usado *mocha*.

5.1 Estado inicial de la base de datos

Algo que resulta muy importante a la hora de realizar un test es conocer el estado en el que se encuentra la base de datos, es decir, qué tablas tiene creadas, qué tuplas existen en cada tabla…

Para partir siempre de un estado conocido en todos los test, se ha optado por eliminar todos los datos de la base de datos antes de cada test, y en los casos que haya sido necesario, cargar una serie de usuarios, organizaciones y licencias, ya conocidas, haciendo uso de la librería *sequelize-fixtures.*
Esta librería permite insertar datos, de una forma sencilla y automatizada, que hayan sido definidos en un fichero JSON (/test/fixtures/fixtures.json en este caso).

Finalmente, antes de realizar cada test se realizará lo siguiente:

Código 5-1. Función *beforeEach* que será ejecutada antes de cada test**.**

```
//Antes de cada test limpiamos la base de datos y cargamos los datos
  beforeEach(function(done){
    //Sincronizamos la base de datos. La opción force:true la limpia
    sequelize.sync({force:true}).then( () => {
      //Cargamos los datos necesarios
      sequelize_fixtures.loadFile('test/fixtures/fixtures.json', models)
            .then(() => done())
{...}
  });
```

5.2 Tests existentes

Todos los test que se han definido a lo largo del desarrollo del proyecto se hayan en el directorio *test* y además dentro de este directorio se han agrupado en 3 subdirectorios más: *models*, *relations* y *routes*.

5.2.1 Testeo de los modelos

En el directorio /test/models se hayan 3 ficheros encargado de realizar test a cada uno de los modelos existentes en la aplicación: usuarios, organizaciones y licencias. A continuación, se explicará qué *test* se han creado para cada uno de estos modelos:

Para el modelo de usuarios se han definido los siguientes *tests:*

- Comprobar que cada vez que se inicia en el modo test la tabla *users* está vacía.
- Comprobar que cuando añadimos un usuario, solo se añade uno.
- Comprobar que si la contraseña es de menos de 8 caracteres no se crea el usuario.
- Comprobar que si la contraseña es de más de 15 caracteres no se crea el usuario.
- Comprobar que si la contraseña está en blanco no se crea el usuario.
- Comprobar que si el rol no es admin o normal no se crea el usuario.
- Comprobar que si el rol está vacío no se crea el usuario.
- Comprobar que si el email tiene un formato inválido no se crea el usuario.
- Comprobar que se puede encontrar un usuario creado por su email.
- Comprobar que un usuario se crea con los parámetros indicados.
- Comprobar que al crear un usuario la contraseña se encripta correctamente y la podemos verificar.
- Comprobar que podemos cambiar la contraseña de un usuario creado previamente.
- Comprobar que si la contraseña actual es incorrecta no se nos permite cambiar la contraseña.
- Comprobar que podemos cambiar la contraseña de un usario que previamente estaba guardado en la base de datos.

Para el modelo de organizaciones se han definido los siguientes *tests:*

- Comprobar que cada vez que se inicia en el modo test la tabla *organiations* está vacía.
- Comprobar que cuando añadimos una organización, solo se añade una.
- Comprobar que si el email es invalido no se crea la organización.
- Comprobar que si el nombre está vacío no se crea la organización.
- Comprobar que si el email está vacío no se crea la organización.
- Comprobar que cuando se crea una organización se crea con los parámetros adecuados.

Para el modelo de licencias se han definido los siguientes *tests:*

- Comprobar que cada vez que se inicia en el modo test la tabla licenses está vacía.
- Comprobar que si el campo *OrganizationId* está vacío no se crea la licencia.
- Comprobar que si el campo *OrganizationId* no existe en la tabla organización no se crea la licencia.
- Comprobar que, si el campo *OrganizationId* existe, se puede crear la licencia.

5.2.2 Testeo de las relaciones entre los modelos

En el directorio test/relations se hayan los tests correspondientes a las relaciones entre los diferentes modelos de la base de datos agrupados en dos ficheros: uno para las relaciones entre usuario y organizaciones, y otro para las relaciones entre licencias y organizaciones.

Para las relaciones usuario-organización se han creado los siguientes tests:

- Comprobar que se puede crear una organización con dos usuarios.
- Comprobar que se puede crear una organización sin usuarios.
- Comprobar que se puede añadir un usuario sin organización.

Para las relaciones licencia-organización se han creado los siguientes tests:

- Comprobar que se puede crear una organización sin licencias.
- Comprobar que se puede crear una organización con varias licencias.

5.2.3 Testeo de las rutas del servidor

En el directorio test/routes existirán una serie de ficheros correspondiente a cada una de las rutas disponibles en el servidor (creación de un usuario, eliminación, listado…). En estos ficheros se hará uso de las librerías *chai* y *chai-http*, utilizadas para realizar peticiones a las rutas del servidor durante los test.
Para poder utilizarla hay, en primer lugar, que importarlas e indicarle a *chai* que use chai-http de la siguiente manera: `chai.use(chaiHttp)`. Una vez hecho esto se podrán realizar peticiones a las rutas.
A continuación, se muestra una forma genérica para enviar una petición al servidor durante los tests:

Código 5-2. Forma genérica de enviar peticiones a las rutas del servidor**.**

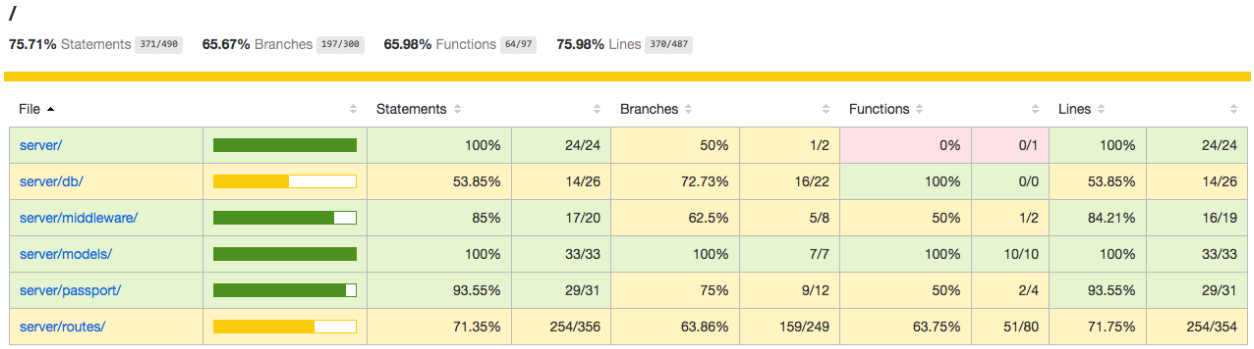
```
chai.request(fichero index del servidor)
       .post(RUTA) //Metodo POST, GET, PUT o DELETE
       .send(datos a enviar) //Envío de la petición con los datos
       .end((err, res) => {
         Función de callback que se ejecutará al recibir la respuesta del
servidor
}
```

Dado la cantidad de líneas de código existentes para la realización de cada test, el código de los mismos no se encontrará en los anexos de la memoria, pero si se desea, puede verlo en el CD-ROM adjunto con esta memoria o bien acceder a la siguiente URL de *GitHub:*
https://github.com/redBorder/licensing-management/tree/master/test

5.3 Coverage

Además de realizar pruebas y comprobar si el código funciona correctamente o no, se ha añadido la posibilidad de saber qué partes de código han sido probadas y cuáles no, algo que es fundamental ya que, si no se comprueban todas y cada una de las líneas de código, se podrán originar fallos cuando dichas situaciones pasen durante el funcionamiento de la aplicación.
Para realizar esta tarea se ha hecho uso de la herramienta conocida como *istanbul*.

Esta herramienta permitirá ejecutar los test, por ejemplo, con *mocha*, y tras eso generará un directorio llamado *coverage* en el que estará la información de las líneas de código testeadas, que partes no se han testeados, qué porcentaje de cada fichero has sido testeado… Además, existirá un fichero html que se podrá visualizar en un navegador mostrándose lo siguiente:



File	Statements		Branches		Functions		Lines	
server/	100%	24/24	50%	1/2	0%	0/1	100%	24/24
server/db/	53.85%	14/26	72.73%	16/22	100%	0/0	53.85%	14/26
server/middleware/	85%	17/20	62.5%	5/8	50%	1/2	84.21%	16/19
server/models/	100%	33/33	100%	7/7	100%	10/10	100%	33/33
server/passport/	93.55%	29/31	75%	9/12	50%	2/4	93.55%	29/31
server/routes/	71.35%	254/356	63.86%	159/249	63.75%	51/80	71.75%	254/354

Figura 5-1. Coverage de los test

Como se puede observar, en la figura anterior se muestra información sobre los diferentes directorios que han sido testeados con la información sobre qué porcentaje de ellos se han probado y cuáles no.

Además, será posible navegar por los directorios y ficheros para ver detalladamente qué líneas se han probado y qué líneas no, por ejemplo, si navegamos hasta el fichero *auth-check.js*:

Figura 5-2. Coverage del fichero *auth-test.js*

En esta imagen se puede observar que las líneas 18, 34 y 38 no han sido probadas y por tanto se debería realizar un test en el que dichas líneas se probasen, evitándose así comportamientos inesperados en un futuro.

5.4 Lanzamiento test en local

Sin embargo, ¿Cómo se crea el directorio anterior? ¿Cómo se lanzan los test?

Para realizar todo esto de una forma sencilla se ha creado una entrada en el fichero *package.json* de tal forma que cuando se ejecute (en el mismo directorio) el comando `npm run test` se lancen los test y se haga uso de *istanbul* para la creación del directorio *coverage.*

La entrada que se ha creado en el fichero *package.json* es la siguiente:

```
    "test": "export MODE_RUN='test' && istanbul cover _mocha -- --
timeout 10000 test/**/*"
```

Como se puede observar, en la línea anterior se configura, en primer lugar, la variable de entorno MODE_RUN con el valor de "test" para que el código pueda conocer que se están ejecutando test. Tras esto, se hace uso de *istanbul* y se lanzan los test que se hayan dentro del directorio *test.*

5.5 Integración con travis

Una vez se han configurado los test de forma local, es el momento de pasar a integrarlos con *travis,* una plataforma que permite ejecutar los test de una aplicación cada vez que se suba un cambio al repositorio de *GitHub* que se ha integrado con *travis.*

Para configurar *travis*, en primer lugar, hay que autorizar el acceso al repositorio de *GitHub* que se quiere integrar en la propia plataforma y una vez realizado esto, crear el fichero *travis.yml* en el directorio raíz de la aplicación.

Código 5-3. Fichero de configuración de travis**.** (.travis.yml)

```
language: node_js
node_js:
    - "stable"
services:
    - docker
after_success:
    - codecov
before_install:
    - pip install --user codecov
    - docker run -e MYSQL_DATABASE=licenses_management_test -e
MYSQL_ROOT_PASSWORD=root -d -p 3310:3306 mysql:5.7
script:
    - DB_PORT=3310 DB_PASSWORD=root npm run test
```

Este fichero no se encarga más que de dar una serie de instrucciones a *travis* para que pueda ejecutar los test correctamente, como pueden ser:

- Qué lenguaje se utilizará (*Node js* en este caso)
- Qué servicios serán necesarios (*docker* en este caso para instalar mysql 5.7 en *travis*, versión que permite tener campos JSON)
- Instalar *codecov* (utilizado para enviar datos del *coverage* a la plataforma *codecov*) y *mysql* utilizando docker

- Qué comando ejecutar para los test (En este caso se le pasará el puerto 3310 que será donde esté *mysql* en *travis*)
- Qué realizar si los test pasan correctamente (En este caso se ejecutará *codecov* para enviar los datos del *coverage* a la plataforma *codecov*)

Si se desea ver todo el historial de test de *travis* se puede acceder a la url https://travis-ci.org/redBorder/licensing-management, sin embargo, se mostrará una captura de dicha dirección donde se puede ver que los test han pasado correctamente:

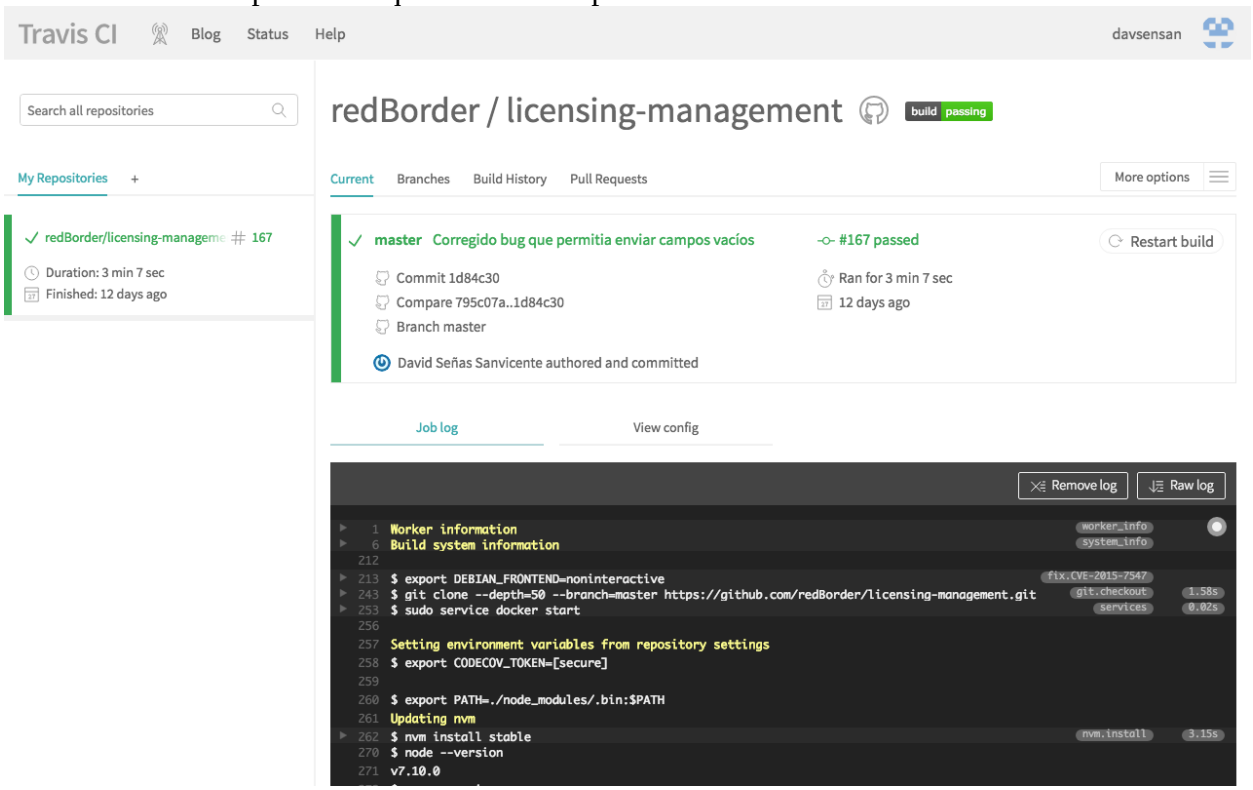


Figura 5-3. Ejemplo de testeo en *travis CI*

5.6 Integración con codecov

Finalmente se han integrado los test realizados al proyecto con una plataforma conocida como *codecov.*

El principal objetivo de esta plataforma es llevar un control del *coverage* de los test para poder verlo en dicha plataforma *online.*

Para integrarla con el proyecto, lo primero que se ha realizado es iniciar sesión en dicha plataforma con *GitHub* y añadir el repositorio *licensing-management.*
Una vez realizado esto, se ha obtenido el *token* de la sesión de *codecov* y se le ha pasado a *travis* mediante variable de entorno y como se puede observar en el fichero de configuración de *travis*, cada vez que los tests pasen de forma correcta se ejecutará el comando *codecov* el cual, gracias al *token* que se le pasó a travis, enviará los datos del *coverage* de ese último test pasado a la plataforma *codecov.*

Si se desea, se puede acceder a dicha plataforma para ver el *coverage* de último test en la URL https://codecov.io/gh/redborder/licensing-management. Sin embargo, a continuación, se mostrará una captura:

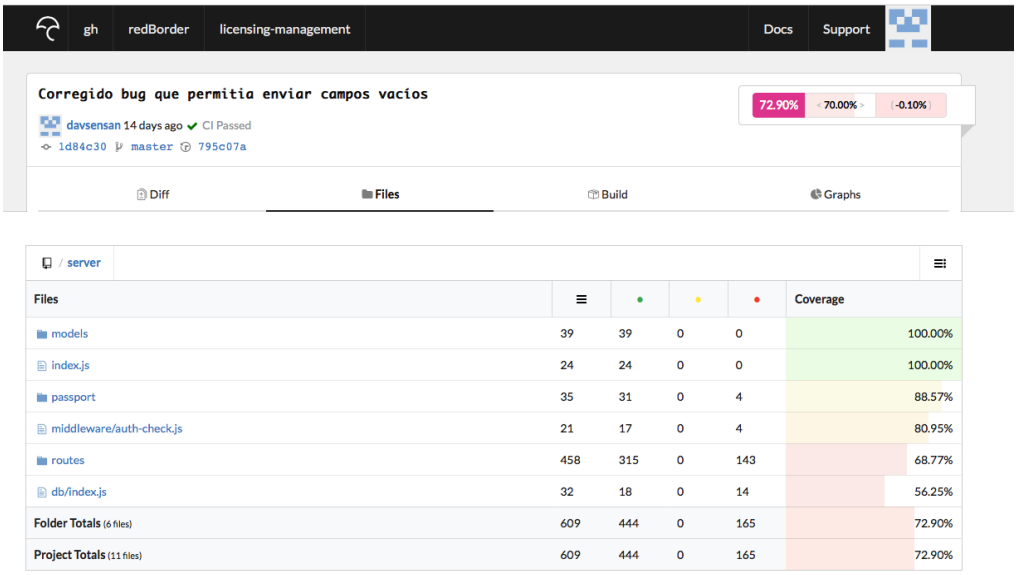


Figura 5-4. Ejemplo de *coverage* en *codecov*

Se puede observar en la captura anterior, algo muy similar a lo que se mostraba cuando se accedía al fichero html que había sido creado mediante *istanbul* en local, solo que en este caso se encuentra en la plataforma *codecov* y se puede llevar un control con todo el historial de cada uno de los cambios realizados en el proyecto y que, además, han pasado los test de forma satisfactoria.

6 DOCKERIZACIÓN

A lo largo de este capítulo se explicará en detalle cómo se ha creado, mediante *docker*, un contenedor con toda la aplicación de forma que sea muy sencillo el despligue de la misma.

El término *dockerizar* es un término que ha sido asignado a la acción de crear un contenedor de una aplicación haciendo uso de la tecnología conocida como *docker*.
Pero, ¿En qué consiste realmente crear un contenedor de una aplicación mediante *docker* y qué ventajas tiene?
Crear un contenedor de una aplicación consiste en crear una imagen de *docker* en la que se encuentre todo el código junto con las dependencias necesarias para su ejecución, de forma que un usuario cualquiera, en cualquier sistema operativo que tenga instalado *docker*, pueda desplegar una aplicación de una forma sencilla.

Además, los contenedores no solo permiten desplegar aplicaciones en local de un modo sencillo, sino que existen multitud de plataformas para el despligue de un contenedor en un servidor remoto.

6.1 Contenedor para producción

A lo largo de este proyecto se optó por realizar un contenedor mediante docker de la aplicación para poder desplegarla en producción de un modo sencillo.

6.1.1 Creación del contenedor

Para ello, lo primero que se necesitó fue la instalación de *docker* y tras ello la creación de una serie de ficheros en el directorio raíz del proyecto que le indicarán a docker como crear la imagen con el contenedor.
El fichero más importante para crear el contenedor de producción será el fichero llamado *dockerfile* que tendrá el siguiente contenido:

Código 5-4. Fichero dockerfile para producción**.** (Dockerfile)

```
FROM node:slim
#Variables de entorno para desarrolo
ENV NODE_ENV=production
ENV MODE_RUN=production

#Creación del directorio de la aplicación dentro del docker
RUN mkdir -p /app_license
WORKDIR /app_license

#Instalación de las depencias de la aplicación para producción
COPY package.json /app_license
RUN npm install --production
```

```
#Copia de los ficheros (Excepto los de dockerignore)
COPY . /app_license

#Construimos el fichero principal con webpack
RUN npm run build

#Activamos el puerto 3000
EXPOSE 3000

#Configuramos como punto de entrada el arranque del servidor
ENTRYPOINT npm run start
```

En el fichero anterior, se realiza lo siguiente:

- En primer lugar, se utiliza la imagen *node* (versión *slim*) como base para este contenedor.
- Se configuran las variables de entorno para producción.
- Se crea el directorio *app_license* dentro del docker que será donde se almacene el código de la aplicación.
- Se copia el fichero *package.json* para instalar después las dependencias de producción.
- Una vez instaladas todas las dependencias de producción, se copia todo el directorio (excepto lo que esté recogido en el fichero *.dockerignore*) al contenedor.
- Se lanza *WebPack* para construir el fichero principal del cliente.
- Se abre el puerto 3000 donde estará el servidor web escuchando.
- Finalmente, se configura el punto de entrada que lanzará el servidor mediante el comando *npm run start* cuando se arranque ese contenedor.

Una vez se ha creado el fichero *dockerfile* y el fichero *.dockerignore*, es el momento de crear la imagen y para ello simplemente hay que ejecutar en el mismo directorio del proyecto el siguiente comando:

```
docker build . -t NOMBRE_CONTENEDOR:ETIQUETA
```

6.1.2 Subida a *Docker Hub*

Docker Hub es una plataforma similar a *GitHub* pero en lugar de subirse repositorios de código se suben imágenes de contenedores de *docker*.

Durante este proyecto se optó por subir a *Docker Hub* el contenedor para producción de manera que cualquier persona pueda obtenerlo y desplegarlo de una forma sencilla sin tener que generarlo y almacenarlo en local. Además, esto también será muy útil para desplegar la aplicación en un servidor.

El repositorio donde se haya la imagen de este proyecto se encuentra en la URL https://hub.docker.com/r/davsensan/licenses-management y si se desea obtener el contenedor de

la aplicación del proyecto, simplemente hay que ejecutar, desde una máquina con *docker* instalado, el comando:

```
docker pull davsensan/licenses-management
```

6.1.3 Despliegue en local para producción

Una vez se ha creado la imagen, (o se ha obtenido de *Docker Hub*), si se desea se puede desplegar en local.
Para ello es imprescindible tener una base de datos levantada (*mysql* por ejemplo) y ejecutar el siguiente comando:

```
docker run –e VAR_ENT=VALOR –E VAR_ENT2=VALOR… -p PUERTO_ESCUCHA:3000
NOMBRE_IMAGEN
```

El comando anterior necesita, si se desean cambiar, las variables de entornos que han sido definidas a lo largo del proyecto y son las siguientes:

- DB_SERVER: Gestor de la base de datos a utilizar. Por defecto "mysql"
- DB_NAME: Nombre de la base de datos a utilizar. Por defecto "licenses_management"
- DB_HOST: Host donde escucha la base de datos. Por defecto "127.0.0.1"
- DB_PORT: Puerto en el que escucha la base de datos. Por defecto "3306"
- DB_USER: Usuario que tendrá acceso a la base de datos. Por defecto "root"
- DB_PASSWORD: Contraseña del usuario que tendrá acceso a la base de datos. Por defecto ""
- DB_LOG: Variable de tipo boolean que indica si queremos o no mensajes de logs por cada gestión con la base de datos. Por defecto "false"
- EMAIL_SERVER: Servidor de correo electrónico desde el que se enviarán los emails. Por defecto "SendPulse"
- EMAIL_USER: Correo electrónico desde el que enviar los correos electrónicos. Por defecto "EMAIL"
- EMAIL_PASSWORD: Contraseña del correo electrónico desde el que se enviarán los emails. Por defecto "PASSWORD"
- PRIVATE_KEY: Clave privada utilizada para el cifrado de las licencias. Por defecto una de prueba.
- JWT_SECRET: Clave para el cifrado del token. Por defecto "Secret key to JWT"

Además, el PUERTO_ESCUCHA será el puerto donde el servidor atienda las peticiones HTTP y NOMBRE_IMAGEN será el nombre de la imagen que se quiere desplegar (si se ha obtendio de *Docker Hub* será *davsensan/licenses-management*).

6.2 Contenedor para desarrollo.

Además de utilizar *docker* para la creación de la imagen para producción, también se ha decidido utilizar esta tecnología para desarrollo, de manera que cualquier usuario, en cualquier sistema operativo puede contribuir al desarrollo de esta aplicación simplemente ejecutando un comando.

El objetivo de esto es que, tras la ejecución de dicho comando, el desarrollador tras hacer cambios en el código pueda ver los cambios directamente en el navegador sin necesidad de ejecutar nada más que ese comando.

6.2.1 *Docker-compose* para desarrollo

Docker compose es una herramienta que permite levantar aplicaciones que necesiten usar más de un contenedor y dado que para esta aplicación es necesario tener una base de datos levantada, se ha optado por hacer uso de esta herramienta, para levantar en un contenedor la base de datos (*mysql*, por ejemplo) y en otro contenedor la propia aplicación que haga uso de la base de datos.

6.2.2 Contenedor para desarrollo

Dado que el contenedor que fue creado anteriormente fue para producción, hay que crear otro contenedor diferente que sea para desarrollo en este caso. Para realizar esto se ha creado un *dockerfile* diferente que en este caso se llamadá *dev.dockerfile*.

Código 5-5. Fichero dockerfile para desarrollo**.** (dev.dockerfile)

```
FROM node:slim
#Variables de entorno para desarrolo
ENV NODE_ENV=development
ENV MODE_RUN=development

#Creación del directorio de la aplicación dentro del docker
RUN mkdir -p /app_license
WORKDIR /app_license

#Activamos el puerto 3000
EXPOSE 3000

#Ejecutamos el comando npm start en modo desarrollo
ENTRYPOINT npm run start:dev
```

En este ficehro se realiza lo siguiente:

- En primer lugar, al igual que para producción, se utiliza la imagen *node* (versión *slim*) como base.
- Se configuran las variables para desarrollo.
- Se crea el directorio donde almacenar el código en el *docker*
- Se activa el puerto 3000 donde estará el servidor
- Y finalmente se configura el punto de entrada que se ejecutará al desplegar el contenedor. En este caso será *npm run:dev* que se encargará de lanzar tanto *webpack* como *nodemon.*

Como se puede observar, en este caso, no se copian los ficheros al *docker* como se hizo en producción, sino que en este caso se montará un volumen que haga referencia a esos ficheros (se verá más adelante cuando se configure *docker compose*).

Al igual que antes, una vez se ha creado este fichero se podría generar la imagen del contenedor, pero esto no se realizará ya que se hará uso de *docker compose.*

6.2.3 Configuración de *docker compose*

La configuración de *docker compose* se realiza mediante un fichero llamado *docker-compose.yml* el cual está situado en el directorio raíz del proyecto:

Código 5-6. Fichero de configuracíon para *docker compose***.** (docker-compose.yml)

```
version: '2'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/data/mysql
    environment:
      MYSQL_ROOT_PASSWORD: redborderlicensepassworddb
      MYSQL_DATABASE: licenses_management
    ports:
      - "3306:3306"
  web:
    volumes:
      - .:/app_license
    depends_on:
      - db
    links:
      - db:db
    build:
      context: .
      dockerfile: dev.dockerfile
    image: licenses_app_dev
    ports:
      - "3000:3000"
    environment:
      DB_PASSWORD: redborderlicensepassworddb
      DB_PORT: 3306
      DB_HOST: db
      DB_DATABASE: licenses_management
      EMAIL_SERVER: SendPulse
      EMAIL_USER: ************
      EMAIL_PASSWORD: *************
volumes:
  db_data:
```

Como se puede observar en el fichero anterior se realiza lo siguiente:

- En primer lugar, se indica la versión de *docker-compose* a utilizar (2 en este caso)
- Tras esto se le indicarán los servicios (contenedores) necesarios. En este caso se usará *mysql* (con el nombre *db*) y la propia aplicación (con nombre *web*).

Cada servicio tendrá una serie de campos y para el servicio *db* (base de datos) son los siguientes:

- image: Imagen para el contenedor que se usará. Si no la encuentra en local la descargará de *Docker Hub*
- volumes: Lugar donde almacenar los datos de la base de datos para que sean persistentes.
- environment: Variables de entornos para este contenedor (En este caso la contraseña de superusuario y el nombre de la base de datos)
- ports: Puerto que tendrá disponible ese contenedor.

Lo campos para *web* (la aplicación de gestión de licencias) son los siguientes:

- volumes: Mediante este campo se le indica que monte los datos del directorio actual en el propio *docker* de forma que cuando se modifiquen en este directorio se "modifiquen" en el *docker*. (No es que se modifiquen, sino que son los mismos datos).
- depends_on: Mediante este campo se le notifica a *docker-compose* que la base de datos debe desplegarse antes que la aplicación.
- links: Este campo permite acceder al contenedor de la base de datos mediante la referencia *db.*
- build: En este campo se le indica que debe generar la imagen a partir del fichero *dev.dockerfile* situado en el mismo directorio.
- image: Nombre con el que debe generar la imagen. Si esta imagen ya existe no la generará de nuevo.
- ports: Al igual que para la base de datos, será el puerto que tenga disponible el servidor (donde se halle el servidor web)
- enviroment: Variables de entorno que necesite la aplicación para desplegarse en modo desarrollo.

6.2.4 Despliegue en local para desarrollo

Una vez se ha configurado *docker compose*, como se ha comentado en el punto anterior, resulta muy sencillo poder desplegar la aplicación para desarrollo en cualquier máquina, ya que simplemente se necesita realizar los siguientes pasos:

- Clonar el repositorio de GitHub en la máquina que se desea desarrollar mediante el siguiente comando: `git clone https://github.com/redBorder/licensing-management.git`
- Modificar en el fichero *docker-compoe.yml* las variables de entorno con el email y la contraseña para el envío de emails.
- Ejecutar el siguiente comando y esperar a que la aplicación esté activa: `docker-compose up`

7 DESPLIEGUE DE LA APLICACIÓN

A lo largo de este capítulo se explicará en detalle cómo se ha llevado a cabo el despligue del portal en un dominio público, y así terminar el ciclo completo de desarrollo de la aplicación (Toma de requisitos, implementación, pruebas, *dockerización* y finalmente, despliegue).

En este caso se ha optado por hacer uso de una plataforma de *Red Hat* para el despliegue de contendores de *docker* conocida como *OpenShift.*

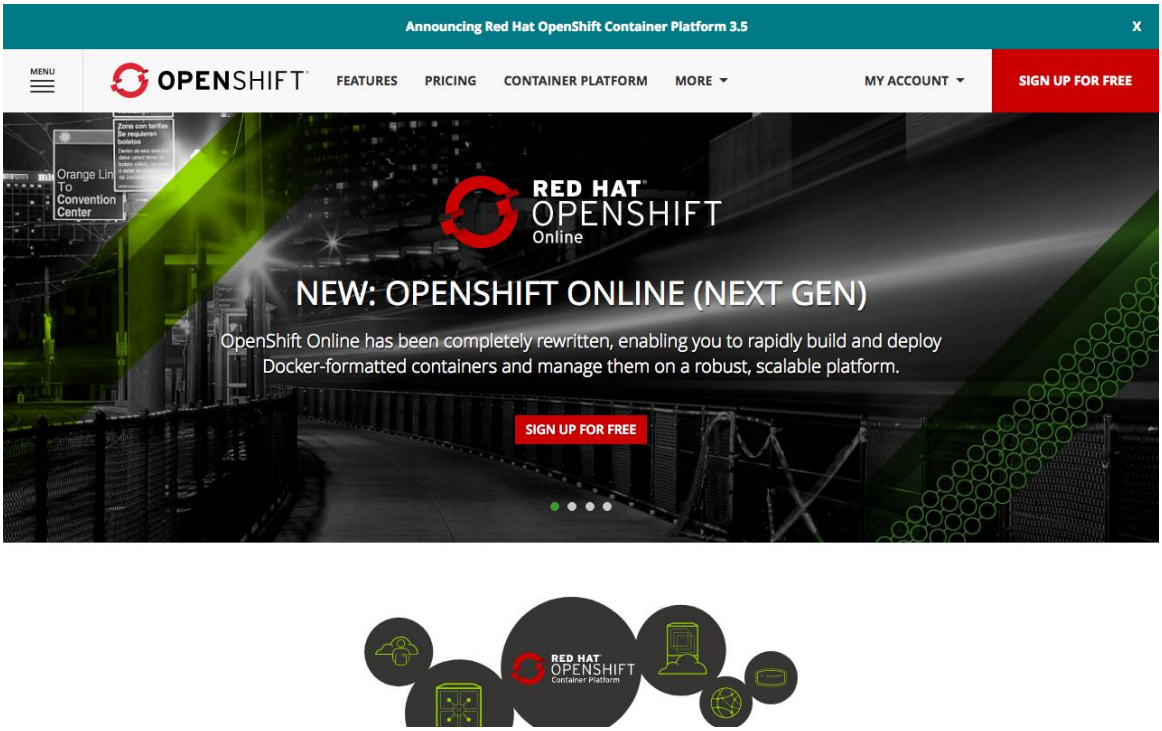


Figura 7-1. Página principal de *OpenShift*

Lo primero que se tuvo que hacer es crearse una cuenta en *Red Hat* para iniciar sesión en *OpenShift*, y tras ello crear un proyecto en dicha plataforma. Debido a que el portal web necesita una base de datos para funcionar correctamente se tuvo que desplegar un contenedor para la base de datos y otro para el portal.
En este caso se optó por desplegar *mysql 5.7*, y tras ello el portal web obteniendo el contenedor de *Docker Hub* (Previamente se había subido el contenedor con la aplicación para producción)
Cabe destacar que tanto para el despliegue de la base de datos como del portal se han requerido pasar diferentes variables de entornos mediante las cuales se han configurado dichos contenedores una vez desplegados (contraseña del usuario de la base de datos, el nombre de la base de datos, email y contraseña para el envío de correo electrónico, puerto y host del servidor de la base de datos…)

Finalmente, una vez que ambos contenedores han sido desplegados en *OpenShift* de forma correcta, es el momento de crear una ruta de acceso al servidor web desde el exterior:
Para esto, *OpenShift* ofrece la posibilidad de crear rutas asociadas a contenedores y gracias a esto se ha creado una ruta asociada al portal web en el puerto 3000 donde estará escuchando el servidor HTTP *express.*
Además, dicha ruta se ha creado de tal forma que utilice el protocolo seguro HTTPS.

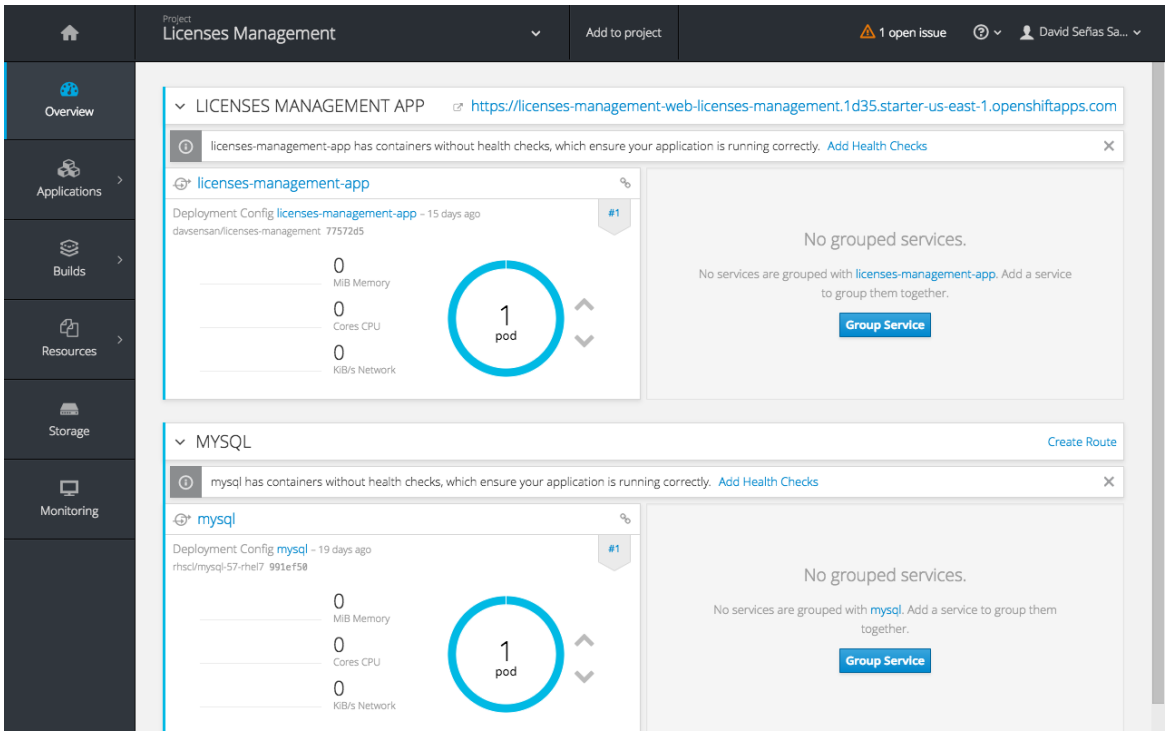


Figura 7-2. Proyecto en *OpenShift*

En la imagen anterior se puede observar que existen dos contenedores funcionando, *licenses-management-app* y *mysl*. Además, se puede ver que el primero de ellos está accesible mediante la ruta siguiente: https://licenses-management-web-licenses-management.1d35.starter-us-east-1.openshiftapps.com/

8 VISTAS DEL PORTAL

En este apartado se expondrán las diferentes vistas que se le mostrarán al usuario cuando acceda al portal, se autentique y navegue por las diferentes opciones disponibles.
Sin embargo aquí se muestran de un modo resumido por lo que si se desea ver la web terminada y desplegada detalladamente, pueden dirigirse a la URL https://licenses-management-web-licenses-management.1d35.starter-us-east-1.openshiftapps.com/ y probar la aplicación completa. (Pueden acceder con el usuario administrador "admin@redborder.com" y la contraseña "a1d2m3i4n5").

8.1 Página principal

Cuando se accede directamente al portal sin haber iniciado sesión previamente, la vista principal que se nos muestra será la siguiente:

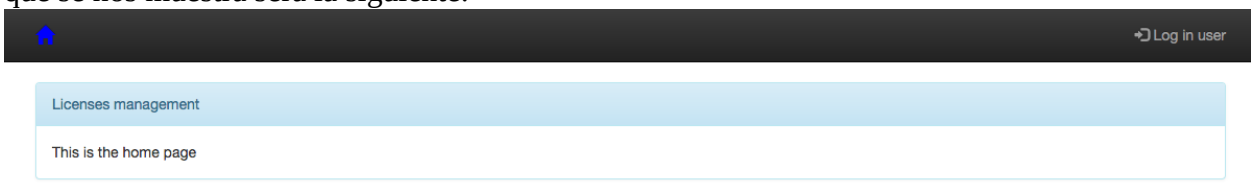


Figura 8-1. Vista principal de la web sin iniciar sesión

Como se puede observar, en esta vista aparece la barra de navegación y un mensaje anunciando que es la página principal. En la barra de navegación, en este caso, solo es posible seleccionar la opción de inicio de sesión (*Log in user*) o pulsar el icono de la casa para volver a la página principal.

8.2 Inicio de sesión

Si se pulsa la opción *Log in user* en la vista anterior, se nos llevará al siguiente formulario:

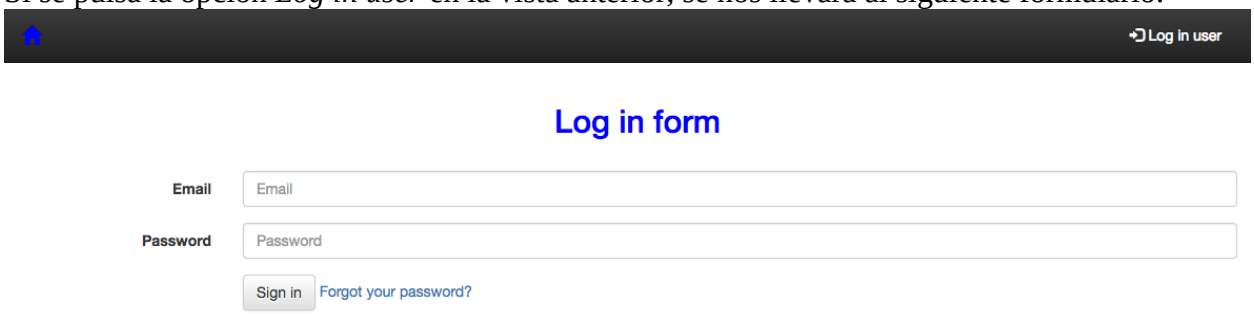


Figura 8-2. Vista para el inicio de sesión

En este caso se está mostrando el componente *LoginPage* para poder iniciar sesión. Además, al lado del botón para enviar el formulario, existe un enlace llamado "*Forgot your password*" en el que al hacer *click* muestre la vista con el formulario para el recordatorio de contraseña. (Componente *ForgotPage*).

8.3 Página principal de un usuario administrador

Si se inicia sesión con un usuario administrador (por ejemplo, con admin@redborder.com) se mostrará la siguiente vista:

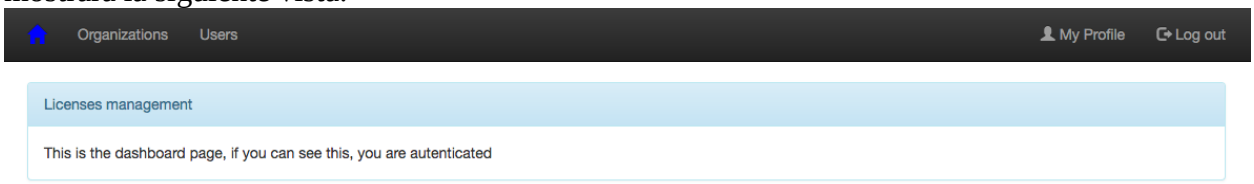


Figura 8-3. Vista principal de la web para un usuario administrador.

Como se puede observar, en este caso ha cambiado la barra superior de navegación y ahora se ofrecen las siguientes opciones diferentes:

- *Organizations:* Mostrar la vista asociada al listado de organizaciones (componente *ListOrgsPage*).
- *Users:* Mostrar la vista asociada al listado de los usuarios (componente *ListUsersPage*).
- *MyProfile:* Mostrar la vista asociada con el cambio de mi perfil (componente *ProfilePage*).
- *Log out:* Cierra la sesión del usuario y redirige a la vista principal.

Además de todas las opciones que se vayan mostrando en la barra de navegación, siempre existirá el icono con la casa para ir a la página principal.

Cabe destacar, que la barra de navegación se ha realizado de forma que, si la ventana del navegado donde se muestra la página web se hace más pequeñas, las opciones anteriores se comprimen en un desplegable de la siguiente manera. En la imagen de la izquierda se muestra antes de pulsar en las 3 barras y en la derecha después de pulsarlas:

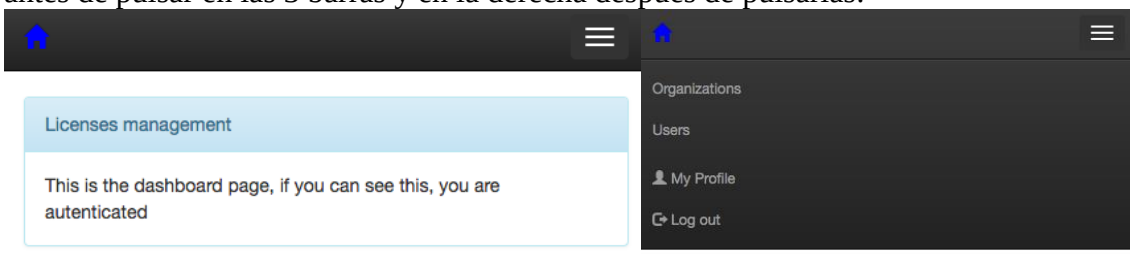


Figura 8-4. Vista principal comprimida de la web para un usuario administrador.

8.4 Página principal de un usuario normal

Si se inicia sesión con un usuario normal se mostrará la siguiente vista:

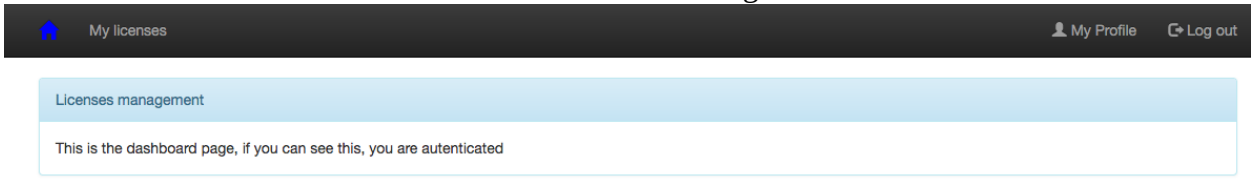


Figura 8-5. Vista principal de la web para un usuario normal.

Como puede verse, en este caso la barra superior de navegación no ofrece las opciones *Organizations* y *Users*, sino que ahora se ofrecen las siguientes:

- *My licenses:* Mostrar la vista asociada con el listado de las licencias de mi organización (componente *ListLicensesPage*). Aquí hay que comentar que si el usuario normal no pertenece a ninguna

organización esta opción no se le mostrará.

- *MyProfile:* Mostrar la vista asociada con el cambio de mi perfil (componente *ProfilePage*).
- *Log out:* Cierra la sesión del usuario y redirige a la vista principal.

8.5 Gestión de mi perfil

Una vez un usuario, administrador, o no, se ha autenticado en la web, podrá gestionar su perfil pinchando en la opción *My Profile* y se mostrará el siguiente formulario:

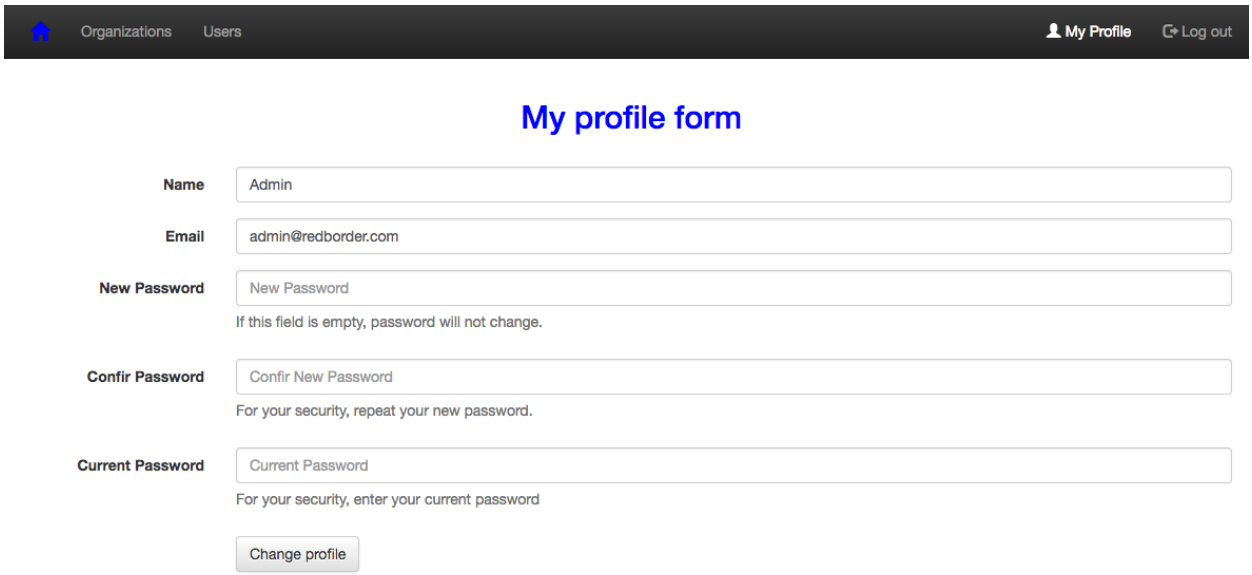


Figura 8-6. Vista principal para gestionar mi perfil.

En este caso, aparecen un formulario con los campos necesarios para cambiar los datos del perfil de un usuario. Además, se puede ver que las entradas correspondientes al nombre y al email aparecen rellenas con el valor que actualmente tenga dicho usuario configuradas.

8.6 Listado de los usuarios

Si un usuario administrador accede a la vista para listar los usuarios pulsando la opción *Users* de la barra de navegación se mostrará algo similar a lo siguiente:

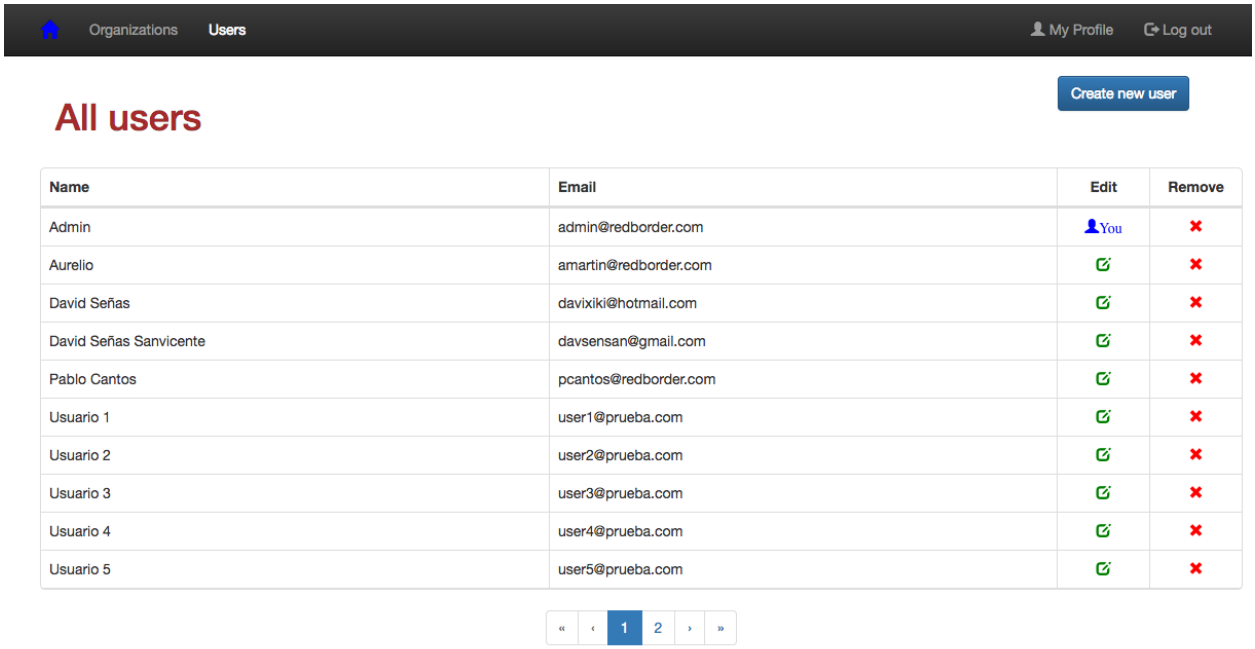


Figura 8-7. Vista con la lista de usuarios

En esta vista se muestran los 10 primeros usuarios del sistema, ya que se hace uso de la paginación y solo se está mostrando la primera página.
Cabe destacar que si existen menos de 10 usuarios en el sistema no se mostraría la barra de navegación entre páginas de la parte inferior.

8.6.1 Creación de un usuario

Además, dentro de esta vista en la esquina superior derecha, existe un botón para la creación de un usuario (*Create new user*) el cual si se pulsa mostrará el siguiente formulario:

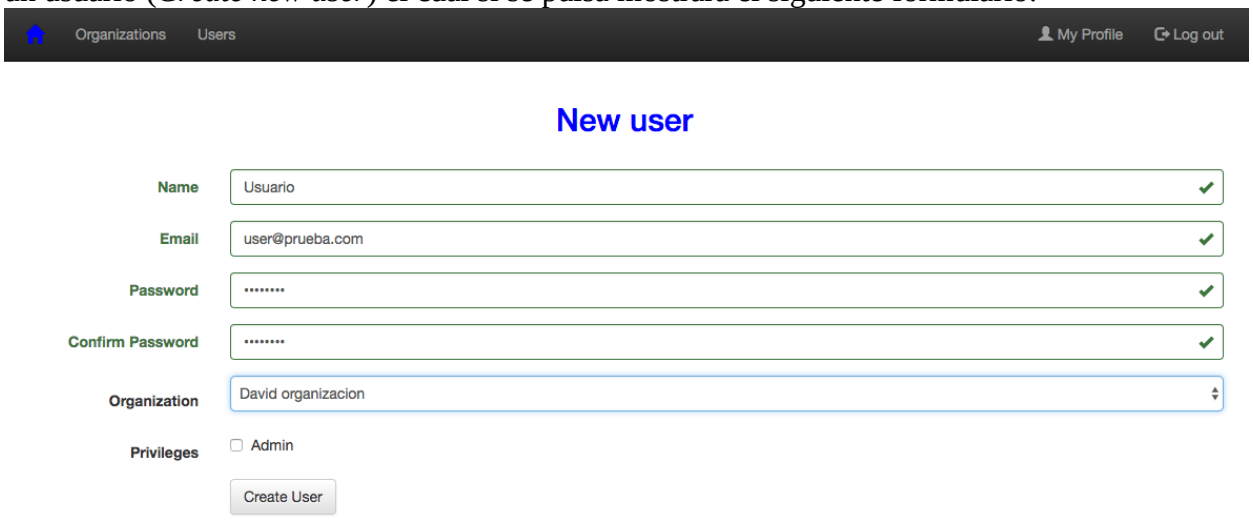


Figura 8-8. Vista para la creación de un usuario

En este ejemplo, ya se han rellenado los datos y se puede observar que cada campo tiene una *v* verde notificando que los datos introducidos son correctos de una forma visual. Si por ejemplo las contraseñas no coincidieran se mostraría una *x* en rojo.

8.6.2 Edición de un usuario

Se puede ver también que, en dicho listado, cada fila tiene un icono verde para la edición, excepto para el usuario que ha iniciado sesión en cuyo caso se ha de utilizar la opción *My profile* para editarlo.

Si se pulsa dicho icono se mostrará un formulario para cambiar el nombre, el email, la organización a la que pertenece o el rol del usuario, como sucede por ejemplo para el Usuario 1:

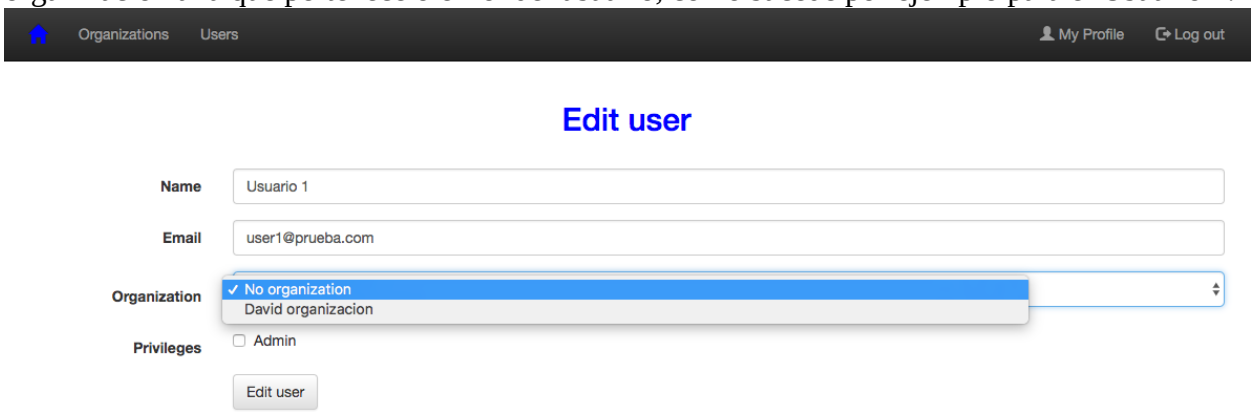


Figura 8-9. Vista para la edición de un usuario

Hay que comentar que para seleccionar la organización se puede desplegar una lista con todas las existentes en el sistema. Esto también se puede realizar al realizar la creación de un usuario.

8.6.3 Eliminación de un usuario

Por último, en la lista de usuarios se puede ver que existe en cada fila correspondiente a un usuario una *x* en rojo para eliminar cada usuario y si se pulsa sobre ella aparecerá un cuadro de confirmación como el siguiente:

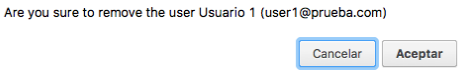


Figura 8-10. Confirmación de eliminación de un usuario

Si en el cuadro anterior se pulsa aceptar la lista de usuarios se recargará, aparecerá un *toast* en la esquina superior derecha confirmando la eliminación y dicho usuario desaparecerá:

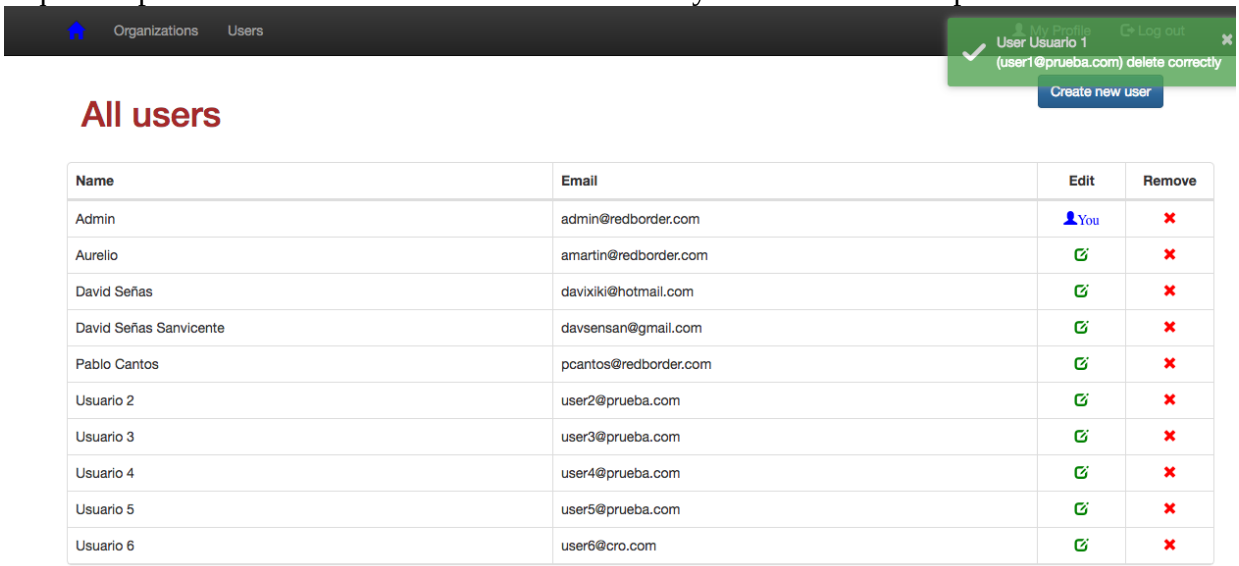


Figura 8-11. Notificación de usuario eliminado

8.7 Listado de las organizaciones

Si un usuario administrador accede a la vista para listar las organizaciones pulsando la opción *Organizations* de la barra de navegación se mostrará una vista muy similar al listado de usuarios, pero en este caso cada organización tendrá los siguientes campos:

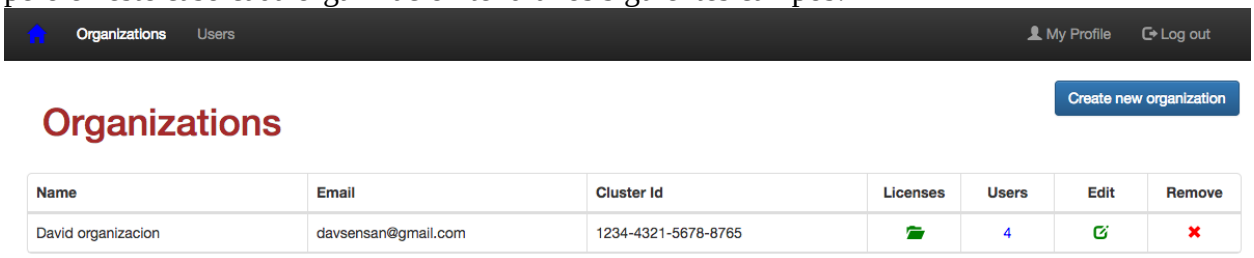


Figura 8-12. Vista con la lista de organizaciones

Al igual que para el listado de usuarios, si hubiese más de 10 organizaciones en el sistema se haría uso de la paginación.

8.7.1 Creación de una organización

Además, dentro de esta vista en la esquina superior derecha, existe un botón para la creación de una organización (*Create new organization*) el cual si se pulsa mostrará el siguiente formulario:

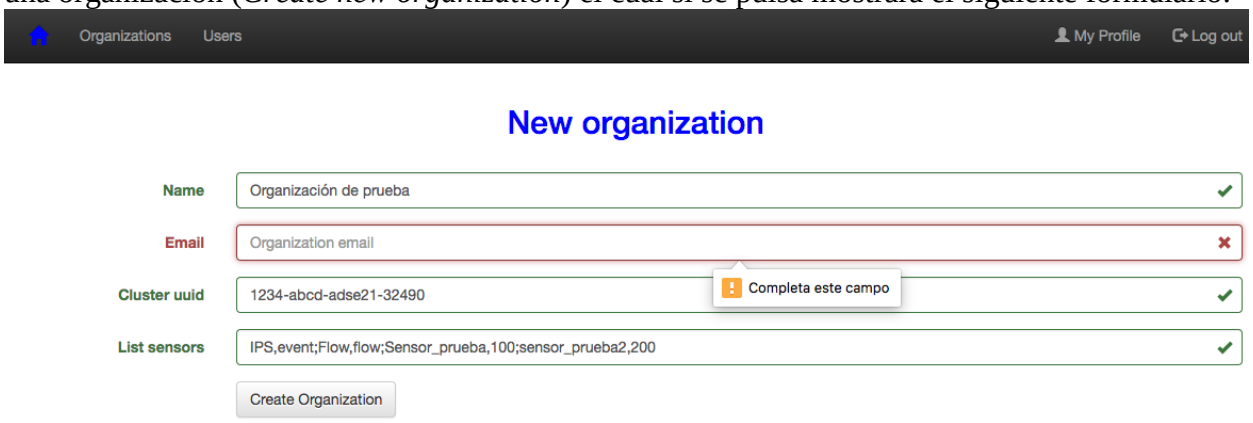


Figura 8-13. Vista para la creación de una organización

En este ejemplo, se ha dejado el email sin rellenar para poder mostrar como aparece una *x* en rojo en ese campo para informar que está incompleto. Además, si se intenta enviar así el formulario no se realizará y aparecerá el mensaje (completa este campo) gracias a HTML5.
Cabe destacar, que la lista de sensores aparece rellena cuando se carga la vista con el valor "IPS,event;Flow,Flow;IPS,ips", sin embargo, para esta organización se ha cambiado eliminando el sensor IPS y añadiendo dos sensores de prueba.

8.7.2 Licencias de una organización

En este caso, en cada fila aparece una columna *licenses* donde aparece una carpeta verde. Si se pulsa dicha carpeta para una fila concreta, se mostrará la vista con el listado de las licencias perteneciente a la organización que está en dicha fila (haciendo uso del componente *ListLicensesPage*):

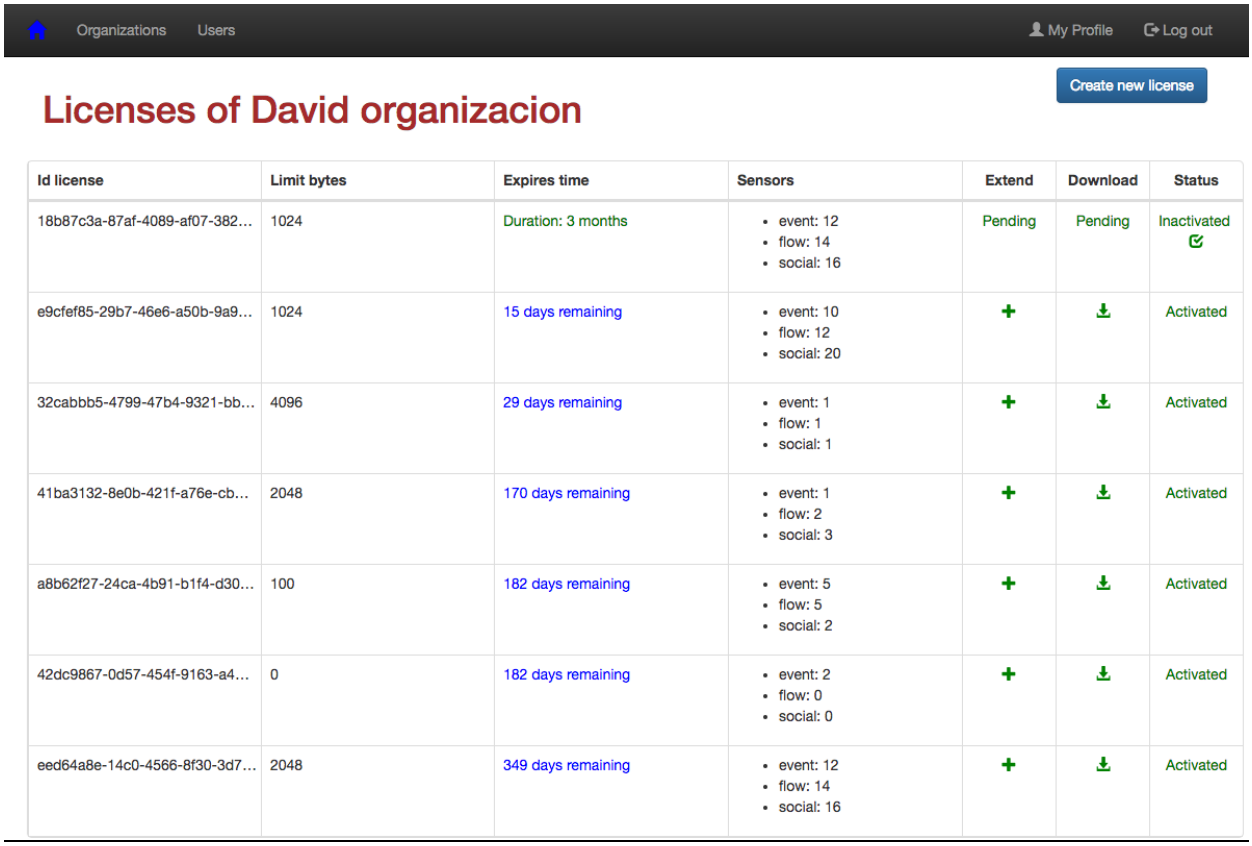


Figura 8-14. Vista para para el listado de las licencias de una organización

En la lista anterior se puede ver que aparece la lista de licencias para la organización *David organización.* Además, al igual que sucede con los usuarios se puede crear una licencia para esa organización o para la que se haya seleccionado en la lista de organizaciones.
Aquí no se entrará en detalle de las diferentes opciones que existen en dicha vista ya que se explicarán más adelante, aunque hay que destacar que sólo en esta vista se les permite a los usuarios administradores activar una licencia inactiva haciendo *click* en el icono verde de la columna *Status* para una licencia inactiva.
Al igual que para todos los listados anteriores, si hubiese más de 10 licencias en el sistema para esta organización se haría uso de la paginación.

8.7.3 Usuarios de una organización

Además del icono de la carpeta para acceder a las licencias de la organización, existe una columna llamada *users* con el número de usuarios de cada organización y si además se pulsa dicho número se mostrarán los usuarios de esa organización:

Figura 8-15. Vista para para el listado de los usuarios de una organización

Además, desde aquí también se podrán crear, editar y eliminar usuarios como se hizo en la vista de la lista de todos los usuarios del sistema.

8.7.4 Edición de una organización

Se puede ver también que, en la lista de organizaciones, cada fila tiene un icono verde para la edición, igual que sucedía en la lista de usuarios

Si se pulsa dicho icono se mostrará un formulario para cambiar el nombre, el email y el identificador del cluster para dicha organización:

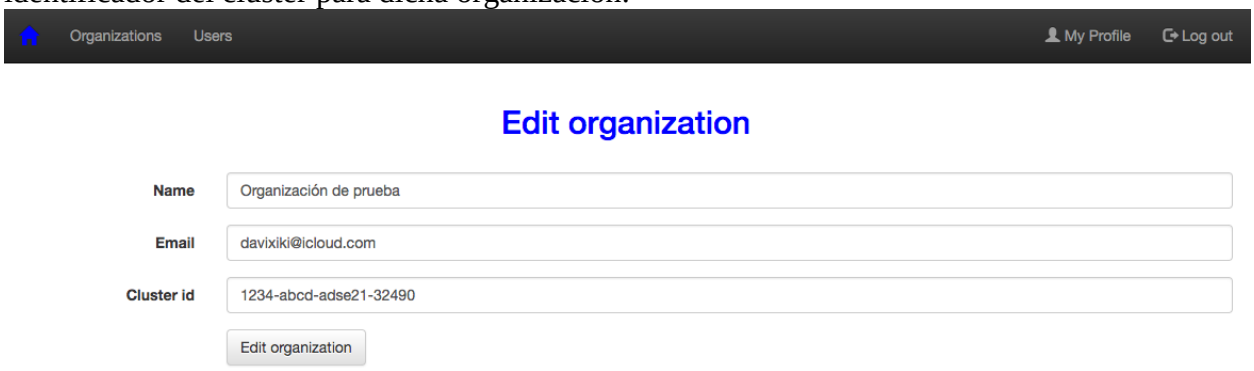


Figura 8-16. Vista para la edición de una organización

Hay que comentar que el campo correspondiente a los sensores no se puede editar una vez se ha creado la organización debido a que, si se edita dicho campo, las licencias creadas previamente podrían ser incongruentes (podrían tener sensores que ya no existen)

8.7.5 Eliminación de una organización

Por último, al igual que para la lista de usuarios, en cada fila de la lista de organizaciones existe una *x* roja para eliminar una organización. Si se pulsa este botón aparecerá lo siguiente:

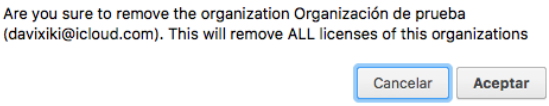


Figura 8-17. Confirmación de eliminación de una organización

Si en el cuadro anterior se pulsa aceptar se eliminará la organización “Organización de prueba” creada anteriormente y todas las licencias de las que dicha organización disponga. Además, los usuarios que perteneciesen a dicha organización ahora no pertenecerán a ninguna.

8.8 Listado de mis licencias

Si un usuario normal accede a la vista para listar las licencias de su organización pulsando la opción *My licenses* de la barra de navegación, se mostrará una vista muy similar al listado de licencias de una organización que se mostró para un usuario administrador (punto 8.7.2), pero en este caso no será posible activar una licencia, sino simplemente conocer su estado:

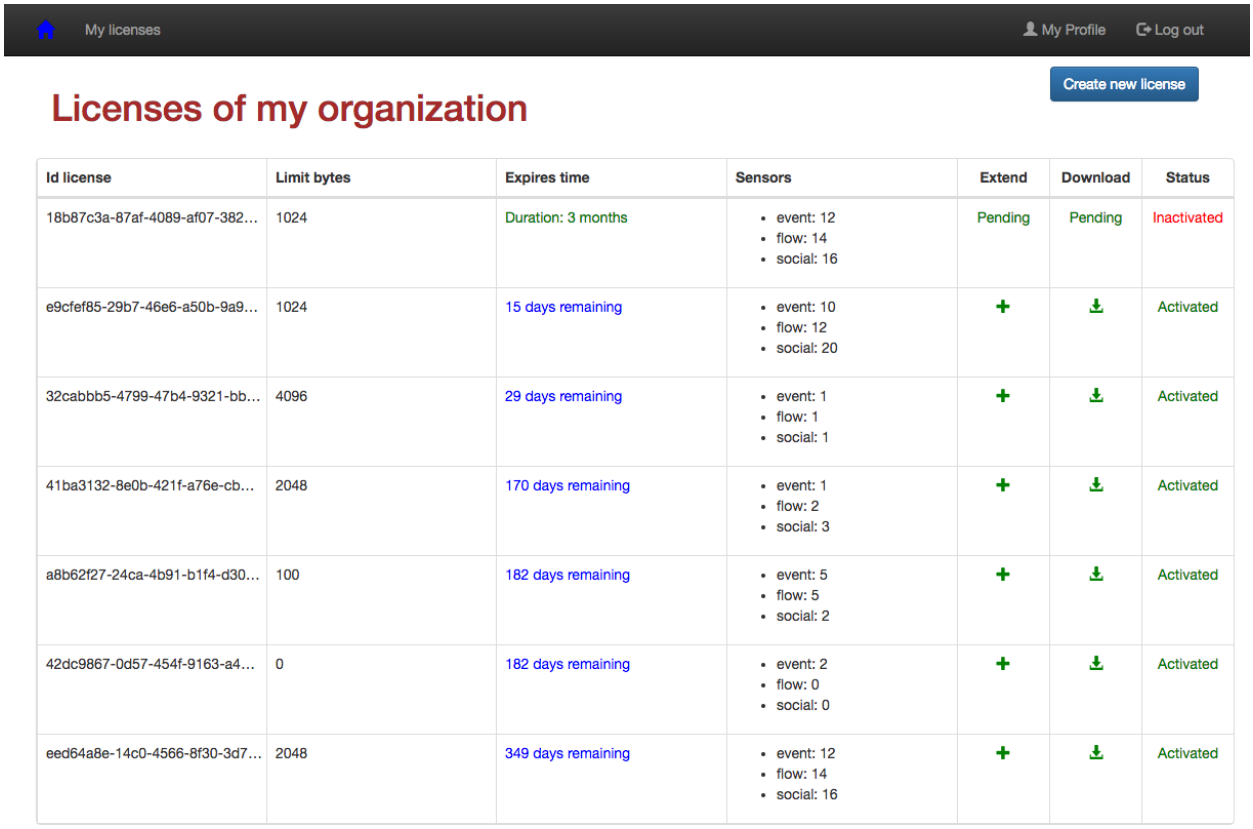


Figura 8-18. Vista con la lista de licencias de mis licencias

Al igual que para todos los listados, si hubiese más de 10 licencias en el sistema para mi organización se haría uso de la paginación.

Cabe destacar que, en esta vista, en la columna *Expires Time* aparecen los días restantes para la expiración de la licencia o la duración de la licencia si no está activada.

8.8.1 Creación de una licencia

En la vista anterior se puede apreciar en la esquina superior derecha un botón llamado *Create new license* y tras pulsarlo se mostrará lo siguiente:

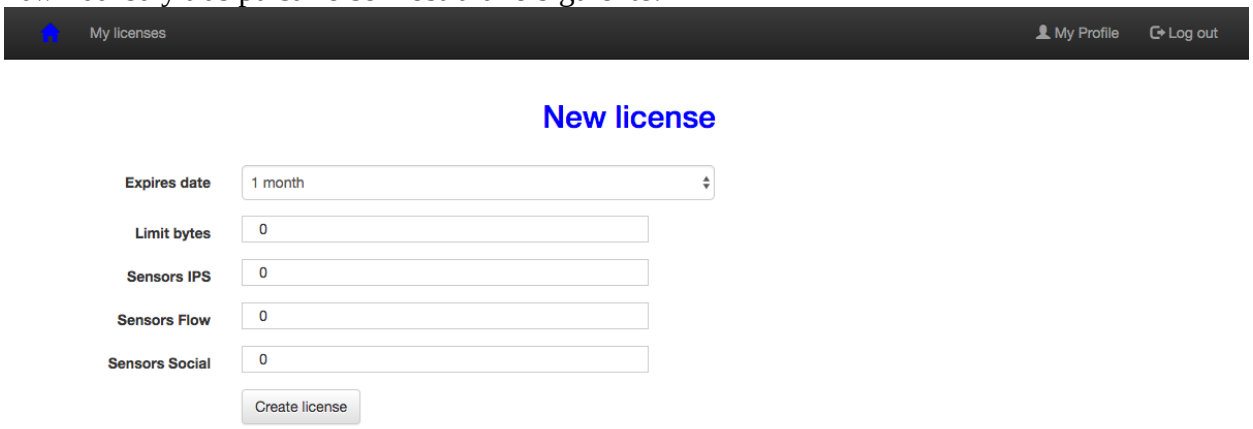


Figura 8-19. Vista con el formulario para la creación de una licencia

En este caso se puede apreciar que aparece un formulario para la creación de una licencia para mi organización y aparecen los sensores que tenía disponibles esa organización con una serie de

campos para elegir el número de sensores. Son campos de tipo numéricos y como máximo se podrán seleccionar 100 sensores de cada tipo.

8.8.2 Extensión de una licencia

Por otro lado, en la lista de licencias se puede observar una columna llamada *extend* con un icono verde con u +, si la licencia está activada, y la palabra *Pending* si la licencia no lo está.
En el caso que la licencia esté activada, el icono del + será un enlace a la siguiente vista:

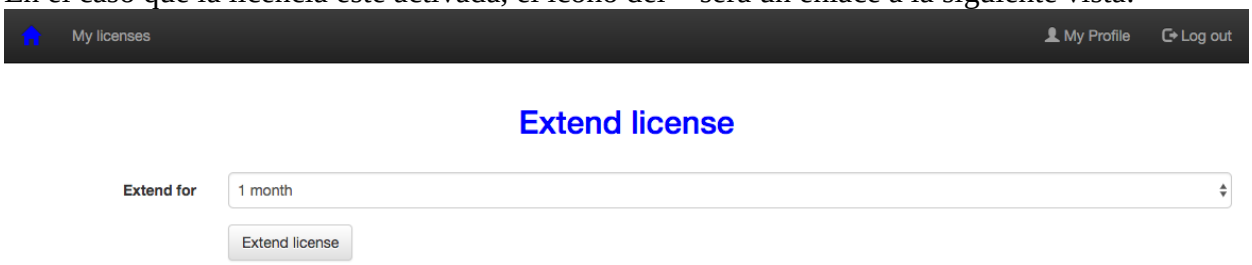


Figura 8-20. Vista con el formulario para la extensión de una licencia

En este caso se mostrará un formulario con un campo que será una lista en la que se podrá seleccionar 1, 3, 6 o 12 meses de duración de la extensión.

8.8.3 Descarga de una licencia

Finalmente, se puede ver que en la lista de licencias existe una columna llamada *Download* en la que existe un icono si la licencia está activada, y tras pulsarlo lanzará la descarga de la licencia que se esté solicitando.

9 CONCLUSIONES Y LÍNEAS DE MEJORA

Una vez desarrollada la aplicación objeto de este proyecto, es hora de considerar las conclusiones que se han obtenido tras él y las posibles mejoras que se le podrían aplicar.
Como conclusiones se pueden destacar los siguientes aspectos:

- La importancia de la toma de requisitos para la realización de una aplicación es crucial ya que de ella depende todo lo demás.
- La gran importancia que está adquiriendo *React* a pesar de ser relativamente moderno, ya que otras grandes plataformas como *Facebook* o *Paypal* han sido migradas a esta tecnología.
- La facilidad de desarrollar interfaces de usuarios con *React* una vez se ha adquirido una base es bastante sencilla, ya que se puede crear un componente y reutilizar tantas veces como se quiera.
- La fluidez con la que se navega por las diferentes opciones de una página web desarrollada con *React* es bastante mayor que con otras tecnologías.
- La cantidad de librerías de tercero existentes para cualquier cosa que se nos ocurra tanto en *React*, como en *nodejs*… no tiene fin, de forma que no es necesario "reinventar la rueda" sino que, simplemente hay que adaptar dichas librerías, desarrollando así mucho más rápido.

Con respecto a mi persona he de decir que gracias a este proyecto he conseguido superar mis expectativas, he conseguido empezar a desarrollar desde cero una aplicación con una tecnología que desconocía totalmente y finalmente he aprendido a base de paciencia y perseverancia, además, una tecnología que hoy día no todo el mundo conoce debido a que es bastante moderna.

Por otro lado, he de decir que la propia empresa RedBorder® ha considerado hacerme un contrato para poder mejorar la aplicación una vez terminado este proyecto y que de esta forma sea utilizada por los propios clientes de la misma, por lo que no puedo estar más satisfecho con mi proyecto de fin de carrera realizado.

Estas mejoras serán, entre otras, las siguientes:

- Cada organización ya no solo dispondrá de un identificador de cluster, sino que una organización podrá disponer de diferentes cluster con un nombre y un identificador, de modo que ahora las licencias se creen para un cluster determinado de una organización.
- Añadir la posibilidad de que los clientes puedan activar las licencias por sensores, es decir, si se tiene una licencia con 100 sensores IPS y 20 Flow activada, poder tomar solo 10 sensores IPS y 10 Flow, quedando 90 y 10 restantes, respectivamente.
- Notificar a los usuarios cuando tengan licencias próximas a caducar.
- Añadir los tipos de sensores diferentes para una organización mediante un selector múltiple en lugar de escribir una cadena separada por comas (,) y puntos y comas (;).
- Posibilidad de gestionar las licencias mediante una API.

ANEXO A
CÓDIGO DE FRONT-END

Código A-1. Componente client/src/components/Base.jsx

```
import React from 'react';
import { Link, IndexLink } from 'react-router';
import Auth from '../modules/Auth';
import PropTypes  from 'prop-types';


/* Componente Base encargado de crear la barra de navegación superior.
Hará uso de React router para la navegación entre las diferentes opciones del menú
Recibirá los siguientes parámetros:
  1) Children: Componente que se mostrará haciendo uso de React-Router bajo la barra de navegación
*/


const Base = ({children}) => (
  <div>
    <nav className="navbar navbar-inverse navbar-fixed-top">
      <div className="container">
        <div className="navbar-header">
          <button type="button" className="navbar-toggle collapsed" data-toggle="collapse"   data-target="#base-collapse"   aria-expanded="false" aria-controls="navbar">
            <span className="sr-only">Toggle navigation</span>
            <span className="icon-bar"></span>
            <span className="icon-bar"></span>
            <span className="icon-bar"></span>
          </button>
          <IndexLink   className="navbar-brand"   to="/"   style={{color: 'blue'}}><span className="glyphicon glyphicon-home"></span></IndexLink>
        </div>
        <div className="collapse navbar-collapse" id="base-collapse">
       {Auth.isUserAuthenticated() ? (
          Auth.isAdmin() ? (
          <div>
            <div>
```

```
              <ul className="nav navbar-nav" >
                <li>
                  <Link to="/listOrgs"> Organizations </Link>
                </li>
                <li>
                  <Link to="/listUsers/all/all"> Users </Link>
                </li>
              </ul>
            </div>
            <div>
              <ul className="nav navbar-nav navbar-right" >
                <li>
                  <Link    to="/user/edit"><span    className="glyphicon
glyphicon-user"></span> My Profile</Link>
                </li>
                <li>
                  <Link      to="/logout"><span      className="glyphicon
glyphicon-log-out"></span> Log out</Link>
                </li>
              </ul>
            </div>
          </div>
          ) : (
            <div>
              {Auth.hasOrganization() ? (
              <ul className="nav navbar-nav" >
                <li>
                  <Link              to={"/listLicenses/"              +
localStorage.getItem('userProfileOrg')  +  "/"  +  encodeURIComponent("my
organization")}> My licenses </Link>
                </li>
              </ul>
              ) : null }
              <ul className="nav navbar-nav navbar-right">
                <li>
                  <Link    to="/user/edit"><span    className="glyphicon
glyphicon-user"></span> My Profile</Link>
                </li>
                <li>
                  <Link      to="/logout"><span      className="glyphicon
```

```
glyphicon-log-out"></span> Log out</Link>
                </li>
              </ul>
            </div>
          )
        ) : (
            <ul className="nav navbar-nav navbar-right">
              <li>
                <Link to="/login"><span className="glyphicon glyphicon-
log-in"></span> Log in user</Link>
              </li>
            </ul>
        )}
        </div>
      </div>
    </nav>
    <div className="container">
    {children}
    </div>
  </div>
);


//Haciendo uso de propTypes se comprueba que existe el componente
'children', ya que es obligatorio
Base.propTypes = {
  children: PropTypes.object.isRequired,
};
```

Código A-2. Componente client/src/components/CreateLicensesForm.jsx

```
import React from 'react';
import {Form, FormControl, FormGroup, Col, ControlLabel, Checkbox,
Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';
/* Componente CreateLicenseForm encargado de crear el formulario para la
creación de un usuario.
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del
formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de
entrada de texto del formulario.
```

```
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) license: Objeto donde se almacenará el usuario a crear.
  5) organizations: Lista de las organizaciones disponibles.
*/

const CreateLicenseForm = ({
  onSubmit,
  onChange,
  onChangeSensors,
  errors,
  license,
  sensors
}) =>
  (<div>
    <div className="row">
      <h2  className="text-center"  style={{color:"blue"}}>  New  license
</h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup                                        controlId="duration"
validationState={errors.duration=="" ? null : errors.duration} >
        <Col componentClass={ControlLabel} sm={2}>
          Expires date
        </Col>
        <Col sm={5}>
          <FormControl      name="duration"      componentClass="select"
onChange={onChange}>
            <option value={1} key={"1"} > 1 month</option>
            <option value={3} key={"2"}> 3 months</option>
            <option value={6} key={"3"}> 6 months</option>
            <option value={12} key={"4"}> 1 year</option>
          </FormControl>
        </Col>
      </FormGroup>


      <FormGroup                                       controlId="limit_bytes"
validationState={errors.limit_bytes=="" ? null : errors.limit_bytes}>
        <Col componentClass={ControlLabel} sm={2}>
          Limit bytes
```

```
        </Col>
        <Col sm={10}>
          <input          className="col-sm-5"          name="limit_bytes"
onChange={onChange} value={license.limit_bytes} type="number" min="0" />
        </Col>
      </FormGroup>
      {
      sensors.map(function(sensor,key){

        return (<div key={key}>
          <FormGroup                                    controlId={sensor}
validationState={errors.sensors[sensor]==""        ?        null        :
errors.sensors[sensor]}>
            <Col componentClass={ControlLabel} sm={2}>
              Sensors {sensor.split(",")[0]}
            </Col>
            <Col sm={10}>
              <input   className="col-sm-5"   name={sensor.split(",")[1]}
onChange={onChangeSensors} value={license[sensor]} type="number" min="0"
max="100" defaultValue="0"/>
            </Col>
          </FormGroup>
          </div>)
      })
      }

      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Create license
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);

//Haciendo uso de propTypes comprobamos si todos los parámetros son
recibidos por el componente y en el formato correcto
CreateLicenseForm.propTypes = {
```

```
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  onChangeSensors: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  license: PropTypes.object.isRequired,
  sensors: PropTypes.array.isRequired
};

export default CreateLicenseForm;
```

Código A-3. Componente client/src/components/CreateOrgForm.jsx

```
import React from 'react';
import { Form, FormControl, FormGroup, Col, ControlLabel, Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';


/* Componente CreateOrgForm encargado de crear el formulario de creación de una organización
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrada de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) org: Objeto donde se almacenará la organización a crear
*/


const CreateOrgForm = ({
  onSubmit,
  onChange,
  errors,
  org
}) => (
  <div>
    <div className="row">
    </div>


    <div className="row">
      <h2     className="text-center"     style={{color:"blue"}}>     New organization </h2>
```

```
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="name" validationState={errors.name=="" ? null
: errors.name} >
        <Col componentClass={ControlLabel} sm={2}>
          Name
        </Col>
        <Col sm={10}>
          <FormControl name ="name" type="name" placeholder="Organization
name" required="true" onChange={onChange} value={org.name}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup  controlId="email"  validationState={errors.email==""  ?
null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
        <Col sm={10}>
          <FormControl         name         ="email"         type="email"
placeholder="Organization   email"   required="true"   onChange={onChange}
value={org.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                                    controlId="cluster_id"
validationState={errors.cluster_id=="" ? null : errors.cluster_id} >
        <Col componentClass={ControlLabel} sm={2}>
          Cluster uuid
        </Col>
        <Col sm={10}>
          <FormControl        name        ="cluster_id"        type="name"
placeholder="Cluster    uuid"    required="true"    onChange={onChange}
value={org.cluster_id}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>
```

```
      <FormGroup                                          controlId="sensors"
validationState={errors.cluster_id=="" ? null : errors.cluster_id} >
        <Col componentClass={ControlLabel} sm={2}>
          List sensors
        </Col>
        <Col sm={10}>
          <FormControl        name        ="sensors"        type="name"
placeholder="Sensor1,id;Sensor2,id;Sensor3,id;..."       required="true"
onChange={onChange} value={org.sensors}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>



      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Create Organization
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);

//Mediante propTypes se comprueba que el componente recibe correctamente
todos los parámetros
CreateOrgForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  org: PropTypes.object.isRequired
};

export default CreateOrgForm;
```

Código A-4. Componente client/src/components/CreateUserForm.jsx

```
import React from 'react';
import  {Form,  FormControl,  FormGroup,  Col,  ControlLabel,  Checkbox,
```

```
Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';


/* Componente CreateUserForm encargado de crear el formulario para la creación de un usuario.
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrada de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) user: Objeto donde se almacenará el usuario a crear.
  5) organizations: Lista de las organizaciones disponibles.
*/
const CreateUserForm = ({
  onSubmit,
  onChange,
  errors,
  user,
  organizations
}) => (
  <div>
    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> New user </h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="name" validationState={errors.name=="" ? null : errors.name} >
        <Col componentClass={ControlLabel} sm={2}>
          Name
        </Col>
        <Col sm={10}>
          <FormControl name ="name" type="name" placeholder="Name" required="true" onChange={onChange} value={user.name}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup controlId="email" validationState={errors.email=="" ?
```

```
null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
        <Col sm={10}>
          <FormControl  name  ="email"  type="email"  placeholder="Email"
required="true" onChange={onChange} value={user.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                                        controlId="password"
validationState={errors.password=="" ? null : errors.password}>
        <Col componentClass={ControlLabel} sm={2}>
          Password
        </Col>
        <Col sm={10}>
          <FormControl           name="password"           type="password"
placeholder="Password"         required="true"         onChange={onChange}
value={user.password}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                                  controlId="confir_password"
validationState={errors.confir_password==""        ?        null        :
errors.confir_password}>
        <Col componentClass={ControlLabel} sm={2}>
          Confirm Password
        </Col>
        <Col sm={10}>
          <FormControl       name="confir_password"       type="password"
placeholder="Confirm     password"required="true"     onChange={onChange}
value={user.confirm_password}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup controlId="organization">
        <Col componentClass={ControlLabel} sm={2}>
          <ControlLabel>Organization</ControlLabel>
```

```
        </Col>
        <Col sm={10}>
          <FormControl    name="organization"    componentClass="select"
placeholder="Select          organization"          onChange={onChange}
value={user.organization}>

            <option value="No" key={"2"}> No organization</option>
            {

                organizations.map((organization, key) => {
                  return   <option   value={organization.id}   key={key}>
{organization.name} </option>
                })
            }
          </FormControl>
        </Col>
      </FormGroup>

      <FormGroup controlId="role">
        <Col componentClass={ControlLabel} sm={2}>
          <ControlLabel>Privileges</ControlLabel>
        </Col>
        <Col sm={10}>
            <Checkbox name="role" onChange={onChange}>
              Admin
            </Checkbox>
        </Col>
      </FormGroup>

      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Create User
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);
```

```
//Haciendo uso de propTypes comprobamos si todos los parámetros son recibidos por el componente y en el formato correcto
CreateUserForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  user: PropTypes.object.isRequired
};

export default CreateUserForm;
```

Código A-5. Componente client/src/components/Dashboard.jsx

```
import React from 'react';
import { Panel } from 'react-bootstrap';

/*
Componente encargado de mostrar la pagina principal cuando un usuario se ha autenticado
No recibe parámetros
*/
const Dashboard = () => (
    <div>
        <Panel header="Licenses management" bsStyle="info">
            This is the dashboard page, if you can see this, you are autenticated
        </Panel>
    </div>
);

export default Dashboard;
```

Código A-6. Componente client/src/components/EditOrgForm.jsx

```
import React from 'react';
import {Form, FormControl, FormGroup, Col, ControlLabel, Checkbox, Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';

/*
Componente encargado de crear el formulario para la edición de una
```

```
organización
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del
formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de
entrad de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) organization: Objeto con la información de la organización a editar.
*/
const EditOrgForm = ({
  onSubmit,
  onChange,
  errors,
  organization
}) => (
  <div>

    <div className="row">
      <h2     className="text-center"     style={{color:"blue"}}>     Edit
organization </h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>

      <FormGroup controlId="name" validationState={errors.name=="" ? null
: errors.name} >
        <Col componentClass={ControlLabel} sm={2}>
          Name
        </Col>
        <Col sm={10}>
          <FormControl   name   ="name"   type="name"   placeholder="Name"
required="true" onChange={onChange} value={organization.name}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup  controlId="email"  validationState={errors.email==""  ?
null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
```

```
        </Col>
        <Col sm={10}>
          <FormControl name ="email" type="email" placeholder="Email"
required="true" onChange={onChange} value={organization.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                                    controlId="cluster_id"
validationState={errors.cluster_id=="" ? null : errors.cluster_id} >
        <Col componentClass={ControlLabel} sm={2}>
          Cluster id
        </Col>
        <Col sm={10}>
          <FormControl     name     ="cluster_id"     type="cluster_id"
placeholder="cluster_id"       required="true"       onChange={onChange}
value={organization.cluster_id}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Edit organization
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);


//Utilización de propTypes para verificar que los parámetros son
recibidos y de forma correcta
EditOrgForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  organization: PropTypes.object.isRequired
};
```

```
export default EditOrgForm;
```

Código A-7. Componente client/src/components/EditUserForm.jsx

```
import React from 'react';
import {Form, FormControl, FormGroup, Col, ControlLabel, Checkbox, Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';
/*
Componente encargado de crear el formulario para la edición de un usuario
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrad de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) user: Objeto con la información del usuario a editar.
  5) organizations: Lista de las organizaciones disponibles.
*/
const EditUserForm = ({
  onSubmit,
  onChange,
  errors,
  user,
  organizations
}) => (
  <div>

    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> Edit user </h2>
      <br></br>
    </div>

    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="name" validationState={errors.name=="" ? null : errors.name} >
        <Col componentClass={ControlLabel} sm={2}>
          Name
        </Col>
```

```
        <Col sm={10}>
          <FormControl name ="name" type="name" placeholder="Name"
required="true" onChange={onChange} value={user.name}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup controlId="email" validationState={errors.email==""  ?
null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
        <Col sm={10}>
          <FormControl name ="email" type="email" placeholder="Email"
required="true" onChange={onChange} value={user.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup controlId="organization">
        <Col componentClass={ControlLabel} sm={2}>
          <ControlLabel>Organization</ControlLabel>
        </Col>
        <Col sm={10}>
          <FormControl    name="organization"    componentClass="select"
placeholder="Select          organization"          onChange={onChange}
value={user.organization}>

            <option value="No">No organization</option>
            {

              organizations.map((organization, key) => {
                return   <option   value={organization.id}   key={key}   >
{organization.name} </option>
              })
            }
          </FormControl>
        </Col>
      </FormGroup>
```

```
      <FormGroup controlId="role">
        <Col componentClass={ControlLabel} sm={2}>
          <ControlLabel>Privileges</ControlLabel>
        </Col>
        <Col sm={10}>
            <Checkbox name="role" onChange={onChange}>
              Admin
            </Checkbox>
        </Col>
      </FormGroup>

      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Edit user
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);
//Utilizando propTypes validamos si se reciben todos los parámetros y de forma correcta
EditUserForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  user: PropTypes.object.isRequired
};

export default EditUserForm;
```

Código A-8. Componente client/src/components/ExtendLicenseForm.jsx

```
import React from 'react';
import {Form, FormControl, FormGroup, Col, ControlLabel, Checkbox, Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';
/* Componente ExtendLicenseForm encargado de crear el formulario para la creación de un usuario.
```

```
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del
formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de
entrada de texto del formulario.
  4) license: Objeto donde se almacenará el usuario a crear.
  5) organizations: Lista de las organizaciones disponibles.
*/


const ExtendLicenseForm = ({
  onSubmit,
  onChange,
  license
}) =>
  (<div>
    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> Extend license
</h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="duration">
        <Col componentClass={ControlLabel} sm={2}>
          Extend for
        </Col>
        <Col sm={10}>
          <FormControl      name="duration"      componentClass="select"
onChange={onChange}>
            <option value={1} key={"1"} > 1 month</option>
            <option value={3} key={"2"}> 3 months</option>
            <option value={6} key={"3"}> 6 months</option>
            <option value={12} key={"4"}> 1 year</option>
          </FormControl>
        </Col>
      </FormGroup>


      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Extend license
```

```
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);


//Haciendo uso de propTypes comprobamos si todos los parámetros son recibidos por el componente y en el formato correcto
ExtendLicenseForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  license: PropTypes.object.isRequired
};


export default ExtendLicenseForm;
```

Código A-9. Componente client/src/components/ForgotForm.jsx

```
import React from 'react';
import {Panel, Form, FormControl, FormGroup, Col, HelpBlock, ControlLabel, Checkbox, Button, FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';
/*
Componente encargado de crear el formulario para el recordatorio de contraseña de usuario
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrad de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) user: Objeto con la información del usuario que solicita el recordatorio de contraseña.
*/
const ForgotForm = ({
  onSubmit,
  onChange,
  errors,
  user
}) => (
```

```
  <div>


    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> Forgot password
</h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>


      <FormGroup  controlId="email"  validationState={errors.email==""  ?
null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
        <Col sm={10}>
          <FormControl  name  ="email"  type="email"  placeholder="Email"
required="true" onChange={onChange} value={user.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Remember me
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);
//Utilizando propTypes verificamos que el componente recibe todos los
parámetros de forma correcta
ForgotForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  user: PropTypes.object.isRequired
};
```

```
export default ForgotForm;
```

Código A-10. Componente client/src/components/Home.jsx

```
import React from 'react';
import { Panel } from 'react-bootstrap';
/*
Componente encargado de mostrar la página de inicio
*/
const Home = () => (
    <div>
        <Panel header="Licenses management" bsStyle="info">
            This is the home page
        </Panel>
    </div>

);
export default Home;
```

Código A-11. Componente client/src/components/ListLicenses.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import { Link } from 'react-router';
import {BootstrapTable, TableHeaderColumn} from 'react-bootstrap-table';

/*
Componente encargado de mostrar una tabla con las licencias recibidas por parámetros.
Recibe los siguientes parámetros:
    1) licenses: Lista de licencias a listar.
    2) orgName: Nombre de la organización a la que pertenecen las licencias que se están mostrando.
    3) orgId: Identificador de la organización para saber qué tipo de licencia hay que crear y a qué organización pertence (admin
    4) expiresFormat: Función encargada de mostrar el número de días restantes hasta la expiración de la licencia
*/
const ListLicenses = ({
    licenses,
    orgId,
    orgName,
```

```
    expiresFormat,
    sensorsFormat,
    downloadFormat,
    extendFormat,
    activateFormat }) => (
        <div>
        <div className="row">
            <div className="col-md-10">
                    <h1  className="text-left"  style={{color:"brown"}}  >
Licenses of {orgName} </h1>
                </div>
                <div className="col-md-2" >
                    <button  className="btn btn-primary  text-right"><Link
style={{color:"white"}}              to={"/license/new/"              +
localStorage.getItem('userProfileId')  +  "/"  +    orgId  }  >Create  new
license </Link></button>
                </div>
            </div>
            <div className="row">
                <br></br>
                <BootstrapTable data={licenses} >
                    <TableHeaderColumn       dataField="id"       hidden
isKey={true}> Id </TableHeaderColumn>
                    <TableHeaderColumn    dataField="license_uuid">    Id
license </TableHeaderColumn>
                    <TableHeaderColumn  dataField="limit_bytes"  >  Limit
bytes </TableHeaderColumn>
                    <TableHeaderColumn             dataField="expires_at"
dataFormat={expiresFormat} > Expires time </TableHeaderColumn>
                    <TableHeaderColumn                dataField="sensors"
dataFormat={sensorsFormat}> Sensors </TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={extendFormat}    dataAlign='center'    width="90">    Extend
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={downloadFormat}   dataAlign='center'   width="90">   Download
</TableHeaderColumn>
                    <TableHeaderColumn                dataField="enabled"
dataFormat={activateFormat}    dataAlign='center'    width="90">    Status
</TableHeaderColumn>
                </BootstrapTable>
            </div>
        </div>
```

```
);


//Haciendo uso de propTypes se comprueban que todos los parámetros son recibidos correctamente
ListLicenses.propTypes = {
    licenses: PropTypes.array.isRequired,
    sensorsFormat: PropTypes.func.isRequired,
    activateFormat: PropTypes.func.isRequired,
    extendFormat: PropTypes.func.isRequired,
    downloadFormat: PropTypes.func.isRequired,
    expiresFormat: PropTypes.func.isRequired,
    orgName: PropTypes.string.isRequired,
    orgId: PropTypes.string.isRequired
}
export default ListLicenses;
```

Código A-12. Componente client/src/components/ListOrgs.jsx

```
import React from 'react';
import { Link } from 'react-router';
import PropTypes from 'prop-types';
import {BootstrapTable, TableHeaderColumn} from 'react-bootstrap-table';


/*
Componente encargado de mostrar una tabla con las organizaciones recibidas por parámetros.
Recibe los siguientes parámetros:
    1) organizations: Lista de organizaciones a listar.
    2) removeOrgFormat: Función encargada de dar formato a la entrada de la tabla correspondiente a la eliminación de una organización
    3) editOrgFormat: Función encargada de dar formato a la entrada de la tabla correspondiente a la edición de una organización
    4) countUsersFormat: Función encargada de dar formato a la entrada de la tabla correspondiente al número de usuarios que existen en esa organización
*/
const ListOrgs = ({
    organizations,
    removeOrgFormat,
    listLicensesFormat,
    editOrgFormat,
```

```
    countUsersFormat}) => (
        <div>


            <div className="row">
                <div className="col-md-10">
                    <h1  className="text-left"  style={{color:"brown"}}  >
Organizations </h1>
                </div>
                <div className="col-md-2">


                    <button  className="btn btn-primary text-right"><Link
style={{color:"white"}}  to="/organization/new">Create  new  organization
</Link></button>
                </div>
            </div>
            <div className="row">
                <br></br>


                <BootstrapTable data={organizations} >
                    <TableHeaderColumn       dataField="name">       Name
</TableHeaderColumn>
                    <TableHeaderColumn  dataField="email"  isKey>  Email
</TableHeaderColumn>
                    <TableHeaderColumn dataField="cluster_id"> Cluster Id
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={listLicensesFormat}  dataAlign="center"  width="90"> Licenses
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={countUsersFormat}   dataAlign="center"   width="90">   Users
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={editOrgFormat}    dataAlign="center"    width="90">    Edit
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={removeOrgFormat}   dataAlign="center"   width="90">   Remove
</TableHeaderColumn>
                </BootstrapTable>
            </div>
        </div>
    );


//Haciendo uso de propTypes se comprueba que todos los parámetros son
```

```
recibidos de forma correcta por el componente
ListOrgs.propTypes = {
    removeOrgFormat: PropTypes.func.isRequired,
    editOrgFormat: PropTypes.func.isRequired,
    listLicensesFormat: PropTypes.func.isRequired,
    countUsersFormat: PropTypes.func.isRequired,
    organizations: PropTypes.array.isRequired
}

export default ListOrgs;
```

Código A-13. Componente client/src/components/ListUsers.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import { Link } from 'react-router';
import {BootstrapTable, TableHeaderColumn} from 'react-bootstrap-table';


/*
Componente encargado de mostrar una tabla con los usuarios recibidos por
parámetros.
Recibe los siguientes parámetros:
    1) users: Lista de usuarios a listar.
    2) removeUserFormat: Función encargada de dar formato a la entrada de
la tabla correspondiente a la eliminación de un usuario
    3) editUserFormat: Función encargada de dar formato a la entrada de
la tabla correspondiente a la edición de un usuario
    4) orgName: Nombre de la organización a la que pertenecen los
usuarios que se están mostrando.
*/
const ListUsers = ({
    users,
    removeUserFormat,
    editUserFormat,
    orgName }) => (
        <div>
        <div className="row">
            <div className="col-md-10">
                { //Si el nombre de la organización es "all" se mostrarán
todos los usuarios
                    orgName === "all" ?
```

```
                    <h1  className="text-left"  style={{color:"brown"}}  >
All users </h1>
                    :
                    <h1  className="text-left"  style={{color:"brown"}}  >
Users of {orgName} </h1>
                    }
                </div>
                <div className="col-md-2" >
                    <button  className="btn btn-primary text-right"><Link
style={{color:"white"}} to="/user/new">Create new user </Link></button>
                </div>
            </div>
            <div className="row">
                <br></br>
                <BootstrapTable data={users} >
                    <TableHeaderColumn       dataField="name">       Name
</TableHeaderColumn>
                    <TableHeaderColumn  dataField="email"  isKey>  Email
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={editUserFormat}    dataAlign="center"    width="90">    Edit
</TableHeaderColumn>
                    <TableHeaderColumn                    dataField="id"
dataFormat={removeUserFormat}   dataAlign="center"   width="90">   Remove
</TableHeaderColumn>
                </BootstrapTable>
            </div>
        </div>
);


//Haciendo uso de propTypes se comprueban que todos los parámetros son
recibidos correctamente
ListUsers.propTypes = {
    orgName: PropTypes.string.isRequired,
    removeUserFormat: PropTypes.func.isRequired,
    editUserFormat: PropTypes.func.isRequired,
    users: PropTypes.array.isRequired
}
export default ListUsers;
```

Código A-14. Componente client/src/components/LoginForm.jsx

```
import React from 'react';
```

```
import {Form, FormControl, FormGroup, Col, ControlLabel, Button, FeedBack} from 'react-bootstrap';
import { Link } from 'react-router';
import PropTypes  from 'prop-types';


/* Componente LoginForm encargado de crear el formulario para el inicio de sesión
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrada de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) user: Objeto donde que contendrá el usuario que solicita el inicio de sesión
*/
const LoginForm = ({
  onSubmit,
  onChange,
  errors,
  user
}) => (
  <div className="container">
    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> Log in form </h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="email" validationState={errors.email=="" ? null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
        <Col sm={10}>
          <FormControl name ="email" type="email" placeholder="Email" required="true" onChange={onChange} value={user.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>
```

```
      <FormGroup                                    controlId="password"
validationState={errors.password=="" ? null : errors.password}>
        <Col componentClass={ControlLabel} sm={2}>
          Password
        </Col>
        <Col sm={10}>
          <FormControl          name="password"          type="password"
placeholder="Password"           required="true"     onChange={onChange}
value={user.password}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Sign in
          </Button>
          <Link to={'/forgot'}> Forgot your password?</Link>
        </Col>
      </FormGroup>
    </Form>
  </div>
);

//Haciendo uso de propTypes se comprueban que todos los parámetros son
recibidos correctamente
LoginForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  user: PropTypes.object.isRequired
};

export default LoginForm;
```

Código A-15. Componente client/src/components/NewPasswordForm.jsx

```
import React from 'react';
import  {Form,  FormControl,  FormGroup,  Col,  ControlLabel,  Button,
```

```
FeedBack} from 'react-bootstrap';
import PropTypes  from 'prop-types';


/* Componente NewPasswordForm encargado de crear el formulario para la creación de una nueva contraseña olvidada.
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrada de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) user: Objeto que contendrá el usuario con la nueva contraseña
*/
const NewPasswordForm = ({
  onSubmit,
  onChange,
  errors,
  user
}) => (
  <div>
    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> New password form </h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="password" validationState={errors.password=="" ? null : errors.password}>
        <Col componentClass={ControlLabel} sm={2}>
          Password
        </Col>
        <Col sm={10}>
          <FormControl name="password" type="password" placeholder="Password" required="true" onChange={onChange} value={user.password}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>
      <FormGroup controlId="confir_password" validationState={errors.confir_password=="" ? null : errors.confir_password}>
```

```
        <Col componentClass={ControlLabel} sm={2}>
          Confirm Password
        </Col>
        <Col sm={10}>
          <FormControl       name="confir_password"       type="password"
placeholder="Confirm   Password"   required="true"   onChange={onChange}
value={user.confir_password}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>
      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Change Password
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);

//Haciendo uso de propTypes se comprueban que todos los parámetros son
recibidos correctamente
NewPasswordForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  user: PropTypes.object.isRequired
};

export default NewPasswordForm;
```

Código A-16. Componente client/src/components/ProfileForm.jsx

```
import React from 'react';
import   {Panel,   Form,   FormControl,   FormGroup,   Col,   HelpBlock,
ControlLabel, Checkbox, Button, FeedBack} from 'react-bootstrap';
import { Link } from 'react-router';
import PropTypes  from 'prop-types';
```

```
/* Componente ProfileForm encargado de crear el formulario para el cambio del perfil de un usuario.
Recibirá los siguientes parámetros:
  1) onSubmit: Función llamada al presionar el boton 'submit' del formulario.
  2) onChange: Función encargada de manejar los cambios en los campos de entrada de texto del formulario.
  3) errors: Objeto utilizado para la validación visual del formulario.
  4) user: Objeto que contendrá el usuario con los nuevos cambios.
*/
const ProfileForm = ({
  onSubmit,
  onChange,
  errors,
  user
}) => (
  <div>
    <div className="row">
      <h2 className="text-center" style={{color:"blue"}}> My profile form </h2>
      <br></br>
    </div>
    <Form horizontal onSubmit={onSubmit}>
      <FormGroup controlId="name" validationState={errors.name=="" ? null : errors.name} >
        <Col componentClass={ControlLabel} sm={2}>
          Name
        </Col>
        <Col sm={10}>
          <FormControl name ="name" placeholder="Name" required="true" onChange={onChange} value={user.name}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup controlId="email" validationState={errors.email=="" ? null : errors.email} >
        <Col componentClass={ControlLabel} sm={2}>
          Email
        </Col>
```

```
        <Col sm={10}>
          <FormControl name ="email" type="email" placeholder="Email"
required="true" onChange={onChange} value={user.email}/>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                                    controlId="new_password"
validationState={errors.new_password=="" ? null : errors.new_password}>
        <Col componentClass={ControlLabel} sm={2}>
          New Password
        </Col>
        <Col sm={10}>
          <FormControl        name="new_password"        type="password"
placeholder="New              Password"              onChange={onChange}
value={user.new_password}/>
          <HelpBlock>If   this   field   is   empty,   password   will   not
change.</HelpBlock>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                              controlId="confir_new_password"
validationState={errors.confir_new_password==""      ?      null      :
errors.confir_new_password}>
        <Col componentClass={ControlLabel} sm={2}>
          Confir Password
        </Col>
        <Col sm={10}>
          <FormControl     name="confir_new_password"     type="password"
placeholder="Confir        New        Password"        onChange={onChange}
value={user.confir_new_password}/>
          <HelpBlock>For     your     security,     repeat     your     new
password.</HelpBlock>
          <FormControl.Feedback />
        </Col>
      </FormGroup>


      <FormGroup                                        controlId="password"
validationState={errors.password=="" ? null : errors.password}>
        <Col componentClass={ControlLabel} sm={2}>
          Current Password
```

```
        </Col>
        <Col sm={10}>
          <FormControl          name="password"          type="password"
placeholder="Current   Password"   required="true"   onChange={onChange}
value={user.password}/>
          <HelpBlock>For    your    security,    enter    your    current
password</HelpBlock>
          <FormControl.Feedback />
        </Col>
      </FormGroup>

      <FormGroup>
        <Col smOffset={2} sm={10}>
          <Button type="submit">
            Change profile
          </Button>
        </Col>
      </FormGroup>
    </Form>
  </div>
);

//Haciendo uso de propTypes se comprueban que todos los parámetros son
recibidos correctamente
ProfileForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.object.isRequired,
  user: PropTypes.object.isRequired
};

export default ProfileForm;
```

Código A-17. Contenedor client/src/containers/BasePage.jsx

```
import React, { Component } from 'react';
import Base from '../components/Base.jsx'
/*
Clase BasePage encargada de crear un componente Base con un compenten
'children' recibido mediante reac-router
*/
```

```
class BasePage extends Component {
  constructor() {
    super();
  }
  render(){
    return <Base children={this.props.children}/>
  }
}
export default BasePage;
```

Código A-18. Contenedor client/src/containers/CreateLicensePage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import CreateLicenseForm from '../components/CreateLicenseForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';



class CreateLicensePage extends Component {


  /**
   * Clase constructora.
   */
  constructor(props, context) {
    super(props, context);
    //Configuración global de las notificaciones toast
    toastr.options={
      "closeButton": true, //Dispondrán de un botón para cerrarlas
      "preventDuplicates": true, //Para prevenir que aparezcan toast
duplicados
      "newestOnTop": true //Los nuevos aparecerán encima
    }
    // Configuramos los estados iniciales
    this.state = {
      errors: {
        duration: '',
        limit_bytes: '',
         sensors: {
        }
```

```
      },
      license: {
        duration: 1,
        limit_bytes: '0',
        sensors: {
        },
        OrganizationId: this.props.params.OrgId,
        UserId: this.props.params.UserId
      },
      sensors: ""
    };

    this.changeSensors = this.changeSensors.bind(this);
    this.processForm = this.processForm.bind(this);
    this.changeLicense = this.changeLicense.bind(this);
  }

  //Justo antes de renderizar el componente se llamará a este método
  componentWillMount(){
    //Utilizando ajax, pedimos los tipos de sensores disponibles para la organización que queremos crear
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión get
    xhr.open('get',         '/api/licenses/new?OrganizationId='         + this.props.params.OrgId);
    // Configuramos el token para la autorización
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    // La respuesta se espera que sea un JSON
    xhr.responseType = 'json';
    // Añadimos el callback para cuando se reciba la respuesta de forma correcta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito
        // Cambiamos el estado, eliminando los errores y almancenamos los sensores
        //Una vez sabemos qué sensores hay, configuramos sus valores iniciales a 0
        xhr.response.sensors.split(';').map(sensor => {
```

```
          this.state.license.sensors[sensor.split(",")[1]] = '0';
        });
        this.setState({
          error: "",
          sensors: xhr.response.sensors
        });
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      }else {
        // En caso de fallo, mediante un toast informamos del mensaje de
error
          {xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    //Enviamos la petición
    xhr.send();
  }


  /**
   * Función encargada de enviar el formulario
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  processForm(event) {
    //Prevenimos el envío del formulario vacío y por defecto
    event.preventDefault();
    //Creamos una cadena de carácteres par enviar en el método post los
parámetros introducidos en el formulario
    const duration = encodeURIComponent(this.state.license.duration);
    const                          limit_bytes                          =
encodeURIComponent(this.state.license.limit_bytes);
    const                        OrganizationId                         =
encodeURIComponent(this.state.license.OrganizationId);
    const UserId = encodeURIComponent(this.state.license.UserId);
    const sensors = JSON.stringify(this.state.license.sensors);
    const                            formData                           =
`sensors=${sensors}&duration=${duration}&UserId=${UserId}&limit_bytes=${l
imit_bytes}&OrganizationId=${OrganizationId}`;
```

```
    //Creación de la petición AJAX para la creación de un usuario
    const xhr = new XMLHttpRequest();
    //Abrimos la conexión post con el servidor
    xhr.open('post', '/api/licenses' );
    //Modificamos la cabecera para indicar que será el envío de un formulario
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    //Configuramos el token que identifica al usuario que está realizando la petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    //Esperaremos una respuesta JSON
    xhr.responseType = 'json';
    //Función que se ejecutará al recibir la respuesta
   xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Si se ha recibido un 200 ok, notificamos con un mensaje que se ha creado correctamente el usuario
        {xhr.response.message && toastr.success(xhr.response.message)}
//Contendrá el mensaje recibido
        // Cambiamos los valores de los errores en el estado del componente para indicar que no hay errores
        this.setState({
          errors: {
            duration: '',
            limit_bytes: '',
            sensors: {
            }
          }
        });
        //Redirigimos al inicio
        this.context.router.replace('/');
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de fallo mostramos el mensaje de error recibido del servidor
        {xhr.response.message && toastr.error(xhr.response.message)}
      }
```

```
    });
    //Enviamos la petición
    xhr.send(formData);
  }

  changeSensors(event){
    const license = this.state.license;

    license.sensors[event.target.name]   =   parseInt(event.target.value,
10);
    this.setState({
      license
    })

  }
  /**
   * Funcion encargada del cambio de algún campo del formulario y en el
objeto 'user' del estado.
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  changeLicense(event) {
    // Obtenemos el valor actual del usuario almacenado en el estado
    const license = this.state.license;
    const field = event.target.name;
    if(event.target.name=="duration"){
      //Auntamos los meses de duración de la licencia
      license.duration=parseInt(event.target.value, 10);
    }
    else{
      license[field] = event.target.value;
    }
      //finalmente almacenamos el nuevo estado del usuario
    this.setState({
        license
      });
  }

  /**
```

```
   * Instanciamos un componente de tipo CreateLicenseForm al que se le
pasan los parámetros correctos
   * definidos en el contendor CreateLicensePage
   */
  render() {
    return (
      <CreateLicenseForm
        onSubmit={this.processForm}
        onChange={this.changeLicense}
        onChangeSensors={this.changeSensors}
        errors={this.state.errors}
        license={this.state.license}
        sensors={this.state.sensors.split(';')}
      />
    );
  }

}
//Comprobamos que se está haciendo uso de react-router
CreateLicensePage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default CreateLicensePage;
```

Código A-19. Contenedor client/src/containers/CreateOrgPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import CreateOrgForm from '../components/CreateOrgForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';


class CreateOrgPage extends Component {

  /**
   * Clase constructora.
   */
```

```
  constructor(props, context) {
    super(props, context);
    //Configuración global de las notificaciones toast
    toastr.options={
      "closeButton": true, //Dispondrán de un botón para cerrarlas
      "preventDuplicates": true, //Para prevenir que aparezcan toast duplicados
      "newestOnTop": true //Los nuevos aparecerán encima
    }
    // Configuramos los valores iniciales para los estados
    this.state = {
      //inicialmente no hay errores
      errors: {
        name:'',
        email:'',
        cluster_id:'',
        sensors: ''
      },
      //Inicialmente no existe ninguna organización
      //Los sensores por defecto para cualquier organización serán IPS, Flow y Social.
      org: {
        email: '',
        name: '',
        cluster_id: '',
        sensors: 'IPS,event;Flow,flow;Social,social'
      }
    }
    //Utilización de bind para poder llamar a estas funciones dentro de la propia clase CreateOrgPage
    this.processForm = this.processForm.bind(this);
    this.changeOrg = this.changeOrg.bind(this);
  }


  /**
   * Función encargada de enviar el formulario
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
```

```
  processForm(event) {
    // Prevenimos el envío del formulario vacío
    event.preventDefault();
    // Creamos una cadena con los valores a enviar en el método post. Se codificaran para que se puedan enviar carácteres especiales(como ñ, espacios, @...)
    const cluster_id = encodeURIComponent(this.state.org.cluster_id);
    const name = encodeURIComponent(this.state.org.name);
    const email = encodeURIComponent(this.state.org.email);
    const sensors = encodeURIComponent(this.state.org.sensors);
    const formData = `name=${name}&email=${email}&cluster_id=${cluster_id}&sensors=${sensors}`;

    // Creación de la peticion AJAX correspondiente a la creación de una organización
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión post
    xhr.open('post', '/api/organizations');
    //Configuramos la cabecera (content-type) para indicar que es una petición de tipo formulario
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    // Configuramos el toquen en la cabecera de la petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    // Esperamos un JSON como respuesta a la petición
    xhr.responseType = 'json';
    // Añadimos el callBack que se ejecutará cuando recibamos la respuesta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito
        this.setState({
          errors:{} //Eliminamos los posibles errores que hubiera en el estado almacenado
        });
        /*
          Lanzamos una notificación mediante un toast para notificar que se ha creado una organización con éxito.
          Este toast mostrará el mensaje recibido en la respuesta
        */
```

```
        {xhr.response.message && toastr.success(xhr.response.message)}
        //Redirigimos al listado de organizaciones
        this.context.router.replace('/listOrgs');
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        /*
          En caso de fallo
          Lanzamos una notificación mediante un toast para notificar que no se ha podido crear la organización.
          Este toast mostrará el mensaje recibido en la respuesta
        */
        {xhr.response.message && toastr.error(xhr.response.message)}
      }
    });
    xhr.send(formData);
  }


  /**
   * Funcion encargada del cambio de algún campo del formulario y en el objeto 'org' del estado.
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  changeOrg(event) {
    //El objeto de la organización a cambiar será el nombre del campo del formulario que se está cambiando
    const field = event.target.name;
    const org = this.state.org;
    org[field] = event.target.value;
    //Cambiamos el estado
    this.setState({
        org
      });


    //Esto se utiliza para la validación visual del formulario en la interfaz gráfica
    if(this.state.org.name.length!=0)  //El campo name no puede estar vacío
```

```
      this.state.errors.name="success";
    else
      this.state.errors.name="error"


    if(this.state.org.cluster_id.length!=0)  //El  campo  cluster_id  no
puede estar vacío
      this.state.errors.cluster_id="success";
    else
      this.state.errors.cluster_id="error"


    if(this.state.org.email.length!=0) //El campo email no puede estar
vacío
      this.state.errors.email="success";
    else
      this.state.errors.email="error"
  }


  /**
   * Instanciamos un componente CreateOrgForm al que se le pasarán las
referencias a
   * las funciones que manejan el envío y cambio del formulario y los
objetos 'errors' y 'org'.
   */
  render() {
    return (
      <CreateOrgForm
        onSubmit={this.processForm}
        onChange={this.changeOrg}
        errors={this.state.errors}
        org={this.state.org}
        />
    );
  }
}


//Verificamos que está el contexto de react-router
CreateOrgPage.contextTypes = {
  router: PropTypes.object.isRequired
};
```

```
export default CreateOrgPage;
```

Código A-20. Contenedor client/src/containers/CreateUserPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import CreateUserForm from '../components/CreateUserForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';



class CreateUserPage extends Component {


  /**
   * Clase constructora.
   */
  constructor(props, context) {
    super(props, context);
    //Configuración global de las notificaciones toast
    toastr.options={
      "closeButton": true, //Dispondrán de un botón para cerrarlas
      "preventDuplicates": true, //Para prevenir que aparezcan toast
duplicados
      "newestOnTop": true //Los nuevos aparecerán encima
    }
    // Configuramos los estados iniciales
    this.state = {
      errors: {
        email: '',
        password: '',
        organization: '',
        confir_password: '',
        name: ''
      },
      user: {
        email: '',
        password: '',
        organization: '',
        confir_password: '',
```

```
        name: '',
        admin: false //Inicialmente el usuario no será administrador
      },
      organizations: []
    };
    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }

  //Justo antes de renderizar el componente se llamará a este método
  componentWillMount(){
    //Utilizando ajax, pedimos la lista de todas las organizaciones registradas
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión get
    xhr.open('get', '/api/users/new');
    // Configuramos el token para la autorización
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    // La respuesta se espera que sea un JSON
    xhr.responseType = 'json';
    // Añadimos el callback para cuando se reciba la respuesta de forma correcta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito
        // Cambiamos el estado, eliminando los errores y almancenando las organizaciones
        this.setState({
          error: "",
          organizations: xhr.response.orgs
        });

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de fallo, mediante un toast informamos del mensaje de error
          {xhr.response.message && toastr.error(xhr.response.message)}
```

```
        }
    });
    //Enviamos la petición
    xhr.send();
  }


  /**
   * Función encargada de enviar el formulario
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  processForm(event) {
    //Prevenimos el envío del formulario vacío y por defecto
    event.preventDefault();
    //Creamos una cadena de carácteres par enviar en el método post los parámetros introducidos en el formulario
    const name = encodeURIComponent(this.state.user.name);
    const email = encodeURIComponent(this.state.user.email);
    const password = encodeURIComponent(this.state.user.password);
    const confir_password = encodeURIComponent(this.state.user.confir_password);
    const organization = encodeURIComponent(this.state.user.organization);
    const role = this.state.user.admin ? "admin" : "normal";
    const formData = `role=${role}&name=${name}&organization=${organization}&email=${email}&password=${password}&confir_password=${confir_password}`;


    //Creación de la petición AJAX para la creación de un usuario
    const xhr = new XMLHttpRequest();
    //Abrimos la conexión post con el servidor
    xhr.open('post', '/api/users');
    //Modificamos la cabecera para indicar que será el envío de un formulario
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    //Configuramos el token que identifica al usuario que está realizando la petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    //Esperaremos una respuesta JSON
    xhr.responseType = 'json';
```

```
    //Función que se ejecutará al recibir la respuesta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Si se ha recibido un 200 ok, notificamos con un mensaje que se
ha creado correctamente el usuario
        {xhr.response.message  &&  toastr.success(xhr.response.message)}
//Contendrá el mensaje recibido
        // Cambiamos los valores de los errores en el estado del
componente para indicar que no hay errores
        this.setState({
          errors: {}
        });
        //Redirigimos al listado de usuarios
        this.context.router.replace('/listUsers/all/all');
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de fallo mostramos el mensaje de error recibido del
servidor
        {xhr.response.message && toastr.error(xhr.response.message)}
        // Cambiamos los errores por los errores recibidos
        const errors = xhr.response.errors ? xhr.response.errors : {};
        // El mensaje de respuesta recibido será un resumen de los
errores que se han producido
        errors.summary = xhr.response.message;
        // Cambiamos el estado de los errores
        this.setState({
          errors
        });
      }
    });
    //Enviamos la petición
    xhr.send(formData);


  }


  /**
   * Funcion encargada del cambio de algún campo del formulario y en el
objeto 'user' del estado.
```

```
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  changeUser(event) {
    // Tomamos nombre del campo del formulario que se está editando
    const field = event.target.name;
    // Obtenemos el valor actual del usuario almacenado en el estado
    const user = this.state.user;
    //Compronamos si estamos modificando el rol o no
    if(field!="role")
    {
      //En caso de no estar modificando el rol, almacenamos el valor
modificado
      user[field] = event.target.value;


    }else{
      //Si se ha cambiado el rol, cambiamos el valor de user.admin de
true a false o viceversa
      user.admin = !user.admin;
    }
    //finalmente almacenamos el nuevo estado del usuario
    this.setState({
        user
      });


    //Esto es para validar el formulario visualmente, solo para el
usuario
    if(this.state.user.password   !==   this.state.user.confir_password){
//La nueva contraseña y la confirmacion han de coincidir
      this.state.errors.password="error";
      this.state.errors.confir_password="error";
    }
    else{
      this.state.errors.password="success";
      this.state.errors.confir_password="success";
    }
    // El tamaño de la contraseña debe estar entre 8 y 15
    if(this.state.user.password.length          <          8          ||
this.state.user.password.length > 15)
      this.state.errors.password="error";
```

```
    if(this.state.user.confir_password.length            <8            ||
this.state.user.confir_password.length > 15)
      this.state.errors.confir_password="error";

    //El campo nombre no puede estar vacío
    if(this.state.user.name.length!=0)
      this.state.errors.name="success";
    else
      this.state.errors.name="error"

    //El campo email no puede estar vacío
    if(this.state.user.email.length!=0)
      this.state.errors.email="success";
    else
      this.state.errors.email="error"
  }

  /**
   * Instanciamos un componente de tipo CreateUserForm al que se le pasan
los parámetros correctos
   * definidos en el contendor CreateUserPage
   */
  render() {
    return (
      <CreateUserForm
        onSubmit={this.processForm}
        onChange={this.changeUser}
        errors={this.state.errors}
        user={this.state.user}
        organizations={this.state.organizations}
      />
    );
  }

}
//Comprobamos que se está haciendo uso de react-router
CreateUserPage.contextTypes = {
  router: PropTypes.object.isRequired
};
```

```
export default CreateUserPage;
```

Código A-21. Contenedor client/src/containers/DashboardPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import Dashboard from '../components/Dashboard.jsx';


class DashboardPage extends Component {

  /**
   * Clase constructora.
   */
  constructor(props) {
    super(props);
    //Obtenemos el nombre y el email almacenados de forma local
    const name = localStorage.getItem('userProfileName');
    const email = localStorage.getItem('userProfileEmail');
  }
  /**
   * Instanciamos un componente de tipo Dashboard.
   */
  render() {
    return (<Dashboard/>);
  }

}
export default DashboardPage;
```

Código A-22. Contenedor client/src/containers/EditOrgPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import EditOrgForm from '../components/EditOrgForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';


class EditOrgPage extends Component {
```

```
  /**
   * Clase constructora.
   */
  constructor(props, context) {
    super(props, context);
    //Configuración global de las notificaciones toast
    toastr.options={
      "closeButton": true, //Dispondrán de un botón para cerrarlas
      "preventDuplicates": true, //Para prevenir que aparezcan toast duplicados
      "newestOnTop": true //Los nuevos aparecerán encima
    }
    //Configuramos el estado inicial
    this.state = {
      errors: {
        name: '',
        email: '',
        cluster_id: ''
      },
      organization: {
        name: '',
        email: '',
        cluster_id: ''
      }
    };
    //Se hace uso de 'bind' para poder llamar a estas funciones desde la propia clase EdtiOrgPage
    this.processForm = this.processForm.bind(this);
    this.changeOrg = this.changeOrg.bind(this);
  }


   //Justo antes de renderizar el componente se llama a este método
  componentWillMount(){
    //Utilizando ajax, pedimos los valores de la organización a editar
    //Creamos la petición AJAX
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión get que realizará una peticion a la url /api/organizations/:idOrg/edit
```

```
    xhr.open('get',   '/api/organizations/'   +   this.props.params.id   +
"/edit");
    //Configuramos el token para identificar al usuario que realizará la
petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    //La respuesta esperada será un JSON
    xhr.responseType = 'json';
    //Configuramos la función que se ejecutará al recibir la respuesta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito
        // Cambiamos el estado
        this.setState({
          error: "", //Eliminamos los posibles errores que existesen
          organization: { //Configuramos los datos de la organización a
editar
            name: xhr.response.org.name,
            email: xhr.response.org.email,
            cluster_id: xhr.response.org.cluster_id
          }
       });
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de error
        // Cambiamos el componente de error
        error = xhr.response.message;
        //mostramos el mensaje recibido con un toast de error
        {error && toastr.success(error)}
        this.setState({
          error
        });
        }
    });
    //Enviamos la petición
    xhr.send();
  }
```

```
  /**
   * Función encargada de enviar el formulario
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  processForm(event) {
    // Prevenimos el envío del formulario vacío o con datos por defecto
    event.preventDefault();


    // Creamos la cadena a enviar en el método post con los datos de la
organización introducidos en el formulario
    const name  = encodeURIComponent(this.state.organization.name);
    const email = encodeURIComponent(this.state.organization.email);
    const                          cluster_id                          =
encodeURIComponent(this.state.organization.cluster_id);
    const                           formData                           =
`email=${email}&name=${name}&cluster_id=${cluster_id}`;
    // Creamos la petición AJAX
    const xhr = new XMLHttpRequest();
    //Abrimos una petición PUT para editar una organización
    xhr.open('put', '/api/organizations/' + this.props.params.id);
    //Configuramos la cabecera para indicar que estamos procesando un
formulario
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    //Configuramos el toknen para identificar al usuario que realizará la
petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    //Esperaremos un JSON de respuesta a la petición
    xhr.responseType = 'json';
    //Añadimos la función que se ejecutará al recibir la respuesta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito, mostramos un mensaje de éxito con la
respuesta recibida del servidor
        {xhr.response.message && toastr.success(xhr.response.message)}
        // Cambiamos el estado del componente
        this.setState({
          errors: {},
          organization: xhr.response.org
```

```
        });
        //Redirigimos al inicio
        this.context.router.replace('/');
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // En caso de error
        // Cambiamos el estado del componente
        const errors = xhr.response.errors ? xhr.response.errors : {};
        // Notificamos con un toast de error el mensaje recibido del
servidor
        {xhr.response.message && toastr.error(xhr.response.message)}
        //Modificamos el estado del componente
        this.setState({
          successMessage:"",
          errors
        });
      }
    });
    //Enviamos el formulario al servidor
    xhr.send(formData);
  }


  /**
   * Funcion encargada del cambio de algún campo del formulario y en el
objeto 'user' del estado.
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  changeOrg(event) {
    //Obtenemos el nombre del campo que va a cambiar
    const field = event.target.name;
    //Obtenemos el valor actual de la organización
    const organization = this.state.organization;
    //Cambiamos el valor del campo por el nuevo valor
    organization[field] = event.target.value;
    //Cambiamos el estado en el componente
    this.setState({
```

```
        organization
      });



    //Esto es para validar el formulario visualmente, solo para el
usuario
    //El nombre no puede estar vacío
    if(this.state.organization.name.length!=0)
      this.state.errors.name="success";
    else
      this.state.errors.name="error"

    //El identificador del cluster no puede estar vacío
    if(this.state.organization.cluster_id.length!=0)
      this.state.errors.cluster_id="success";
    else
      this.state.errors.cluster_id="error"

    //El email no puede estar vacío
    if(this.state.organization.email.length!=0)
      this.state.errors.email="success";
    else
      this.state.errors.email="error"



  }


  /**
   * Instanciamos un componente de tipo EditOrgForm, pasándole los
parámetros correctos definidos en el componente EditOrgPage
   */
  render() {
    return (
      <EditOrgForm
        onSubmit={this.processForm}
        onChange={this.changeOrg}
        errors={this.state.errors}
        organization={this.state.organization}
      />
```

```
    );
  }

}

//Nos aseguramos que estamos usando react-router
EditOrgPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default EditOrgPage;
```

Código A-23. Contenedor client/src/containers/EditUserPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import EditUserForm from '../components/EditUserForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';


class EditUserPage extends Component {

  /**
   * Class constructor.
   */
  constructor(props, context) {
    super(props, context);
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    // set the initial component state
    this.state = {
      errors: {
        name: '',
        email: ''
      },
      user: {
```

```
        name: '',
        email: '',
        organization: 'No',
        admin: false
      },
      organizations: []
    };

    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }

   //Justo antes de renderizar el componente se llama a este método
  componentWillMount(){
     //Utilizando ajax, en el constructor pedimos la lista de
organizaciones registradas
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/users/' + this.props.params.id + "/edit");
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          organizations: xhr.response.orgs,
          user: {
            name: xhr.response.user.name,
            email: xhr.response.user.email,
            organization: xhr.response.user.OrganizationId || "No" //Si
no tiene organización se pondrá "No"
          }
       });
      } else {
        // failure
        // change the component state
```

```
        error = xhr.response.message;
        {error && toastr.success(error)}
        this.setState({
          error
        });
        }
    });
    xhr.send();
  }



  /**
   * Process the form.
   *
   * @param {object} event - the JavaScript event object
   */
  processForm(event) {
    // prevent default action. in this case, action is the form submission event
    event.preventDefault();
    // create a string for an HTTP body message
    const name  = encodeURIComponent(this.state.user.name);
    const email = encodeURIComponent(this.state.user.email);
    const role = this.state.user.admin ? "admin" : "normal";
    const organization = this.state.user.organization;
    const formData = `email=${email}&name=${name}&role=${role}&organization=${organization}`;
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('put', '/api/users/' + this.props.params.id);
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
```

```
        {xhr.response.message && toastr.success(xhr.response.message)}
        // change the component-container state
        this.setState({
          errors: {},
        });
        //Redirigimos al inicio
        this.context.router.replace('/');
      } else {
        // failure
        // change the component state
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          successMessage:"",
          errors
        });
      }
    });
    xhr.send(formData);
}

/**
 * Change the user object.
 *
 * @param {object} event - the JavaScript event object
 */
changeUser(event) {
    const field = event.target.name;
    const user = this.state.user;
    if(field!="role")
    {
      user[field] = event.target.value;

    }else{
      user.admin = !user.admin;
    }
    this.setState({
        user
```

```
      });

    //Esto es para validar el formulario visualmente, solo para el
usuario

    if(this.state.user.name.length!=0)
      this.state.errors.name="success";
    else
      this.state.errors.name="error"

    if(this.state.user.email.length!=0)
      this.state.errors.email="success";
    else
      this.state.errors.email="error"



  }

  /**
   * Render the component.
   */
  render() {
    return (
      <EditUserForm
        onSubmit={this.processForm}
        onChange={this.changeUser}
        errors={this.state.errors}
        user={this.state.user}
        organizations={this.state.organizations}
      />
    );
  }

}

EditUserPage.contextTypes = {
  router: PropTypes.object.isRequired
};
```

```
export default EditUserPage;
```

Código A-24. Contenedor client/src/containers/ExtendLicensePage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import ExtendLicenseForm from '../components/ExtendLicenseForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';



class ExtendLicensePage extends Component {


  /**
   * Clase constructora.
   */
  constructor(props, context) {
    super(props, context);
    //Configuración global de las notificaciones toast
    toastr.options={
      "closeButton": true, //Dispondrán de un botón para cerrarlas
      "preventDuplicates": true, //Para prevenir que aparezcan toast
duplicados
      "newestOnTop": true //Los nuevos aparecerán encima
    }
    // Configuramos los estados iniciales
    this.state = {
      license: {
        duration: '1', //Inicialmente la extensión es de 1 mes
        expires_at: '',
        limit_bytes: '',
        sensors: {
        },
        OrganizationId: '',
        UserId: ''
      },
    };


    this.changeDuration = this.changeDuration.bind(this);
```

```
    this.processForm = this.processForm.bind(this);
  }


  //Justo antes de renderizar el componente se llamará a este método
  componentWillMount(){
    //Utilizando ajax, pedimos los tipos de sensores disponibles para la
organización que queremos crear
    const xhr = new XMLHttpRequest();
    //Abrimos una conexión get
    xhr.open('get',          '/api/licenses/extend?LicenseId='          +
this.props.params.LicenseId );
    // Configuramos el token para la autorización
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    // La respuesta se espera que sea un JSON
    xhr.responseType = 'json';
    // Añadimos el callback para cuando se reciba la respuesta de forma
correcta
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // En caso de éxito
        // Cambiamos el estado, eliminando los errores y almancenando la
licencia
        this.setState({
          error: "",
          license: xhr.response.license
        });


      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      }else {
        // En caso de fallo, mediante un toast informamos del mensaje de
error
        {xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    //Enviamos la petición
    xhr.send();
  }
```

```
  /**
   * Función encargada de enviar el formulario
   *
   * @parametros {objeto} event - Objeto event de JavaScript
   */
  processForm(event) {
    //Prevenimos el envío del formulario vacío y por defecto
    event.preventDefault();
    //Justo antes de enviar los datos le sumamos a la fecha de expiracion los meses añadidos. Si la licencia a expirado, le sumamos la ampliación a partir del día de hoy
    const new_expires = new Date(this.state.license.expires_at) < new Date() ? new Date() : new Date(this.state.license.expires_at);
    new_expires.setMonth(new_expires.getMonth() + parseInt(this.state.license.duration));
    console.log(new_expires);
    //Creamos una cadena de carácteres par enviar en el método post todos los parámetros (aunque solo cambia la fecha de expiración)
    const duration = encodeURIComponent(-1); //Si el tiempo de extensión es -1 significa que es una extensión de licencia
    const expires_at = encodeURIComponent(new_expires);
    const license_uuid = encodeURIComponent(this.state.license.license_uuid);
    const limit_bytes = encodeURIComponent(this.state.license.limit_bytes);
    const OrganizationId = encodeURIComponent(this.state.license.OrganizationId);
    const UserId = encodeURIComponent(this.state.license.UserId);
    const sensors = JSON.stringify(this.state.license.sensors);
    const formData = `expires_at=${expires_at}&license_uuid=${license_uuid}&sensors=${sensors}&duration=${duration}&UserId=${UserId}&limit_bytes=${limit_bytes}&OrganizationId=${OrganizationId}`;

    //Creación de la petición AJAX para la creación de un usuario
    const xhr = new XMLHttpRequest();
    //Abrimos la conexión post con el servidor
    xhr.open('post', '/api/licenses' );
    //Modificamos la cabecera para indicar que será el envío de un formulario
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
```

```
    //Configuramos el token que identifica al usuario que está realizando
la petición
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    //Esperaremos una respuesta JSON
    xhr.responseType = 'json';
    //Función que se ejecutará al recibir la respuesta
   xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Si se ha recibido un 200 ok, notificamos con un mensaje que se
ha creado correctamente el usuario
        {xhr.response.message   &&   toastr.success(xhr.response.message)}
//Contendrá el mensaje recibido
        //Redirigimos al inicio
        this.context.router.replace('/');
      } else {
        // En caso de fallo mostramos el mensaje de error recibido del
servidor
        {xhr.response.message && toastr.error(xhr.response.message)}
      }
    });
    //Enviamos la petición
    xhr.send(formData);
  }


  changeDuration(event){
    // Obtenemos el valor actual del usuario almacenado en el estado
    const license = this.state.license;
    //modificamos el campo de expiración
    if(event.target.name=="duration"){
      license.duration = event.target.value;
      //Manejamos la fecha para sumarle el numero de meses recibido
      this.setState({
        license
      });
    } //En caso contrario no hacemos nada
  }


  /**
   * Instanciamos un componente de tipo ExtendLicenseForm al que se le
pasan los parámetros correctos
```

```
   * definidos en el contendor ExtendLicensePage
   */
  render() {
    return (
      <ExtendLicenseForm
        onSubmit={this.processForm}
        onChange={this.changeDuration}
        license={this.state.license}
      />
    );
  }

}
//Comprobamos que se está haciendo uso de react-router
ExtendLicensePage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default ExtendLicensePage;
```

Código A-25. Contenedor client/src/containers/ForgotPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import ForgotForm from '../components/ForgotForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';

class ForgotPage extends Component {

  /**
   * Class constructor.
   */
  constructor(props, context) {
    super(props, context);
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
```

```
    }
    // set the initial component state
    this.state = {
      errors: {
        email: '',
      },
      user: {
        email: '',
      }
    };

    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }

  /**
   * Process the form.
   *
   * @param {object} event - the JavaScript event object
   */
  processForm(event) {
    // Prevención del envío por defecto
    event.preventDefault();

    // Cadena que contiene el email codificado en URI
    const email = encodeURIComponent(this.state.user.email);
    const formData = `email=${email}`;

    // Creación de la petición AJAX
    const xhr = new XMLHttpRequest();
    xhr.open('post', '/auth/forgot');
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
     if (xhr.status === 200) {
        // En caso de éxito se borra el estado
        this.setState({
          errors: {}
```

```
      });
      //Notificamos que se ha hecho correctamente
      {xhr.response.message && toastr.success(xhr.response.message)}
      // Volvemos a la url raiz /
      this.context.router.replace('/');
    } else {
      // Cambiamos los errores y los notificamos
      const errors = xhr.response.errors ? xhr.response.errors : {};
      {xhr.response.message && toastr.error(xhr.response.message)}

      this.setState({
        errors
      });
    }
  });
  xhr.send(formData);

}

/**
 * Change the user object.
 *
 * @param {object} event - the JavaScript event object
 */
changeUser(event) {
  const field = event.target.name;
  const user = this.state.user;
  user[field] = event.target.value;
  this.setState({
    user
  });
}

/**
 * Render the component.
 */
render() {
  return (
```

```
      <ForgotForm
        onSubmit={this.processForm}
        onChange={this.changeUser}
        errors={this.state.errors}
        user={this.state.user}
      />
    );
  }

}

ForgotPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default ForgotPage;
```

Código A-26. Contenedor client/src/containers/HomePage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import Home from '../components/Home.jsx';

class HomePage extends Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (<Home/>);
  }
}

export default HomePage;
```

Código A-27. Contenedor client/src/containers/ListLicensesPage.jsx

```
import React, { Component } from 'react';
import {Pagination} from 'react-bootstrap';
import ListLicenses from '../components/ListLicenses.jsx'
import Auth from '../modules/Auth';
```

```
import { Link } from "react-router";
import toastr from 'toastr';
import FileSaver from 'file-saver';
import PropTypes  from 'prop-types';



class ListLicensesPage extends Component {
  constructor() {
    super();
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    this.state={

                                                                    lice
nses: [

                                                                    ],
      activePage: 1,
      number_licenses: '',
      orgName: ''
    }

    this.activateFormat=this.activateFormat.bind(this);
    this.expiresFormat=this.expiresFormat.bind(this);
    this.extendFormat=this.extendFormat.bind(this);
    this.downloadFormat=this.downloadFormat.bind(this);
    this.sensorsFormat=this.sensorsFormat.bind(this);
    this.handleSelectPage=this.handleSelectPage.bind(this);
  }

  //Justo antes de renderizar el componente se llama a este método
  componentWillMount(){
    this.state.orgName=this.props.params.orgName;

                                                                    this
.loadLicenses(this.state.activePage, this.props.params.id);
  }
```

```
  loadLicenses(page, id){
    //Utilizando ajax, en el constructor pedimos la lista de licencias
para esa organización
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/organizations/' + id + '/licenses?page=' +
page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          licenses: xhr.response.licenses,
          number_licenses: xhr.response.number_licenses,
        });
      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // failure
        {
          xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    xhr.send();
  }

  sensorsFormat(cell, row){
    const sensores = JSON.parse(JSON.parse(cell));
    let lista = [];

    for(const sensor in sensores){
          lista.push(<li key={sensor}>{sensor}: {sensores[sensor]}</li>);
    }
```

```
    return (<div>
        <ul>
        {lista}
        </ul>
    </div>);
  }

  extendFormat(cell, row){
    if(row.enabled){
      return (
        <Link to={"/extendLicense/" + cell }
        className="glyphicon glyphicon-plus"
        style={{color:"green"}}>
        </Link>
        );
    }
    else{
      return (<div style={{color:"green"}} >Pending</div>)
    }
  }

  activateFormat(cell, row){
    if(!row.enabled && Auth.isAdmin()){
      return (
        <div style={{color:"green"}}>
          Inactivated <br></br>
          <Link onClick={() => {
              const xhr = new XMLHttpRequest();
              xhr.open('PUT', "/api/licenses/activate/" + row.id);
              //Configuramos el token que identifica al usuario que está
realizando la petición
              xhr.setRequestHeader('Authorization',                `bearer
${Auth.getToken()}`);
              xhr.overrideMimeType('text/plain; charset=x-user-defined');
              xhr.addEventListener('load', () => {
                if (xhr.status === 200) {
                  // Si se ha recibido un 200 ok notificamos con un toast
                  toastr.success("License  "  +  row.id  +  "  activated
correctly")
```

```
                  //Redirigimos al inicio
                  this.context.router.replace("/");
                }else{
                  // En caso de fallo mostramos el mensaje de error
recibido del servidor
                  {xhr.response.message                                    &&
toastr.error(xhr.response.message)}
                }
              });
              xhr.send();
            }}
          className="glyphicon glyphicon-check"
          style={{color:"green"}}>
          </Link>
        </div>
        );
    }
    else{
      if(row.enabled)
        return (<div style={{color:"green"}} >Activated</div>)
      else
        return (<div style={{color:"red"}} >Inactivated</div>)
    }
  }

  downloadFormat(cell, row){
    if(row.enabled){
      return (<div>
            <Link style={{color:"green"}}
            onClick={() => {
              const xhr = new XMLHttpRequest();
              xhr.open('GET',    "/api/licenses/download?LicenseId="    +
cell);
              //Configuramos el token que identifica al usuario que está
realizando la petición
              xhr.setRequestHeader('Authorization',               `bearer
${Auth.getToken()}`);
              xhr.overrideMimeType('text/plain; charset=x-user-defined');
              xhr.addEventListener('load', () => {
                if (xhr.status === 200) {
```

```
                  // Si se ha recibido un 200 ok, notificamos con un
mensaje que se ha creado correctamente el usuario
                  toastr.success("Download file...");
                  const   blob   =   new   Blob([xhr.response],   {type:
"text/plain;charset=utf-8"});
                  FileSaver.saveAs(blob, row.license_uuid + ".lic");


                } else if (xhr.status === 404){
                  //Si obtenemos un error 404 es que esa licencia no
existe, lo notificamos con un toast de error
                  toastr.error("Error. <br></br> License not found!")
                }else{
                  // En caso de fallo mostramos el mensaje de error
recibido del servidor
                  {xhr.response.message                                &&
toastr.error(xhr.response.message)}
                }
              });
              xhr.send();
            }}
            className="glyphicon glyphicon-download-alt"></Link>
        </div>);
    }
    else{
      return (<div style={{color:"green"}} >Pending</div>)
    }
  }


  expiresFormat(cell, row){
    if(row.enabled){
      const expires_period_days = Math.round((new Date(row.expires_at) -
new Date())/(24*60*60*1000)); //horas*minutos*segundos*milisegundos de un
dia
      if(expires_period_days<0)
        return          (          <div          style={{'color':"red",
'fontWeight':"bold"}}>¡¡Expires {-expires_period_days} days ago!!</div>)
      else if(expires_period_days<7)
        return  (  <div  style={{color:"red"}}>{expires_period_days}  days
remaining</div>)
      else if (expires_period_days<15)
        return ( <div style={{color:"orange"}}>{expires_period_days} days
```

```
remaining</div>)
      else
        return  ( <div  style={{color:"blue"}}>{expires_period_days}  days
remaining</div>)
    }
    else{
        if(row.duration<0){
          return (<div style={{color:"green"}}>{"Extension"}</div>)
        }
        else{
          return    (<div    style={{color:"green"}}>{"Duration:    "    +
row.duration + " months"}</div>)
        }
    }
  }
  //Manejador para seleccionar la pagina a visualizar
  handleSelectPage(eventKey) {
    this.loadLicenses(eventKey, this.props.params.id)
    this.setState({
      activePage: eventKey,
    });
  }



  render(){
    return (
      <div className="container">
        <div>
          <ListLicenses                        orgName={this.state.orgName}
extendFormat={this.extendFormat}     activateFormat={this.activateFormat}
downloadFormat={this.downloadFormat}   expiresFormat={this.expiresFormat}
licenses={this.state.licenses}               orgId={this.props.params.id}
sensorsFormat={this.sensorsFormat}/>
        </div>
        {
        this.state.number_licenses > 10 ?
        <div className="text-center">
          <Pagination
            first
            last
            next
```

```
            prev
            ellipsis
            boundaryLinks
            bsSize="medium"
            items={Math.ceil(this.state.number_licenses/10)}         //10
usuarios por página. Redondeamos para arriba
            maxButtons={5}
            activePage={this.state.activePage}
            onSelect={this.handleSelectPage} />
        </div>
        : null
      }
      </div>
        )
  }
}
//Comprobamos que se está haciendo uso de react-router
ListLicensesPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default ListLicensesPage;
```

Código A-28. Contenedor client/src/containers/ListOrgsPage.jsx

```
import React, { Component } from 'react';
import {Pagination} from 'react-bootstrap';
import ListOrgs from '../components/ListOrgs.jsx'
import Auth from '../modules/Auth';
import { Link } from 'react-router';
import toastr from 'toastr';
import PropTypes  from 'prop-types';


class ListOrgsPage extends Component {
  constructor() {
    super();
    toastr.options={
      "closeButton": true,
```

```
      "preventDuplicates": true,
      "newestOnTop": true
    }
    this.state={
      activePage: 1,

                                                                  orga
nizations: [
                                                                  ],
      number_orgs: ''
    }
    this.handleSelectPage=this.handleSelectPage.bind(this);
    //Obtenemos el mensaje si hemos eliminado una organizacion
correctamente, lo notificamos y eliminamos
    localStorage.getItem('successRemoveOrg')                        &&
toastr.success(localStorage.getItem('successRemoveOrg'))            &&
localStorage.removeItem('successRemoveOrg')
  }
  //Justo antes de renderizar el componente se llama a este método
  componentWillMount(){

                                                                  this
.loadOrgs(this.state.activePage);
    this.state.organizations
  }


  loadOrgs(page){
     //Utilizando ajax, en el constructor pedimos la lista de
organizations registrados
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/organizations?page=' + page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          organizations: xhr.response.orgs,
```

```
        number_orgs: xhr.response.number_orgs
      });
    } else if(xhr.status === 404){
      //No authorizated deauthenticateUser
      this.context.router.replace('/logout');
    } else {
      // failure
      {xhr.response.message && toastr.error(xhr.response.message)}
      }
  });
  xhr.send();
}


//Manejadores de la tabla

listLicensesFormat(cell, row){
    return (<div>
         <Link to={"/listLicenses/" +
        row.id + "/" + encodeURIComponent(row.name)}
        className="glyphicon glyphicon-folder-open"
        style={{color:"green"}} ></Link>
       </div>);
}

editOrgFormat(cell, row){
    return (<div>
         <Link to={"/organization/edit/" +
        row.id }
        className="glyphicon glyphicon-edit"
        style={{color:"green"}} ></Link>
       </div>);
}

removeOrgFormat(cell, row){
    return (<div>
          <Link style={{color:"red"}}
          onClick={() => {
```

```
              if(confirm('Are you sure to remove the organization ' +
row.name + " (" + row.email + "). This will remove ALL licenses of this
organizations"  )){
                //Utilizando ajax, en el constructor pedimos la lista de
usuarios registrados
                const xhr = new XMLHttpRequest();
                xhr.open('delete', '/api/organizations/' + row.id);
                // set the authorization HTTP header
                xhr.setRequestHeader('Authorization',             `bearer
${Auth.getToken()}`);
                xhr.responseType = 'json';
                xhr.addEventListener('load', () => {
                  if (xhr.status === 200) {
                    //Almacenamos el mensaje de respuesta
                    localStorage.setItem('successRemoveOrg',
xhr.response.message);
                    //Recargamos la página para que recargue la lista de
usuarios
                    window.location.reload();
                  } else if(xhr.status === 404){
                      //No authorizated deauthenticateUser
                      this.context.router.replace('/logout');
                  } else {
                    // failure
                    {xhr.response.message                                &&
toastr.error(xhr.response.message)}
                    }
                });
                xhr.send();
              }
            }}
            className="glyphicon glyphicon-remove"></Link>
        </div>);
  }


  countUsersFormat(cell,row){
    return (<Link style={{color:"blue"}} to={"/listUsers/" + row.id + "/"
+ encodeURIComponent(row.name) }>{row.Users.length}</Link>);
  }


  //Manejador para seleccionar la pagina a visualizar
```

```
  handleSelectPage(eventKey) {
    this.loadOrgs(eventKey)
    this.setState({
      activePage: eventKey,
    });
  }


  render(){
    return (
      <div className="container">
        <div>
          <ListOrgs              organizations={this.state.organizations}
removeOrgFormat={this.removeOrgFormat} editOrgFormat={this.editOrgFormat}
countUsersFormat={this.countUsersFormat}
listLicensesFormat={this.listLicensesFormat}/>
        </div>
        {
          this.state.number_orgs > 10 ?
        <div className="text-center">
          <Pagination
            first
            last
            next
            prev
            ellipsis
            boundaryLinks
            bsSize="medium"
            items={Math.ceil(this.state.number_orgs/10)}             //10
organizations por página. Redondeamos para arriba
            maxButtons={5}
            activePage={this.state.activePage}
            onSelect={this.handleSelectPage} />
        </div>
        : null
      }
      </div>
        )
  }
}
```

```
//Comprobamos que se está haciendo uso de react-router
ListOrgsPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default ListOrgsPage;
```

Código A-29. Contenedor client/src/containers/ListUsersPage.jsx

```
import React, { Component } from 'react';
import {Pagination} from 'react-bootstrap';
import ListUsers from '../components/ListUsers.jsx'
import Auth from '../modules/Auth';
import { Link } from "react-router";
import toastr from 'toastr';
import PropTypes  from 'prop-types';



class ListUsersPage extends Component {
  constructor() {
    super();
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    this.state={
      users: [
      ],
      activePage: 1,
      number_users: '',
      orgName: ''
    }



    this.handleSelectPage=this.handleSelectPage.bind(this);
    //Obtenemos el mensaje si hemos eliminado un usuario correctamente,
lo notificamos y eliminamos
    localStorage.getItem('successRemoveUser')                          &&
toastr.success(localStorage.getItem('successRemoveUser'))             &&
localStorage.removeItem('successRemoveUser')
```

```
  }

  //Justo antes de renderizar el componente se llama a este método
  componentWillMount(){
    this.state.orgName=this.props.params.orgName;
    this.loadUsers(this.state.activePage, this.props.params.id);
  }

  loadUsers(page, id){
    if(id=="all"){
     //Utilizando ajax, en el constructor pedimos la lista de usuarios
registrados
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/users?page=' + page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          users: xhr.response.users,
          number_users: xhr.response.number_users
        });

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // failure
        {
          xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    xhr.send();
```

```
    }
    else if(id=="null"){
    //Utilizando ajax, en el constructor pedimos la lista de usuarios
registrados
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/organizations/null/users?page=' + page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // success
        // change the component-container state
        this.setState({
          error: "",
          users: xhr.response.users,
          number_users: xhr.response.number_users,
        });

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      } else {
        // failure
        {
          xhr.response.message && toastr.error(xhr.response.message)}
        }
    });
    xhr.send();
    }else
    {
    //Utilizando ajax, en el constructor pedimos la lista de usuarios
registrados
    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('get', '/api/organizations/' + id + '/users?page=' + page);
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
```

```
  xhr.responseType = 'json';
  xhr.addEventListener('load', () => {
    if (xhr.status === 200) {
      // success
      // change the component-container state
      this.setState({
        error: "",
        users: xhr.response.users,
        number_users: xhr.response.number_users,
      });

    } else if(xhr.status === 404){
      //No authorizated deauthenticateUser
      this.context.router.replace('/logout');
    } else {
      // failure
      {
        xhr.response.message && toastr.error(xhr.response.message)}
      }
  });
  xhr.send();
  }
}

 //Manejadores de la tabla

editUserFormat(cell, row){
    return (<div>
        {
          row.id!=localStorage.getItem('userProfileId')
        ?
          <Link to={"/user/edit/" +
          row.id}
          className="glyphicon glyphicon-edit"
          style={{color:"green"}} ></Link>
        :
        <span style={{color:"blue"}}
        className="glyphicon glyphicon-user">
```

```
          You</span>
          }
         </div>);
  }

  removeUserFormat(cell, row){
      return (<div>
            <Link style={{color:"red"}}
            onClick={() => {
              if(confirm('Are you sure to remove the user ' + row.name +
" (" + row.email + ")" )){
                // create an AJAX request
                const xhr = new XMLHttpRequest();
                xhr.open('delete', '/api/users/' + row.id);
                // set the authorization HTTP header
                xhr.setRequestHeader('Authorization',             `bearer
${Auth.getToken()}`);
                xhr.responseType = 'json';
                xhr.addEventListener('load', () => {
                  if (xhr.status === 200) {
                    //Almacenamos el mensaje de respuesta
                    localStorage.setItem('successRemoveUser',
xhr.response.message);
                    //Recargamos la página para que recargue la lista de
usuarios
                    window.location.reload();
                  } else if(xhr.status === 404){
                    //No authorizated deauthenticateUser
                    this.context.router.replace('/logout');
                  } else {
                    // failure
                    {xhr.response.message                                &&
toastr.error(xhr.response.message)}
                    }
                });
                xhr.send();
              }
            }}
            className="glyphicon glyphicon-remove"></Link>
         </div>);
```

```
  }



  //Manejador para seleccionar la pagina a visualizar
  handleSelectPage(eventKey) {
    this.loadUsers(eventKey, this.props.params.id)
    this.setState({
      activePage: eventKey,
    });
  }



  render(){
    return (
      <div className="container">
        <div>
          <ListUsers                          orgName={this.state.orgName}
users={this.state.users}             editUserFormat={this.editUserFormat}
removeUserFormat={this.removeUserFormat}/>
        </div>
        {
        this.state.number_users > 10 ?
        <div className="text-center">
          <Pagination
            first
            last
            next
            prev
            ellipsis
            boundaryLinks
            bsSize="medium"
            items={Math.ceil(this.state.number_users/10)}  //10  usuarios
por página. Redondeamos para arriba
            maxButtons={5}
            activePage={this.state.activePage}
            onSelect={this.handleSelectPage} />
        </div>
        : null
      }
```

```
      </div>
        )
  }
}

//Comprobamos que se está haciendo uso de react-router
ListUsersPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default ListUsersPage;
```

Código A-30. Contenedor client/src/containers/LoginPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import LoginForm from '../components/LoginForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';

class LoginPage extends Component {

  /**
   * Constructor de la clase.
   */
  constructor(props, context) {
    super(props, context);
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    // set the initial component state
    this.state = {
      errors: {
        email: '',
        password: ''
      },
      user: {
        email: '',
```

```
        password: ''
      }
    };

    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }

  /**
   * Función que procesa el formulario.
   *
   * @param {object} event - Evento JavaScript
   */
  processForm(event) {
    // Previene que se envíen valores por defecto
    event.preventDefault();

    // Crea la cadena con los valores a enviar al servidor
    const email = encodeURIComponent(this.state.user.email);
    const password = encodeURIComponent(this.state.user.password);
    const formData = `email=${email}&password=${password}`;

    // Creación de la petición AJAX
    const xhr = new XMLHttpRequest();
    xhr.open('post', '/auth/login');
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Todo correcto
        this.setState({
          errors: {}
        });
        {xhr.response.message && toastr.success(xhr.response.message)}
        // Almacenamos en local los datos de un usuario
        localStorage.setItem('userProfileId', xhr.response.user.id);
        localStorage.setItem('userProfileOrg',
xhr.response.user.OrganizationId);
```

```
        localStorage.setItem('userProfileName', xhr.response.user.name);
        localStorage.setItem('userProfileEmail',
xhr.response.user.email);
        localStorage.setItem('userProfileRole', xhr.response.user.role);

        // Almacenamos el token
        Auth.authenticateUser(xhr.response.token);



        // Cambiamos la url a la página de inicio
        this.context.router.replace('/');
      } else {
        // Si hay error lo notificamos
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);

  }

  /**
   * Cambia el objeto user
   *
   * @param {object} event - Evento JavaScript
   */
  changeUser(event) {
    const field = event.target.name;
    const user = this.state.user;
    user[field] = event.target.value;
    this.setState({
      user
    });

    //Esto es para validar el formulario visualmente, solo para el
usuario
```

```
    if(event.target.name=="password"   &&   user[field].length   <   8   ||
user[field].length > 15)
      this.state.errors.password="error";
    else
      this.state.errors.password="success"


  }

  /**
   * Render the component.
   */
  render() {
    return (
      <LoginForm
        onSubmit={this.processForm}
        onChange={this.changeUser}
        errors={this.state.errors}
        user={this.state.user}
      />
    );
  }

}

LoginPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default LoginPage;
```

Código A-31. Contenedor client/src/containers/NewPasswordPage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import NewPasswordForm from '../components/NewPasswordForm.jsx';
import PropTypes  from 'prop-types';
```

```
import toastr from 'toastr';

class NewPasswordPage extends Component {

  /**
   * Class constructor.
   */
  constructor(props, context) {
    super(props, context);
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    // set the initial component state
    this.state = {
      errors: {
        password: '',
        confir_password: ''
      },
      user: {
        password: '',
        confir_password: ''
      }
    };

    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }

  /**
   * Process the form.
   *
   * @param {object} event - the JavaScript event object
   */
  processForm(event) {
    // Prevención del envío por defecto
    event.preventDefault();
    // Cadena que contiene el email codificado en URI
```

```
    const confir_password = encodeURIComponent(this.state.user.confir_password);
    const password = encodeURIComponent(this.state.user.password);
    const formData = `password=${password}&confir_password=${confir_password}`;
    // Creación de la petición
    const xhr = new XMLHttpRequest();
    //El token se le pasa como parametros en la URL
    xhr.open('post', '/auth/reset/' + this.props.params.token );
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        //Borramos los posibles errores
        this.setState({
          errors: {}
        });
        //Notificamos con un toast que todo ha ido bien
        {xhr.response.message && toastr.success(xhr.response.message)}
        //Volvemos a la página principal
        this.context.router.replace('/');
      } else {
        //En caso de fallo cambiamos el objeto errors y lo notificamos con un toast
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);
  }


  /**
   * Change the user object.
   *
   * @param {object} event - the JavaScript event object
```

```
   */
  changeUser(event) {
    const field = event.target.name;
    const user = this.state.user;
    user[field] = event.target.value;
    this.setState({
      user
    });



    //Esto es para validar el formulario visualmente, solo para el
usuario
    if(this.state.user.password !== this.state.user.confir_password){
      this.state.errors.password="error";
      this.state.errors.confir_password="error";
    }
    else{
      this.state.errors.password="success";
      this.state.errors.confir_password="success";
    }
    if(this.state.user.password.length          <          8          ||
this.state.user.password.length > 15)
      this.state.errors.password="error";
    if(this.state.user.confir_password.length           <8           ||
this.state.user.confir_password.length > 15)
      this.state.errors.confir_password="error";
  }


  /**
   * Render the component.
   */
  render() {
    return (
      <NewPasswordForm


        onSubmit={this.processForm}
        onChange={this.changeUser}
        errors={this.state.errors}
        user={this.state.user}
```

```
      />
    );
  }
}
NewPasswordPage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default NewPasswordPage;
```

Código A-32. Contenedor client/src/containers/ProfilePage.jsx

```
import React, { Component } from 'react';
import Auth from '../modules/Auth';
import ProfileForm from '../components/ProfileForm.jsx';
import PropTypes  from 'prop-types';
import toastr from 'toastr';

class ProfilePage extends Component {

  /**
   * Class constructor.
   */
  constructor(props, context) {
    super(props, context);
    toastr.options={
      "closeButton": true,
      "preventDuplicates": true,
      "newestOnTop": true
    }
    const name = localStorage.getItem('userProfileName');
    const email = localStorage.getItem('userProfileEmail')

    // set the initial component state
    this.state = {
      errors: {
        name: '',
        new_password: '',
        confir_new_password: '',
```

```
        email: '',
        password: ''
      },
      user: {
        name: name,
        new_password: '',
        confir_new_password: '',
        email: email,
        password: ''
      }
    };

    this.processForm = this.processForm.bind(this);
    this.changeUser = this.changeUser.bind(this);
  }

  /**
   * Process the form.
   *
   * @param {object} event - the JavaScript event object
   */
  processForm(event) {
    // prevent default action. in this case, action is the form submission event
    event.preventDefault();
    // create a string for an HTTP body message
    const name  = encodeURIComponent(this.state.user.name);
    const email = encodeURIComponent(this.state.user.email);
    const new_password = encodeURIComponent(this.state.user.new_password);
    const confir_new_password = encodeURIComponent(this.state.user.confir_new_password);
    const password = encodeURIComponent(this.state.user.password);
    const formData = `email=${email}&password=${password}&name=${name}&new_password=${new_password}&confir_new_password=${confir_new_password}`;

    // create an AJAX request
    const xhr = new XMLHttpRequest();
    xhr.open('post', '/api/changeProfile');
```

```
    xhr.setRequestHeader('Content-type',         'application/x-www-form-
urlencoded');
    // set the authorization HTTP header
    xhr.setRequestHeader('Authorization', `bearer ${Auth.getToken()}`);
    xhr.responseType = 'json';
    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        // Si ha ido bien borramos los errores
        this.setState({
          errors: {}
        });

        // Cambiamos el valor por el de los nuevos items recibidos.
        localStorage.setItem('userProfileName', xhr.response.user.name);
        localStorage.setItem('userProfileEmail',
xhr.response.user.email);
        //Notificamos el éxito con un toast
        {xhr.response.message && toastr.success(xhr.response.message)}

        // Cambiamos la url a la raiz /
        this.context.router.replace('/');

      } else if(xhr.status === 404){
        //No authorizated deauthenticateUser
        this.context.router.replace('/logout');
      }
       else {

        // En caso de fallo lo notificamos con un toast y modificamos el
objeto errors
        const errors = xhr.response.errors ? xhr.response.errors : {};
        {xhr.response.message && toastr.error(xhr.response.message)}
        this.setState({
          errors
        });
      }
    });
    xhr.send(formData);
```

```
  }

  /**
   * Change the user object.
   *
   * @param {object} event - the JavaScript event object
   */
  changeUser(event) {
    const field = event.target.name;
    const user = this.state.user;
    user[field] = event.target.value;
    this.setState({
      user
    });



    //Esto es para validar el formulario visualmente, solo para el
usuario
    if(this.state.user.new_password                                    ===
this.state.user.confir_new_password
      && this.state.user.new_password.length > 8
      && this.state.user.new_password.length < 15
      && this.state.user.confir_new_password.length > 8
      && this.state.user.confir_new_password.length < 15
      ||            (this.state.user.new_password.length==0            &&
this.state.user.confir_new_password.length == 0)){
      this.state.errors.new_password="success";
      this.state.errors.confir_new_password="success";
    }
    else{
      this.state.errors.new_password="error";
      this.state.errors.confir_new_password="error";
    }


    if(this.state.user.password.length          <          8          ||
this.state.user.password.length > 15 || this.state.user.password.length==
0 )
      this.state.errors.password="error";
    else
      this.state.errors.password="success"
```

```
    if(this.state.user.name.length!=0)
      this.state.errors.name="success";
    else
      this.state.errors.name="error"


  }

  /**
   * Render the component.
   */
  render() {
    return (
      <ProfileForm
        onSubmit={this.processForm}
        onChange={this.changeUser}
        errors={this.state.errors}
        user={this.state.user}
      />
    );
  }

}

ProfilePage.contextTypes = {
  router: PropTypes.object.isRequired
};

export default ProfilePage;
```

Código A-33. Módulo de ayuda client/src/modules/Auth.js

```
class Auth {

  /**
   * Authenticate a user. Save a token string in Local Storage
   *
   * @param {string} token
```

```
   */
  static authenticateUser(token) {
    localStorage.setItem('token', token);
  }

  /**
   * Check if a user is authenticated - check if a token is saved in
Local Storage
   *
   * @returns {boolean}
   */
  static isUserAuthenticated() {
    return localStorage.getItem('token') !== null;
  }

  /**
   * Deauthenticate a user. Remove a token from Local Storage.
   *
   */
  static deauthenticateUser() {
    localStorage.removeItem('token');
  }

  /**
   * Get a token value.
   *
   * @returns {string}
   */

  static getToken() {
    return localStorage.getItem('token');
  }

  /**
   * Check if a user is admin.
   *
   * @returns {true}
   */
```

```
  static isAdmin() {
    return  localStorage.getItem('userProfileRole')=="admin"  ?  true  :
false;
  }


   /**
   * Check if a user belong to a organization.
   *
   * @returns {true}
   */

  static hasOrganization() {
    return localStorage.getItem('userProfileOrg')=="null" ? false : true;
  }
}

export default Auth;
```

Código A-34. Componente principal client/src/app.jsx

```
import React from 'react';
import { hashHistory, Router } from 'react-router';
import routes from './routes.js';



class App extends React.Component{
    render(){
        return(
            <Router history={hashHistory} routes={routes} />
        );
    }
}

export default App;
```

Código A-35. Fichero de entrada del cliente client/src/index.jsx

```
import { render } from 'react-dom';
import App from './app.jsx';
import injectTapEventPlugin from 'react-tap-event-plugin';
```

```
injectTapEventPlugin();

render(
  <App />,
  document.getElementById('root')
);
```

Código A-36. Fichero de rutas para react-router client/src/routes.js

```
import BasePage from './containers/BasePage.jsx';
import HomePage from './containers/HomePage.jsx';
import DashboardPage from './containers/DashboardPage.jsx';
import LoginPage from './containers/LoginPage.jsx';
import ForgotPage from './containers/ForgotPage.jsx';
import NewPasswordPage from './containers/NewPasswordPage.jsx';
import ProfilePage from './containers/ProfilePage.jsx';
import CreateUserPage from './containers/CreateUserPage.jsx';
import CreateOrgPage from './containers/CreateOrgPage.jsx';
import CreateLicensePage from './containers/CreateLicensePage.jsx';
import ListUsersPage from './containers/ListUsersPage.jsx';
import EditUserPage from './containers/EditUserPage.jsx';
import EditOrgPage from './containers/EditOrgPage.jsx';
import ListOrgsPage from './containers/ListOrgsPage.jsx';
import ListLicensesPage from './containers/ListLicensesPage.jsx';
import ExtendLicensePage from './containers/ExtendLicensePage.jsx';
import Auth from './modules/Auth';

const routes = {
  // base component (wrapper for the whole application).
  component: BasePage,
  childRoutes: [

    {
      path: '/',
      getComponent: (location, callback) => {
        if (Auth.isUserAuthenticated()) {
          callback(null, DashboardPage);
        } else {
          callback(null, HomePage);
        }
```

```
    }
  },

  {
    path: '/login',
    component: LoginPage
  },

  {
    path: '/logout',
    onEnter: (nextState, replace) => {
      Auth.deauthenticateUser();

      // change the current URL to /
      replace('/');
    }
  },
  {
    path: '/forgot',
    component: ForgotPage
  },
  {
    path: '/reset/:token',
    component: NewPasswordPage
  },
  {
    path: '/user/edit',
    component: ProfilePage
  },
  {
    path: '/user/new',
    component: CreateUserPage
  },
  {
    path: '/organization/new',
    component: CreateOrgPage
  },
  {
```

```
      path: '/license/new/:UserId/:OrgId',
      component: CreateLicensePage
    },
    {
      path: '/listUsers/:id/:orgName',
      component: ListUsersPage
    },
    {
      path: '/listLicenses/:id/:orgName',
      component: ListLicensesPage
    },
    {
      path: '/user/edit/:id',
      component: EditUserPage
    },
    {
      path: '/organization/edit/:id',
      component: EditOrgPage
    },
    {
      path: '/listOrgs',
      component: ListOrgsPage
    },
    {
      path: '/extendLicense/:LicenseId',
      component: ExtendLicensePage
    }
  ]
};

export default routes;
```

ANEXO B
CÓDIGO DE BACK-END

Código B-1. Fichero de entrada server/index.js

```
//En primer lugar ocultamos los console.logs para producción.
process.env.NODE_ENV=="production" ? console.log = function  () {} : null

const express = require('express');
const passport = require('passport');
const bodyParser = require('body-parser');
const app = express();

//Inicializamos sequelize
const connectDB = require('./db').connectDB;

//Lanzamos la conexión a la base de datos mediante la funcion connectDB
connectDB();

// Decimos donde están los ficheros estáticos
app.use(express.static('./server/static/'));
app.use(express.static('./client/dist/'));



app.use(bodyParser.urlencoded({ extended: false }));

// Configuramos passport para la autenticación. Esto será un middleware
app.use(passport.initialize());
// Cargamos las estrategias que usará passport para el inicio de sesión y
para la creación de usuarios
const localSignupStrategy = require('./passport/local-signup');
const localLoginStrategy = require('./passport/local-login');
passport.use('local-signup', localSignupStrategy);
passport.use('local-login', localLoginStrategy);



// Este middleware comprueba si un usuario está autenticado
```

```
const authCheckMiddleware = require('./middleware/auth-check');
app.use('/api', authCheckMiddleware);

// Importamos todas las rutas
const authRoutes = require('./routes/auth');
const apiRoutes = require('./routes/api');
app.use('/auth', authRoutes);
app.use('/api', apiRoutes);

// Finalmente, arrancamos express en el puerto 3000
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000 or http://127.0.0.1:3000');
                                                                    });

module.exports = app;
```

Código B-2. Página html estática devuelta por el servidor server/static/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Licensing management of RedBorder</title>
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon"/>
    <link rel="stylesheet" href="css/style.css"/>
    <link                                                rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.min.css">
    <link                                                rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap.min.css">
    <link                                                rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap-theme.min.css">
    <link   rel="stylesheet"   href="https://npmcdn.com/react-bootstrap-table/dist/react-bootstrap-table-all.min.css">
  </head>
  <body>
    <div id="root"></div>

    <script src="/js/app.js"></script>
```

```
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"><
/script>
    <!-- Latest compiled and minified JavaScript -->
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.min.j
s"></script>
  </body>
</html>
```

Código B-3. Fichero para la inicialización de la base de datos server/db/index.js

```
const Sequelize = require('sequelize')
const config = require('../../config/config.json')
const MODE_RUN = process.env.MODE_RUN || "development"
const DB_config = config[MODE_RUN]

const      sequelize      =      new      Sequelize(process.env.DB_NAME
|| DB_config.database,
                              process.env.DB_USER || DB_config.user,
                              process.env.DB_PASSWORD
|| DB_config.password,
        //Options
        {
        dialect: process.env.DB_SERVER || DB_config.BD,
        host: process.env.DB_HOST || DB_config.host,
        port:   process.env.DB_PORT || DB_config.port,
        logging: (process.env.DB_LOG=="true" || DB_config.log =="true") ?
console.log : false
});

//Cargamos los diferentes modelos
const models = require('../models')(sequelize);

const connectDB = () => {
    if(process.env.MODE_RUN == "test"){
        sequelize.sync({force:true}).then( () => {
```

```
            console.log("Connected to DB");
        }, (err) => {
            console.log("Error connecting DB, retrying...");
            setTimeout(connectDB, 5000);
        })
    }
    else{
        sequelize.sync().then(() => {
            console.log("Connected to DB");
            //If there aren't users, create one admin user by default in
production mode...
            models.User.findAll({where: {
                role: "admin"
            }})
            .then((users) => {
                if(users.length == 0){
                    const NewUser = models.User.build({
                      name: "Admin",
                      email: "admin@redborder.com",
                      password: "adminadmin",
                      role: "admin"
                    })
                NewUser.save().then(() => {
                    console.log("New default admin user created.");
                    console.log("   Email: admin@redborder.com");
                    console.log("   Password: adminadmin");
                    console.log("Please, change this user profile");
                });
              }
            })
        }, (err) => {
            //Sequelize error
            console.log("Error connecting DB, retrying...")
            setTimeout(connectDB, 5000);
        });
    }
}


module.exports.sequelize=sequelize;
```

```
module.exports.connectDB=connectDB;
```

Código B-4. Fichero de entrada para la creación de los modelos server/models/index.js

```
module.exports = function (sequelize){
    //Importamos todos los modelos, hacemos las relaciones entre ellos y los devolvemos
    const User = require('./user')(sequelize);
    const Organization = require('./organization')(sequelize);
    const License = require('./license')(sequelize);
    //Un usuario pertenece a una organización
    User.belongsTo(Organization);
    //Una oganización tiene muchos usuarios o ninguno en caso de que sea null la clave externa
    Organization.hasMany(User);
    //Una licencia es creada por un usuario ("pertenece" a un usuario).
    //Si el usuario se borra, se pierde la referencia al usuario poniendose el campo UserId a null
    License.belongsTo(User);
    //Una licencia pertenece a una organizacion, y no puede no pertener a ninguna (no puede ser null)
    //Si una organización se borra, se borrarán todas las licencias asociadas a ella
    Organization.hasMany(License, {foreignKey: {allowNull: false}, onDelete: 'Cascade'});
    return {
        User: User,
        Organization: Organization,
        License: License
    };
};
```

Código B-5. Fichero del modelo *Users* server/models/user.js

```
const passwordHash = require('password-hash');
const DataTypes = require('sequelize/lib/data-types');

module.exports = function(sequelize) {
  const User = sequelize.define('User',
     {
       id:{
           type : DataTypes.UUID,
```

```
            primaryKey: true,
            defaultValue: DataTypes.UUIDV4,
            validate: {
                isUUID: 4
            }
        },
        name: {
            type: DataTypes.STRING,
            aallowNull: false,
            validate: {
                notEmpty: { msg: "Field name shouldn't be empty" }
            }
        },
        email: {
            type: DataTypes.STRING,
            allowNull: false,
            unique: true,
            validate: {
                isEmail: { msg: "Email should be a correct email" },
                notEmpty: {msg: "Field email shouldn't be empty'"},
            }
        },
        hashed_password: {
            type: DataTypes.STRING,
            validate: {
                notEmpty: {msg: "Field password shouldn't be empty"},
                not: {args: ["wrong password"], msg: "Format password is
incorrect. Password should be between 8 and 15 alphanumeric characters"}
            }
        },
        role: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                isIn: {
                    args: [['normal', 'admin']],
                    msg: "Rol user must be normal or admin"
                    }
            }
```

```
        },
        resetPasswordToken: {
            type: DataTypes.STRING,
            allowNull: true
        },

        resetPasswordExpires:{
            type: DataTypes.DATE,
            allowNull: true
        }

    },
    //Expansion of the model user
    {
        setterMethods   : {
            password: function(password){
                if(password.length > 15 || password.length < 8 || typeof
password != "string"){
                    this.setDataValue('hashed_password',
"wrong_password"); //Lanza error si la contraseña mide lo que no debe
                }
                else{
                    this.setDataValue('hashed_password',
passwordHash.generate(password));
                }
            },
            email: function(email){
                this.setDataValue('email', email.toLowerCase());
            }

        },
        getterMethods : {
            password: function(){
                return this.hashed_password;
            }
        },
        instanceMethods: {
            verifyPassword: function(password){
                return                      passwordHash.verify(password,
```

```
this.hashed_password);
            },
            changePassword: function(password, new_password){
                if(passwordHash.verify(password, this.hashed_password)){
                    this.setDataValue('hashed_password',
passwordHash.generate(new_password));
                    return true;
                }
                else
                    return false;
            }
        }
    });
    return User;
}
```

Código B-6. Fichero del modelo *Organizations* server/models/organization.js

```
const DataTypes = require('sequelize/lib/data-types');

module.exports = function(sequelize) {
  const Organization = sequelize.define('Organization',
      {
        id:{
            type : DataTypes.UUID,
            primaryKey: true,
            defaultValue: DataTypes.UUIDV4,
            validate: {
                isUUID: 4
            }
        },
        cluster_id: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                notEmpty: { msg: "Field cluster id shouldn't be empty" }
            }
        },
        name: {
            type: DataTypes.STRING,
```

```
            allowNull: false,
            validate: {
                notEmpty: { msg: "Field name shouldn't be empty" }
            }
        },
        email: {
            type: DataTypes.STRING,
            allowNull: false,
            unique: true,
            validate: {
                isEmail: { msg: "Email should be a correct email" },
                notEmpty: {msg: "Field email shouldn't be empty'"},
            }
        },
        sensors: {
            type: DataTypes.STRING, //Lista de sensores separados por ;
            allowNull: false,
            validate: {
                notEmpty: {msg: "Field sensors shouldn't be empty'"},
            }
        }

    },
    //Expansion of the model user
    {
        setterMethods   : {
            email: function(email){
                this.setDataValue('email', email.toLowerCase());
            }
        }
    });
    return Organization;
}
```

Código B-7. Fichero del modelo *Licenses* server/models/license.js

```
const DataTypes = require('sequelize/lib/data-types');

module.exports = function(sequelize) {
```

```
const License = sequelize.define('Licenses',
    {
      id:{
          type : DataTypes.UUID,
          primaryKey: true,
          defaultValue: DataTypes.UUIDV4,
          validate: {
              isUUID: 4
          }
      },
      license_uuid : {
          type : DataTypes.UUID,
          allowNull: false,
          defaultValue: DataTypes.UUIDV4,
          validate : {
              notEmpty: {
                  msg: "Field licenses uuid shouldn't be empty"
              }
          }
      },
      duration: {
          type: DataTypes.INTEGER,
          allowNull: false,
          validate: {
              notEmpty: {
                  msg: "Field duration shouldn't be empty"
              }
          }
      },
      expires_at: {
          type: DataTypes.DATE,
      },
      sensors : {
          type: DataTypes.JSON,
          allowNull: false,
          validate:{
              notEmpty: {msg: "Field sensors shouldn't be empty"}
          }
      },
```

```
        limit_bytes : {
            type: DataTypes.INTEGER,
            allowNull: false,
            validate : {
                notEmpty: {
                    msg: "Field limit bytes shouldn't be empty"
                }
            }
        },
        enabled : {
            type: DataTypes.BOOLEAN,
            allowNull: false,
            defaultValue: false,
            validate : {
                notEmpty: {
                    msg: "Field enabled shouldn't be empty"
                }
            }
        }
    });
    return License;
}
```

Código B-8. Fichero para la estrategia de incio de sesión de *Passport* server/passport/local-login.js

```
const jwt = require('jsonwebtoken');
const PassportLocalStrategy = require('passport-local').Strategy;
const config_json = require('../../config/config.json');
const MODE_RUN = process.env.MODE_RUN || "development"
const config = config_json[MODE_RUN]

//Incializamos sequelize
const sequelize = require('../db').sequelize;

//Cargamos los modelos
const models = require('../models')(sequelize);

module.exports = new PassportLocalStrategy({
  usernameField: 'email',
```

```
  passwordField: 'password',
  session: false,
  passReqToCallback: true
}, (req, email, password, done) => {
  const userData = {
    email: email.trim().toLowerCase(),
    password: password.trim()
  };
  return models.User.findOne({where:{
    email: userData.email}})
  .then(function(Found_User, err){
        if(err || !Found_User){
            const error = new Error('Incorrect email or password');
            error.name = 'IncorrectCredentialsError';
            return done(error);
          }
        else
          {
            //Comprobamos si la contraseña es correcta
            if(Found_User.verifyPassword(password)){
              const payload ={
                sub: Found_User.id
              };
              // Creamos el token
              const token = jwt.sign(payload, process.env.JWT_SECRET ||
config.jwtSecret); //Algoritmo por defecto HS256
              const data = {
                name: Found_User.name,
                role: Found_User.role,
                id: Found_User.id,
                OrganizationId: Found_User.OrganizationId
              };
              return done(null, token, data);
            }
            else{
              const error = new Error('Incorrect email or password');
              error.name = 'IncorrectCredentialsError';
              return done(error);
            }
```

```
        }
      }, function(err){
            return done(err);
      });
});
```

Código B-9. Fichero para la estrategia de registro de *Passport* server/passport/local-signup.js

```
const PassportLocalStrategy = require('passport-local').Strategy;

//Incializamos sequelize
const sequelize = require('../db').sequelize;

//Cargamos los modelos
const models = require('../models')(sequelize);

/**
 * Return the Passport Local Strategy object.
 */
module.exports = new PassportLocalStrategy({
  usernameField: 'email', //Por defecto busca username, pero mi usuario
será el email
  passwordField: 'password', //La contraseña será el password
  session: false,
  passReqToCallback: true //Para poder leer el nombre del body
}, (req, email, password, done) => {
  const NewUser = models.User.build({
      name: req.body.name.trim(),
      email: email.trim(),
      password: password.trim(),
      OrganizationId:            (req.body.organization==''            ||
req.body.organization=="No") ? null : req.body.organization,
      role: req.body.role
    });
  NewUser.save().then(function(NewUser) {
      return done(null);
    }, function(err){
      return done(err);
  });
```

```
});
```

Código B-10. Fichero con el *middleware* para acceder a las rutas *api* server/middleware/auth-check.js

```
const jwt = require('jsonwebtoken');
const config_json = require('../../config/config.json');

const MODE_RUN = process.env.MODE_RUN || "development"
const config = config_json[MODE_RUN]

//Incializamos sequelize
const sequelize = require('../db').sequelize;

//Cargamos los modelos
const models = require('../models')(sequelize);



module.exports = (req, res, next) => {
  if (!req.headers.authorization) {
    return res.status(404).end(); //Por seguridad enviamos el 404 para que el usuario no sepa que existe tal página
  }
  // Obtenemos el token de la cabecera del mensaje
  const token = req.headers.authorization.split(' ')[1];


  // Decodificamos el token haciendo uso de la clave secreta
  return jwt.verify(token, process.env.JWT_SECRET || config.jwtSecret, (err, decoded) => {
    // Si ha habido error, significa que el token no es correcto y se devuelve un 401 (no autorizado)
    if (err) { return res.status(401).end(); }
    //En caso de no haber error se obtiene el identificador del usuario del token
    const userId = decoded.sub;
    //Se añade el identificador del usuario a la petición (para usarse en los siguientes "middleware")
    req.userId = userId;
    // Confirmamos si el usuario existe en la base de datos (puede ser un token válido pero haberse borrado de la base de datos)
    return models.User.findOne({where: {id: userId}})
    .then(function(User){
      if(!User) //Si el usuario no existe, no le estará permitido su
```

```
acceso y pensará que no existe la página
        return res.status(404).end();
      else
        return next();
      }, function(err){
        return res.status(404).end();
      })
    });
}
```

Código B-11. Fichero con las rutas *auth* server/routes/auth.js

```
const express = require('express');
const passport = require('passport');
const router = new express.Router();
const nodemailer = require('nodemailer');
const mockTransport = require('nodemailer-mock-transport');
const async = require('async');
const crypto = require('crypto');
const MODE_RUN = process.env.MODE_RUN || "development"
const email = require('../../config/config.json')[MODE_RUN].email;

//Incializamos sequelize
const sequelize = require('../db').sequelize;

//Cargamos los modelos
const models = require('../models')(sequelize);

//Configuramos el envío de emails
const transportMock = mockTransport({ //Configuramos el mock para el envío de correos
              service: process.env.EMAIL_SERVER || email.server,
              auth: {
                user: process.env.EMAIL_USER || email.email,
                pass: process.env.EMAIL_PASSWORD || email.password
              }
            });
const smtpTransport = MODE_RUN=="test" ? //Si estamos en modo test utilizamos el mock
            nodemailer.createTransport(transportMock)
```

```
            :
            nodemailer.createTransport({
              service: process.env.EMAIL_SERVER || email.server,
              auth: {
                user: process.env.EMAIL_USER || email.email,
                pass: process.env.EMAIL_PASSWORD || email.password
              }
            });


/**
 * Validate the NewPassword form
 *
 * @param {object} payload - the HTTP body message
 * @returns {object} The result of validation. Object contains a boolean validation result,
 *                       errors tips, and a global message for the whole form.
 */
function validateNewPasswordForm(payload) {
  let isFormValid = true;
  let message = '';



  if (!(payload.confir_password.trim()===payload.password.trim())){
    isFormValid = false;
    message = "Password must be equal"
  }


  if (!payload || typeof payload.password !== 'string' || payload.password.trim().length < 8 || payload.password.trim().length > 15
    || typeof payload.confir_password !== 'string' || payload.confir_password.trim().length < 8 || payload.confir_password.trim().length > 15 ) {
    isFormValid = false;
    message = 'Password should be between 8 and 15 alphanumeric characters.';
  }


  return {
    success: isFormValid,
```

```
    message
  };
}


/**
 * Función para la validación del formulario de inicio de sesión
 *
 * @param {object} payload - el cuerpo del mensaje HTTP
 * @returns {object} Resultado de la validacion. Contendrá un mensaje y
una bandera.
 */
function validateLoginForm(payload) {
  let isFormValid = true;
  let message = '';


  if   (!payload   ||   typeof   payload.password   !==   'string'   ||
payload.password.trim().length == 0 || typeof payload.email !== "string"
|| payload.email.trim().length == 0)  {
    isFormValid = false;
    message = 'Please provide your credentials.';
  }


  return {
    success: isFormValid,
    message
  };
}


/**
 * Validate the Reset form
 *
 * @param {object} payload - the HTTP body message
 * @returns {object} The result of validation. Object contains a boolean
validation result,
 *                       errors tips, and a global message for the whole
form.
 */
function validateForgotForm(payload) {
  let isFormValid = true;
```

```
  let message = '';

  if (!payload || typeof payload.email !== 'string' ||
payload.email.trim().length == 0 ) {
    isFormValid = false;
    message = 'Please provide your credentials.';
  }

  return {
    success: isFormValid,
    message
  };
}

router.post('/login', (req, res, next) => {
  const validationResult = validateLoginForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  return passport.authenticate('local-login', (err, token, userData) => {
    if (err) {
      if (err.name === 'IncorrectCredentialsError') {
        return res.status(400).json({
          success: false,
          message: err.message
        });
      }
      return res.status(400).json({
        success: false,
        message: 'Could not process the form.'
      });
    }
    return res.json({
      success: true,
      message: 'You have successfully logged in!',
      token,
```

```
      user: {
        name: userData.name,
        role: userData.role,
        id: userData.id,
        OrganizationId: userData.OrganizationId,
        email: req.body.email.toLowerCase()
       }
    });
  })(req, res, next);
});

router.post('/forgot', function(req, res, next) {
  const validationResult = validateForgotForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  async.waterfall([
    function(done) {
      crypto.randomBytes(20, function(err, buf) {
        var token = buf.toString('hex');
        done(err, token);
      });
    },
    function(token, done) {
      models.User.findOne({
                    where: {
                        email: req.body.email
                    }
                })
        .then(function(user, err){
          if (!user) {
            return res.status(400).json({
            success: false,
            message: "No user registered with this email!"
            });
```

```
          }

          //Se ha creado anteriormente un token aleatorio
          user.resetPasswordToken = token;
          user.resetPasswordExpires = Date.now() + 3600000; // 1 hour

          user.save().then(function(user) {
            done(err, token, user);
            }, function(err){
               done(err, token, user);
          });
       });
    },
    function(token, user, done) {
      var mailOptions = {
        to: user.email.toLowerCase(),
        from: 'davsensan@gmail.com',
        subject: 'Redborder licensing management Password Reset',
        text: 'You are receiving this because you (or someone else) have
requested the reset of the password for your account in licensing
management of RedBorder.\n\n' +
          'Please click on the following link before one hour, or paste
this into your browser to complete the process:\n\n' +
          'https://' + req.headers.host + '/#/reset/' + token + '\n\n' +
          'If you did not request this, please ignore this email and your
password will remain unchanged.\n'
      };
      smtpTransport.sendMail(mailOptions,function(err) {
        res.status(200).json({
          success: true,
          message:    "An    e-mail    has    been    sent    to    "    +
user.email.toLowerCase() + " with further instructions."
        });
        done(err, 'done');
      });

    }
  ], function(err) {
    if (err) return next(err);
  });
```

```
});

router.post('/reset/:token', function(req, res) {
  const validationResult = validateNewPasswordForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  async.waterfall([
    function(done) {
      models.User.findOne({
                    where: {
                        resetPasswordToken: req.params.token,
                        resetPasswordExpires: {$gt: Date.now()}
                    }
                })
      .then(function(user, err){
        if (!user) {
          return res.status(400).json({
          success: false,
          message: "Password reset token is invalid or has expired."
          });
        }

        user.password = req.body.password;
        user.resetPasswordToken = null;
        user.resetPasswordExpires = null;

        user.save().then(function(user, err) {
            done(err, user);
          });
        });
    },
    function(user, done) {
      var mailOptions = {
        to: user.email.toLowerCase(),
```

```
        from: 'davsensan@gmail.com',
        subject: 'Your password has been changed in licensing management
of RedBorder',
        text: 'Hello,\n\n' +
          'This is a confirmation that the password for your account ' +
user.email.toLowerCase() + ' has just been changed.\n'
      };
      smtpTransport.sendMail(mailOptions,function(err) {
        res.status(200).json({
        success: true,
        message: "An e-mail has been sent to " + user.email.toLowerCase()
+ " with confirmation. The password has been changed"
        });
        done(err, 'done');
      });
    }
  ], function(err) {
    if (err) return next(err);
  });
});



module.exports = router;
```

Código B-12. Fichero con las rutas *api* server/routes/api.js

```
const jwt = require('jsonwebtoken');
const express = require('express');
const router = new express.Router();
const passport = require('passport');
const MODE_RUN = process.env.MODE_RUN || "development"
const email = require('../../config/config.json')[MODE_RUN].email;
const nodemailer = require('nodemailer');
const mockTransport = require('nodemailer-mock-transport');
const path = require('path');
const mime = require('mime');
const fs = require('fs');
const NodeRSA = require('node-rsa');

//Clave privada para la firma
const       private_key       =       process.env.PRIVATE_KEY       ||
```

```
require('../private.key.json');
const key = new NodeRSA(private_key);

//Incializamos sequelize
const sequelize = require('../db').sequelize;

//Cargamos los modelos
const models = require('../models')(sequelize);

//Configuramos el envío de emails
const transportMock = mockTransport({ //Configuramos el mock para el envío de correos
              service: process.env.EMAIL_SERVER || email.server,
              auth: {
                user: process.env.EMAIL_USER || email.email,
                pass: process.env.EMAIL_PASSWORD || email.password
              }
            });
const smtpTransport = MODE_RUN=="test" ? //Si estamos en modo test utilizamos el mock
            nodemailer.createTransport(transportMock)
            :
            nodemailer.createTransport({
              service: process.env.EMAIL_SERVER || email.server,
              auth: {
                user: process.env.EMAIL_USER || email.email,
                pass: process.env.EMAIL_PASSWORD || email.password
              }
            });
/**
 * Validate the create license form
 *
 * @param {object} payload - the HTTP body message
 * @returns {object} The result of validation. Object contains a boolean validation result
 *                   and a global message for the whole form.
 */
function validateCreateLicenseForm(payload) {
  let isFormValid = true;
```

```
  let message = '';


  if (!payload || typeof payload.duration !== 'string' ||
payload.duration.trim().length === 0) {
    isFormValid = false;
    message = 'Please provide the expires date ';
  }


  if (!payload || typeof payload.limit_bytes !== 'string' ||
payload.limit_bytes.trim().length === 0) {
    isFormValid = false;
    message = message != "" ? message + 'and please provide a limit bytes
correct ' : "Please provide a limit bytes correct ";


  }
  if (!payload || typeof payload.sensors !== 'string') {
    isFormValid = false;
    message = message != "" ? message + 'and please provide sensors ' :
"Please provide sensors ";


  }
  else{
      const sensors_object = JSON.parse(payload.sensors);
      for(const sensor in sensors_object){
          if(sensors_object[sensor].length == 0){
            isFormValid = false
            message = message != "" ? message + 'and please provide a
number of sensor ' + sensor + " " : 'Please provide a number of sensor '
+ sensor
          }
        }
  }


  return {
    success: isFormValid,
    message
  };
}


/**
```

```
 * Validate the create user form
 *
 * @param {object} payload - the HTTP body message
 * @returns {object} The result of validation. Object contains a boolean validation result
 *                   and a global message for the whole form.
 */
function validateCreateOrgForm(payload) {
  let isFormValid = true;
  let message = '';
  if (!payload || typeof payload.name !== 'string' || payload.name.trim().length === 0) {
    isFormValid = false;
    message = 'Please provide the organization name ';
  }

  if (!payload || typeof payload.email !== 'string' || payload.email.trim().length === 0) {
    isFormValid = false;
    message = message != "" ? message + 'and please provide a organization email address ' : "Please provide a organization email address ";

  }

  if (!payload || typeof payload.cluster_id !== 'string' || payload.cluster_id.trim().length === 0) {
    isFormValid = false;
    message = message != "" ? message + 'and please provide a cluster id ' : "Please provide a cluster id ";

  }

  return {
    success: isFormValid,
    message
  };
}
```

```
/**
 * Validate the create user form
 *
 * @param {object} payload - the HTTP body message
 * @returns {object} The result of validation. Object contains a boolean validation result
 *                   and a global message for the whole form.
 */
function validateCreateUserForm(payload) {
  let isFormValid = true;
  let message = '';
  if (!payload || typeof payload.name !== 'string' || payload.name.trim().length === 0) {
    isFormValid = false;
    message = 'Please provide your name ';
  }

  if (!payload || typeof payload.email !== 'string' || payload.email.trim().length === 0) {
    isFormValid = false;
    message = message != "" ? message + 'and please provide your email address ' : "Please provide your email address ";

  }

  if (!(payload.confir_password.trim()===payload.password.trim())){
    isFormValid = false;
    message = message != "" ? message + 'and new passwords must be the same ' : 'New passwords must be the same ';
  }

  if ((typeof payload.password !== 'string' || payload.password.trim().length < 8) ||
   (typeof payload.confir_password !== 'string' || payload.confir_password.trim().length < 8)) {
    isFormValid = false;
    message = message != "" ? message + 'and new password should be between 8 and 15 alphanumeric characters ' : 'New password should be between 8 and 15 alphanumeric characters ';
  }
```

```
  return {
    success: isFormValid,
    message
  };
}

/**
 * Función para validar el formulario para el cambio de perfil
 *
 * @param {object} payload - el cuerpo del mensaje HTTP
 * @returns {object} El resultado de la validación. Objeto con una bandera y un mensaje de error si procede
 */
function validateChangeProfileForm(payload) {
  let isFormValid = true;
  let message = '';

  if (!payload || typeof payload.name !== 'string' || payload.name.trim().length === 0) {
    isFormValid = false;
    message = 'Please provide your name ';
  }

  if (!payload || typeof payload.email !== 'string' || payload.email.trim().length === 0) {
    isFormValid = false;
    message = message != "" ? message + 'and please provide your email address ' : "Please provide your email address ";

  }

  if (payload.new_password && payload.confir_new_password &&
    !(payload.confir_new_password.trim()===payload.new_password.trim())){
    isFormValid = false;
    message = message != "" ? message + 'and new passwords must be the same ' : 'New passwords must be the same ';
  }

  if (payload.new_password && (typeof payload.new_password !== 'string' || payload.new_password.trim().length < 8) ||
```

```
   payload.confir_new_password && (typeof payload.confir_new_password !==
'string' || payload.confir_new_password.trim().length < 8)) {
    isFormValid = false;
    message = message != "" ? message + 'and new password should be
between 8 and 15 alphanumeric characters ' : 'New password should be
between 8 and 15 alphanumeric characters ';
  }


  if(!payload.password){
    isFormValid = false;
    message = message != "" ? message + "and password couldn't be empty "
: "Password couldn't be empty ";
  }


  return {
    success: isFormValid,
    message
  };
}


router.post('/changeProfile', (req, res) => {
  const validationResult = validateChangeProfileForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user.verifyPassword(req.body.password)){
        return res.status(401).json({
            success: false,
            message: "Current password is not correct.",
          });
      }
      if(req.body.new_password)
```

```
        user.password = req.body.new_password;
        user.email = req.body.email.trim().toLowerCase();
        user.name = req.body.name;
        user.save().then(function(){
          return res.status(200).json({
            success: true,
            message: "You have changed your profile correctly!",
            user
          });
        }, function(){
          return res.status(400).json({
            success: false,
            message: "Error changing user profile. This email alredy
exist"
          });
        })
      }, function(err){
        return res.status(400).json({
        success: false,
        message: "Error. User not found.",
        });
      })
});

router.post('/users', (req, res, next) => {
  const validationResult = validateCreateUserForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
```

```
        return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
        return passport.authenticate('local-signup', (err) => {
          if(err){
            if (err.message=="Validation error") {
                return res.status(409).json({
                  success: false,
                  message: "This email is already registered"
                });
              }
            else {
                  return res.status(409).json({
                  success: false,
                  message: err.message
                });
              }
          }
            const mailOptions = {
              to: req.body.email.toLowerCase(),
              from: 'davsensan@gmail.com',
              subject: 'Your email has been registered in RedBorder
licenses',
              text: 'Hello,\n\n' +
                'You have been registered in RedBorder. Your email is ' +
req.body.email.toLowerCase() + ' and your password is ' +
req.body.password + '.\n'
                 + "Please, log in and change your password"
            };
```

```
            smtpTransport.sendMail(mailOptions,function(err) {
              res.status(200).json({
              success: true,
              message: 'User registered successfully. A email has been
send to ' + req.body.email.trim() + ' with the password'
              });
            });

        })(req, res, next);
      }
    })
});

router.get('/users', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.User.findAndCount({
            limit: 10,
            offset: 10*(req.query.page-1),
            order: 'name'
        }).then(function(result){
```

```
            return res.status(200).json({
              success: true,
              users: result.rows,
              number_users: result.count
            })
          })
      }
    })
  });

router.delete('/users/:id', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.User.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(user_delete){
            if(!user_delete)
              return res.status(400).json({
                success: false,
                message: "User doesn't exists"
```

```
              })
             const name = user_delete.name;
             const email = user_delete.email;
             models.User.destroy({
             where: {id: req.params.id}
            }).then(function(affectedRows){
             if(affectedRows==1)
               return res.status(200).json({
               success: true,
               message: "User " + name + " (" + email + ") delete
correctly"
             })
             else
               return res.status(400).json({
               success: false,
               message: "Error removing user " + name
             })
           })
         })
       }
     })
  });


router.put('/users/:id', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
```

```
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.User.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(user_edit){
            if(!user_edit)
              return res.status(400).json({
                success: false,
                message: "User doesn't exists"
              })
            user_edit.name=req.body.name;
            user_edit.email=req.body.email;
            user_edit.role=req.body.role;
            user_edit.OrganizationId= (req.body.organization == "No" ||
req.body.organization == "" ) ? null: req.body.organization;
            user_edit.save()
            .then(function(user_save){
              return res.status(200).json({
              success: true,
              message: "User " + user_save.name + " edited correctly",
              user: user_save
              })
            }).catch(function (err) {
              return res.status(400).json({
              success: false,
              message: "Error editing user " + user_edit.name + '. Email
already exists.'
              })
            });
          })
      }
    })
  });
```

```
router.post('/organizations', (req, res) => {
  const validationResult = validateCreateOrgForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
          return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if (user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          const NewOrganization = models.Organization.build({
            cluster_id: req.body.cluster_id.trim(),
            name: req.body.name.trim(),
            email: req.body.email.trim(),
            sensors: req.body.sensors.trim()
          });
          NewOrganization.save().then(function(NewOrganization) {
            return res.status(200).json({
              success: true,
              message: 'Organization ' + NewOrganization.name + ' created
```

```
correctly'
            });
          }, function(err){
            return res.status(409).json({
              success: false,
              message: 'Error saving organization ' + req.body.name + '.
Email already exists.'
            });
        });
      }
    })
});

router.get('/organizations', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
          return res.status(404).json({
            success: false,
            message: "This user doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.Organization.findAll({
          include: [{
            "model": models.User
          }],
          limit: 10,
          offset: 10*(req.query.page-1),
```

```
          order: 'name'
        }).then(function(orgs){
          models.Organization.count({}).then(function(number_orgs){
            return res.status(200).json({
              success: true,
              orgs: orgs,
              number_orgs: number_orgs
            })
          });
        })
      }
    })
  });


router.delete('/organizations/:id', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.Organization.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(org_delete){
            if(!org_delete)
              return res.status(400).json({
                success: false,
                message: "Organization doesn't exists"
```

```
            })
            const name = org_delete.name;
            const email = org_delete.email;
            models.Organization.destroy({
            where: {id: req.params.id}
           }).then(function(affectedRows){
            if(affectedRows==1)
              return res.status(200).json({
              success: true,
              message: "Organization " + name + " (" + email + ") delete
correctly"
            })
            else
              return res.status(400).json({
              success: false,
              message: "Error removing organization " + name
            })
          })
        })
      }
    })
  });

router.put('/organizations/:id', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.Organization.findOne({
            where: {
```

```
            id: req.params.id
          }
        }).then(function(org_edit){
          if(!org_edit)
            return res.status(400).json({
              success: false,
              message: "Organization doesn't exists"
            })
          org_edit.name=req.body.name;
          org_edit.email=req.body.email;
          org_edit.cluster_id=req.body.cluster_id;
          org_edit.sensors=req.body.sensors;
          org_edit.save()
          .then(function(org_save){
            return res.status(200).json({
            success: true,
            message: "Organization " + org_save.name + " edited
correctly",
            org: org_save
            })
          }).catch(function (err) {
            const message = err.message=="Validation error" ? "Email
already exists." : err.message;
            return res.status(400).json({
            success: false,
            message: "Error editing organization " + org_edit.name + '.
' + message
            })
          });
        })
      }
    })
  });


//Metodo get al que se llama al crear un usuario. Devuelve la lista de
organizaciones disponibles
router.get('/users/new', (req, res) => {
  models.User.findOne({
        where: {
```

```
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.Organization.findAll({
          order: 'name'
        }).then(function(list_orgs){
          return res.status(200).json({
            success: true,
            orgs: list_orgs,
          })
        })
      }
    })
  });

//Metodo get al que se llama al editar un usuario. Devuelve dicho usuario
router.get('/users/:id/edit', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
```

```
            message: "This users doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.Organization.findAll({
          order: 'name'
        }).then(function(list_orgs){
           models.User.findOne({
            where: {
                id: req.params.id
            }
          }).then(function(user_edit){
          return res.status(200).json({
            success: true,
            orgs: list_orgs,
            user: user_edit
          })
        })
      })
      }
    })
  });

  //Metodo get al que se llama al editar un usuario. Devuelve dicho
usuario
router.get('/organizations/:id/edit', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
```

```
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
        models.Organization.findOne({
          where: {
            id: req.params.id
          }
        }).then(function(org){
          return res.status(200).json({
            success: true,
            org: org
          })
        })
      }
    })
  });

router.get('/organizations/:id/users', (req, res) => {
 models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
```

```
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.User.findAndCount({
            where : {
              OrganizationId:    req.params.id=="null"    ?    null    :
req.params.id
            },
            limit: 10,
            offset: 10*(req.query.page-1),
            order: 'name'
        }).then(function(result){
              return res.status(200).json({
                success: true,
                users: result.rows,
                number_users: result.count
              })
          })
      }
    })
  });


  //Devuelve las licencias que pertenecen a una organización.
  //Solo pueden acceder a él usuarios administradores y que pertenezcan a
la organización que mostrará las licencias
  router.get('/organizations/:id/licenses', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
```

```
            success: false,
            message: "This users doesn't exist",
          });
      }
      else   if(user.role   !=   "admin"   &&   user.OrganizationId   !=
req.params.id ){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.License.findAndCount({
            where : {
              OrganizationId: req.params.id
            },
            limit: 10,
            offset: 10*(req.query.page-1),
            order: [['enabled'],['expires_at']]
        }).then(function(result){
              return res.status(200).json({
                success: true,
                licenses: result.rows,
                number_licenses: result.count
              })
          })
      }
    })
  });


router.post('/licenses', (req, res) => {
  const validationResult = validateCreateLicenseForm(req.body);
  if (!validationResult.success) {
    return res.status(400).json({
      success: false,
      message: validationResult.message,
    });
  }
```

```
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      //Si la licencia que se quiere crear no es para la organización a la que pertenecemos... no podemos
      else if(user.OrganizationId != req.body.OrganizationId && user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          //Si no se le ha pasado el license_uuid significa que estamos extendiendo una licencia, en ese caso el tiempo de expiración tambien existirá.
            const NewLicense = req.body.license_uuid ? models.License.build({
              expires_at: req.body.expires_at,
              license_uuid: req.body.license_uuid,
              duration: req.body.duration.trim(),
              limit_bytes: req.body.limit_bytes.trim(),
              OrganizationId: req.body.OrganizationId.trim(),
              UserId: user.id,
              sensors: req.body.sensors
            })
            :
             models.License.build({
              duration: req.body.duration.trim(),
              limit_bytes: req.body.limit_bytes.trim(),
```

```
              OrganizationId: req.body.OrganizationId.trim(),
              UserId: user.id,
              sensors: req.body.sensors
            })

          NewLicense.save().then(function(NewLicense) {
            return res.status(200).json({
              success: true,
              message: 'License created correctly'
            });
          }, function(err){
            return res.status(400).json({
              success: false,
              message: 'Error creating license.<br></br> The sensors and
limit bytes must be numbers'
            });
        });
      }
    })
  });

router.get('/licenses/new', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
      }
      //Si la licencia que se quiere crear no es para la organización a
la que pertenecemos... no podemos
      else   if(user.OrganizationId   !=   req.query.OrganizationId   &&
user.role != "admin"){
        return res.status(401).json({
            success: false,
```

```
            message: "You don't have permissions",
          });
      }
      else
      {
        models.Organization.findOne({
          where: { id: req.query.OrganizationId}
        }).then(function(organization) {
          return res.status(200).json({
            success: true,
            sensors: organization.sensors
          });
        }, function(err){
          return res.status(400).json({
            success: false,
            message: 'Error, this organization id not exists'
          });
        });
      }
    })
});

router.get('/licenses/extend', (req, res) => {
models.User.findOne({
      where: {
          id: req.userId
      }
  }).then(function(user){
    models.License.findOne({
      where: { id: req.query.LicenseId}
    }).then(function(license) {
      if(!user){
      return res.status(404).json({
          success: false,
          message: "This users doesn't exist",
        });
      }
      //Si  la  licencia  que  se  quiere  extender  no  es  para  la
```

```
organización a la que pertenecemos o no somos admin... no podemos
        else   if(user.OrganizationId   !=   license.OrganizationId   &&
user.role != "admin")
        {
          return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
        }
        else if(!license.enabled){
          return res.status(401).json({
              success: false,
              message: "This license is pending of activation",
            });
        }
        else
        {
          license.sensors = JSON.parse(JSON.parse(license.sensors));
          return res.status(200).json({
            success: true,
            license: license
            });
        }
      }, function(err){
        return res.status(400).json({
          success: false,
          message: 'Error, this license not exists'
        });
      });
    })
  });


  // safeURLBase64Encode :: string -> toString
  const safeURLBase64Encode = (data) =>
    new Buffer(data)
      .toString('base64')
      .replace(/\//g, '_')
      .replace(/\+/g, '-');
```

```
  router.get('/licenses/download', (req, res) => {
    models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      models.License.findOne({where: {
        id: req.query.LicenseId
      }}).then(function(license){
        if(!user){
        return res.status(404).json({
            success: false,
            message: "This users doesn't exist",
          });
        }
        //Si la licencia que se quiere descargar no es para la organización a la que pertenecemos o no somos administradores... no podemos
        else if(user.OrganizationId != license.OrganizationId && user.role != "admin"){
          return res.status(401).json({
              success: false,
              message: "You don't have permissions",
            });
        }
        else if(!license.enabled){
          return res.status(401).json({
              success: false,
              message: "This license is pending of activation",
            });
        }
        else
        {
            models.Organization.findOne({
              where: {
                id: license.OrganizationId
              }
            }).then(function(organization){
```

```
              const license_json = {
                info: {
                  uuid: license.id,
                  cluster_uuid: organization.cluster_id,
                  expire_at: license.expires_at.getTime()/1000,
                  limit_bytes: license.limit_bytes,
                  sensors: JSON.parse(JSON.parse(license.sensors)),
                  createdAt: license.createdAt.toISOString()
                }
              }

              license_json.encoded_info                                    =
safeURLBase64Encode(JSON.stringify(license_json.info));
              //Firmado de la licencia
              license_json.signature                                       =
safeURLBase64Encode(key.sign(license_json.encoded_info));
              res.writeHead(200,   {'Content-Type':   'application/force-
download','Content-disposition':'attachment;         filename='         +
req.query.LicenseId + '.lic'});
              return
res.end(safeURLBase64Encode(JSON.stringify(license_json)));
            }, function(err){
              return res.status(404).json({
                success: false,
                message: "License not found!"
              })
          })
        }
      });
    });
  });

  router.put('/licenses/activate/:id', (req, res) => {
  models.User.findOne({
        where: {
            id: req.userId
        }
    }).then(function(user){
      if(!user){
        return res.status(404).json({
```

```
            success: false,
            message: "This users doesn't exist",
          });
      }
      else if(user.role != "admin"){
        return res.status(401).json({
            success: false,
            message: "You don't have permissions",
          });
      }
      else
      {
          models.License.findOne({
            where: {
              id: req.params.id
            }
          }).then(function(license_activate){
            if(!license_activate)
              return res.status(400).json({
                success: false,
                message: "License doesn't exists"
              })
            //Cambiamos el tiempo de expiración y la marcamos como
activada
            //Si la duración es negativa significa que es una extensión y
por tanto el tiempo de expiración ya está configurado
            if(license_activate.duration>0){
              const now = new Date();
              const expires_time = now.setMonth(now.getMonth() +
license_activate.duration);
              license_activate.expires_at=expires_time;
            }
            license_activate.enabled=true;
            license_activate.save()
            .then(function(license_saved){
              models.Organization.findOne({
                where:{
                  id: license_saved.OrganizationId
                }
```

```
              })
              .then(function(org){
                const mailOptions = {
                  to: org.email,
                  from: 'davsensan@gmail.com',
                  subject: "Your license has been activated",
                  text: 'Hello,\n\n' +
                    'Your license ' + license_saved.license_uuid + " has
been activated until " + license_saved.expires_at + ".\n You can use this
license since right now.\n Thank you!"
                };
                smtpTransport.sendMail(mailOptions,function(err) {
                  res.status(200).json({
                    success: true,
                    message: "License " + license_saved.id + " activated
correctly",
                    license: license_saved
                  })
                });
              })
            }).catch(function (err) {
              return res.status(400).json({
              success: false,
              message: "Error to activate license " + license_activate.id
              })
            });
          })
      }
    })
  });



module.exports                          =                          router;
```

ANEXO C
CÓDIGO CONFIGURACIÓN *DOKCKERIZACIÓN*

Código C-1. Fichero *dockerfile* para producción /Dockerfile

```
FROM node:slim
#Variables de entorno para desarrolo
ENV NODE_ENV=production
ENV MODE_RUN=production

#Creación del directorio de la aplicación dentro del docker
RUN mkdir -p /app_license
WORKDIR /app_license

#Instalación de las depencias de la aplicación para producción
COPY package.json /app_license
RUN npm install --production

#Copia de los ficheros (Excepto los de dockerignore)
COPY . /app_license

#Construimos el fichero principal con webpack
RUN npm run build

#Activamos el puerto 3000
EXPOSE 3000

#Configuramos como punto de entrada el arranque del servidor
ENTRYPOINT npm run start
```

Código C-2. Fichero *.dockerignore*

```
node_modules
client/dist
```

Código C-3. Fichero *dockerfile* para desarrollo /dev.dockerfile

```
FROM node:slim
#Variables de entorno para desarrolo
ENV NODE_ENV=development
```

```
ENV MODE_RUN=development

#Creación del directorio de la aplicación dentro del docker
RUN mkdir -p /app_license
WORKDIR /app_license

#Activamos el puerto 3000
EXPOSE 3000

#Ejecutamos el comando npm start en modo desarrollo
ENTRYPOINT npm run start:dev
```

Código C-4. Fichero /docker-compose.yml

```
version: '2'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/data/mysql
    environment:
      MYSQL_ROOT_PASSWORD: redborderlicensepassworddb
      MYSQL_DATABASE: licenses_management
    ports:
      - "3307:3307"
  web:
    volumes:
      - .:/app_license
    depends_on:
      - db
    links:
      - db:db
    build:
      context: .
      dockerfile: dev.dockerfile
    image: licenses_app_dev
    ports:
      - "3000:3000"
    environment:
      DB_PASSWORD: redborderlicensepassworddb
```

```
      DB_PORT: 3307
      DB_HOST: db
      DB_DATABASE: licenses_management
      EMAIL_SERVER: SendPulse
      EMAIL_USER:
      EMAIL_PASSWORD:
volumes:
  db_data:
```

ANEXO D
CÓDIGO CONFIGURACIÓN DE *WEBPACK*

Código D-1. Fichero de configuración de webpack /webpack-config.js

```
const path = require('path');

process.noDeprecation = true;
module.exports = {
  // El punto de entrada estará en la siguiente ruta
  entry: path.join(__dirname, '/client/src/index.jsx'),

  // Nombre y directorio del fichero de salida
  output: {
    path: path.join(__dirname, '/client/dist/js'),
    filename: 'app.js',
  },

  module: {

    // Loaders utilizado para la transpilación
    loaders: [{
      test: /\.jsx?$/,
      include: path.join(__dirname, '/client/src'),
      loader: 'babel-loader',
      query: {
        presets: ["react", "es2015"]
      }
    }],
  },
}
```

ANEXO E
CÓDIGO CONFIGURACIÓN DE *TRAVIS*

Código E-1. Fichero de configuración de travis /.travis.yml

```
language: node_js
node_js:
    - "stable"
services:
    - docker
after_success:
    - codecov
before_install:
    - pip install --user codecov
    -   docker   run   -e   MYSQL_DATABASE=licenses_management_test   -e
MYSQL_ROOT_PASSWORD=root -d -p 3310:3306 mysql:5.7
script:
    - DB_PORT=3310 DB_PASSWORD=root npm run test
```

ANEXO F
CÓDIGO CONFIGURACIÓN DE *NPM*

Código F-1. Fichero package.json

```
{
  "name": "licenses_management",
  "version": "1.0.0",
  "description": "Portal web de licencias de RedBorder",
  "main": "index.js",
  "scripts": {
    "start:dev": "npm-run-all --parallel build:dev nodemon",
    "start": "node ./server/index.js",
    "nodemon": "nodemon --use_strict ./server/index.js",
    "build:dev": "webpack -w",
    "build": "webpack",
    "test": "export MODE_RUN='test' && istanbul cover _mocha -- --timeout
10000 test/**/*"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/redBorder/licensing-management.git"
  },
  "author": "David Señas",
  "license": "MIT",
  "dependencies": {
    "async": "^2.4.0",
    "babel-core": "^6.24.1",
    "babel-loader": "^7.0.0",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "body-parser": "^1.17.1",
    "create-react-class": "^15.5.3",
    "crypto": "0.0.3",
    "express": "^4.15.2",
    "file-saver": "^1.3.3",
    "jquery": "^3.2.1",
```

```
    "jsonwebtoken": "^7.4.0",
    "mysql": "^2.13.0",
    "node-rsa": "^0.4.2",
    "nodemailer": "^4.0.1",
    "nodemailer-mock-transport": "^1.3.0",
    "passport": "^0.3.2",
    "passport-local": "^1.0.0",
    "password-hash": "^1.2.2",
    "prop-types": "^15.5.9",
    "react": "^15.5.4",
    "react-bootstrap": "^0.31.0",
    "react-bootstrap-table": "^3.3.3",
    "react-dom": "^15.5.4",
    "react-router": "^3",
    "react-router-bootstrap": "^0.24.2",
    "react-tap-event-plugin": "^2.0.1",
    "sequelize": "^3.30.4",
    "toastr": "^2.1.2",
    "validator": "^7.0.0",
    "webpack": "^2.5.1"
  },
  "devDependencies": {
    "assert": "^1.4.1",
    "chai": "^3.5.0",
    "chai-http": "^3.0.0",
    "istanbul": "^0.4.5",
    "mocha": "^3.3.0",
    "nodemon": "^1.11.0",
    "npm-run-all": "^4.0.2",
    "sequelize-fixtures": "^0.5.6"
  }
}
```

ÍNDICE DE CÓDIGOS

ÍNDICE DE FIGURAS