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

Aplicación de gestión de juegos para la educación con framework Spring y Primefaces

Autora: Ana María Lobón Roldán
Tutora: Teresa Ariza Gómez
Cotutor: Francisco Javier Muñoz Calle

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación de gestión de juegos para la educación con framework Spring y Primefaces

Autora:
Ana María Lobón Roldán

Tutora:
Teresa Ariza Gómez
Profesora titular

Cotutor:
Francisco Javier Muñoz Calle

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Grado: Aplicación de gestión de juegos para la educación con framework Spring y Primefaces

Autora: Ana María Lobón Roldán

Tutora: Teresa Ariza Gómez

Cotutor: Francisco Javier Muñoz Calle

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A los que han estado,

A los que están

Agradecimientos

Y el final de otra etapa ha llegado. Quizás no es el final que había imaginado, pero después de todo es el final.

Gracias a todas esas personas que han formado parte de esta bonita etapa de mi vida, llena de caídas y tropiezos, pero también de ilusiones y triunfos.

Gracias a ese maldito edificio rojo y a todos sus docentes por todas sus enseñanzas, tanto de conocimiento como de superación.

Gracias a mis tutores de este proyecto, Teresa y Javi, así como a Jesús, gracias al cual nació esta idea. Por el apoyo y la paciencia.

Gracias a todos mis niños de teleco, por acogerme en mi soledad de las específicas de Telemática. Y sobre todo por los momentos vividos fuera de la escuela.

Gracias María, eres esa persona que no encontraba, esa con la que encajas. Por las locuras vividas y las que viviremos.

Gracias Irene, por descubrirme la existencia de este grado que parecía ser el mix perfecto para mí. Pero, sobre todo, gracias por ser mi constante desde aquella post romería.

Gracias a mis padres y a mi hermana, por soportarme y apoyarme en la distancia.

Gracias a ti, que fuiste mi familia durante esta larga etapa. Estuviste ahí desde el minuto uno y formaste parte de todos los aspectos de mi vida, haciendo que los momentos duros los fueran menos y los felices los fueran aún más.

Ana Lobón

Conil de la Frontera, 2020

Resumen

Este proyecto forma parte de otro más grande que pretende evolucionar el proyecto de Aplicación de Juegos Didácticos en el Aula (AJDA), de forma que se cree una nueva plataforma de juegos educativos adaptada a las nuevas tecnologías haciendo que su uso sea más intuitivo y automatizado.

En nuestro proyecto se ha realizado el diseño e implementación de la aplicación web para la gestión de usuarios, juegos y partidas. Esta aplicación permite el registro y acceso de los usuarios, los cuales podrán desde ella configurar diferentes partidas, seleccionando el juego y los equipos que se enfrentarán, entre otros parámetros. Por otro lado, también se incluye la gestión de usuarios y juegos para superusuarios y administradores de la plataforma, de manera que podrán crear, modificar y eliminar usuarios y juegos libremente.

Para esta aplicación web se ha optado por separar la aplicación web del acceso a base de datos. De forma que tendremos por un lado una aplicación web al uso, en la que se ha usado la tecnología de JSF junto con la potente librería de componentes Primefaces. Por otro lado, una API REST que servirá de interfaz de acceso a una base de datos relacional PostgreSQL y ha sido implementada con ayuda del framework Spring.

En este documento se detallan todos los aspectos funcionales y técnicos de interés para comprender en profundidad los entresijos de esta aplicación.

Abstract

This project is part of a larger one that aims to evolve the Project of Aplicación de Juegos Didácticos en el Aula (AJDA), so that a new educational games platform adapted to new technologies is created making its use more intuitive and automated.

In our project we have carried out the design and implementation of the web application for the management of users, games and matches. This application allows the registration and access of users, who will be able to configure different matches from it, selecting the game and the teams that will face, among other parameters. On the other hand, it also includes the management of users and games for superusers and administrators of the platform, so that they can create, modify, and delete users and games freely.

For this web application, we have chosen to separate the web application from database access. So, we will have on the one hand a web application to use, in which JSF technology has been used together with the powerful Primefaces component library. On the other hand, a REST API that will serve as an interface for accessing a PostgreSQL relational database and has been implemented with the help of the Spring framework.

This document details all the functional and technical aspects of interest to fully understand the ins and outs of this application.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS UML

INTRODUCCIÓN

La tecnología se alimenta a sí misma.
La tecnología hace posible más tecnología.
- Alvin Toffler -

La educación debe estar en constante cambio y adaptarse a las nuevas tecnologías. Vivimos en un mundo en el que las nuevas generaciones nacen con un dispositivo electrónico debajo del brazo. Esto hace necesario que los profesores incluyan las nuevas tecnologías como herramienta de apoyo a la hora de impartir sus lecciones.

El proyecto global, del que forma parte este proyecto, trata de esto mismo. El objetivo es crear una herramienta de apoyo para todo el conjunto de docentes, desde los maestros de infantil hasta los profesores universitarios. Esta herramienta tiene como objetivo repasar las diferentes lecciones impartidas o incluso evaluar el nivel de la clase de una forma divertida y didáctica mediante una Plataforma de Juegos Interactivos.

1. Motivación y objetivos

El proyecto Descartes es una herramienta de autor multipropósito y consecuentemente es posible el desarrollo de recursos educativos interactivos de cualquier materia o área de conocimiento. Desde la asociación "Red Educativa Digital Descartes" se pone al servicio de la comunidad educativa de la aldea global, y en general de cualquier usuario, entre otros, el proyecto de Aplicación de Juegos Didácticos en el Aula (AJDA). [1]

El proyecto AJDA es un proyecto educativo dentro del cual se han elaborado una gran variedad de recursos, cuyos contenidos (preguntas, respuestas, palabras, cifras, frases...) se generan a través de formularios y se guardan en ficheros de texto que han sido catalogados y clasificados en esta web. Esto permite que el profesorado pueda elaborar de manera sencilla sus propios contenidos para los juegos o utilizar los ya elaborados. [1]

A continuación, se explicará un ejemplo básico de cómo se usaría la Aplicación de Juegos Didácticos en el Aula [2]:

1. Buscar y elegir el juego que se desee:

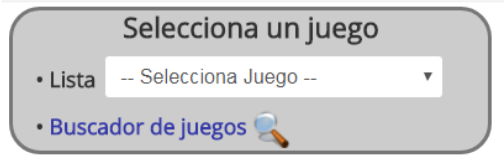


Figura 1: Ejemplo AJDA - Seleccionar Juego

2. Acceder al juego elegido, pulsando en “Acceder al juego”

Figura 2: Ejemplo AJDA - Acceder al juego

3. Seleccionar la modalidad de introducción de preguntas en el juego: las preguntas pueden introducirse de forma escrita, oral o no contener preguntas.

Figura 3: Ejemplo AJDA - Modalidad de preguntas

4. Elegir la configuración o parámetros iniciales del juego: se deben introducir los nombres de los jugadores, opciones específicas y generales del juego, carga de ficheros de preguntas para juegos con esta modalidad, etc. A través del botón ‘Continuar partida’, se puede seguir el juego a partir de una partida previamente guardada.

Figura 4: Ejemplo AJDA - Configuración del Juego

5. Pulsar el botón de 'JUGAR'

Figura 5: Ejemplo AJDA - Jugar

6. Comenzar la partida: en la pantalla irán apareciendo las preguntas y el profesor tendrá que ir introduciendo las respuestas que los alumnos vayan dando. El modelo de las respuestas y los resultados dependerá del juego seleccionado.

Figura 6: Ejemplo AJDA - Comenzar partida

El objetivo de este proyecto es crear una nueva plataforma de juegos didácticos basada en AJDA, pero adaptándolo a las nuevas tecnologías de forma que su uso sea más intuitivo y automatizado. Las principales mejoras que este proyecto pretende aportar se detallan a continuación:

- Todo el proceso de preparación de las partidas (pasos del 1 al 4) podrá realizarse con antelación, pues el acceso del profesor será mediante un usuario con el cual podrá guardar y gestionar múltiples partidas, de forma que cuando llegue a clase solo tenga que pulsar en 'Jugar' para que el juego comience.
- Los alumnos podrán interactuar directamente con el juego, respondiendo las preguntas desde sus propios dispositivos (ordenador, smartphone o Tablet).

2. Contexto del proyecto

Como ya se ha mencionado con anterioridad, este proyecto forma parte de otro más grande. Por tanto, empezaremos describiendo el proyecto principal para luego adentrarnos en el subproyecto que nos concierne.

El proyecto completo se subdivide en tres subproyectos:

- Aplicación A: Aplicación web para la gestión de usuarios, juegos y partidas.
- Aplicación B: Aplicación web para la creación de ficheros de preguntas
- Aplicación C: Aplicación web multiusuario

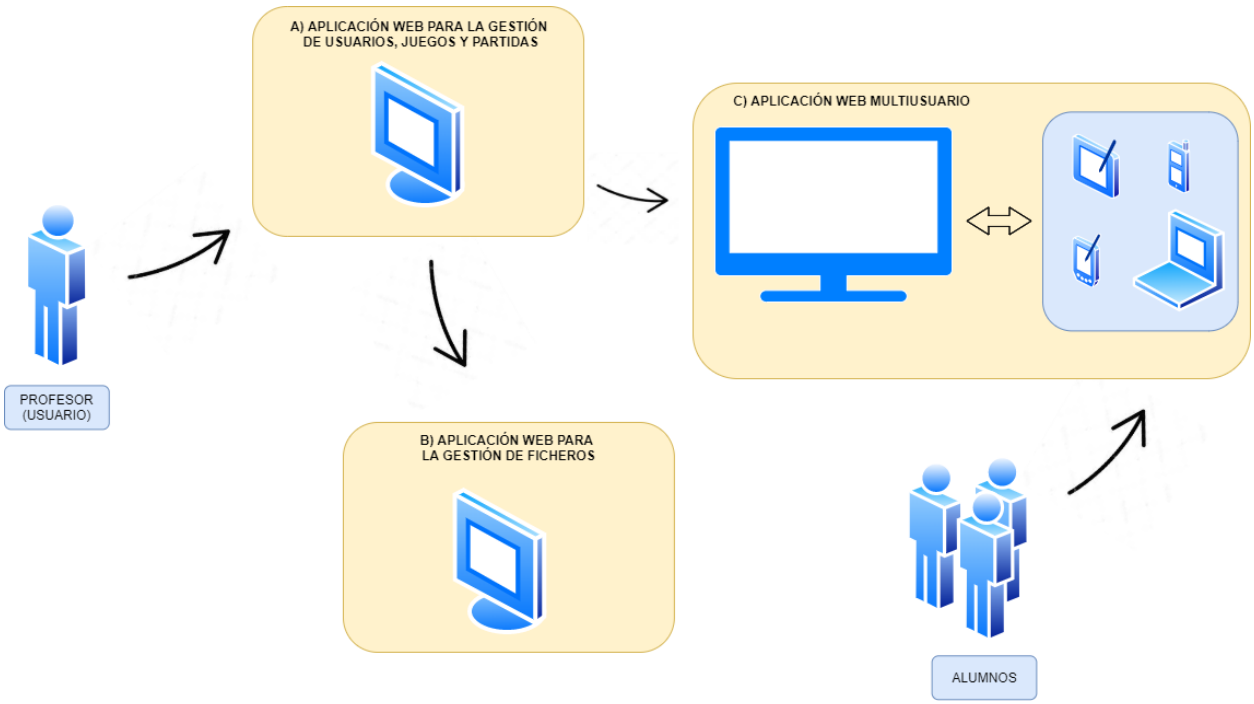


Figura 7: Escenario general

En la Figura 7 se muestra la relación de interacción entre las 3 aplicaciones, tanto entre sí como con el usuario. Se puede ver como el punto de acceso a la Plataforma es la Aplicación A, la cual está disponible para el profesor. Desde esta aplicación principal el profesor, una vez registrado, tendrá acceso a las siguientes funcionalidades:

- Creación de plantilla de partida. En esta plantilla deberá configurar parámetros como el juego que se usará y su correspondiente configuración de parámetros, el número de grupos que participarán, el nombre de los grupos, el fichero de preguntas a usar, etc.
- Listado de partidas en curso.
- Listado de partidas terminadas y sus estadísticas.
- Listado de juegos disponibles.
- Acceso a la Aplicación B (Aplicación web para la gestión de ficheros de preguntas)
- Acceso a la Aplicación C (Aplicación web multiusuario)

La idea es que el profesor durante la preparación de la clase, deje configurada la plantilla de la partida que

desea jugar en clase. Durante esta configuración deberá elegir el fichero de preguntas, lo cual se realiza desde la Aplicación B. Desde esta aplicación se podrá:

- Elegir un fichero de preguntas ya existente.
- Crear un nuevo fichero de preguntas, para lo cual existe una aplicación de edición de ficheros para que éste siga el formato correcto.

Una vez en clase, el profesor, simplemente, tendrá que acceder a la Aplicación A y comenzar una nueva partida de la plantilla ya configurada. Esto lo redirigirá a la Aplicación C, mostrando por pantalla un código. Este código deberán introducirlo los alumnos desde sus propios dispositivos en la aplicación para alumnos que les dará acceso al juego. La aplicación asignará cada alumno a un grupo.

Cuando el juego comience a cada alumno les aparecerá en sus dispositivos las posibles respuestas a la pregunta correspondiente. Los alumnos irán respondiendo pregunta tras pregunta hasta que finalice el juego. Una vez finalizado la aplicación C registrará los resultados en la Aplicación A para que el profesor tenga accesible los resultados finales de las partidas y las estadísticas finales.

De forma paralela a este proyecto, también se ha desarrollado otro proyecto similar a este pero integrado con la Enseñanza Virtual. La idea es que lo alumnos accedan a las partidas desde la Enseñanza Virtual y se autogeneren las calificaciones en la plataforma.

3. Estructura del Proyecto

El subproyecto que se detallará en este documento es la ***APLICACIÓN A: Aplicación Web para la gestión de usuarios, juegos y partidas***, mencionada en el apartado anterior.

Pasaremos a detallar la arquitectura de esta aplicación y la interacción entre los diferentes elementos que encontraremos en ella.

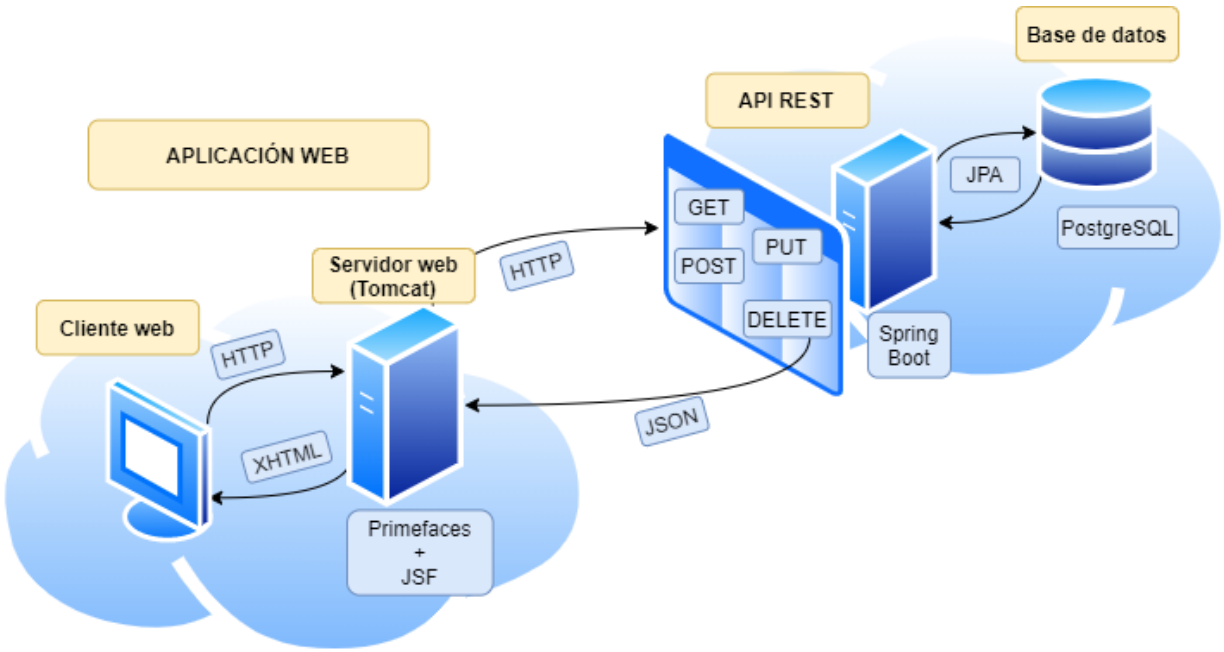


Figura 8: Arquitectura del proyecto

La arquitectura del proyecto se divide en tres elementos:

- **Aplicación web:** este elemento será el punto de acceso del profesor a la aplicación desde el *cliente web*. En el *servidor web (Tomcat)* tendremos una aplicación creada con el framework *Primefaces* sobre *JavaServer Faces (JSF).*
- **API REST:** este componente es creado con el framework *Spring Boot*. Permite acceder a los recursos de la base de datos mediante las operaciones http GET, POST, PUT y DELETE. El acceso a la base de datos se realiza usando el *framework JPA*.
- **Base de datos:** base de datos relacional creada con el sistema gestor de bases de datos PostgreSQL, en la que se almacenan todos los datos de la aplicación.

Como se puede ver en la Figura 8, la API REST hace de intermediaria entre la Aplicación Web y la Base de Datos, de manera que la Aplicación Web va realizando peticiones a la API REST para introducir, modificar, eliminar y obtener información de la base de datos.

A lo largo de esta memoria se irán detallando todos los aspectos de la aplicación desarrollada. Primeramente, se detallarán las tecnologías y herramientas usadas para el desarrollo. A continuación, se definirá la arquitectura y diseño seguidos en la implementación en el que se detallarán el modelo entidad-relación de la base de datos, la definición de cada uno de los recursos que ofrece la API REST y finalmente, el diseño de la aplicación web mediante el uso de diagramas de casos de uso y diagramas de secuencia. Después de esto, se ahondará en los aspectos más técnicos a destacar de la implementación. Se finalizará toda esta explicación, con unas conclusiones y las propuestas de mejoras futuras por las que este proyecto debe continuar.

TECNOLOGÍAS Y HERRAMIENTAS

En este capítulo, se recogen todas las tecnologías y herramientas utilizadas para la implementación de este proyecto.

1. Tecnologías

Primefaces

Primefaces es una librería de código abierto que ofrece sofisticados componentes UI (Interfaz de Usuario) para JavaServer Faces (JSF) [3]. La versión de Primefaces usada es 6.2

Figura 9: Logo de Primefaces

JavaServer Faces (**JSF**) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE [4]

Figura 10: Logo de JSF

Las principales características de JSF son: [5]

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.
- Conversión de datos y validación automática de la entrada del usuario.
- Navegación entre vistas.
- Internacionalización
- A partir de la especificación 2.0 un modelo estándar de comunicación Ajax entre la vista y el servidor.

Estas dos tecnologías son la base de la aplicación web de nuestra aplicación.

Spring Boot

Spring Boot es una infraestructura ligera que elimina la mayor parte del trabajo de configurar las aplicaciones basadas en Spring. Además, ofrece más de 30 Boot Starters. Estos Starters son una dependencia de POM que configuras en tu aplicación y a partir de esta se incluye de forma automática en tu proyecto todas las dependencias necesarias, ya sean Spring o no. Podemos encontrar starters para trabajar con JDB, Tomcat, JSON, REST, realización de tests, etc.

Figura 11: Logo de Spring Boot

En nuestro caso, estamos usando Spring Boot para la API REST, de forma que el starter que configuramos es spring-boot-starter ya que haremos uso de varios starters, como el de REST, JSON y Tomcat.

PostgreSQL

PostgreSQL es un poderoso sistema gestor de bases de datos relacional de código abierto con más de 30 años de desarrollo activo consiguiendo una sólida reputación de confianza, solidez y rendimiento. [6]

Figura 12: Logo de Postgresql

Por todo ello, este ha sido el gestor de bases de datos elegido para el proyecto y con el cual se ha gestionado y plasmado el modelo de datos.

JPA

JPA (Java Persistence API) es un modelo de persistencia desarrollado por Sun Microsystems para la plataforma Java EE. [7]

Figura 13: Logo de Java

Esta API proporciona a los desarrolladores de Java un recurso de mapeo de los recursos de una base de datos relacional a objetos java, que facilita la administración de datos relacionales en aplicaciones Java. [8]

Por tanto, esta es el modelo que se ha usado en la API REST para el acceso a la base de datos.

JSON

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C,

incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. [9]

Figura 14: Logo de JSON

Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos. [9] Por tanto, es el lenguaje en el que intercambia datos la API REST.

Además, dada su versatilidad también se hace uso de él en la configuración de los juegos. Los juegos comparten una serie de parámetros comunes, sin embargo, cada juego puede necesitar otros parámetros propios que no comparten entre ellos. Por tanto, se hace uso de JSON para establecer esos parámetros de configuración propios de cada juego.

2. Herramientas

Spring Tool Suite 3 (STS 3)

El IDE (Entorno de Desarrollo Integrado) que se ha utilizado para este proyecto ha sido Spring Tool Suite (STS) en su versión 3.9.11 para Linux.

La distribución Spring Tool Suite 3 (STS 3) está basada en el paquete Eclipse JEE e incluye los componentes de Spring IDE, la integración del servidor Tomcat para Eclipse y varios plugins del Eclipse IDE preinstalados. [10]

Figura 15: Logo de Spring

La elección de esta herramienta como entorno de desarrollo se debe a las herramientas que ofrece para facilitar para el desarrollo de la API REST, la cual usa Spring Boot. Además, como es una versión basada en el IDE de Eclipse, también ofrece todas las utilidades que éste, por tanto, también ofrece muchas herramientas para el desarrollo de la Aplicación web.

El STS 3 presenta las siguientes características:

- **Multilenguaj**e: este entorno está pensado para aplicaciones Spring (sobre java), sin embargo, al igual que Eclipse se adapta a proyectos de diferentes lenguajes.
- **Perspectivas y vistas:** según el tipo de proyecto sobre el que estemos trabajando, nos ofrece diferentes perspectivas y vistas con herramientas específicas.
- **Extensible:** existe una gran cantidad de plugins que permite ampliar las herramientas según las necesidades.

Las herramientas del STS que se han utilizado para el desarrollo son las siguientes:

- **Explorador de paquetes:** permite navegar por las clases y librerías de los diferentes proyectos.

- **Perspectiva Java:** facilita el desarrollo de aplicaciones Java, ofreciendo autocompletado de métodos, generar automáticamente getters, setters, constructores, etc
- **Perspectiva Debug:** permite depurar el código de manera fácil e intuitiva.
- **Servidor Tomcat Integrado:** facilita la configuración del servidor Tomcat para el desarrollo de la Aplicación Web. Este servidor Tomcat es independiente para cada aplicación.
- **Perspectiva Git:** permite de forma sencilla integrar un repositorio Git con los proyectos.

PgAdmin 3

pgAdmin 3 es la plataforma de administración y desarrollo de código abierto más popular y rica en características para PostgreSQL, la base de datos de código abierto más avanzada del mundo. [11]

Figura 16: Logo de PgAdmin3

Esta herramienta se ha utilizado principalmente para ver el estado de la base de datos y añadir, modificar y eliminar registros en las tablas de la base de datos.

SoapUI

SoapUI es una herramienta para la realización de pruebas a aplicaciones orientadas a servicios, tanto SOAP (Simple Object Access Protrocol) como REST (Transferencia de Estado Representacional).

Figura 17: Logo de SoapUI

Esta herramienta se ha utilizado para realizar pruebas contra los recursos de la API REST, ya que permite crear de manera fácil las peticiones a los recursos y ver la respuesta que se recibe.

JavaDoc

Javadoc es un estándar de Oracle para la generación de documentación de clases Java en formato HTML. La mayoría de los IDEs los generan automáticamente. [12]

Figura 18: Logo de JavaDoc

Con este estándar se ha generado la documentación de todas las clases Java de la Aplicación Web a través del IDE Spring Tool Suite 3.

Swagger UI

Swagger es un marco de software de código abierto respaldado por un gran ecosistema de herramientas que ayuda a los desarrolladores a diseñar, construir, documentar y consumir servicios web RESTful. Si bien la mayoría de los usuarios identifican Swagger por la herramienta Swagger UI, el conjunto de herramientas Swagger que incluye soporte para documentación automatizada, generación de código y generación de casos de prueba. [13]

Figura 19: Logo de Swagger

Por tanto, esta herramienta es la que se ha utilizado para la documentación de la API REST. Desde esta interfaz se puede consultar la especificación de cada uno de los recursos accesibles en nuestra API, y además realizar pruebas desde la misma.

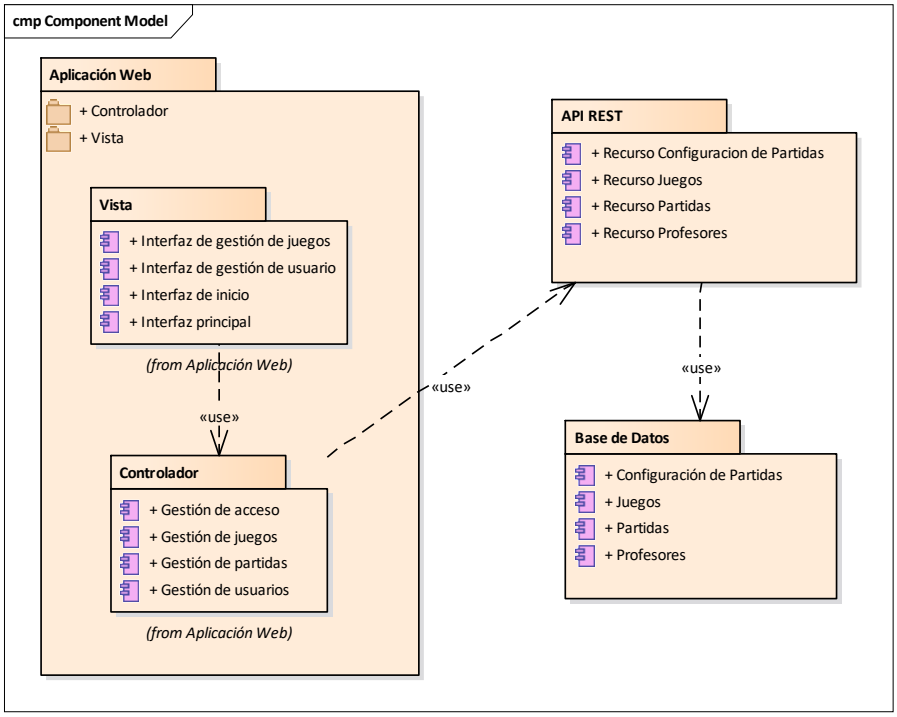
ARQUITECTURA Y DISEÑO

En este apartado se realizará la explicación detallada de la arquitectura y diseño de la aplicación. En primer lugar, se explicará de forma general cada uno de los componentes de la aplicación y la interacción entre ellos. Posteriormente se irán detallando en profundidad todos los componentes.

1. Introducción

Una aplicación web con Primefaces sigue una arquitectura MVC (Modelo-Vista-Controlador). En el diagrama de componentes UML 1 puede verse reflejada esta arquitectura mediante los componentes del proyecto:

- Vista: compuesta por todas las interfaces de interacción con el usuario de la Aplicación Web.
- Controlador: compuesto por todas las *Beans* de gestión de la Aplicación Web.
- Modelo: estaría compuesto por la base de datos y la API REST.



UML 1: CMP-001 Arquitectura

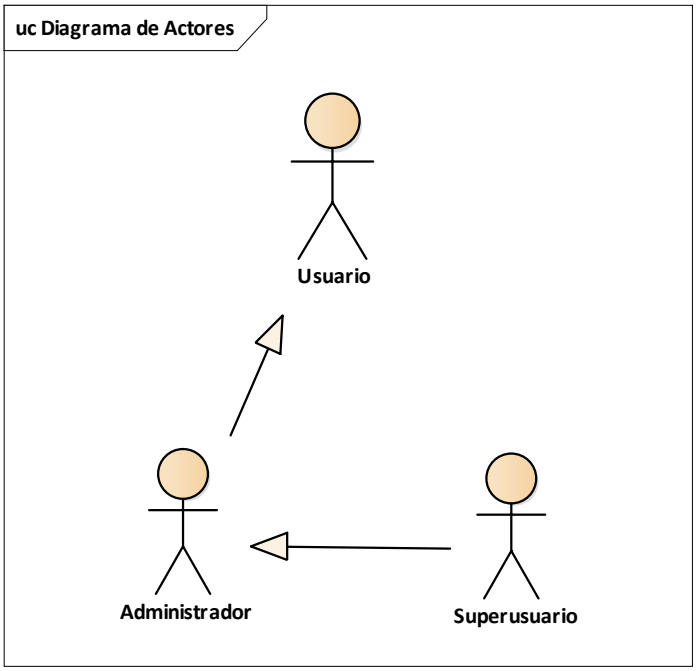
Antes de comenzar con el detalle de cada uno de los componentes, es necesario conocer los actores que intervendrán en el proyecto.

El único actor que interactuará con nuestra aplicación será el profesor, por tanto, este será nuestro único usuario. Sin embargo, el usuario puede tener diferentes roles que le permitirá tener control sobre diferentes gestiones de la aplicación:

- Usuario: gestión de sus propias plantillas y partidas.
- Administrador: gestión de juegos.
- Superusuario: gestión de otros usuarios.

Como puede verse en el diagrama UML 2, el administrador extiende del usuario genérico y el

superusuario del administrador. Por tanto, el administrador puede hacer gestiones como usuario y administrador y el superusuario además de sus gestiones también podrá hacer las que se les permite a los otros dos.



UML 2: CU-001 Actores

2. Modelo E/R de la Base de Datos

El primer componente que vamos a detallar es la base de datos. En la base de datos de esta aplicación necesitaremos almacenar la información de los usuarios registrados (profesores), los juegos disponibles, las configuraciones de partidas (plantillas) que vayan creando los usuarios y las partidas que éstos vayan iniciando. Por tanto, tendremos una base de datos relacional creada con el Gestor de Bases de Datos Postgresql.

En la Figura 20 se puede ver el modelo E/R (Entidad/Relación) que representa la información que se maneja en la aplicación y la relación existente entre ella.

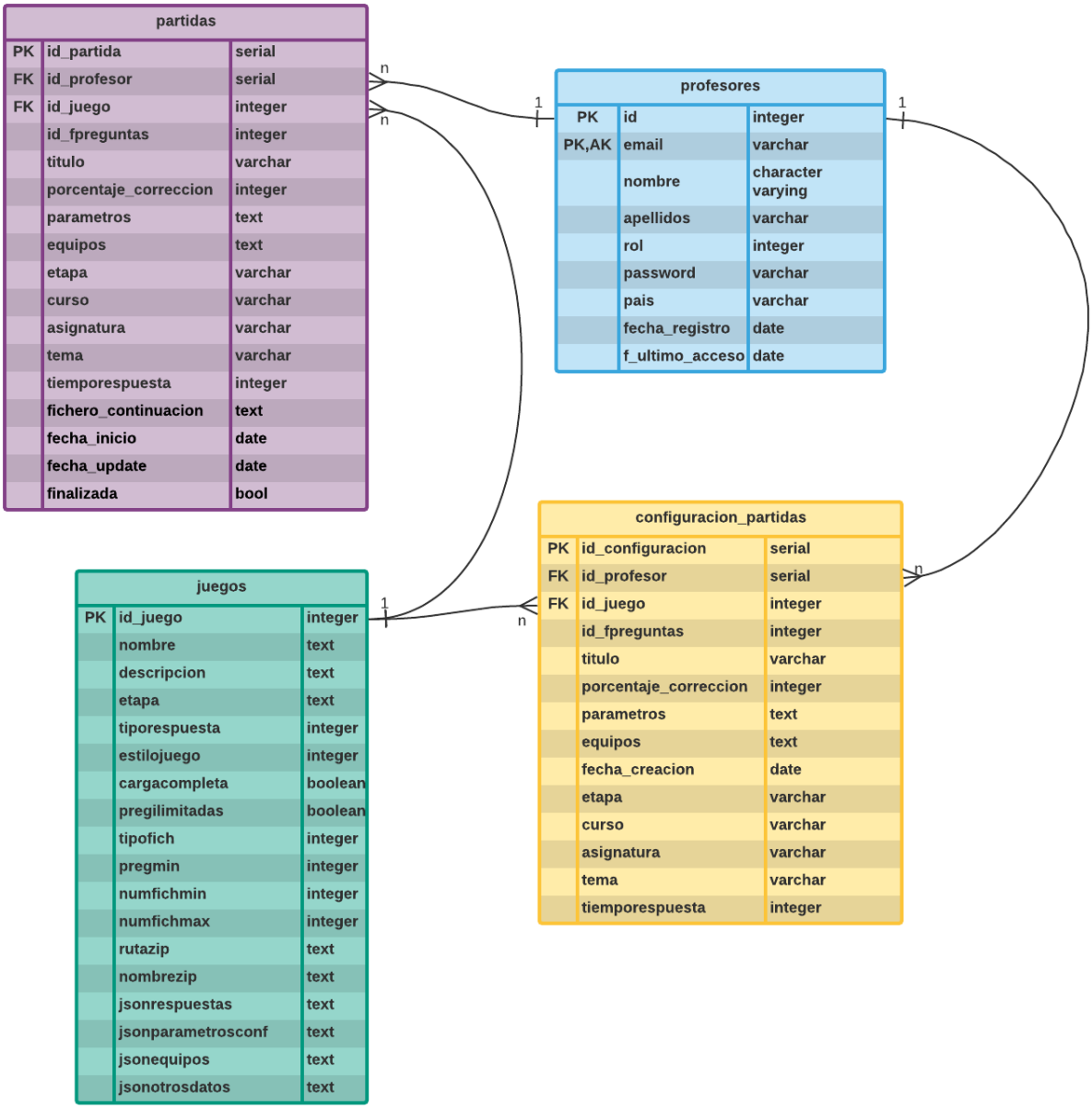


Figura 20: Modelo E/R de la BBDD

Tabla Juegos

En la tabla *juegos* se almacena toda la información de los juegos existentes en la plataforma.

En la Tabla 1 se recogen todos sus campos agrupados por funcionalidad junto con una breve descripción.

Clasificación	Campo	Tipo de dato	Descripción
Clave	id_juego	integer	Identificador único del juego (Primary Key)
Ubicación del juego	rutazip	text	Ruta donde se ubica el zip que contiene el juego
	nombrezip	text	Nombre del zip que contiene el juego
Información general del juego	nombre	text	Nombre con el que se mostrará el juego
	descripcion	text	Descripción asociada al juego
	etapa	text	Etapa educativa para la que está recomendado el juego GEN – General INF – Infantil PRI – Primaria SEC – Secundaria UNI - Universidad
	tiporespuesta	integer	Identificador del tipo de respuesta asociado al juego 0 – Sin respuesta 1 – Respuestas con opciones (A,B,C,...) 2 – Cifras 3 – Campos de texto 4 – Respuestas de relacionar
	estilojuego	integer	Identificador del estilo del juego 0 – Clásico 1 – Concurso TV 2 – Deporte 3 – Ideado por AJDA
Características generales de configuración	cargacompleta	boolean	Indica si el juego permite la carga completa del fichero de preguntas

	pregilimitadas	Boolean	Indica si el juego tiene límite de número de preguntas
	tipofich	integer	Tipo de fichero aceptado por el juego
	pregmin	integer	Número mínimo de preguntas que necesita el juego
	numfichmin	integer	Número mínimo de ficheros que necesita el juego
	numfichmax	integer	Número máximo de ficheros que acepta el juego
Características específicas de configuración	jsonparametrosconf	text	Parámetros específicos del juego
	jsonequipos	text	Parámetros de configuración de grupos y equipos que permite el juego.
	jsonrespuestas	text	Parámetros de configuración de la estructura de respuestas que espera el juego
	jsonotrosdatos	text	Parámetros de configuración de otros datos que necesita el juego

Tabla 1: Tabla *juegos*

Tabla Profesores

En la tabla *profesores* se almacena la información de cada uno de los profesores registrados en la plataforma.

En la Tabla 2 se recogen todos sus campos agrupados por funcionalidad junto con una breve descripción.

Clasificación	Campo	Tipo de dato	Descripción
Claves	id	Integer	Identificador único del profesor (clave primaria)
	email	varchar	Correo electrónico del profesor. Es la clave alternativa, pues es única, sin embargo, se puede modificar.

Información interna	password	varchar	Contraseña de acceso del usuario
	rol	integer	Identificador del rol del usuario. 0 – Usuario 1 – Administrador 2 – Superusuario
	fecha_registro	date	Fecha de registro del usuario
	f_ultimo_acceso	date	Fecha del último acceso del usuario
Información personal	nombre	varchar	Nombre del usuario
	apellidos	varchar	Apellidos del usuario
	pais	varchar	País de procedencia del usuario

Tabla 2: Tabla *profesores*

Tabla Configuración de Partidas

En la tabla *configuracion_partidas* se almacena la información de cada una de las plantillas que genera cada profesor (id_profesor). Para cada plantilla deberá seleccionarse un juego (id_juego), además, un mismo juego podrá estar configurado en diferentes plantillas.

En la Tabla 3 se recogen todos sus campos agrupados por funcionalidad junto con una breve descripción.

Clasificación	Campo	Tipo de dato	Descripción
Claves	id_configuracion	integer	Identificador único de la plantilla (Primary Key)
	id_profesor	integer	Identificador del profesor propietario de la plantilla (Foreign Key – Asociada a la tabla *profesores*)
	id_juego	integer	Identificador del juego seleccionado para la plantilla (Foreign Key – Asociada a la tabla *juegos*)
Información interna	fecha_creacion	date	Fecha en la que se crea la plantilla
Información	titulo	varchar	Nombre de la plantilla

	etapa	varchar	Etapa educativa hacia la que está enfocada la plantilla
	curso	varchar	Curso académico hacia el que está enfocado la plantilla
	asignatura	varchar	Asignatura relacionada a las preguntas del juego
	tema	varchar	Tema de la asignatura hacia el que se orientan las preguntas del juego
Configuración	id_fpreguntas	integer	Identificador del fichero de preguntas seleccionado para la plantilla
	porcentaje_correccion	integer	Porcentaje de respuestas correctas en un grupo/equipo para dar por correcta una pregunta
	tiempo_respuesta	integer	Tiempo máximo para responder a cada pregunta
	parametros	text	Configuración de los parámetros específicos del juego
	equipos	text	Configuración de los grupos y equipos (número de grupos, nombres, etc)

Tabla 3: Tabla *configuracion_partidas*

Tabla Partidas

En la tabla *partidas* se almacena la información de cada una de las partidas iniciadas por los profesores. Esta tabla es prácticamente una copia de la Configuración de Partida, pero añadiendo campos para determinar el estado y poder continuar las partidas. Esto es así para que las plantillas puedan editarse sin que afecte a las partidas ya iniciadas.

En la Tabla 4 se recogen todos sus campos agrupados por funcionalidad junto con una breve descripción.

Clasificación	Campo	Tipo de dato	Descripción
Claves	id_partida	serial	Identificador único de la partida (Primary Key)
	id_profesor	serial	Identificador del profesor propietario de la partida

			(Foreign Key – Asociada a la tabla *profesores*)
	id_juego	integer	Identificador del juego seleccionado para la partida (Foreign Key – Asociada a la tabla *juegos*)
Información interna	fecha_inicio	date	Fecha en la que se crea la partida
	fecha_update	date	Fecha de la última actualización de la partida
	finalizada	bool	Indica si la partida está finalizada
Información	titulo	varchar	Nombre de la partida
	etapa	varchar	Etapa educativa hacia la que está enfocada la partida
	curso	varchar	Curso académico hacia el que está enfocado la partida
	asignatura	varchar	Asignatura relacionada a las preguntas del juego
	tema	varchar	Tema de la asignatura hacia el que se orientan las preguntas del juego
Configuración	id_fpreguntas	integer	Identificador del fichero de preguntas seleccionado para la plantilla
	porcentaje_correccion	integer	Porcentaje de respuestas correctas en un grupo/equipo para dar por correcta una pregunta
	tiempo_respuesta	integer	Tiempo máximo para responder a cada pregunta
	parametros	text	Configuración de los parámetros específicos del juego
	equipos	text	Configuración de los grupos y equipos

			(número de grupos, nombres, etc)
	fichero_continuacion	text	Fichero que guarda el estado de una partida para su posterior continuación

Tabla 4: Tabla *partidas*

3. API REST

El componente API REST de nuestra aplicación es la interfaz de acceso al modelo de datos, explicado en el apartado anterior.

Por tanto, podemos dividir los recursos en cuatro grupos, tal y como se puede ver en la Figura 21:

- **Configuración de partidas:** Recursos para la gestión de Configuraciones de Partidas.
- **Juegos:** Recursos para la gestión de juegos.
- **Partidas:** Recursos para la gestión de partidas.
- **Usuarios:** Recursos para la gestión de usuarios.

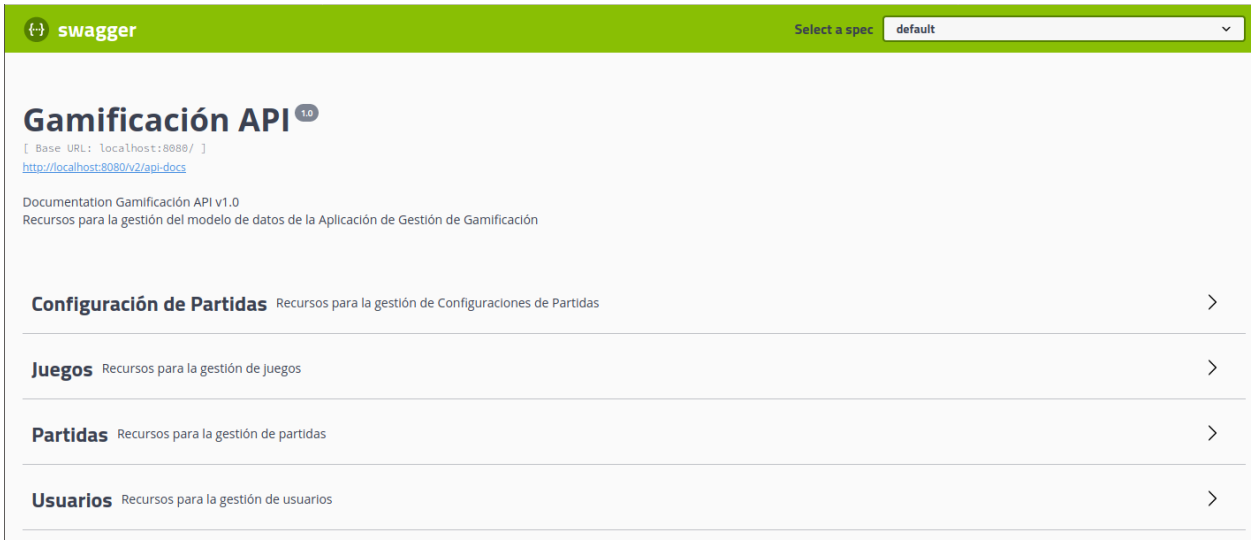


Figura 21: API REST - Agrupación de recursos

A continuación, se irán detallando todos los recursos disponibles para cada uno de estos grupos:

Juegos

Los diferentes recursos que podemos ver en la Figura 22, manejarán el tipo de objeto Juego, el cual se corresponde con el modelo de datos visto en la Tabla 1 del apartado anterior.

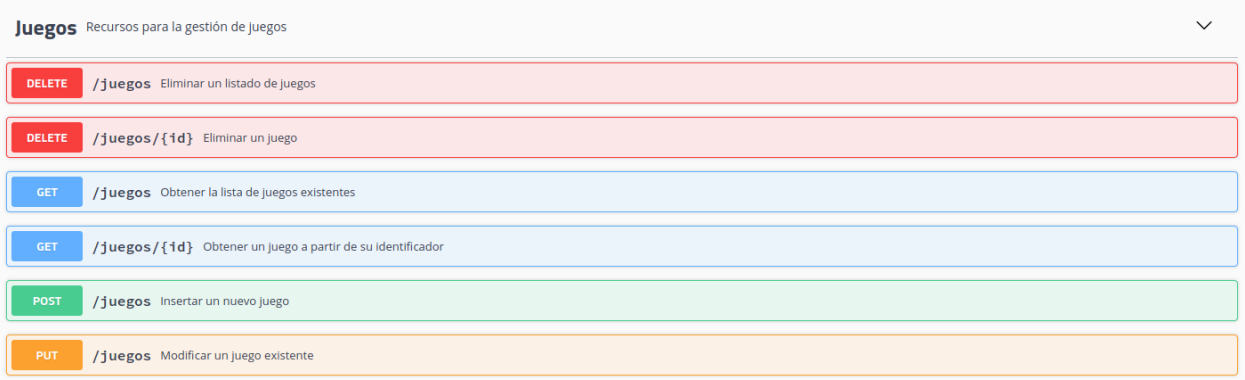


Figura 22: API REST - Recursos de Juegos

- **GET**
  - **/juegos:** Obtener la lista de juegos existentes.
    - **Petición:** Es una petición simple al recurso, sin ningún parámetro de entrada.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será la lista de objetos de tipo Juego con los juegos existentes en la base de datos.
  - **/juegos/{id}:** Obtener un juego a partir de su identificador.
    - **Petición:** El único parámetro de entrada será el id, que se corresponderá con el identificador del juego del cual queremos obtener la información. Este parámetro es de tipo *path* y deberá ser un número entero.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será el objeto de tipo Juego correspondiente al identificador de entrada.
- **POST**
  - **/juegos:** Insertar un nuevo juego**.**
    - **Petición:** El parámetro de entrada debe ser un objeto de tipo Juego
    - **Respuesta correcta (200 OK):** body de respuesta vacío. Indica que el insert en la base de datos se ha realizado satisfactoriamente.
- **PUT**
  - **/juegos:** Modificar un juego existente.
    - **Petición:** El parámetro de entrada debe ser un objeto de tipo Juego
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que el update en la base de datos se ha realizado satisfactoriamente.

- **DELETE**
  - **/juegos:** Elimina un listado de juegos.
    - **Petición:** El parámetro de entrada es un listado de enteros. Cada uno de esos enteros, se corresponde al identificador del juego que se desea eliminar.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si los recursos existían se han eliminado y si no existían no se ha hecho nada.
  - **/juegos/{id}:** Elimina un juego.
    - **Petición:** El único parámetro de entrada será el id, que se corresponderá con el identificador del juego el cual queremos eliminar. Este parámetro es de tipo *path* y

deberá ser un número entero.

- **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si el recurso existía se ha eliminado y si no existía no se ha hecho nada.

Usuarios

Los diferentes recursos que podemos ver en la Figura 23, manejarán el tipo de objeto Profesor, el cual se corresponde con el modelo de datos visto en la Tabla 2 del apartado anterior.

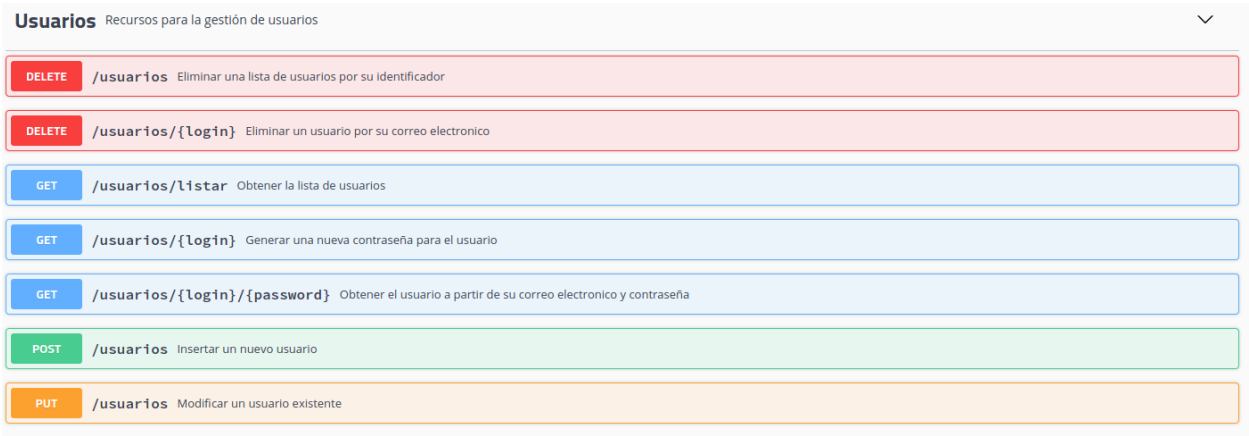


Figura 23: API REST - Recursos de Usuarios

- **GET**
  - **/usuarios:** Obtener la lista de usuarios.
    - **Petición**: Es una petición simple al recurso, sin ningún parámetro de entrada.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será la lista de objetos de tipo Profesor con los usuarios existentes en la base de datos.
  - **/usuarios/{login}:** Generar una nueva contraseña para el usuario.
    - **Petición:** El único parámetro de entrada será el login, que se corresponderá con el correo electrónico asociado al usuario para el cual queremos generar una contraseña automática. Este parámetro es de tipo *path* y deberá ser un string.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será el objeto de tipo Profesor correspondiente al login de entrada. En este objeto se devolverá como contraseña una nueva autogenerada, la cual se habrá actualizado en la base de datos.
  - **/usuarios/{login}/{password}:** Obtener el usuario a partir de su correo electrónico y contraseña.
    - **Petición:** Los parámetros de entrada será, el login, que se corresponderá con el correo electrónico del usuario, y la password, que debe ser la contraseña de acceso del usuario. Estos parámetros son de tipo *path* y deberán ser ambos String.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será el objeto de tipo Profesor correspondiente al login de entrada. Esta respuesta únicamente se dará en caso de que la contraseña se corresponda con la registrada para el login dado.
- **POST**
  - **/usuarios:** Insertar un nuevo usuario.

        - **Petición:** El parámetro de entrada debe ser un objeto de tipo Profesor.
        - **Respuesta correcta (200 OK):** body de respuesta vacío. Indica que el insert en la base de datos se ha realizado satisfactoriamente.

- **PUT**
    - **/usuarios:** Modificar un usuario existente.
        - **Petición:** El parámetro de entrada debe ser un objeto de tipo Profesor.
        - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que el update en la base de datos se ha realizado satisfactoriamente.

- **DELETE**
    - **/usuarios:** Elimina un listado de usuarios.
        - Petición: El parámetro de entrada es un listado de enteros. Cada uno se esos enteros, se corresponde al identificador del usuario que se desea eliminar.
        - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si los recursos existían se han eliminado y si no existían no se ha hecho nada.
    - **/usuarios/{login}:** Elimina un usuario por su correo electrónico.
        - **Petición:** El único parámetro de entrada será el login, que se corresponderá con el correo electrónico del usuario el cual queremos eliminar. Este parámetro es de tipo *path* y deberá ser un número entero.
        - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si el recurso existía se ha eliminado y si no existía no se ha hecho nada.

Configuración de Partidas

Los diferentes recursos que podemos ver en la Figura 24, manejarán el tipo de objeto ConfPartida, el cual se corresponde con el modelo de datos visto en la Tabla 3 del apartado anterior.

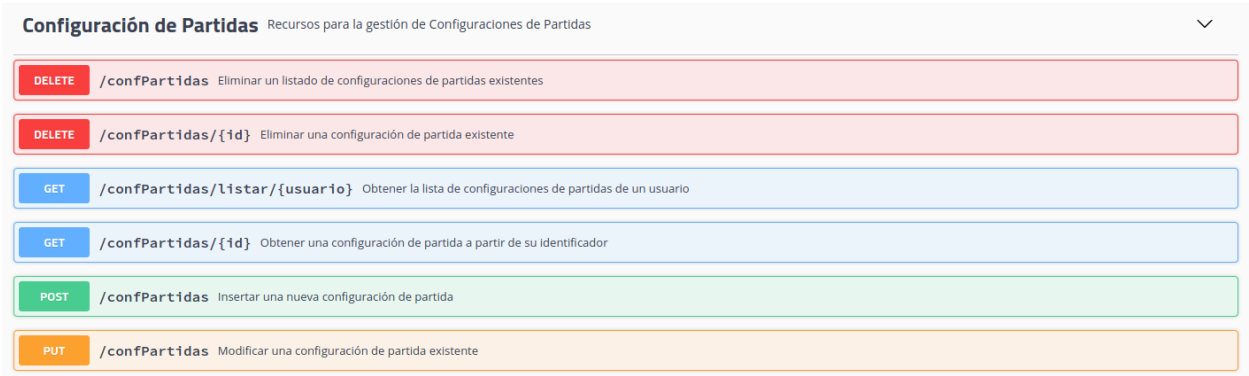


Figura 24: API REST - Recursos de Configuración de Partidas

- **GET**
    - **/confPartidas/listar/{usuario}:** Obtener la lista de configuraciones de partidas de un usuario.
        - **Petición**: El único parámetro de entrada será el *usuario*, que se corresponderá con el correo electrónico asociado al usuario para el cual queremos consultar sus

configuraciones de partida. Este parámetro es de tipo *path* y deberá ser un string.

- **Respuesta correcta (200 OK)**: el body de la respuesta será la lista de objetos de tipo ConfPartida con los juegos existentes en la base de datos.

- **/confPartidas/{id}:** Obtener una configuración de partida a partir de su identificador.
  - **Petición:** El único parámetro de entrada será el id, que se corresponderá con el identificador de la configuración de partida de la cual queremos obtener la información. Este parámetro es de tipo *path* y deberá ser un número entero.
  - **Respuesta correcta (200 OK)**: el body de la respuesta será el objeto de tipo ConfPartida correspondiente al identificador de entrada.

- **POST**
  - **/confPartidas:** Insertar una nueva configuración de partida**.**
    - **Petición:** El parámetro de entrada debe ser un objeto de tipo ConfPartida.
    - **Respuesta correcta (200 OK):** body de respuesta vacío. Indica que el insert en la base de datos se ha realizado satisfactoriamente.

- **PUT**
  - **/confPartidas:** Modificar una configuración de partida existente.
    - **Petición:** El parámetro de entrada debe ser un objeto de tipo ConfPartida.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que el update en la base de datos se ha realizado satisfactoriamente.

- **DELETE**
  - **/confPartidas:** Eliminar un listado dado de configuraciones de partidas existentes.
    - **Petición**: El parámetro de entrada es un listado de enteros. Cada uno se esos enteros, se corresponde al identificador de la configuración de partida que se desea eliminar.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si los recursos existían se han eliminado y si no existían no se ha hecho nada.
  - **/confPartidas/{id}:** Eliminar una configuración de partida existente.
    - **Petición:** El único parámetro de entrada será el id, que se corresponderá con el identificador de la configuración de partida la cual queremos eliminar. Este parámetro es de tipo *path* y deberá ser un número entero.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si el recurso existía se ha eliminado y si no existía no se ha hecho nada.

Partidas

Los diferentes recursos que podemos ver en la Figura 25, manejarán el tipo de objeto Partida, el cual se corresponde con el modelo de datos visto en la Tabla 4 del apartado anterior.

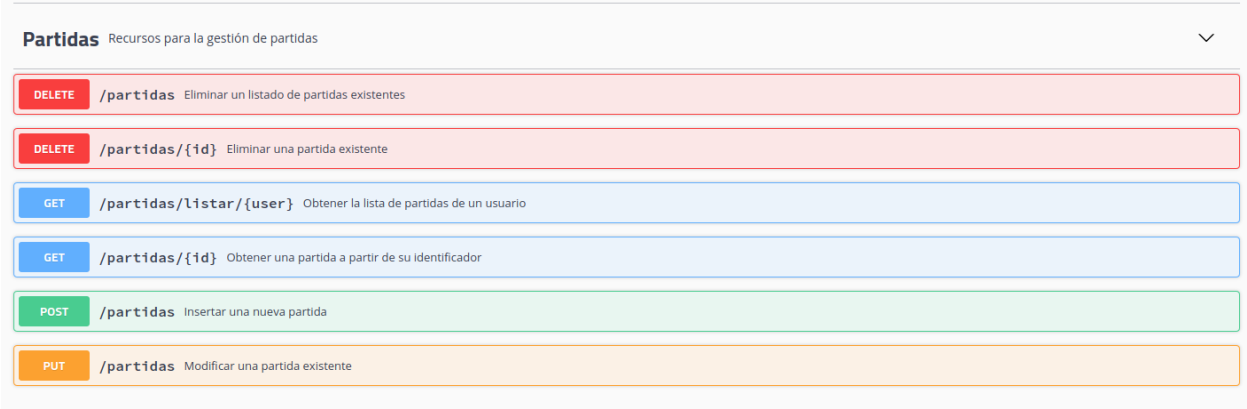


Figura 25: API REST - Recursos de Partidas

- **GET**
  - **/partidas/listar/{usuario}:** Obtener la lista de partidas de un usuario.
    - **Petición**: El único parámetro de entrada será el *usuario*, que se corresponderá con el correo electrónico asociado al usuario para el cual queremos consultar sus partidas. Este parámetro es de tipo *path* y deberá ser un string.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será la lista de objetos de tipo Partida con los juegos existentes en la base de datos.
  - **/partidas/{id}:** Obtener una configuración de partida a partir de su identificador.
    - **Petición:** El único parámetro de entrada será el id, que se corresponderá con el identificador de la partida de la cual queremos obtener la información. Este parámetro es de tipo *path* y deberá ser un número entero.
    - **Respuesta correcta (200 OK)**: el body de la respuesta será el objeto de tipo Partida correspondiente al identificador de entrada.
- **POST**
  - **/partidas:** Insertar una nueva partida**.**
    - **Petición:** El parámetro de entrada debe ser un objeto de tipo Partida.
    - **Respuesta correcta (200 OK):** body de respuesta vacío. Indica que el insert en la base de datos se ha realizado satisfactoriamente.
- **PUT**
  - **/ partidas:** Modificar una partida existente.
    - **Petición:** El parámetro de entrada debe ser un objeto de tipo Partida.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que el update en la base de datos se ha realizado satisfactoriamente.
- **DELETE**
  - **/ partidas:** Eliminar un listado de partidas existentes.
    - **Petición**: El parámetro de entrada es un listado de enteros. Cada uno se esos enteros, se corresponde al identificador de la partida que se desea eliminar.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si los recursos existían se han eliminado y si no existían no se ha hecho nada.

- **/ partidas /{id}:** Eliminar una configuración de partida existente.
    - **Petición:** El único parámetro de entrada será el id, que se corresponderá con el identificador de la partida la cual queremos eliminar. Este parámetro es de tipo *path* y deberá ser un número entero.
    - **Respuesta correcta (200 OK):** sin body de respuesta. Indica que la petición se ha procesado correctamente, es decir, si el recurso existía se ha eliminado y si no existía no se ha hecho nada.

A nivel general, para todos los recursos, la API puede devolver como respuesta a las peticiones principalmente los siguientes errores:

- **400 Bad Request:** se devuelve cuando los parámetros de entrada de la petición son incorrectos, es decir, cuando no son del tipo de dato o estructura esperada.
- **500 Internal Server Error**: se devuelve cuando se produce un error inesperado en el servidor y no se puede procesar la petición.

4. Aplicación Web

A continuación, se detallará el último componente de nuestro proyecto, la Aplicación Web. A partir de las diferentes interfaces a las que puede acceder el usuario, iremos detallando todos los casos de uso que ofrece cada interfaz.

Interfaz de inicio

La interfaz de inicio es la interfaz de acceso a la aplicación. El resto de las interfaces son inaccesibles sin iniciar sesión previamente, lo cual es posible a través de esta interfaz, la cual se muestra en la Figura 26:

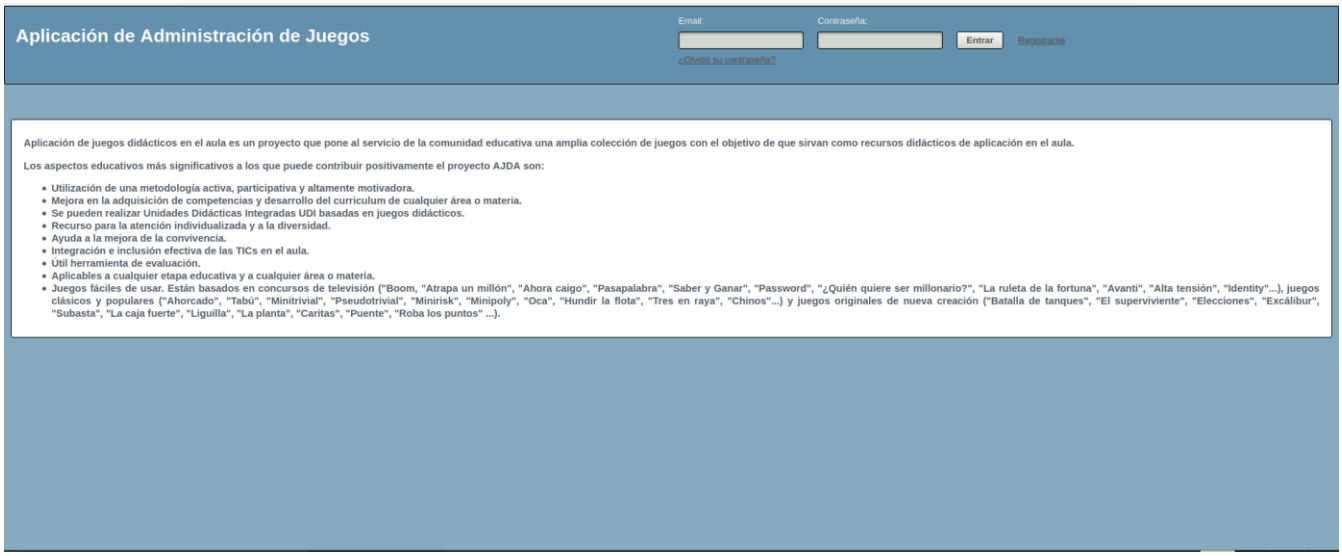
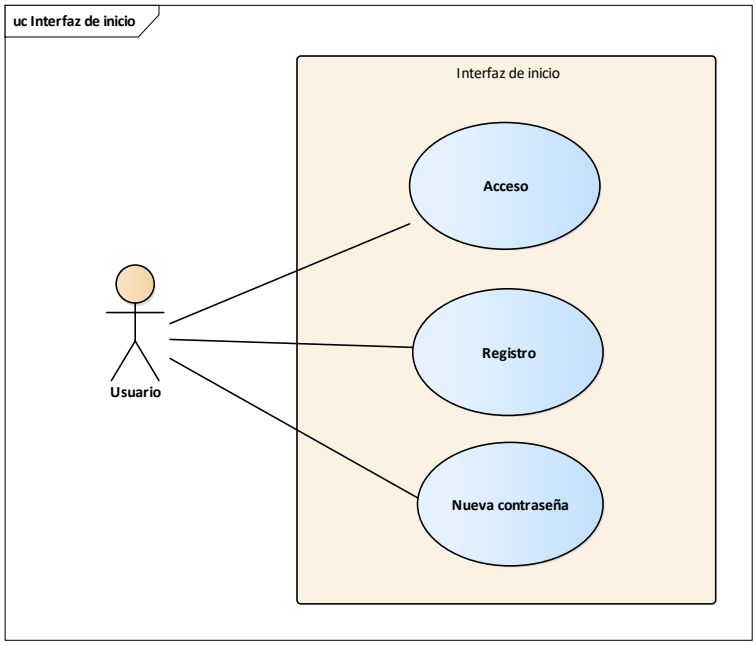


Figura 26: Interfaz de inicio

Los diferentes casos de uso que permite esta interfaz se ven reflejados en la figura UML 3:



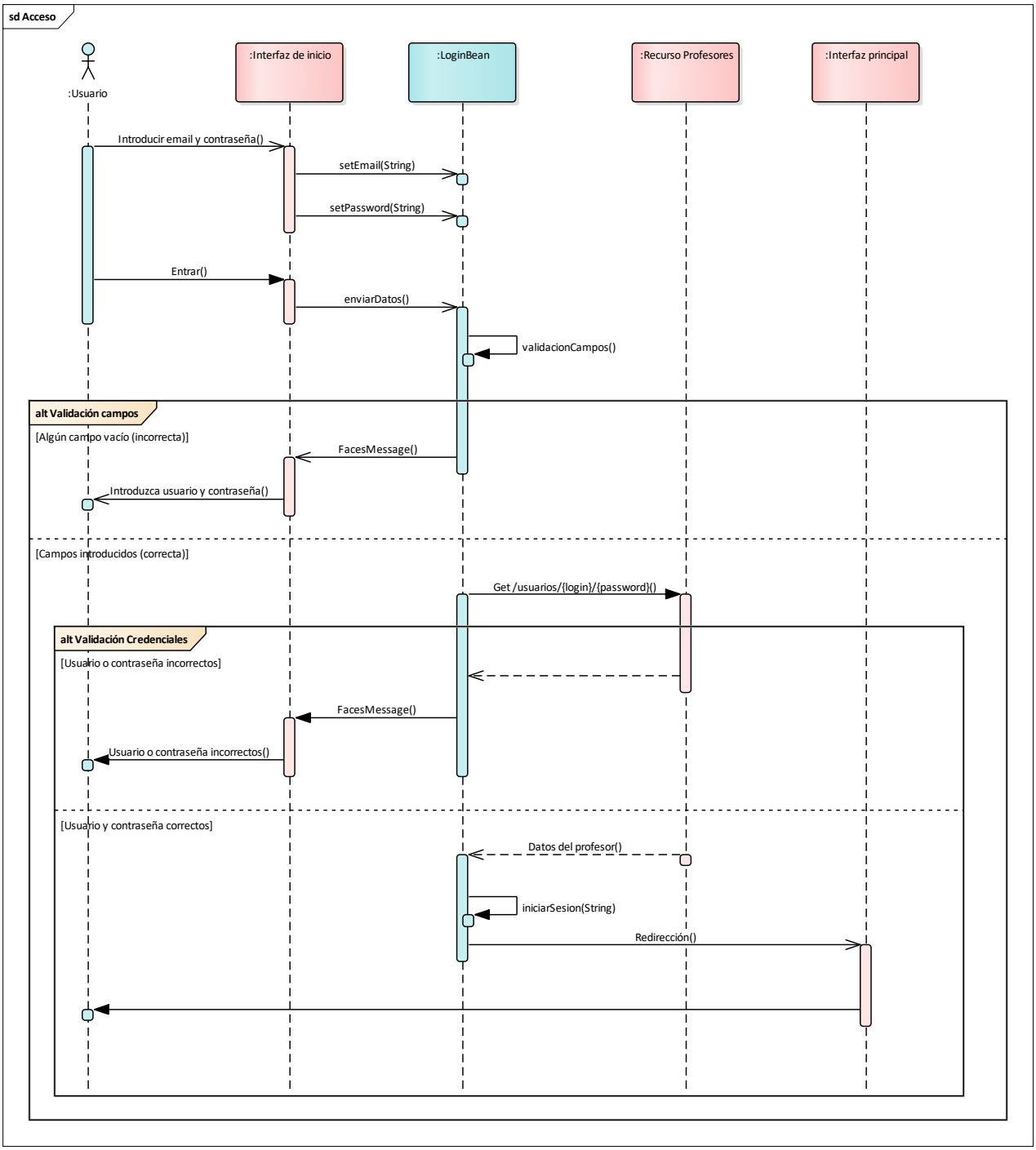
UML 3: CU-002 Interfaz de inicio

Desde la interfaz de inicio, el usuario, sin importar el rol, puede interactuar con los casos de uso Acceso, Registro o Nueva contraseña.

- **Acceso:** permite el acceso a la aplicación mediante el uso de un usuario (email) y contraseña.
- **Registro:** permite crear un usuario genérico para la aplicación. El usuario creado, por defecto, tendrá rol de *usuario*.
- **Nueva contraseña:** permite autogenerar una nueva contraseña para un usuario en caso de haberla olvidado.

A continuación, se detallarán los diferentes casos de usos:

Acceso



UML 4: SEQ-001 Acceso

En el diagrama UML 4 se detalla el caso de uso *Acceso.*

Esta secuencia se inicia cuando el usuario introduce su usuario y contraseña y pulsa sobre el botón *Entrar* (ver Figura 26). En ese momento, toma el control el método *enviarDatos()* del bean *LoginBean*. En este método, se siguen cuatro pasos:

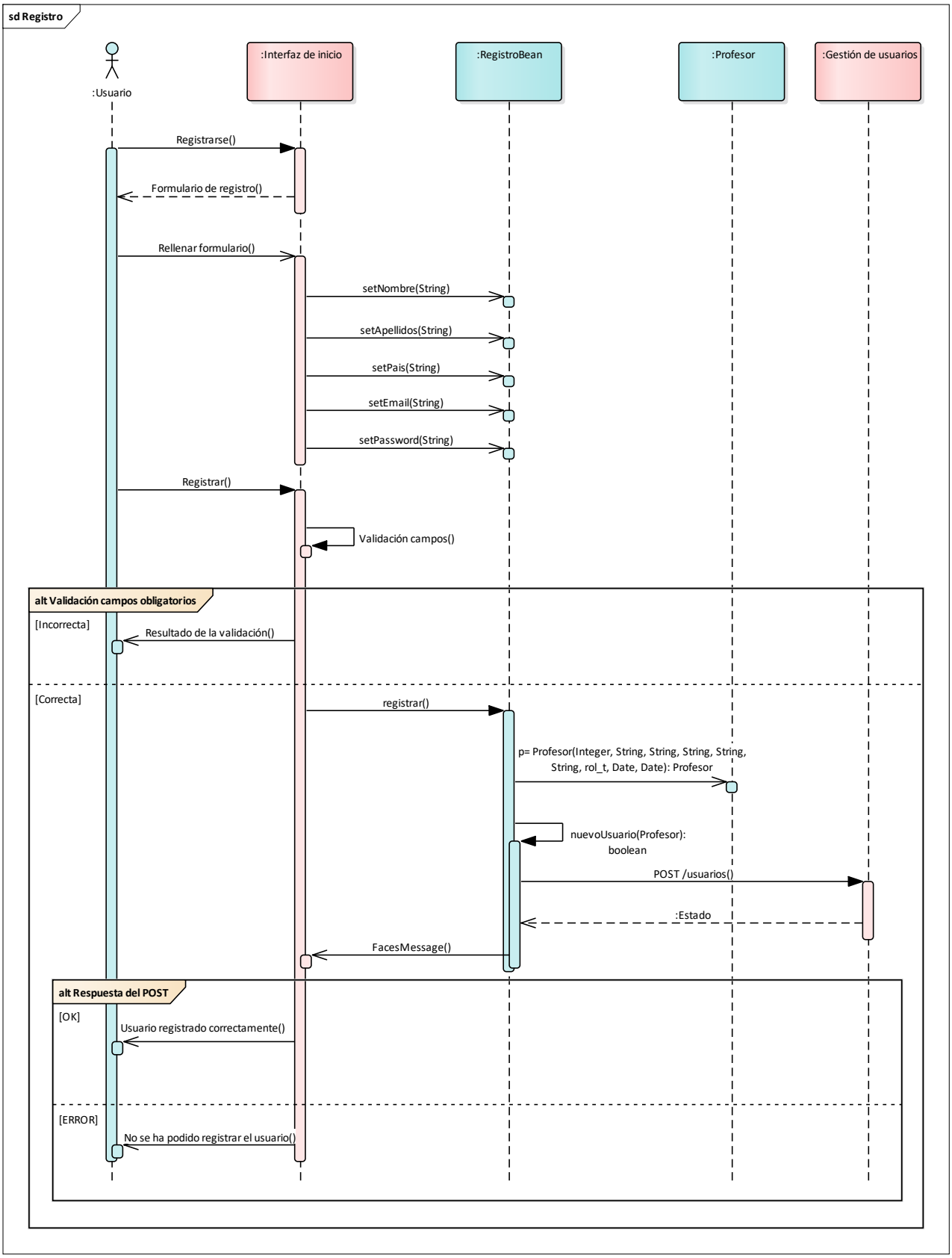
1. **Validación de campos:** se comprueba que los campos usuario y contraseña no estén vacíos.
2. **Enviar credenciales:** se realiza una petición GET al recurso /usuarios/{login}/{password} del

componente API REST. Esta petición devuelve un JSON con los datos del usuario, siempre y cuando el email y contraseña sean correctos. En caso contrario no devuelve ninguna información.

3. **Iniciar sesión:** a la sesión HTTP existente se le añade el atributo profesor cuyo valor será el recurso obtenido en la petición GET.
4. **Redirección a la Interfaz Principal:** se realiza la redirección a la interfaz principal de formar que el usuario ya se encuentra dentro de la aplicación.

Estos cuatro pasos sólo se realizan en caso de que se pasen tanto la validación de campos como la validación de credenciales. En caso de que alguna de estas validaciones falle, se finaliza el flujo y se envía a la interfaz de acceso un mensaje correspondiente al error producido que se mostrará al usuario.

Registro



UML 5: SEQ-002 Registro

En el diagrama UML 5 se detalla el caso de uso *Registro*.

La secuencia se inicia cuando el usuario clicka sobre *Registrarse* en la Interfaz de inicio (Ver Figura 26). En ese momento se abre un dialogo con el formulario de registro necesario para crear el nuevo usuario (Figura 27).

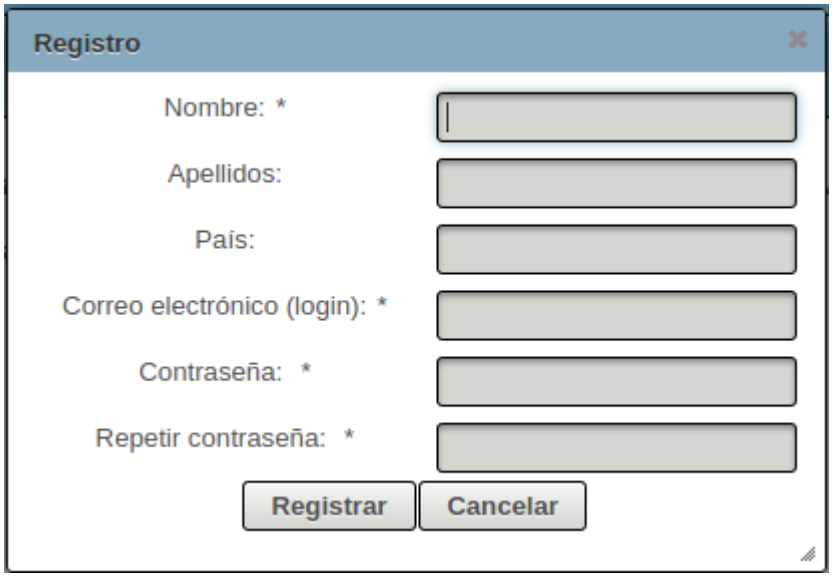


Figura 27: Formulario de registro de Profesor

El usuario deberá rellenar el formulario y pulsar sobre el botón registrar. Antes de iniciar el proceso de registro se realiza la validación del formulario. Se deben cumplir los siguientes requisitos para dar como correcta la validación:

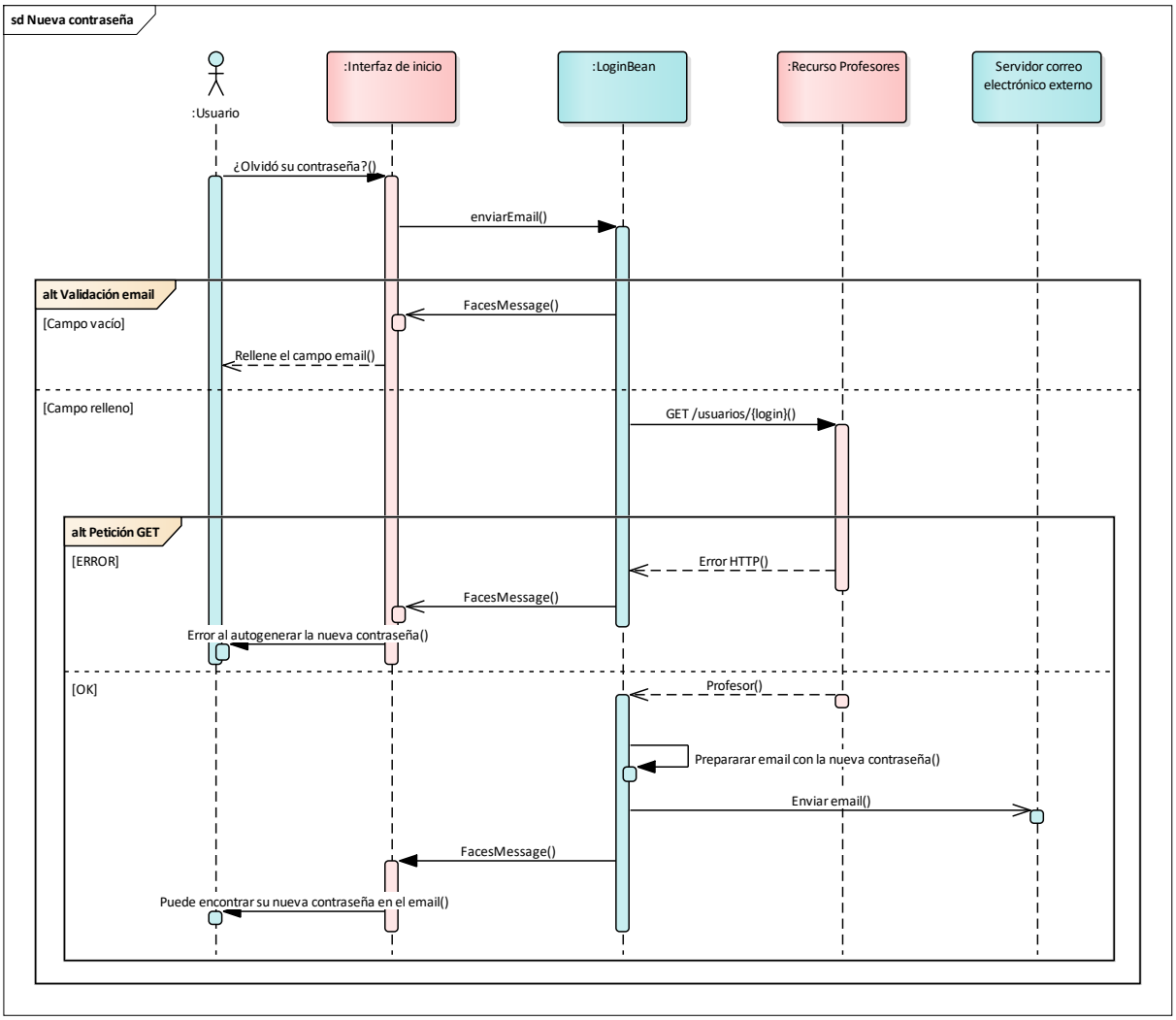
- Los campos requeridos no están vacíos: Nombre, Correo electrónico y Contraseña.
- Los campos Contraseña y Repetir contraseña coinciden.
- La contraseña tiene como mínimo 8 caracteres.

Si falla alguno de los requisitos se notificará al usuario mediante FacesMessages.

Por el contrario, si la validación es correcta se cede el control al bean *RegistroBean* a través del método *registrar*. En este método se crea un objeto Profesor con los datos del formulario y rol, por defecto, USER y se hace uso del método *nuevoUsuario* para realizar la petición POST al recurso /usuarios, el cual crea un usuario con los datos establecidos.

Finalmente, desde el método *nuevoUsuario* se informa al usuario si el registro ha ido bien o no.

Nueva contraseña



UML 6: SEQ-003 Nueva contraseña

En el diagrama UML 6 se detalla el caso de uso *Nueva contraseña*. Este caso de uso permite al usuario generar de forma automática una nueva contraseña de acceso.

Esta secuencia se inicia en el momento en el que el usuario clica sobre *¿Olvidó su contraseña?* La interfaz de inicio cede inmediatamente el control al bean *LoginBean* usando el método *enviarEmail*.

El primer paso que se realiza es la validación del campo email. En caso de estar vacío, no se puede autogenerar la contraseña de forma que se informa al usuario y finaliza el flujo. Por el contrario, si el campo está relleno, se lleva a cabo la petición GET al recurso /usuarios/{login}. Cuando se realiza esta petición sin indicar contraseña, el servicio REST te devuelve el recurso del usuario indicado, pero con una nueva contraseña autogenerada.

Si la petición se realiza correctamente, se prepara el email indicando la nueva contraseña y se envía. Una vez realizado el envío se comunica al usuario que puede encontrar la nueva contraseña en su bandeja de correo electrónico.

Por el contrario, si se produce algún error durante la petición GET se finaliza el flujo y se informa al usuario.

Interfaz principal

La interfaz principal está pensada para que el usuario tenga accesible de forma rápida y visual las plantillas y partidas en curso del propio usuario, tal y como puede observarse en la Figura 28. De esta forma toda la gestión de estas se pueda realizar desde la pantalla principal.

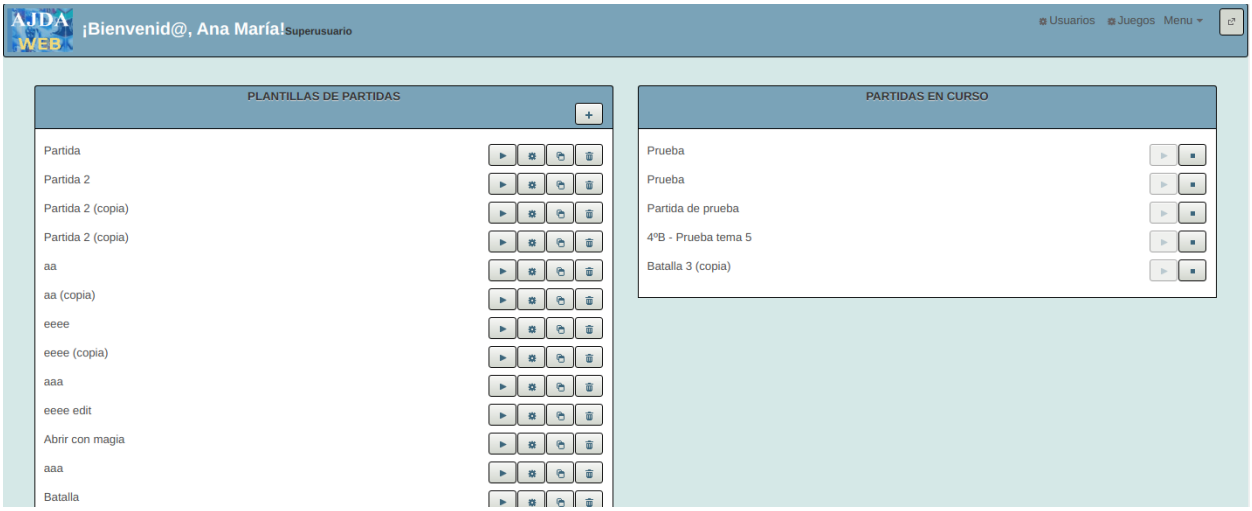
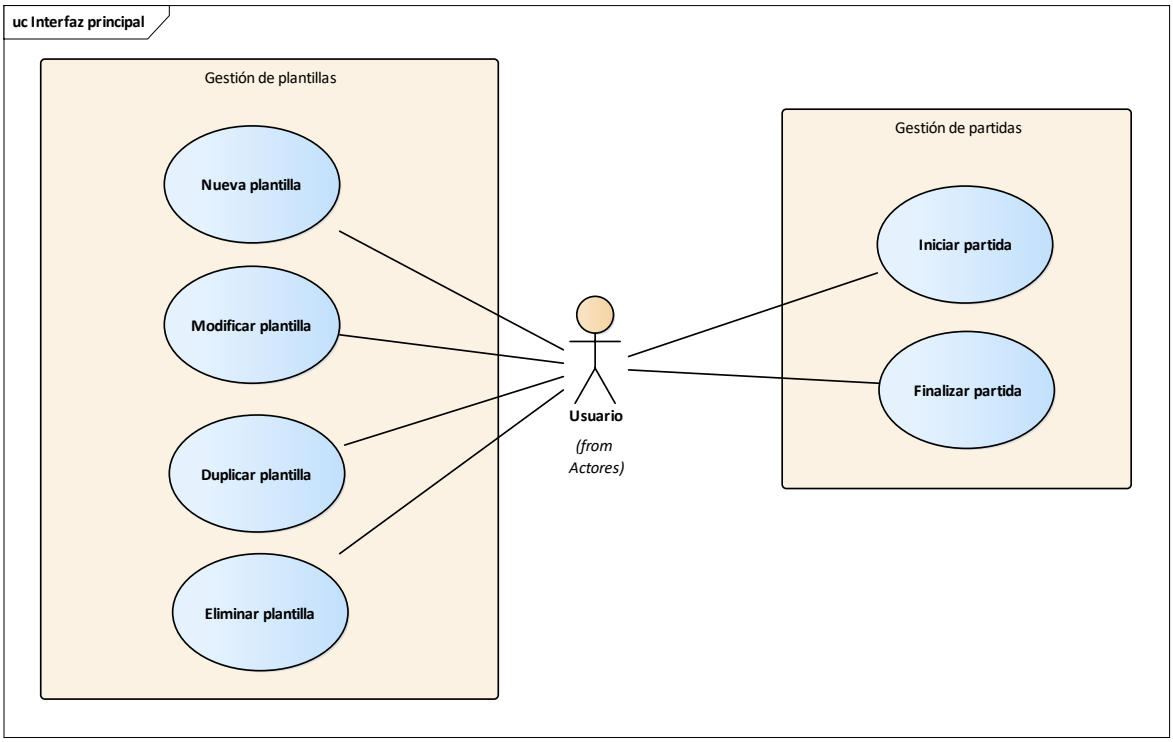


Figura 28: Interfaz principal

Por otro lado, esta interfaz es común para cualquier tipo de usuario. La diferencia es que, según el rol del usuario, tendrá acceso extra a la gestión de juegos y/o usuarios. Este acceso extra se habilitará en la cabecera de navegación, tal y como puede verse en la Figura 29. La cabecera de navegación, a su vez es común para las interfaces de gestión de juegos y usuarios.

Figura 29: Cabeceras de navegación

Los diferentes casos de uso que permite esta interfaz se ven reflejados en la figura UML 7. Como se puede observar, podemos dividir los casos de usos en dos perímetros, por un lado, los casos de uso relacionados con las plantillas/configuraciones de partidas y, por otro lado, los casos de uso para la gestión de las partidas en curso.



UML 7: CU-003 Interfaz principal

Con respecto a la gestión de plantillas/configuración de partidas tenemos los siguientes casos de uso:

- **Nueva plantilla:** permite la configuración y creación de una nueva plantilla/configuración de partida.
- **Modificar plantilla:** permite modificar la configuración de una plantilla/configuración de partida ya existente.
- **Duplicar plantilla:** permite crear una copia de una plantilla/configuración de partida ya existente.
- **Eliminar plantilla:** permite eliminar una plantilla/configuración de partida ya existente.

Por otro lado, los casos de uso para la gestión de las partidas son el punto de unión con la Aplicación web multiusuario, por tanto, son una simulación del estado en el que se quedarían las partidas si ambas aplicaciones estuvieran integradas:

- **Iniciar partida:** permite simular que se ha comenzado a jugar una nueva partida a raíz de una plantilla/configuración de partida ya existente y se ha dejado sin finalizar. Por tanto, este caso de uso creará una partida, la cual aparecerá en el listado de partidas en curso.
- **Finalizar partida:** permite simular que la partida se ha continuado y ha finalizado. Por tanto, la partida se modificará para cambiar su estado ha finalizado, de forma que ya no aparecerá en el listado de partidas en curso.

Nueva plantilla y Modificar plantilla

Los casos de uso *Nueva plantilla* y *Modificar plantilla* se explicarán en un mismo apartado, pues ambos casos comparten los diferentes procesos por los que pasa la secuencia. La diferencia es que en el caso de *Nueva plantilla* se trabaja con un objeto desde cero y en el caso de uso *Modificar plantill*a interviene un objeto que ya contiene información y, por tanto, las operaciones finales serán crear una plantilla y modificar una plantilla, respectivamente.

Estos dos casos de uso son diferentes al resto que encontraremos en esta explicación, pues pasará por diferentes fases antes de llegar a la fase final de guardado:

FASE 1: Abrir el diálogo de configuración de la plantilla.

Para ambos casos de uso, se mostrará el mismo diálogo que contendrá el formulario a rellenar con los datos de la plantilla.

Para el caso de uso Nueva plantilla, este diálogo se muestra al usuario cuando pulsa el botón Nueva plantilla. Este botón es el que se encuentra en la cabecera del listado de plantilla con icono de suma (Figura 28).

Por otro lado, para el caso de uso de Modifica plantilla, el usuario debe pulsar el botón Modificar de una plantilla en concreto del listado. El botón modificar es el tercer botón que aparece en cada elemento del listado, cuyo icono es una rueda dentada (Figura 28).

En la Figura 30 podemos ver el diálogo que se abrirá al usuario. El dialogo de la izquierda se corresponde al que se abre cuando se quiere crear una nueva plantilla, pues como se ve los campos están vacíos. Por el contrario, en el dialogo de la izquierda los campos ya están rellenos, pues se corresponde a una plantilla ya existente que se quiere modificar.

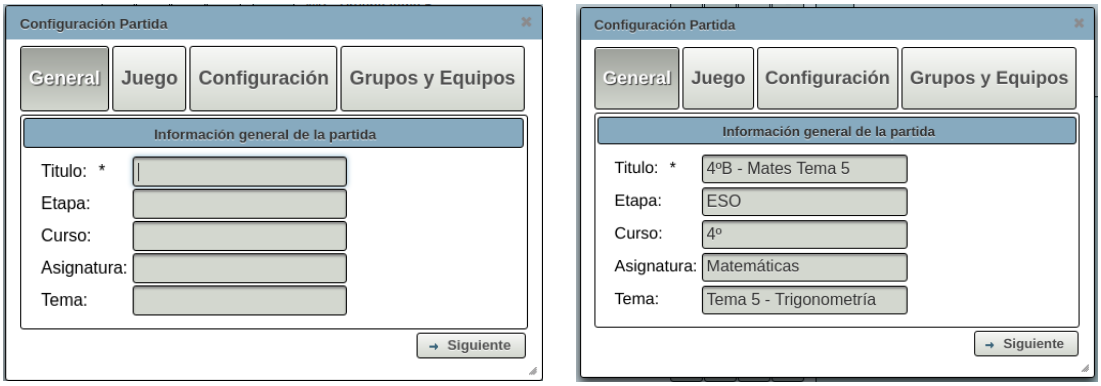


Figura 30: Dialogo de Configuración de partida

FASE 2: Completar la información del dialogo de la configuración de la plantilla.

Como se puede ver en la Figura 30, tenemos cuatro pestañas. En cada una de ella, tendremos diversos campos que completar antes de guardar la plantilla. Es un proceso secuencial, hasta que no completemos los campos obligatorios de cada formulario no podremos pasar a la siguiente pestaña.

Por otro lado, podemos comprobar que todos estos campos se corresponden con las columnas de la Tabla 3: Tabla *configuracion_partidas* del modelo E/R de la base de datos.

Las cuatro pestañas de la configuración de partida/plantilla son las siguientes:

- **General:** en este formulario encontramos la información básica de la partida, como se puede ver en la Figura 30. El único campo obligatorio en este apartado es el título de la plantilla, pues este título se utilizará para que el usuario pueda identificar la plantilla en el listado de plantillas. Esta validación se realiza de forma automática en la interfaz, marcando de rojo el campo en caso de estar vacío.
- **Juego:** en este paso lo único que hay que seleccionar es el juego, lo cual es obligatorio. Esta comprobación se realiza en el listener de cambio de pestaña cuando la siguiente es *Configuración*, en caso de que no se haya seleccionado se mantiene la misma pestaña y se indica en un message.

  En la Figura 31 vemos los dos posibles estados en los que se puede encontrar la pestaña.

  El diálogo de la izquierda es el diálogo cuando no hay ningún juego seleccionado. Tenemos, por un lado, el listado de juegos disponibles y por otro, un conjunto de filtros para poder acotar dicho listado. Cuando se selecciona un juego se procesa el método *onSelectJuego* del bean *configurarPartida,* en el cual se prepara la información para mostrar la información del juego, como se puede ver en el diálogo de la derecha de la Figura 31. La información que se muestra del juego es la descripción, una tabla con

la información más relevante, la carátula del juego y por último una captura de muestra del juego. Además, tenemos un botón para poder cambiar el juego seleccionado, que nos volvería a cargar el diálogo inicial de selección.

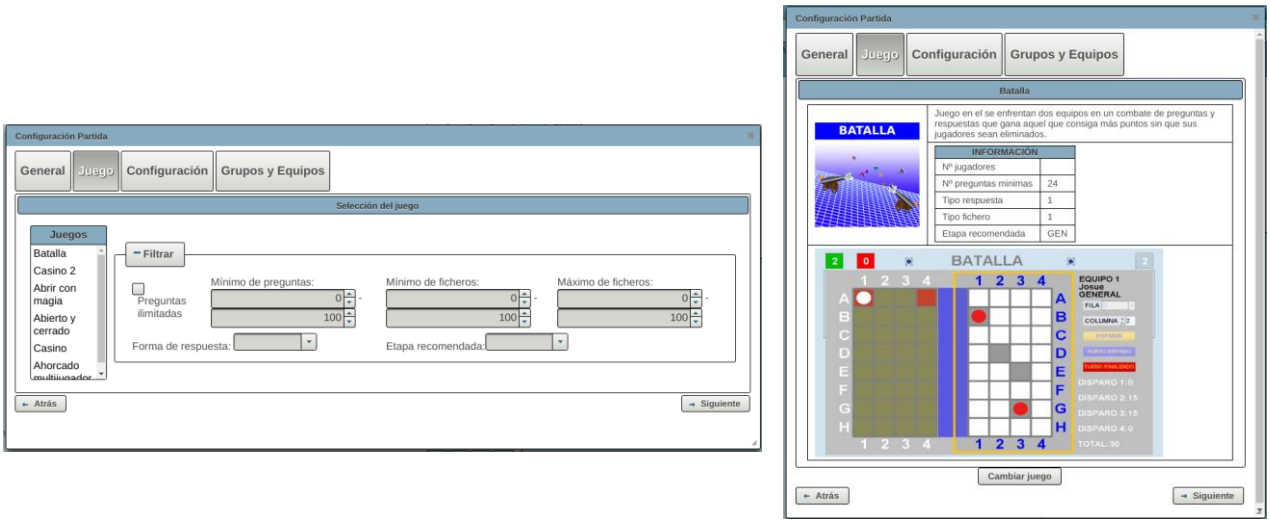


Figura 31: Dialogo de Configuración Partida – Pestaña Juego

- **Configuración:** en esta pestaña se realizará la configuración técnica de la partida, como podemos ver en la Figura 32. Por un lado, tenemos el Factor de corrección, que es el factor que se tomará como referencia para determinar si un grupo o equipo ha acertado la respuesta, y el Tiempo de respuesta, tiempo disponible para contestar cada pregunta. Por otro lado, también se deberá elegir el fichero de preguntas el cual contendrá las preguntas que usará el juego. Este campo es provisional, pues realmente es el punto de conexión con la Aplicación de gestión de ficheros de preguntas, de la cual se habló en la introducción de este documento. Por tanto, aunque se puede seleccionar un fichero, internamente no se realiza ningún procesado sobre él.

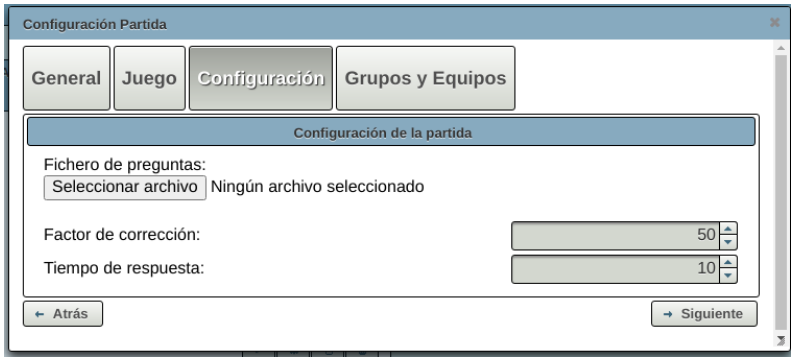


Figura 32: Dialogo de Configuración Partida – Pestaña Configuración

- **Grupos y Equipos:** es la última pestaña de configuración. En ella encontraremos un formulario generado a raíz del json *jsonequipos* asociado al juego seleccionado. Podemos encontrar tres casos diferentes en esta pestaña:
    - **Sin configuración de grupos y equipos:** existen juegos que no necesitan configuración de grupos y equipos y, por tanto, el juego tendrá el campo *jsonequipos* vacío. En este caso se mostrará el mensaje que se observa en la Figura 33.

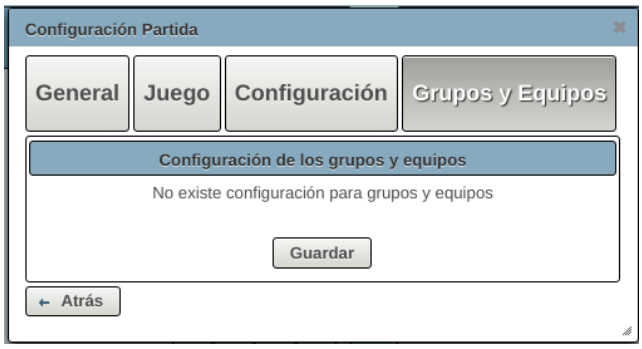


Figura 33: Dialogo de Configuración Partida – Pestaña Grupos y Equipos I

- **Configuración de grupos:** este caso se da para juegos que solo necesitan configuración en grupos. Tal y como se ve en la Figura 34, en estos casos tendremos un *spinner* para seleccionar el número de grupos que participará en la partida, la cantidad de grupos vendrá limitada por lo que permita el juego seleccionado. Una vez establecidos el número de grupos, se podrá personalizar el nombre de cada grupo.

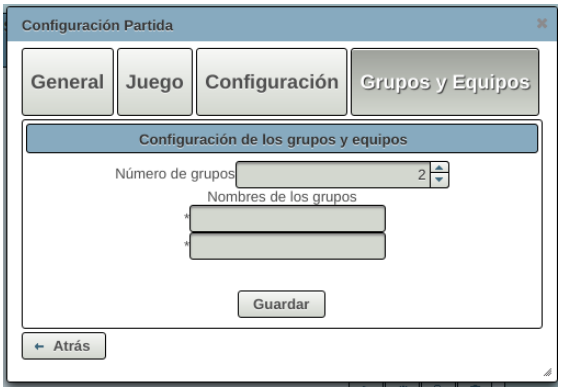


Figura 34: Dialogo de Configuración Partida – Pestaña Grupos y Equipos II

El formato de JSON que conlleva esta casuística se muestra en la Figura 35. Tenemos dos objetos principales: nombresGrupos y numeroGrupos. En general en ambos objetos tendremos *textoWeb* y *variablesJuego*, la primera sirve para personalizar el texto que aparecerá en el formulario y la segunda es un array dónde se guardarán los datos introducidos en el formulario, cada elemento será un par nombre-valor dónde el nombre es el nombre de la variable que utilizará el juego.

En el objeto *numeroGrupos*, se configura el número de grupos mínimo y máximo que acepta el juego, así como el valor por defecto que tendrá dicha variable. En este objeto, solo tendremos una variable de juego, en la que se guardará el número de grupos que seleccione el usuario.

En el objeto *nombresGrupos*, debemos configurar tantas *variablesJuego* como número de grupos máximos que acepte el juego. De forma, que solo se le dará valor a tantas variables como grupos se hayan seleccionado y en ellas se guardarán los nombres de los grupos.

```
{
   "grupos":{
      "nombresGrupos":{
         "textoWeb":"Nombre del grupo",
         "variablesJuego":[
               {
                "nombre":"jug1",
                "valor":""
               }
           ]
      },
      "numeroGrupos":{
         "textoWeb":"Número de grupos",
         "rango":{
            "valorMinimo":1,
            "valorMaximo":1,
            "valorDefecto":1
           },
        "variablesJuego":[{
           "nombre":"-",
           }]
         }
     }
}
```

Figura 35: Estructura *jsonequipos* - Configuración de grupos

- **Configuración de grupos organizados en equipos:** para esta casuística primero debemos elegir el número de equipos que habrá en la partida y luego cuántos grupos compondrá cada equipo. Como se puede ver en la Figura 36, será necesario elegir nombres tanto para los equipos como para cada grupo.

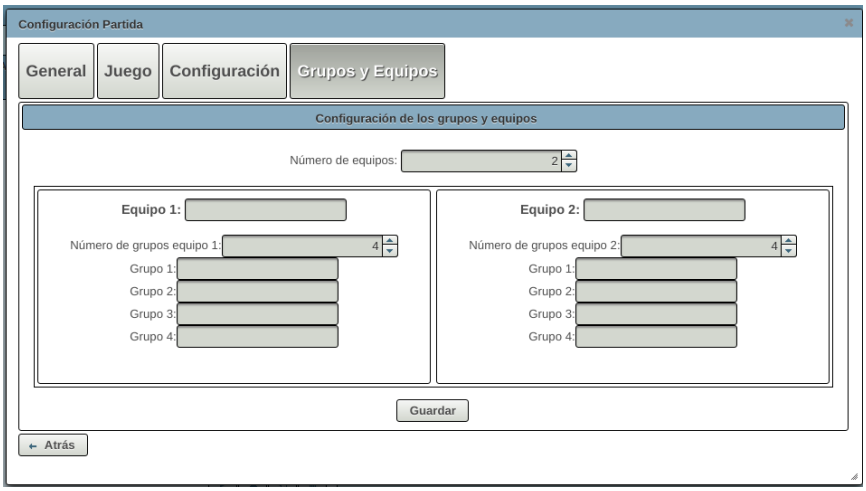


Figura 36: Dialogo de Configuración Partida – Pestaña Grupos y Equipos III

La estructura general del JSON para este caso es diferente a la anterior, pero el concepto es el mismo. Podemos encontrar esta estructura en la Figura 37. Tendremos dos objetos principales: *numeroEquipos* y *equipos.*

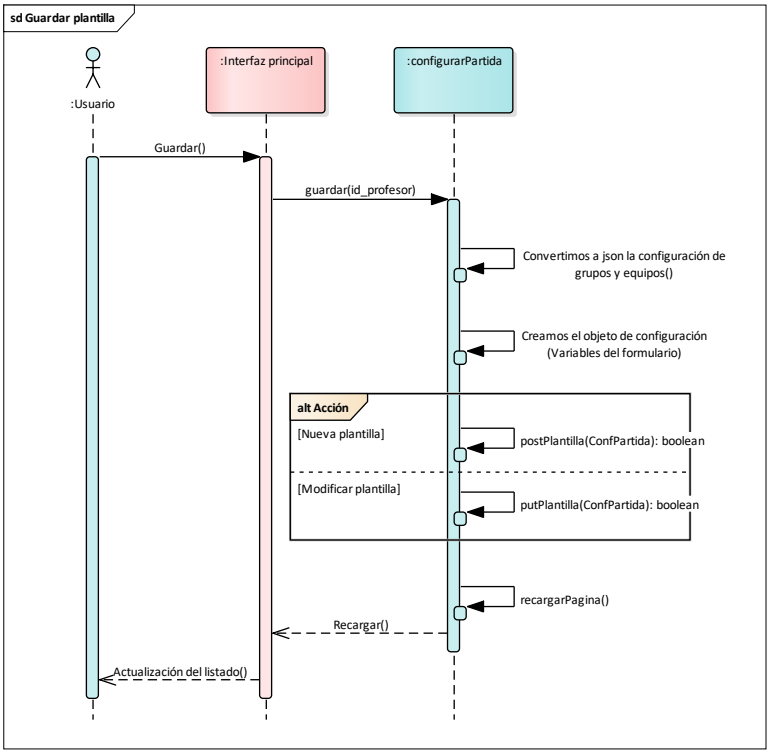
numeroEquipos es el objeto análogo a *numeroGrupos* del caso anterior. Se deben configurar los mismos campos y servirá para configurar el número de equipos que tendrá la partida.

La diferencia viene en el elemento *equipos*, este elemento es un array, que deberá tener tantos elementos como número máximo de equipos permita el juego. Cada uno de estos elementos, tendrá tres objetos: *nombreEquipo*, *numeroGrupos* y *grupos*. En nombreEquipos solo será necesario configurar el textoWeb para la visualización y nombre, el campo valor se rellenará al guardar. Por otro lado, numeroGrupos será el objeto análogo a numeroGrupos del caso anterior, pero en lugar de corresponderse con el global de grupos, será el número de grupos que ese equipo podrá tener. Y finalmente, el objeto *grupos* será un listado de variables de juego, donde habrá tanto elementos como número máximo de grupos permita ese equipo.

```
{ "equipos":{
      "numeroEquipos":{
          "textoWeb":"Nombre del equipo",
          "rango":{
              "valorMinimo":1,
              "valorMaximo":1,
              "valorDefecto":1
          },
          "variablesJuego":[{
            "nombre":"jug1",
            "valor":""
            }]
      },
      "equipos":[{
        "nombreEquipo":{
            "nombre":"",
            "valor":"",
            "textoWeb":""
          },
          "numeroGrupos":{
            "textoWeb":"",
            "rango":{
              "valorMinimo":1,
              "valorMaximo":1,
              "valorDefecto":1
              },
              "variablesJuego":[{
                "nombre":"jug1",
                "valor":""
                }]
              },
            "grupos":[{
                "nombre":"",
                "valor":"",
                "textoWeb":""
              }]
       }]
    }
}
```


Figura 37: Estructura jsonequipos - Configuración de grupos organizados en equipos

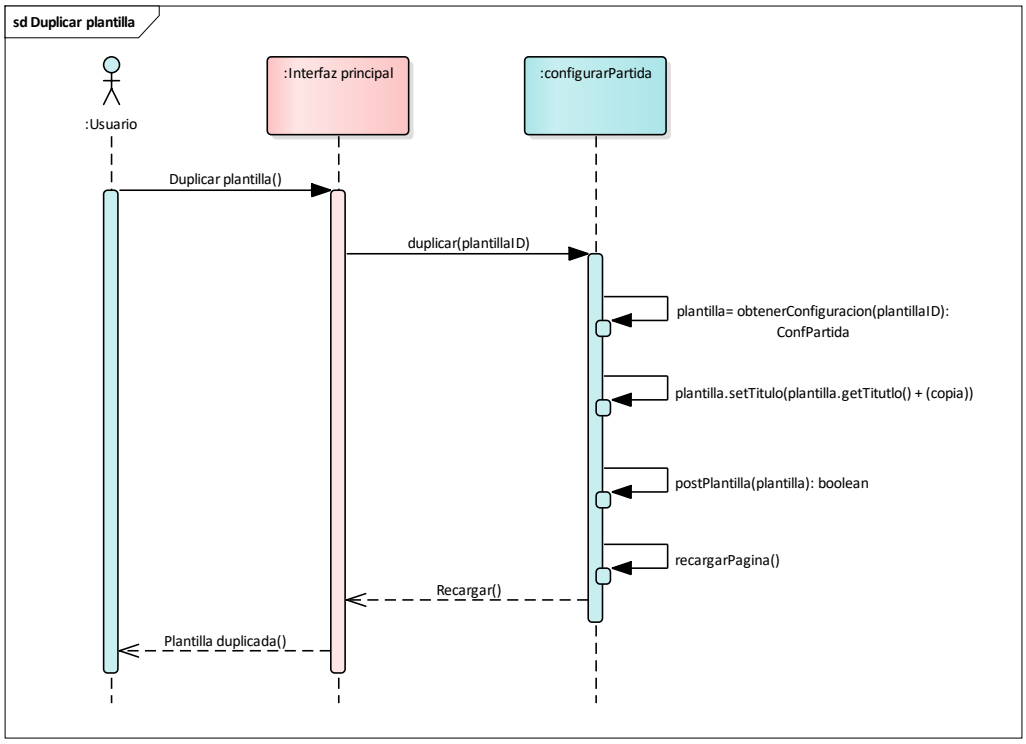
FASE 3: Guardar la plantilla. Una vez hemos llegado al final del formulario, el usuario podrá pulsar sobre el botón Guardar, que iniciará la secuencia que se detalla en el diagrama UML 8.



UML 8: SEQ-004 Nueva plantilla/Modificar plantilla

Cuando se inicia esta secuencia el bean *configurarPartida* toma el control y se ejecuta el método *guardar* al cual se le pasa el identificador del profesor. En primer lugar, se comprueba si la plantilla tiene asociada una configuración de grupos y equipos y en caso afirmativo se convierte el objeto en json, para tener todas las variables necesarias para inicializar el objeto plantilla. Una vez hemos creado el objeto de la plantilla, realizamos llamada rest correspondiente según estemos creando una nueva plantilla o modificándola, mediante los métodos *postPlantilla* o *putPlantilla* correspondientemente.

Duplicar plantilla



UML 9: SEQ-005 Duplicar plantilla

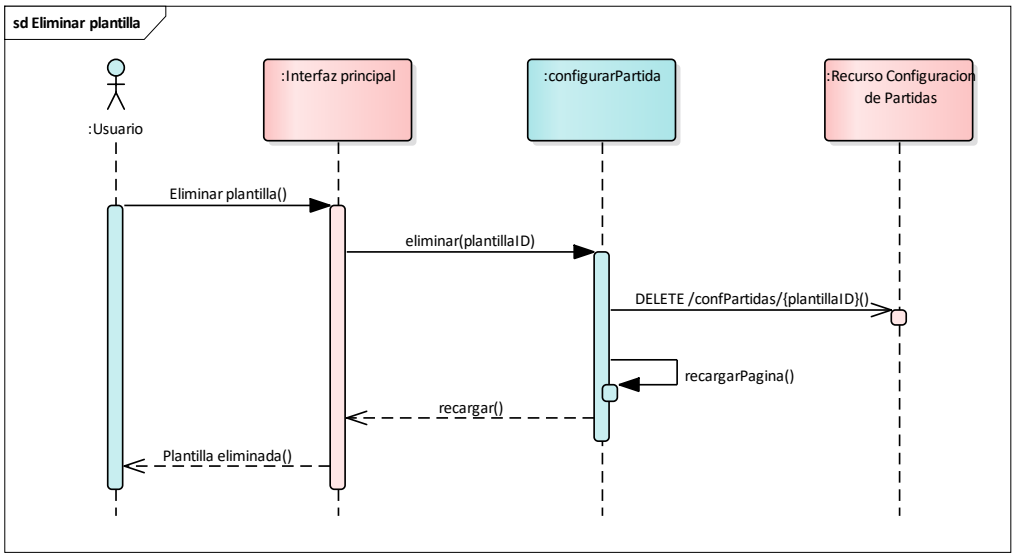
En el diagrama UML 9 se detalla el caso de uso *Duplicar plantilla*.

Este caso de uso se inicia cuando el usuario pulsa sobre el botón de *Duplicar plantilla*, el cual puede reconocerse por un icono con dos ventanas superpuestas (icono de copia).

Cuando se pulsa dicho botón, toma el control del método *duplicar* del bean *configurarPartida*. Desde la interfaz se manda como parámetro, el identificador de la plantilla seleccionada. El método *duplicar* lleva a cabo los siguientes pasos:

1. Se obtiene el objeto plantilla mediante el método *obtenerConfiguracion*. Este método realiza la petición GET al recurso *ConfPartida* de la API para el identificador de la plantilla seleccionada.
2. Se modifica el título de la plantilla, añadiéndole la coletilla *(copia),* para identificar que es la plantilla duplicada.
3. Se realiza la inserción de plantilla en la base de datos haciendo uso del método *postPlantilla*. Este método es el encargado de realizar la petición POST al recurso *ConfPartida* de la API REST para crear una nueva plantilla en la base de datos.
4. Se recarga la página para recargar el listado de plantillas y que se visualice la copia creada.

Eliminar plantilla



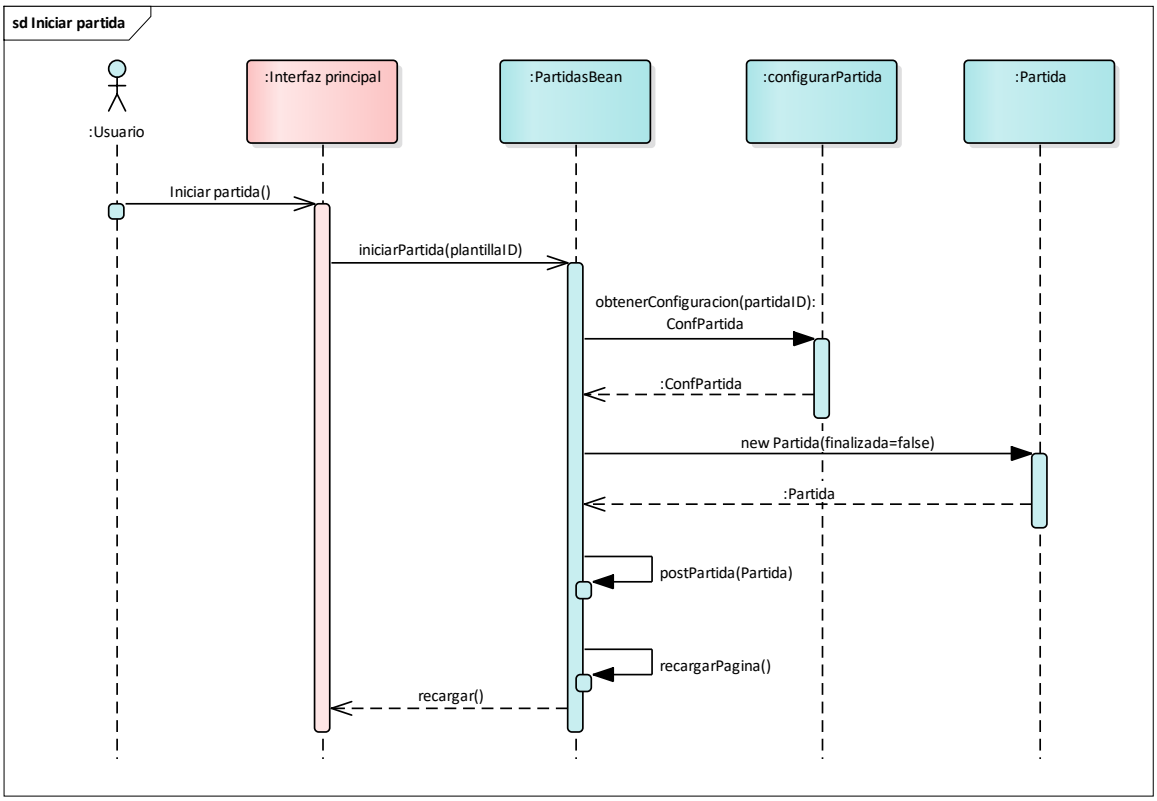
UML 10: SEQ-006 Eliminar plantilla

En el diagrama UML 10 se detalla el caso de uso *Eliminar plantilla*.

Esta secuencia se inicia cuando el usuario pulsa sobre el botón de *Eliminar plantilla*, el cual puede reconocerse por un icono de papelera.

Cuando se pulsa dicho botón, toma el control del método *eliminar* del bean *configurarPartida*. Desde la interfaz se manda como parámetro, el identificador de la plantilla seleccionada. Este método simplemente realiza la petición DELETE al recurso confPartidas de la API REST para eliminar la plantilla seleccionada. Una vez se ha realizado la petición se recarga la página para que se refleje el cambio en la aplicación.

Iniciar partida



UML 11: SEQ-007 Iniciar partida

En el diagrama UML 11 se detalla el caso de uso *Iniciar partida*.

La secuencia de *Iniciar partida* se da cuando el usuario pulsa sobre el botón con icono de play de una plantilla/configuración de partida, el cual se puede observar en la Figura 28.

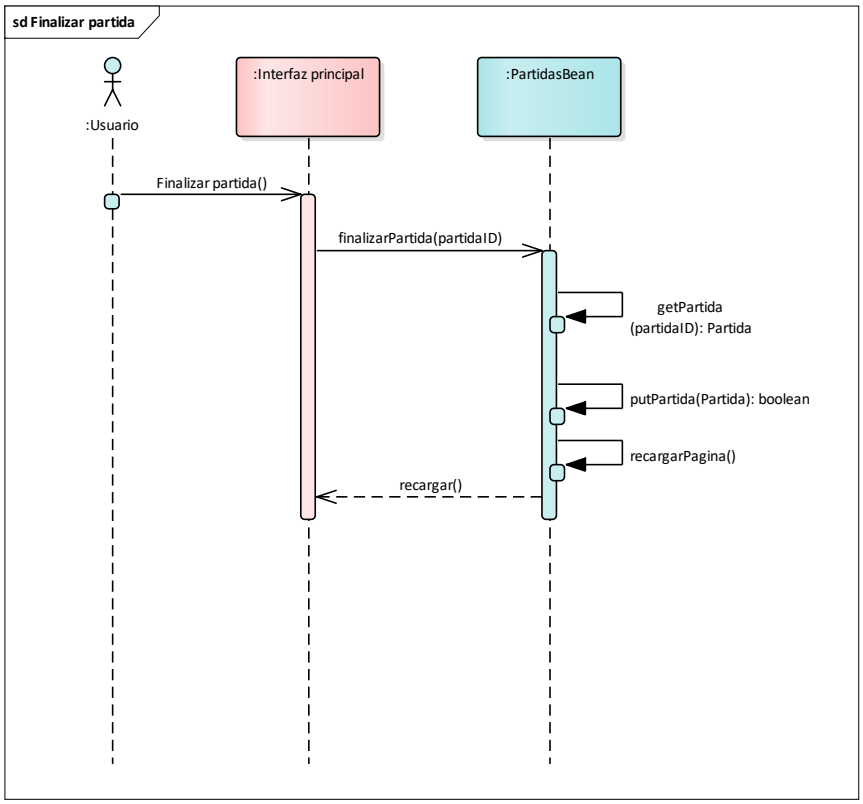
Cuando el usuario pulsa sobre dicho botón, *PartidasBean* toma el control mediante el método *iniciarPartida*. En este momento, obtiene el identificador de la plantilla/configuración de partida seleccionada y con dicho identificador hace uso del método *obtenerConfiguracion* del bean *configurarPartida*, el cual realiza la petición GET a la API REST para obtener el objeto con la configuración selecciona.

A partir de este objeto *ConfPartida*, se crea su correspondiente objeto *Partida*, copiando lo campos en los que estos dos objetos coinciden e inicializando a false el campo *finalizada*, para indicar que es una partida en curso.

Una vez tenemos, la Partida inicializada se procede a realizar la operación POST de la partida para guardarla, tal y como se puede ver en el diagrama UML 11 haciendo uso del método *postPartida.*

Si todos los pasos se realizan correctamente se recarga la página para actualizar el listado de partidas en curso.

Finalizar partida



UML 12: SEQ-008 Finalizar partida

En el diagrama UML 12 se detalla el diagrama de secuencia del caso de uso *Finalizar partida*.

Para iniciar esta secuencia, el usuario debe pulsar sobre el botón con icono de stop de alguna partida en curso del listado.

En este momento, se da paso al método *finalizarPartida* del bean *PartidasBean.* En este método, en primer lugar, se obtiene el identificador de partida que se ha seleccionado y con él, mediante el método *putPartida*, se realiza la petición GET al servicio REST para obtener el objeto Partida con toda la información.

Se actualiza el estado de la partida, poniendo el campo finalizada a true y se actualiza la partida en la base de datos realizando el PUT en la API REST, con el método *putPartida*.

Finalmente, si no se produce ningún error, se recarga la página para actualizar el listado de partidas en curso.

Interfaz de gestión de usuarios

La interfaz de gestión de usuarios, la cual se muestra en la Figura 38, solo está disponible para usuarios con rol de superusuario, para los cuales se habilitará el acceso en la barra de herramientas en todas las interfaces.

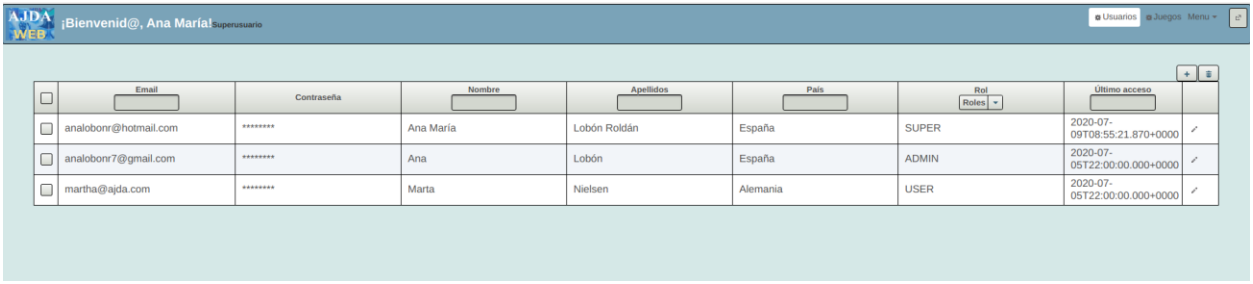
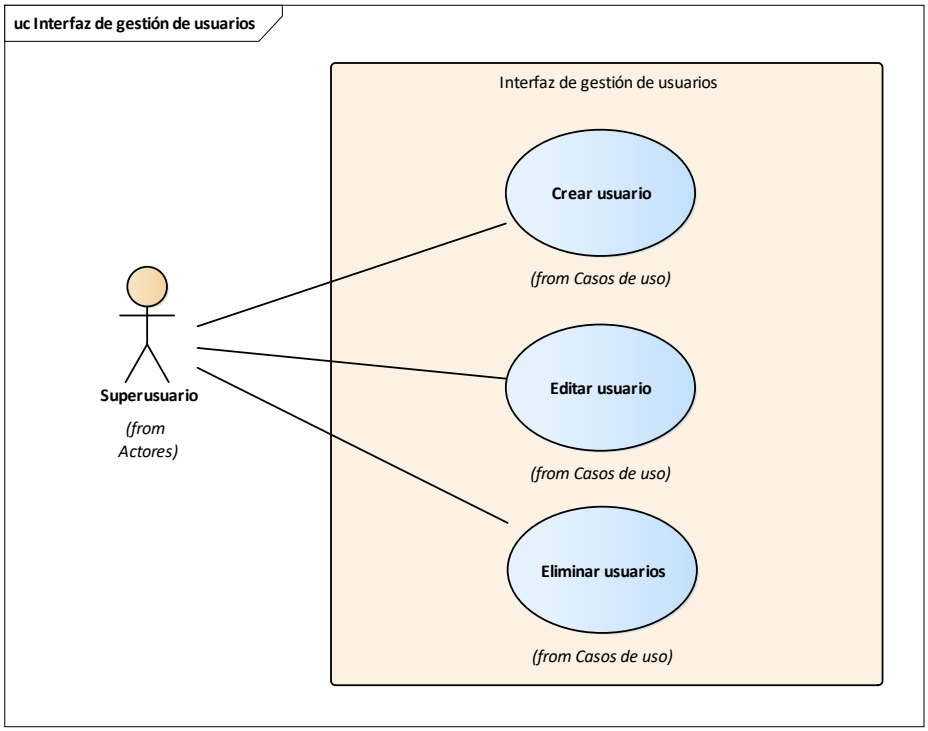


Figura 38: Interfaz de gestión de usuarios

En esta interfaz se mostrará el listado de todos los usuarios registrados en la aplicación, para los cuales se muestra el email con el que están registrados, la contraseña oculta, el nombre, apellidos, país de procedencia, el rol y la fecha en la que accedió a la aplicación por última vez.

A continuación, en el diagrama UML 13: CU-004 Interfaz de gestión de usuariosUML 13, podemos ver el diagrama de Caso de Uso para esta interfaz.



UML 13: CU-004 Interfaz de gestión de usuarios

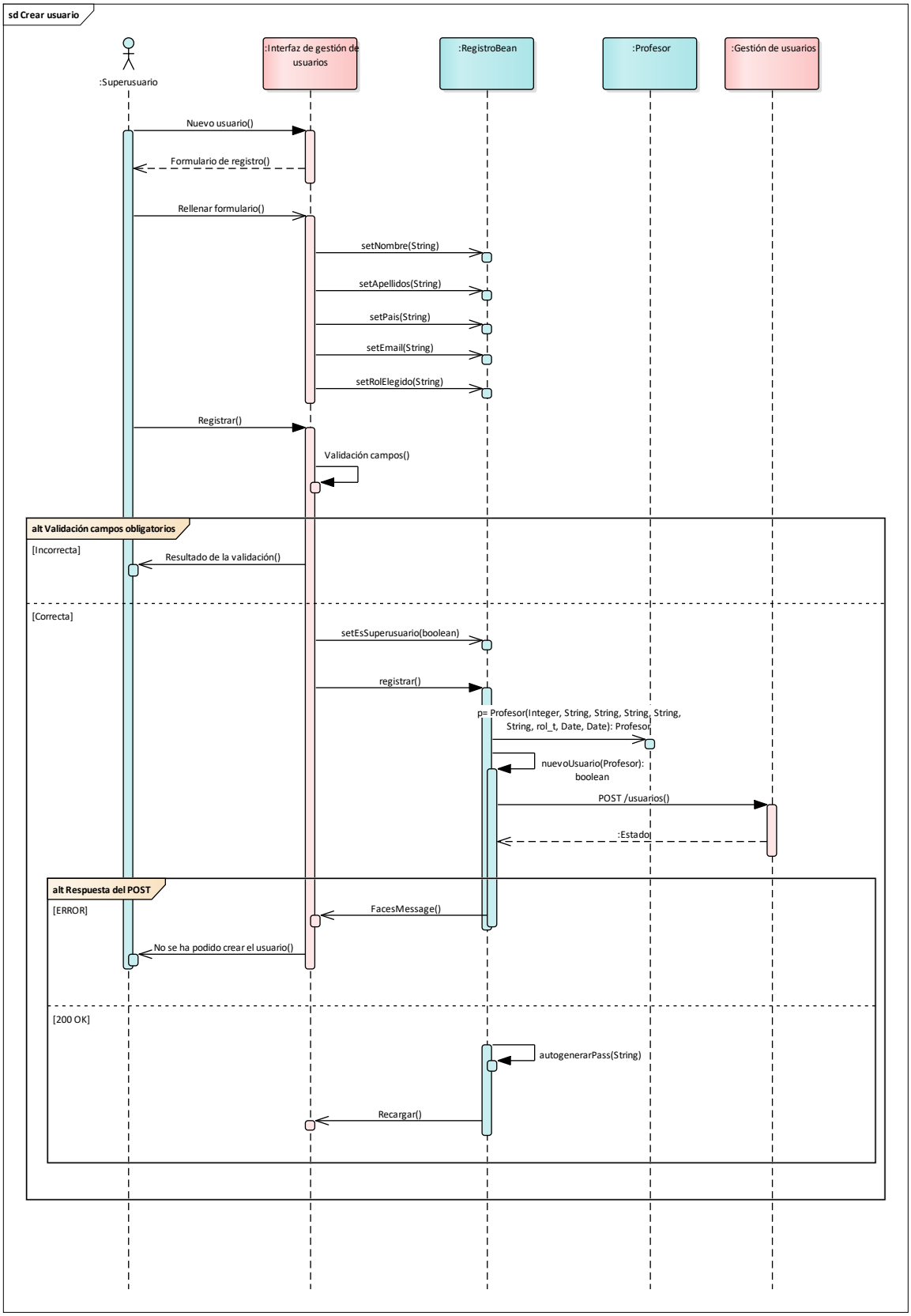
Tenemos tres casos de usos principales:

- **Crear usuario:** permite crear un nuevo usuario. Las diferencias entre este caso de uso y el de registro de la interfaz de inicio, es que en este caso se puede seleccionar el rol que el usuario tendrá y la contraseña será autogenerada, la cual se notificará al usuario por correo electrónico.
- **Editar usuario:** permite editar un usuario existente. Se podrán editar todos los campos que se muestran en la tabla de usuarios (Figura 38), excepto la contraseña de acceso y la fecha de último acceso.

- **Eliminar usuarios:** permite seleccionar uno o varios usuarios y eliminarlos del sistema.

A parte de estos casos, también se puede realizar un filtrado de usuarios mediante los inputs que encontramos en la cabecera de la tabla de usuarios.

Crear usuario



UML 14: SEQ-009 Crear usuario

En el diagrama UML 14 se detalla el caso de uso *Crear usuario.*

Esta secuencia se inicia cuando el superusuario pulsa sobre el botón Nuevo usuario, el cual tiene un icono de

`+' tal y como se puede ver en la Figura 38. Entonces al superusuario se le muestra un diálogo con el formulario para registrar un nuevo usuario, el cual podemos ver en la Figura 39.

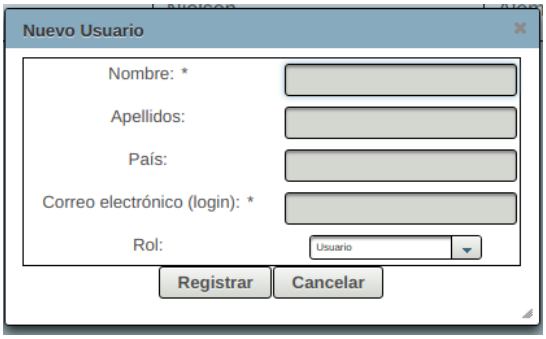


Figura 39: Formuario nuevo usuario

El superusuario deberá rellenar los campos y pulsar sobre el botón registrar, en un primer momento se realizará una validación de los campos requeridos y si éstos están rellenos comenzará el flujo de registro del usuario.

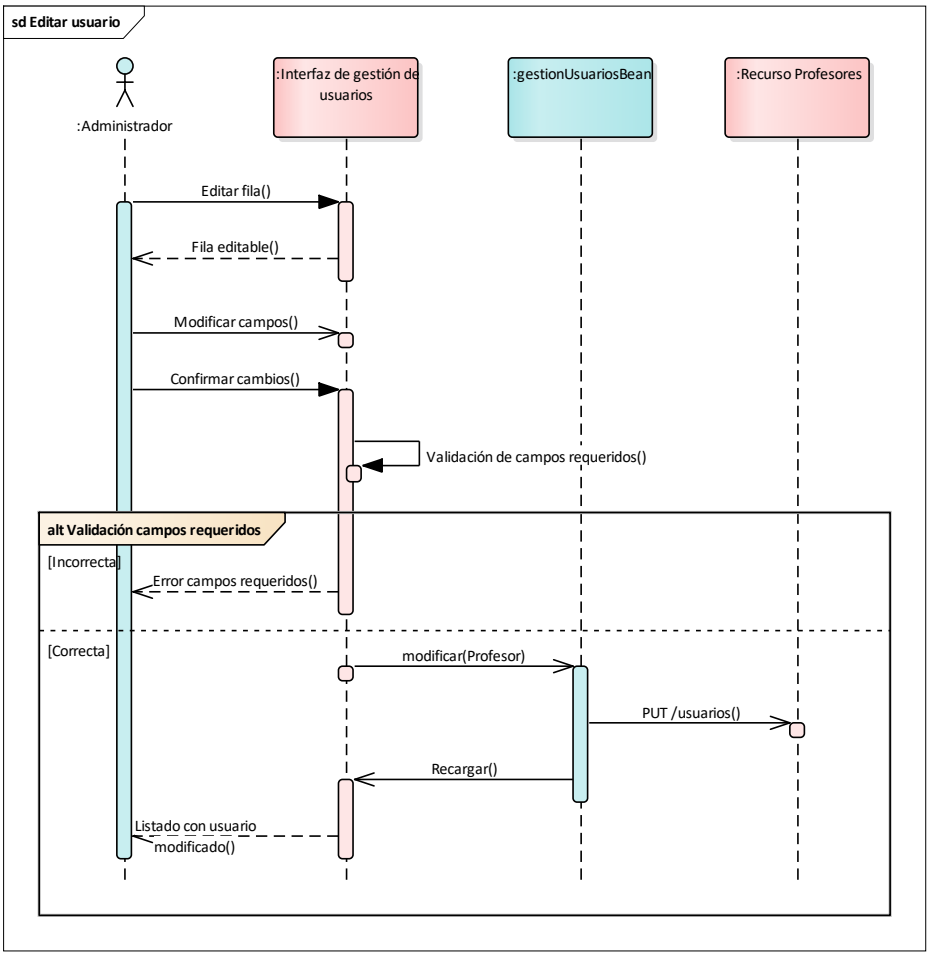
La creación del nuevo usuario está gestionada por el bean RegistroBean, mediante el mismo método que gestiona el registro de un usuario desde la interfaz de inicio (UML 5). Sin embargo, la diferencia es que en este caso se inicializa previamente el atributo esSuperusuario, el cual indica que el rol del usuario que está gestionando el registro es un superusuario y, por tanto, el flujo diferirá.

Tal y como podemos ver en el diagrama de secuencia la primera parte del método *registrar*, es exactamente igual a la que vemos en el diagrama UML 5. Creamos el objeto Profesor con los datos del formulario y realizamos la operación POST */usuarios* hacia la API REST. Una vez el registro se realiza correctamente, en este caso hacemos uso del método autogenerarPass, el cual realiza la llamada GET /usuarios/{email} a la API la cual nos devuelve el usuario con una nueva contraseña autogenerada y generamos un correo electrónico informando del registro al usuario. Este detalle, podemos verlo en el diagrama UML 14.

Finalmente, se recarga la interfaz de gestión de usuarios para que se actualice el listado de usuarios, mostrando el nuevo usuario.

En el diagrama UML 14 podemos ver que para cualquier error que se produzca se finaliza la secuencia y se notifica al usuario mediante un mensaje que aparecerá en la interfaz.

Editar usuario



UML 15: SEQ-010 Editar usuario

En el diagrama UML 15 se detalla el caso de uso *Editar usuario.*

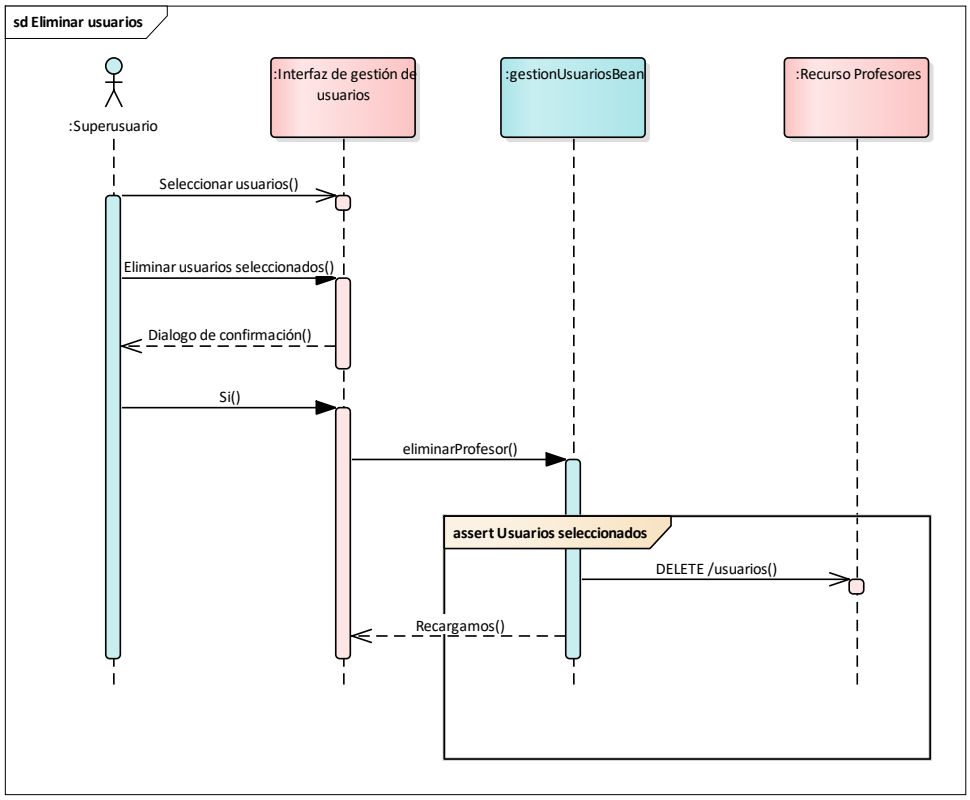
Como podemos ver en la Figura 38, cada fila de la tabla tiene un icono de un lápiz, cuando clicamos sobre uno de ellos, los campos modificables del usuario seleccionado se hacen editables. De forma, que la secuencia de edición del usuario comenzaría al confirmar los cambios con el tick.

De igual manera que en los casos de usos anteriores, primero se realiza una validación de campos requeridos. Una vez pasada, vemos que toma el control del proceso el bean *gestionUsuariosBean* a través del método *modificar*.

El método modificar, recibe directamente el objeto Profesor con los nuevos datos y realiza la petición PUT al recurso /usuarios de la API REST y recarga la interfaz, para que se vuelvan a cargar los usuarios existentes mostrando los cambios realizados.

En el diagrama UML 15 podemos ver que para cualquier error que se produzca se finaliza la secuencia y se notifica al usuario mediante un mensaje que aparecerá en la interfaz.

Eliminar usuarios



UML 16: SEQ-011 Eliminar usuarios

En el diagrama UML 16 se detalla el caso de uso *Eliminar usuarios.*

Como podemos ver en la Figura 38, cada fila de la tabla tiene en su primera columna un checkbox, lo cual nos permiten seleccionar múltiples usuarios.

Para iniciar la secuencia de eliminar usuarios, primero debemos seleccionar los usuarios que queremos eliminar y pulsar sobre el icono de la papelera (Eliminar usuarios). Esto nos mostrará un diálogo de confirmación, para evitar eliminar usuarios por error, y una vez realizamos la confirmación comenzaría el proceso de eliminación.

Este proceso, se realiza con el método *eliminarProfesor* del bean *gestionUsuariosBean*. Este método simplemente obtiene el listado de usuarios seleccionados y realiza la petición DELETE /usuarios enviando dicho listado como body. Una vez realizada la petición recarga la interfaz, actualizando el listado de usuarios.

Ante cualquier error que se produzca se finaliza la secuencia y se notifica al usuario mediante un mensaje que aparecerá en la interfaz, tal y como se hace en el resto de los casos de uso.

Interfaz de gestión de juegos

La interfaz de gestión de juegos, la cual se muestra en la Figura 40, solo está disponible para usuarios con rol de administrador o superusuario, para los cuales se habilitará el acceso en la barra de herramientas en todas las interfaces.

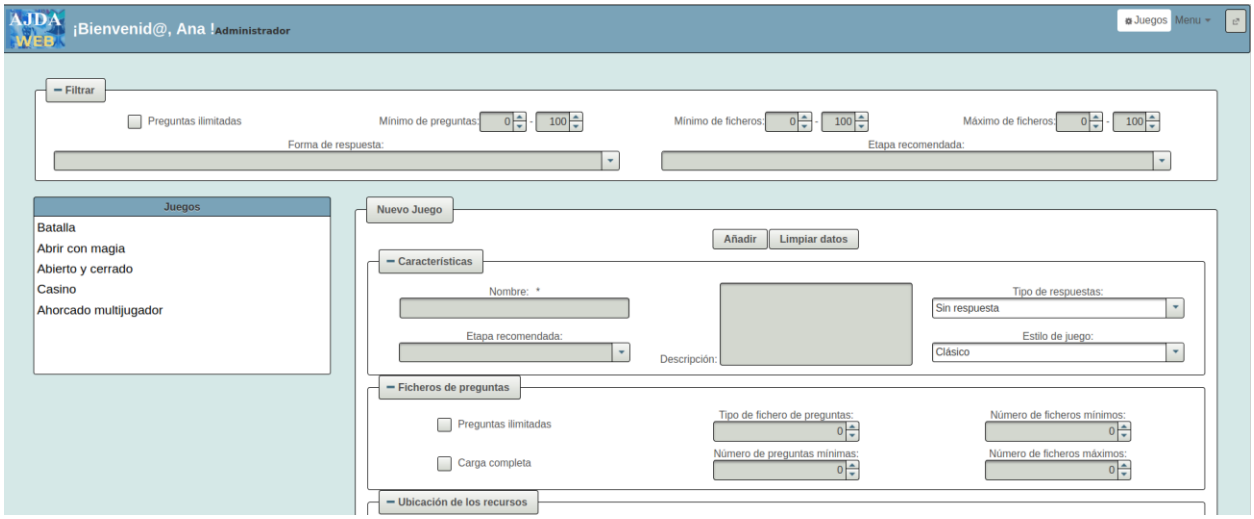
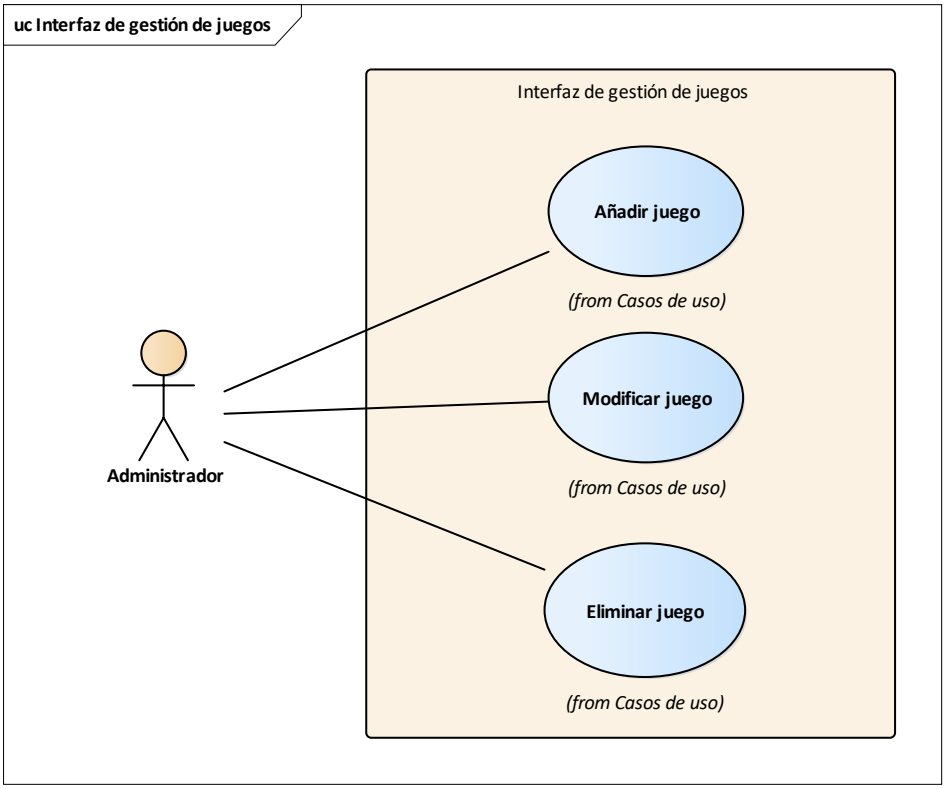


Figura 40: Interfaz de gestión de juegos

En esta interfaz podemos distinguir tres zonas:

- Zona superior: panel con los posibles filtros para encontrar juegos.
- Zona inferior izquierda: listado de juegos, inicialmente aparecerá el listado completo de juegos existentes, los cuales podrán filtrarse con el conjunto de filtros.
- Zona inferior derecha: formulario para el registro y modificaciones de juegos.

A continuación, en el diagrama UML 17, podemos ver el diagrama de Caso de Uso para esta interfaz.

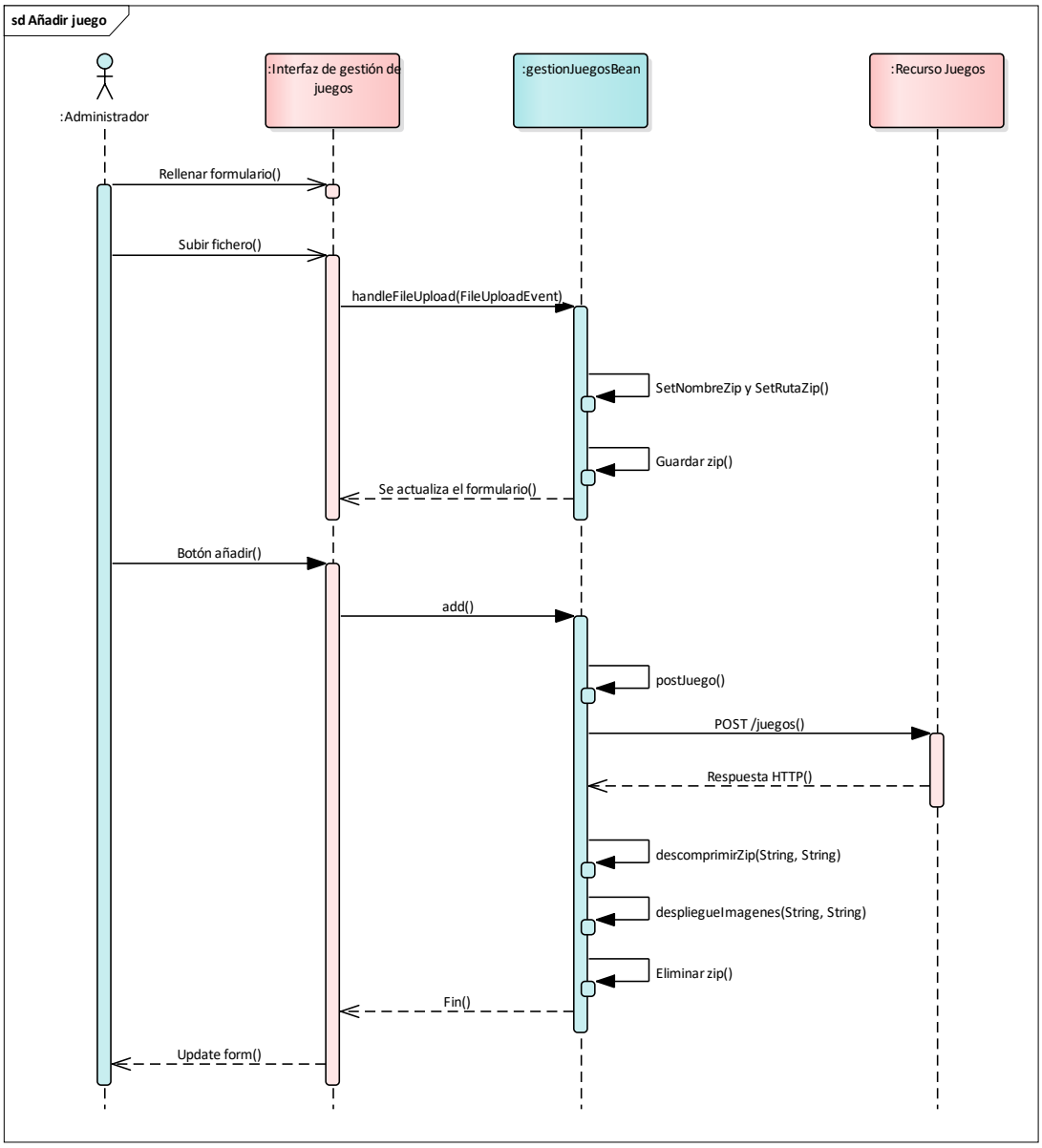


UML 17: CU-005 Interfaz de gestión de juegos

Los principales casos de usos son:

- **Añadir juego:** permite crear un nuevo juego, a partir de los datos rellenos en el formulario.
- **Modificar juego:** permite modificar los datos de un juego. Al seleccionar un juego del listado, su información se rellenará en el formulario, permitiendo al usuario modificar dicha información.
- **Eliminar juego:** permite eliminar un juego seleccionado.

Añadir juego



UML 18: SEQ-012 Añadir juego

En el diagrama UML 18 se detalla el caso de uso *Añadir juego.*

Para iniciar esta secuencia el usuario debe rellenar el formulario que se puede observar en la Figura 41. Los campos del formulario se corresponden con las columnas de las tablas del Modelo E/R de la Base de Datos (Tabla 1).

Uno de los campos obligatorios es el fichero .zip que contiene el juego, por tanto, antes de añadir el juego, debemos subirlo al servidor. En la sección 'Ubicación de los recursos' bien podemos pulsar en *Examinar* o arrastrar directamente el archivo. Una vez seleccionado pulsamos en Subir y se iniciará la secuencia que vemos en el diagrama, ejecutándose el método handleFileUpload de gestionJuegosBean, el cual subirá el fichero al servidor y asignará valor a los atributos del juego nombre y ruta del zip.

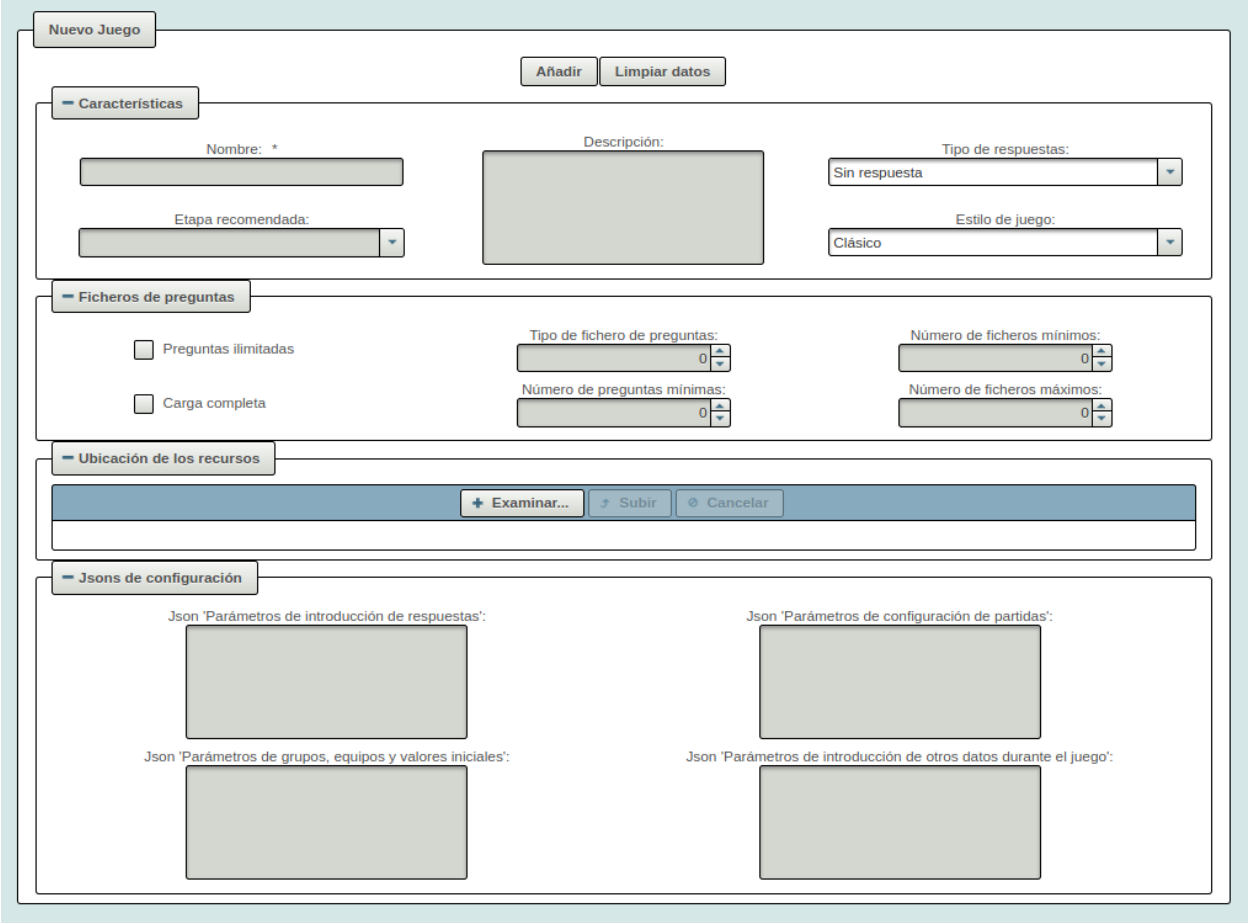
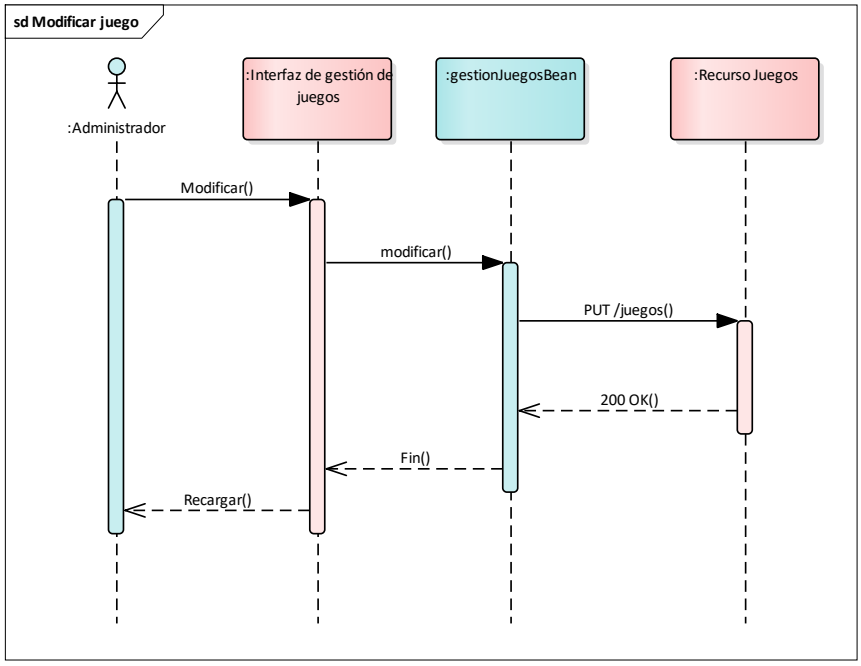


Figura 41: Formulario de datos del juego

Una vez el formulario esté completo con los datos del juego, el usuario deberá pulsar sobre el botón *Añadir*. En este momento vuelve a tomar control el bean *gestionJuegosBean*, ejecutándose el método *add*. Este método realiza la inserción del juego en la base de datos haciendo uso del recurso POST /juegos, si la inserción se realiza sin errores se descomprime el zip en el servidor, se despliegan las imágenes en el directorio del Tomcat correspondiente para que se puedan visualizar junto a la información del juego y se elimina el zip.

Modificar juego



UML 19: SEQ-013 Modificar juego

En el diagrama UML 19 se detalla el caso de uso *Modificar juego.*

Para iniciar esta secuencia el usuario debe seleccionar un juego del listado de la sección inferior derecha (Figura 40). Entonces se rellenará el formulario ya mencionado en el caso de uso anterior, además se cambiará el botón de Añadir por el de Guardar Cambios y se incluirá también un botón de eliminar, tal y como se muestra en la Figura 42.

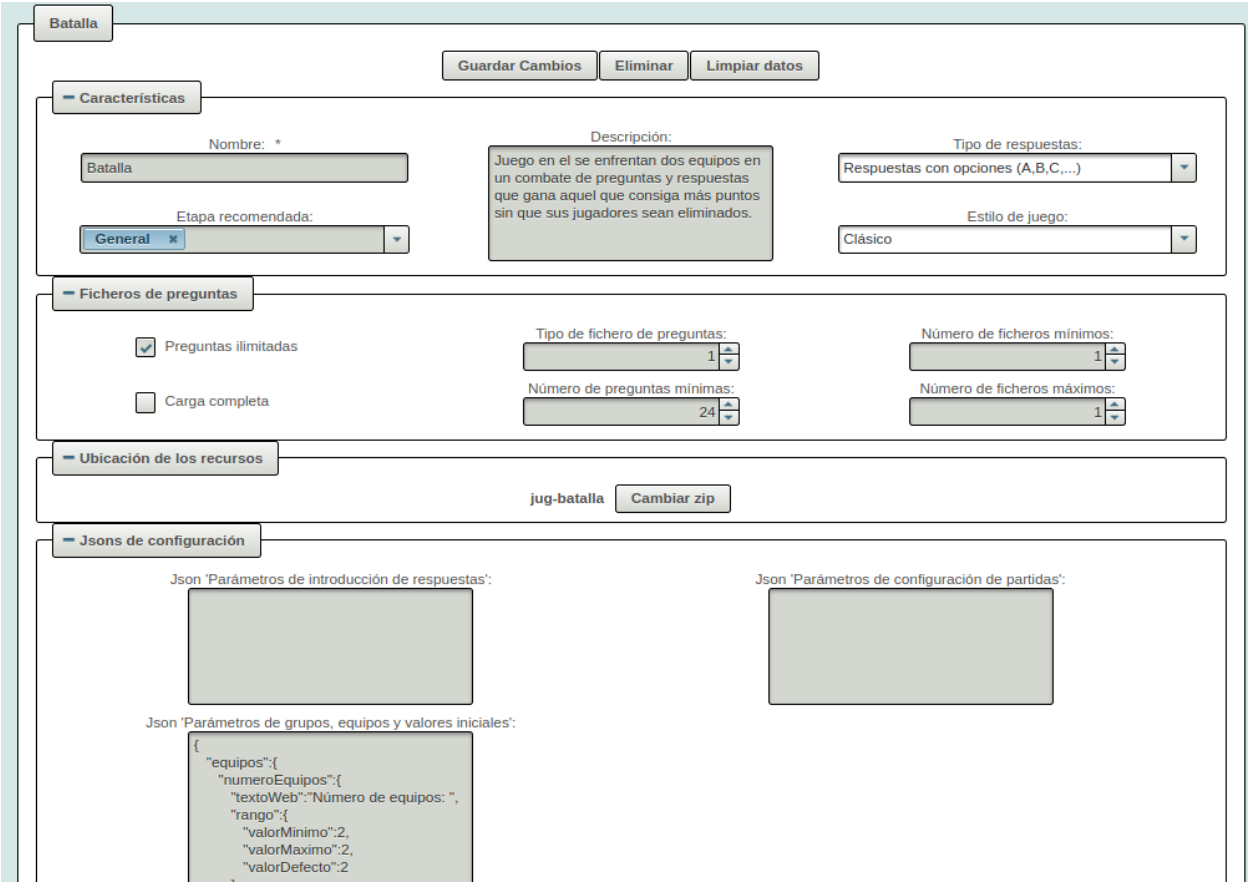
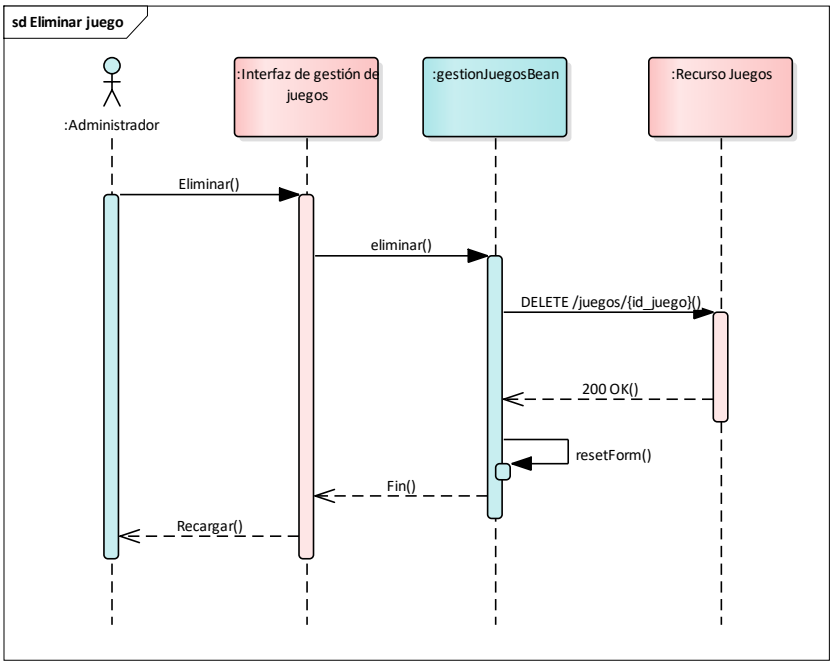


Figura 42: Formulario juego seleccionado

El usuario podrá realizar las modificaciones necesarias sobre cualquier campo del formulario y una vez quiera hacer efectivo los cambios deberá pulsar sobre el botón *Guardar cambios*. Entonces, se iniciará el método modificar del bean gestionJuegosBean, el cual simplemente realizará la petición PUT al recurso /juegos de la API REST para actualizar el registro del juego con los nuevos datos.

Eliminar juego



UML 20: SEQ-014 Eliminar juego

En el diagrama UML 20 se detalla el caso de uso *Eliminar juego.*

Al igual que en el caso de uso anterior, *Modificar juego*, para iniciar esta secuencia el usuario debe seleccionar un juego del listado de la sección inferior derecha y se rellenará el formulario (Figura 42). Como ya se mencionó, al seleccionar el juego aparece el botón 'Eliminar' el cual debe pulsar el usuario para eliminar el juego. Cuando este botón se pulsa, el bean *gestionJuegosBean* inica el método *eliminar*, el cual realiza la petición DELETE al recurso /juego/{id_juego} de la API REST para eliminar el juego seleccionado. Si todo va bien, se reiniciará el formulario y se recargará la interfaz para mostrar el nuevo listado de juegos.

IMPLEMENTACIÓN

En este apartado se detallarán los datos más relevantes relacionados con la implementación de la aplicación.

1. API REST

Estructura del Proyecto

La API REST de nuestro proyecto es una aplicación Spring Boot, tal y como se explica en el apartado 2, en el que se detalla las Tecnologías y Herramientas utilizadas para el desarrollo de la aplicación. Por tanto, la estructura y elementos del proyecto se corresponde con una API REST de Spring Boot. Podemos ver esta organización en la Figura 43.

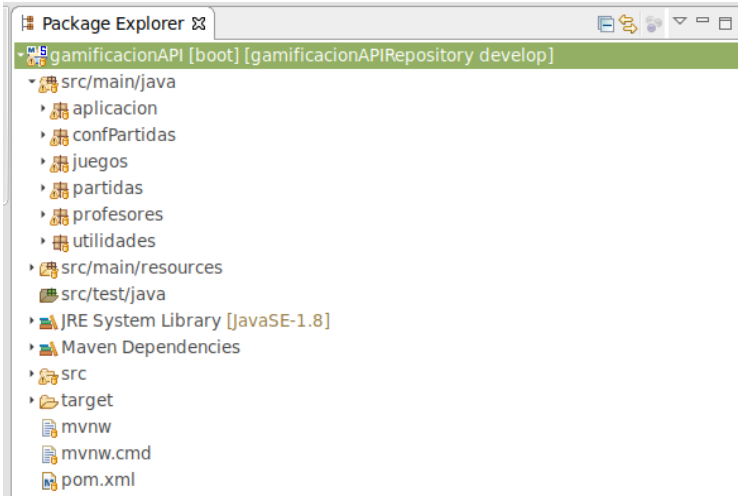


Figura 43: Estructura del proyecto API REST

En la raíz del proyecto podemos ver el fichero pom.xml. En este fichero, se definen las dependencias que el proyecto necesita. En nuestro caso, necesitaremos tres dependencias:

- **Spring-boot-starter:** es uno de los paquetes starter que ofrece Spring Boot a raíz del cual se descargarán todas las dependencias que necesita el proyecto para ser un proyecto API REST.
- **Postgresql:** esta dependencia es necesaria para poder conectar nuestra API con la base de datos.
- **Javax persistence:** es el motor que se ha elegido para desarrollar nuestra Java Persistence API (JPA).

A continuación, pasaremos a explicar los diferentes paquetes en los que están organizadas las clases java (src/main/java). Podemos ver el contenido de estos paquetes en la Figura 44.

- **Aplicación:** en este paquete tendremos las clases principales de la aplicación. En primer lugar, tenemos la clase GamificaciónAPIApplication.java, esta clase extiende de Spring Boot y se encarga de configurar y lanzar nuestra aplicación. Por otro lado, tenemos los diferentes Controllers, uno por cada recurso de acceso, o lo que es lo mismo, por cada una de las tablas de la base de datos. En estos Controllers se determinan cada uno de los recursos que ofrece la API REST, definiendo las operaciones, los paths y los métodos que recogerán las llamadas.
- **Recursos:** podemos ver que tenemos un paquete de clases por cada recurso: **confPartidas, juegos, partidas y profesores**. En cada paquete, encontramos los tres mismos tipos de clases:
    - **<recurso>.java:** esta clase es el POJO (Plain Old Java Object) o clase simple en la que se mapeará la información de la base de datos, pues mediante diferentes anotaciones de la

librería Javax persistence, esta clase se define como Entidad. De la misma forma, mediante estas anotaciones también se define el nombre de la tabla SQL con la que se corresponde, al igual que los campos de esta u otros rasgos específicos que se deban establecer.

- **<recurso>DAO.java:** en esta clase se define la interfaz con los métodos disponibles para gestionar la base de datos.
- **JPA<recurso>DAO.**java: esta clase es la implementación de la interfaz de acceso a la base de datos. Como indica su nombre la implementación está realizada con JPA.

- Utilidades: en este paquete se definen diferentes clases auxiliares que se necesitan para la implementación.

Figura 44: Estructura paquetes java - API REST

Por último, en el directorio de resources se definirán diferentes recursos que necesite la aplicación. En este caso tendremos dos clases:

- **Application.properties:** es el fichero de configuración de la aplicación Spring Boot. En nuestro caso, está vacío pues con la configuración predeterminada es suficiente. Pero, por ejemplo, en este fichero se podría cambiar el puerto en el que se lanza el Tomcat interno de la aplicación.
- **META-INF/persistence.xml:** este fichero es muy importante para poder conectar correctamente con la base de datos. En este fichero, se configura la unidad de persistencia, necesaria para poder crear la transacción sobre la que trabajará JPA. En esta unidad de persistencia, es necesario configurar el proveedor JPA, las clases que se utilizarán y lo más importante, la configuración de la conexión con la base de datos.

Gestión de errores

Como se ha comentado en capítulos anteriores, en caso de que se produzca algún error interno en las diferentes operaciones se devuelve un error 500 Internal Server Error.

Todos los errores están controlados en los diferentes Controllers, de forma que si se detecta un error interno se debe lanzar una excepción para que en lugar de devolver un 200 OK se devuelva el error 500 Internal Server Error junto con una descripción del error. Para ello, ha sido necesario crear una clase con una excepción customizada *CustomInternalServerErrorException.*

Documentación

En el apartado de Herramientas utilizadas para el desarrollo del proyecto, se explica que la herramienta para documentar la API REST es Swagger UI. Para ello, se ha utilizado la librería springfox-swagger-ui.

Esta librería genera de forma automática la página de Swagger UI a raíz del fichero de configuración SwaggerConfig.java y las diferentes anotaciones que se incluyen en los Controllers. Con estas anotaciones se definen los nombres y las descripciones de los diferentes recursos, así como las descripciones de los diversos parámetros.

2. Aplicación Web

Estructura del proyecto

La Aplicación Web de nuestro proyecto es un proyecto web dinámico. Podemos dividir su estructura en tres partes:

- **Src:** en este directorio encontraremos las múltiples clases java que componen nuestro proyecto. Estas clases se han dividido en diferentes paquetes agrupados según su funcionalidad, tal y como puede verse en la Figura 45. A continuación, detallaremos cada uno de estos paquetes:
    - **Beans:** en este paquete encontramos todos los "java beans" de nuestro proyecto. Estas clases son el motor controlador de nuestra aplicación, pues a través de ellas podemos conectar la capa visual con el modelo de datos para mostrarlos e interactuar con ellos.
    - **Constantes:** este paquete contiene diversas clases java que contienen valores constantes en nuestra aplicación. Estas clases están pensadas de forma que, si se quiere modificar alguno de estos valores, se pueda hacer desde un sitio centralizado sin necesidad de realizar cambios a lo largo de todo el código. Encontramos las siguientes clases:
        - **Configuracion.java:** este fichero está pensado como un fichero general de configuración. Actualmente, en él únicamente se configura los datos del correo electrónico remitente para la información a los usuarios.
        - **Mensajes.java:** en este fichero se configura todos aquellos mensajes de información o error dirigidos al usuario.
        - **Rutas.java:** en este fichero se configurarán todas las rutas de directorios necesarias en la aplicación. En ella, podemos encontrar todas las rutas relacionadas con el zip y las imágenes de los juegos.
        - **URLs.java:** como su nombre indica en este fichero se configurarán todas las URLs necesarias en la aplicación. Podemos encontrar las urls de redirección de los diferentes ficheros xhtml de nuestra aplicación y los diferentes endpoints con destino a nuestra API REST.
        - **valoresDesplegables.java:** en este fichero se definen diferentes tipos de datos enumerados y arrays con la información que se mostrará en los diferentes desplegables de la aplicación.
    - **Converters:** aquí encontramos clases personalizadas para poder mapear nuestras propias clases desde la capa visual al controlador.
    - **Pojos:** los pojos son clases básicas que simplemente contienen una serie de atributos y sus correspondientes getters y setters. Principalmente, aquí encontramos las clases de nuestro modelo de datos que usaremos para interactuar con la API REST. Dentro de este paquete encontramos un paquete interno *configuracionEquipos,* en el que se definen todos los pojos para poder mapear el json de grupos y equipos que se configura en los juegos.

Figura 45: Src - Estructura del proyecto - Aplicación web

- **WebContent:** nuestra aplicación web se despliega en un servidor Tomcat, por tanto, este directorio contendrá todo el contenido web que necesita nuestra aplicación. Podemos ver la estructura de este directorio en la Figura 46. A continuación, se detallarán los aspectos más relevantes:
    - **Ficheros xhtml:** como se puede apreciar tenemos un xhtml por cada interfaz de nuestra aplicación. Estos ficheros deben estar en la raíz del directorio WebContent.
    - **Resources:** aquí encontraremos todos los recursos que necesita nuestra aplicación, como pueden ser los ficheros css y las imágenes de la web.
    - **WEB-INF:** en este directorio encontraremos archivos de configuración como, por ejemplo, el web.xml del servidor o faces-config.xml en el cual se configuran aspectos relativos a JSF como las beans.

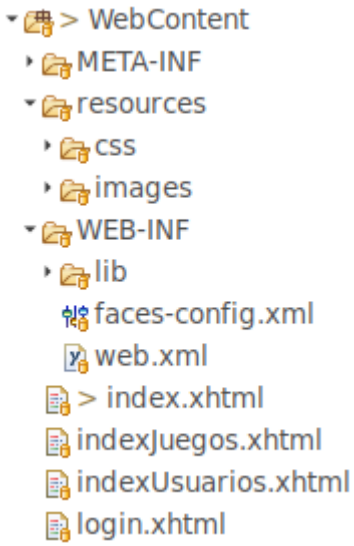


Figura 46: WebContent - Estructura del proyecto - Aplicación web

- **Librerías:** finalmente tenemos las librerías necesarias en nuestro proyecto, las cuales podemos observar en la Figura 47:
    - **JRE System Library:** librería de java. Es necesario el uso de Java 8, pues java 10 da múltiples problemas de compatibilidad con JSF.
    - **Apache Tomcat v8.5:** referencia al Tomcat que arrancará nuestra aplicación.

- **JSF:** librería básica necesaria para la aplicación. Se hace uso de la versión 2.2.0
- **Primefaces-6.1:** librería de primefaces que es base de nuestro proyecto.
- **Spring:** librería auxiliar, la cual se utiliza para diferentes necesidades. Por ejemplo, se hace uso de ella para realizar las peticiones HTTP a la API REST.
- **Jackson y Gson:** librerías auxiliares para el manejo de JSON como objetos en nuestra aplicación.
- **Mail:** librería auxiliar para el envío de correos electrónicos.

Figura 47: Librerías - Estructura del proyecto - Aplicación web

Gestión de errores

A nivel general en toda la aplicación web los errores internos que se puedan producir están gestionados de la misma manera.

En los beans se intentan controlar todos ellos mediante diferentes comprobaciones y excepciones, de forma que cuando se detecta un error que no permite seguir con el flujo, éste se para y se genera un mensaje de contexto FacesMessage.

Estos mensajes tendrán una cabecera y una descripción concordante al error producido, los cuales se definen en la clase Mensajes.java la cual se menciona en el subapartado anterior, y se mostrará al usuario mediante diversos componentes de tipo growl o message que ofrece Primefaces.

Documentación

La documentación de las clases java de este proyecto se ha realizado con Javadoc. El propio IDE utilizado para el desarrollo del proyecto, ofrece un plugin que permite autogenerar esta documentación a partir de las anotaciones que se añaden en las diversas clases.

La documentación se autogenera como una página web estática, a la cual podemos acceder mediante el index.html generado. Desde este index, podemos navegar a los diferentes paquetes y clases que conforman nuestra aplicación. En ella tendremos la descripción de cada una de las clases y los diferentes parámetros de entrada y salida que necesitan.

CONCLUSIONES

En este último capítulo se realizará una reflexión general del trabajo realizado y se detallarán las líneas de mejora que quedan abiertas para la continuación de este proyecto.

1. Conclusiones

Tras todo el trabajo realizado, es el momento de valorar en retrospectiva las tecnologías utilizadas en el desarrollo de este proyecto.

Por un lado, me gustaría destacar la flexibilidad que ofrece la API REST como interfaz de acceso a los datos. Es un punto importante para el futuro de la aplicación, pues va a facilitar la integración con el resto de los componentes del proyecto final. Esto permite una gran independencia entre estos componentes, pues la información se encontrará centralizada y accesible a cada uno de ellos sin necesidad de interactuar entre ellos.

El otro punto por valorar es el uso de Primefaces en nuestro proyecto. La mayor ventaja que nos ofrece esta librería son los potentes componentes que nos facilita. Podemos encontrar componentes para cualquier cosa que queramos incluir y además su fácil uso. Si bien, la curva de aprendizaje es lenta y compleja, finalmente cuando se consigue dominar es una librería que aporta muchísimo valor.

Finalmente, a nivel personal este proyecto me ha aportado mucho y, además, me ha ayudado a encontrarle sentido a todo el camino recorrido durante el grado. A nivel técnico me ha encantado ya que eran tecnologías nuevas para mí y he podido consolidar los conocimientos adquiridos durante mi formación en el grado, como la gestión de bases de datos con PostgeSQL, en aplicaciones web e incluso aplicar los conocimientos adquiridos en la asignatura Ingeniería del Software, que en el momento de cursarla me parecía algo abstracto.

2. Líneas de mejora

Para concluir, este proyecto deja un amplio camino para avanzar en su desarrollo y mejora continua. Por tanto, se dejarán aquí recogidas las principales líneas de mejora en las que se debe continuar:

- **Aplicación web:** la aplicación web para la gestión de usuarios, juegos y partidas deja por sí misma un amplio escenario de avance. Todos esos puntos de avance se han dejado indicados en la propia web, poniendo los elementos deshabilitados, los cuales podemos ver en la Figura 48.
    - **Buscador de partidas:** sección en la que el usuario pueda visualizar el histórico de partidas realizadas, tanto las que están en curso como las finales.
    - **Buscador de plantillas:** actualmente, cada usuario puede crear sus propias configuraciones de partidas. La idea es que esto evolucione de manera que las configuraciones puedan ser públicas o privadas y, por tanto, los usuarios puedan acceder a las configuraciones o plantillas de otros usuarios. Por tanto, este buscador será el motor de búsqueda de todas las plantillas existentes, tanto propias como ajenas.
    - **Estadísticas:** la idea es que en esta sección el usuario pueda visualizar las estadísticas finales de cada partida finalizada. Así como realizar comparaciones entre diferentes partidas que sirva de forma de evaluación.
    - **Datos personales:** pequeña sección en la que el usuario pueda modificar los datos personales dados durante el registro.

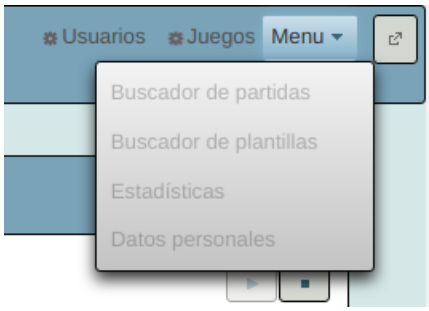


Figura 48: Aplicación web - Lineas de mejora

- **Seguridad:** durante el desarrollo de este proyecto se planteó incluir la seguridad de acceso a la aplicación mediante el protocolo OAuth2. Finalmente, en este proyecto nos centramos en la aplicación web, sin profundizar en este aspecto. Por tanto, este es un punto importante que queda por trabajar.
- **Integración:** por otro lado, se deben integrar las tres aplicaciones que conforman el gran proyecto final y que dará sentido a esta aplicación. En la aplicación web, se han quedado preparados los lugares dónde se deben integrar las otras dos aplicaciones:
    - **Aplicación web para la creación de ficheros de preguntas:** esta aplicación debe integrarse en la configuración de partida/plantilla, pues se debe seleccionar un fichero de preguntas (Figura 49). Por tanto, aquí debería incluirse un enlace al buscador de ficheros que incluye esta aplicación para seleccionar el fichero. Por otro lado, también debería incluirse en el menú de la aplicación de gestión un acceso a esta aplicación de ficheros.

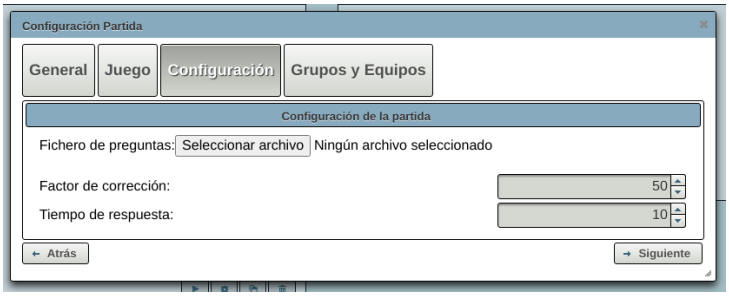


Figura 49: Línea de mejora - Aplicación web para la creación de ficheros de preguntas

    - Aplicación web multiusuario: esta aplicación debe integrarse en dos puntos, los cuales están marcados en la Figura 50.
        - **Botón Iniciar partida:** este botón deberá crear una partida en curso tal y como hace actualmente y a su vez redirigir a la aplicación web multiusuario para comenzar a jugar la partida.
        - **Botón Continuar partida:** este botón deberá redirigir a la aplicación web multiusuario para simplemente continuar la partida que se dejó en curso.

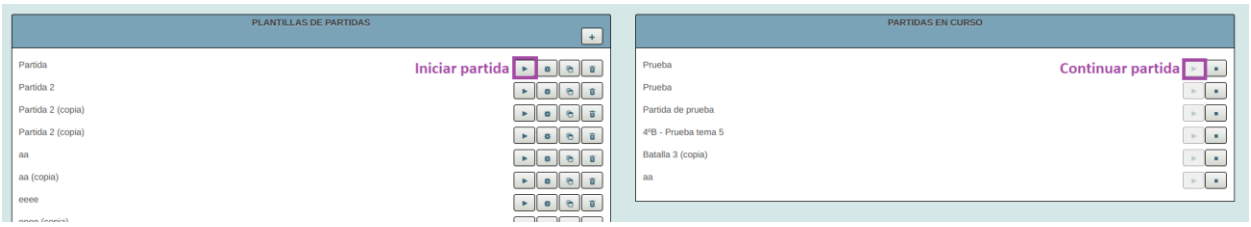


Figura 50: línea de mejora - Aplicación web multiusuario

REFERENCIAS

[1] Proyecto Descartes, «Proyecto Descartes,» [En línea]. Available: http://newton.proyectodescartes.org/. [Último acceso: 09 04 2020].

[2] Proyecto Descartes, «Aplicación de Juegos Didácticos en el Aula (AJDA),» [En línea]. Available: http://newton.proyectodescartes.org/juegosdidacticos/index.php?lang=es. [Último acceso: 09 04 2020].

[3] PrimeTek Informatics, «Primefaces,» [En línea]. Available: https://www.primefaces.org/showcase/. [Último acceso: 21 04 2020].

[4] Wikipedia, «Wikipedia - JSF,» [En línea]. Available: https://es.wikipedia.org/wiki/JavaServer_Faces.

[5] Universidad de Alicante, «Introducción a JavaServer Faces,» [En línea]. Available: http://www.jtech.ua.es/j2ee/publico/jsf-2012-13/sesion01-apuntes.html#Caracter%C3%ADsticas+de+JSF.

[6] PostgreSQL Global Development Group, «PostgreSQL,» [En línea]. Available: https://www.postgresql.org/. [Último acceso: 4 Junio 2020].

[7] Wikipedia, «JPA Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Java_Persistence_API. [Último acceso: 4 Junio 2020].

[8] Oracle, «Oracle JPA,» [En línea]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html. [Último acceso: 4 Junio 2020].

[9] json.org, «Json Org,» [En línea]. Available: https://www.json.org/json-es.html. [Último acceso: 4 Abril 2020].

[10] M. Lippert, «Wiki Spring Tool Suite 3 - Git Hub,» [En línea]. Available: https://github.com/spring-projects/toolsuite-distribution/wiki/Spring-Tool-Suite-3.

[11] pgAdmin, «pgAdmin,» [En línea]. Available: https://www.pgadmin.org/.

[12] Wikipedia, «Wikipedia - JavaDoc,» [En línea]. Available: https://es.wikipedia.org/wiki/Javadoc#:~:text=Javadoc%20es%20una%20utilidad%20de,los%20IDEs%20los%20generan%20autom%C3%A1ticamente.. [Último acceso: 5 Julio 2020].

[13] Wikipedia, «Wikipedia Swagger,» [En línea]. Available: https://en.wikipedia.org/wiki/Swagger_(software). [Último acceso: 5 Julio 2020].

[14] Trajano US, «Instalación y ejecución de la máquina virutal de linux,» [En línea]. Available: http://trajano.us.es/~fjfj/mv/maq_virtual.html.

[15] Mas Linux, [En línea]. Available: https://maslinux.es/guia-de-instalacion-completa-de-tomcat-en-gnu-linux/.

ANEXO A: MANUAL DE INSTALACIÓN

1. Entorno de Desarrollo

El entorno de desarrollo sobre el que se debe trabajar es la máquina virtual que proporciona el departamento de telemática. Los pasos que se detallarán a continuación son los mismos que se encuentran en la web del departamento [14]. Por otro lado, se debe usar el usuario dit, pues tiene toda la configuración necesaria.

A continuación, se van a detallar los diferentes pasos que se han de realizar para la instalación, configuración y ejecución de la máquina virtual en el sistema operativo Windows. Prácticamente los mismos pasos también son aplicables a cualquier otro sistema operativo. La máquina virtual suministrada contiene una copia del Linux instalado en el CDC, con el software usado en las prácticas.

La máquina virtual tiene configurada las siguientes parejas de usuario/clave:

- salas / salas
- dit / dit
- root / root

Fase 0. Activar la virtualización por hardware Intel VT-x/AMD-v en la BIOS del ordenador

Reinicie su ordenador, pulse la tecla para entrar en la configuración de la BIOS (depende de cada ordenador: ESC, F10,...) y active la virtualización por hardware.

Fase 1. Descarga de Ficheros

1. **Descargar el software de virtualización a utilizar.** Elija uno de los siguientes 2 programas de virtualización:
    a. VirtualBox (descargue también el último VirtualBox Extension Pack)
    b. VMware Player (requiere un sistema operativo de 64 bits)
2. **Descargar el programa usado para descomprimir la máquina virtual**
    a. **7-zip**
3. **Descargar la máquina virtual comprimida, en el directorio recomendado "C:\DIT\"**
    a. Descarga directa: **UbuntuDit-18r2.7z**
    b. Verificación del archivo
    SHA256: 502B995FD443B52A4A0B34A6D535B0AA059BB896A8C90DC3BACBD2393C281A61

Existen muchos programas para verificar la integridad de los archivos. Por ejemplo, puede usar el mismo programa 7-zip o los programas md5sum, sha1sum y sha256sum de Linux.

Fase 2. Preparación del Software

1. Instalar el software de virtualización elegido, VirtualBox o VMware Player.
2. Si ha elegido usar VirtualBox, añada el VirtualBox Extension Pack. Desde el menú Archivo, seleccione Preferencias. En la ventana que se muestra, vaya a la categoría Extensiones y ahí añada el fichero descargado.
3. Instalar el software de compresión 7zip.
4. Descomprimir la máquina virtual descargada. Para ello, sobre el fichero "UbuntuDit-18r2.7z" pulsar el botón derecho y hacer clic en la entrada "Extraer aquí" dentro del submenú "7-Zip" del menú contextual de Windows.

Fase 3. Instalación del Entorno

Elija una de las siguientes 3 opciones:

1. En VirtualBox con la configuración por defecto:
    a. Arrancar el programa VirtualBox.
    b. Dentro del menú "Máquina", hacer clic sobre "Añadir..." (Ctrl A).
    c. En la ventana de navegación se debe indicar la ubicación del fichero "UbuntuDit-18r2.vbox". La ubicación de este fichero, si se han seguido las indicaciones anteriores, será "C:\DIT\UbuntuDit-18r2\". Por último, pulsar "Abrir".
    d. Para encender la máquina virtual sólo es necesario seleccionarla y pulsar el botón "Iniciar" (flecha verde).
    e. (Opcional) Tras arrancar la máquina, ejecute en un terminal el siguiente comando (se le pedirá la contraseña del usuario):
    sudo apt remove -y open-vm-tools-desktop open-vm-tools
2. En VirtualBox con configuración personalizada:
    a. Arrancar el programa VirtualBox.
    b. Dentro del menú "Máquina", hacer clic sobre "Nueva..." (Ctrl N).
    c. Pulsar "Siguiente >".
    d. Indicar el nombre de la nueva máquina virtual, por ejemplo "UbuntuDit-18r2".
    e. Indicar que el sistema operativo es "Linux" y la versión es "Ubuntu 64bits".
    f. Pulsar "Siguiente >".
    g. Indicar la cantidad de memoria que se le va a asignar a la máquina virtual, por ejemplo 4096 MB.
    h. Pulsar "Siguiente >".
    i. En la ventana de "Disco duro", indicar la opción "Usar un archivo de disco duro virtual existente". Tras ello, pulsar el icono que hay a la derecha de la lista desplegable.
    j. En la nueva ventana, pulsar el icono "Agregar". Aparecerá nuevamente otra ventana de navegación donde debemos indicar la ubicación del fichero "UbuntuDit-18r2.vmdk". La ubicación de este fichero, si se han seguido las indicaciones anteriores, será "C:\DIT\UbuntuDit-18r2\". Por último, pulsar "Abrir".

k. Señalar el "Disco Duro" con nombre "UbuntuDit-18r2.vmdk" y pulsar "Seleccionar".

l. Pulsar "Crear".

m. Para encender la máquina virtual sólo es necesario pulsar el botón "Iniciar".

n. (Opcional) Tras arrancar la máquina, ejecute en un terminal el siguiente comando (se le pedirá la contraseña del usuario):
`sudo apt remove -y open-vm-tools-desktop open-vm-tools`

3. En VMware Player:

a. Arrancar el programa VMware Player.

b. Hacer clic sobre "Open a Virtual Machine".

c. En la ventana de navegación se debe indicar la ubicación del fichero "UbuntuDit-18r2.vmx". La ubicación de este fichero, si se han seguido las indicaciones anteriores, será "C:\DIT\UbuntuDit-18r2\". Por último, pulsar "Abrir".

d. Pulsar "Play virtual machine". *Nota:* Una alternativa a los pasos anteriores es hacer doble clic sobre el archivo "C:\DIT\UbuntuDit-18r2\UbuntuDit-18r2.vmx".

e. En la siguiente ventana, indicar la opción "I copied it" y pulsar "OK".

f. Tras arrancar la máquina, ejecute en un terminal el siguiente comando (se le pedirá la contraseña del usuario):
`sudo vbox-uninstall-guest-additions`

g. Reinicie la máquina para terminar.

2. Postgresql

Configuración Postgresql

La máquina virtual del departamento ya tiene instalado postgreSQL. Además, también está configurado el usuario dit para poder crear bases de datos en postgresql, por tanto, solo será necesario arrancar el servidor postgresql. Abrimos un terminal y ejecutamos:

```
sudo service postgresql start
```

Creación BBDD

Creamos la base de datos para el proyecto con nombre “gamificacionBD”:

```
createdb gamificacionBD
```

Creación de tablas

Creación de las tablas a partir de los scripts SQL proporcionados en el paquete de entrega del proyecto.

```
cd <directorio de las tablas>
psql
dit=> \i creaTablas.sql
```

3. Tomcat v8.5

Necesitamos tener instalado el servidor Tomcat en su versión 8.5. Para ello debemos seguir los siguientes pasos: [15]

1. Instalar java 7 (se necesita Java 7 o superior):

```
sudo apt-get install openjdk-8*
```

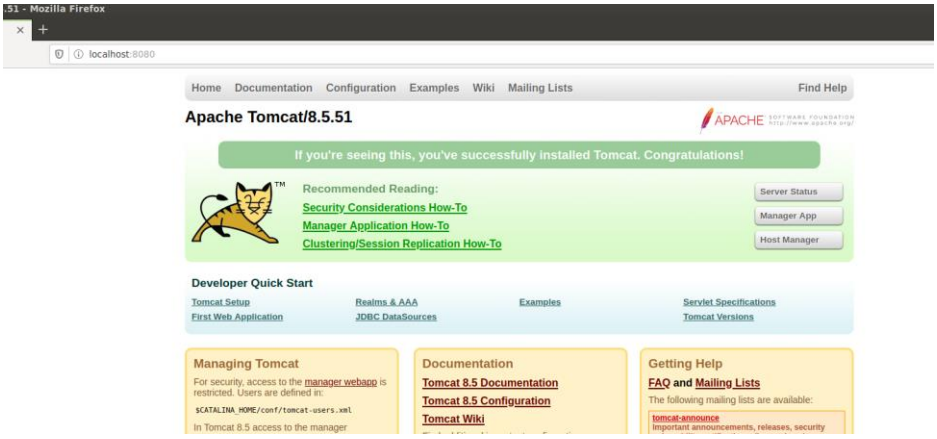
2. Instalar Tomcat 8.5

```
>> wget https://downloads.apache.org/tomcat/tomcat-8/v8.5.51/bin/apache-tomcat-8.5.51.tar.gz
>> tar -xvzf apache-tomcat-8.5.51.tar.gz
>> sudo mv apache-tomcat-8.5.51 /opt/tomcat
```

```
export CATALINA_HOME=/opt/tomcat/
```

3. Configurar CATALINA_HOME
4. Arrancar tomcat
5. Comprobar que tomcat está arrancado correctamente: acceder a http://localhost:8080/

```
$CATALINA_HOME/bin/startup.sh
```



6. Parar tomcat

```
$CATALINA_HOME/bin/shutdown.s
```

4. Spring Tool Suite

Instalación de STS

Paso 1: Accedemos a https://spring.io/tools3/sts/all

Paso 2: Descargamos la versión para Linux.

Paso 3: Copiamos el archivo descargado a /opt y descomprimimos el archivo.

```
cd Descargas
sudo mv spring-tool-suite-3.9.11.RELEASE-e4.14.0-linux-gtk-x86_64.tar.gz /opt
cd /opt/
sudo tar zxvf spring-tool-suite-3.9.11.RELEASE-e4.14.0-linux-gtk-x86_64.tar.gz
sudo ln -s /opt/sts-bundle/sts-3.9.11.RELEASE/STS /usr/local/bin/sts
```

Paso 4: Creamos el acceso directo

Creamos el fichero para configurar el acceso directo:

```
cd
gedit stsLauncher.desktop
```

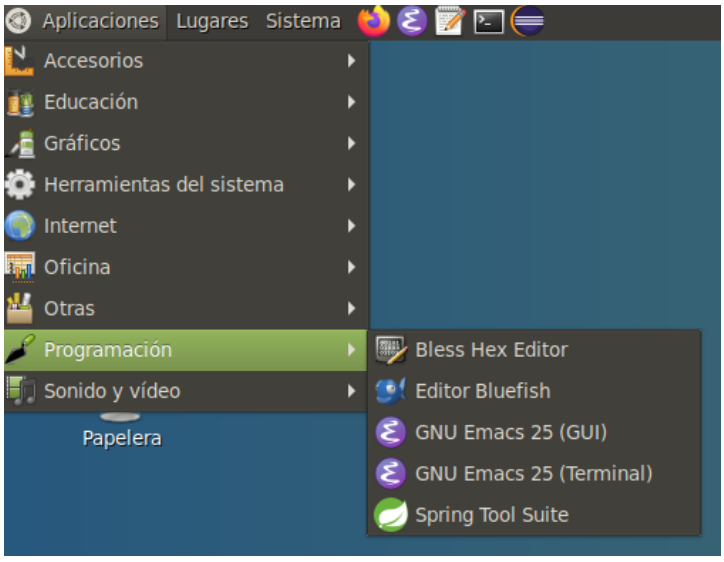
Se nos abrirá el editor de textos *gedit* con un documento en blanco. Debemos escribir el siguiente código[1]:

```
[Desktop Entry]
Name=Spring Tool Suite
Comment=Spring Tool Suite 3.9.11
Exec=/opt/sts-bundle/sts-3.9.11.RELEASE/STS
Icon=/opt/sts-bundle/sts-3.9.11.RELEASE/icon.xpm
StartupNotify=true
Terminal=false
Type=Application
Categories=Development;IDE;Java;
```

Lo movemos al directorio correcto:

```
sudo mv stsLauncher.desktop /usr/share/applications/
```

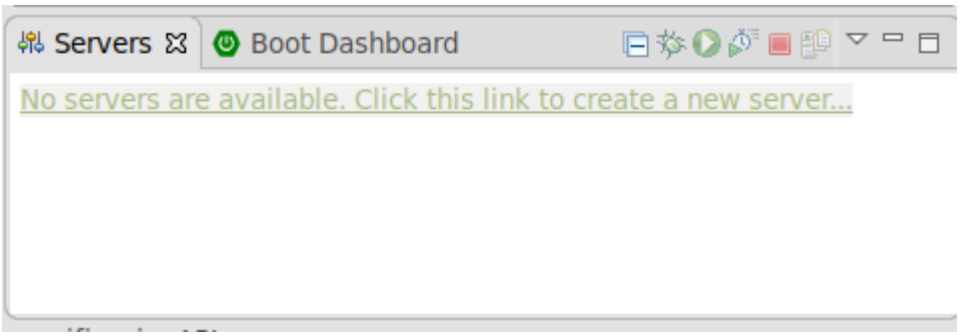
Resultado: encontraremos el STS en Aplicaciones > Programación



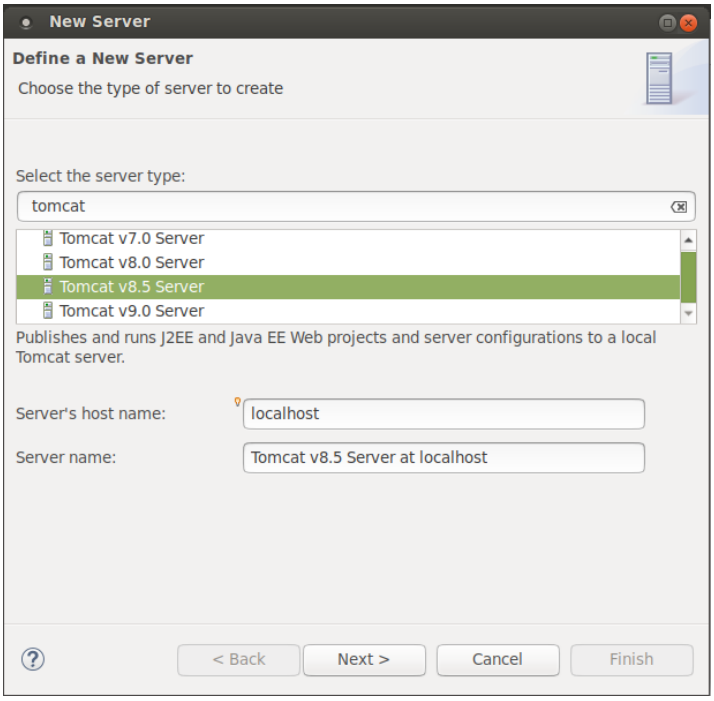
Configuración del servidor web (Tomcat)

1. En el STS nos vamos a la pestaña servers y clikcamos en *'No servers are avaible. Click this link to créate a new server…'*

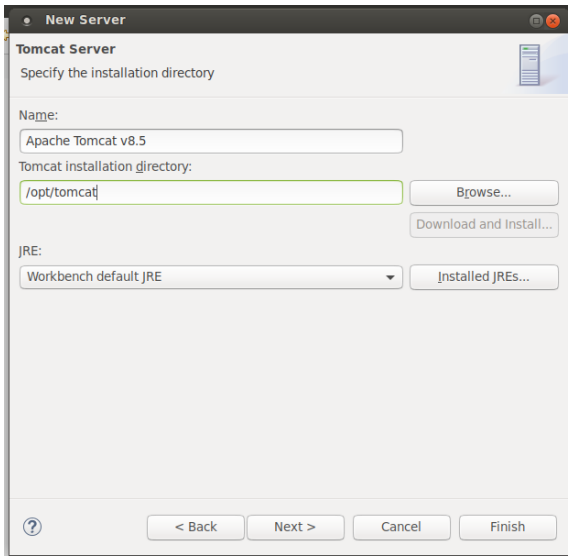
[1] La versión del STS puede variar, por tanto, adaptar el código si fuera necesario.



2. Buscamos la versión Tomcat que tengamos instalada. En nuestro caso Tomcat 8.5

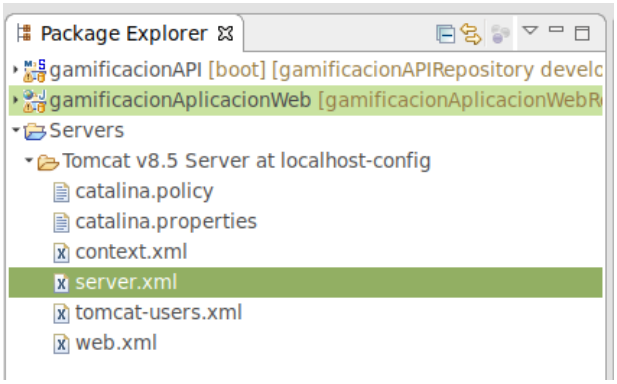


3. Completamos el directorio de instalación: '/opt/tomcat/'



4. Pulsamos en Finish y ya tendremos el servidor Tomcat configurado.

5. El servidor Tomcat de la API REST se ejecutará por defecto en el puerto 8080. Por tanto, vamos a cambiar el puerto de escucha del servidor Tomcat de la Aplicación web para que no haya conflictos. Abrimos el fichero server.xml del servidor:



6. Buscamos la etiqueta <connector> y cambiamos el atributo port. Por ejemplo, en este caso hemos puesto el puerto 8081.

```
-->
<Connector connectionTimeout="20000" port="8081" protocol="HTTP/1.1" redirectPort="8443"/>
<!-- A "Connector" using the shared thread pool-->
```

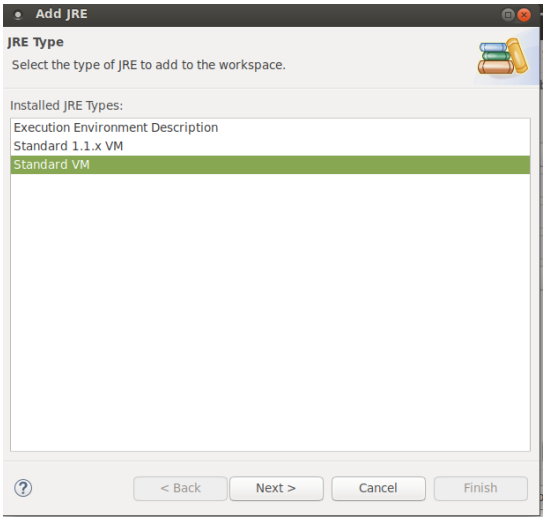
Instalación y configuración Java 8

Para este proyecto necesitamos tener instalado Java 8, pues si no tendremos problemas de compatibilidad.

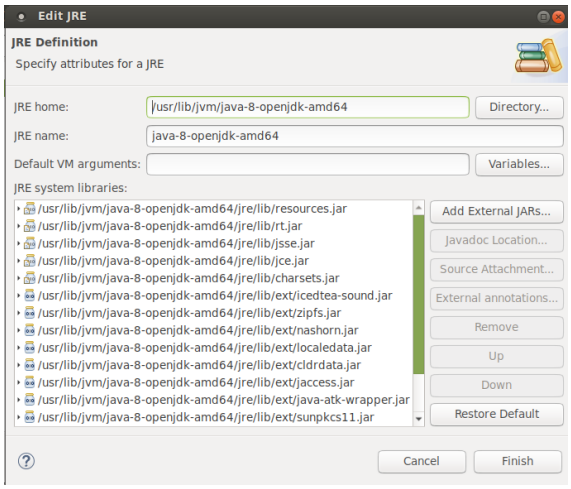
1. Instalamos Java 8:

```
sudo apt install openjdk-8-jdk
```

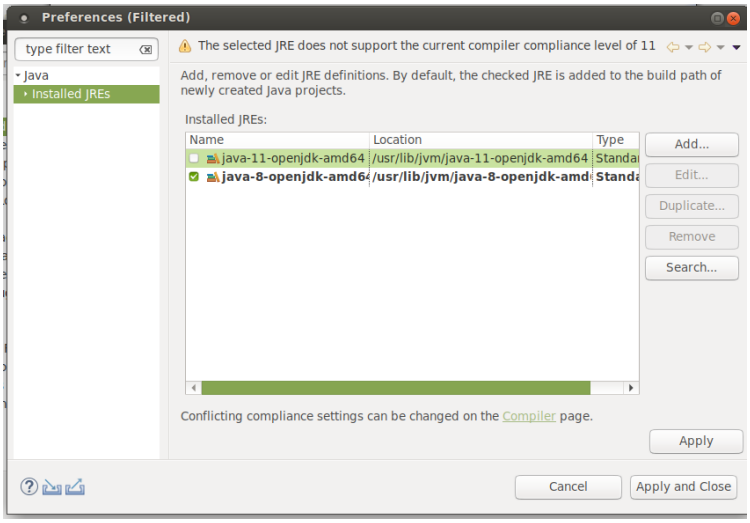
2. Nos vamos al STS.
3. Accedemos a Window > Preferences
4. Seleccionamos Java > Installed JREs
5. Pulsamos sobre Add…
6. Como tipo de JRE seleccionamos *Standard VM*



7. Configuramos el JRE home con el directorio home del java 8 recién instalado y finalizamos.



8. El resultado final debe ser el siguiente. Debemos tener la librería de java 8 marcada, para su uso por defecto.



Importar los proyectos

En el paquete de entrega del proyecto tenemos las siguientes carpetas:

- **gamificacionAPI**: es la carpeta contenedora del proyecto de la API REST, exportado como File System.
- **gamificacionAplicacionWeb**: es la carpeta contenedora del proyecto de la Aplicación web, exportado como File System.
- **LIB_HOME:** contiene todas las librerías necesarias para ambos proyectos.

Pasos para la correcta importación de los proyectos:

1. Copiamos el directorio LIB_HOMBE en /home/dit/
2. Creamos el proyecto para gamificacionAPI en el STS.
    a. File > New… > Project…
    b. Seleccionamos un proyecto general (General > Project). Pulsamos en Next.
    c. Ponemos nombre al proyecto. Project Name: gamificacionAPI.

    d. Pulsamos en Finish.

3. Importamos gamificacionAPI en el STS.
    a. File > Import…
    b. Seleccionamos General > File System
    c. From directory: gamificacionAPI (seleccionamos nuestro proyecto del paquete de entrega)
    d. Seleccionamos la raíz del proyecto.
    e. Into folder: seleccionamos el proyecto que hemos creado en el paso 2.
    f. Pulsamos Finish.
4. Creamos el proyecto para gamificacionAplicacionWeb en el STS.
    a. File > New… > Project…
    b. Seleccionamos un proyecto general (General > Project). Pulsamos en Next.
    c. Ponemos nombre al proyecto. Project Name: gamificacionAplicacionWeb.
    d. Pulsamos en Finish.
5. Importamos gamificacionAplicacionWeb en el STS.
    a. File > Import…
    b. Seleccionamos General > File System
    c. From directory: gamificacionAplicacionWeb (seleccionamos nuestro proyecto del paquete de entrega)
    d. Seleccionamos la raíz del proyecto.
    e. Into folder: seleccionamos el proyecto que hemos creado en el paso 4.
    f. Pulsamos Finish.

ANEXO B: MANUAL DE USUARIO

En este anexo encontramos el Manual de usuario de la ***Aplicación Web para la gestión de usuarios, juegos y partidas.***

1. Pantalla de acceso

El acceso a la aplicación se realiza a través de la pantalla de acceso que se muestra en la Figura 51. Esta pantalla nos ofrece tres funcionalidades: registrarse en la aplicación, acceder con un usuario y contraseña y recibir un correo electrónico con una nueva contraseña.

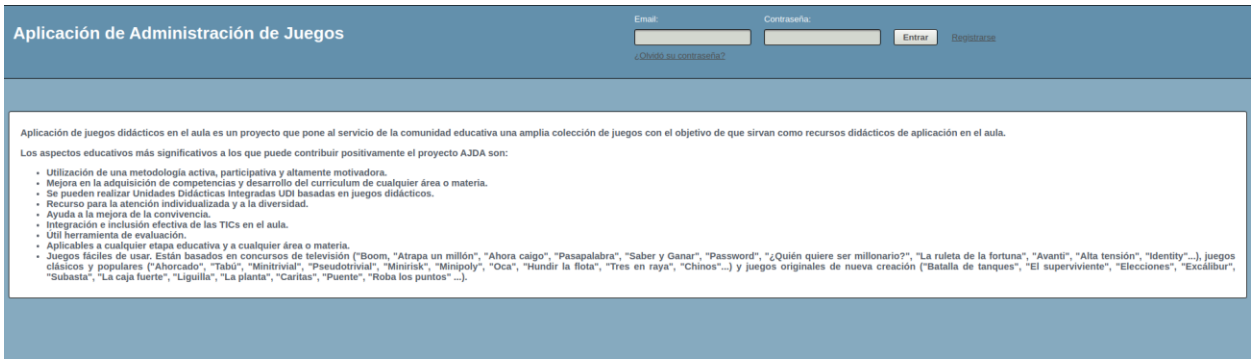


Figura 51: Anexo B - Pantalla de acceso

Registrarse

Permite crear un usuario con el que tener acceso a la aplicación. Se deben seguir los siguientes pasos:

1. Clicar en *Registrarse*
2. Completar el siguiente formulario:

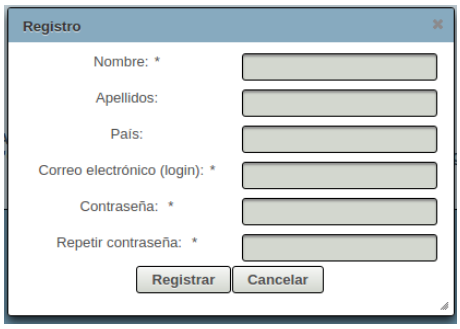


Figura 52: Anexo B - Formulario de registro

Condiciones:

- Los campos marcados con un asterisco son campos obligatorios que se deben completar.
- La contraseña debe tener al menos 8 caracteres.
- El correo electrónico introducido será nuestro usuario de acceso a la aplicación. No puede introducirse un correo ya registrado en la aplicación.

3. Pulsar en *Registrar*

¿Olvidó su contraseña?

Recibirá un correo electrónico con una nueva contraseña autogenerada al email de login. Pasos:

1. Introducir el correo electrónico para el que queremos recuperar la contraseña en el campo *email*.
2. Pulsar sobre *¿Olvidó su contraseña?*

Entrar

1. Introducir el usuario (correo electrónico de registro) en el campo email y su correspondiente contraseña en el campo *Contraseña.*
2. Clicar en el botón *Entrar*

2. Funcionalidades de usuario

En la Figura 53 están indicados los diferentes botones que ofrecen funcionalidad al usuario. A continuación, se detallarán los pasos a completar para llevar a cabo cada una de las acciones.

Figura 53: Anexo B - Pantalla principal

Nueva plantilla

Permite realizar la configuración para crear una nueva plantilla. Pasos a seguir:

1. Botón nueva plantilla
2. Completar la pestaña *General* (Figura 54):
    a. Campo título (obligatorio): título que identificará a la plantilla en el listado de la pantalla principal.
    b. Campo etapa: define la etapa educativa asociada a la plantilla
    c. Campo curso: curso educativo asociado a la plantilla
    d. Campo asignatura: asignatura asociada a la plantilla.
    e. Campos tema: tema de evaluación y sobre el que serán las preguntas del fichero de preguntas.

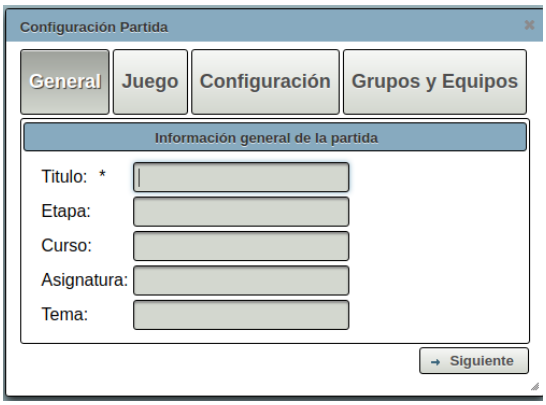


Figura 54: Anexo B - Nueva plantilla - General

3. Pulsar en *Siguiente.*
4. Seleccionar un juego del listado. Se puede filtrar previamente para acotar el listado de juegos. (Figura 55). **Este paso es obligatorio**.
    a. Una vez seleccionado el juego podemos volver a elegir otro pulsando sobre el botón *Cambiar juego.*

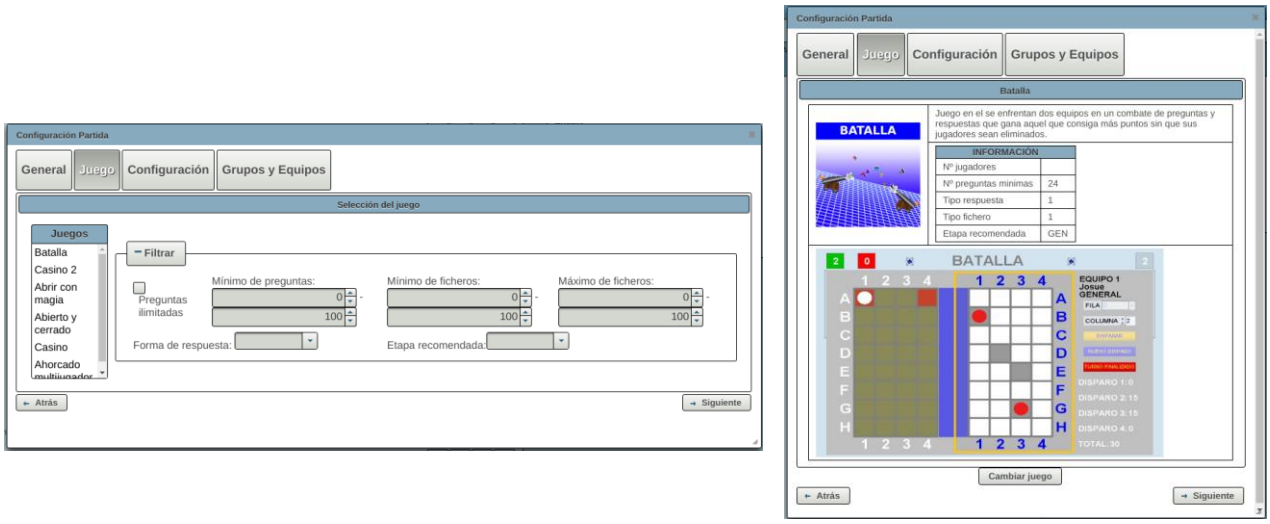


Figura 55: Anexo B - Nueva plantilla - Juego

5. Pulsamos en siguiente.
6. Completamos la pestaña Configuración. En este caso no hay ningún campo obligatorio.

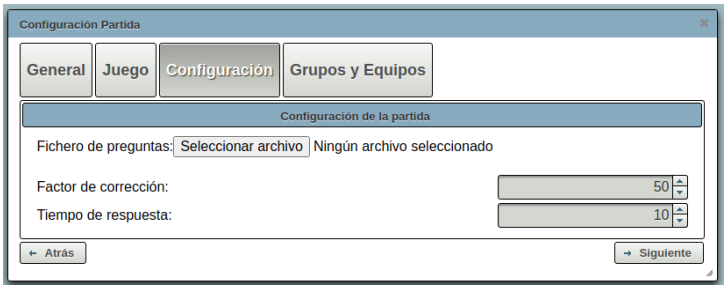


Figura 56: Anexo B - Nueva plantilla - Configuración

7. Pulsamos en siguiente.

8. Completar formulario de Grupos y Equipos. Según el juego seleccionado hay tres opciones de formulario:
    a. Sin configuración de grupos y equipos: no hay que completar ningún campo.

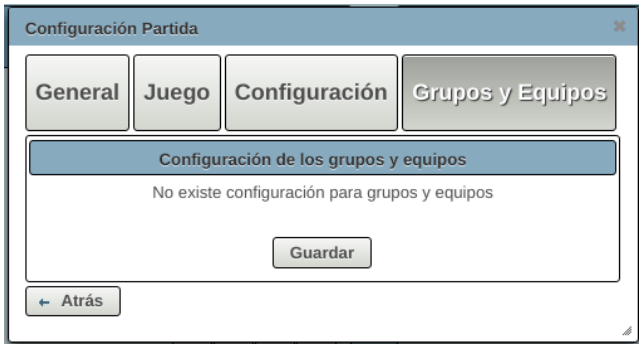


Figura 57: Anexo B - Nueva plantilla - Grupos y Equipos I

    b. Solo grupos: el juego solo admite dividir los jugadores en grupos. Se tendrá que seleccionar el número de grupos que se quieren configurar y sus correspondientes nombres.

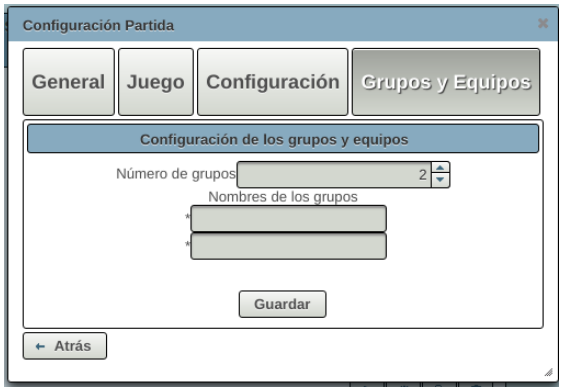


Figura 58: Anexo B - Nueva plantilla - Grupos y Equipos II

    c. Equipos formados por grupos: para juegos que divide a los jugadores en grupos y a su vez los grupos en equipos.
        i. Seleccionar el número de equipos
        ii. Completar el nombre de cada equipo
        iii. Seleccionar el número de grupos de cada equipo
        iv. Completar el nombre de cada grupo

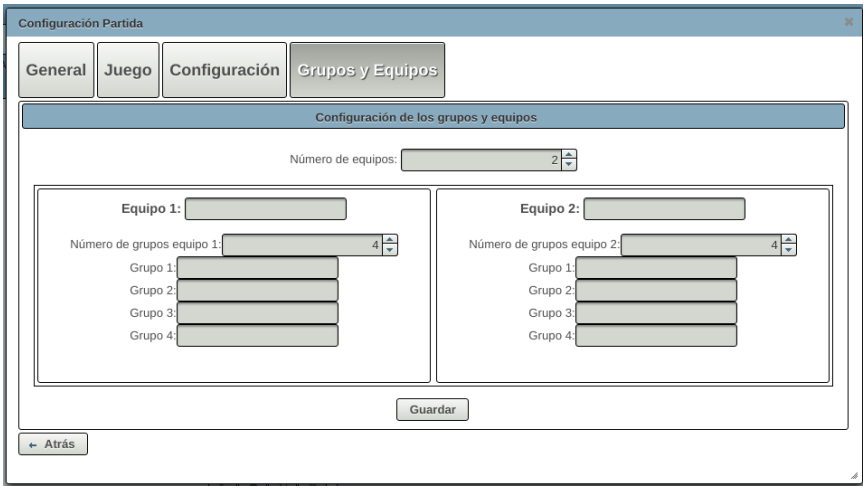


Figura 59: Anexo B - Nueva plantilla - Grupos y Equipos III

9. Pulsar sobre guardar

Modificar plantilla

Permite modificar una plantilla ya existente. Pasos a seguir:

1. Pulsar sobre el botón *Modificar plantilla*.
2. Seguir el proceso detallado en el apartado anterior *Nueva plantilla.*

Duplicar plantilla

Permite duplicar la configuración de una plantilla ya existente. Pasos a seguir:

1. Pulsar sobre el botón *Duplicar plantilla*.

Eliminar plantilla

Permite eliminar una plantilla existente. Pasos a seguir:

1. Pulsar sobre el botón *Eliminar plantilla.*
2. Pulsar sobre el botón *Sí* para confirmar la eliminación.

Iniciar partida

Permite comenzar a jugar una partida a partir de una plantilla.

1. Pulsar sobre Iniciar partida.

Terminar partida

Permite finalizar una partida que se inició y se dejó a medias.

1. Pulsar sobre Terminar partida.

Salir

Permite al usuario salir de la aplicación, desloggeando el usuario.

1. Pulsar sobre el botón de Salir

3. Funcionalidades de administrador

El administrador además de todas las funcionalidades anteriores también tiene la capacidad de realizar la gestión de juegos. En la Figura 60 están indicados los diferentes botones que ofrecen funcionalidad para la gestión de juegos. A continuación, se detallarán los pasos necesarios para llevar a cabo las diferentes acciones.

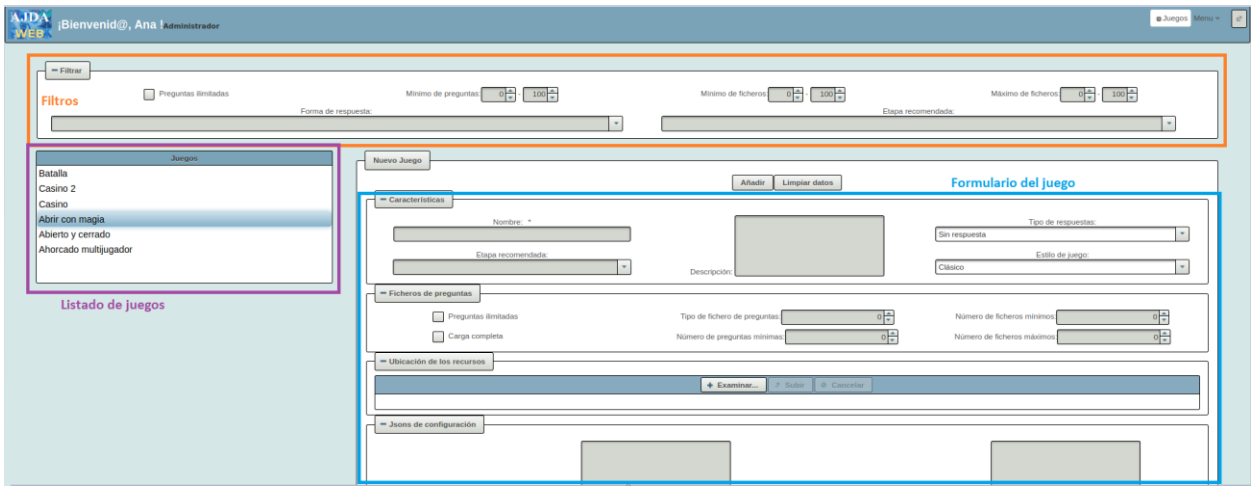


Figura 60: Anexo B - Gestión de juegos

Nuevo juego

Permite añadir un nuevo juego a la plataforma. Pasos a seguir:

1. Ir a Gestión de juegos.
2. Rellenar el formulario.
    a. Nombre es un campo obligatorio. Identifica al nombre del juego.
    b. Ubicación de los recursos. En esta sección se debe adjuntar el .zip que contiene el juego y todo su contenido (imágenes, capturas, etc.). Pasos para adjuntar:
        i. Examinar… y seleccionar el archivo o arrastrar en el recuadro.
        ii. Pulsar en el botón Subir.
3. Pulsar el botón añadir.

Modificar juego

Permite modificar un juego de los existentes en el listado de juegos. Pasos a seguir:

1. Ir a Gestión de juegos.
2. Seleccionar el juego del listado.
3. Modificar los campos deseados en el formulario.
4. Pulsar sobre el botón Guardar cambios.

Eliminar juego

Permite eliminar un juego de los existentes en el listado de juegos. Pasos a seguir:

1. Ir a Gestión de juegos.
2. Seleccionar el juego del listado.
3. Pulsar el botón Eliminar.

Además de las funcionalidades detalladas, tenemos una sección de filtros mediante los cuales se puede limitar el listado de juegos. Por otro lado, el botón Limpiar datos, sirve para borrar los datos del formulario y deseleccionar el juego seleccionado.

4. Funcionalidades de superusuario

El superusuario además de todas las funcionalidades anteriores, también tiene la capacidad de realizar la gestión de usuarios. En la Figura 61 están indicados los diferentes botones que ofrecen funcionalidad para la gestión de usuarios. A continuación, se detallarán los pasos necesarios para llevar a cabo las diferentes acciones.

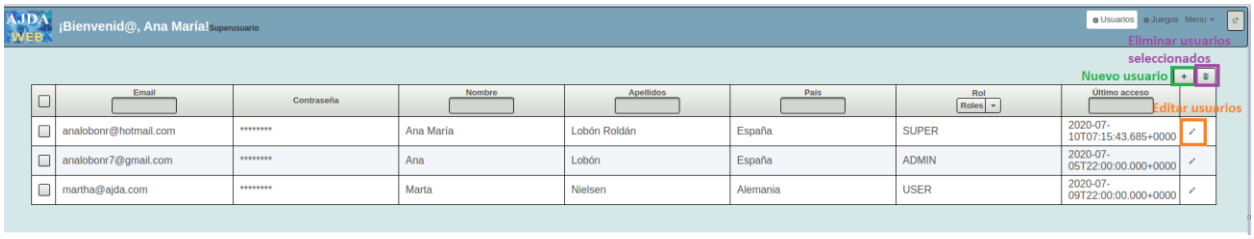


Figura 61: Anexo B - Pantalla de gestión de usuarios

Nuevo usuario

Permite crear un nuevo usuario. En este caso el superusuario puede seleccionar el rol que tendrá dicho usuario. Además, al usuario registrado le llegará un correo electrónico con su contraseña de acceso, la cual es autogenerada. Pasos a seguir:

1. Pulsar sobre el botón nuevo usuario.
2. Completar el formulario que podemos ver en la Figura 62. Condiciones:
    a. Los campos Nombre y Correo electrónico son obligatorios.

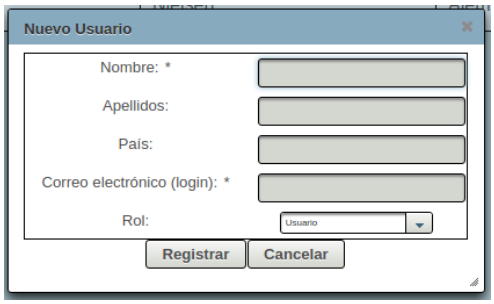


Figura 62: Anexo B - Formulario de nuevo usuario

3. Pulsar sobre el botón registrar.

Editar usuarios

Permite editar la información de un usuario. Pasos a seguir:

1. Pulsar sobre el botón de Editar usuarios.
2. Modificar los campos editables:
    a. Correo electrónico
    b. Nombre
    c. Apellidos
    d. País
    e. Rol

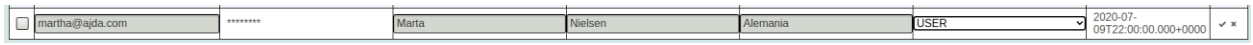


Figura 63: Anexo B - Modificar usuario

3. Pulsar sobre el tick de confirmación

Eliminar usuario seleccionados

Permite eliminar los usuarios seleccionados. Pasos a seguir:

1. Seleccionar usuarios pulsando sobre el checkbox de la tabla.

Email	Contraseña	Nombre	Apellidos	País	Rol	Último acceso	
analobonr@hotmail.com	********	Ana María	Lobón Roldán	España	SUPER	2020-07-10T08:48:53.889+0000	
analobonr7@gmail.com	********	Ana	Lobón	España	ADMIN	2020-07-05T22:00:00.000+0000	
martha@ajda.com	********	Marta	Nielsen	Alemania	USER	2020-07-09T22:00:00.000+0000	

Figura 64: Anexo B - Selección de usuario a eliminar

2. Pulsar en el botón de Eliminar usuarios.
3. Pulsar sí en el diálogo de confirmación de eliminación.