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

# Diseño y configuración de un entorno para la práctica con entrega continua

Autor: Francisco Javier Gómez Bellido

Tutor: Isabel Román Martínez

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

Sevilla, 2020

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

# Diseño y configuración de un entorno para la práctica con entrega continua


Autor:

Francisco Javier Gómez Bellido

Tutor:

Isabel Román Martínez

Profesora Colaboradora

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



Sevilla, 2020

Trabajo Fin de Grado: Diseño y configuración de un entorno para la práctica con entrega continua

Autor: Francisco Javier Gómez Bellido
Tutor: Isabel Román Martínez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

# Agradecimientos

*El amor se siente con el corazón, no con el cuerpo.*

Gabriel García Márquez

Me gustaría empezar agradeciendo a la Escuela Técnica Superior de Ingeniería (ETSI) y a todas las personas que hacen que esto sea posible. En especial a Isabel Román Martínez, muchas gracias por tu apoyo, por mostrarme qué es la constancia y el trabajo bien hecho.

Cuando mi etapa en la escuela llegaba a su fin, Manuel Morales Castro me ofrecía una oportunidad para iniciar mi siguiente etapa, una etapa que me ha hecho crecer como persona, tanto a nivel personal como a nivel profesional, por esto, estaré eternamente agradecido a Manuel Morales y a todos mis compañeros de 'Workshare' por ayudarme a ser la persona que soy hoy en día.

A lo largo de estos años he conocido a muchas personas, las cuales nos hemos estado acompañando en esta aventura, apoyándonos y sacándonos una sonrisa y unas risas cada vez que podíamos. Porque cada uno de vosotros forma parte de mí, José Enrique Romero Olías, Valle Naranjo Cano, Manuel Ángel Jiménez Quesada, Assumpta María Cabral Otero, Luis Chacón Palos, a toda la familia que me llevo del grado y de aquel primer año en la residencia Cartuja II, GRACIAS.

Una vez hablado de amigos, debo agradecer a mis compañeros de vida durante estos 6 (que deberían haber sido 4) años, por toda la ayuda que me habéis dado siempre, por haber estado a mi lado, guiándome y apoyándome, intentando sacar lo mejor de mí en todo momento. Sois muy grandes, por todos los momentos que hemos vivido juntos y los que quedan, Guillermo Palomino Lozano, Abraham Pérez Hernández, Ildefonso Jiménez Silva, Sergio Román Gonzales, Daniel Vela Calderón, Diego López Morilla.

Seguimos yendo un poco más y ahora es el turno de más que un amigo, un hermano, Jesús Macías Félix, quién siempre me ha apoyado, ha estado conmigo siempre y pase lo que pase, sabré que puedo contar con él, gracias por estar siempre.

Llegados a este punto, no me podía quedar sin agradecer, a las personas que me trajeron al mundo, cuidado y guiado hacía el camino a seguir para ser la mejor persona posible, estaré siempre eternamente agradecidos al igual que a mis hermanos, Jesús y Ana, gracias por aguantarme y ayudarme en todo momento, os quiero.

Por último, pero el agradecimiento más grande, quiero agradecer a una persona que apareció en mi vida hace 3 años, el destino decidió no ponernos nada fácil, pero aún así, nunca nos hemos separado, ¿es eso amor? Hemos sabido esperar y nuestro momento llegó, gracias a ti, María Laó Cañadas, estoy escribiendo ahora mismo este documento, cerrando una etapa, juntos. Te quiero, por todo lo que hemos vivido, estamos viviendo y nos queda por vivir. Desde Fresa a ., paseando por el río y pasando por Londres, te.

*Francisco Javier Gómez Bellido*
*Desde Jerez a ., pasando por Sevilla y Londres*

*Sevilla, 2020*

# Resumen

El uso de nuevas metodologías para la gestión de proyectos software, conocidas como metodologías 'ágiles'se encuentran en auge hoy en día. Empresas punteras en el sector TIC como Google, Facebook o Spotify llevan a cabo el desarrollo de sus productos siguiendo metodologías de este tipo. En el Grado de Ingeniería de las Tecnologías de Telecomunicación numerosos estudiantes continuan su carrera profesional dedicándose al desarrollo software, por esta razón aquí se realiza una revisión teórica de las tecnologías que permiten llevar a cabo un desarrollo ágil y además la preparación de una práctica para la asignatura 'Ingeniería de Software'que ofrezca a los alumnos una visión práctica sobre qué son las metodologías ágiles y cómo llevarlas a cabo a la hora de desarrollar. El entorno preparado para esta práctica puede ser de gran utilidad en otras prácticas de la asignatura u otras asignaturas relacionadas con 'DevOps'.

La realización de la práctica se desarrolla en un entorno, preparado en este proyecto, en el que los alumnos podrán llevar a cabo un desarrollo ágil, modificar el código, ejecutar pruebas automatizadas y desplegar los cambios en un nuevo entorno de pruebas con solo crear una petición de integración. Para ello, se ha partido de una aplicación web desarrollada en *Java* previamente por un alumno en su trabajo fin de grado [19]. Sobre esta aplicación se han desarrollado pruebas unitarias automatizadas y se ha configurado el proyecto para ser gestionado con *Apache Maven*, tecnología de la que se habla en la revisión teórica realizada en este trabajo. Posteriormente se han configurado dos servidores: *Tomcat*, para poder desplegar la aplicación, y *Gitlab* que será el encargado de almacenar los repositorios de la práctica y permitir la ejecución de tareas automatizadas sobre el proyecto, para que cuando se cree una petición de integración, se ejecuten las pruebas y se desplieguen los cambios sobre el entorno de pruebas. Además se ha realizado la puesta en marcha de un servicio que realiza copias de seguridad periódicas sobre *Gitlab* y un script en el que mediante *Rake Tasks* se borran los datos del servidor e inicializa lo necesario para la realización de la práctica.

La práctica busca que los alumnos desarrollen modificaciones sobre el código de la aplicación, para ello deberán crear la tarea equivalente a un requisito dado, realizar los cambios necesarios sobre el repositorio, crear una petición de integración e integrar los cambios en la rama principal de desarrollo.

Los alumnos podrán trabajar de forma directa con las metodologías y conceptos que se trabajan en la revisión teórica y aprender cómo funcionan, para que puedan aplicarlos en su futuro profesional.

# Abstract

Nowadays the usage of some new methodologies to manage projects is increasing, they are known as agile methodologies. Some of the most important technology companies as Google, Facebook, Spotify are using an agile methodology in the development of their products. In the Telecommunications Engineering bachelor's degree, there is a high number of students who will continue his career developing software, for that reason in this project has a theoretical review of some of the technologies useful to do agile development and a practice for the 'Ingeniería de Software'subject which will help the students to learn what are the agile methodologies and how it can be used for development, the developed environment could be useful for other practices from that subject or any other subject related with 'DevOps'.

For the practice it has been made an environment where the students can do agile development, they could modify the code, run automated tests and deploy the changes to the test environment just creating a merge request. To do that, it has been based on a web application developed previously by a student for his thesis using *Java*, this application is focused to teach. Over this application, it has been made automated tests and it has been configured to be managed using *Apache Maven*, which has been reviewed in the theoretical review of this project. After that the *Tomcat* server has been configured to be able to deploy the application and the *Gitlab* server to store the repositories of the practice and it will allow the students to run automated tasks in the project which will allow to run the tests and deploy the application to a test environment when a merge request is created. Furthermore a service to create *Gitlab* backups periodically has been configured and the creation of a script that uses *Rake Tasks* to erase the data from the server and it initialize the necessary data for the realization of the practice.

The practice requires the students to make some changes to the application, the students so they should create the corresponding task for a specific requirement, they should make the changes to the code, create a merge request and merge those changes into the principal development branch.

The students will be able to work directly using the methodologies and concepts which have been worked in the theoretical review and they will learn how it works so they will be able to use them in their career.

# Índice

# 1 Introducción

*No te rindas, aun estas a tiempo*
*de alcanzar y comenzar de nuevo,*
*aceptar tus sombras, enterrar tus miedos,*
*liberar el lastre, retomar el vuelo.*

MARIO BENEDETTI

Este TFG ha sido desarrollado con el objetivo de realizar una revisión teórica y preparar una práctica para los alumnos del Grado de Ingeniería de las Tecnologías de Telecomunicación sobre los conceptos más populares en la actualidad relacionados con la gestión de proyectos usando metodologías ágiles, en concreto, sobre aplicaciones web desarrolladas usando el lenguaje de programación *Java*.

## 1.1 Motivación

A medida que el alumno avanza en el Grado de Ingeniería de las Tecnologías de Telecomunicación (a partir de ahora GITT) va ampliando su conocimiento sobre el desarrollo software, lo que le capacita para extender sus conocimientos en el futuro y trabajar en el sector. Hay una corriente del desarrollo software que está adquiriendo gran popularidad, esta área es el desarrollo ágil. Mostrar al alumno los fundamentos de estos modelos de proceso, y algunos recursos de interés para dar soporte a las actividades implicadas en los mismos, puede resultar muy interesante para su formación.

En este trabajo de fin de grado se va a plantear una práctica que consiga acercar al alumno al desarrollo ágil y a las herramientas usadas. Se parte de un trabajo de fin de grado previo [19], que fue desarrollado por Fabián Muñoz. En dicho proyecto se diseña y codifica una aplicación web desarrollada con tecnología *Servlet* de *Java*, conforme a patrones de diseño, con finalidad docente. En el presente trabajo se reutilizará este código.

En este trabajo se verán conceptos y contenidos útiles para que el alumno pueda desempeñar una actividad laboral en una empresa del sector del desarrollo software. Esto le facilitará su integración en un equipo que lleve a cabo un desarrollo ágil.

En el GITT se trabaja el desarrollo software de forma teórica y práctica en varias asignaturas, a continuación, se encuentran algunas de ellas y un resumen de los contenidos estudiados por el alumno en el grado, extraídos de los distintos programas de las respectivas asignaturas los cuales se pueden encontrar en [1].

- **Fundamentos de programación I.** Esta asignatura de primer curso sirve al alumno como introducción al desarrollo software, se trabaja el lenguaje de programación *C*, se realiza una introducción a algoritmos básicos en *C* y a sistemas operativos *UNIX*.
- **Fundamentos de programación II.** El alumno se inicia en un lenguaje de programación orientado a objetos como es *Java*.

- **Fundamentos de Internet.** El alumno trabaja los principios básicos de las redes y servicios de comunicación, aprendiendo los conceptos de arquitectura de capas, protocolos fundamentales relacionados con el funcionamiento de Internet.
- **Fundamentos de Aplicaciones y Servicios Telemáticos.** El alumno trabaja por primera vez con herramientas para la creación de aplicaciones Web, aplicaciones Web dinámicas y utilización de bases de datos.
- **Ingeniería de software.** Esta asignatura acerca al alumno a la gestión de software durante todo su ciclo de vida, y cómo llevarla a cabo, se le presentan métodos, procedimientos y recursos disponibles para realizar una correcta gestión de proyectos software.

Además de estas asignaturas, se cursan otras en el grado que aportan al alumno conocimientos sobre gestión de proyectos o el funcionamiento, gestión y administración de servicios telemáticos, gestión de bases de datos, etc. Algunas de estas asignaturas son:

- **Proyectos de telemática.** Se forma al alumno en la elaboración de proyectos dentro del ámbito de la ingeniería telemática.
- **Servicios Telemáticos Avanzados.** El alumno se capacita para instalar, administrar y gestionar los servicios de aplicación, y aprenderá el funcionamiento de los protocolos involucrados en dichos servicios y las principales técnicas de seguridad asociadas a los mismos.

Este es el contenido principal relacionado con el desarrollo software y gestión de proyectos y servicios telemáticos, como se puede observar, la única asignatura en la que el alumno trabaja la gestión íntegra del ciclo de vida del desarrollo software, así como sus herramientas es *Ingeniería de software*.

Como se ha visto anteriormente, el único contenido tanto práctico como teórico sobre metodologías ágiles es visto en la asignatura *Ingeniería de software* de forma teórica. En este Trabajo Fin de Grado se propone una práctica para la asignatura *Ingeniería de Software* que complemente la docencia del grado con conceptos sobre la gestión y realización del desarrollo ágil, y se facilitan herramientas que ayuden al alumno a llevarla a cabo. Con este proyecto el alumno usará un entorno de trabajo preparado para dar soporte a los procesos de integración y entrega continuas, que facilitan llevar a cabo un desarrollo ágil dentro de un supuesto grupo de trabajo. Se espera que este trabajo consiga, junto a la materia que se trabaja en el grado, aportar al alumno la capacidad de poder profundizar y llevar a cabo una gestión y/o desarrollo ágil en el ámbito profesional.

Este proyecto además refuerza los conocimiento vistos previamente de forma práctica en la asignatura *Ingeniería de Software*, ya que en ella se trabaja con herramientas como *Git*, entornos de desarrollo como *Eclipse* y se realizan pruebas sobre aplicaciones *Java*. Todas estas herramientas serán necesarias para el desarrollo de la práctica propuesta en este proyecto.

Uno de los objetivos que se intenta conseguir a través de este trabajo es la introducción al alumno al concepto 'DevOps' en un proyecto software.

DevOps, de sus iniciales en inglés 'Development and Operations', en español Desarrollo y Operaciones, es una práctica que tiene como objetivo unificar el desarrollo y la operación de software, por lo tanto, es la práctica en la cuál se realizan las tareas de instalar, gestionar y mantener los entornos de desarrollo y producción para facilitar al programador el desarrollo del proyecto, como se dice en [12], 'DevOps' se basa en la integración entre desarrolladores software y administradores de sistemas, permitiendo desarrollar software más rápidamente, con mayor calidad, menor coste y una altísima frecuencia de entregas.

## 1.2 Antecedentes

Para la realización de este trabajo, y por lo tanto la preparación de la práctica propuesta, se ha preparado un entorno que funcione correctamente para el proyecto realizado en [19].

En el trabajo previo [19], se trata el desarrollo de una aplicación Web usando distintos patrones de diseño conocidos, además de una buena estructura, comentada y documentada. Algunos de los patrones de diseño usados en la aplicación son:

- **Cliente-Servidor.** Este patrón consiste en la realización de una arquitectura en la que un cliente solicita funcionalidades ofrecidas por un servidor, basándose en una comunicación entre el cliente y el servidor mediante peticiones y respuestas. Entre las ventajas que proporciona el uso de este patrón de diseño se pueden encontrar:
  - **Integridad de los datos.** El servidor es el único elemento capaz de acceder y modificar los datos, por lo que tiene control total sobre las modificaciones que se realizan a los datos y puede restringir cualquier acción que pueda dañar la integridad de los mismos.
  - **Existencia de una interfaz que hace que el desarrollo del servidor sea invisible para el cliente.** La forma en la que está desarrollado el servidor puede ser modificada y actualizada ya que el cliente interactúa con la interfaz del servidor y esta no cambiaría.
- **Modelo-Vista-Controlador.** Este patrón está basado en la separación de la lógica de control, los modelos de datos y la presentación de la información al usuario, es usado por muchos de los marcos de trabajo más famosos en la actualidad, entre ellos *Ruby* on *Rails*. El *Modelo* es la representación software de los recursos del proyecto, sería el encargado de realizar peticiones a la base de datos, almacenar, representar y modificar los datos. La *Vista* es la encargada de darle forma a los datos y presentar todo esto al usuario, por último, el *Controlador*, encargado de recibir las órdenes del cliente, solicitar los datos necesarios y realizar las operaciones sobre los modelos, indicando a la vista el resultado de las operaciones realizadas. Algunas de las ventajas que aporta este patrón de diseño son:
  - La implementación se realiza de forma modular, además esto será útil para el futuro desarrollo de los módulos, si un módulo se actualiza no cambiará nada, ya que su interfaz no será modificada.
  - Facilita el control y manejo de errores.

Para la parte del cliente, el *frontend*, usa tecnologías web como *HTML*, *CSS*, *Javascript*, *JSP*, *EL* y *JSTL*. En el servidor, *backend*, se usa la tecnología *Servlet* de Java, además de *SQL* para la base de datos y *PostgreSQL* para su gestión.

## 1.3 Objetivos

En este TFG se pretende:

- Revisar teóricamente las tecnologías más populares en la actualidad para proyectos *Java*, así como metodologías de gestión de proyectos, profundizar en las metodologías ágiles y cómo llevarlas a cabo con mayor facilidad en el desarrollo de proyectos software.
- Analizar de forma crítica y extender el trabajo en el cuál este proyecto se ha basado [19], así como sus tecnologías, los patrones de diseño usados y el modelo de datos de la aplicación.
- Realizar pruebas automatizadas al proyecto previo [19], desplegar la aplicación en un servidor *Tomcat* y, además, configurar el proyecto para la utilización de *Apache Maven* para la ejecución de estas tareas y la gestión de dependencias.
- Preparar una práctica para los alumnos de la asignatura 'Ingeniería de Software' del 'Grado en Ingeniería de la Tecnología de la Telecomunicación' que les sirva como complemento a lo visto de forma teórica en la asignatura. Esto implica preparar un entorno donde alojar repositorios y poder ejecutar tareas automatizadas para la realización de desarrollo ágil en la práctica, configurar tareas automatizadas, realizar copias de seguridad periódicas sobre el servicio de alojamiento de repositorios y realizar un script de inicialización de los datos necesarios para la realización de la práctica.
- Desarrollar una memoria donde sea documentado todo el trabajo desarrollado en el proyecto.

## 1.4 Estructura del documento

Este documento explica en el segundo capítulo (2) las tecnologías usadas en este trabajo, con el objetivo de que aporten al lector los conocimientos mínimos requeridos para el entendimiento crítico y posible desarrollo sobre el proyecto. A continuación, en el capítulo número tres (3), se encuentra la preparación del entorno para la realización de la práctica, se documenta el escenario y todos los pasos necesarios para su preparación. En el último capítulo (4) se pueden encontrar las conclusiones alcanzadas una vez finalizado el trabajo y las posibles líneas futuras que pueden ser desarrolladas partiendo de él.

# 2 Estado de la técnica

*If it hurts, do it more often, and bring the pain forward.*

JEZ HUMBLE

El siguiente capitulo se ha realizado con el objetivo de explicar todas las tecnologías usadas en el proyecto y facilitar al lector los conocimientos necesarios para entender el entorno desarrollado.

## 2.1 Sistema de control de versiones, *Git*

Un sistema de control de versiones es aquel que permite gestionar de forma fácil e interactiva las versiones de un proyecto software, por tanto, facilita la gestión de los cambios que se realizan en un proyecto o en la configuración de este. Actualmente se pueden encontrar varios sistemas de control de versiones como son *Git*, *Mercurial*, *Subversion*, etc. En este trabajo se va a utilizar *Git*, ya que es el sistema de control de versiones por excelencia, más extendido profesionalmente, así como el visto y usado en las prácticas de GITT.

En numerosas ocasiones se confunde *Git* (*Sistema de control de versiones*) con servidores para alojar proyectos software como *Github*, *Gitlab*, *Bitbucket*, etc.

*Git* es un sistema de control de versiones distribuido, es un proyecto open source desarrollado en 2005 por Linus Torvalds, fundador y mantenedor del núcleo Linux, surgió para ayudar en la colaboración del desarrollo de Linux. *Git* está pensado para proveer de eficiencia y confiabilidad en el mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

A continuación, se explican las principales funcionalidades y ventajas de usar un sistema de control de versiones [11], como *Git*, en un proyecto de desarrollo software.

- Historial de cambios, *Git* almacena los ficheros en todas sus distintas versiones, esto permite deshacer cambios si algo ha fallado, o incluso volver a una versión estable del producto.
- Relacionado con el punto anterior, y es que *Git* permite asociar etiquetas a versiones en el proyecto, lo que facilita su identificación para volver a ellas cuando se desee.
- Mejor colaboración entre desarrolladores, ya que se puede controlar el avance del código de los distintos miembros del equipo.
- Al intentar unir distintos cambios permitirá resolver y detectar con mayor facilidad los posibles conflictos entre ellos.

Como se ha comentado anteriormente, *Git* ha sido el *Sistema de control de versiones* elegido para el desarrollo de todas las modificaciones realizadas sobre el proyecto original con lo que se pueden observar todos los cambios realizados y revertirlos si se desea.

## 2.2 Apache Maven

La automatización de tareas sobre un proyecto desarrollado usando el lenguaje *Java* se puede realizar con la herramienta llamada *Maven*, desarrollada por 'Apache Software Foundation' y lanzada al mercado por primera vez en 2002.

Como dice Carlos Yagüe en [10], "Apache Maven es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida, incluyendo las fases de compilación y empaquetado, para que pueda ser utilizado por otros desarrolladores y equipos de desarrollo."

Lo primero que hay que realizar para configurar el uso de *Maven* en un proyecto es crear un fichero llamado 'pom.xml', este nombre proviene de las siglas en inglés Project Object Model. Posteriormente en este fichero se indican todos los parámetros necesarios para que *Maven* ayude al desarrollador en la gestión del proyecto que está realizando y gestionar las dependencias existentes en la aplicación.

## 2.3 Apache Tomcat

*Tomcat* es un servidor web, desarrollado por Apache Software Foundation en 1999, implementa las especificaciones de *Servlets* y *JavaServer Pages* (*JSP*). Las especificaciones *Servlets* y *JavaServer Pages* (*JSP*) son desarrolladas por Sun Microsystems y vienen dadas por la JCP (Java Community Process).

*Tomcat* está desarrollado usando el lenguaje de programación *Java*, por lo que puede ser usado en cualquier sistema operativo que tenga instalado su máquina virtual. En la actualidad, *Apache Tomcat* dispone de hasta 9 versiones (*Tomcat* 9.x, soporta hasta la versión de *Servlet* 4.0).

La jerarquía de directorios de instalación de Tomcat incluye:

- bin - arranque, cierre, y otros scripts y ejecutables.
- common - clases comunes que pueden utilizar el contenedor de servlets de *Tomcat*, Catalina, y las aplicaciones web.
- conf - ficheros XML y los correspondientes DTD para la configuración de Tomcat.
- logs - logs de Catalina y de las aplicaciones.
- server - clases utilizadas solamente por Catalina.
- shared - clases compartidas por todas las aplicaciones web.
- webapps - directorio que contiene las aplicaciones web.
- work - almacenamiento temporal de ficheros y directorios.

## 2.4 Pruebas automatizadas en un proyecto software

Las pruebas automatizadas en un proyecto software son de gran importancia, aunque a veces se argumenta que es una pérdida de tiempo para el proyecto, sin embargo, como dice [17], la realización de pruebas automatizadas 'es una de las actividades más importantes y fundamentales en el desarrollo de un proyecto, ya que posibilita los procesos, métodos de trabajo y herramientas necesarias para garantizar la calidad de cualquier desarrollo', las pruebas permiten una mayor iteración sobre el proceso de desarrollo y más ágil, ayuda a otros desarrolladores a entender el funcionamiento del código y además asegura una mayor calidad y resistencia a errores. El código quedará más estructurado y será más fácil su comprensión.

Dentro de una aplicación se pueden hacer pruebas a distintos componentes del sistema, estas se pueden clasificar según los niveles de aplicación que prueben.

- Unidad. Las pruebas de unidad son las encargadas de comprobar el funcionamiento de todos y cada uno de los elementos base que forman la aplicación, por ejemplo, los modelos de una aplicación que haya sido desarrollada usando un patrón Modelo-Vista-Controlador.

- Integración. Se comprueba que las unidades conectadas entre ellas funcionan correctamente y, además, realizan las funciones requeridas.
- Regresión. Se realizan sobre código ya probado anteriormente para verificar que sigue siendo correcto tras haber realizado cambios o añadido código, para ello se pueden usar herramientas de captura/reproducción, como el marco de trabajo *Selenium* [7].
- Humo. Estas pruebas validan funcionalidades generales críticas de la aplicación, si el plan de proyecto está en riesgo se detectará con estas pruebas.
- Validación. Valida el cumplimiento de los requisitos funcionales.
- Sistema. Se comprueba que la aplicación al completo funciona correctamente.

En cada proyecto software se pueden realizar todo tipo de pruebas, pero se debe definir un plan para estas, este plan definirá los tipos de pruebas que serán desarrolladas sobre el proyecto y la cantidad a realizar de cada tipo. Por ejemplo, se podría probar una aplicación solamente con pruebas manuales, es decir, un equipo dedicado a comprobar la aplicación al completo, o solamente con pruebas de unidad, el problema que se encuentra con esta situación es que se comprobaría que cada elemento de la aplicación funciona correctamente de forma independiente, pero no se tendría certeza sobre el funcionamiento conjunto. Por ello surge el concepto de 'pirámide del testing', el cual se explica en la siguiente sección.

### 2.4.1 La pirámide del testing

La pirámide del testing es una figura que representa gráfica y claramente la cantidad de pruebas que se deben hacer a una aplicación. Como se ve a continuación, la mayor cantidad de pruebas deben ser unitarias, menor cantidad de integración y por último menos sobre la interfaz gráfica. ¿Por qué? Como dice [5], con esto se consigue anticiparse a los errores que se puedan producir y tener controlados los elementos base de la aplicación para asegurar que no son propagados hacia arriba, además, los test de unidad documentan la funcionalidad de la aplicación.

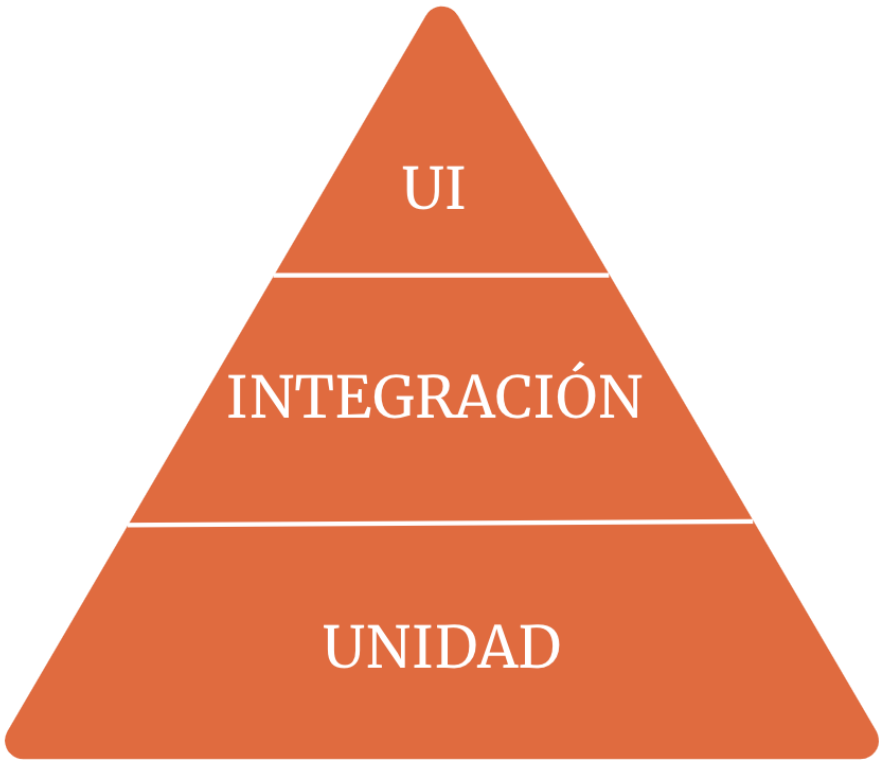


**Figura 2.1** La pirámide del testing.

### 2.4.2 Pruebas en una aplicación *Java*

Para la realización de las pruebas en la aplicación *Java* se ha usado el entorno de trabajo *JUnit*, desarrollado por *Kent Beck*, como dice [13], 'JUnit se trata de un entorno Open Source para la automatización de las pruebas (tanto unitarias, como de integración) en los proyectos Software. Provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad'.

*JUnit* permite realizar pruebas sobre la aplicación, tanto unitarias como de integración y regresión. Para realizar las pruebas se comprueba que, ante cierta entrada sobre el código, la respuesta obtenida es la esperada.

## 2.5 Gitlab

*Gitlab* es un servidor open source para alojar proyectos software, pero no solamente eso, *Gitlab* proporciona una solución, como su propia página web indica [15], para todo el proceso de *DevOps* en una sola aplicación. Su instalación básica ofrece un servicio de alojamiento de repositorios, permite gestionar las tareas mediante un tablero que permite ver las tareas que están esperando a ser desarrolladas, las tareas que están en progreso y las que están finalizadas. *Gitlab* ofrece una solución para ejecutar tareas automatizadas en segundo plano.

Su principal funcionalidad es alojar proyectos software, donde se puede subir un repositorio, ver los commits, cambios realizados sobre el proyecto, etc. En la siguiente imagen se puede observar la página mostrada tras la creación de un proyecto vacío.

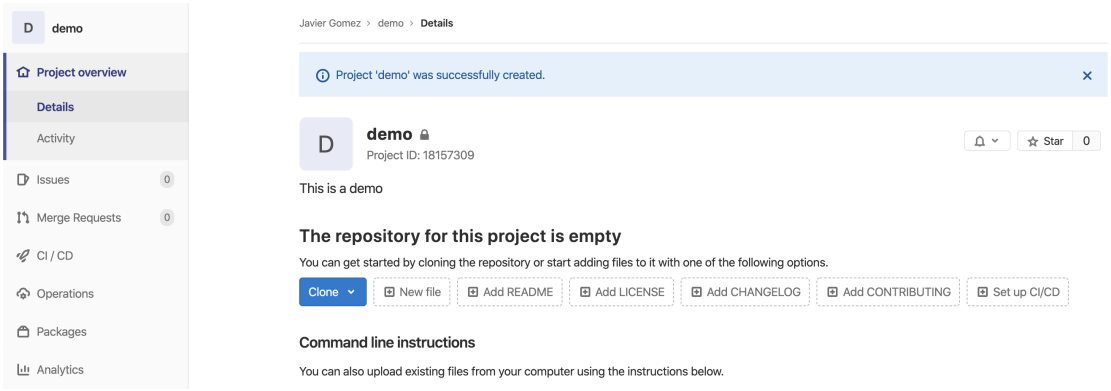


**Figura 2.2** Repositorio vacío en Gitlab.

Como se ha comentado anteriormente, *Gitlab* también tiene la opción de gestionar las tareas existentes en el proyecto y clasificarlas según su estado: esperando a ser realizadas, en progreso, bloqueadas o realizadas. Esta funcionalidad es requerida en la actualidad para poner en práctica muchas de las metodologías de gestión de proyectos ágiles existentes. En la siguiente imagen se puede observar el aspecto de esta funcionalidad.

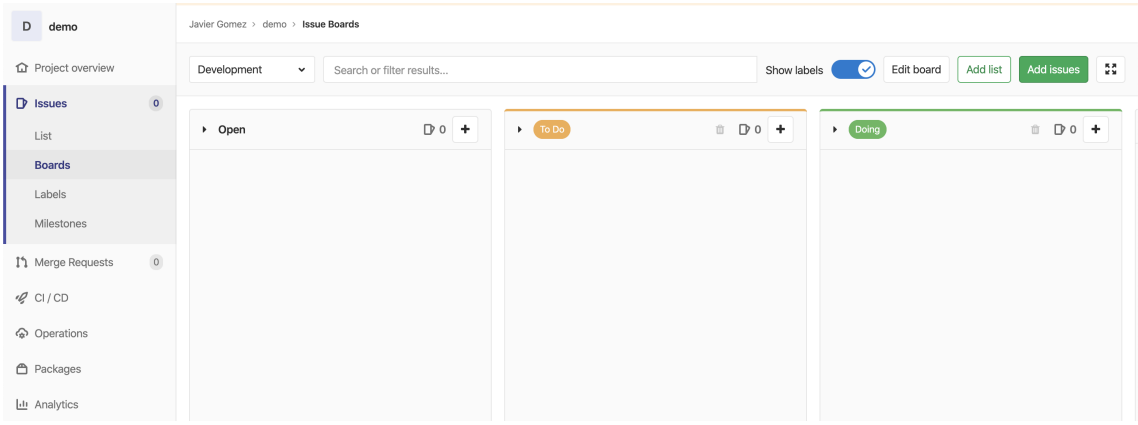


**Figura 2.3** Tablero de gestión de tareas en Gitlab.

Por último, *Gitlab* es capaz de ejecutar tareas automatizadas, configuradas en un archivo llamado 'gitlab-ci.yml', explicado en el capítulo 3.7, para ello *Gitlab* hace uso de otro servicio que ejecute estas tareas en segundo plano, este servicio se llama *Gitlab Runner* o ejecutor. En la siguiente imagen se puede encontrar un ejemplo de ejecución de tareas automatizadas.

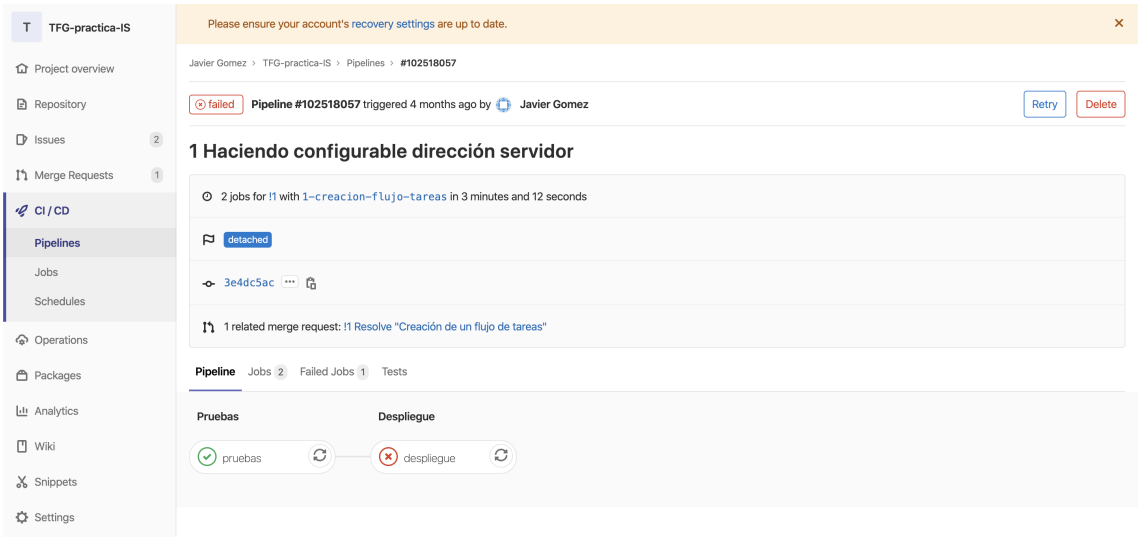


**Figura 2.4** Ejecución de tareas automatizadas en Gitlab.

*Gitlab* se desarrolla usando *Ruby* con el marco de trabajo web *Rails*, esta combinación es conocida como *Ruby on Rails* y fue publicada en 2005. En la siguiente sección se procede a profundizar sobre este lenguaje de programación web.

## 2.6 Ruby On Rails

*Ruby* es un lenguaje interpretado, orientado a objetos, desarrollado por Yukihiro Matsumoto quién comenzó a trabajar en él en 1993, y lo presentó públicamente en 1995. Junto a el entorno de trabajo *Rails*, forman la combinación con la cual se han estado desarrollando muchas de las aplicaciones web en los últimos años.

Es un lenguaje de propósito general, es decir, con Ruby se pueden desarrollar todo tipo de aplicaciones diferentes: aplicaciones de servicio web, clientes de correo electrónico, procesamiento de datos en Backend, aplicaciones de red, etc. Además, *Ruby* es un lenguaje de programación dinámico y flexible, con estrategias como la meta programación (escribir programas que escriben o manipulan otros programas, o a sí mismos).

*Ruby* es un lenguaje de programación de alto nivel, significa que está lejos del lenguaje de máquina y es muy cercano al lenguaje natural.

Las dependencias en *Ruby* se llaman gemas, una gema en Ruby es un conjunto de código empaquetado. Se pueden utilizar las gemas por ejemplo para añadir funcionalidad que ya ha implementado otro desarrollador para así reutilizar código. Estas dependencias pueden ser gestionadas de manera automatizada por la herramienta 'bundler'.

*Ruby on Rails* fue la combinación elegida para el desarrollo de la aplicación *Gitlab*, su código se puede encontrar en '/opt/gitlab/embedded/service/gitlab-rails'.

Cabe destacar, por su interés en la realización de este proyecto, la existencia de 'scripts' ejecutados en *Ruby on Rails*. Estos scripts son llamados *Rake Tasks* y permiten realizar modificaciones sobre datos existentes en la aplicación.

## 2.7 Metodologías ágiles

Antes de introducir las metodologías ágiles, se va a explicar el modelo de proceso que más popularidad ha obtenido en el sector.

Los proyectos, como todo en la vida, necesitan ser gestionados para su correcta realización, ya que sin una correcta organización es fácil que una tarea en la se necesite la colaboración de más de dos desarrolladores se vuelva misión imposible. Para ello, en todo proyecto de ingeniería se lleva a cabo una organización y gestión del proyecto.

Esta organización puede variar dependiendo del ámbito del proyecto y de las necesidades de este. Concretamente, en la industria software se pueden definir unas fases que deben estar en todo proceso, estas fases son:

1. Recogida de requisitos. En esta fase el equipo debe hablar con el cliente, entender lo que quiere y especificar los requisitos que el proyecto deberá cumplir.
2. Planificación y especificación de las tareas a realizar. Se deben planificar tareas para los desarrolladores, estas tareas deben conducir a la realización de software que cumpla todos los requisitos especificados. Estas tareas deben de ser desarrolladas en los plazos planificados.
3. Diseño. Los técnicos deberán realizar un análisis de los requisitos y ofrecer la mejor solución desde un punto de vista técnico que satisfaga las necesidades del cliente.
4. Desarrollo. Los desarrolladores comienzan a trabajar en la solución propuesta y deben cumplir los objetivos en el plazo indicado.
5. Validación. Se comprueba que el proyecto funciona correctamente y, además, cumple con lo recogido en los requisitos.
6. Despliegue. El proyecto es desplegado en producción. Dando acceso al cliente y a los usuarios finales a la aplicación.
7. Evolución y mantenimiento. Una vez desplegado se debe dar soporte al producto y si fuese necesario desarrollar funcionalidades adicionales.

En esta sección se va a explicar qué son las metodologías ágiles, así como el por qué de su popularidad en la actualidad. Empezando desde el inicio en el que se parte de modelos de procesos heredados de otras ingenierías en los que no es posible la iteración sobre el proyecto, o al menos no merece la pena esta iteración. La metodología tradicional más conocida es el modelo en cascada, también conocido en inglés como *waterfall*.

Como dice [6], el modelo en cascada fue el primer modelo utilizado en la industria del software, es un modelo secuencial, cada fase es llevada a cabo de forma única, es decir, cuando una fase termina se comienza la siguiente, teniendo un producto al final que será el entregado.

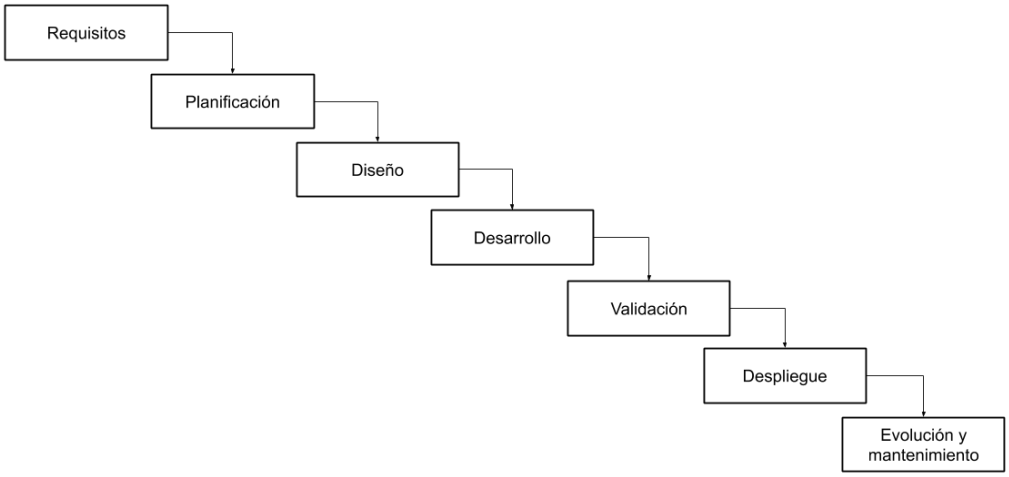


**Figura 2.5** Modelo cascada.

Este modelo ha variado a lo largo del tiempo y puede ser complementado con bucles de retroalimentación. Todavía se usa hoy en día si los requisitos y características del proyecto están claramente definidos durante la fase de *recogida de requisitos y creación de las tareas a realizar*. A partir de este han surgido otros modelos de gestión de proyectos, en los que se realizan distintos flujos sobre las fases de gestión de proyectos, tienen una mayor iteración sobre una de las fases, etc...

En este proyecto se tratan herramientas que facilitan el desarrollo de una metodología ágil, llegados a este punto, ¿qué es una metodología ágil? Como dice [16], 'las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno'.

Las metodologías ágiles se basan en una mayor iteración sobre el ciclo de entrega de un proyecto, realizando entregas de forma periódica al cliente, obteniendo así retroalimentación en el proyecto, esto permite que las prioridades del proyecto cambien y el resultado final sea el deseado por el cliente. Las metodologías ágiles más conocidas son SCRUM, Programación extrema (XP), Kanban, etc. El proceso de desarrollo del proyecto es iterativo y quedaría como se puede ver en la siguiente imagen.

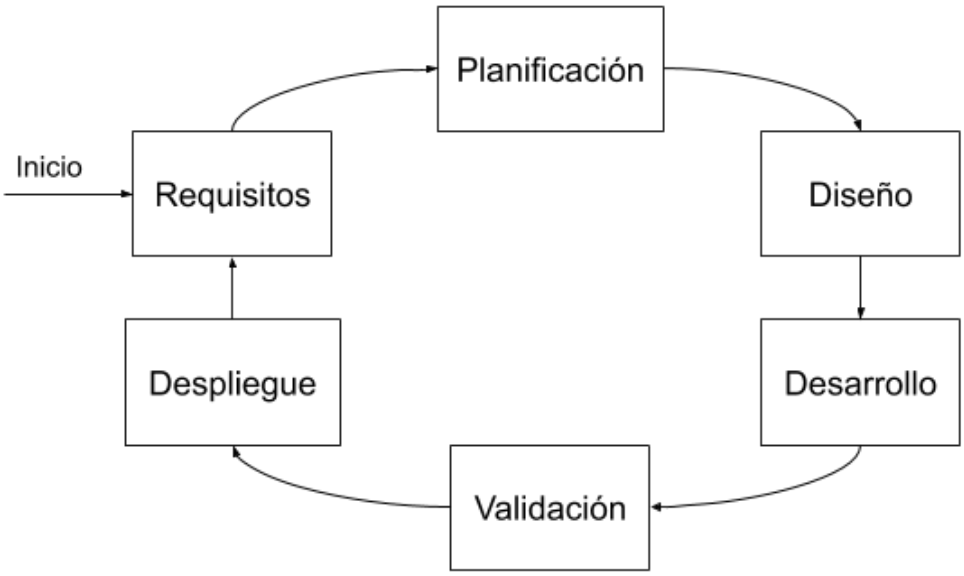


**Figura 2.6** Modelo Agile.

Hace diecinueve años se reunieron diecisiete expertos recogiendo en un manifiesto unos principios y valores que se debían seguir en toda metodología ágil para ser llevada a cabo de forma exitosa, diecinueve años después se sigue usando como referencia. Este manifiesto es explicado en el siguiente apartado.

### 2.7.1 Principios del Manifiesto Ágil

El 12 de febrero de 2001 se reunieron diecisiete expertos, convocados por *Kent Beck*, para discutir sobre el desarrollo software. En la reunión se definió el término 'Métodos ágiles', para ello definieron cuatro valores y doce principios incluidos en el Manifiesto Ágil que todo método ágil de cumplir.

Los valores recogidos en el Manifiesto Ágil [8] son:

1. **Individuos e interacciones sobre procesos y herramientas.** Las personas son el principal factor de éxito de un proyecto software. Esto quiere decir, se debe prestar mayor atención a los trabajadores, los integrantes del equipo.
2. **Software funcionando sobre documentación extensiva.** La documentación no debe realizarse a menos que sea necesaria. Estos documentos deben ser cortos y precisos.
3. **Colaboración con el cliente sobre negociación contractual.** Siempre se debe escuchar al cliente y eso es más importante que los requisitos recogidos en el contrato inicial, debe existir una relación continua entre el equipo y el cliente.
4. **Respuesta ante el cambio sobre seguir un plan.** El equipo debe ser capaz de afrontar cambios que puedan surgir en el proyecto.

Acompañados de los siguientes principios [9]:

1. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Se acepta que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Se entrega software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software que funciona es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo, para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

### 2.7.2 SCRUM

Como se define en [18], '*SCRUM* define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicado para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas Sprint. El resultado de cada Sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.'

Para la realización de *SCRUM*, se mantiene en el proyecto una pila de tareas ordenadas por prioridad en la que el equipo de desarrollo pueda ver las tareas en las que debería trabajar posteriormente, esta pila se conoce como *'Backlog'*.

La principal diferencia de *SCRUM* sobre las demás metodologías ágiles es la realización de *'Sprints'*, un *Sprint* es un periodo de tiempo definido en el que el equipo debe llevar a cabo una iteración del ciclo de vida de desarrollo, es decir, en él se deben planificar las tareas con mayor prioridad, diseñar la solución que se va a realizar, desarrollar los nuevos cambios, validarlos y desplegar los cambios. La duración de un *Sprint* puede variar desde 1 semana hasta un par de meses, una vez que se decida la duración esta debe ser constante durante el ciclo de vida del proyecto.

Dentro de los *Sprints* se realizan reuniones de corta duración en las que hay que analizar las siguientes tareas que el equipo deberá coger del *Backlog*, estimarlas y prepararlas. Otra reunión deberá cerrar el *Sprint* actual y empezar el nuevo con las nuevas tareas. Además, cabe destacar la existencia de una reunión diaria, cuya duración debe ser aproximadamente de 15 minutos, en la cual el desarrollador debe responder a tres preguntas:

1. ¿Qué hizo desde la última reunión?
2. ¿Ha encontrado algún problema en el desarrollo de la tarea?
3. ¿Qué va a hacer hasta la próxima reunión diaria?

### 2.7.3 Informe CHAOS

El 'informe CHAOS' es un informe realizado por 'The Standish Group' de forma periódica en el que analiza la cantidad de proyectos llevados a cabo de forma satisfactoria frente a los que han tenido algún problema o han fracasado, analizando posteriormente los factores que han influido en esto. Resulta interesante el informe realizado en 2015, en el cual analiza el resultado de los proyectos realizados con una metodología ágil frente a proyectos realizados con un modelo en cascada, en él se puede observar que casi el doble de proyectos realizados siguiendo una metodología ágil se realiza de forma satisfactoria.

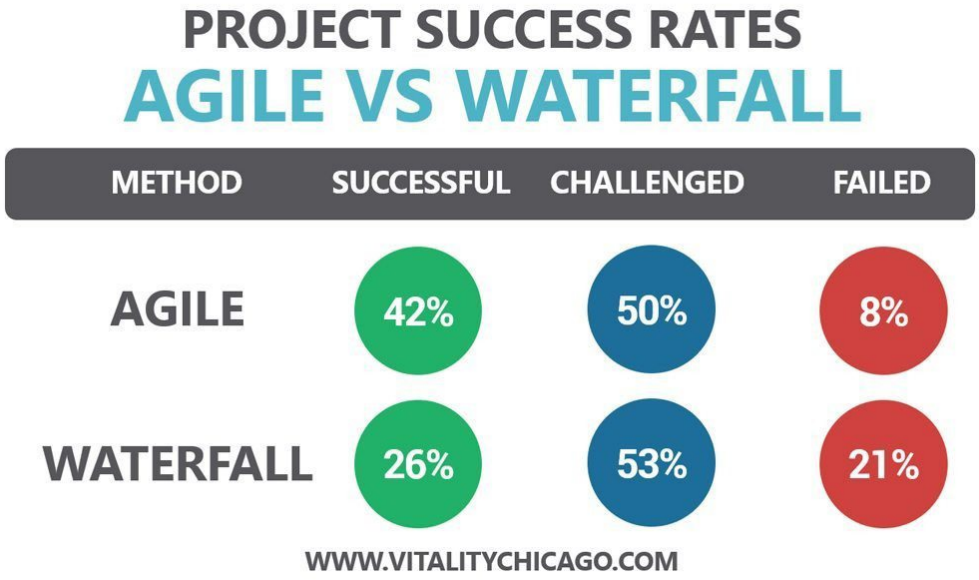


**Figura 2.7** Informe CHAOS.

### 2.7.4 Integración y entrega continua

Integración y entrega continua son procedimientos para darle agilidad al desarrollo, por lo que cada vez son más usados en metodologías ágiles.

Como dice [4], con la Integración Continua (CI) los desarrolladores añaden sus modificaciones frecuentemente a la rama principal de un repositorio común. En lugar de desarrollar partes del código de manera independiente e integrarlas al final del ciclo de desarrollo.

La idea es reducir los problemas encontrados a la hora de integrar los cambios, para ello se hace que los desarrolladores realicen integraciones con mayor frecuencia. En la práctica, un desarrollador a menudo descubrirá conflictos entre el código nuevo y el existente en el momento de la integración. Si esta se realiza con frecuencia, la expectativa es que la resolución de conflictos será más fácil.

La Entrega Continua (CD) consiste en la automatización del proceso de despliegue de software para permitir integraciones sencillas en el entorno de producción. Con la Entrega Continua, los lanzamientos de nuevos cambios ocurren de manera frecuente y rutinaria. Los equipos continúan con las tareas diarias de desarrollo con la confianza de que pueden desplegar a producción un lanzamiento de calidad, en cualquier momento que deseen, de forma sencilla.

Debido a la alta frecuencia de entregas se complica la realización de pruebas manuales sobre la aplicación en cada despliegue, por lo que se necesita tener pruebas automatizadas que ofrezcan al equipo confianza de que no se están creando errores con las nuevas modificaciones.

El proceso consiste en una compleja automatización de los procesos de despliegue, pruebas y actualización de servidores de producción.

## 2.8 Docker

*Docker* es un proyecto open source, desarrollado con el lenguaje de programación *Go*. *Docker* es un software que permite la automatización de aplicaciones dentro de contenedores software. Desarrollado por Solomon Hykes, el desarrollo de *Docker* comenzó como un proyecto interno dentro de la empresa en la que trabajaba, 'dotCloud', el proyecto fue liberado cómo código abierto en marzo de 2013.

Antes de seguir con *Docker* se debe explicar qué es un contenedor software. Los contenedores software, explicado de forma breve, son entornos ligeros preparados para proporcionar a las aplicaciones los archivos, las variables, las bibliotecas y la configuración necesaria para ejecutarse, maximizando de esta forma su portabilidad.

Como dice el artículo [2], la tecnología Docker usa el kernel de Linux y las funciones de este, como 'Cgroups' y 'namespaces', de modo que los contenedores puedan ejecutarse de manera independiente. El propósito de los contenedores es esta independencia: la capacidad de ejecutar varios procesos y aplicaciones por separado para hacer un mejor uso de su infraestructura y, al mismo tiempo, conservar la seguridad que tendría con sistemas separados.

*Docker* ofrece un modelo de implementación de contenedores basado en imágenes. Esto permite compartir una aplicación, o un conjunto de servicios, con todas sus dependencias en varios entornos. Además, automatiza la implementación de la aplicación (o conjuntos combinados de procesos que constituyen una aplicación) en este entorno de contenedores. *Docker* otorga a los usuarios un acceso sin precedentes a las aplicaciones, ofrece la capacidad de implementar rápidamente y controlar las versiones y su distribución.

# 3 Desarrollo del trabajo

*Y si este es el final, le encontraremos la belleza.*
*Quizás en realidad, ahora es cuando todo empieza.*

RESIDENTE

Para la práctica propuesta se ha configurado un entorno que utiliza herramientas como *Gitlab* y permite la ejecución de tareas automatizadas relacionadas con proyectos web desarrollados usando la tecnología *Servlet*, este proyecto ha sido integrado con *Apache Maven*, *Git* y pruebas automatizadas usando *JUnit*. Permite llevar a cabo los conceptos de integración y entrega continua que facilitan la gestión del proyecto usando un modelo de gestión de proyecto ágil. Este entorno será utilizado por los alumnos de la asignatura 'Ingeniería de Software' del grado para realizar una práctica que les permitirá conocer las herramientas usadas en un desarrollo ágil y como son usadas, conociendo así las ventajas de un desarrollo ágil con un enfoque práctico.

## 3.1 Escenario propuesto

Para la realización de la práctica se ha preparado un escenario donde los alumnos puedan probar un entorno preparado para llevar a cabo un desarrollo ágil. Se trabajará sobre una aplicación web desarrollada con objetivo de ser usada en docencia. En este apartado se va a explicar cómo quedaría el escenario propuesto y como funciona todo. Para ello, se puede encontrar a continuación un diagrama de los distintos componentes que forman el escenario.

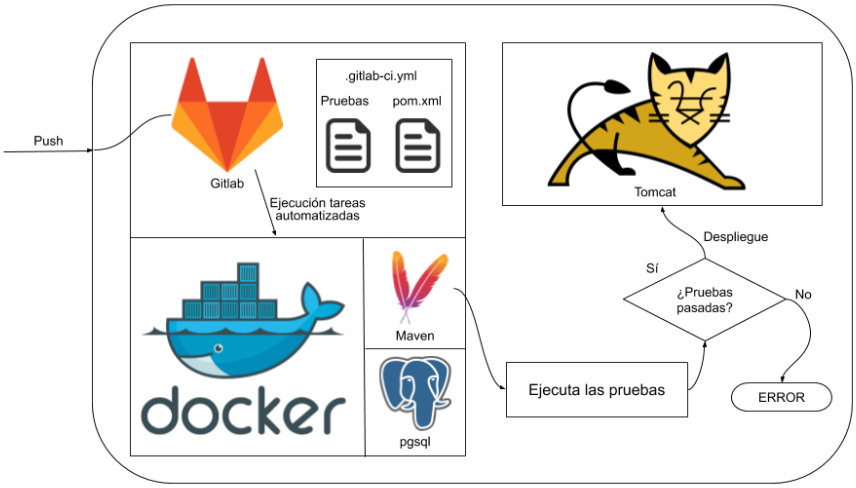


**Figura 3.1** Escenario propuesto para la práctica.

Todo el proceso comienza en *Gitlab*, este es el servicio con el que el desarrollador debe interactuar, por ejemplo, para clonar o actualizar los repositorios deseados. Cuando tras subir una versión del proyecto *Gitlab* detecta en la misma un fichero '.gitlab-ci.yml' (en 3.17 se encuentra el fichero '.gitlab-ci.yml' desarrollado para este proyecto), lo lee y ejecutará mediante el ejecutor compartido las tareas automatizadas descritas en él. En el caso del escenario que se propone estas tareas se ejecutarán sobre una imagen Docker, con todas las herramientas necesarias para las pruebas automatizadas, cuando se actualice o cree una petición de integración en el proyecto, y se comprobará si estas han sido realizadas de forma satisfactoria. Si ninguno de los comandos ejecutados en la tarea automatizada falla, la versión desarrollada de la aplicación será desplegada en un entorno de pruebas en el servidor *Tomcat* y si falla, el usuario podrá ver el resultado de los comandos ejecutados.

## 3.2 Práctica propuesta

En esta sección se procede a explicar la práctica desarrollada para que el alumno alcance los objetivos de aprendizaje propuestos. Para esta práctica los alumnos trabajarán en pareja. Cada miembro de la pareja tomará un rol distinto (desarrollador o líder de equipo) dentro del proceso de desarrollo de una nueva tarea. A cada pareja se le proporcionará un nuevo requisito solicitado por un cliente ficticio y deberán incorporar la funcionalidad requerida al código.

Los roles que se deben asignar en cada pareja:

- **Desarrollador.** Será la persona encargada de asignarse una nueva tarea, desarrollarla (asegurándose de que todo sigue funcionando correctamente) e integrar los cambios.
- **Líder de equipo + desarrollador.** Este rol es el encargado de crear las tareas necesarias teniendo en cuenta las tareas que hay que realizar y las prioridades. Será el encargado de revisar el código del compañero en la petición de integración, asegurándose de que los cambios siguen los estándares definidos por el equipo y de que no se va a introducir ningún error en la aplicación.

1. Cada pareja tendrá dos usuarios distintos, correspondientes a cada rol existente en el equipo, deberán decidir el rol que será llevado a cabo por cada miembro de la pareja durante toda la duración de la práctica. Los usuarios son DesarrolladorX_Y y LiderEquipoX_Y, donde X será el grupo de prácticas al que pertenezca la pareja e Y el número de la pareja dentro de ese grupo.
2. Se solicita que los alumnos inicien sesión en *Gitlab* con sus respectivos usuarios y prueben la interfaz, con el objetivo de familiarizarse con ella.
3. El miembro de la pareja que desarrolle el rol de líder de equipo deberá dirigirse en el servidor *Gitlab* a la sección 'Boards' y crear un nuevo ticket, que representa la tarea referida al requisito recibido en el enunciado de la práctica perteneciente distinto para cada pareja, un posible cambio podría ser eliminar el constructor vacío de la entidad Libro.
4. El desarrollador deberá dirigirse a la sección 'Boards' como en el apartado anterior, deberá poder ver el ticket creado por el líder de equipo, deberá asignarse el ticket a sí mismo y dentro de este crear una nueva petición de integración y una rama que será en la que trabajará.
5. Una vez creada esta rama el miembro de la pareja que esté realizando el rol de desarrollador, debe llevar a cabo el cambio solicitado en la rama creada, una vez haya realizado el cambio debe actualizar la rama en el servidor *Gitlab*, ¿cuál es el resultado de las pruebas? La prueba correspondiente a la parte del código que haya modificado deberá fallar debido a que no se han actualizado las pruebas correspondientes a ese cambio.
6. El alumno debe actualizar la prueba que ha resultado errónea, una vez solucionada la prueba deberá actualizar su rama alojada en el servidor *Gitlab*, ¿cuál es el resultado de las pruebas ahora? El miembro que lleva a cabo el rol de líder de equipo debe revisar la petición de integración y aprobar su integración.
7. Debido a que las pruebas han sido ejecutadas correctamente, se podrá encontrar los cambios que acaban de ser desarrollados en el servidor *Tomcat*, en el puerto 4000 de la dirección donde se encuentra el servidor, cada alumno tendrá su versión desplegada en la ruta con su nombre de usuario, es decir, el alumno 'alumnoY' podrá encontrar sus cambios en 'http://URLSERVIDOR:4000/alumnoY'.
8. El líder de equipo deberá revisar los cambios realizados en esta rama, aprobando los cambios si considera que los cambios son correctos, o solicitando algún cambio.

9. Si la petición de integración ha sido aprobada, el desarrollador deberá integrar estos cambios sobre la rama principal, la rama 'master'. A continuación, debe actualizar el estado del ticket en el que se ha trabajado indicando que esa funcionalidad ha sido desarrollada.

Para poder llevar a cabo la práctica en el aula, se ha tenido en cuenta que hay 3 grupos, en los cuales el número de alumnos es 20, por lo que se ofrecerán 10 requisitos diferentes para que cada pareja haga cambios distintos sobre el mismo repositorio, enfrentándose a la realidad de un desarrollo colaborativo. Un ejemplo de requisito podría ser: 'Por motivos de seguridad se solicita que la entidad Cliente no pueda ser creada vacía'.

## 3.3 Punto de partida. Proyecto inicial

La versión de partida es un proyecto, desarrollado como Trabajo Fin de Grado en [19]. Este proyecto, como se ha comentado con anterioridad, se realizó dando especial importancia a buenas prácticas y patrones de diseño interesantes para la docencia de desarrollo software, con el uso de esta aplicación los alumnos podrán:

- Conocer varios patrones de diseño y arquitectura software como el Cliente-Servidor o el Modelo-Vista-Controlador.
- Analizar el ciclo de vida de una aplicación.
- Aprender acerca de la gestión de datos y manejar un gestor de base de datos concreto.
- Perfeccionar el manejo de varios lenguajes de programación, tanto del lado del cliente como del servidor.
- Ser capaces de depurar código para detectar errores u optimizarlo.

Este proyecto consiste en una aplicación Web, desarrollada usando la tecnología *Servlet* de Java. La aplicación consiste en una aplicación para la gestión de los libros en una biblioteca, donde los usuarios puedan reservar los libros existentes en ella.

A continuación, se encuentra un diagrama de los casos de uso desarrollados, con los distintos actores (Cliente y Administrador, heredados de la clase abstracta Usuario).

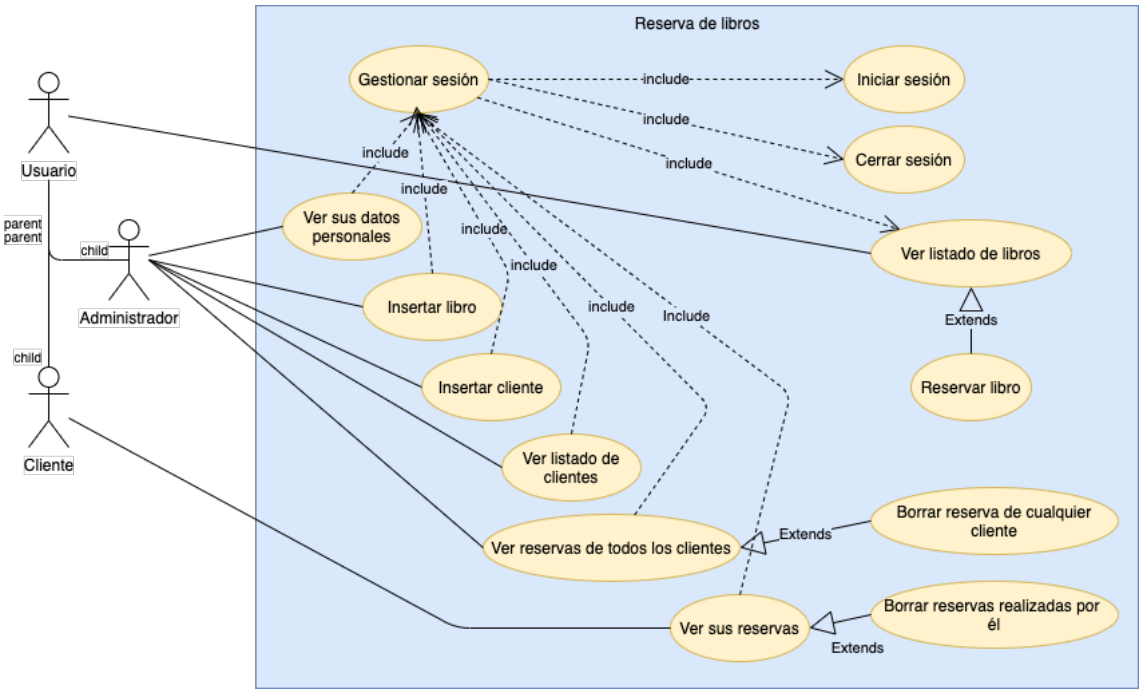


**Figura 3.2** Casos de Uso de la aplicación.

### 3.3.1 Modelos de datos

La información persistente se almacena en una base de datos relacional gestionada con *PostgreSQL*. La representación de los datos en la base de datos relacional está formada por cuatro tablas: 'clientes', 'libros', 'generos' y 'reservas', a continuación, se profundiza más en cada una de ellas.

- **clientes.** Representación de los usuarios que acceden a la aplicación. Los campos contenidos en la tabla *clientes* son:
  - cliente. Esta columna de la tabla almacena el nombre de usuario y será usado por el cliente para acceder a la aplicación. Es una cadena de caracteres y sus valores deben ser únicos en toda la tabla.
  - nombre. Dedicado al nombre real del usuario al que hace referencia este modelo. Es una cadena de caracteres.
  - apellido1. Dedicado al primer apellido del usuario al que hace referencia este modelo. Es una cadena de caracteres.
  - apellido2. Dedicado al segundo apellido del usuario al que hace referencia este modelo. Es una cadena de caracteres.
  - dni. Esta columna de la tabla almacena el DNI del usuario, este campo será usado como clave primaria para identificar al cliente dentro de la tabla, como definición de clave primaria, su valor debe ser único en toda la tabla. Esta columna debe ser el DNI sin letra, es un número entero.
  - direccion. Dedicado a la dirección del cliente. Es una cadena de caracteres.
  - admin. Este campo indica si el cliente tiene capacidades de administrador, este campo es de tipo bit, tomando el valor 1 (verdadero) cuando el usuario puede realizar acciones de administrador y 0 (falso) en caso contrario.
  - contrasena. Este campo es la contraseña del usuario, junto al atributo cliente, forman las credenciales de autenticación de los usuarios en la aplicación. Es una cadena de caracteres.
- **libros.** Esta tabla representa los libros existentes en la biblioteca gestionada por la aplicación. Las columnas contenidas en esta tabla son:
  - titulo. Esta columna de la tabla almacena el título de un libro. Es una cadena de caracteres.
  - isbn. Este campo representa el ISBN de los libros existentes en la aplicación. Es la clave primaria de la tabla *libros*, y su valor es un número.
  - autor. Dedicado al nombre del escritor del libro. Es una cadena de caracteres.
  - editorial. Dedicado al nombre de la editorial del libro. Es una cadena de caracteres.
  - anio_pub. Esta columna representa el año de publicación del libro. Su valor es un número.
  - genero. Este campo es el género del libro, es una clave externa hacía la tabla *generos*, en la cual este valor es su clave primaria, es una cadena de caracteres.
  - observaciones. Dedicado a las posibles observaciones sobre el libro en cuestión. Es una cadena de caracteres.
- **generos.** Representa los distintos géneros de libros existentes en la biblioteca. El único campo que almacena esta tabla es 'genero', una cadena de caracteres que almacena el nombre del genero de los libros. Esta columna actúa como clave primaria de esta tabla.
- **reservas.** Esta tabla representa las reservas de libros realizadas por un cliente en la biblioteca. Esta entidad es una relación de las entidades *clientes* y *libros*, por lo que solo contiene dos columnas:
  - isbn. Esta columna es la clave primaria de la tabla *libros*, en esta tabla actúa como clave externa para hacer referencia al libro reservado.
  - dni. Como en el caso anterior, esta columna es la clave primaria de la tabla *clientes*, actuando como clave externa para hacer referencia al cliente que ha reservado el libro.

A continuación, se muestra un diagrama entidad-relación que ayudará a entender los distintos modelos de datos y su representación, las distintas entidades que se puede encontrar en la aplicación y sus relaciones.

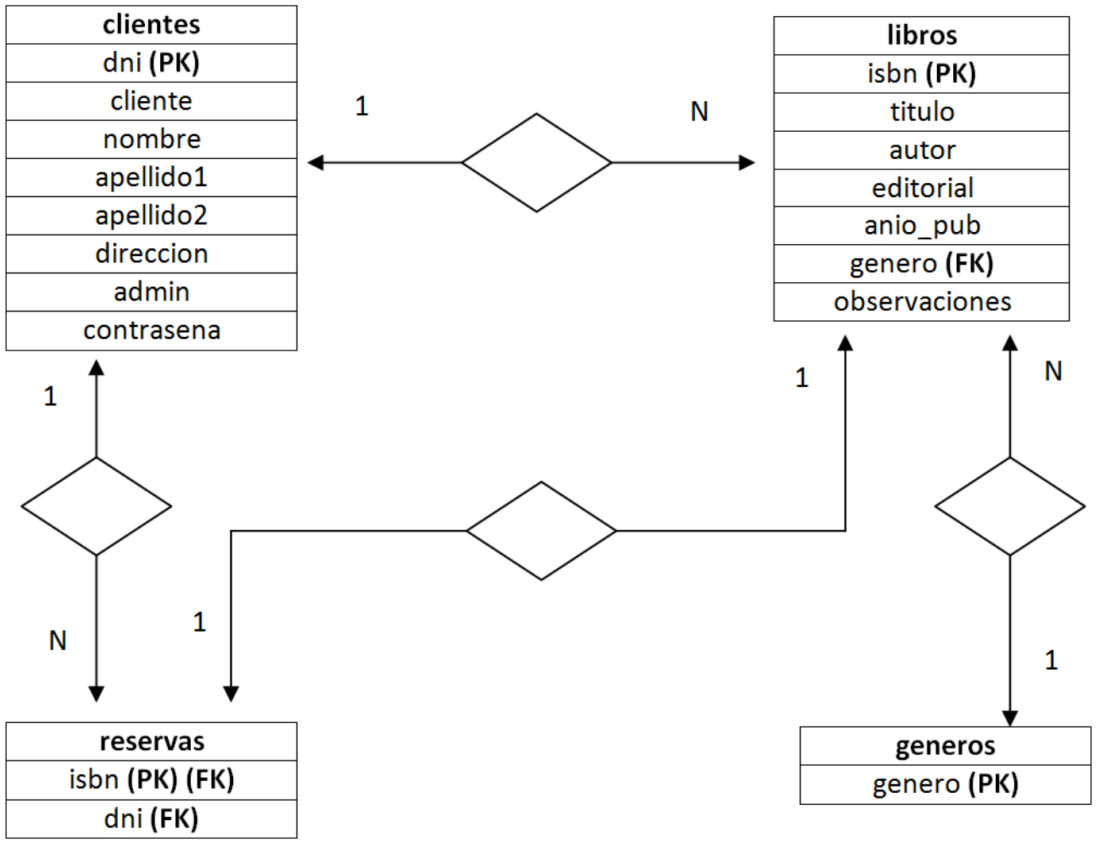


**Figura 3.3** Modelo Entidad-Relación de la aplicación.

Este sería el punto de partida, se puede encontrar más información en [19].

## 3.4 Realización de pruebas sobre la aplicación

Para la aplicación de partida no se habían codificado pruebas automatizadas. Sin embargo, para seguir un modelo de proceso ágil es imprescindible que estas pruebas existan y sean de calidad, ya que no sólo se utilizan para verificar la primera versión del software, sino que formarán parte de las pruebas de regresión, cuando evolucione el software, y será necesario ejecutarlas en numerosas ocasiones, y posiblemente modificarlas y extenderlas.

Por ello se han codificado pruebas unitarias para los modelos existentes en la aplicación, desarrollados en *Java* por lo que se ha usado el marco de trabajo *JUnit* para la realización de estas.
A continuación, se procede a explicar las distintas pruebas.

### 3.4.1 Pruebas unitarias realizadas a la entidad Cliente

Se codifican pruebas unitarias a todos los constructores de la entidad Cliente, para comprobar que funcionan correctamente y tienen los valores correctos cuando son creados.

- Se comprueba que el constructor vacío (sin parámetros) funciona correctamente, para ello se crea una instancia de la clase Cliente y si esta falla, es decir, lanza una excepción al crear una instancia de la clase Cliente, la prueba fallará.

**Código 3.1** Prueba unitaria sobre el primer constructor de la entidad Cliente.

```
  @Test
  public final void testConstructor1ClienteOK() {
    System.out.println("Comienza la prueba que comprueba que el
        constructor 1 de la entidad Cliente funciona bien");

    Cliente cliente = new Cliente();
  }
```

- El constructor que acepta un solo parámetro, el dni, instancia una clase de la entidad Cliente y se comprueba que el parámetro dni se asigna correctamente.

**Código 3.2** Prueba unitaria sobre el segundo constructor de la entidad Cliente.

```
  @Test
  public final void testConstructor2ClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el constructor
      2 de la entidad Cliente funciona bien");

  Integer dni = 111111;

  Cliente cliente;

  cliente = new Cliente(dni);

  assertEquals("El dni debería haberse asignado correctamente", cliente.
      getDni(), dni);
}
```

- La siguiente prueba comprueba que el constructor que acepta el dni, cliente (nombre de usuario), nombre, primer apellido, segundo apellido, dirección, un valor que indica si el cliente es administrador y la contraseña funciona correctamente y los parámetros son correctamente asignados.

**Código 3.3** Prueba unitaria sobre el tercer constructor de la entidad Cliente.

```
  @Test
  public final void testConstructor3ClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el constructor
      3 de la entidad Cliente funciona bien");

  Integer dni = 111111;
  String usuario = "Usuario1";
  String nombre = "Nombre";
  String apellido1 = "Primer apellido";
  String apellido2 = "Segundo apellido";
  String direccion = "Dirección del cliente";
  Boolean admin = true;
  String contrasena = "contraseña";

  Cliente cliente;
```

```
  cliente = new Cliente(dni, usuario, nombre, apellido1, apellido2,
      direccion, admin, contrasena);

  assertEquals("El dni debería haberse asignado correctamente", cliente.
      getDni(), dni);
  assertEquals("El cliente debería haberse asignado correctamente",
      cliente.getCliente(), usuario);
  assertEquals("El nombre debería haberse asignado correctamente", cliente.
      getNombre(), nombre);
  assertEquals("El apellido1 debería haberse asignado correctamente",
      cliente.getApellido1(), apellido1);
  assertEquals("El apellido2 debería haberse asignado correctamente",
      cliente.getApellido2(), apellido2);
  assertEquals("La direccion debería haberse asignado correctamente",
      cliente.getDireccion(), direccion);
  assertEquals("El valor admin debería haberse asignado correctamente",
      cliente.getAdmin(), admin);
  assertEquals("La contrasena debería haberse asignado correctamente",
      cliente.getContrasena(), contrasena);
}
```

- La siguiente prueba comprueba que el constructor con los argumentos dni, nombre, primer apellido, segundo apellido, dirección y admin, un valor que indica si el cliente es administrador, funciona correctamente y los parámetros son correctamente asignados.

**Código 3.4** Prueba unitaria sobre el cuarto constructor de la entidad Cliente.

```
  @Test
  public final void testConstructor4ClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el constructor
      4 de la entidad Cliente funciona bien");

  Integer dni = 111111;
  String nombre = "Nombre";
  String apellido1 = "Primer apellido";
  String apellido2 = "Segundo apellido";
  String direccion = "Dirección del cliente";
  Boolean admin = true;

  Cliente cliente;

  cliente = new Cliente(dni, nombre, apellido1, apellido2, direccion,
      admin);

  assertEquals("El dni debería haberse asignado correctamente", cliente.
      getDni(), dni);
  assertEquals("El nombre debería haberse asignado correctamente", cliente.
      getNombre(), nombre);
  assertEquals("El apellido1 debería haberse asignado correctamente",
      cliente.getApellido1(), apellido1);
  assertEquals("El apellido2 debería haberse asignado correctamente",
      cliente.getApellido2(), apellido2);
  assertEquals("La direccion debería haberse asignado correctamente",
      cliente.getDireccion(), direccion);
```

```
  assertEquals("El valor admin debería haberse asignado correctamente",
      cliente.getAdmin(), admin);
}
```

### 3.4.2 Pruebas unitarias realizadas a la entidad Libro

Se codifican pruebas unitarias a todos los constructores de la entidad Libro, para comprobar que funcionan correctamente y tienen los valores correctos cuando son creados.

- Se comprueba que el constructor vacío (sin parámetros) funciona correctamente, para ello se crea una instancia de la clase Libro y si esta falla, es decir, lanza una excepción al crear una instancia de la clase Libro, la prueba fallará.

**Código 3.5** Prueba unitaria sobre el primer constructor de la entidad Libro.

```
    @Test
    public final void testConstructor1LibroOK() {
      System.out.println("Comienza la prueba que comprueba que el
          constructor 1 funciona bien");

        new Libro();
    }
```

- El constructor que acepta un solo parámetro, el isbn, instancia una clase de la entidad Libro y se comprueba que el parámetro isbn se asigna correctamente.

**Código 3.6** Prueba unitaria sobre el segundo constructor de la entidad Libro.

```
    @Test
    public final void testConstructor2LibroOK() {
      System.out.println("Comienza la prueba que comprueba que el
          constructor 2 funciona bien");

      Long isbn = 123456L;
      Libro libro;

      libro = new Libro(isbn);

      assertEquals("El ISBN debería haberse asignado correctamente", libro.
          getIsbn(), isbn);
    }
```

- La siguiente prueba comprueba que el constructor que acepta el isbn, titulo, autor, editorial, anio_pub, genero y observaciones funciona correctamente y los parámetros son correctamente asignados.

**Código 3.7** Prueba unitaria sobre el tercer constructor de la entidad Libro.

```
    @Test
    public final void testConstructor3LibroOK() {
      System.out.println("Comienza la prueba que comprueba que el
          constructor 3 funciona bien");
```

```
    Long isbn = 123456L;
    String titulo = "Don Quijote de la Mancha";
    String autor = "Miguel de Cervantes Saavedra";
    String editorial = "Francisco de Robles";
    Integer anio_pub = 1605;
    String genero = "Novela de aventuras";
    String observaciones = null;

    Libro libro;

      libro = new Libro(isbn, titulo, autor, editorial, anio_pub, genero,
          observaciones);

    assertEquals("El ISBN debería haberse asignado correctamente", libro.
        getIsbn(), isbn);
    assertEquals("El titulo debería haberse asignado correctamente",
        libro.getTitulo(), titulo);
    assertEquals("El autor debería haberse asignado correctamente",
        libro.getAutor(), autor);
    assertEquals("La editorial debería haberse asignado correctamente",
        libro.getEditorial(), editorial);
    assertEquals("El anio de publicación debería haberse asignado
        correctamente", libro.getAnio_pub(), anio_pub);
    assertEquals("El genero debería haberse asignado correctamente",
        libro.getGenero(), genero);
    assertEquals("Las observaciones deben haberse asignado correctamente
        ", libro.getObservaciones(), observaciones);
  }
```

- La siguiente prueba comprueba que el constructor con los argumentos isbn, titulo, autor, editorial, anio_pub, genero, observaciones y el dni del cliente que ha realizado una reserva de ese libro, funciona correctamente y los parámetros son correctamente asignados.

**Código 3.8** Prueba unitaria sobre el cuarto constructor de la entidad Libro.

```
  @Test
  public final void testConstructor4LibroOK() {
    System.out.println("Comienza la prueba que comprueba que el
        constructor 4 funciona bien");

    Long isbn = 123456L;
    String titulo = \"Don Quijote de la Mancha";
    String autor = \"Miguel de Cervantes Saavedra";
    String editorial = \"Francisco de Robles";
    Integer anio\_pub = 1605;
    String genero = \"Novela de aventuras";
    String observaciones = null;
    Integer dni = 111111;

    Libro libro;

      libro = new Libro(isbn, titulo, autor, editorial, anio_pub, genero,
          observaciones, dni);
```

```
    assertEquals("El ISBN debería haberse asignado correctamente", libro.
        getIsbn(), isbn);
    assertEquals("El titulo debería haberse asignado correctamente",
        libro.getTitulo(), titulo);
    assertEquals("El autor debería haberse asignado correctamente",
        libro.getAutor(), autor);
    assertEquals("La editorial debería haberse asignado correctamente",
        libro.getEditorial(), editorial);
    assertEquals("El anio de publicación debería haberse asignado
        correctamente", libro.getAnio_pub(), anio_pub);
    assertEquals("El genero debería haberse asignado correctamente",
        libro.getGenero(), genero);
    assertEquals("Las observaciones deben haberse asignado correctamente
        ", libro.getObservaciones(), observaciones);
    assertEquals("El dni deben haberse asignado correctamente", libro.
        getDni(), dni);
  }
```

### 3.4.3 Pruebas unitarias realizadas a la entidad NuevoCliente

Se codifican pruebas unitarias a todos los constructores de la entidad NuevoCliente, para comprobar que funcionan correctamente y tienen los valores correctos cuando son creados.

- Se comprueba que el constructor vacío (sin parámetros) funciona correctamente, para ello se crea una instancia de la clase NuevoCliente y si esta falla, es decir, lanza una excepción al crear una instancia de la clase NuevoCliente, la prueba fallará.

**Código 3.9** Prueba unitaria sobre el primer constructor de la entidad NuevoCliente.

```
@Test
public final void testConstructor1NuevoClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el
      constructor 1 de la entidad NuevoCliente funciona bien");

  NuevoCliente nuevoCliente = new NuevoCliente();
}
```

- La siguiente prueba comprueba que el constructor que acepta el dni, cliente (nombre de usuario), nombre, primer apellido, segundo apellido, dirección, un valor que indica si el cliente es administrador y la contraseña funciona correctamente y los parámetros son correctamente asignados.

**Código 3.10** Prueba unitaria sobre el segundo constructor de la entidad NuevoCliente.

```
@Test
public final void testConstructor2NuevoClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el
      constructor 2 de la entidad NuevoCliente funciona bien");

  Integer dni = 111111;
  String usuario = "Usuario1";
  String nombre = "Nombre";
  String apellido1 = "Primer apellido";
  String apellido2 = "Segundo apellido";
```

```
    String direccion = "Dirección del nuevoCliente";
    Boolean admin = true;
    String contrasena = "contraseña";

    NuevoCliente nuevoCliente;

    nuevoCliente = new NuevoCliente(dni, usuario, nombre, apellido1,
        apellido2, direccion, admin, contrasena);

    assertEquals("El dni debería haberse asignado correctamente",
        nuevoCliente.getDni(), dni);
    assertEquals("El nuevoCliente debería haberse asignado correctamente",
         nuevoCliente.getCliente(), usuario);
    assertEquals("El nombre debería haberse asignado correctamente",
        nuevoCliente.getNombre(), nombre);
    assertEquals("El apellido1 debería haberse asignado correctamente",
        nuevoCliente.getApellido1(), apellido1);
    assertEquals("El apellido2 debería haberse asignado correctamente",
        nuevoCliente.getApellido2(), apellido2);
    assertEquals("La direccion debería haberse asignado correctamente",
        nuevoCliente.getDireccion(), direccion);
    assertEquals("El valor admin debería haberse asignado correctamente",
        nuevoCliente.getAdmin(), admin);
    assertEquals("La contrasena debería haberse asignado correctamente",
        nuevoCliente.getContrasena(), contrasena);
  }
```

- La siguiente prueba comprueba que el constructor con los argumentos dni, nombre, primer apellido, segundo apellido, dirección y admin, un valor que indica si el cliente es administrador, funciona correctamente y los parámetros son correctamente asignados.

**Código 3.11** Prueba unitaria sobre el tercer constructor de la entidad NuevoCliente.

```
  @Test
  public final void testConstructor3NuevoClienteOK() {
    System.out.println("Comienza la prueba que comprueba que el
        constructor 3 de la entidad NuevoCliente funciona bien");

    Integer dni = 111111;
    String nombre = "Nombre";
    String apellido1 = "Primer apellido";
    String apellido2 = "Segundo apellido";
    String direccion = "Dirección del nuevoCliente";
    Boolean admin = true;

    NuevoCliente nuevoCliente;

    nuevoCliente = new NuevoCliente(dni, nombre, apellido1, apellido2,
        direccion, admin);

    assertEquals("El dni debería haberse asignado correctamente",
        nuevoCliente.getDni(), dni);
    assertEquals("El nombre debería haberse asignado correctamente",
        nuevoCliente.getNombre(), nombre);
```

```
    assertEquals("El apellido1 debería haberse asignado correctamente",
        nuevoCliente.getApellido1(), apellido1);
    assertEquals("El apellido2 debería haberse asignado correctamente",
        nuevoCliente.getApellido2(), apellido2);
    assertEquals("La direccion debería haberse asignado correctamente",
        nuevoCliente.getDireccion(), direccion);
    assertEquals("El valor admin debería haberse asignado correctamente",
        nuevoCliente.getAdmin(), admin);
  }
```

### 3.4.4 Pruebas unitarias realizadas a la entidad Reserva

Se codifican pruebas unitarias a todos los constructores de la entidad Reserva, para comprobar que funcionan correctamente y tienen los valores correctos cuando son creados.

- Se comprueba que el constructor vacío (sin parámetros) funciona correctamente, para ello se crea una instancia de la clase Reserva y si esta falla, es decir, lanza una excepción al crear una instancia de la clase Reserva, la prueba fallará.

**Código 3.12** Prueba unitaria sobre el primer constructor de la entidad Reserva.

```
  @Test
  public final void testConstructor1ReservaOK() {
    System.out.println("Comienza la prueba que comprueba que el
        constructor 1 de la entidad Reserva funciona bien");

    Reserva reserva = new Reserva();
  }
```

- El constructor que acepta un solo parámetro, el isbn, instancia una clase de la entidad Reserva y se comprueba que el parámetro dni se asigna correctamente.

**Código 3.13** Prueba unitaria sobre el segundo constructor de la entidad Reserva.

```
  @Test
  public final void testConstructor2ReservaOK() {
    System.out.println("Comienza la prueba que comprueba que el
        constructor 2 de la entidad Reserva funciona bien");

    Long isbn = 123456L;
    Reserva reserva;

    reserva = new Reserva(isbn);

    assertEquals("El ISBN debería haberse asignado correctamente", reserva.
        getIsbn(), isbn);
  }
```

- La siguiente prueba comprueba que el constructor que acepta el isbn y el dni funciona correctamente y los parámetros son correctamente asignados.

**Código 3.14** Prueba unitaria sobre el tercer constructor de la entidad Reserva.

```
@Test
public final void testConstructor3ReservaOK() {
  System.out.println("Comienza la prueba que comprueba que el
      constructor 3 de la entidad Reserva funciona bien");

  Long isbn = 123456L;
  Integer dni = 111111;


  Reserva reserva;

  reserva = new Reserva(isbn, dni);

  assertEquals("El ISBN debería haberse asignado correctamente", reserva.
      getIsbn(), isbn);
  assertEquals("El dni debe haberse asignado correctamente", reserva.
      getDni(), dni);
}
```

- La siguiente prueba comprueba que el constructor con los argumentos isbn, titulo, autor, editorial, anio_pub, genero, observaciones y el dni del cliente que ha realizado una reserva de ese libro, funciona correctamente y los parámetros son correctamente asignados.

**Código 3.15** Prueba unitaria sobre el cuarto constructor de la entidad Reserva.

```
@Test
public final void testConstructor4ReservaOK() {
  System.out.println("Comienza la prueba que comprueba que el
      constructor 4 de la entidad Reserva funciona bien");

  Long isbn = 123456L;
  String titulo = "Don Quijote de la Mancha";
  String autor = "Miguel de Cervantes Saavedra";
  String editorial = "Francisco de Robles";
  Integer anio_pub = 1605;
  String genero = "Novela de aventuras";
  String observaciones = null;
  Integer dni = 111111;

  Reserva reserva;

  reserva = new Reserva(isbn, titulo, autor, editorial, anio_pub, genero,
       observaciones, dni);

  assertEquals("El ISBN debería haberse asignado correctamente", reserva.
      getIsbn(), isbn);
  assertEquals("El titulo debería haberse asignado correctamente",
      reserva.getTitulo(), titulo);
  assertEquals("El autor debería haberse asignado correctamente",
      reserva.getAutor(), autor);
  assertEquals("La editorial debería haberse asignado correctamente",
      reserva.getEditorial(), editorial);
  assertEquals("El anio de publicacion debería haberse asignado
      correctamente", reserva.getAnio_pub(), anio_pub);
```

```
    assertEquals("El genero debería haberse asignado correctamente",
        reserva.getGenero(), genero);
    assertEquals("Las observaciones deben haberse asignado correctamente",
         reserva.getObservaciones(), observaciones);
    assertEquals("El dni debe haberse asignado correctamente", reserva.
        getDni(), dni);
  }
```

Las pruebas realizadas en el proyecto son unas pruebas sencillas que servirían para verificar el correcto funcionamiento de las clases usadas en la aplicación, se necesitaría más realización de pruebas para estar seguros de no tener errores en la aplicación con solo ejecutarlas.

## 3.5 Automatización del proyecto

Una vez se tiene una pequeña muestra de pruebas realizadas sobre el proyecto, se configura *Apache Maven* para la gestión y construcción del proyecto, lo que facilita la integración y entrega continua en el proyecto. Para ello se modifica primero la estructura de ficheros del proyecto original, adecuándola a la usada por un proyecto gestionado por *Maven*, la estructura que se encuentra a continuación es la estructura inicial del proyecto.

```
.
|--- WebContent
|   |--- META-INF
|   |   --- MANIFEST.MF
|   |--- WEB-INF
|   |   |--- classes
|   |   |   |--- controlador
|   |   |   |   |--- Borrar_Reserva.class
|   |   |   |   |--- Insert_cliente.class
|   |   |   |   |--- Insert_libro.class
|   |   |   |   |--- Listar_Clientes.class
|   |   |   |   |--- Listar_Libros.class
|   |   |   |   |--- Listar_Reservas.class
|   |   |   |   |--- Listar_Reservas_Cliente.class
|   |   |   |   |--- Login.class
|   |   |   |   --- Reservar_Libro.class
|   |   |   |--- modelo
|   |   |   |   |--- Cliente.class
|   |   |   |   |--- Conexion.class
|   |   |   |   |--- Libro.class
|   |   |   |   |--- NuevoCliente.class
|   |   |   |   --- Reserva.class
|   |   |   --- reserva_libros
|   |   |       --- Constantes.class
|   |   |--- lib
|   |   |   |--- jstl-1.2.jar
|   |   |   --- postgresql.jar
|   |   --- web.xml
|   |--- borrar.jsp
|   |--- borrar_reserva_correcto.jsp
|   |--- borrar_reserva_error.jsp
|   |--- css
|   |   --- estilorp.css
|   |--- datos.jsp
|   |--- images
|   |   --- logoDep.jpg
```

```
|   |--- inicio.jsp
|   |--- inicio_tras_error.jsp
|   |--- ins_cliente_correcto.jsp
|   |--- ins_cliente_error.jsp
|   |--- ins_libro_correcto.jsp
|   |--- ins_libro_error.jsp
|   |--- js
|   |   |--- compr_nuevo_cliente.js
|   |   |--- compr_nuevo_libro.js
|   |   |--- funciones.js
|   |   |--- funcionesrp.js
|   |   |--- global.js
|   |   --- jsinicio.js
|   |--- listar_clientes.jsp
|   |--- listar_libros.jsp
|   |--- login.jsp
|   |--- logout.jsp
|   |--- menu.jsp
|   |--- nuevo_cliente.jsp
|   |--- nuevo_libro.jsp
|   |--- reserva_libro_correcto.jsp
|   |--- reservar.jsp
|   |--- reservas.jsp
|   --- reservas_cliente.jsp
|--- build
|   --- classes
|       |--- controlador
|       |   |--- Insert_libro.class
|       |   |--- Inserta_Cliente.class
|       |   |--- Listar_Clientes.class
|       |   --- Login.class
|       |--- modelo
|       |   |--- Cliente.class
|       |   |--- Conexion.class
|       |   --- Libro.class
|       --- reserva_libros
|           --- Constantes.class
|--- src
|   |--- controlador
|   |   |--- Borrar_Reserva.java
|   |   |--- Insert_cliente.java
|   |   |--- Insert_libro.java
|   |   |--- Listar_Clientes.java
|   |   |--- Listar_Libros.java
|   |   |--- Listar_Reservas.java
|   |   |--- Listar_Reservas_Cliente.java
|   |   |--- Login.java
|   |   --- Reservar_Libro.java
|   |--- modelo
|   |   |--- Cliente.java
|   |   |--- Conexion.java
|   |   |--- Libro.java
|   |   |--- NuevoCliente.java
|   |   --- Reserva.java
|   --- reserva_libros
|       --- Constantes.java
```

Las modificaciones que se han realizado son:

- Se ha eliminado la carpeta build y su contenido, se compilará y creará la carpeta cuando se necesite partiendo de los ficheros fuente que se disponen en el proyecto.
- La carpeta 'WebContent' ha sido movida a 'src/webapp'.

A continuación, se puede observar la nueva estructura del proyecto.

```
.
|--- src
|   |--- controlador
|   |   |--- Borrar_Reserva.java
|   |   |--- Insert_cliente.java
|   |   |--- Insert_libro.java
|   |   |--- Listar_Clientes.java
|   |   |--- Listar_Libros.java
|   |   |--- Listar_Reservas.java
|   |   |--- Listar_Reservas_Cliente.java
|   |   |--- Login.java
|   |   --- Reservar_Libro.java
|   |--- modelo
|   |   |--- Cliente.java
|   |   |--- Conexion.java
|   |   |--- Libro.java
|   |   |--- NuevoCliente.java
|   |   --- Reserva.java
|   |--- reserva_libros
|   |   --- Constantes.java
|   --- webapp
|       |--- WEB-INF
|       |   --- web.xml
|       |--- borrar.jsp
|       |--- borrar_reserva_correcto.jsp
|       |--- borrar_reserva_error.jsp
|       |--- css
|       |   --- estilorp.css
|       |--- datos.jsp
|       |--- images
|       |   --- logoDep.jpg
|       |--- inicio.jsp
|       |--- inicio_tras_error.jsp
|       |--- ins_cliente_correcto.jsp
|       |--- ins_cliente_error.jsp
|       |--- ins_libro_correcto.jsp
|       |--- ins_libro_error.jsp
|       |--- js
|       |   |--- compr_nuevo_cliente.js
|       |   |--- compr_nuevo_libro.js
|       |   |--- funciones.js
|       |   |--- funcionesrp.js
|       |   |--- global.js
|       |   --- jsinicio.js
|       |--- listar_clientes.jsp
|       |--- listar_libros.jsp
|       |--- login.jsp
|       |--- logout.jsp
|       |--- menu.jsp
|       |--- nuevo_cliente.jsp
```

```
|       |--- nuevo_libro.jsp
|       |--- reserva_libro_correcto.jsp
|       |--- reservar.jsp
|       |--- reservas.jsp
|       --- reservas_cliente.jsp
--- test
    --- unidad
        |--- ClienteTest.java
        |--- LibroTest.java
        |--- NuevoClienteTest.java
        |--- ReservaTest.java
        --- package-info.java
```

Los alumnos deberán tener instalado *Maven* en el equipo, para ello deberán:

1. Descargar el fichero 'tar.gz' de la siguiente URL *http://apache.uvigo.es/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz*.
2. Descomprimir el fichero con el siguiente comando.

```
$ tar xzvf apache-maven-3.6.3-bin.tar.gz
```

3. Mover la carpeta resultante de la extracción anterior a la carpeta '/opt' y añadir el directorio a la variable de entorno PATH.

```
$ mv apache-maven-3.6.3 /opt/apache-maven-3.6.3
$ export PATH=/opt/apache-maven-3.6.3/bin:$PATH
```

Después se procede a crear el fichero que indique a *Maven* cómo gestionar el proyecto, para ello se crea un fichero llamado 'pom.xml' en el directorio raíz del proyecto, este fichero es el encargado de indicar a Maven toda la información requerida sobre el proyecto. El fichero 'pom.xml' que ha sido creado para este proyecto tiene la siguiente forma:

**Listing 3.16** Fichero pom.xml.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      <!-- Estos párametros son necesarios para Maven -->
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <!-- Versión del archivo POM soportada, este parametro lo necesita Maven -->
  <modelVersion>4.0.0</modelVersion>
  <!-- Id del grupo de desarrollo -->
  <groupId>com.reserva_libro.etsi</groupId>
  <!-- Id del proyecto -->
  <artifactId>reserva_libros</artifactId>
  <!-- Formato de salida, en este caso interesa war para el despliegue en
      Tomcat -->
  <packaging>war</packaging>
  <!-- Versión del proyecto -->
  <version>1</version>

  <!-- Información útil para la construcción de la aplicación -->
  <build>
    <!-- Directorio dónde se encuentra el código Java -->
```

```
<sourceDirectory>${project.basedir}/src</sourceDirectory>
<!-- Directorio dónde se encuentran las pruebas -->
<testSourceDirectory>${project.basedir}/test</testSourceDirectory>

<!-- Información relacionada con los plugins necesarios -->
<plugins>
  <!-- Declaración de un plugin, este plugin es lo necesita Maven -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.19.1</version>
  </plugin>

  <!-- Declaración de un plugin, este plugin lo necesita Maven para crear
      el fichero war -->
  <plugin>
    <!-- Id del grupo de desarrollo de este plugin -->
    <groupId>org.apache.maven.plugins</groupId>
    <!-- Id del plugin -->
    <artifactId>maven-war-plugin</artifactId>
    <!-- Versión del plugin -->
    <version>2.6</version>
    <!-- Configuración adicional para el plugin -->
    <configuration>
      <!-- Ubicación del fichero 'web.xml'\ -->
      <webXml>${project.basedir}/src/webapp/WEB-INF/web.xml</webXml>
      <webResources>
        <resource>
          <!-- Dónde se encuentra el código necesario que no es Java para la
               construcción de la aplicación -->
          <directory>${project.basedir}/src/webapp</directory>
        </resource>
      </webResources>
    </configuration>
  </plugin>

  <!-- Declaración de un plugin, este plugin lo necesita Maven para
      desplegar sobre Tomcat -->
  <plugin>
    <!-- Id del grupo de desarrollo de este plugin -->
    <groupId>org.apache.tomcat.maven</groupId>
    <!-- Id del plugin -->
    <artifactId>tomcat7-maven-plugin</artifactId>
    <!-- Versión del plugin -->
    <version>2.2</version>
    <!-- Configuración adicional para el plugin -->
    <configuration>
      <!-- URL dónde se encuentra el servicio, es configurable mediante
          linea de comandos con los argumentos -Dserver.addr y -Dserver.port
           -->
      <url>http://${server.addr}:${server.port}/manager/text</url>
      <!-- Se indica que es un servidor Tomcat -->
      <server>TomcatServer</server>
      <!-- Ubicación dónde será desplegada la aplicación -->
      <path>/${name}</path>
      <!-- Usuario necesario para el despliegue -->
      <username>admin</username>
```

```
        <!-- Contraseñas necesaria para el despliegue -->
        <password>1234</password>
      </configuration>
    </plugin>
  </plugins>
</build>

<!-- Configuración de variables por defecto -->
<properties>
  <!-- URL a la que intentará desplegar la aplicación por defecto -->
  <server.addr>localhost</server.addr>
  <!-- Puerto en el que intentará desplegar la aplicación por defecto -->
  <server.port>4000</server.port>
  <!-- Parametro -source pasado al compilador java requerido por Maven -->
  <maven.compiler.source>7</maven.compiler.source>
  <!-- Parametro -target pasado al compilador java requerido por Maven -->
  <maven.compiler.target>7</maven.compiler.target>
</properties>

<!-- Declaración de dependencias de la aplicación -->
<dependencies>
  <!-- Declaración de dependencia, necesaria para el funcionamiento de la
      aplicación -->
  <dependency>
    <!-- Id del grupo de desarrollo de esta dependencia -->
    <groupId>javax.servlet</groupId>
    <!-- Id de la dependencia -->
    <artifactId>javax.servlet-api</artifactId>
    <!-- Versión de la dependencia -->
    <version>3.0.1</version>
    <!-- Necesario por la dependencias -->
    <scope>provided</scope>
  </dependency>

  <!-- Declaración de dependencia, necesaria para el funcionamiento de la
      aplicación -->
  <dependency>
    <!-- Id del grupo de desarrollo de esta dependencia -->
    <groupId>postgresql</groupId>
    <!-- Id de la dependencia -->
    <artifactId>postgresql</artifactId>
    <!-- Versión de la dependencia -->
    <version>9.1-901-1.jdbc4</version>
  </dependency>

  <!-- Declaración de dependencia, necesaria para el funcionamiento de la
      aplicación -->
  <dependency>
    <!-- Id del grupo de desarrollo de esta dependencia -->
    <groupId>jstl</groupId>
    <!-- Id de la dependencia -->
    <artifactId>jstl</artifactId>
    <!-- Versión de la dependencia -->
    <version>1.2</version>
  </dependency>

  <!-- Declaración de dependencia, necesaria para las pruebas -->
```

```
    <dependency>
      <!-- Id del grupo de desarrollo de esta dependencia -->
      <groupId>junit</groupId>
      <!-- Id de la dependencia -->
      <artifactId>junit</artifactId>
      <!-- Versión de la dependencia -->
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Con esta configuración se pueden ejecutar las pruebas sin instalar nada más que *Maven* y ejecutar el siguiente comando:

```
$ mvn test
```

Y desplegar el proyecto con un simple comando.

```
$ mvn -Dname=${Path en el que será desplegada la versión} -Dserver.addr=${
    Dirección donde se halla el servidor tomcat}$ tomcat7:redeploy
```

Los parámetros '-Dname', '-Dserver.addr' y '-Dserver.port' son opcionales y sus valores por defecto son '/', 'localhost' y '4000' respectivamente.

## 3.6 Puesta en marcha y configuración de *Gitlab*

En este apartado se va a trabajar la instalación de *Gitlab* para el alojamiento de repositorios y la ejecución de tareas automatizadas.

Para ello, se descarga la versión de *Gitlab* auto administrada, la versión 'Self-managed' de la siguiente web *https://about.gitlab.com/install/* y se siguen los pasos según el tipo de instalación que se quiera realizar. En el escenario que se propone en esta práctica se han seguido los siguientes pasos:

1. Se actualizan e instalan los 'OpenSSH' y 'ca-certificates', paquetes requeridos por *Gitlab* para su correcto funcionamiento, para ello se deben ejecutar los siguientes comandos.

   ```
   $ sudo apt-get update
   $ sudo apt-get install -y curl openssh-server ca-certificates
   ```

2. A continuación, se añade el repositorio de *Gitlab* a las fuentes del sistema, para ello se debe descargar mediante el comando 'curl' el script necesario y ejecutar dicho script, todo ello con el siguiente comando:

   ```
   $ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/
       script.deb.sh | sudo bash
   ```

3. Se instala el servicio de *Gitlab*, indicando la URL pública del servidor en la variable de entorno 'EXTERNAL_URL'.

   ```
   $ sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-
       ce
   ```

4. Con esto, ya se tendría *Gitlab* escuchando en el puerto 80, ya se puede navegar hasta la aplicación, crear usuarios y almacenar repositorios.

A continuación, se explica cómo iniciar y configurar un ejecutor para poder ejecutar tareas automatizadas en el servidor *Gitlab*.

### 3.6.1 Configuración para ejecutar las tareas automatizadas

Para la ejecución de tareas automatizadas, *Gitlab* se apoya en un servicio externo, que ejecuta estas tareas en segundo plano, este servicio es conocido como *Gitlab Runner*. Para instalarlo en el equipo se deben seguir los pasos indicados en la siguiente url *https://docs.gitlab.com/runner/install/index.html*. En el caso del entorno que se propone en este documento, se han seguido los siguientes pasos:

1. Se descarga la versión que se desee, en este caso v11.4.2 y la versión para el sistema operativo que se esté usando y arquitectura, en este caso es Linux, AMD64.

```
$ curl -LJO https://s3.amazonaws.com/gitlab-runner-downloads/v11.4.2/
    binaries/gitlab-runner-linux-amd64
```

2. El fichero descargado debe ser ejecutado, para ello se puede usar el siguiente comando y *Gitlab Runner* se instalará.

```
$ dpkg -i gitlab-runner_linux-amd64.deb
```

Ya se tendría el servidor *Gitlab Runner* iniciado y esperando a ser configurado para ejecutar las tareas automatizadas de un repositorio, en el caso del entorno propuesto se usa *Docker* para la preparación del entorno con el software necesario para la ejecución y despliegue de la aplicación.

Se ha elegido *Docker* por su facilidad a la hora de instalar el software necesario, ya que si no se realiza con *Docker* se debería instalar previamente todos los recursos necesarios, entre ellos una base de datos *PostgreSQL* y es posible tener problemas con las versiones instaladas y soportadas por la aplicación. Además, el software ejecutado no tiene acceso a fichero externos al contenedor, con lo que ofrece una capa extra de seguridad y una vez finalizada la ejecución en el contenedor *Docker*, todo dato relacionado con el contenedor es eliminado del sistema y no queda rastro de esta ejecución, todas las ejecuciones son independientes. Esta capa de seguridad es especialmente interesante en este entorno ya que la ejecución de comandos en las tareas automatizadas solo tendrá acceso a datos existentes en el propio contenedor, no podrá editar ningún tipo de dato del sistema.

Se deberá instalar *Docker* en el equipo en el que se haya instalado *Gitlab Runner* previamente. Para ello se han seguido los siguientes pasos:

1. Se actualizan e instalan los paquetes necesarios para el funcionamiento correcto de *Docker*, para ello se deben ejecutar los siguientes comandos, si los paquetes están ya instalados en el sistema no serán instalados.

```
$ sudo apt-get update
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

2. Con el siguiente comando se añade la clave GPG al equipo.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
    add -
```

3. A continuación se añade el repositorio de *Docker* en el sistema.

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

4. Ya estaría el equipo configurado para poder descargar e instalar *Docker* y sus paquetes dependientes. Con el siguiente comando ya estaría instalado *Docker*.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

5. Para finalizar, se ejecuta el siguiente comando para comprobar que *Docker* funciona correctamente.

```
$ sudo docker run hello-world
```

Llegado a este punto, lo único que queda es configurar una nueva instancia del servicio *Gitlab Runner* (a partir de hora ejecutor) para ejecutar las tareas automatizadas sobre los repositorios que vayan a ser usados, esto será explicado a continuación. Ante esta situación se tiene la posibilidad de configurar para cada repositorio un ejecutor cada vez que se cree un repositorio nuevo, pero en este proyecto se ha ido un poco más allá y se ha optado por configurar un ejecutor que automatiza las tareas sobre todos los repositorios, para ello se necesita un usuario administrador, para dar a un usuario permisos de administrador debe realizar los siguientes pasos:

1. En el sistema donde se está ejecutando el servicio *Gitlab* se debe ejecutar el siguiente comando para entrar a la consola de *Ruby on Rails* para editar un usuario existente y darle permisos de administrador:

```
$ sudo gitlab-rails console production
```

2. En la consola de *Rails* se debe seguir los siguientes pasos para dar permisos de administrador al usuario que se desee.
   a) El siguiente código encuentra el usuario que se quiera dar permisos de administrador (este usuario debió ser creado previamente en la aplicación), para ello se ofrecen dos posibilidades, mediante el nombre de usuario:

```
user = User.find_by(username: 'Nombre de usuario')
```

   O a través del correo electrónico:

```
user = User.find_by(email: 'Email del usuario')
```

   b) Una vez se tiene el usuario al que se desea dar permisos de administrador, se da permisos y se guarda el usuario con el siguiente código.

```
user.admin = true
user.save!
```

3. El usuario ya tiene permisos de administrador, cuando inicie sesión puede ir a la sección de administración que se encuentra en *Gitlab* en la ubicación '/admin'.

Una vez se tiene un usuario con permisos de administrador se procede a configurar un ejecutor para todos los repositorios.

1. Iniciar sesión con un usuario que tenga permisos de administrador y navegar a la url donde se encuentra el servidor de *Gitlab*, a la sección '/admin'.

2. Una vez iniciada sesión se debe seleccionar en la barra lateral la opción 'Runners' como se ve en la siguiente imagen.

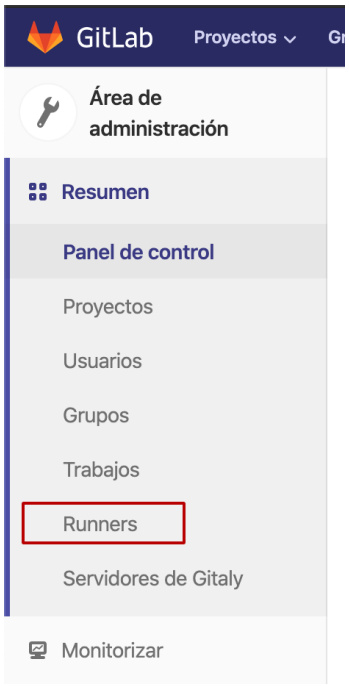


**Figura 3.4** Configuración de runners en la zona de administración de *Gitlab*.

3. Aquí se encuentra (en la sección de la derecha) el token que se debe usar para registrar el ejecutor como ejecutor compartido, como se puede observar en la siguiente imagen.

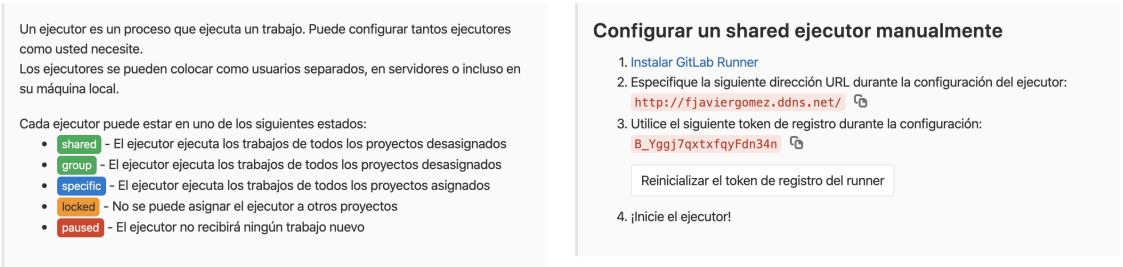


**Figura 3.5** Ejemplo de configuración necesaria para configurar un *Runner* compartido.

4. A continuación se debe indicar por la línea de comandos al servicio *Gitlab Runner* que debe crear una instancia de un ejecutor y se deben facilitar los parámetros necesarios para la identificación como runner compartido, que se ha obtenido en el paso anterior. Para ello se deben seguir los pasos indicados a continuación:

   **a)** Ejecutar el siguiente comando:

```
$ sudo gitlab-runner register
```

   **b)** Introducir la URL de tu servidor *Gitlab*.

```
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.
    com )
$ {La URL de tu servidor}
```

**c)** Introduce el token obtenido en **3.** para registrar el ejecutor.

```
Please enter the gitlab-ci token for this runner
$ {Aquí debe introducir el token obtenido anteriormente}
```

**d)** Introducir una descripción para este ejecutor.

```
Please enter the gitlab-ci description for this runner
$ [hostname] my-runner
```

**e)** Introducir si se desea una etiqueta para identificar el ejecutor en proceso de creación.

```
Please enter the gitlab-ci tags for this runner (comma separated):
$ my-tag,another-tag
```

**f)** Introducir el ejecutor de tareas que se desea usar, en este caso se introduce 'docker'.

```
Please enter the executor: ssh, docker+machine, docker-ssh+machine
    , kubernetes, docker, parallels, virtualbox, docker-ssh, shell
    :
$ docker
```

**g)** Al elegir docker, pregunta por la imagen docker que usará por defecto si otra imagen no es indicada en el fichero '.gitlab-ci.yml', en el caso del entorno preparado da igual ya que será siempre indicado, pero introducimos la misma que será usada, 'maven:3.6-jdk-13', esta imagen almacenada en *Docker Hub* [3], la plataforma para compartir imágenes desarrolladas por otras empresas o desarrolladores, contiene *Apache Maven* y el entorno de desarrollo de *Java*, *JDK* 13.

```
Please enter the Docker image (eg. ruby:2.6):
$ maven:3.6-jdk-13
```

## 3.7 Automatización de tareas

Llegados a este punto ya se tiene todo el entorno preparado para almacenar repositorios, además de poder ejecutar tareas automatizadas sobre ellos, ahora bien, ¿cómo se configura la ejecución de estas tareas? Para la configuración y ejecución de tareas automatizadas se debe crear un fichero '.gitlab-ci.yml' en el que se indiquen las distintas fases que debe ejecutar y qué acciones debe ejecutar en cada una de ellas.

A continuación se puede observar el fichero '.gitlab-ci.yml' creado para la práctica.

**Código 3.17** Fichero .gitlab-ci.yml para la ejecución de tareas automatizadas.

```
stages: # Este atributo sirve para indicar las distintas fases que se tendrán
    en la tarea automatizada.
- pruebas # La primera fase es 'pruebas'.
- despliegue # La segunda fase es 'despliegue'.

default: # Este atributo indica los parámetros que se tomarán por defecto en
    todas las fases.
  image: maven:3.6-jdk-13 # La imagen docker usada por defecto en las fases es
      'maven:3.6-jdk-13'.

services: # Servicios extras que se necesiten en el entorno.
```

```
- postgres:10 # Es necesario la ejecución de una base de datos postgres para la
    ejecución de las pruebas.

variables: # Variables globales en el entorno.
  POSTGRES_DB: reserva_libros # Esta variable es usada por postgres y creará
      una base de datos llamada 'reserva_libros'.
  POSTGRES_USER: dit # Postgres creará un usuario llamado 'dit'.
  POSTGRES_PASSWORD: "dit" # La contraseña es 'dit'.

pruebas: # Inicio de la primera tarea.
  stage: pruebas # Pertenece a la primera fase llamada 'pruebas'.
  only: # Esta tarea sólo debe ejecutarse en los casos indicados a continuación
      .
    - merge_requests # Sólo debe ejecutarse para modificaciones sobre una
        petición de integración.
  script: # Qué debe ser ejecutado en esta tarea.
    - echo 'Ejecutando pruebas' # Mostrar un mensaje por la línea de comandos
        que indique el inicio de la ejecución de las pruebas.
    - cd ./reserva_libros/ # Se cambia el directorio al directorio donde se
        encuentra la aplicación.
    - mvn clean test # Se ejecutan las pruebas existentes en el proyecto, con
        el argumento clean se indica que elimine todos los ficheros que cree
        para la ejecución de las pruebas.

despliegue: # Inicio de la segunda tarea.
  stage: despliegue # Pertenece a la segunda fase llamada 'despliegue'.
  only:
    - merge_requests # Sólo debe ejecutarse para modificaciones sobre una
        petición de integración.
  variables:
    DB_ADDR: 'localhost' # Variable de entorno que será usada por la aplicación
         para definir la dirección donde se encuentra la base de datos.
  script:
    - echo 'Desplegando' # Mostrar un mensaje por la línea de comandos que
        indique el inicio del despliegue.
    - cd ./reserva_libros/ # Se cambia el directorio al directorio donde se
        encuentra la aplicación.
    - mvn -Dmaven.test.skip=true -Dname=$GITLAB_USER_LOGIN -Dserver.addr
        =172.17.0.1 tomcat7:redeploy # Se inicia el despliegue sin ejecutar las
         pruebas, sobre la dirección '172.17.0.1'\ y la aplicación será
        desplegada en el path con valor el nombre de usuario del usuario que
        ejecutó este despliegue.
```

Con este fichero al actualizar una petición de integración ('merge_request'), se ejecutarán las distintas tareas automatizadas, si en alguna de ellas un comando devuelve un código de error, la ejecución finalizará y no continuará ejecutando las distintas tareas. Las pruebas existentes en el proyecto serán ejecutadas y si el resultado de estas es correcto se procederá al despliegue de la aplicación en el servidor *Tomcat*, el cuál se procede a explicar en el siguiente apartado.

## 3.8 Servidor Tomcat

Como se ha visto es necesaria la existencia de un servidor *Tomcat* en el sistema para el despliegue de la aplicación, por ello se van a explicar los pasos necesarios para la instalación.

1. Se debe descargar el paquete *Tomcat* de su web oficial, para ello se debe acceder a la siguiente dirección *http://apache.uvigo.es/tomcat/tomcat-8/v8.5.53/bin/apache-tomcat-8.5.53.tar.gz*.

2. Se crea una carpeta para *Tomcat* en '/opt/tomcat' y se descomprime el fichero anteriormente descargado en ella, para ello se deben ejecutar los siguientes comandos:

```
$ sudo mkdir /opt/tomcat
$ sudo tar xzvf apache-tomcat-8*tar.gz -C /opt/tomcat --strip-
    components=1
```

3. Se configura el inicio automático del servicio cuando el equipo inicia, con el comando 'chkconfig', este comando sirve para configurar la administración de servicios con 'SystemV' de forma sencilla.

```
$ chkconfig --level 2345 \opt\tomcat\bin\startup.sh on
```

## 3.9 Copias de seguridad creadas de forma automática

Se procede a explicar la realización de una copia de seguridad diaria de una instancia *Gitlab*. Se ha decidido realizar copias de seguridad de forma periódicas ya que este entorno se realizará para llevar a cabo una práctica donde los alumnos subirán ficheros, tienen que colaborar entre ellos y si hay algún problema se debe poder restaurar la aplicación a un estado anterior. Para configurar la realización de copias de seguridad de forma periódicas se han seguido los pasos indicados en [14].

Los requisitos necesarios en el sistema son los siguientes:

- **Rsync.** Es una aplicación que ofrece transferencia rápida de archivos de forma incremental. Se puede comprobar la existencia de *Rsync* en el sistema con el siguiente comando:

```
$ rsync --version
```

  Si no se encuentra instalado en el sistema bastaría con ejecutar el comando

```
$ sudo apt-get install rsync
```

- **Tar.** Se debe tener la versión 1.30 o superior, para comprobarlo se debe ejecutar el comando

```
$ tar --version
```

Una vez el sistema cumpla los requisitos se procede a la configuración de la ejecución automática de copias de seguridad.

1. Se debe editar el fichero '/etc/gitlab/gitlab.rb', añadiendo el siguiente contenido para que las copias de seguridad no permanezcan más de 7 días y saturen el almacenamiento.

```
## Limit backup lifetime to 7 days - 604800 seconds
gitlab_rails['backup_keep_time'] = 604800
```

2. Se debe reconfigurar la instancia de *Gitlab* para que los cambios se vean reflejados:

```
$ sudo gitlab-ctl reconfigure
```

3. A continuación se configura *cron* para que realice las copias de seguridad de forma periódica, para ello se inicia sesión en la terminal como 'root' usando el siguiente comando:

```
$ sudo su -
```

4. Una vez se inicie sesión como 'root', se edita cron con el siguiente comando.

```
$ crontab -e
```

5. Editando la configuración de *cron*, se indica que se realice todos los días a las 2 AM añadiendo el siguiente contenido:

```
0 2 * * * /opt/gitlab/bin/gitlab-backup create CRON=1
```

Con esta configuración se tendría una copia de seguridad periódica de todos los datos existentes en el servidor *Gitlab*.

### 3.9.1 Restauración de copia de seguridad

Cuando se desee restaurar una copia de seguridad, lo primero que debemos saber es a que copia de seguridad se desea restaurar el servicio *Gitlab*, para ello, se pueden encontrar las distintas copias de seguridad en el directorio: */var/opt/gitlab/backups*. A continuación, se debe parar los servicios conectados a la base de datos de *Gitlab* con los siguientes comandos.

```
$ sudo gitlab-ctl stop unicorn
$ sudo gitlab-ctl stop puma
$ sudo gitlab-ctl stop sidekiq

# Verificar el estado de los servicios
$ sudo gitlab-ctl status
```

Una vez se encuentran parados los servicios, se ejecuta el siguiente comando donde se debe indicar el timestamp de la copia de seguridad a la que se desea restaurar el servicio.

```
$ sudo gitlab-backup restore BACKUP={Timestamp de la copia de seguridad}
```

Por último, se debe reiniciar *Gitlab* y se encontrará en el estado restablecido.

```
$ sudo gitlab-ctl restart
```

## 3.10 Inicialización de datos necesarios para la práctica

La práctica propuesta en este proyecto necesita de usuarios y datos configurados en los distintos repositorios. Para ello se ha desarrollado un script que haga esto. Bastaría con ejecutar el siguiente comando:

```
$ ./inicializar_datos
```

Este script moverá el archivo con unas *Rake Tasks* a el directorio donde deben estar las rake tasks en *Gitlab*, después ejecutará la *Rake Task* que inicializa todos los datos necesarios para la práctica. El contenido del script es el siguiente:

**Código 3.18** Script de inicialización de datos.

```
#! /bin/bash

sudo cp ./create.rake /opt/gitlab/embedded/service/gitlab-rails/lib/tasks/
    gitlab/create.rake

actualizar_repositorio() {
  nombre_repositorio=$1
```

```
  echo "El nombre de mi repositorio es $nombre_repositorio"

  if [ -z nombre_repositorio ]; then
    echo "Se debe pasar el nombre del repositorio como parámetro."
  else
    git remote add $nombre_repositorio http://localhost/dit/$nombre_repositorio.
        git

    git fetch $nombre_repositorio
    borrar_ramas_repositorio_remoto $nombre_repositorio
    git push -u $nombre_repositorio --all --force

    git remote remove $nombre_repositorio
  fi
}

borrar_ramas_repositorio_remoto() {
  nombre_repositorio=$1

  if [ -z nombre_repositorio ]; then
    echo "Se debe pasar el nombre del repositorio como parámetro."
  else
    git branch -r |
    grep $nombre_repositorio/ |
    grep -v 'master$' |
    grep -v HEAD |
    cut -d/ -f2 |
    while read branch; do
      echo "Borrando rama remota $branch de $nombre_repositorio"
      git push $nombre_repositorio :$branch
    done
  fi
}

if [ $? -eq 0 ]; then
  sudo gitlab-rake dit:inicializar

  actualizar_repositorio proyecto1
  actualizar_repositorio proyecto2
  actualizar_repositorio proyecto3

  sudo gitlab-rake cache:clear
fi
```

### 3.10.1 Rake tasks

Uno de los objetivos de este proyecto es la elaboración de un script para la inicialización de los datos necesarios para la práctica, para ello, cómo se ha presentado anteriormente se ha realizado un script que lo realice, uno de los pasos de este script es la ejecución de una *Rake Task*, codificada en *Ruby*. Se han desarrollado 4 *Rake Tasks* en este proyecto, tres de ellas para inicializar los datos de cada grupo, cada una borra los datos relacionados con ese repositorio y crea los siguientes datos de nuevo:

- Tres repositorios llamados 'ProyectoGX' donde X es el número del grupo perteneciente a la *Rake Task* ejecutada, por ejemplo, en el caso del grupo 1, será llamado 'ProyectoG1'. El creador de este repositorio será el usuario 'dit' del que se hablará posteriormente.

- Diez usuarios con el rol 'Desarrollador', que será el usado en cada pareja de la práctica por alumno que tome dicho rol, los datos de los usuarios son los siguientes, siendo X el número de grupo e Y el número de pareja.
  Nombre: 'DeveloperX-Y'; Email: 'devX-Y@dit.com'; Password:'developerX-Y'; Username: 'devX-Y'
- Diez usuarios con el rol 'Líder de equipo', estos usuarios deberán ser usados por el miembro de la pareja que desarrolle la práctica usando dicho rol, los datos de los usuarios son los siguientes, siendo X el número de grupo e Y el número de pareja.
  Nombre: 'LiderX-Y'; Email: 'liderX-Y@dit.com'; Password: 'liderX-Y'; Username: 'liderX-Y'

.

Para ejecutar dichas rake task se debe ejecutar el siguiente comando, donde X es el número del grupo:

```
$ sudo gitlab-rake dit:inicializar_grupoX
```

La otra *Rake Task* desarrollada se encarga de borrar todas las tablas existentes en la base de datos, configurarlas de nuevo, crear un usuario con permisos de administrador (usuario: 'dit', email: 'admin@dit.com' y contraseña: 'D!t12345') y ejecutar las *Rake Tasks* comentadas anteriormente. Para ejecutar esta *Rake Task* se debe ejecutar el siguiente comando:

```
$ sudo gitlab-rake dit:inicializar
```

A continuación, se encuentra el fichero con las distintas *Rake Tasks* desarrolladas.

**Código 3.19** Rake tasks para inicializar los datos.

```
require 'active_record/fixtures'

N_PAREJAS = 10

namespace :dit do
  desc "DIT | Inicializar los datos"
  task :inicializar => :environment do
    puts "Inicializando datos"

    Rake::Task['gitlab:db:drop_tables'].invoke
    Rake::Task['gitlab:db:configure'].invoke

    admin = User.create!(name: "Dit", email: "admin@dit.com", password:"D!
        t12345", username: "dit", admin: true)
    admin.confirm

    Rake::Task['dit:inicializar_grupo'].invoke(1)
    Rake::Task['dit:inicializar_grupo'].reenable
    Rake::Task['dit:inicializar_grupo'].invoke(2)
    Rake::Task['dit:inicializar_grupo'].reenable
    Rake::Task['dit:inicializar_grupo'].invoke(3)

    ApplicationSetting.update(signup_enabled: false, default_branch_protection:
         0)
    Ci::Runner.create!(token: "dit", runner_type: "instance_type",
        first_day_of_week: 1)
  end

  desc "DIT | Inicializar datos grupo"
  task :inicializar_grupo, [:n_grupo] => :environment do |_, args|
    n_grupo = args[:n_grupo]
```

```
unless n_grupo
  puts "Must specify a migration version as an argument".color(:red)
  exit 1
end

admin = User.find_by(username: "dit")

puts "No se ha ejecutado la tarea inicial." && return unless admin.present?

puts "Inicializando Grupo #{n_grupo}"

repositorio = Project.find_by(name:"ProyectoG#{n_grupo}")
if repositorio.present?
  puts("El repositorio \"ProyectoG#{n_grupo}\" está siendo reiniciado")

  Projects::DestroyService.new(repositorio, admin).execute
end

repositorio = Projects::CreateService.new(admin, name: "ProyectoG#{n_grupo}
    ", path: "proyecto#{n_grupo}", skip_disk_validation: true).execute
repositorio.add_maintainer(admin)

destroy_users = Users::DestroyService.new(admin)

for i in 1..N_PAREJAS do
  # Creando Desarollador
  dev = User.find_by(username: "dev#{n_grupo}-#{i}")

  if dev.present?
    puts("Usuario dev#{n_grupo}-#{i} reiniciado")

    destroy_users.execute(dev)
  end

  dev = Users::CreateService.new(
    admin,
    name: "Developer#{n_grupo}-#{i}",
    email: "dev#{n_grupo}-#{i}@dit.com",
    password:"developer#{n_grupo}-#{i}",
    username: "dev#{n_grupo}-#{i}"
  ).execute
  dev.confirm

  repositorio.add_user(dev,30)

  # Creando Lider de equipo

  lider = User.find_by(username: "lider#{n_grupo}-#{i}")

  if lider.present?
    puts("Usuario lider#{n_grupo}-#{i} reiniciado")

    destroy_users.execute(lider)
  end

  lider = Users::CreateService.new(
    admin,
```

```
        name: "Lider#{n_grupo}-#{i}",
        email: "lider#{n_grupo}-#{i}@dit.com",
        password:"lider#{n_grupo}-#{i}",
        username: "lider#{n_grupo}-#{i}"
      ).execute
      lider.confirm

      repositorio.add_user(lider,30)
    end
  end
end # namespace end: dit
```

## 3.11 Realización de la práctica

Llegados a ese punto se tiene todo el entorno preparado para la realización de la práctica, a continuación, se va a desarrollar paso a paso el procedimiento que el alumno debe seguir para la realización de esta.

Se toma como ejemplo una pareja de trabajo que pertenece al grupo 2 y su número de pareja es el 7.

1. Cada pareja recibe un requisito en el enunciado de la práctica, por ejemplo: 'Se solicita que la entidad Libro no posea un constructor que admita sólo el ISBN'.
2. Los miembros de la pareja deberán decidir quién toma el rol de 'Desarrollador', será el encargado de realizar las modificaciones al código, y quién toma el rol de 'Líder de equipo', creará la tarea y revisará que los cambios a integrar son correctos. A continuación, se encuentran los datos con los que deberán acceder y realizar sus cambios sobre el repositorio al que tienen acceso. Se toma como ejemplo una pareja que pertenece al grupo número 2 y su número de pareja es el 7, deberán realizar sus cambios sobre el repositorio 'ProyectoG2'(sólo tendrán acceso a este).

| | **Usuario** | **Clave** |
|---|---|---|
| **Desarrollador** | dev2-7 | developer2-7 |
| **Lider de equipo** | lider2-7 | lider2-7 |

3. El miembro de la pareja con rol 'Líder de equipo' deberá crear una tarea en el repositorio del servidor *Gitlab* en el que deberá quedar reflejado la descripción de los cambios requeridos en la aplicación, una vez creado deberá asignarle la tarea al otro miembro del equipo como se puede ver en las siguientes capturas.

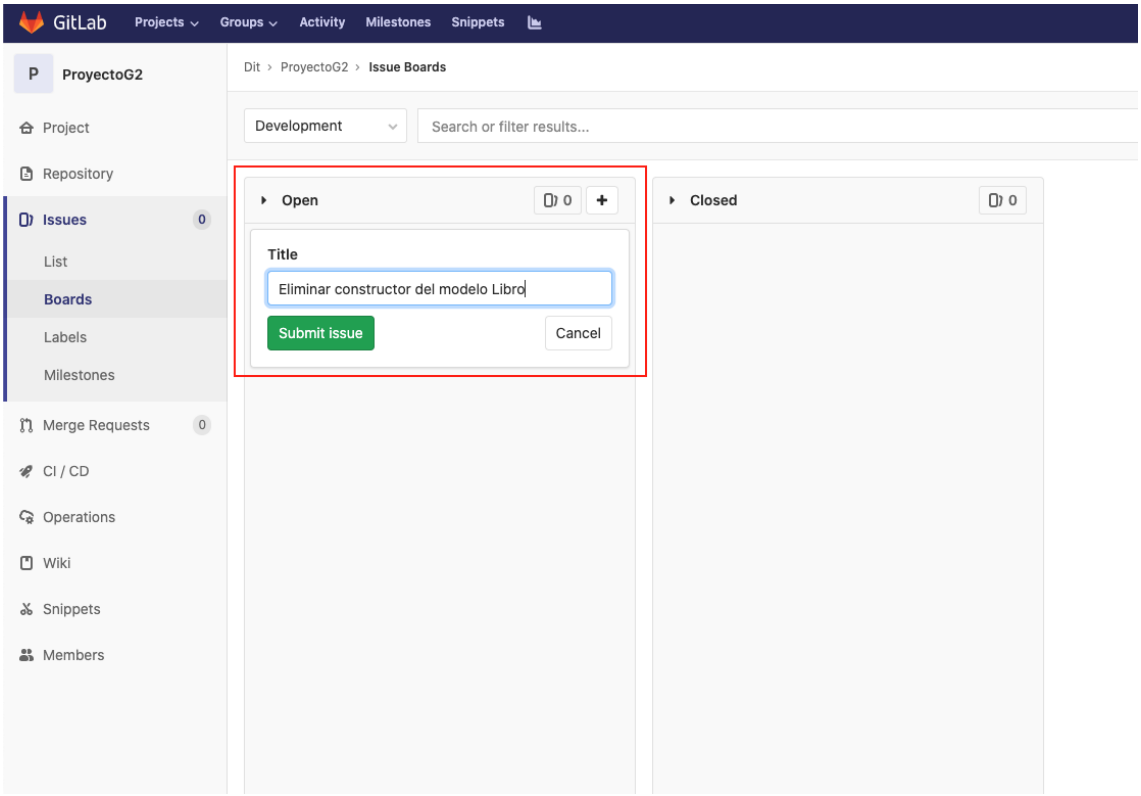


**Figura 3.6** Creación de una tarea en *Gitlab*.

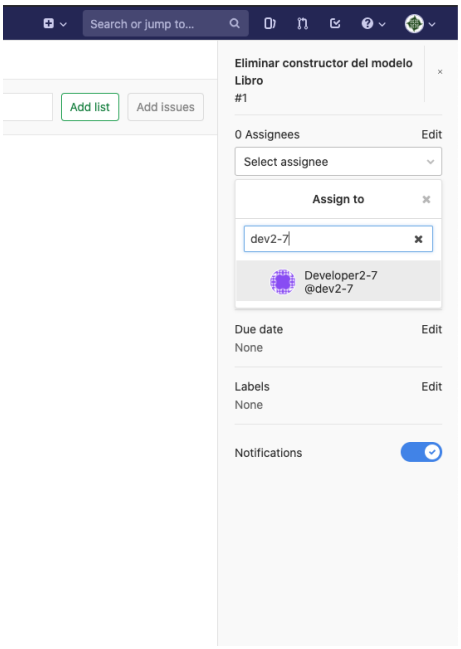


**Figura 3.7** Asignación de una tarea a un miembro del equipo.

4. A continuación, el alumno que haya tomado el rol de 'Desarrollador' deberá ir al tablero de tareas y crear una rama correspondiente a esa tarea como se ve en la siguiente captura. Esta rama será donde deberá hacer las modificaciones necesarias.

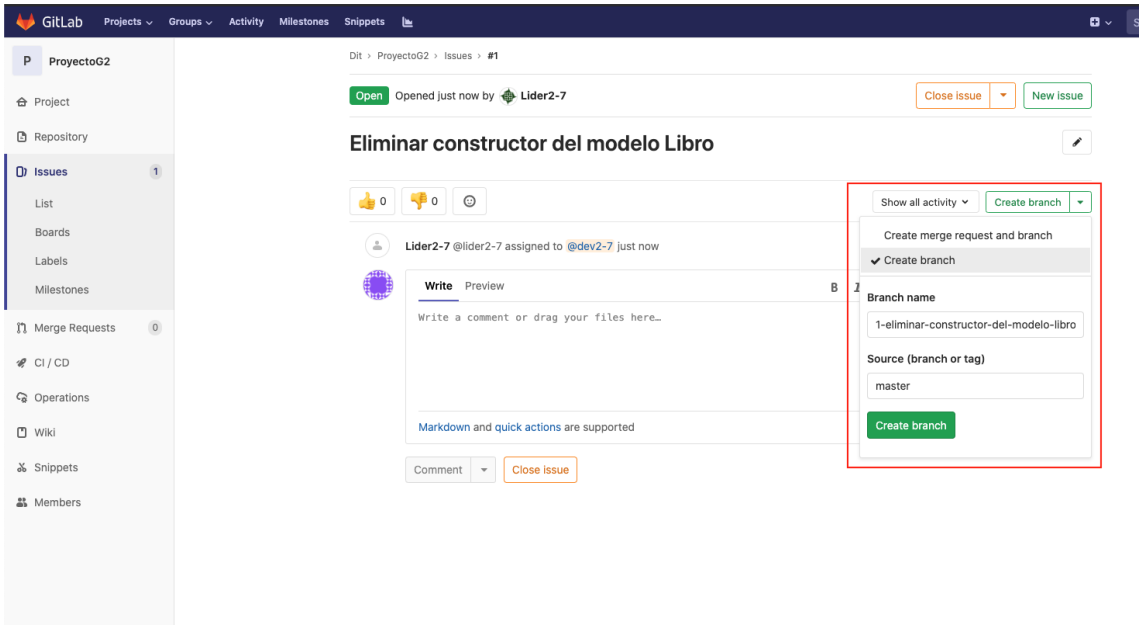


**Figura 3.8** Creación de una rama relacionada con la tarea.

5. Tras haber creado la rama correspondiente a la tarea, se debe realizar las modificaciones necesarias sobre el código en esta rama, para ello, en el ejemplo propuesto, el alumno debe ir al archivo *reserva¡ibros/src/modelo/Libro.javayeliminarelcontenidoexistenteentrelaslneas*38*y*47, *elcdigoaeliminarsepuedeobservarac*

```
  /**
 * Constructor con parmetro la clave primaria (ISBN)
 */
public Libro(Long isbn) {
  this.isbn = isbn;

  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();

}
```

6. Una vez se tenga la modificación realizada, se debe integrar en el proyecto. Para ello se debe realizar un commit con la modificación anterior y subir la rama al servidor, en el cual se debe de crear una petición de integración como se observa a continuación.

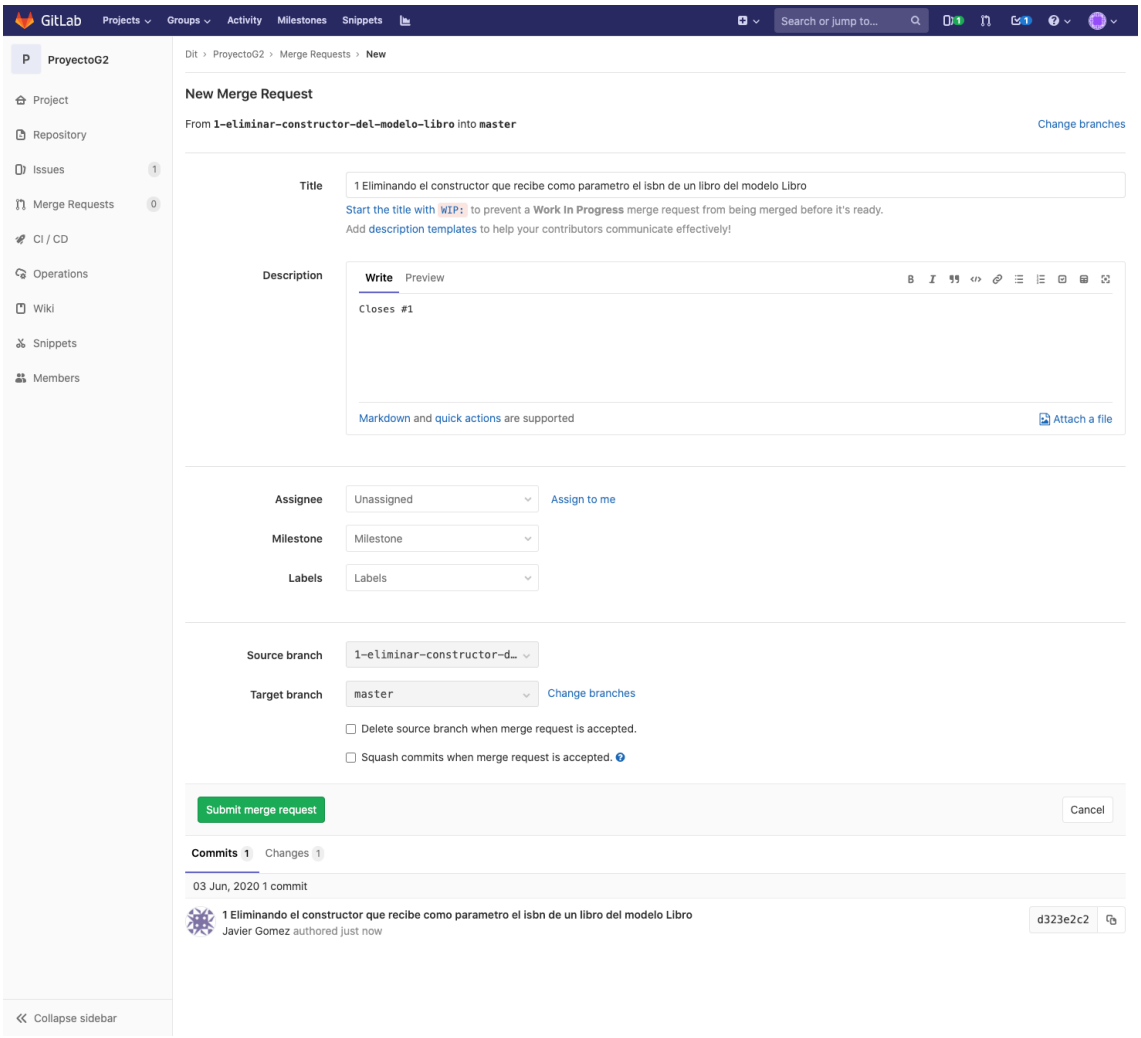


**Figura 3.9** Creación de una petición de integración.

7. Al crear una petición de integración se han debido ejecutar las tareas automatizadas, el alumno debe comprobar el estado de estás, la tarea de ejecución de pruebas ha debido fallar ya que no se han actualizado las pruebas en la aplicación.

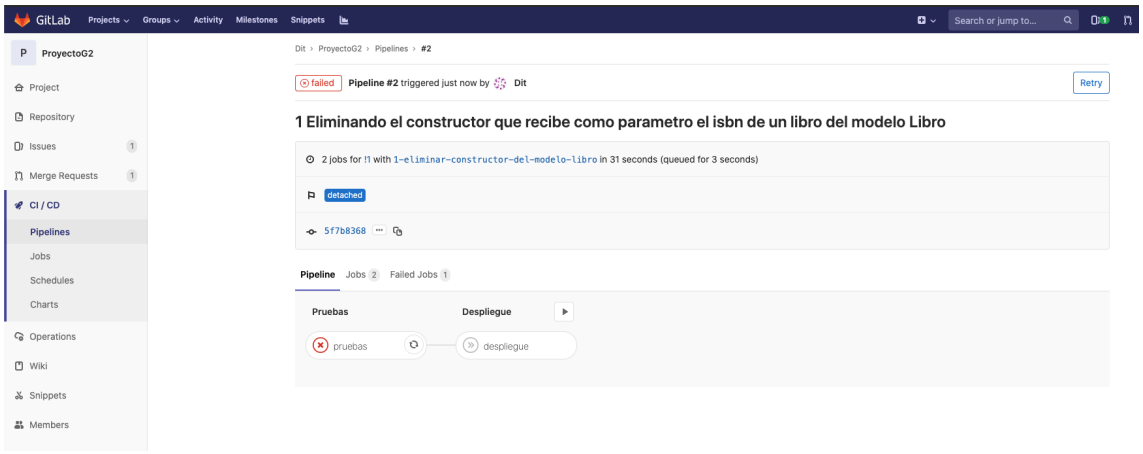


**Figura 3.10** Pruebas con resultado fallido.

8. Se debe analizar el resultado de estas y modificar la prueba necesaria para que las pruebas pasen, se debe editar el fichero $reserva_libros/test/unidad/LibroTest.java, concretamente, eliminar el contenido desde la linea 19 hasta la 30.

```
  @Test
public final void testConstructor2LibroOK() {
  System.out.println("Comienza la prueba que comprueba que el constructor
      2 funciona bien");

  Long isbn = 123456L;

  Libro libro;

  libro = new Libro(isbn);

  assertEquals("El ISBN debería haberse asignado correctamente", libro.
      getIsbn(), isbn);
}
```

9. Una vez actualizadas las pruebas, añada los cambios a la rama y actualícela en el servidor. Compruebe el estado de las tareas automatizadas en este caso. Deben de haber finalizado correctamente, sus cambios han sido desplegados en un entorno de pruebas, en el ejemplo, puede ser encontrado en: 'http://URLSERVIDOR:4000/dev2-7'.

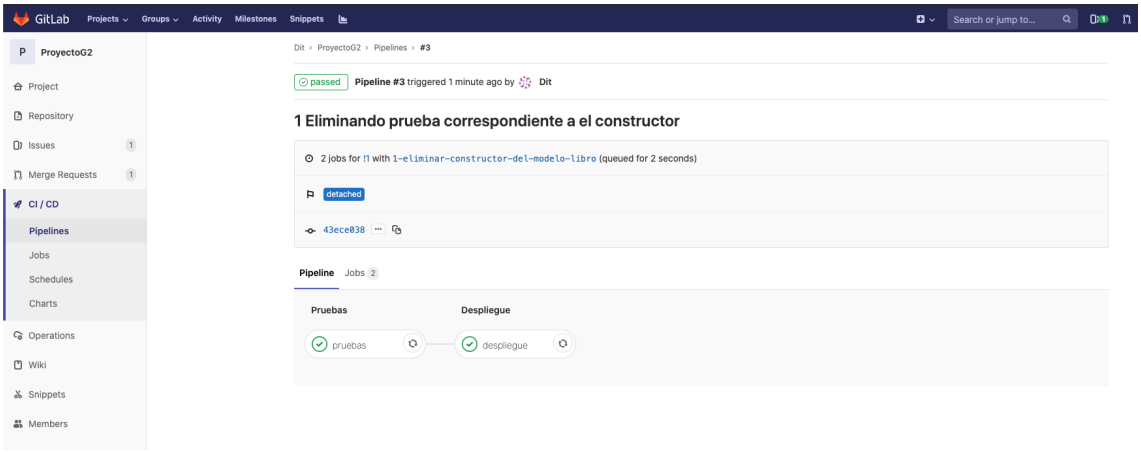


**Figura 3.11** Pruebas con resultado correcto.

10. El alumno que tome el rol 'Líder de equipo' debe revisar el código de la petición de integración y aprobar los cambios si así lo cree.

11. Por último se deben integrar los cambios en la rama principal del repositorio, para ello solo debe clicar el botón como se observa a continuación.

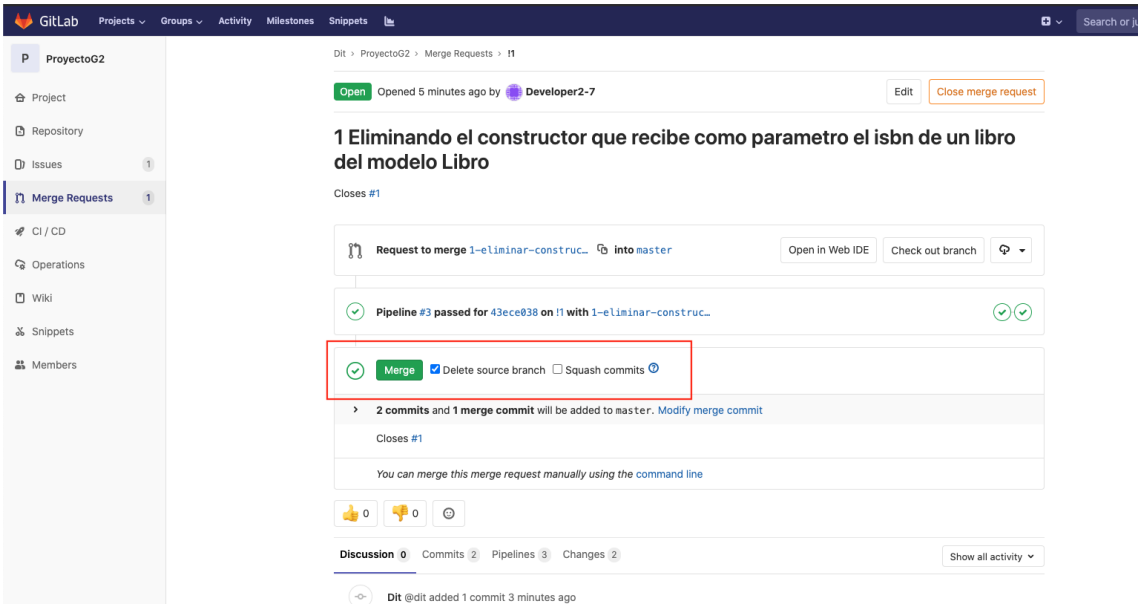


**Figura 3.12** Integrar la petición de integración en el repositorio.

# 4 Conclusiones y líneas futuras

*La vida es una serie de colisiones con el futuro; no es una suma de lo que hemos sido, sino de lo que anhelamos ser.*

José Ortega y Gasset

Se presentan a continuación las conclusiones tras la realización del trabajo y posibles líneas futuras que podrían ser desarrolladas partiendo de este proyecto.

Se han conseguido cubrir todos los objetivos presentados al inicio de este trabajo. Con el entorno desplegado el alumno podrá trabajar sobre un entorno preparado para llevar a cabo un modelo de gestión de proyectos ágil y con ello complementar la teoría vista en clase. El alumno podrá tener una clara visión de las tareas que se están llevando a cabo en el proyecto mediante la tabla de tareas, centrarse en llevar a cabo el desarrollo de la tarea que corresponda y *Gitlab* se encargará de comprobar que la aplicación funciona correctamente mediante las pruebas, gracias a la configuración propuesta, desplegará los cambios en un entorno de desarrollo donde puede ser probada manualmente por el usuario y posteriormente integrada en producción, todo esto sin que el desarrollador deba preocuparse. Este proceso permite una mayor cantidad de entregas al cliente y se obtendrá una mayor retroalimentación del propio cliente que permitirá tener un proyecto lo más completo posible en el tiempo acordado.

Se han establecido los límites del trabajo donde se ha considerado suficiente para la realización de un Trabajo de Fin de Grado, pero se podría seguir desarrollando este proyecto, a continuación, se le presenta varias líneas de trabajo futuras sobre el proyecto anteriormente expuesto.

- Desarrollo de más pruebas, actualmente la aplicación cuenta con un bajo porcentaje de la aplicación cubierto por las pruebas, se propone la realización de pruebas que cubran todo el código.
- Desarrollo de pruebas de integración, se han realizado solamente pruebas unitarias sobre las entidades de los modelos, en esta línea de trabajo se propone la realización de más pruebas, en este caso pruebas de integración que verifiquen que la aplicación funciona correctamente en su conjunto.
- Desarrollo de pruebas de validación, donde se compruebe la interacción con la misma aplicación web como usaría un usuario normal, esto se puede conseguir con un marco de trabajo como *Selenium*. Intentando llevar a cabo el concepto de 'Pirámide del testing' visto anteriormente.
- Cambios en la aplicación desarrollada en su momento por Fabián Muñoz, añadiendo más patrones de diseño software.
- Creación de una fase en la tarea automatizada que cree el documento JavaDoc y lo publique directamente.
- Despliegue real en un entorno de 'producción' cuando los cambios son integrados correctamente en la rama 'master'.
- Uso del entorno desarrollado para otras prácticas. Este entorno puede ser usado tanto para otras prácticas de la asignatura Ingeniería de Software como para otras asignaturas del grado relacionadas con 'DevOps'desarrollando un poco más la parte de automatización de tareas.

Como conclusión, se ha de destacar que la preparación del entorno propuesto para la práctica ha sido un completo acierto, así como el uso del proyecto de partida. Se han cumplido los objetivos por los que se empezaba el desarrollo de esta práctica.

Se espera que el alumno la valore y le sea muy útil, ahorrándole quebraderos de cabeza cuando se incorpore a una empresa que lleve a cabo un desarrollo ágil.

# Apéndice A
# Guía para el profesor

## A.1 Ficheros para la inicialización de los datos

Lo primero que se debe hacer es instalar el servidor *Gitlab* en el equipo que se vaya a usar, para ello se debe ejecutar los siguientes comandos.

```
$ sudo apt-get update
$ sudo apt-get install -y curl openssh-server ca-certificates
```

Posteriormente se añade el repositorio de *Gitlab* a las fuentes del sistema, para ello se debe descargar mediante el comando 'curl' el script necesario y ejecutar dicho script.

```
$ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/
    script.deb.sh | sudo bash
```

Se instala el servicio de *Gitlab*, indicando la URL pública del servidor en la variable de entorno 'EXTERNAL_URL', esta dirección será usada solo para mostrar la información dentro del servidor, por ejemplo, para mostrar la URL desde donde clonar un repositorio.

```
$ sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ce
```

Ya se tendría *Gitlab* escuchando en el puerto 80, donde se puede navegar hasta la aplicación para crear usuarios, almacenar repositorios, etc.

El próximo paso necesario es la instalación y configuración de un ejecutor del servidor *Gitlab Runner*. Para ello, se debe descargar la versión que se desee, en este caso v11.4.2 y la versión para el sistema operativo que se esté usando y arquitectura, en este caso es Linux AMD64, con el siguiente comando.

```
$ curl -LJO https://s3.amazonaws.com/gitlab-runner-downloads/v11.4.2/binaries
    /gitlab-runner-linux-amd64
```

El fichero descargado debe ser ejecutado, para ello se puede usar el siguiente comando y *Gitlab Runner* se instalará.

```
$ dpkg -i gitlab-runner_linux-amd64.deb
```

Para la creación de un ejecutor válido para el entorno propuesto se debe de instalar *Docker* en el sistema ya que será usado como base para la ejecución de las tareas automatizadas, para ello, se actualizan e instalan los paquetes necesarios para el funcionamiento correcto de *Docker*.

```
$ sudo apt-get update
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

Se debe añadir la clave GPG al equipo para poder instalar *Docker*.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
    -
```

Y añadir el repositorio al sistema.

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Ya estaría el equipo configurado para poder descargar e instalar *Docker* y sus paquetes dependientes. Con el siguiente comando ya estaría instalado *Docker*.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Para finalizar, se ejecuta el siguiente comando para comprobar que *Docker* funciona correctamente.

```
$ sudo docker run hello-world
```

También es necesario para la ejecución de las pruebas automatizadas en el entorno propuesto la instalación del servidor *Tomcat* en el sistema, para ello se debe descargar el paquete *Tomcat* de su web oficial, para ello se debe acceder a la siguiente dirección *http://apache.uvigo.es/tomcat/tomcat-8/v8.5.53/bin/apache-tomcat-8.5.53.tar.gz*. Posteriormente se crea una carpeta para *Tomcat* en '/opt/tomcat' y se descomprime el fichero anteriormente descargado en ella con los siguientes comandos.

```
$ sudo mkdir /opt/tomcat
$ sudo tar xzvf apache-tomcat-8*tar.gz -C /opt/tomcat --strip-components=1
```

Por último, se configura el inicio automático del servicio cuando el equipo inicia, con el comando 'chk-config', este comando sirve para configurar la administración de servicios con 'SystemV' de forma sencilla.

```
$ chkconfig --level 2345 \opt\tomcat\bin\startup.sh on
```

Ya se encuentra instalado en el sistema todo lo necesario para la realización de la práctica, el siguiente paso sería ejecutar el script de inicialización de datos para tener el usuario con permisos de administrador creado y poder configurar el ejecutor compartido para la ejecución de tareas automatizadas.

```
$ ./inicializar_datos
```

A continuación se ha de configurar el ejecutor para poder realizar las tareas automatizadas de los distintos repositorios. Para ello, se inicia el proceso de configuración como se ve a continuación. Cuando solicite un token, se debe incluir 'dit', este es un valor que se ha configurado por defecto para no tener que configurar un ejecutor cada vez que se reinician los datos en la aplicación.

```
$ sudo gitlab-runner register

Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com )
$ {La URL de tu servidor}

Please enter the gitlab-ci token for this runner
$ dit

Please enter the gitlab-ci description for this runner
$ [hostname] my-runner

Please enter the gitlab-ci tags for this runner (comma separated):
$ my-tag,another-tag

Please enter the executor: ssh, docker+machine, docker-ssh+machine,
    kubernetes, docker, parallels, virtualbox, docker-ssh, shell:
$ docker

Please enter the Docker image (eg. ruby:2.6):
$ maven:3.6-jdk-13
```

Y con esto el entorno estaría preparado para ser usado en la realización de la práctica.

## A.2 Inicialización de datos necesarios para la práctica

Cada cierto tiempo se debe restaurar el estado inicial de los datos existentes en el servidor *Gitlab*, para ello se han desarrollado una serie de scripts, unidos en uno con el que es posible inicializar todos los datos necesarios con el siguiente comando.

```
$ ./inicializar_datos
```

## A.3 Modificar datos de la aplicación

Se puede acceder a la consola de *Rails* con el siguiente comando (no se recomienda si no tienes conocimientos sobre *Ruby on Rails*).

```
$ sudo gitla-rails console
```

Una vez se ha accedido a la consola se pueden observar todos los datos en la aplicación, a continuación, se indica una serie de comandos que pueden ser útiles.

- Encontrar un usuario por email o nombre de usuario.

```
    User.find(email: "user@email.com")

    User.find(username: "user")
```

- Editar un usuario.

```
    user = User.find(email: "user@email.com") \\Encontrar usuario que se
        quiera editar

    user.update!(username: "New username")
```

- Cambiar la contraseña de un usuario.

```
user = User.find(email: "user@email.com") \\Encontrar usuario al que
    se quiera cambiar la contraseña

user.reset_password(password, password)
```

- Borrar un usuario.

```
user = User.find(email: "user@email.com") \\Encontrar usuario que se
    quiera editar

user.destroy!
```

- Encontrar un proyecto por su nombre.

```
Project.find(name: "Project Name")
```

## A.4 Cambiar datos del usuario admin

Al ejecutar la *Rake Task* de inicialización de datos se crea un usuario con permisos de administrador, para cambiar datos de este usuario, se debe de modificar el archivo *create.rake*, la línea 13.

```
admin = User.create!(name: "Dit", email: "admin@dit.com", password:"D!t12345"
    , username: "dit", admin: true)
```

Dónde se puede encontrar y modificar el nombre de usuario, correo electrónico, contraseña, nombre de usuario, etc.

## A.5 Cambiar información sobre los distintos proyectos

Se ha desarrollado una *Rake Task* que inicializa todos los datos necesarios para un proyecto, para ello se le debe pasar como argumento el número del grupo. Para modificar los datos de los distintos proyectos, se pueden encontrar ciertas líneas interesantes, estas son:

- Modificar datos sobre el proyecto (línea 48 del fichero *create.rake*). Se pueden modificar datos tales cómo el usuario creador del proyecto, el nombre del proyecto, el path que se debe insertar en la url para acceder a él, etc.

```
repositorio = Projects::CreateService.new(admin, name: "ProyectoG#{
    n_grupo}", path: "proyecto#{n_grupo}", skip_disk_validation: true).
    execute
```

- Editar información sobre los usuarios para los miembros de las parejas que realicen el rol de desarrollador (línea 63 del fichero *create.rake*). Se pueden modificar datos tales cómo el nombre del usuario, correo electrónico, contraseña, nombre de usuario, etc.

```
dev = Users::CreateService.new(
    admin,
    name: "Developer#{n_grupo}-#{i}",
    email: "dev#{n_grupo}-#{i}@dit.com",
    password:"developer#{n_grupo}-#{i}",
    username: "dev#{n_grupo}-#{i}"
  ).execute
```

- Editar información sobre los usuarios para los miembros de las parejas que realicen el rol de líder de equipo (línea 84 del fichero *create.rake*). Se pueden modificar datos tales cómo el nombre del usuario, correo electrónico, contraseña, nombre de usuario, etc.

```
lider = Users::CreateService.new(
        admin,
        name: "Lider#{n_grupo}-#{i}",
        email: "lider#{n_grupo}-#{i}@dit.com",
        password:"lider#{n_grupo}-#{i}",
        username: "lider#{n_grupo}-#{i}"
      ).execute
```

# Apéndice B
# Ficheros

Este código está disponible para su descarga en la siguiente URL: *https://gitlab.com/FJaviGom/tfg-practica-is*

## B.1 Modelos

### B.1.1 Libro.java

**Código B.1** Modelo de la entidad Libro.

```
package modelo;

import java.sql.*;
import java.util.ArrayList;

import reserva_libros.Constantes;

public class Libro {

  private Long isbn;

  private String titulo;

  private String autor;

  private String editorial;

  private Integer anio_pub;

  private String genero;

  private String observaciones;

  //Campo usado para ver el cliente que tiene reservado un libro
  private Integer dni;

  public Conexion obj_conn;


  /**
   * Constructor vaco
```

```
 */
public Libro(){
  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();
}

/**
 * Constructor con parmetro la clave primaria (ISBN)
 */
public Libro(Long isbn) {
  this.isbn = isbn;

  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();

}

/**
 * Constructor con todos los campos
 */

public Libro(Long isbn, String titulo, String autor, String editorial,
    Integer anio_pub, String genero,
    String observaciones) {
  this.isbn = isbn;
  this.titulo = titulo;
  this.autor = autor;
  this.editorial = editorial;
  this.anio_pub = anio_pub;
  this.genero = genero;
  this.observaciones = observaciones;

  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();
}

/**
 * Constructor con todos los campos y el dni del Cliente que tiene reservado
     el Libro
 */

public Libro(Long isbn, String titulo, String autor, String editorial,
    Integer anio_pub, String genero,
    String observaciones, Integer dni) {
  this.isbn = isbn;
  this.titulo = titulo;
  this.autor = autor;
  this.editorial = editorial;
  this.anio_pub = anio_pub;
  this.genero = genero;
  this.observaciones = observaciones;
  this.dni = dni;

  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();
}
```

```
/**
 * @return the isbn
 */
public Long getIsbn() {
  return isbn;
}

/**
 * @param isbn the isbn to set
 */
public void setIsbn(Long isbn) {
  this.isbn = isbn;
}

/**
 * @return the titulo
 */
public String getTitulo() {
  return titulo;
}

/**
 * @param titulo the titulo to set
 */
public void setTitulo(String titulo) {
  this.titulo = titulo;
}

/**
 * @return the autor
 */
public String getAutor() {
  return autor;
}

/**
 * @param autor the autor to set
 */
public void setAutor(String autor) {
  this.autor = autor;
}

/**
 * @return the editorial
 */
public String getEditorial() {
  return editorial;
}

/**
 * @param editorial the editorial to set
 */
public void setEditorial(String editorial) {
  this.editorial = editorial;
}

/**
```

```
     * @return the anio_pub
     */
    public Integer getAnio_pub() {
      return anio_pub;
    }

    /**
     * @param anio_pub the anio_pub to set
     */
    public void setAnio_pub(Integer anio_pub) {
      this.anio_pub = anio_pub;
    }

    /**
     * @return the genero
     */
    public String getGenero() {
      return genero;
    }

    /**
     * @param genero the genero to set
     */
    public void setGenero(String genero) {
      this.genero = genero;
    }

    /**
     * @return the observaciones
     */
    public String getObservaciones() {
      return observaciones;
    }

    /**
     * @param observaciones the observaciones to set
     */
    public void setObservaciones(String observaciones) {
      this.observaciones = observaciones;
    }

    /**
     * Lo usamos para obtener el cliente que tiene reservado un libro
     */
    /**
     * @return the dni
     */
    public Integer getDni() {
      return dni;
    }

    /**
     * @param dni the dni to set
     */
    public void setDni(Integer dni) {
      this.dni=dni;
    }
```

```
  /*************** Mtodos ******/

  /**
   *
   * @param
   * @return
   */

  public Integer insertar_Libro(){
    //Verificamos que estamos en una sesion, usuario administrador

//     Cliente_Bean cliente = new Cliente_Bean();
//     int resultado = -1;

//     if(cliente.getCorrecto() && cliente.getAdmin()) {

      //   int resultado = -1;
         //Nos conectamos a la base de datos

    int resultado = 0;

    try {

    String consultaSql = "INSERT INTO libros (isbn, titulo, autor, editorial,
        anio_pub, genero, observaciones) VALUES ('"+
                isbn+"', '"+
                titulo+"', '"+
                autor+"', '"+
                editorial+"', '"+
                anio_pub+"', '"+
                genero+"', '"+
                observaciones+"');";

    resultado = obj_conn.insertar(consultaSql);
    } catch (SQLException e) {
      obj_conn.cerrarConexion();
      e.printStackTrace();

      return -1;  //Se est intentando registrar una PK duplicada
        }
    obj_conn.cerrarConexion();
    return resultado;
    /*     try {
          Class.forName(Constantes.DB_DRIVER).newInstance();

          Connection conn = DriverManager.getConnection(Constantes.DB_URL,
              Constantes.DB_USER, Constantes.DB_PASS);
            Statement st = conn.createStatement();
            int resUpdate = st.executeUpdate( consultaSql );
            if (resUpdate > 0 ) {
              resultado = 0;
            } else {
              resultado = 1;
            }
          st.close();
```

```
            conn.close();

        } catch (Exception e) {
          resultado = 1;
          e.printStackTrace();
        }
//          out.println(resultado);
        //}
       return resultado; */
      }

  public ArrayList<Libro> listar_Libros(){
    ArrayList<Libro> libros = new ArrayList<Libro>();

/*    String consultaSql = "SELECT isbn, titulo, autor, editorial, anio_pub,
    genero, observaciones" +
        " FROM libros;";
*/

    String consultaSql = "SELECT l.isbn, l.titulo, l.autor, l.editorial, l.
        anio_pub, l.genero, l.observaciones, r.dni" +
        " FROM libros l LEFT JOIN reservas r ON l.isbn = r.isbn;";

    libros = obj_conn.listar_Libros(consultaSql);

    obj_conn.cerrarConexion();

    return libros;
  }

}
```

### B.1.2 Reserva.java

**Código B.2** Modelo de la entidad Reserva.

```
/**
 *
 */
package modelo;

import java.sql.*;
import java.util.ArrayList;

/**
 * @author Fabian
 *
 */
public class Reserva {
  // Campos de la clase Reserva referentes a la clase Libro
  private Long isbn;

  private String titulo;

  private String autor;
```

```
  private String editorial;

  private Integer anio_pub;

  private String genero;

  private String observaciones;

  // Campos de la clase Reserva referentes a la clase Cliente
  private Integer dni;

  Conexion obj_conn;


  /**
   * Constructor vaco
   */
  public Reserva(){
    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();
  }

  /**
   * Constructor con parmetro la clave primaria (ISBN)
   */
  public Reserva(Long isbn) {
    this.isbn = isbn;

    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();

  }

  /**
   * Constructor con los campos isbn y dni
   */
  public Reserva(Long isbn, Integer dni){
    this.isbn = isbn;
    this.dni = dni;

    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();
  }

  public Reserva(Long isbn, String titulo, String autor, String editorial,
      Integer anio_pub, String genero,
      String observaciones, Integer dni) {

/*
    Libro libro = new Libro(isbn, titulo, autor, editorial, anio_pub, genero,
        observaciones);
    Cliente cliente = new Cliente(dni);
    this.isbn = libro.getIsbn();
    this.titulo = libro.getTitulo();
    this.autor = libro.getAutor();
    this.editorial = libro.getEditorial();
```

```
    this.anio_pub = libro.getAnio_pub();
    this.genero = libro.getGenero();
    this.observaciones = libro.getObservaciones();
    this.dni = cliente.getDni();
*/

    this.isbn = isbn;
    this.titulo = titulo;
    this.autor = autor;
    this.editorial = editorial;
    this.anio_pub = anio_pub;
    this.genero = genero;
    this.observaciones = observaciones;
    this.dni=dni;


    //Instanciamos un objeto de la clase conexin
//    obj_conn = new Conexion();
  }

  /**
   * Constructor con todos los campos
   * @param libro
   * @param cliente
   */
  public Reserva(Libro libro, Cliente cliente){
    this.isbn = libro.getIsbn();
    this.titulo = libro.getTitulo();
    this.autor = libro.getAutor();
    this.editorial = libro.getEditorial();
    this.anio_pub = libro.getAnio_pub();
    this.genero = libro.getGenero();
    this.observaciones = libro.getObservaciones();
    this.dni = cliente.getDni();

    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();

  }


  // Setter y Getter
  /**
   * @return the isbn
   */
  public Long getIsbn() {
    return isbn;
  }

  /**
   * @param isbn the isbn to set
   */
  public void setIsbn(Long isbn) {
    this.isbn = isbn;
  }

  /**
```

```
 * @return the titulo
 */
public String getTitulo() {
  return titulo;
}

/**
 * @param titulo the titulo to set
 */
public void setTitulo(String titulo) {
  this.titulo = titulo;
}

/**
 * @return the autor
 */
public String getAutor() {
  return autor;
}

/**
 * @param autor the autor to set
 */
public void setAutor(String autor) {
  this.autor = autor;
}

/**
 * @return the editorial
 */
public String getEditorial() {
  return editorial;
}

/**
 * @param editorial the editorial to set
 */
public void setEditorial(String editorial) {
  this.editorial = editorial;
}

/**
 * @return the anio_pub
 */
public Integer getAnio_pub() {
  return anio_pub;
}

/**
 * @param anio_pub the anio_pub to set
 */
public void setAnio_pub(Integer anio_pub) {
  this.anio_pub = anio_pub;
}

/**
 * @return the genero
```

```
 */
public String getGenero() {
  return genero;
}

/**
 * @param genero the genero to set
 */
public void setGenero(String genero) {
  this.genero = genero;
}

/**
 * @return the observaciones
 */
public String getObservaciones() {
  return observaciones;
}

/**
 * @param observaciones the observaciones to set
 */
public void setObservaciones(String observaciones) {
  this.observaciones = observaciones;
}

/**
 * @return the dni
 */
public Integer getDni() {
  return dni;
}

/**
 * @param dni the dni to set
 */
public void setDni(Integer dni) {
  this.dni=dni;
}


/*************** Mtodos ******/

public Integer reservar_Libro(){
  int resultado = 0;

  try {

  String consultaSql = "INSERT INTO reservas (isbn, dni) VALUES ('"+
        isbn+"', '"+
        dni+"');";

  resultado = obj_conn.insertar(consultaSql);
  } catch (SQLException e) {
    obj_conn.cerrarConexion();
    return -1;  //Se est intentando registrar una PK duplicada
        //e.printStackTrace();
```

```
      }
  obj_conn.cerrarConexion();
  return resultado;
    }

public Integer borrar_Reserva(Long n_isbn){
  int resultado = 0;

  try {

  String consultaSql = "DELETE FROM reservas"+
      " WHERE isbn="+ n_isbn;

  obj_conn.borrar(consultaSql);
  resultado = 1;
  } catch (SQLException e) {
    obj_conn.cerrarConexion();
    return 0;  //Se est intentando registrar una PK duplicada
       //e.printStackTrace();
      }
  obj_conn.cerrarConexion();
  return resultado;
  }

public ArrayList<Reserva> listar_Reservas(){
  ArrayList<Reserva> reservas = new ArrayList<Reserva>();

  String consultaSql = "SELECT libros.*, reservas.dni" +
      "  FROM libros, reservas WHERE libros.isbn = reservas.isbn;";

  /*
       String sql="SELECT libros.*,reservas.dni FROM libros, reservas" +
           " WHERE libros.isbn = reservas.isbn;";
  */

  reservas = obj_conn.listar_Reservas(consultaSql);

  obj_conn.cerrarConexion();

  return reservas;
}

public ArrayList<Reserva> listar_Reservas_Cliente(Integer dni){
  ArrayList<Reserva> reservas = new ArrayList<Reserva>();
  System.out.println("DNI_cliente:"+dni);

  String consultaSql = "SELECT libros.*, reservas.dni" +
      "  FROM libros, reservas WHERE libros.isbn = reservas.isbn AND reservas.
          dni = "+ dni + ";";

  /*
       String sql="SELECT libros.*,reservas.dni FROM libros, reservas" +
           " WHERE libros.isbn = reservas.isbn;";
  */

  reservas = obj_conn.listar_Reservas(consultaSql);
```

```
    obj_conn.cerrarConexion();

    return reservas;
  }

}
```

### B.1.3 Cliente.java

**Código B.3** Modelo de la entidad Cliente.

```
package modelo;

import java.sql.*;
import java.util.*;

import reserva_libros.Constantes;

import modelo.Conexion;

/**
 * Clase que almacena los datos de un usuario.
 * Pensado para usarse en el ámbito de sesión.
 * Además es capaz de comprobar si el usuario/clave existe
 * en la base de datos. El uso habitual sería:
 *  - Inicialmente, se crea un objeto en la sesión.
 *  - Se da valores a las propiedades usuario y clave.
 *  - Se llama a getCorrecto(). Esta función buscará en la
 *    base de datos si el usuario/clave existe. Si lo encuentra
 *    rellena las propiedades nombre y administrador con la información
 *    almacenada en la base de datos y establece correcto a verdadero.
 *    Una vez encontrado que el usuario es correcto, ya no se vuelve
 *    a comprobar más.
 */
public class Cliente {

  /**
   * ID del usuario. Propiedad de solo lectura (solo get).
   * El valor se obtiene de la base de datos cuando se llama
   * a getCorrecto().
   */
//  private int idc;

  /**
   * Nombre completo real del usuario. Propiedad de solo lectura (get).
   * El valor se obtiene de la base de datos cuando se llama
   * a getCorrecto().
   */
//  private Integer dni;
  protected Integer dni;
  /**
   * Login del usuario. El utilizado en el formulario de
   * entrada. Propiedad de lectura/escritura (get/set).
   */
/*
```

```
  private String cliente;

  private String nombre;

  private String apellido1;

  private String apellido2;

  private String direccion;
*/

  protected String cliente;

  protected String nombre;

  protected String apellido1;

  protected String apellido2;

  protected String direccion;

  /**
   * Indica si este usuario es un administrador. Propiedad de solo lectura (get
       ).
   * El valor se obtiene de la base de datos cuando se llama
   * a getCorrecto(). Será falso si es un usuario normal.
   */
// private Boolean admin;
  protected Boolean admin;
  /**
   * Clave del usuario. El utilizado en el formulario de
   * entrada. Propiedad de lectura/escritura (get/set).
   */
// private String contrasena;
  protected String contrasena;

  Conexion obj_conn;


  /**
   *
   * Indica si este usuario es correcto. Propiedad de solo lectura (get).
   * Si es falso, el usuario no existe y no está autorizado.
   */
// private boolean correcto;
  protected boolean correcto;

  /**
   * Crea un usuario y da valores iniciales.
   */
  public Cliente() {
    correcto=false;
    admin=false;

    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();
  }
```

```
public Cliente(Integer dni) {
  this.dni = dni;

  correcto=false;
  admin=false;

  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();
}

public Cliente(Integer dni, String cliente, String nombre, String apellido1,
    String apellido2, String direccion,
     Boolean admin, String contrasena) {
  this.dni = dni;
  this.cliente = cliente;
  this.nombre = nombre;
  this.apellido1 = apellido1;
  this.apellido2 = apellido2;
  this.direccion = direccion;
  this.admin = admin;
  this.contrasena = contrasena;

  correcto=false;
  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();

}

public Cliente(Integer dni, String nombre, String apellido1, String apellido2
    , String direccion,
    Boolean admin) {
  this.dni = dni;
  this.nombre = nombre;
  this.apellido1 = apellido1;
  this.apellido2 = apellido2;
  this.direccion = direccion;
  this.admin = admin;

  correcto=false;
  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();

}

public String getCliente() {
  return cliente;
}
public void setCliente(String cliente) {
  this.cliente = cliente;
}
public Integer getDni() {
  return dni;
}
public void setDni(Integer dni) {
  this.dni=dni;
}
```

```
  public String getNombre() {
    return nombre;
  }
  public void setNombre(String nombre) {
    this.nombre=nombre;
  }
  public String getApellido1() {
    return apellido1;
  }
  public void setApellido1(String apellido1) {
    this.apellido1=apellido1;
  }
  public String getApellido2() {
    return apellido2;
  }
  public void setApellido2(String apellido2) {
    this.apellido2=apellido2;
  }
  public String getDireccion() {
    return direccion;
  }
  public void setDireccion(String direccion) {
    this.direccion=direccion;
  }
  public String getContrasena() {
    return contrasena;
  }
  public void setContrasena(String contrasena) {
    this.contrasena = contrasena;
  }
/*  public int getIdc() {
    return idc;
  } */
  public Boolean getAdmin() {
    return admin;
  }

  public void setAdmin(Boolean admin) {
    this.admin = admin;
  }


  /**
   * Determina si el cliente es un usuario vlido.
   * Mientras la propiedad "correcto" sea falsa, cada vez que se llame
   * a este mtodo se conectar a la base de datos, buscará si existe
   * un usuario con los campos cliente y contrasea iguales a los de las
   * propiedades cliente y contrasea. Si lo encuentra, rellena el resto de
   * propiedades del bean y establece correcto a True.
   * Si se produce algn error/excepcin o no se encuentra el cliente,
   * la propiedad correcto no se modificar y seguir valiendo False.
   * Si la variable correcto es True, no se hace nada y solo se devuelve True.
   * @return True si el cliente est en la base de datos.
   */
  public boolean getCorrecto() {
    if (!correcto) {
      try {
```

```
      // Verificamos en la base de datos
      Class.forName(Constantes.DB_DRIVER).newInstance();
      Connection conn = DriverManager.getConnection(Constantes.DB_URL,
          Constantes.DB_USER, Constantes.DB_PASS);

      Statement st = conn.createStatement();
      String consulta = "SELECT dni, nombre, apellido1, apellido2, direccion,
           admin" +
          " FROM clientes WHERE cliente='" + cliente +
          "' AND contrasena='" + contrasena + "';";
      ResultSet rs = st.executeQuery(consulta);
      if (rs.next()) {
        dni = rs.getInt("dni");
        nombre = rs.getString("nombre");
        apellido1 = rs.getString("apellido1");
        apellido2 = rs.getString("apellido2");
        direccion = rs.getString("direccion");
        admin = rs.getBoolean("admin");
        //Si hemos encontrado al menos una lnea, es correcto
        correcto=true;
      }
      // Liberamos los recursos
      rs.close();
      st.close();
      conn.close();

    } catch (SQLException e) {
      //Usuario continua siendo inválido
      e.printStackTrace();
    } catch (Exception e) {
      //Usuario continua siendo inválido
      //Driver no encontrado
      e.printStackTrace();
    }
  }

  return correcto;
}

public Integer login (String c, String p){
  int resultado = 0;

  try {

    String consultaSql = "SELECT dni, nombre, apellido1, apellido2, direccion,
         admin" +
        " FROM clientes WHERE cliente='" + c +
        "' AND contrasena='" + p + "';";

  resultado = obj_conn.consultar(consultaSql);
  return resultado;
  } catch (SQLException e) {
    obj_conn.cerrarConexion();
        //e.printStackTrace();
  }
  return 0;
}
```

```
  public Integer insertar_Cliente(){
    //Verificamos que estamos en una sesion, usuario administrador

//      Cliente_Bean cliente = new Cliente_Bean();
//      int resultado = -1;

//      if(cliente.getCorrecto() && cliente.getAdmin()) {

      //    int resultado = -1;
          //Nos conectamos a la base de datos

    int resultado = 0;

    try {

    byte admin_byte = (byte)(admin?1:0);

    String consultaSql = "INSERT INTO clientes (dni, cliente, nombre, apellido1
        , apellido2, direccion, admin, contrasena) VALUES ('"+
                dni+"', '"+
                cliente+"', '"+
                nombre+"', '"+
                apellido1+"', '"+
                apellido2+"', '"+
                direccion+"', '"+
                admin_byte+"', '"+
                contrasena+"');";

    resultado = obj_conn.insertar(consultaSql);
    } catch (SQLException e) {
      obj_conn.cerrarConexion();
      return -1;  //Se est intentando registrar una PK duplicada
          //e.printStackTrace();
        }
    obj_conn.cerrarConexion();
    return resultado;
  }

  public ArrayList<Cliente> listar_Clientes(){
    ArrayList<Cliente> clientes = new ArrayList<Cliente>();

    String consultaSql = "SELECT dni, cliente, nombre, apellido1, apellido2,
        direccion, admin, contrasena" +
        " FROM clientes;";

    clientes = obj_conn.listar_Clientes(consultaSql);
    obj_conn.cerrarConexion();

    return clientes;
  }
}
```

### B.1.4 NuevoCliente.java

**Código B.4** Modelo de la entidad NuevoCliente.

```
/**
 *
 */
package modelo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;

import reserva_libros.Constantes;

/**
 * @author Fabian
 *
 */
public class NuevoCliente { //extends Cliente {

/*
  public NuevoCliente() {
    super();
  }

  public NuevoCliente(Integer dni, String cliente, String nombre, String
      apellido1, String apellido2, String direccion,
       Boolean admin, String contrasena) {
    super(dni, cliente, nombre, apellido1, apellido2, direccion, admin,
        contrasena);
  }

  public NuevoCliente(Integer dni, String nombre, String apellido1, String
      apellido2, String direccion,
       Boolean admin) {
    super(dni, nombre, apellido1, apellido2, direccion, admin);
  }
*/

  /**
   * ID del usuario. Propiedad de solo lectura (solo get).
   * El valor se obtiene de la base de datos cuando se llama
   * a getCorrecto().
   */
// private int idc;

  /**
   * Nombre completo real del usuario. Propiedad de solo lectura (get).
   * El valor se obtiene de la base de datos cuando se llama
   * a getCorrecto().
   */
  private Integer dni;

  /**
   * Login del usuario. El utilizado en el formulario de
```

```
 * entrada. Propiedad de lectura/escritura (get/set).
 */

private String cliente;

private String nombre;

private String apellido1;

private String apellido2;

private String direccion;


/**
 * Indica si este usuario es un administrador. Propiedad de solo lectura (get
    ).
 * El valor se obtiene de la base de datos cuando se llama
 * a getCorrecto(). Será falso si es un usuario normal.
 */
private Boolean admin;

private byte admin_byte;

/**
 * Clave del usuario. El utilizado en el formulario de
 * entrada. Propiedad de lectura/escritura (get/set).
 */
private String contrasena;

Conexion obj_conn;


/**
 *
 * Indica si este usuario es correcto. Propiedad de solo lectura (get).
 * Si es falso, el usuario no existe y no está autorizado.
 */
private boolean correcto;


/**
 * Crea un usuario y da valores iniciales.
 */
public NuevoCliente() {
  correcto=false;
  admin=false;

  //Instanciamos un objeto de la clase conexin
  obj_conn = new Conexion();
}

public NuevoCliente(Integer dni, String cliente, String nombre, String
    apellido1, String apellido2, String direccion,
     Boolean admin, String contrasena) {
  this.dni = dni;
  this.cliente = cliente;
```

```
    this.nombre = nombre;
    this.apellido1 = apellido1;
    this.apellido2 = apellido2;
    this.direccion = direccion;
    this.admin = admin;
    this.contrasena = contrasena;

    correcto=false;
    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();

  }

  public NuevoCliente(Integer dni, String nombre, String apellido1, String
      apellido2, String direccion,
      Boolean admin) {
    this.dni = dni;
    this.nombre = nombre;
    this.apellido1 = apellido1;
    this.apellido2 = apellido2;
    this.direccion = direccion;
    this.admin = admin;

    correcto=false;
    //Instanciamos un objeto de la clase conexin
    obj_conn = new Conexion();

  }

  public String getCliente() {
    return cliente;
  }
  public void setCliente(String cliente) {
    this.cliente = cliente;
  }
  public Integer getDni() {
    return dni;
  }
  public void setDni(Integer dni) {
    this.dni=dni;
  }
  public String getNombre() {
    return nombre;
  }
  public void setNombre(String nombre) {
    this.nombre=nombre;
  }
  public String getApellido1() {
    return apellido1;
  }
  public void setApellido1(String apellido1) {
    this.apellido1=apellido1;
  }
  public String getApellido2() {
    return apellido2;
  }
  public void setApellido2(String apellido2) {
```

```
    this.apellido2=apellido2;
  }
  public String getDireccion() {
    return direccion;
  }
  public void setDireccion(String direccion) {
    this.direccion=direccion;
  }
  public String getContrasena() {
    return contrasena;
  }
  public void setContrasena(String contrasena) {
    this.contrasena = contrasena;
  }
/*  public int getIdc() {
    return idc;
  } */
  public Boolean getAdmin() {
    return admin;
  }

  public void setAdmin(Boolean admin) {
    this.admin = admin;
  }
  public byte getAdmin_byte() {
    return admin_byte;
  }

  public void setAdmin_byte(byte admin_byte) {
    this.admin_byte = admin_byte;
  }

  /**
   * Determina si el cliente es un usuario vlido.
   * Mientras la propiedad "correcto" sea falsa, cada vez que se llame
   * a este mtodo se conectar a la base de datos, buscará si existe
   * un usuario con los campos cliente y contrasea iguales a los de las
   * propiedades cliente y contrasea. Si lo encuentra, rellena el resto de
   * propiedades del bean y establece correcto a True.
   * Si se produce algn error/excepcin o no se encuentra el cliente,
   * la propiedad correcto no se modificar y seguir valiendo False.
   * Si la variable correcto es True, no se hace nada y solo se devuelve True.
   * @return True si el cliente est en la base de datos.
   */
  public boolean getCorrecto() {
    if (!correcto) {
      try {
        // Verificamos en la base de datos
        Class.forName(Constantes.DB_DRIVER).newInstance();
        Connection conn = DriverManager.getConnection(Constantes.DB_URL,
            Constantes.DB_USER, Constantes.DB_PASS);

        Statement st = conn.createStatement();
        String consulta = "SELECT dni, nombre, apellido1, apellido2, direccion,
             admin" +
            " FROM clientes WHERE cliente='" + cliente +
            "' AND contrasena='" + contrasena + "';";
```

```
        ResultSet rs = st.executeQuery(consulta);
        if (rs.next()) {
          dni = rs.getInt("dni");
          nombre = rs.getString("nombre");
          apellido1 = rs.getString("apellido1");
          apellido2 = rs.getString("apellido2");
          direccion = rs.getString("direccion");
          admin = rs.getBoolean("admin");
          //Si hemos encontrado al menos una lnea, es correcto
          correcto=true;
        }
        // Liberamos los recursos
        rs.close();
        st.close();
        conn.close();

      } catch (SQLException e) {
        //Usuario continua siendo inválido
        e.printStackTrace();
      } catch (Exception e) {
        //Usuario continua siendo inválido
        //Driver no encontrado
        e.printStackTrace();
      }
    }

    return correcto;
  }

  public Integer login (String c, String p){
    int resultado = 0;

    try {

      String consultaSql = "SELECT dni, nombre, apellido1, apellido2, direccion,
           admin" +
          " FROM clientes WHERE cliente='" + c +
          "' AND contrasena='" + p + "';";

    resultado = obj_conn.consultar(consultaSql);
    return resultado;
    } catch (SQLException e) {
      obj_conn.cerrarConexion();
          //e.printStackTrace();
    }
    return 0;
  }

  public Integer insertar_Cliente(){
    //Verificamos que estamos en una sesion, usuario administrador

//      Cliente_Bean cliente = new Cliente_Bean();
//      int resultado = -1;

//      if(cliente.getCorrecto() && cliente.getAdmin()) {

      //    int resultado = -1;
```

```
        //Nos conectamos a la base de datos

    int resultado = 0;

    try {

    byte admin_byte = (byte)(admin?1:0);

    String consultaSql = "INSERT INTO clientes (dni, cliente, nombre, apellido1
        , apellido2, direccion, admin, contrasena) VALUES ('"+
                dni+"', '"+
                cliente+"', '"+
                nombre+"', '"+
                apellido1+"', '"+
                apellido2+"', '"+
                direccion+"', '"+
                admin_byte+"', '"+
                contrasena+"');";

    resultado = obj_conn.insertar(consultaSql);
    } catch (SQLException e) {
      obj_conn.cerrarConexion();
      return -1;  //Se est intentando registrar una PK duplicada
          //e.printStackTrace();
        }
    obj_conn.cerrarConexion();
    return resultado;
  }

  public ArrayList<Cliente> listar_Clientes(){
    ArrayList<Cliente> clientes = new ArrayList<Cliente>();

    String consultaSql = "SELECT dni, cliente, nombre, apellido1, apellido2,
        direccion, admin, contrasena" +
        " FROM clientes;";

    clientes = obj_conn.listar_Clientes(consultaSql);
    obj_conn.cerrarConexion();

    return clientes;
  }
}
```

### B.1.5 Conexion.java

**Código B.5** Modelo de la entidad Conexión.

```
package modelo;

import java.sql.*;
import java.util.*;

import reserva_libros.Constantes;

public class Conexion {
```

```
 //private Connection conn = null;
 Connection conn; //Cargar el driver
 Statement st; //Conectarse a la BD
 ResultSet rs; //Procesar consultas sql
 public String mensajeError = "";

 public Conexion() {
  try {
    Class.forName(Constantes.DB_DRIVER).newInstance();
//    System.out.println("Driver JDBC cargado con xito");

  /*  Connection conn = DriverManager.getConnection(Constantes.DB_URL,
        Constantes.DB_USER, Constantes.DB_PASS); */

  } catch (ClassNotFoundException | InstantiationException |
      IllegalAccessException e ) {
   e.printStackTrace();
   mensajeError = "Error al cargar el Driver JDBC";
   System.err.println(mensajeError);
//  mensajeError = "Imposible acceder a la base de datos.";
  }
  try {
  conn = DriverManager.getConnection(Constantes.DB_URL,
       Constantes.DB_USER, Constantes.DB_PASS);
//  System.out.println("Conexin a la Base de Datos realizada con xito");
} catch (SQLException e) {
  mensajeError = "Imposible acceder a la base de datos.";
  System.err.println(mensajeError);
  e.printStackTrace();
  }
 }

 public Connection getConexion(){
  return conn;
 }

 public void cerrarConexion(){
  try {
   conn.close();
   conn = null;
  } catch (SQLException e) {
   e.printStackTrace();
  }
 }

 public Integer insertar(String sql) throws SQLException{
   st = conn.createStatement();
   return st.executeUpdate(sql);
 }

 public Integer consultar(String sql) throws SQLException{
   st = conn.createStatement();
   rs = st.executeQuery(sql);

   if(rs.next()) {
     return 1;
```

```
  }
  else{
    return 0;
  }
}

public Integer borrar(String sql) throws SQLException{
  st = conn.createStatement();
  return st.executeUpdate(sql);
}

 public ArrayList<Cliente> listar_Clientes(String sql){
   ArrayList<Cliente> clientes = new ArrayList<Cliente>();

   try {

     st = conn.createStatement();
     rs = st.executeQuery(sql);

     while(rs.next()){
       clientes.add(new Cliente(rs.getInt("dni"), rs.getString("cliente"),rs.
           getString("nombre"), rs.getString("apellido1"), rs.getString("
           apellido2"), rs.getString("direccion"), rs.getBoolean("admin"), rs.
           getString("contrasena")));
     }
   } catch (SQLException e) {
     e.printStackTrace();
   }

   return clientes;
 }

 public ArrayList<Libro> listar_Libros(String sql){
   ArrayList<Libro> libros = new ArrayList<Libro>();

   try {

     st = conn.createStatement();
     rs = st.executeQuery(sql);

     while(rs.next()){
       libros.add(new Libro(rs.getLong("isbn"), rs.getString("titulo"), rs.
           getString("autor"), rs.getString("editorial"), rs.getInt("anio_pub")
           , rs.getString("genero"), rs.getString("observaciones"), rs.getInt("
           dni")));
     }
   } catch (SQLException e) {
     e.printStackTrace();
   }

   return libros;
 }

 public ArrayList<Reserva> listar_Reservas(String sql){
   ArrayList<Reserva> reservas = new ArrayList<Reserva>();

   try {
```

```
      st = conn.createStatement();
      rs = st.executeQuery(sql);

      while(rs.next()){
        reservas.add(new Reserva(rs.getLong("isbn"), rs.getString("titulo"), rs.
            getString("autor"), rs.getString("editorial"), rs.getInt("anio_pub")
            , rs.getString("genero"), rs.getString("observaciones"), rs.getInt("
            dni")));
        }
      } catch (SQLException e) {
        e.printStackTrace();
      }

    return reservas;
    }
}
```

## B.2 Controladores

### B.2.1 Listar_Reservas_Cliente.java

**Código B.6** Controlador encargado de listar las reservas de los clientes.

```
package controlador;

import java.io.IOException;
import java.util.ArrayList;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import modelo.Cliente;
import modelo.Libro;
import modelo.Reserva;

/**
 * Servlet implementation class Listar_Reservas
 */
@WebServlet(description = "Servlet controlador encargado de gestionar el
    listado de las reservas de todos los clientes", urlPatterns = { "/
    listado_reservas_cliente" })
public class Listar_Reservas_Cliente extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Listar_Reservas_Cliente() {
        super();
    }
```

```
  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    Reserva reserva = new Reserva();

    //Obtener la instacia del objeto Cliente creado en la autenticacin del
        cliente en login.jsp a travs del Bean Cliente
    Cliente cliente = (Cliente) request.getSession().getAttribute("cliente");
    Integer dni = cliente.getDni();

    ArrayList<Reserva> reservas = reserva.listar_Reservas_Cliente(dni);

    request.getSession().setAttribute("reservas", reservas);

    request.getRequestDispatcher("reservas.jsp").forward(request, response);

  }

}
```

### B.2.2 Reservar_Libro.java

**Código B.7** Controlador encargado de reservar.

```
package controlador;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import modelo.*;

/**
 * Servlet implementation class Reservar_Libro
 */
@WebServlet(name = "reservar_libro", urlPatterns = { "/reservar_libro" })
public class Reservar_Libro extends HttpServlet {
  private static final long serialVersionUID = 1L;
```

```
    /**
     * @see HttpServlet#HttpServlet()
     */
    public Reservar_Libro() {
        super();
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
/*
    // Obtener dni de usuario con la sesion activa para reservas
    HttpSession sesion_activa = request.getSession();
    Integer dni = (Integer)sesion_activa.getAttribute("dni_cliente_activo");
*/

    Long isbn = Long.parseLong(request.getParameter("isbn"));
//    Integer dni = Integer.parseInt(request.getParameter("dni"));

    //Obtener la instancia del objeto Cliente creado en la autenticacion del
        cliente en login.jsp a traves del Bean Cliente
    Cliente cliente = (Cliente) request.getSession().getAttribute("cliente");
    Integer dni = cliente.getDni();

    Reserva reserva_libro = new Reserva(isbn,dni);

    reserva_libro.reservar_Libro();

    request.getRequestDispatcher("reserva_libro_correcto.jsp").forward(request,
        response);
  }

}
```

### B.2.3 Insert_cliente.java

**Código B.8** Controlador encargado de crear un nuevo cliente.

```
package controlador;

import java.io.IOException;
import java.util.logging.Level;
```

```
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.Cliente;
import modelo.NuevoCliente;

/**
 * Servlet implementation class Insert_cliente
 */
@WebServlet("/insercion_cliente")
public class Insert_cliente extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Insert_cliente() {
        super();
        // TODO Auto-generated constructor stub
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    Integer dni = null;
    String cliente = null;
    String nombre = null;
    String apellido1 = null;
    String apellido2 = null;
    String direccion = null;
    String contrasena = null;

    boolean control = true;

    boolean b_dni = true;
    boolean b_cliente = true;
    boolean b_nombre = true;
    boolean b_apellido1 = true;
    boolean b_apellido2 = true;
    boolean b_direccion = true;
```

```
    boolean b_contrasena = true;

    // Comprobacion de que el campo dni no sea vacio
    if(request.getParameter("dni") == null || request.getParameter("dni")==""){
      Logger.getLogger(getClass().getName()).log(
                Level.WARNING, "ERROR EN EL SERVIDOR: El campo dni no puede
                    estar vacio");

      b_dni = false;
    }
    else {
           // Comprobacion de que isbn solo esta formado por caracteres
               numericos
        try {
          dni = Integer.parseInt(request.getParameter("dni"));
        } catch (Exception e) {
        Logger.getLogger(getClass().getName()).log(
                    Level.WARNING, "ERROR EN EL SERVIDOR: El campo isbn slo puede
                         estar formado por caracteres numricos");

        b_dni = false;
        }
    }

        if(b_dni == true) {
        // Convertimos el numero a un String y comparamos (mediante el metodo
            length()) el
        // numero de digitos que debe tener dicho numero)
        String dni_string = Integer.toString(dni);

        if (dni_string.length() != 8){
          Logger.getLogger(getClass().getName()).log(
                      Level.WARNING, "ERROR EN EL SERVIDOR: El campo dni tiene
                          que tener 8 caracteres");

                control = false;
//              request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
             }
        }

    // Comprobacion de que el campo cliente no sea vaco
    if(request.getParameter("cliente") == null || request.getParameter("cliente
        ")==""){
      Logger.getLogger(getClass().getName()).log(
                Level.WARNING, "ERROR EN EL SERVIDOR: El campo cliente no puede
                    estar vacio");

      b_cliente = false;
    }
    else {
      cliente = request.getParameter("cliente");
    }
    // Comprobacion de que el campo cliente tenga entre 5 y 12 caracteres
    if(b_cliente == true){
      if(cliente.length() < 5 || cliente.length() > 12){
           Logger.getLogger(getClass().getName()).log(
```

```
                        Level.WARNING, "ERROR EN EL SERVIDOR: El campo cliente
                            tiene que estar entre 5 y 12 caracteres");

            control = false;
//            request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }


    // Comprobacion de que el campo nombre no sea vaco
    if(request.getParameter("nombre") == null || request.getParameter("nombre")
        ==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo nombre no puede
                   estar vacio");

      b_nombre = false;
    }
    else {
      nombre = request.getParameter("nombre");
    }
    // Comprobacion de que el campo nombre tenga menos de 20 caracteres
    if(b_nombre == true){
      if(nombre.length() > 20){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo nombre
                           tiene que tener menos de 20 caracteres");

           control = false;
//           request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }


    // Comprobacion de que el campo apellido1 no sea vaco
    if(request.getParameter("apellido1") == null || request.getParameter("
        apellido1")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo apellido1 no
                   puede estar vacio");

      b_apellido1 = false;
    }
    else {
      apellido1 = request.getParameter("apellido1");
    }
    // Comprobacion de que el campo apellido1 tenga menos de 20 caracteres
    if(b_apellido1 == true){
      if(apellido1.length() > 20){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo apellido1
                            tiene que tener menos de 20 caracteres");

           control = false;
```

```
//          request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }


    // Comprobacion de que el campo apellido2 no sea vaco
    if(request.getParameter("apellido2") == null || request.getParameter("
        apellido2")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo apellido2 no
                   puede estar vacio");

      b_apellido2 = false;
    }
    else {
      apellido2 = request.getParameter("apellido2");
    }
    // Comprobacion de que el campo apellido2 tenga menos de 20 caracteres
    if(b_apellido2 == true){
      if(apellido2.length() > 20){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo apellido2
                            tiene que tener menos de 20 caracteres");

           control = false;
//           request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }


    // Comprobacion de que el campo direccion no sea vaco
    if(request.getParameter("direccion") == null || request.getParameter("
        direccion")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo direccion no
                   puede estar vacio");

      b_direccion = false;
    }
    else {
      direccion = request.getParameter("direccion");
    }
    // Comprobacion de que el campo apellido2 tenga menos de 40 caracteres
    if(b_direccion == true){
      if(direccion.length() > 40){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo direccion
                            tiene que tener menos de 20 caracteres");

           control = false;
//           request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }
```

```
//    Integer dni = Integer.parseInt(request.getParameter("dni"));
//    String cliente = request.getParameter("cliente");
//    String nombre = request.getParameter("nombre");
//    String apellido1 = request.getParameter("apellido1");
//    String apellido2 = request.getParameter("apellido2");
//    String direccion = request.getParameter("direccion");
  //  boolean admin = Boolean.valueOf(request.getParameter("admin"));
    Boolean admin = Boolean.parseBoolean(request.getParameter("admin"));
    String admin_string = request.getParameter("admin");
    try{
      if(admin_string.compareTo("on")==0){
        admin=false;
      }
      else{
        admin=true;
      }
    }catch(java.lang.NullPointerException e){};

  //  Integer admin = Integer.parseInt(request.getParameter("admin"));
  //  byte admin = Byte.valueOf(request.getParameter("admin"));
  //  String admin = request.getParameter("admin");


    // Comprobacion de que el campo contrasea no sea vaco
    if(request.getParameter("contrasena") == null || request.getParameter("
        contrasena")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo contrasena no
                   puede estar vacio");

      b_contrasena = false;
    }
    else {
      contrasena = request.getParameter("contrasena");
    }
    // Comprobacion de que el campo contrasena tenga entre 5 y 12 caracteres
    if(b_contrasena == true){
      if(contrasena.length() < 5 || contrasena.length() > 12){
           Logger.getLogger(getClass().getName()).log(
                      Level.WARNING, "ERROR EN EL SERVIDOR: El campo
                          contrasena tiene que estar entre 5 y 12 caracteres");


           control = false;
//           request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }


//    String contrasena = request.getParameter("contrasena");

//    byte admin_byte = (byte)(admin?1:0);


      if (control==true){
```

```
        Cliente cliente_insert = new Cliente(dni, cliente, nombre, apellido1,
            apellido2,
            direccion, admin, contrasena);

        if (cliente_insert.insertar_Cliente() > 0){
          request.getRequestDispatcher("ins_cliente_correcto.jsp").forward(
              request, response);
        }
        else {
          request.getRequestDispatcher("ins_cliente_error.jsp").forward(request,
               response);
        }
      }
      else { //if(control==false)
        request.getRequestDispatcher("ins_cliente_error.jsp").forward(request,
            response);
      }
  }
}
```

### B.2.4 Listar_Clientes.java

**Código B.9** Controlador encargado de listar los clientes.

```
package controlador;

import java.io.IOException;
import java.util.*;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.Cliente;

/**
 * Servlet implementation class Listar_Clientes
 */
@WebServlet("/listar_clientes")
public class Listar_Clientes extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Listar_Clientes() {
        super();
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    Cliente cliente = new Cliente();

    ArrayList<Cliente> clientes = cliente.listar_Clientes();

    request.getSession().setAttribute("clientes", clientes);

    request.getRequestDispatcher("listar_clientes.jsp").forward(request,
        response);

  }

}
```

### B.2.5 Borrar_Reserva.java

**Código B.10** Controlador encargado de boorrar reservas.

```
package controlador;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.*;

/**
 * Servlet implementation class Borrar_Reserva
 */
@WebServlet("/borrar_reserva")
public class Borrar_Reserva extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Borrar_Reserva() {
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
```

```
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    Long isbn = Long.parseLong(request.getParameter("isbn"));

  // String dni = request.getParameter("dni");

    Reserva reserva_borrar = new Reserva(isbn);

    if (reserva_borrar.borrar_Reserva(isbn) > 0){
      request.getRequestDispatcher("borrar_reserva_correcto.jsp").forward(
          request, response);
    }
    else {
      request.getRequestDispatcher("borrar_reserva_error.jsp").forward(request,
           response);
    }

  }

}
```

### B.2.6 Login.java

**Código B.11** Controlador encargado del inicio de sesión de los usuarios.

```
package controlador;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import modelo.Cliente;

/**
 * Servlet implementation class Login
 */
@WebServlet("/login")
public class Login extends HttpServlet {
```

```
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Login() {

    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
//    String cliente = request.getParameter("cliente");
//    String contrasena = request.getParameter("contrasena");

    String cliente = null;
    String contrasena = null;

    boolean control = true;

    boolean b_cliente = true;
    boolean b_contrasena = true;

    // Comprobacion de que el campo cliente no sea vaco
    if(request.getParameter("cliente") == null || request.getParameter("cliente
        ")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo cliente no puede
                   estar vacio");

      b_cliente = false;
    }
    else {
      cliente = request.getParameter("cliente");
    }
    // Comprobacion de que el campo cliente tenga entre 5 y 12 caracteres
    if(b_cliente == true){
      if(cliente.length() < 5 || cliente.length() > 12){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo cliente
                           tiene que estar entre 5 y 12 caracteres");

           control = false;
//           request.getRequestDispatcher("inicio_tras_error.jsp").forward(
    request, response);
```

```
      }
    }

    // Comprobacion de que el campo contrasea no sea vaco
    if(request.getParameter("contrasena") == null || request.getParameter("
        contrasena")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo contrasena no
                   puede estar vacio");

      b_contrasena = false;
    }
    else {
      contrasena = request.getParameter("contrasena");
    }
    // Comprobacion de que el campo contrasena tenga entre 5 y 12 caracteres
    if(b_contrasena == true){
      if(contrasena.length() < 5 || contrasena.length() > 12){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo
                           contrasena tiene que estar entre 5 y 12 caracteres");


           control = false;
//           request.getRequestDispatcher("ins_cliente_error.jsp").forward(
    request, response);
      }
    }

      if (control==true){
        Cliente cliente_login = new Cliente();
    //podriamos haber creado el constructor con cliente y contrasena y nos
        quitamos el meterlos asi

/*
    // Creado para poder enviar el dni del cliente que quiere realizar la
        reserva
    Integer dni_cliente_activo = cliente_login.getDni();
    HttpSession sesion = request.getSession();
    sesion.setAttribute("dni_cliente_activo", dni_cliente_activo);
*/
        if(cliente_login.login(cliente, contrasena) > 0){
          request.getRequestDispatcher("datos.jsp").forward(request, response);

/*      //Para obtener dni de usuario con la cuenta activa para reservas
        HttpSession sesion_activa = request.getSession();
      sesion_activa.setAttribute("dni_cliente_activo", cliente_login.getDni());
*/
      }
        else {
    //  alert("Error en el usuario o contrasea. Por favor, introdzcalos
        correctamente.");
          request.getRequestDispatcher("inicio_tras_error.jsp").forward(request,
               response);
        }
      }
      else { //if(control==false)
```

```
        request.getRequestDispatcher("inicio_tras_error.jsp").forward(request,
            response);
      }
  }
}
```

### B.2.7 Listar_Reservas.java

**Código B.12** Controlador encargado de listar todas las reservas.

```
package controlador;

import java.io.IOException;
import java.util.ArrayList;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.Libro;
import modelo.Reserva;

/**
 * Servlet implementation class Listar_Reservas
 */
//@WebServlet(description = "Servlet controlador encargado de gestionar el
    listado de las reservas de todos los clientes", urlPatterns = { "/
    listado_reservas" })
public class Listar_Reservas extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Listar_Reservas() {
        super();
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
```

```
    Reserva reserva = new Reserva();

    ArrayList<Reserva> reservas = reserva.listar_Reservas();

    request.getSession().setAttribute("reservas", reservas);

    request.getRequestDispatcher("reservas.jsp").forward(request, response);

  }

}
```

### B.2.8 Listar_Libros.java

**Código B.13** Controlador encargado de listar los libros.

```
package controlador;

import java.io.IOException;
import java.util.*;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.Libro;
import modelo.Reserva;

/**
 * Servlet implementation class Listar_Libros
 */
//@WebServlet(description = "Servlet controlador encargado de gestionar el
    listado de todos los libros", urlPatterns = { "/listado_libros" })
public class Listar_Libros extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Listar_Libros() {
        super();
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response
      ) throws ServletException, IOException {
    doPost(request, response);
  }

  /**
```

```
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
       response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    Libro libro = new Libro();

    ArrayList<Libro> libros = libro.listar_Libros();

    Integer num_libros= libros.size();

    Reserva reserva = new Reserva();

    ArrayList<Reserva> reservas = reserva.listar_Reservas();

    request.getSession().setAttribute("libros", libros);
    request.getSession().setAttribute("num_libros", num_libros);
    request.getSession().setAttribute("reservas", reservas);


    request.getRequestDispatcher("listar_libros.jsp").forward(request, response
        );

  }

}
```

### B.2.9 Insert_libro.java

**Código B.14** Controlador encargado de crear libros.

```
package controlador;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

import modelo.Libro;

/**
 * Servlet implementation class Insert_libro
 */
@WebServlet(description = "Servlet controlador encargado de gestionar la
    insercion de un libro", urlPatterns = { "/insercion_libro" })
public class Insert_libro extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
```

```
    */
  public Insert_libro() {
      super();
      // TODO Auto-generated constructor stub
  }

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response
    ) throws ServletException, IOException {
  doPost(request, response);
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
     response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
  Long isbn = null;
  String titulo = null;
  String autor = null;
  String editorial = null;
  Integer anio_pub = null;
  String genero = null;
  String observaciones = null;

  boolean control = true;

  boolean b_isbn = true;
  boolean b_titulo = true;
  boolean b_autor = true;
  boolean b_editorial = true;
  boolean b_anio_pub = true;
  boolean b_genero = true;

  // Comprobacion de que el campo isbn no sea vaco
  if(request.getParameter("isbn") == null || request.getParameter("isbn")==""
      ){
    Logger.getLogger(getClass().getName()).log(
             Level.WARNING, "ERROR EN EL SERVIDOR: El campo isbn no puede
                 estar vacio");

    b_isbn = false;
  }
  else {
    // Comprobacion de que isbn slo est formado por caracteres numricos
      try {
        isbn = Long.parseLong(request.getParameter("isbn"));
      } catch (Exception e) {
      Logger.getLogger(getClass().getName()).log(
                 Level.WARNING, "ERROR EN EL SERVIDOR: El campo isbn slo puede
                      estar formado por caracteres numricos");

      b_isbn = false;
```

```
        }
    }

        if(b_isbn == true) {
        // Convertimos el numero a un String y comparamos (mediante el metodo
            length()) el
        // numero de digitos que debe tener dicho numero)
        String isbn_string = Long.toString(isbn);

        if (isbn_string.length() != 10){
          Logger.getLogger(getClass().getName()).log(
                      Level.WARNING, "ERROR EN EL SERVIDOR: El campo isbn tiene
                          que tener 10 caracteres");

              control = false;
//             request.getRequestDispatcher("ins_libro_error.jsp").forward(
    request, response);
          }
        }


    // Comprobacion de que el campo titulo no sea vaco
    if(request.getParameter("titulo") == null || request.getParameter("titulo")
        ==""){
      Logger.getLogger(getClass().getName()).log(
                Level.WARNING, "ERROR EN EL SERVIDOR: El campo titulo no puede
                    estar vacio");

      b_titulo = false;
    }
    else {
      titulo = request.getParameter("titulo");
    }
    // Comprobacion de que el campo titulo tenga menos de 50 caracteres
    if(b_titulo == true){
      if(titulo.length() > 50){
           Logger.getLogger(getClass().getName()).log(
                       Level.WARNING, "ERROR EN EL SERVIDOR: El campo titulo
                           tiene que tener menos de 50 caracteres");

           control = false;
//           request.getRequestDispatcher("ins_libro_error.jsp").forward(
    request, response);
      }
    }

    // Comprobacion de que el campo autor no sea vaco
    if(request.getParameter("autor") == null || request.getParameter("autor")==
        ""){
      Logger.getLogger(getClass().getName()).log(
                Level.WARNING, "ERROR EN EL SERVIDOR: El campo autor no puede
                    estar vacio");

      b_autor = false;
    }
    else {
      autor = request.getParameter("autor");
```

```
    }
    // Comprobacion de que el campo autor tenga menos de 50 caracteres
    if(b_autor == true){
      if(autor.length() > 50){
           Logger.getLogger(getClass().getName()).log(
                      Level.WARNING, "ERROR EN EL SERVIDOR: El campo autor
                          tiene que tener menos de 50 caracteres");

           control = false;
//           request.getRequestDispatcher("ins_libro_error.jsp").forward(
    request, response);
      }
    }

    // Comprobacion de que el campo editorial no sea vaco
    if(request.getParameter("editorial") == null || request.getParameter("
        editorial")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo editorial no
                   puede estar vacio");

      b_editorial = false;
    }
    else {
      editorial = request.getParameter("editorial");
    }
    // Comprobacion de que el campo editorial tenga menos de 50 caracteres
    if(b_editorial == true){
      if(editorial.length() > 50){
           Logger.getLogger(getClass().getName()).log(
                      Level.WARNING, "ERROR EN EL SERVIDOR: El campo editorial
                           tiene que tener menos de 50 caracteres");

           control = false;
//           request.getRequestDispatcher("ins_libro_error.jsp").forward(
    request, response);
      }
    }

    // Comprobacion de que el campo anio_pub no sea vaco
    if(request.getParameter("anio_pub") == null || request.getParameter("
        anio_pub")==""){
      Logger.getLogger(getClass().getName()).log(
               Level.WARNING, "ERROR EN EL SERVIDOR: El campo anio_pub no puede
                    estar vacio");

      b_anio_pub = false;
    }
    else {
      // Comprobacion de que anio_pub slo est formado por caracteres numricos
        try {
        anio_pub = Integer.parseInt(request.getParameter("anio_pub"));
        } catch (Exception e) {
        Logger.getLogger(getClass().getName()).log(
                   Level.WARNING, "ERROR EN EL SERVIDOR: El campo anio_pub slo
                       puede estar formado por caracteres numricos");
```

```
        b_anio_pub = false;
        }
    }

      if(b_anio_pub == true) {
      // Convertimos el numero a un String y comparamos (mediante el metodo
          length()) el
        // numero de digitos que debe tener dicho numero
      String anio_pub_string = Long.toString(anio_pub);

      if (anio_pub_string.length() > 4){
        Logger.getLogger(getClass().getName()).log(
            Level.WARNING, "ERROR EN EL SERVIDOR: El campo anio_pub no puede
                tener ms de 4 caracteres");

          control = false;
//          request.getRequestDispatcher("ins_libro_error.jsp").forward(request,
     response);
          }
      }

    // Comprobacion de que el campo genero no sea vaco (de que se haya
        seleccionado un gnero)
    if(request.getParameter("genero") == null || request.getParameter("genero")
         == "" || request.getParameter("genero").equals("0"))/*Integer.valueOf(
        request.getParameter("genero")) == 0*/{
      Logger.getLogger(getClass().getName()).log(
                Level.WARNING, "ERROR EN EL SERVIDOR: El campo genero no puede
                    estar vacio. Es obligatorio elegir un gnero de la lista.");

      b_genero = false;
    }
    else {
      genero = request.getParameter("genero");
    }

  // Long isbn = Long.parseLong(request.getParameter("isbn"));
  // String titulo = request.getParameter("titulo");
  // String autor = request.getParameter("autor");
  // String editorial = request.getParameter("editorial");
  // Integer anio_pub = Integer.parseInt(request.getParameter("anio_pub"));
  // String genero = request.getParameter("genero");
  // String observaciones = request.getParameter("observaciones");
  // boolean control = true;

    observaciones = request.getParameter("observaciones");


        if (control==true){
          Libro libro_insert = new Libro(isbn, titulo, autor, editorial,
              anio_pub, genero,
        observaciones);

          if (libro_insert.insertar_Libro() > 0){
      request.getRequestDispatcher("ins_libro_correcto.jsp").forward(request,
          response);
          }
```

```
          else {
            request.getRequestDispatcher("ins_libro_error.jsp").forward(request,
                response);
          }
        }
        else {
        request.getRequestDispatcher("ins_libro_error.jsp").forward(request,
            response);

        }
  }
}
```

## B.3 Fichero de constantes

### B.3.1 Constantes.java

**Código B.15** Constantes necesarias.

```
package reserva_libros;

public class Constantes {
  private static String DB_ADDR = System.getenv("DB_ADDR") != null ? System.
      getenv("DB_ADDR") : "postgres";
  private static String DB_PORT = System.getenv("DB_PORT") != null ? System.
      getenv("DB_PORT") : "5432";

  public static final String DB_DRIVER = "org.postgresql.Driver";
  public static final String DB_URL = "jdbc:postgresql://" + DB_ADDR + ":" +
      DB_PORT + "/reserva_libros";
  public static final String DB_USER = "dit";
  public static final String DB_PASS = "dit";
}
```

## B.4 Ficheros de la interfaz

### B.4.1 menu.jsp

**Código B.16** Fichero JSP con el menú de la aplicación.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>

<%-- Crea el menu --%>

<%-- Solo ADMINISTRADORES --%>
<jsp:useBean id="cliente" class="modelo.Cliente" scope="session" />
<% if (cliente.getAdmin()) { %>
  <p class='boton'><a href="nuevo_libro.jsp">Insertar libro</a></p>
  <p class='boton'><a href="nuevo_cliente.jsp">Insertar cliente</a></p>
  <p class='boton'><a href="listado_reservas">Reservas</a></p>
```

```
  <p class='boton'><a href="listar_clientes">Listado de clientes</a></p> <%--
      Llama primero al controlador para que se cargue la lista a mostrar antes
      de mostrarla --%>
<% } %>

<%-- Solo CLIENTES --%>
<% if (!cliente.getAdmin()) { %>
  <p class='boton'><a href="listado_reservas_cliente">Mis reservas</a></p>
<% } %>

<%-- TODOS --%>
<p class='boton'><a href="listado_libros">Listado de libros</a></p>
<p class='boton'><a href="datos.jsp">Datos</a></p>
<p class='boton'><a href="logout.jsp">Salir</a></p>
```

### B.4.2 reservar.jsp

**Código B.17** Fichero para realizar una reserva.

```
<%@ page language="java" pageEncoding="UTF-8"
  import="java.sql.*, reserva_libros.Constantes"
  trimDirectiveWhitespaces="true" %>

<jsp:useBean id="cliente" class="modelo.Cliente" scope="session" />

<%
/*
Si el usuario es correcto la respuesta será:
  0  = exito
  1  = error
En caso contrario, no devolverá nada.
*/

//Verificamos que estamos en una sesion
if(cliente.getCorrecto()) {

    int resultado = -1;
    //Nos conectamos a la base de datos
    String consultaSql = "INSERT INTO reservas (isbn, dni) VALUES ('"+
        request.getParameter("isbn")+"', '"+
        cliente.getDni()+"');";
    try {
    Class.forName(Constantes.DB_DRIVER).newInstance();

    Connection conn = DriverManager.getConnection(Constantes.DB_URL,
        Constantes.DB_USER, Constantes.DB_PASS);
      Statement st = conn.createStatement();
      int resUpdate = st.executeUpdate( consultaSql );
      if (resUpdate > 0 ) {
        resultado = 0;
      } else {
        resultado = 1;
      }
    st.close();
      conn.close();
```

```
  } catch (Exception e) {
    resultado = 1;
    e.printStackTrace();
  }
    out.println(resultado);
}
%>
```

### B.4.3 ins_cliente_error.jsp

**Código B.18** Fichero JSP para mostrar un error cuando la creación de un cliente falla.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Insercin Cliente</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
            />
      </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Error al insertar el nuevo cliente </h1>
  </div>

</body>
</html>
```

### B.4.4 reservas.jsp

**Código B.19** Vista para comprobar todas las reservas.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>

<html>

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
    <title>Reservas</title>
  </head>

  <body>
    <div id="cabecera">
    <div id="logotipo">
        <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
             />
        </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1>Reservas</h1>
    <form action="borrar_reserva" method="post">
      <div id="lista-div">

        <table id="lista-tabla">
          <thead>
            <tr>
              <th>ISBN</th>
              <th>Ttulo</th>
              <th>Autor</th>
              <th>Editorial</th>
              <th>Ao de publicacin</th>
              <th>Gnero</th>
              <th>Observaciones</th>
              <th>DNI_cliente</th>
            </tr>
          </thead>
          <c:forEach var="listado_reservas" items="${sessionScope.reservas}">
            <tr>
              <td>${listado_reservas.getIsbn()}</td>
              <td>${listado_reservas.getTitulo()}</td>
              <td>${listado_reservas.getAutor()}</td>
              <td>${listado_reservas.getEditorial()}</td>
              <td>${listado_reservas.getAnio_pub()}</td>
              <td>${listado_reservas.getGenero()}</td>
              <td>${listado_reservas.getObservaciones()}</td>
              <td>${listado_reservas.getDni()}</td>
              <td>
```

```
              <a href="borrar_reserva?isbn=${listado_reservas.getIsbn()}" >
                  Borrar</a>
            </td>
          </tr>
        </c:forEach>
      </table>
    </div>
  </form>
  </div>
 </body>
</html>
```

### B.4.5 listar_libros.jsp

**Código B.20** Vista para listar todos los libros.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />
<title>Listado de libros</title>
</head>
<body>
    <div id="cabecera">
    <div id="logotipo">
        <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
            />
        </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1>Listado de libros</h1>
    <form action="reservar_libro" method="post">
    <div id="lista-div">

      <table id="lista-tabla">
        <thead>
          <tr>
            <th>ISBN</th>
            <th>Ttulo</th>
```

```
            <th>Autor</th>
            <th>Editorial</th>
            <th>Ao de publicacin</th>
            <th>Gnero</th>
            <th>Observaciones</th>
          </tr>
        </thead>
        <c:set var = "num_libros" scope = "session" value = "${sessionScope.num
            _libros}"/>
        <c:set var = "bandera" scope = "session" value = "${0}"/>
        <c:forEach var="listado_libros" items="${sessionScope.libros}">
          <tr>
            <td>${listado_libros.getIsbn()}</td>
            <td>${listado_libros.getTitulo()}</td>
            <td>${listado_libros.getAutor()}</td>
            <td>${listado_libros.getEditorial()}</td>
            <td>${listado_libros.getAnio_pub()}</td>
            <td>${listado_libros.getGenero()}</td>
            <td>${listado_libros.getObservaciones()}</td>
            <td>
              <!--
              Comprobacion de que el usuario es administrador
              Si es administrador no aparece la opcin de Reservar libros (el
                  administrador no tiene permitido reservar libros).
              En caso contrario s
              -->
              <c:if test="${cliente.getAdmin() == false}">

                <c:if test = "${bandera < num_libros}">
                    <c:choose>
                    <c:when test="${listado_libros.getDni() != 0}">
                      <c:set var="bandera" value="${bandera+1}"/>
                        Reservado
                      </c:when>
                      <c:otherwise>
                        <c:set var="bandera" value="${bandera+1}"/>
                        <a href="reservar_libro?isbn=${listado_libros.getIsbn()
                            }">Reservar</a>
                      </c:otherwise>
                    </c:choose>
                </c:if>
              </c:if>
            </td>
          </tr>
        </c:forEach>
      </table>
    </div>
  </form>
  </div>
</body>
</html>
```

### B.4.6 estilorp.css

**Código B.21** Fichero de estilos usados en la aplicación.

```
body {
  font-family:Arial, Helvetica, sans-serif;
}

.estiloAlternativo {
  font-family: monospace;
  background-color: lavender;
}

a {
  text-decoration: none;
}

a img {
  border: none;
}

.boton {
  background-color: #090963;
  color: white;
  text-transform: uppercase;
}

.boton:hover {
  background-color: #5a5;
}


#cabecera{
  border: thick solid #090963;

}

.acceso {
  display: block;
  float: right;
  color: white;
  text-align: right;
  font-size: 11px;
  padding: 10px 7px;
  font-weight: bold;
  text-transform: uppercase;
  margin: 2px;
}


#formacceso {
  border: 2px solid #090963;
  color: #090963;
  background-color: white;
  display: block;
  font-size: 11px;
  padding: 0px 5px;
  font-weight: bold;
  text-transform: uppercase;
  width: 60%;
```

```
  margin: auto;

}

#formacceso .boton {
  font-size: 10px;
  font-weight: bold;
  padding: 1px;
  margin: 0px;
}

#nombreusuario {
  background-color: white;
  color: #090963;
}

#pie {
    position: fixed;
    background-color: white;
    bottom: 0;
    margin: 0;
    height: 35px;
    width: 100%;
    border-top: 1px solid rgb(180, 180, 180);
}

#pie p {
    padding: 0px 20px;
    font-size: 0.85em;
    /*Debido a que el texto de la tarea se puede malinterpretar,
      se admite como font-size (0.85em, 0.15em, 15% y 85%)*/
    margin-top: 4px;
}

#bienvenida, #crear, #lista {
  clear: both;
  text-align: center;
  margin-left: auto;
  margin-right: auto;
}

#bienvenida {
  padding-top: 10%;
}

#crear, #lista{
  padding-top: 4%;
  width: 75%;
}


#datos li {
  list-style: square;
  padding: 30px;
  display: block;
  font-size: 1.5em;
  color: #090963;
```

```
}

.datovalor {
  font-weight: bold;
}

#formcrear {
  border: 2px solid #090963;
  color: #090963;
  background-color: white;
}

#formcrear form {
  padding-top: 4%;
  width: 75%;
  margin-left: auto;
  margin-right: auto;
}

#formcrear label {
  display: block;
  text-align: left;
  font-weight: bold;
  margin-top: 10px;
}

#formcrear input[type="text"] {
  height: 20px;
  width: 100%;
}

#formcrear .boton {
  font-weight: bold;
  margin: 10px;
  padding: 10px;
}

#formcrear #condiciones {
  height: 80px;
  width: 100%;
  resize: none;
}

#exito, #error {
  background-color: #5a5;
  color: white;
  font-size: 1.25em;
  text-align: center;
  margin-left: auto;
  margin-right: auto;
  font-weight: bold;
  width: 40%;
}

#error {
  background-color: #a55;
}
```

```
#lista td {
  border-top: 2px solid #a55;
  border-bottom: 2px solid #a55;
  text-align: left;
  padding-left: 20px;
}

#lista td:hover {
  background-color: #fec;
}

#lista table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 40px;
}

.detalle {
  display: block;
}

.textonota {
  border: 1px dotted #a55;
}

#menu, #contenido {
  margin-top: 30px;
  float: left;
}

#menu {

  clear: both;
  width: 20%;
  text-align: left;
  border: thin solid #090963;

}


#menu p {
  padding: 5px;
  margin: 10px;
  background-color: aliceblue;
}

#menu p a {
  color:  #090963;
  display: block;
}

#menu p:hover {
  background-color: #090963;
}
```

```
#menu p:hover a {
  color: white;
}

#contenido {
  width: 70%;
  padding-left: 50px;
}

#contenido h1 {
  border-bottom: thin solid #090963;
}

#lista-tabla, #lista-tabla tr, #lista-tabla td{
  border: 1px solid #090963;
  padding: 5px;
  margin: 0px;
  border-collapse: collapse;
}

#lista-tabla {
  width: 100%;
}

.bBorrar {
  background-color: #a55;
}

img {
  width: 600px;
    height: 100px;
}
```

### B.4.7 funcionesrp.js

**Código B.22** Fichero con más funciones usadas.

```
//Petición AJAX GET
//peticion - URL de la solicitud HTTP
//funcionRespuesta - función a llamar cuando termine la petición
//parametro - parámetro adicional que hay que enviar a la función para que
// esta sepa a qué petición se refiere
function ajax(peticion, funcionRespuesta, parametro) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.open("GET",peticion,true);
  xmlhttp.onreadystatechange = function(){
    if ( xmlhttp.readyState==4 ) {
      if (xmlhttp.status==200) {
        //Respuesta recibida completamente (4) y sin
        //errores del servidor (codigo HTTP 200)
        //Cambiamos página con la respuesta
        funcionRespuesta(xmlhttp, parametro);
      } else {
        funcionRespuesta(null, parametro);
      }
```

```
    }
    };

  xmlhttp.send(); //enviamos
}

//Petición AJAX POST
//peticion - URL de la solicitud HTTP
//datosPost - Parámetros a pasar mediante POST (tipo formulario)
//funcionRespuesta - función a llamar cuando termine la petición
//parametro - parámetro adicional que hay que enviar a la función para que
//esta sepa a qué petición se refiere
function ajaxPost(peticion, datosPost, funcionRespuesta, parametro) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.open("POST",peticion,true);
  xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
  xmlhttp.onreadystatechange = function(){
    if ( xmlhttp.readyState==4 ) {
      if (xmlhttp.status==200) {
        //Respuesta recibida completamente (4) y sin
        //errores del servidor (codigo HTTP 200)
        //Cambiamos página con la respuesta
        funcionRespuesta(xmlhttp, parametro);
      } else {
        funcionRespuesta(null, parametro);
      }

    }
    };

  xmlhttp.send(datosPost); //enviamos
}

//Función para acortar las llamadas a getElementById
function elem(id) {
  return document.getElementById(id);
}


function procesarBorrado(boton) {
  //Buscamos la fila "fila-IDL" y la eliminamos
  var fila = elem("fila-"+boton.value);
  if (fila != null ) {
    fila.parentNode.removeChild(fila);
  }
}

function procesarReserva(boton) {
  //Buscamos la celda "accion-IDL" y la ponemos no disponible
  var celda = elem("accion-"+boton.value);
  if (celda != null ) {
    celda.innerHTML="Reservado";
  }
}

function resultadoReserva(xmlHttp, parametro) {
```

```
  var texto = "";
  if (xmlHttp == null) {
    //Envío incorrecto
    texto="-2 ERROR: AJAX";
  } else {
    texto = xmlHttp.responseText.trim();
    if (texto == "-1") {
      texto +=" ERROR: NO IMPLEMENTADO";
    } else if (texto == "0") {
      texto +=" CORRECTO";
      if (parametro != null) {
        //Si era un botón borrar, llamamos a procesar borrado
        if (parametro.classList.contains('bBorrar')) {
          procesarBorrado(parametro);
        } else if (parametro.classList.contains('bReservar')){
          //Si era un botón reservar, llamados a procesar reserva
          procesarReserva(parametro);
        }
      }

    } else if (texto == "1") {
      texto +=" ERROR: SQL";
    } else {
      texto +=" ERROR: DESCONOCIDO";
    }
  }
  texto +="\nISBN= "+parametro.value;


  alert(texto);
}

//Valida el formulario de insertar
function procesaInsertar() {
  var isbn = elem("isbn").value;
  var titulo = elem("titulo").value;
  var autor = elem("autor").value;
  var editorial = elem("editorial").value;
  var anio_pub = elem("anio_pub").value;
  var genero = elem("genero").value;
  var observaciones = elem("observaciones").value;

  var resultado = true;

  // Comprobaciones inserción campos vacíos

  // Comprobación vacío isbn
  resultado = compruebaVacio(isbn,"ISBN");

  // Comprobación vacío titulo
  resultado = compruebaVacio(titulo,"Título");

  // Comprobación vacío autor
  resultado = compruebaVacio(autor,"Autor");

  // Comprobación vacío editorial
  resultado = compruebaVacio(editorial,"Editorial");
```

```
  // Comprobación vacío titulo
  resultado = compruebaVacio(anio_pub,"Año de publicación");

  // Comprobación vacío genero
  resultado = compruebaVacio(genero,"género");

  // Comprobación de selección de un género de la lista
  if(genero == "0" || genero == ''){
    genero = "";
    resultado = false;
    alert("Seleccione un género");
  }

  if (resultado) {
    var peticion = "insertar_libro.jsp";
    var datosPost="isbn="+encodeURIComponent( isbn ) +
    "&titulo="+encodeURIComponent( titulo ) +
    "&autor="+encodeURIComponent( autor ) +
    "&editorial="+encodeURIComponent( editorial ) +
    "&anio_pub="+encodeURIComponent( anio_pub ) +
    "&genero="+encodeURIComponent( genero ) +
    "&observaciones="+encodeURIComponent( observaciones );

    //Enviamos
    ajaxPost(peticion, datosPost,resultadoReserva,elem("isbn"));
  }

  // Siempre se devuelve false, para que no se envíe directamente,
  // ya que se hace por AJAX
  return false;
}

function enviarReserva(pagina, boton) {
  var isbn = boton.value;
  //Generamos la URL de la petición
  //IMPORTANTE, los campos se envían codificados en base64, para que los
  // caracteres no ascii no den problemas
  var peticion = pagina;
  var datosPost="isbn="+isbn;
  //Enviamos la petición. El resultado se informará a la función resultadoEnvio
  ajaxPost(peticion, datosPost, resultadoReserva, boton);

}

window.onload = function() {
  //  Tareas a realizar al principio.
  // Generar peticiones AJAX en cada boton
  //Botones bBorrar
  var listaBorrar = document.querySelectorAll(".bBorrar");
  for (var boton in listaBorrar) {
    listaBorrar[boton].onclick = function() {
      enviarReserva( "borrar.jsp", this );
      };
  }
  //Botones bReservar
  var listaReservar = document.querySelectorAll(".bReservar");
```

```
  for (var boton in listaReservar) {
    listaReservar[boton].onclick = function() {
      enviarReserva( "reservar.jsp", this );
      };
  }
  //formulario de inserción
  var formulario = elem("formcrear");
  if (formulario != null) {
    formulario.onsubmit = procesaInsertar;
  }
}
```

### B.4.8 compr_nuevo_libro.js

**Código B.23** Fichero para comprobar la creación de un libro.

```
//Valida el formulario de insertar

function compr_nuevo_libro(){

  var isbn = document.getElementById("isbn").value;
  var titulo = document.getElementById("titulo").value;
  var autor = document.getElementById("autor").value;
  var editorial = document.getElementById("editorial").value;
  var anio_pub = document.getElementById("anio_pub").value;
  var genero = document.getElementById("genero").value;
  var observaciones = document.getElementById("observaciones").value;
/*
  var fecha_actual = new Date();
  var anio_actual = fecha.getFullYear();
*/
  var resultado = true;

  /*****************************/
  /******* Comprobaciones *******/
  /*****************************/

  /****** Comprobaciones sobre campo isbn ******/
  /** Comprobamos que el campo ISBN no sea vacío */
  if(resultado == true){
    resultado = compruebaVacio(isbn,"ISBN");
    if (resultado == true){
      /** Si el campo ISBN no es vacío, comprobamos que el campo ISBN tenga 10
          caracteres numéricos */
      resultado = compruebaISBN(isbn,long_isbn,"ISBN");
      }
    }

  /****** Comprobaciones sobre campo titulo ******/
  if (resultado == true){
    /** Comprobamos que el campo Titulo no sea vacío */
    resultado = compruebaVacio(titulo,"Título");
    if (resultado == true){
```

```
      /** Comprobamos el tamaño máximo del título (para controlar los datos
          introducidos) */
      resultado = compruebaMax(titulo,max_titulo,"titulo");
      }
    }

  /****** Comprobación sobre campo autor ******/
  if (resultado == true){
    /** Comprobamos que el campo Autor no sea vacío */
    resultado = compruebaVacio(autor,"Autor");
    if (resultado == true){
      /** Comprobamos el tamaño máximo del título (para controlar los datos
          introducidos) */
      resultado = compruebaMax(autor,max_autor,"autor");
      }
    }

  /****** Comprobación sobre campo editorial ******/
  if (resultado == true){
    /** Comprobamos que el campo Editorial no sea vacío */
    resultado = compruebaVacio(editorial,"Editorial");
    if (resultado == true){
      /** Comprobamos el tamaño máximo del título (para controlar los datos
          introducidos) */
      resultado = compruebaMax(editorial,max_editorial,"editorial");
      }
    }

  /****** Comprobación sobre campo anio_pub ******/
  if (resultado == true){
    /** Comprobamos que el campo Año de publicación no sea vacío */
    resultado = compruebaVacio(anio_pub,"Año de publicación");
    if (resultado == true){
      /** Comprobamos el tamaño máximo del título (para controlar los datos
          introducidos) */
      resultado = compruebaAnioPub(anio_pub,max_digitos_anio,"Año de
          publicacion");
  /*    if (resultado == true){
        //Comprobamos que el año introducido es menor que el año actual
        resultado = compruebaAnioCorrecto(anio_pub);
        } */
      }
    }

  /****** Comprobación sobre campo genero ******/
  if (resultado == true){
    /** Comprobación de selección de un género de la lista */
    if(genero == "0" || genero == ''){
      genero = "";
      resultado = false;
      alert("Seleccione un género");
      }
    }

  // Comprobación vacío año de publicación
//  resultado = compruebaVacio(anio_pub,"Año de publicación");
```

```
  // Comprobación vacío genero
//  resultado = compruebaVacio(genero,"género");


  /*** Comprobación de isbn ***/
  // Comprobación de que ISBN es un número
  // Trataremos el ISBN como un código numérico de 10 dígitos (formato ISBN
      hasta 2007)
//  resultado = compruebaEsNumerico(isbn,long_isbn,"ISBN");
//  if (resultado == true) {
  // Comprobación del número de dígitos del ISBN
//    resultado = compruebaLongitud(isbn,min_isbn,max_isbn,"ISBN");
//  }

  /*** Comprobación de título ***/
  // Comprobamos el tamaño máximo del título (para controlar los datos
      introducidos)
//  resultado = compruebaMax(titulo,max_titulo,"titulo");

  /*** Comprobación de autor ***/
  // Comprobamos el tamaño máximo del autor (para controlar los datos
      introducidos)
//  resultado = compruebaMax(autor,max_autor,"autor");

  /*** Comprobación de editorial ***/
  // Comprobamos el tamaño máximo de la editorial (para controlar los datos
      introducidos)
//  resultado = compruebaMax(editorial,max_editorial,"editorial");

  /*** Comprobación de año de publicación ***/
  // Comprobamos que los caracteres introducidos son numéricos

//  resultado = compruebaEsNumerico(anio_pub);
/*  if (resultado == true) {
    // Comprobamos el tamaño máximo del año de publicación (para controlar los
        datos introducidos)
    resultado = compruebaMax(anio_pub,max_digitos_anio,"año de publicacion");
    if (resultado == true){
      // Comprobamos si el año introducido es menor que el año actual
      if(anio_pub > anio_actual){
        alert('[ERROR] El año de publicacion introducido es incorrecto. Por
            favor, introduzca un año correcto.');
      }
    }
  }

  // Comprobación de selección de un género de la lista
  if(genero == "0" || genero == ''){
    genero = "";
    resultado = false;
    alert("Seleccione un género");
  }
*/
  return resultado;
}
```

### B.4.9 funciones.js

**Código B.24** Fichero con funciones usadas en los demás ficheros JavaScripts.

```
/**
 *
 */

function compruebaVacio(param,texto) {
  var result = true;
  if (param == null || param == "" || param == '') {
    alert('[ERROR] El campo ' + texto + ' no puede estar vacio.');
    result = false;
  }
return result;
}

function compruebaLongitud(param,min_long,max_long,texto) {
  var result = true;
  if (param.length < min_long || param.length > max_long)
  {
    if(min_long == max_long){
      //Caso especial DNI (tiene que cumplir el tamaño de 8 caracteres(DNI sin
          letra))
      alert('[ERROR] El campo ' + texto + ' tiene que tener '+ min_long + '
          caracteres.');
    }
    else{
      // Resto de casos
      alert('[ERROR] El campo ' + texto + ' tiene que estar entre '+ min_long +
           ' y ' + max_long + ' caracteres.');
    }
  result = false;
  }
return result;
}

function compruebaMax(param,max_long,texto) {
  var result = true;
  if (param.length > max_long)
    {
      alert('[ERROR] El campo ' + texto + ' no puede tener mas de '+ max_long +
           ' caracteres.');

      result = false;
    }
  return result;
}

function compruebaISBN(param,tamanio,texto) {
  var result = true;
  if( !(/^[0-9]\d{9}$/.test(param))){
  //if (isNaN(param)==true) {//|| (/^[1-9]\d$/.test(param)==false) ) {
    alert('[ERROR] El campo ' + texto + ' tiene que tener '+ tamanio + '
        caracteres numericos.');
```

```
    result = false;
  }
  return result;
}

function compruebaAnioPub(param,tamanio,texto) {
  var result = true;
  if( !(/^[0-9]\d{3}$/.test(param))){
  //if (isNaN(param)==true) {//|| (/^[1-9]\d$/.test(param)==false) ) {
    alert('[ERROR] El campo ' + texto + ' tiene que tener '+ tamanio + '
        caracteres numericos.');

    result = false;
  }
  return result;
}

/*function compruebaAnioCorrecto(param){
  var result = true;
  var fecha_actual = new Date();
  var anio_actual = fecha.getFullYear();

  if (param > anio_actual){
    alert('[ERROR] El año de publicacion introducido es incorrecto. Por favor,
        introduzca un año correcto.');

    result = false;
  }
  return result;
} */
/*function compruebaEsNumerico(param,tamanio,texto) {
  var result = true;
//  if(!(/^[0-9]\d{9}$/.test(param))){
  if (isNaN(param)==true) {
    //|| (param.length != tamanio)) {|| (/^[1-9]\d$/.test(param)==false) ) {
    alert('[ERROR1] El campo ' + texto + ' tiene que tener '+ tamanio + '
        caracteres numéricos.');

    result = false;
  }
  else {
    if(param.length != 10){
      alert('[ERROR2] El campo ' + texto + ' tiene que tener '+ tamanio + '
          caracteres numéricos.');
    result = false;
    }
  }
  return result;
}*/
```

### B.4.10 global.js

**Código B.25** Fichero con variables usadas en los demás ficheros JavaScripts.

```
/**** Fichero usado para definir las variables globales al resto de ficheros js
     ****/

var min_cont = 5;
var max_cont = 12;
var min_client = 5;
var max_client = 12;

var min_dni = 8;
var max_dni = 8;
var max_nombre = 20;
var max_apellido = 20;
var max_direccion = 40;

// Variables relacionadas con Libro
var long_isbn = 10;

var max_titulo = 50;
var max_autor = 50;
var max_editorial = 50;

var max_digitos_anio = 4;
```

### B.4.11 compr_nuevo_cliente.js

**Código B.26** Archivo con la función JavaScript necesaria para la validación de la creación de un nuevo cliente.

```
//Comprueba el formulario de inserción de un nuevo cliente
function compruebaNuevoCliente() {
  var dni = document.getElementById("dni").value;
  var cliente = document.getElementById("cliente").value;
  var nombre = document.getElementById("nombre").value;
  var apellido1 = document.getElementById("apellido1").value;
  var apellido2 = document.getElementById("apellido2").value;
  var direccion = document.getElementById("direccion").value;
  var admin = document.getElementById("admin").value;
  var contrasena = document.getElementById("contrasena").value;
  var resultado = true;

  resultado = compruebaVacio(dni,"dni");
  if (resultado == true){
    resultado = compruebaLongitud(dni,min_dni,max_dni,"dni");
  }

  resultado = compruebaVacio(cliente,"cliente");
  if (resultado == true){
    resultado = compruebaLongitud(cliente,min_client,max_client,"cliente");
  }

  resultado = compruebaVacio(nombre,"nombre");
  if (resultado == true){
    resultado = compruebaMax(nombre,max_nombre,"nombre");
```

```
  }

  resultado = compruebaVacio(apellido1,"primer apellido");
  if (resultado == true){
    resultado = compruebaMax(apellido1,max_apellido,"primer apellido");
  }

  resultado = compruebaVacio(apellido2,"segundo apellido");
  if (resultado == true){
    resultado = compruebaMax(apellido2,max_apellido,"segundo apellido");
  }

  resultado = compruebaVacio(direccion,"direccion");
  if (resultado == true){
    resultado = compruebaMax(direccion,max_direccion,"direccion");
  }

  resultado = compruebaVacio(contrasena,"contraseña");
  if (resultado == true){
    resultado = compruebaLongitud(cliente,min_client,max_client,"contraseña");
  }

  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return resultado;
}
```

### B.4.12 jsinicio.js

**Código B.27** Archivo con funciones JavaScript necesarias para el inicio de sesión.

```
var nombreCookie = "estilo";

//Función para acortar las llamadas a getElementById
function elem(id) {
  return document.getElementById(id);
}


//Valida el formulario de login
function validacionLogin() {
  var cliente = document.getElementById("cliente").value;
  var contrasena = document.getElementById("contrasena").value;
  var resultado = true;
/*  var min_cont = 5;
  var max_cont = 12;
  var min_client = 5;
  var max_client = 12; */

  resultado = compruebaVacio(cliente,"cliente");
  if (resultado == true){
    resultado = compruebaLongitud(cliente,min_client,max_client,"cliente");
  }
```

```
  resultado = compruebaVacio(contrasena,"contraseña");
  if (resultado == true){
    resultado = compruebaLongitud(cliente,min_client,max_client,"contraseña");
  }

  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return resultado;
}


//Función que obtiene las cookies
function getCookie(cname) {
    var name = cname + "=";
    var ca = document.cookie.split(';');
    for(var i=0; i<ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0)==' ') {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0) {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}

//Crea, modifica
function setCookie(cname,cvalue,exdays) {
    var d = new Date();
    d.setTime(d.getTime() + (exdays*24*60*60*1000));
    var expires = "expires=" + d.toGMTString();
    document.cookie = cname+"="+cvalue+"; "+expires;
}

//Borra una cookie
function deleteCookie(name) {
  document.cookie = name + '=; expires=Thu, 01 Jan 1970 00:00:01 GMT;';
}

//Comprueba si existe la cookie, para el formulario
function checkCookie() {
    checkbox = elem("estiloAlt");
    if (checkbox != null) {
      var estilo=getCookie(nombreCookie);
      if (estilo != "") {
        checkbox.checked = true;
      } else {
        checkbox.checked = false;
      }
    }
}

window.onload = function() {
  //  Tareas a realizar al principio.
  checkCookie();
```

```
  elem("formacceso").onsubmit=validacionLogin;
}
```

### B.4.13 ins_libro_correcto.jsp

**Código B.28** Fichero para la creación de un nuevo libro.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Insercin Libro</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
      <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
          />
      </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Libro insertado correctamente </h1>
  </div>

</body>
</html>
```

### B.4.14 nuevo_cliente.jsp

**Código B.29** Formulario para la creación de un nuevo cliente.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.sql.*, reserva_libros.Constantes, modelo.
        Cliente" %>

<%-- Comprobacion de login correcto. Insertamos login.jsp --%>
<%@include file="login.jsp" %>
<%
if (request.isRequestedSessionIdValid() && !cliente.getAdmin()) {
  response.sendError(HttpServletResponse.SC_FORBIDDEN, "Debe ser administrador
      .");
```

```
}
 %>

<!DOCTYPE html>
<html>

  <head>
    <title>Nuevo cliente</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
    <!--
    <script src="js/global.js"></script>
    <script src="js/funciones.js"></script>
    <script src="js/compr_nuevo_cliente.js"></script>
    -->
  </head>

  <body>
    <div id="cabecera">
    <div id="logotipo">
        <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniería Telemá
            tica" />
        </a>
    </div>
  </div>



  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1>Insertar nuevo cliente</h1>
      <div id="crear">
    <div id="formcrear">
      <form action="insercion_cliente" method="post">
      <div class="dni-div">
        <label for="dni"><strong>DNI</strong></label>
        <input id="dni" type="text" value="" name="dni" maxlength="100"></input>

      </div>
      <div class="cliente-div">
        <label for="cliente"><strong>Cliente</strong></label>
        <input id="cliente" type="text" value="" name="cliente" maxlength
            ="100"></input>
      </div>
      <div class="nombre-div">
        <label for="nombre"><strong>Nombre</strong></label>
        <input id="nombre" type="text" value="" name="nombre" maxlength="100"></
            input>
      </div>
      <div class="apellido1-div">
        <label for="apellido1"><strong>Primer apellido</strong></label>
```

```
        <input id="apellido1" type="text" value="" name="apellido1" maxlength
            ="100"></input>
      </div>
      <div class="apellido2-div">
        <label for="apellido2"><strong>Segundo apellido</strong></label>
        <input id="apellido2" type="text" value="" name="apellido2" maxlength
            ="100"></input>
      </div>
      <div class="direccion-div">
        <label for="direccion"><strong>Dirección</strong></label>
        <input id="direccion" type="text" value="" name="direccion" maxlength
            ="100"></input>
      </div>
      <div class="admin-div">
        <label for="admin"><strong>Administrador</strong></label>
        <input id="admin" type="checkbox" value="" name="admin"></input>
      </div>
      <div class="contrasena-div">
        <label for="contrasena"><strong>Contraseña</strong></label>
        <input id="contrasena" type="password" value="" name="contrasena"
            maxlength="100"></input>
      </div>
      <input class="boton" id="insertar" type="submit" value="Insertar" name="
          insertar"></input>
      <input class="boton" id="limpiar" type="reset" value="Limpiar datos" name
          ="limpiar"></input>
      </form>

    </div>
    </div>
  </div>

  </body>
</html>
```

### B.4.15 inicio_tras_error.jsp

**Código B.30** Fichero JSP para mostrar un error al intentar iniciar sesión.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>

  <head>
    <title>Reserva de libros</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
  <script src="js/global.js"></script>
  <script src="js/jsinicio.js"></script>
  <script src="js/funciones.js"></script>

  </head>
```

```
  <body>

  <div id="bienvenida">
    <h1> Error en el usuario o contrasea. Por favor, introdzcalos correctamente
        .</h1>
    <h1>Bienvenido a</h1>
    <p><img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
        /></p>
      <h3>Acceda con su usuario y clave</h3>
    <div id="formacceso">
      <form action="login" method="post">
        <p> <label for="usuario">Cliente:</label><br/>
        <input type="text" name="cliente" size="20" id="cliente" /></p>
        <p> <label for="clave">Contrasea:</label> <br/>
        <input type="password" name="contrasena" size="20" id="contrasena" /></
            p>
        <p>
        <input class="boton" type="submit" name="entrar" value="Entrar" />
        <input class="boton" type="button" name="resetear" value="Limpiar
            campos" onclick = "location='inicio.jsp'"/>
        </p>
      </form>
    </div>
  </div>

  </body>
</html>
```

### B.4.16 reserva_libro_correcto.jsp

**Código B.31** Fichero JSP para mostrar un mensaje indicando que la reserva de un libro ha sido realizada correctamente.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Reserva Libro</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
      <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
          />
      </a>
    </div>
  </div>

  <div id="menu">
```

```
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Reserva realizada correctamente </h1>
  </div>

</body>
</html>
```

### B.4.17 ins_cliente_correcto.jsp

**Código B.32** Fichero JSP para mostrar un mensaje indicando que el cliente ha sido creado correctamente.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Insercin Cliente</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
      <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
          />
      </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Cliente insertado correctamente </h1>
  </div>

</body>
</html>
```

### B.4.18 login.jsp

**Código B.33** Vista para iniciar sesión.

```
<%@ page language="java" pageEncoding="UTF-8" %>
```

```
<%-- Inicializa el bean "usuario" con los datos pasados
  del formulario: usuario y clave --%>
<jsp:useBean id="cliente" class="modelo.Cliente" scope="session" />
<jsp:setProperty name="cliente" property="*" />

<%-- Si es incorrecto... --%>
<%
  if (!cliente.getCorrecto()) {
    //Establecemos un atributo con ambito request para indicar que
    // hay un error
    request.setAttribute("error", true);

    //Cerrar sesión
    session.invalidate();

    //reenviamos
    pageContext.forward("inicio.jsp");
  }
%>
```

### B.4.19 web.xml

**Código B.34** Fichero web.xml con información relacionada con la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
    java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/
    javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
    version="3.0">
  <display-name>reserva_libros</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>inicio.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>Listar_Libros</servlet-name>
    <servlet-class>controlador.Listar_Libros</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Listar_Libros</servlet-name>
    <url-pattern>/listado_libros</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>Listar_Reservas</servlet-name>
    <servlet-class>controlador.Listar_Reservas</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Listar_Reservas</servlet-name>
    <url-pattern>/listado_reservas</url-pattern>
  </servlet-mapping>
```

```
</web-app>
```

### B.4.20 listar_clientes.jsp

**Código B.35** Vista para listar todos los clientes.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />
<title>Listado de clientes</title>
</head>
<body>
    <div id="cabecera">
    <div id="logotipo">
        <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
            />
        </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1>Listado de clientes</h1>
    <div id="lista-div">

      <table id="lista-tabla">
        <thead>
          <tr>
            <td>DNI</td>
            <td>Cliente</td>
            <td>Nombre</td>
            <td>Primer apellido</td>
            <td>Segundo apellido</td>
            <td>Direccin</td>
            <td>Administrador</td>
            <td>Contrasea</td>
          </tr>
        </thead>
        <c:forEach var="listado_clientes" items="${sessionScope.clientes}">
          <tr>
```

```
            <td>${listado_clientes.getDni()}</td>
            <td>${listado_clientes.getCliente()}</td>
            <td>${listado_clientes.getNombre()}</td>
            <td>${listado_clientes.getApellido1()}</td>
            <td>${listado_clientes.getApellido2()}</td>
            <td>${listado_clientes.getDireccion()}</td>
            <td>${listado_clientes.getAdmin()}</td>
            <td>${listado_clientes.getContrasena()}</td>
          </tr>
        </c:forEach>
      </table>
    </div>
  </div>
</body>
</html>
```

### B.4.21 borrar_reserva_correcto.jsp

**Código B.36** Archivo JSP para mostrar un mensaje indicando que la reserva ha sido borrada de forma correcta.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Eliminacin Reserva</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
      <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
          />
      </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Reserva eliminada correctamente </h1>
  </div>

</body>
</html>
```

### B.4.22 inicio.jsp

**Código B.37** Vista de inicio.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>

  <head>
    <title>Reserva de libros</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
    <!--
  <script src="js/global.js"></script>
  <script src="js/jsinicio.js"></script>
  <script src="js/funciones.js"></script>
  -->
  </head>

  <body>

  <div id="bienvenida">
    <h1>Bienvenido a</h1>
    <p><img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
        /></p>
      <h3>Acceda con su usuario y clave</h3>
    <div id="formacceso">
      <form action="login" method="post">
        <p> <label for="usuario">Cliente:</label><br/>
        <input type="text" name="cliente" size="20" id="cliente" /></p>
        <p> <label for="clave">Contrasea:</label> <br/>
        <input type="password" name="contrasena" size="20" id="contrasena" /></
            p>
        <p>
        <input class="boton" type="submit" name="entrar" value="Entrar" />
        <input class="boton" type="button" name="resetear" value="Limpiar
            campos" onclick = "location='inicio.jsp'"/>
        </p>
      </form>
    </div>
  </div>

  </body>
</html>
```

### B.4.23 borrar.jsp

**Código B.38** JSP para borrar una reserva.

```
<%@ page language="java" pageEncoding="UTF-8"
  import="java.sql.*, reserva_libros.Constantes"
  trimDirectiveWhitespaces="true"%>

<jsp:useBean id="cliente" class="modelo.Cliente" scope="session" />
```

```
<%
/*
Si el usuario es correcto la respuesta será:
  0  = exito
  1  = error
En caso contrario, no devolverá nada.
*/

//Verificamos que estamos en una sesion
if(cliente.getCorrecto()) {

   int resultado = -1;
    String consultaSql = "DELETE FROM reservas"+
      " WHERE isbn="+ request.getParameter("isbn"); //+
      //
      //"  AND dni="+ cliente.getDni(); esto sobra porque la clave isbn es única

    //Nos conectamos a la base de datos
    try {
    Class.forName(Constantes.DB_DRIVER).newInstance();

    Connection conn = DriverManager.getConnection(Constantes.DB_URL,
        Constantes.DB_USER, Constantes.DB_PASS);
      Statement st = conn.createStatement();
      int resUpdate = st.executeUpdate( consultaSql );
      if (resUpdate > 0 ) {
        resultado = 0;
      } else {
        resultado = 1;
      }
    st.close();
      conn.close();


  } catch (Exception e) {
    resultado = 1;
    e.printStackTrace();
  }
    out.println(resultado);
}
%>
```

### B.4.24 nuevo_libro.jsp

**Código B.39** Formulario para crear un nuevo libro.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.sql.*, java.util.Vector,java.util.
        ArrayList, modelo.Cliente, reserva_libros.Constantes" %>

<%-- Comprobacion de login correcto. Insertamos login.jsp --%>
<%@include file="login.jsp" %>
<%
if (request.isRequestedSessionIdValid() && !cliente.getAdmin()) {
```

```
  response.sendError(HttpServletResponse.SC_FORBIDDEN, "Debe ser administrador
      .");
}
%>

<!DOCTYPE html>
<html>

  <head>
    <title>Nuevo libro</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
    <!--
    <script type="text/javascript" src="js/global.js"></script>
    <script type="text/javascript" src="js/compr_nuevo_libro.js"></script>
  <script type="text/javascript" src="js/funciones.js"></script>
  -->
  </head>

  <body>
    <div id="cabecera">
    <div id="logotipo">
        <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniería Telemá
            tica" />
        </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1>Insertar nuevo libro</h1>
      <div id="crear">
    <div id="formcrear">
      <form method="post" action="insercion_libro" onsubmit="return compr_nuevo
          _libro();">
      <div class="isbn-div">
        <label for="isbn"><strong>ISBN</strong></label>
        <input id="isbn" type="text" value="" name="isbn" maxlength="100"></
            input>
      </div>
      <div class="titulo-div">
        <label for="titulo"><strong>Título</strong></label>
        <input id="titulo" type="text" value="" name="titulo" maxlength="100"></
            input>
      </div>
      <div class="autor-div">
        <label for="autor"><strong>Autor</strong></label>
        <input id="autor" type="text" value="" name="autor" maxlength="100"></
            input>
      </div>
      <div class="editorial-div">
        <label for="editorial"><strong>Editorial</strong></label>
```

```
      <input id="editorial" type="text" value="" name="editorial" maxlength
          ="100"></input>
    </div>
    <div class="anio_pub-div">
      <label for="anio_pub"><strong>Año de publicación</strong></label>
      <input id="anio_pub" type="text" value="" name="anio_pub" maxlength
          ="100"></input>
    </div>
    <div class="genero-div">
      <label for="genero"><strong>Género</strong></label>
      <select id="genero" name="genero">
      <%
    //Mensaje de error, si se produce algún error o excepción al acceder a la
    // base de datos.
    String mensajeError = "";
    //Consulta SQL
      String sql="SELECT genero FROM generos;";
    //constantes

   //      String combobox = request.getParameter("combobox");
    //Nos conectamos a la base de datos
    try {
      Class.forName(Constantes.DB_DRIVER).newInstance();

      Connection conn = DriverManager.getConnection(Constantes.DB_URL,
          Constantes.DB_USER, Constantes.DB_PASS);
      try {
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        ArrayList vec=new ArrayList();
    //    Vector vec=new Vector();
    //    combobox.removeAllItems();
        while (rs.next()) {
          vec.add(rs.getString(1));
      //    combobox.addItem(rs.getString(1));
        } %>

        <option value="0" selected>Seleccione</option>
        <%for(int i=0;i<vec.size();i++){%>
        <option value="<%=vec.get(i).toString().trim()%>"><%=vec.get(i).
            toString().trim()%></option>
        <% }
        rs.close();
        st.close();
      } catch (SQLException e) {
        mensajeError = "No se ha podido obtener el listado de libros." + e;
      }
      conn.close();
    } catch (SQLException e) {
      e.printStackTrace();
      mensajeError = "Imposible acceder a la base de datos.";
    }   %>


      </select>
    </div>
    <div class="observaciones-div">
```

```
        <label for="observaciones"><strong>Observaciones</strong></label>
        <input id="observaciones" type="text" value="" name="observaciones"
            maxlength="100"></input>
      </div>
      <input class="boton" id="insertar" type="submit" value="Insertar" name="
          insertar"></input>
      <input class="boton" id="limpiar" type="reset" value="Limpiar datos" name
          ="limpiar"></input>
      </form>

    </div>
    </div>
  </div>

  </body>
</html>
```

### B.4.25 datos.jsp

**Código B.40** Archivo para mostrar datos en la aplicación.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="modelo.Cliente" %>

<%-- Comprobacion de login correcto. Insertamos login.jsp --%>
<%@include file="login.jsp" %>


<!DOCTYPE html>
<html>

  <head>
    <title>Reserva de libros</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
  </head>

  <body>
    <div id="cabecera">
    <div id="logotipo">
        <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniería Telemá
            tica" />
        </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1>Datos del cliente</h1>
    <div id="datos" class="${cookie.estilo.value ? 'estiloAlternativo' : '' }">
```

```
    <ul>
      <li>Dni:
        <span id="d_dni" class="datovalor">
          ${cliente.dni }
        </span></li>
      <li>Cliente:
        <span id="d_cliente" class="datovalor">
          ${cliente.cliente }
        </span></li>
      <li>Nombre:
        <span id="d_nombre" class="datovalor">
          ${cliente.nombre }
        </span></li>
      <li>Primer apellido:
        <span id="d_apellido1" class="datovalor">
          ${cliente.apellido1 }
        </span></li>
      <li>Segundo apellido:
        <span id="d_apellido2" class="datovalor">
          ${cliente.apellido2 }
        </span></li>
      <li>Dirección:
        <span id="d_direccion" class="datovalor">
          ${cliente.direccion }
        </span></li>
      <li>Administrador:
        <span id="d_admin" class="datovalor">
          ${cliente.admin ? "SÍ" : "NO" }
        </span></li>
    </ul>
    </div>
  </div>

  </body>
</html>
```

### B.4.26 reservas_cliente.jsp

**Código B.41** Fichero JSP para mostrar las reservas pertenecientes a un cliente.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.sql.*, reserva_libros.Constantes, modelo.
        Cliente" %>

<%-- Comprobacion de login correcto. Insertamos login.jsp --%>
<%@include file="login.jsp" %>

<!DOCTYPE html>
<html>

  <head>
    <title>Mis reservas</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/estilorp.css" />
    <script src="js/global.js"></script>
```

```
  <script src="js/funcionesrp.js"></script>
</head>

<body>
  <div id="cabecera">
  <div id="logotipo">
      <a href="datos.jsp">
      <img src="images/logoDep.jpg" alt="Departamento de Ingeniería Telemá
          tica" />
      </a>
  </div>
</div>



<div id="menu">
  <%-- Insertamos menu.jsp --%>
  <jsp:include page="menu.jsp"></jsp:include>
</div>

<div id="contenido">
  <h1>Mis reservas</h1>
    <div id="lista-div">

  <table id="lista-tabla">
    <thead>
      <tr>
        <th>ISBN</th>
        <th>Título</th>
        <th>Autor</th>
        <th>Editorial</th>
        <th>Año de publicación</th>
        <th>Género</th>
        <th>Observaciones</th>
      </tr>
    </thead>
    <%
    //Mensaje de error, si se produce algún error o excepción al acceder a la
    // base de datos.
    String mensajeError = "";
    //Consulta SQL
      String sql="SELECT libros.* FROM libros, reservas" +
          " WHERE libros.isbn = reservas.isbn AND reservas.dni = "+ cliente.
              getDni() + ";";
    //constantes
    //Nos conectamos a la base de datos
    try {
      Class.forName(Constantes.DB_DRIVER).newInstance();

      Connection conn = DriverManager.getConnection(Constantes.DB_URL,
          Constantes.DB_USER, Constantes.DB_PASS);
      try {
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
          //Generamos tabla
          long isbn = rs.getLong(1);
```

```
            String titulo = rs.getString(2);
            String autor = rs.getString(3);
            String editorial = rs.getString(4);
            int anio_pub = rs.getInt(5);
            String genero = rs.getString(6);
            String observaciones = rs.getString(7);
            %>
            <tr id='libro-<%=isbn%>'>
              <td><%=isbn%></td>
              <td><%=titulo%></td>
              <td><%=autor%></td>
              <td><%=editorial%></td>
              <td><%=anio_pub%></td>
              <td><%=genero%></td>
              <td><%=observaciones%></td>
              <td>
       <!--         <button value="Borrar" onClick="location='borrar_reserva'">
            Borrar</button>
-->
              </td>
            </tr>
            <%
          }
          rs.close();
          st.close();
        } catch (SQLException e) {
          mensajeError = "No se ha podido obtener el listado de libros." + e;
        }
        conn.close();
      } catch (SQLException e) {
        e.printStackTrace();
        mensajeError = "Imposible acceder a la base de datos.";
      }
    %>
    </table>
    <br><br><hr>

  </div>
  <%
  //Muestra mensaje de error
  if ( !mensajeError.isEmpty() ) {
  %>
    <div id="error"><p> ERROR:
    <%=mensajeError%>
    </p></div>
    <%
  }
  %>
  </div>

  </body>
</html>
```

### B.4.27 borrar_reserva_error.jsp

**Código B.42** Código para mostrar un error tras borrar una reserva.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Eliminacin Reserva</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
            />
      </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Error al eliminar la reserva </h1>
  </div>

</body>
</html>
```

### B.4.28 logout.jsp

**Código B.43** JSP para cerrar la sesión.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>

<%
  session.invalidate();
  pageContext.forward("inicio.jsp");
%>
```

### B.4.29 ins_libro_error.jsp

**Código B.44** Fichero JSP para mostrar un error al insertar un libro.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
    org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" href="css/estilorp.css" />

<title>Insercin Libro</title>
</head>
<body>
  <div id="cabecera">
    <div id="logotipo">
      <a href="datos.jsp">
        <img src="images/logoDep.jpg" alt="Departamento de Ingeniera Telemtica"
            />
      </a>
    </div>
  </div>

  <div id="menu">
    <%-- Insertamos menu.jsp --%>
    <jsp:include page="menu.jsp"></jsp:include>
  </div>

  <div id="contenido">
    <h1> Error al insertar el nuevo libro </h1>
  </div>

</body>
</html>
```

## B.5 Ficheros de configuración

### B.5.1 .gitlab-ci.yml

**Código B.45** Configuración de tareas automatizadas en gitlab.

```
stages: # Este atributo sirve para indicar las distintas fases que se tendrán
    en la tarea automatizada.
- pruebas # La primera fase es 'pruebas'.
- despliegue # La segunda fase es 'despliegue'.

default: # Este atributo indica los parámetros que se tomarán por defecto en
    todas las fases.
  image: maven:3.6-jdk-13 # La imagen docker usada por defecto en las fases es
      'maven:3.6-jdk-13'.

services: # Servicios extras que se necesiten en el entorno.
- postgres:10 # Es necesario la ejecución de una base de datos postgres para la
     ejecución de las pruebas.

variables: # Variables globales en el entorno.
  POSTGRES_DB: reserva_libros # Esta variable es usada por postgres y creará
      una base de datos llamada 'reserva_libros'.
```

```
  POSTGRES_USER: dit # Postgres creará un usuario llamado ‘dit’.
  POSTGRES_PASSWORD: "dit" # La contraseña es ‘dit’.

pruebas: # Inicio de la primera tarea.
  stage: pruebas # Pertenece a la primera fase llamada ‘pruebas’.
  only: # Esta tarea sólo debe ejecutarse en los casos indicados a continuación
      .
    - merge_requests # Sólo debe ejecutarse para modificaciones sobre una
        petición de integración.
  script: # Qué debe ser ejecutado en esta tarea.
    - echo ’Ejecutando pruebas’ # Mostrar un mensaje por la línea de comandos
        que indique el inicio de la ejecución de las pruebas.
    - cd ./reserva_libros/ # Se cambia el directorio al directorio donde se
        encuentra la aplicación.
    - mvn clean test # Se ejecutan las pruebas existentes en el proyecto, con
        el argumento clean se indica que elimine todos los ficheros que cree
        para la ejecución de las pruebas.

despliegue: # Inicio de la segunda tarea.
  stage: despliegue # Pertenece a la segunda fase llamada ‘despliegue’.
  only:
    - merge_requests # Sólo debe ejecutarse para modificaciones sobre una
        petición de integración.
  variables:
    DB_ADDR: ’localhost’ # Variable de entorno que será usada por la aplicación
         para definir la dirección donde se encuentra la base de datos.
  script:
    - echo ’Desplegando’ # Mostrar un mensaje por la línea de comandos que
        indique el inicio del despliegue.
    - cd ./reserva_libros/ # Se cambia el directorio al directorio donde se
        encuentra la aplicación.
    - mvn -Dmaven.test.skip=true -Dname=$GITLAB_USER_LOGIN -Dserver.addr
        =172.17.0.1 tomcat7:redeploy # Se inicia el despliegue sin ejecutar las
         pruebas, sobre la dirección ‘172.17.0.1’\ y la aplicación será
        desplegada en el path con valor el nombre de usuario del usuario que
        ejecutó este despliegue.
```

### B.5.2 pom.xml

**Código B.46** Fichero POM de la aplicación.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd"> <!-- Estos pá
                          rametros son necesarios para Maven -->

  <modelVersion>4.0.0</modelVersion> <!-- Versión del archivo POM soportada,
      este parametro lo necesita Maven -->
  <groupId>com.reserva_libro.etsi</groupId> <!-- Id del grupo de desarrollo -->
  <artifactId>reserva_libros</artifactId> <!-- Id del proyecto -->
  <packaging>war</packaging> <!-- Formato de salida, en este caso interesa war
      para el despliegue en Tomcat -->
  <version>1</version> <!-- Versión del proyecto -->
```

```
<build> <!-- Información útil para la construcción de la aplicación -->
  <sourceDirectory>${project.basedir}/src</sourceDirectory> <!-- Directorio d
      ónde se encuentra el código Java -->
  <testSourceDirectory>${project.basedir}/test</testSourceDirectory> <!--
      Directorio dónde se encuentran las pruebas -->

  <plugins> <!-- Información relacionada con los plugins necesarios -->
    <plugin> <!-- Declaración de un plugin, este plugin es lo necesita Maven
        -->
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.19.1</version>
    </plugin>

    <plugin> <!-- Declaración de un plugin, este plugin lo necesita Maven
        para crear el fichero war -->
      <groupId>org.apache.maven.plugins</groupId> <!-- Id del grupo de
          desarrollo de este plugin -->
      <artifactId>maven-war-plugin</artifactId> <!-- Id del plugin -->
      <version>2.6</version> <!-- Versión del plugin -->
      <configuration> <!-- Configuración adicional para el plugin -->
        <webXml>${project.basedir}/src/webapp/WEB-INF/web.xml</webXml> <!--
            Ubicación del fichero 'web.xml'\ -->
        <webResources>
          <resource>
            <directory>${project.basedir}/src/webapp</directory> <!-- Dónde se
                 encuentra el código necesario que no es Java para la
                construcción de la aplicación -->
          </resource>
        </webResources>
      </configuration>
    </plugin>

    <plugin> <!-- Declaración de un plugin, este plugin lo necesita Maven
        para desplegar sobre Tomcat -->
      <groupId>org.apache.tomcat.maven</groupId> <!-- Id del grupo de
          desarrollo de este plugin -->
      <artifactId>tomcat7-maven-plugin</artifactId> <!-- Id del plugin -->
      <version>2.2</version> <!-- Versión del plugin -->
      <configuration> <!-- Configuración adicional para el plugin -->
        <url>http://${server.addr}:${server.port}/manager/text</url> <!-- URL
            dónde se encuentra el servicio, es configurable mediante linea de
            comandos con los argumentos -Dserver.addr y -Dserver.port -->
        <server>TomcatServer</server> <!-- Se indica que es un servidor Tomcat
             -->
        <path>/${name}</path> <!-- Ubicación dónde será desplegada la aplicaci
            ón -->
        <username>admin</username> <!-- Usuario necesario para el despliegue
            -->
        <password>1234</password> <!-- Contraseñas necesaria para el
            despliegue -->
      </configuration>
    </plugin>
  </plugins>
</build>

<properties> <!-- Configuración de variables por defecto -->
```

```
    <server.addr>localhost</server.addr> <!-- URL a la que intentará desplegar
        la aplicación por defecto -->
    <server.port>4000</server.port> <!-- Puerto en el que intentará desplegar
        la aplicación por defecto -->
    <maven.compiler.source>7</maven.compiler.source> <!-- Parametro -source
        pasado al compilador java requerido por Maven -->
    <maven.compiler.target>7</maven.compiler.target> <!-- Parametro -target
        pasado al compilador java requerido por Maven -->
  </properties>

  <dependencies> <!-- Declaración de dependencias de la aplicación -->
    <dependency> <!-- Declaración de dependencia, necesaria para el
        funcionamiento de la aplicación -->
      <groupId>javax.servlet</groupId> <!-- Id del grupo de desarrollo de esta
          dependencia -->
      <artifactId>javax.servlet-api</artifactId> <!-- Id de la dependencia -->
      <version>3.0.1</version> <!-- Versión de la dependencia -->
      <scope>provided</scope> <!-- Necesario por la dependencias -->
    </dependency>

    <dependency> <!-- Declaración de dependencia, necesaria para el
        funcionamiento de la aplicación -->
      <groupId>postgresql</groupId> <!-- Id del grupo de desarrollo de esta
          dependencia -->
      <artifactId>postgresql</artifactId> <!-- Id de la dependencia -->
      <version>9.1-901-1.jdbc4</version> <!-- Versión de la dependencia -->
    </dependency>

    <dependency> <!-- Declaración de dependencia, necesaria para el
        funcionamiento de la aplicación -->
      <groupId>jstl</groupId> <!-- Id del grupo de desarrollo de esta
          dependencia -->
      <artifactId>jstl</artifactId> <!-- Id de la dependencia -->
      <version>1.2</version> <!-- Versión de la dependencia -->
    </dependency>

    <dependency> <!-- Declaración de dependencia, necesaria para las pruebas
        -->
      <groupId>junit</groupId> <!-- Id del grupo de desarrollo de esta
          dependencia -->
      <artifactId>junit</artifactId> <!-- Id de la dependencia -->
      <version>4.11</version> <!-- Versión de la dependencia -->
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

## B.6 Pruebas unitarias

### B.6.1 NuevoClienteTest.java

**Código B.47** Pruebas para el modelo NuevoCliente.

```
package unidad;
```

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import org.junit.Test;

import modelo.NuevoCliente;
import modelo.Conexion;

public class NuevoClienteTest {

  @Test
  public final void testConstructor1NuevoClienteOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 1
        del modelo NuevoCliente funciona bien");

    NuevoCliente nuevoCliente = new NuevoCliente();
  }

  @Test
  public final void testConstructor2NuevoClienteOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 2
        del modelo NuevoCliente funciona bien");

    Integer dni = 111111;
    String usuario = "Usuario1";
    String nombre = "Nombre";
    String apellido1 = "Primer apellido";
    String apellido2 = "Segundo apellido";
    String direccion = "Dirección del nuevoCliente";
    Boolean admin = true;
    String contrasena = "contraseña";

    NuevoCliente nuevoCliente;

    nuevoCliente = new NuevoCliente(dni, usuario, nombre, apellido1, apellido2,
         direccion, admin, contrasena);

    assertEquals("El dni debería haberse asignado correctamente", nuevoCliente.
        getDni(), dni);
    assertEquals("El nuevoCliente debería haberse asignado correctamente",
        nuevoCliente.getCliente(), usuario);
    assertEquals("El nombre debería haberse asignado correctamente",
        nuevoCliente.getNombre(), nombre);
    assertEquals("El apellido1 debería haberse asignado correctamente",
        nuevoCliente.getApellido1(), apellido1);
    assertEquals("El apellido2 debería haberse asignado correctamente",
        nuevoCliente.getApellido2(), apellido2);
    assertEquals("La direccion debería haberse asignado correctamente",
        nuevoCliente.getDireccion(), direccion);
    assertEquals("El valor admin debería haberse asignado correctamente",
        nuevoCliente.getAdmin(), admin);
    assertEquals("La contrasena debería haberse asignado correctamente",
        nuevoCliente.getContrasena(), contrasena);
  }

  @Test
  public final void testConstructor3NuevoClienteOK() {
```

```
    System.out.println("Comienza la prueba que comprueba que el constructor 3
        del modelo NuevoCliente funciona bien");

    Integer dni = 111111;
    String nombre = "Nombre";
    String apellido1 = "Primer apellido";
    String apellido2 = "Segundo apellido";
    String direccion = "Dirección del nuevoCliente";
    Boolean admin = true;

    NuevoCliente nuevoCliente;

    nuevoCliente = new NuevoCliente(dni, nombre, apellido1, apellido2,
        direccion, admin);

    assertEquals("El dni debería haberse asignado correctamente", nuevoCliente.
        getDni(), dni);
    assertEquals("El nombre debería haberse asignado correctamente",
        nuevoCliente.getNombre(), nombre);
    assertEquals("El apellido1 debería haberse asignado correctamente",
        nuevoCliente.getApellido1(), apellido1);
    assertEquals("El apellido2 debería haberse asignado correctamente",
        nuevoCliente.getApellido2(), apellido2);
    assertEquals("La direccion debería haberse asignado correctamente",
        nuevoCliente.getDireccion(), direccion);
    assertEquals("El valor admin debería haberse asignado correctamente",
        nuevoCliente.getAdmin(), admin);
  }
}
```

### B.6.2 ClienteTest.java

**Código B.48** Pruebas para el modelo Cliente.

```
package unidad;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import org.junit.Test;

import modelo.Cliente;
import modelo.Conexion;

public class ClienteTest {

  @Test
  public final void testConstructor1ClienteOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 1
        del modelo Cliente funciona bien");

    Cliente cliente = new Cliente();
  }

  @Test
  public final void testConstructor2ClienteOK() {
```

```
  System.out.println("Comienza la prueba que comprueba que el constructor 2
      del modelo Cliente funciona bien");

  Integer dni = 111111;

  Cliente cliente;

  cliente = new Cliente(dni);

  assertEquals("El dni debería haberse asignado correctamente", cliente.
      getDni(), dni);
}

@Test
public final void testConstructor3ClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el constructor 3
      del modelo Cliente funciona bien");

  Integer dni = 111111;
  String usuario = "Usuario1";
  String nombre = "Nombre";
  String apellido1 = "Primer apellido";
  String apellido2 = "Segundo apellido";
  String direccion = "Dirección del cliente";
  Boolean admin = true;
  String contrasena = "contraseña";

  Cliente cliente;

  cliente = new Cliente(dni, usuario, nombre, apellido1, apellido2, direccion
      , admin, contrasena);

  assertEquals("El dni debería haberse asignado correctamente", cliente.
      getDni(), dni);
  assertEquals("El cliente debería haberse asignado correctamente", cliente.
      getCliente(), usuario);
  assertEquals("El nombre debería haberse asignado correctamente", cliente.
      getNombre(), nombre);
  assertEquals("El apellido1 debería haberse asignado correctamente", cliente.
      getApellido1(), apellido1);
  assertEquals("El apellido2 debería haberse asignado correctamente", cliente.
      getApellido2(), apellido2);
  assertEquals("La direccion debería haberse asignado correctamente", cliente.
      getDireccion(), direccion);
  assertEquals("El valor admin debería haberse asignado correctamente",
      cliente.getAdmin(), admin);
  assertEquals("La contrasena debería haberse asignado correctamente",
      cliente.getContrasena(), contrasena);
}

@Test
public final void testConstructor4ClienteOK() {
  System.out.println("Comienza la prueba que comprueba que el constructor 4
      del modelo Cliente funciona bien");

  Integer dni = 111111;
  String nombre = "Nombre";
```

```
    String apellido1 = "Primer apellido";
    String apellido2 = "Segundo apellido";
    String direccion = "Dirección del cliente";
    Boolean admin = true;

    Cliente cliente;

    cliente = new Cliente(dni, nombre, apellido1, apellido2, direccion, admin);

    assertEquals("El dni debería haberse asignado correctamente", cliente.
        getDni(), dni);
    assertEquals("El nombre debería haberse asignado correctamente", cliente.
        getNombre(), nombre);
    assertEquals("El apellido1 debería haberse asignado correctamente", cliente.
        getApellido1(), apellido1);
    assertEquals("El apellido2 debería haberse asignado correctamente", cliente.
        getApellido2(), apellido2);
    assertEquals("La direccion debería haberse asignado correctamente", cliente.
        getDireccion(), direccion);
    assertEquals("El valor admin debería haberse asignado correctamente",
        cliente.getAdmin(), admin);
  }
}
```

### B.6.3 ReservaTest.java

**Código B.49** Pruebas para el modelo Reserva.

```
package unidad;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import org.junit.Test;

import modelo.Reserva;
import modelo.Conexion;

public class ReservaTest {

  @Test
  public final void testConstructor1ReservaOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 1
        del modelo Reserva funciona bien");

    Reserva reserva = new Reserva();
  }

  @Test
  public final void testConstructor2ReservaOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 2
        del modelo Reserva funciona bien");

    Long isbn = 123456L;
    Reserva reserva;
```

```
    reserva = new Reserva(isbn);

    assertEquals("El ISBN debería haberse asignado correctamente", reserva.
        getIsbn(), isbn);
  }

  @Test
  public final void testConstructor3ReservaOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 3
        del modelo Reserva funciona bien");

    Long isbn = 123456L;
    Integer dni = 111111;


    Reserva reserva;

    reserva = new Reserva(isbn, dni);

    assertEquals("El ISBN debería haberse asignado correctamente", reserva.
        getIsbn(), isbn);
    assertEquals("El dni debe haberse asignado correctamente", reserva.getDni()
        , dni);
  }

  @Test
  public final void testConstructor4ReservaOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 4
        del modelo Reserva funciona bien");

    Long isbn = 123456L;
    String titulo = "Don Quijote de la Mancha";
    String autor = "Miguel de Cervantes Saavedra";
    String editorial = "Francisco de Robles";
    Integer anio_pub = 1605;
    String genero = "Novela de aventuras";
    String observaciones = null;
    Integer dni = 111111;

    Reserva reserva;

    reserva = new Reserva(isbn, titulo, autor, editorial, anio_pub, genero,
        observaciones, dni);

    assertEquals("El ISBN debería haberse asignado correctamente", reserva.
        getIsbn(), isbn);
    assertEquals("El titulo debería haberse asignado correctamente", reserva.
        getTitulo(), titulo);
    assertEquals("El autor debería haberse asignado correctamente", reserva.
        getAutor(), autor);
    assertEquals("La editorial debería haberse asignado correctamente", reserva.
        getEditorial(), editorial);
    assertEquals("El anio de publicacion debería haberse asignado correctamente
        ", reserva.getAnio_pub(), anio_pub);
    assertEquals("El genero debería haberse asignado correctamente", reserva.
        getGenero(), genero);
```

```
    assertEquals("Las observaciones deben haberse asignado correctamente",
        reserva.getObservaciones(), observaciones);
    assertEquals("El dni debe haberse asignado correctamente", reserva.getDni()
        , dni);
  }
}
```

### B.6.4 LibroTest.java

**Código B.50** Pruebas para el modelo Libro.

```
package unidad;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import org.junit.Test;

import modelo.Libro;
import modelo.Conexion;

public class LibroTest {

  @Test
  public final void testConstructor1LibroOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 1
        funciona bien");

    Libro libro = new Libro();
  }

  @Test
  public final void testConstructor2LibroOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 2
        funciona bien");

    Long isbn = 123456L;

    Libro libro;

    libro = new Libro(isbn);

    assertEquals("El ISBN debería haberse asignado correctamente", libro.
        getIsbn(), isbn);
  }

  @Test
  public final void testConstructor3LibroOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 3
        funciona bien");

    Long isbn = 123456L;
    String titulo = "Don Quijote de la Mancha";
    String autor = "Miguel de Cervantes Saavedra";
    String editorial = "Francisco de Robles";
    Integer anio_pub = 1605;
```

```
    String genero = "Novela de aventuras";
    String observaciones = null;

    Libro libro;

    libro = new Libro(isbn, titulo, autor, editorial, anio_pub, genero,
        observaciones);

    assertEquals("El ISBN debería haberse asignado correctamente", libro.
        getIsbn(), isbn);
    assertEquals("El titulo debería haberse asignado correctamente", libro.
        getTitulo(), titulo);
    assertEquals("El autor debería haberse asignado correctamente", libro.
        getAutor(), autor);
    assertEquals("La editorial debería haberse asignado correctamente", libro.
        getEditorial(), editorial);
    assertEquals("El anio de publicacion debería haberse asignado correctamente
        ", libro.getAnio_pub(), anio_pub);
    assertEquals("El genero debería haberse asignado correctamente", libro.
        getGenero(), genero);
    assertEquals("Las observaciones deben haberse asignado correctamente",
        libro.getObservaciones(), observaciones);
  }

  @Test
  public final void testConstructor4LibroOK() {
    System.out.println("Comienza la prueba que comprueba que el constructor 4
        funciona bien");

    Long isbn = 123456L;
    String titulo = "Don Quijote de la Mancha";
    String autor = "Miguel de Cervantes Saavedra";
    String editorial = "Francisco de Robles";
    Integer anio_pub = 1605;
    String genero = "Novela de aventuras";
    String observaciones = null;
    Integer dni = 111111;

    Libro libro;

    libro = new Libro(isbn, titulo, autor, editorial, anio_pub, genero,
        observaciones, dni);

    assertEquals("El ISBN debería haberse asignado correctamente", libro.
        getIsbn(), isbn);
    assertEquals("El titulo debería haberse asignado correctamente", libro.
        getTitulo(), titulo);
    assertEquals("El autor debería haberse asignado correctamente", libro.
        getAutor(), autor);
    assertEquals("La editorial debería haberse asignado correctamente", libro.
        getEditorial(), editorial);
    assertEquals("El anio de publicacion debería haberse asignado correctamente
        ", libro.getAnio_pub(), anio_pub);
    assertEquals("El genero debería haberse asignado correctamente", libro.
        getGenero(), genero);
    assertEquals("Las observaciones deben haberse asignado correctamente",
        libro.getObservaciones(), observaciones);
```

```
    assertEquals("El dni debe haberse asignado correctamente", libro.getDni(),
        dni);
  }
}
```

## B.7 Ficheros para la inicialización de los datos

### B.7.1 create.rake

**Código B.51** Rake task para la creación de los datos para la realización de la práctica.

```
require 'active_record/fixtures'

N_PAREJAS = 10

namespace :dit do
  desc "DIT | Inicializar los datos"
  task :inicializar => :environment do
    puts "Inicializando datos"

    Rake::Task['gitlab:db:drop_tables'].invoke
    Rake::Task['gitlab:db:configure'].invoke

    admin = User.create!(name: "Dit", email: "admin@dit.com", password:"D!
        t12345", username: "dit", admin: true)
    admin.confirm

    Rake::Task['dit:inicializar_grupo'].invoke(1)
    Rake::Task['dit:inicializar_grupo'].reenable
    Rake::Task['dit:inicializar_grupo'].invoke(2)
    Rake::Task['dit:inicializar_grupo'].reenable
    Rake::Task['dit:inicializar_grupo'].invoke(3)

    ApplicationSetting.update(signup_enabled: false, default_branch_protection:
         0)
    Ci::Runner.create!(token: "dit", runner_type: "instance_type",
        first_day_of_week: 1)
  end

  desc "DIT | Inicializar datos grupo"
  task :inicializar_grupo, [:n_grupo] => :environment do |_, args|
    n_grupo = args[:n_grupo]

    unless n_grupo
      puts "Must specify a migration version as an argument".color(:red)
      exit 1
    end

    admin = User.find_by(username: "dit")

    puts "No se ha ejecutado la tarea inicial." && return unless admin.present?

    puts "Inicializando Grupo #{n_grupo}"
```

```
    repositorio = Project.find_by(name:"ProyectoG#{n_grupo}")
    if repositorio.present?
      puts("El repositorio \"ProyectoG#{n_grupo}\" está siendo reiniciado")

      Projects::DestroyService.new(repositorio, admin).execute
    end

    repositorio = Projects::CreateService.new(admin, name: "ProyectoG#{n_grupo}
        ", path: "proyecto#{n_grupo}", skip_disk_validation: true).execute
    repositorio.add_maintainer(admin)

    destroy_users = Users::DestroyService.new(admin)

    for i in 1..N_PAREJAS do
      # Creando Desarollador
      dev = User.find_by(username: "dev#{n_grupo}-#{i}")

      if dev.present?
        puts("Usuario dev#{n_grupo}-#{i} reiniciado")

        destroy_users.execute(dev)
      end

      dev = Users::CreateService.new(
        admin,
        name: "Developer#{n_grupo}-#{i}",
        email: "dev#{n_grupo}-#{i}@dit.com",
        password:"developer#{n_grupo}-#{i}",
        username: "dev#{n_grupo}-#{i}"
      ).execute
      dev.confirm

      repositorio.add_user(dev,30)

      # Creando Lider de equipo

      lider = User.find_by(username: "lider#{n_grupo}-#{i}")

      if lider.present?
        puts("Usuario lider#{n_grupo}-#{i} reiniciado")

        destroy_users.execute(lider)
      end

      lider = Users::CreateService.new(
        admin,
        name: "Lider#{n_grupo}-#{i}",
        email: "lider#{n_grupo}-#{i}@dit.com",
        password:"lider#{n_grupo}-#{i}",
        username: "lider#{n_grupo}-#{i}"
      ).execute
      lider.confirm

      repositorio.add_user(lider,30)
    end
  end
end # namespace end: dit
```

### B.7.2 inicializar_datos

**Código B.52** Script de inicialización de datos.

```
#! /bin/bash

sudo cp ./create.rake /opt/gitlab/embedded/service/gitlab-rails/lib/tasks/
    gitlab/create.rake

actualizar_repositorio() {
  nombre_repositorio=$1
  echo "El nombre de mi repositorio es $nombre_repositorio"

  if [ -z nombre_repositorio ]; then
    echo "Se debe pasar el nombre del repositorio como parámetro."
  else
    git remote add $nombre_repositorio http://localhost/dit/$nombre_repositorio.
        git

    git fetch $nombre_repositorio
    borrar_ramas_repositorio_remoto $nombre_repositorio
    git push -u $nombre_repositorio --all --force

    git remote remove $nombre_repositorio
  fi
}

borrar_ramas_repositorio_remoto() {
  nombre_repositorio=$1

  if [ -z nombre_repositorio ]; then
    echo "Se debe pasar el nombre del repositorio como parámetro."
  else
    git branch -r |
    grep $nombre_repositorio/ |
    grep -v 'master$' |
    grep -v HEAD |
    cut -d/ -f2 |
    while read branch; do
      echo "Borrando rama remota $branch de $nombre_repositorio"
      git push $nombre_repositorio :$branch
    done
  fi
}

if [ $? -eq 0 ]; then
  sudo gitlab-rake dit:inicializar

  actualizar_repositorio proyecto1
  actualizar_repositorio proyecto2
  actualizar_repositorio proyecto3

  sudo gitlab-rake cache:clear
fi
```

# Índice de Figuras

# Índice de Códigos

# Bibliografía


[1] *Asignaturas del grado en ingeniería de las tecnologías de telecomunicación*, https://www.us.es/estudiar/que-estudiar/oferta-de-grados/grado-en-ingenieria-de-las-tecnologias-de-telecomunicacion#edit-group-plani, [Online; accessed 3-March-2020].

[2] *Contenedores. ¿qué es docker?*, https://www.redhat.com/es/topics/containers/what-is-docker, [Online; accessed 16-April-2020].

[3] *Docker hub*, https://hub.docker.com/, [Online; accessed 16-May-2020].

[4] *Integración continua y entrega continua (ci/cd)*, https://www.aplyca.com/es/blog/integracion-entrega-continua-ci-cd, [Online; accessed 4-March-2020].

[5] *La famosa pirámide de cohn y la dura realidad*, https://medium.com/@wc.testing.qa/la-famosa-pir%C3%A1mide-de-cohn-y-la-dura-realidad-e1250dfbe5f3, [Online; accessed 5-February-2020].

[6] *Modelo en cascada*, https://es.ryte.com/wiki/Modelo_en_Cascada, [Online; accessed 5-February-2020].

[7] *Selenium*, https://www.selenium.dev/, [Online; accessed 12-May-2020].

[8] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas, *Manifiesto Ágil*, https://agilemanifesto.org/iso/es/manifesto.html, [Online; accessed 5-February-2020].

[9] _____, *Principios del manifiesto Ágil*, https://agilemanifesto.org/iso/es/principles.html, [Online; accessed 5-February-2020].

[10] Carlos Yagüe, *Qué es apache maven*, https://openwebinars.net/blog/que-es-apache-maven/, [Online; accessed 28-February-2020].

[11] Gabriel Fernando Chriboga, *Sistemas de control de versiones*, https://portfoliogabrielfcr.wordpress.com/2013/10/25/62/, [Online; accessed 5-February-2020].

[12] José Ruiz Cristina, *Qué es devops y sobre todo qué no es devops*, https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops, [Online; accessed 5-February-2020].

[13] Junta de Andalucía, *Marco y desarrollo de la junta de andalucía. junit*, http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/248, [Online; accessed 5-February-2020].

[14] Gitlab, *Copia de seguridad gitlab*, https://docs.gitlab.com/ee/raketasks/backup_restore.html, [Online; accessed 5-February-2020].

[15] Gitlab, *The devops lifecycle with gitlab*, https://about.gitlab.com/stages-devops-lifecycle/, [Online; accessed 19-February-2020].

[16] Integración continua y Entrega continua (CI/CD), *Metodologías ágiles*, https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/, [Online; accessed 5-February-2020].

[17] Cristina López-Goicochea, *La importancia del testing de software y de la automatización de pruebas*, *https://www.atsistemas.com/es/blog/la-importancia-del-testing-de-software-y-de-la-automatizacin-de-pruebas*, [Online; accessed 5-February-2020].

[18] Ailin Orjuela Duarte and Mauricio Rojas C., *Las metodologías de desarrollo Ágil como una oportunidad para la ingeniería del software educativo*, Revista Avances en Sistemas e Informática **5** (2008), no. 2, 159–171.

[19] Fabián Muñoz Peñas, *Aplicación web conforme a patrones para la docencia*, Ph.D. thesis, ETSI. Universidad de Sevilla, 2018.