



Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Aplicación web para la corrección de prácticas de programación

Memoria del Trabajo Fin de Grado en Ingeniería Informática

Autor:

Alejandro Quesada Mendo

Tutores: Mayte González de Lena Alonso, José F. Vélez Serrano

Septiembre 2020

Agradecimientos

Quiero agradecerle a los tutores el apoyo y la idea sobre la que se ha construido este proyecto. Y también, a aquellos docentes que han mantenido vivas mis ganas de seguir aprendiendo.

Sobre todo quiero agradecerle este proyecto a mis padres, que siempre me han apoyado y ayudado a cumplir mis objetivos.

Resumen

La corrección de prácticas y exámenes de programación suele resultar un proceso tedioso para los docentes. Este proceso suele requerir del uso de diversas herramientas de manera simultánea y corregir cada práctica individualmente. Es un proceso potencialmente automatizable.

Para intentar resolver el problema se ha decidido desarrollar un proyecto doble, por un lado, una aplicación de consola capaz de procesar, de manera automática, proyectos de estudiantes a partir de proyectos de referencia que incluyen tests unitarios, desarrollados por los docentes. Por otro lado, una aplicación web capaz de gestionar los proyectos de referencia y de actuar a modo de interfaz gráfica de la aplicación de consola. Ambos proyectos trabajan conjuntamente, pero en esta memoria solo se comenta el desarrollo de la segunda parte.

La aplicación web resultante permite discernir entre distintos tipos de usuario (docentes y estudiantes), gestionar proyectos de referencia (crear nuevos proyectos, borrarlos y actualizar su contenido), analizar el posible fraude entre un conjunto de proyectos y obtener informes de compilación y tests de cada uno de los proyectos procesados. Todo ello a través de una interfaz gráfica amigable y minimalista y entorno de ejecución seguro.

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	3
1.3	Estado del arte	3
1.4	Estructura de la memoria	3
2	Análisis del problema	5
2.1	Descripción del sistema actual	5
2.2	Modelo del problema a resolver	6
2.3	Requisitos	8
2.3.1	Requisitos funcionales	8
2.3.2	Requisitos no funcionales	9
2.4	Metodología	9
3	Diseño e implementación	11
3.1	Herramientas utilizadas	11
3.1.1	Java e IntelliJ IDEA	11
3.1.2	Maven y Spring Boot	12
3.1.3	Linux	12
3.1.4	Bootstrap	12
3.1.5	MySQL	12
3.1.6	JPlag	13
3.1.7	Docker	13
3.1.8	JRater	13
3.2	Arquitectura del software	14
3.2.1	Estructura del proyecto	14
3.2.2	Base de Datos, Entidades y Repositorios	16
3.2.3	Interfaz Gráfica y vistas	19
3.2.4	Controladores y Servicios	26
3.2.5	Control de Sesión	28
3.2.6	Paquete de herramientas	28
3.3	Despliegue del software: Dockerización	29
4	Métricas	33
4.1	Métricas temporales	33
4.2	Métricas sobre calidad del software	35

5 Conclusiones	37
5.1 Objetivos conseguidos	37
5.2 Lecciones aprendidas	38
5.2.1 Netbeans y sus ficheros de configuración	38
5.2.2 Usar SonarQube desde el principio del proyecto	38
5.2.3 Requisitos de la aplicación antes de dockerizar	38
5.3 Trabajos futuros	39
5.3.1 Ciberseguridad	39
5.3.2 Pruebas de Software	40
5.3.3 Gestión de usuarios	40
5.3.4 Soporte para nuevos lenguajes	40
5.3.5 Permitir la descarga de los informes	41
5.3.6 Visualización del código de un estudiante junto a su informe de corrección	41
Bibliografía	43

Índice de figuras

1	Diagrama de actividad del <i>script</i> usado por los profesores	6
2	Diagrama de casos de uso para los alumnos	6
3	Diagrama de casos de uso para los profesores	7
4	Diagrama de actividad para el caso de uso “Autoevaluación de una práctica”	7
5	Diagrama de actividad para el caso de uso “Autoevaluación de un conjunto de prácticas”	8
6	Diagrama de actividad para el caso de uso “subir una nueva práctica”	8
7	Esquema del funcionamiento de la arquitectura Modelo Vista Controlador	15
8	Esquema del funcionamiento de Spring MVC	15
9	Diagrama de clases general del proyecto	16
10	Diagrama Entidad-Relación de la base de datos del proyecto	17
11	Diagrama de clases de las entidades y repositorios	18
12	Pantalla principal sin inicio de sesión	19
13	Pantalla de una práctica sin inicio de sesión	19
14	Pantalla de una práctica sin inicio de sesión. Cuadro de diálogo ante formato incorrecto	20
15	Pantalla de una práctica sin inicio de sesión. Bloqueo mientras se realiza la autoevaluación	20
16	Pantalla de Informe individual sin inicio de sesión	21
17	Pantalla de inicio de sesión	21
18	Pantalla principal con inicio de sesión	22
19	Pantalla principal con inicio de sesión. Cuadro de diálogo para borrar una práctica	22
20	Pantalla de actualización de una práctica	23
21	Pantalla de Nueva Práctica	23
22	Pantalla de una práctica con inicio de sesión	24
23	Pantalla de informe global	24
24	Pantalla de informe individual con inicio de sesión	25
25	Diagrama de actividad entre las distintas pantallas de la aplicación	25
26	Diagrama de clases de los controladores y servicios	27
27	Diagrama de despliegue del proyecto	31
28	Diagrama de Gantt del proyecto	34
29	Tabla de hitos asociada al diagrama de Gantt	34
30	Análisis de calidad con SonarQube	35

Nomenclatura

CSS	Cascading Style Sheets, lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado como HTML
EDA	Estructuras de Datos Avanzadas
HTML	HyperText Markup Language, es un lenguaje de marcado para la elaboración de páginas web
HTTPS	HTTP Secure, la versión segura del protocolo HTTP
JPA	Java Persistence API, o API de Persistencia para el lenguaje Java
JRE	Java Runtime Environment, o Entorno de Ejecución de Java
LOB	List Of Bytes, o Lista de Bytes
MVC	Model View Controller, o Modelo Vista Controlador. Es un tipo de arquitectura para aplicaciones web
POM	Project Object Model, fichero de descripción de dependencias de un proyecto Maven
SQL	Structured Query Language, o Lenguaje de Consulta Estructurado. También define un tipo de base de datos llamado relacional
TLS	Transport Layer Security, es un protocolo de seguridad en la capa de transporte, sucesor del protocolo SSL
UML	Unified Modeling Language, o Lenguaje Unificado de Modelado
XML	eXtensible Markup Language, o Lenguaje de Marcado Extensible

Capítulo 1

Introducción

Es una práctica común en algunas asignaturas de ingeniería informática, que el método de evaluación consista en la entrega de prácticas y exámenes de programación [1]. La corrección de dichas prácticas y exámenes puede resultar una tarea tediosa para el profesorado ya que, el proceso de corrección requiere varias aplicaciones abiertas simultáneamente y una serie de pasos manuales a repetir para cada uno de los estudiantes.

Este trabajo se centra en el caso particular de la asignatura de Estructuras de Datos Avanzadas (o EDA), impartida en el tercer curso de Ingeniería Informática en el campus de Vicálvaro de la Universidad Rey Juan Carlos [2]. El objetivo general de esta asignatura consiste en introducir las propiedades y el funcionamiento de algunas de las estructuras de datos avanzadas mas importantes, así como su aplicación para resolver eficientemente determinados problemas.

Esta asignatura tiene un enfoque esencialmente práctico. La evaluación consiste en la entrega de una serie de prácticas evaluables y un examen final. En estas pruebas se exige a los estudiantes que programen alguna funcionalidad de una estructura de datos o que usen las estructuras de datos estudiadas para resolver un problema de forma eficiente.

La asignatura se imparte usando Java [3] como lenguaje de programación y Netbeans [4] como entorno de desarrollo para el lenguaje Java. Ambas tecnologías permiten la implementación y ejecución de tests unitarios de forma sencilla tanto para estudiantes como docentes. Por otra parte, la entrega de prácticas y exámenes se realiza íntegramente a través del Aula Virtual [5] de la universidad.

Un test o prueba unitaria es una función cuyo objetivo es comprobar el correcto funcionamiento de una unidad de código. Por ejemplo, el Algoritmo 1 comprueba si el método de encolado de una cola `enqueue(1)` es correcto.

Durante este capítulo se va a introducir el problema a resolver, así como cuál ha sido la motivación para la realización del proyecto, una enumeración de los objetivos del mismo y una vista sobre cuáles eran las soluciones ya existentes al problema planteado. Por último, se realiza una introducción a la estructura de este documento.

1.1. Motivación

El trabajo nace motivado por un intento de mejorar el sistema de corrección de la asignatura de EDA.

Algoritmo 1 Ejemplo de test unitario

```
@Test
void testEnqueue() {
    int size = queue.size();
    for (int i=1; i<=10; i++) {
        queue.enqueue(i);
        size++;
        assertEquals(size, queue.size());
    }
}
```

El uso de pruebas automáticas como apoyo a la evaluación de este tipo de asignaturas de programación supone una gran ayuda tanto para los estudiantes como para los docentes. Los estudiantes son capaces de comprobar si el código que están generando cumple con las especificaciones requeridas por el enunciado. Los docentes, por otra parte, son capaces de evaluar rápidamente la corrección del código de los estudiantes.

No obstante, aunque gracias a las pruebas automáticas el proceso de corrección puede haberse agilizado, sigue siendo poco eficiente: para evaluar a cada estudiante, el docente debe tener abierta el Aula Virtual, desde donde descargar los proyectos de los estudiantes y Netbeans para abrir cada uno de los proyectos de uno en uno. Para cada proyecto, debe compilar el proyecto y ejecutar todas las pruebas automáticas, revisando especialmente aquellas que fallen, cerrar el proyecto y puntuar al estudiante en el Aula Virtual.

Desde el punto de vista del estudiante, las pruebas automáticas también pueden llegar a ser de gran utilidad, sin embargo, no son capaces de comprobar, por ejemplo, si el estudiante está usando librerías no permitidas para la realización de las prácticas, lo que supondría que una práctica no fuese válida.

La corrección de este tipo de asignaturas implica que los docentes van a tener que ejecutar el código de los estudiantes para comprobar su correcta implementación, confiando en la ausencia de *malware* (software malicioso) en dicho código. Una manera de evitar que esto comprometiese el estado de la máquina del docente sería usar entornos seguros en los que testear los códigos de los estudiantes.

Sería interesante la existencia de una aplicación que intentara automatizar al máximo el proceso de corrección. De forma que los docentes de asignaturas de programación solo tengan que subir los proyectos desarrollados por los estudiantes a la aplicación y, como resultado, obtengan un informe global sobre el resultado de la ejecución de las pruebas, un informe individual sobre cada uno de los estudiantes y un informe anticopia para evitar fraude. De este modo, se ahorraría mucho tiempo de ejecución manual de las pruebas y se conseguiría una visión general de los resultados de la práctica o del examen. Además, se evitarían riesgos que comprometiesen la seguridad de la máquina en la que se ejecuta la aplicación.

Por otro lado, también sería interesante que los estudiantes, usando la aplicación, obtuviesen *feedback* (retroalimentación) real sobre la corrección de sus prácticas antes de entregarlas de forma oficial a través del Aula Virtual, de manera automática.

1.2. Objetivos

Los objetivos generales de la aplicación a desarrollar durante el proyecto son:

- Agilizar el proceso de corrección de prácticas y exámenes en asignaturas de programación, permitiendo la revisión visual del código, de los resultados de compilación y de la ejecución de los test unitarios.
- Ofrecer *feedback* a los estudiantes sobre el grado de validez de la práctica que está realizando antes de entregarla de forma oficial a través del Aula Virtual.
- Obtener un informe anticopia de las prácticas y exámenes entregados por parte de los estudiantes.
- Ejecutar los proyectos de los estudiantes desde un entorno seguro, evitando el riesgo de ejecutar código potencialmente peligroso.
- Ofrecer el servicio desde un entorno web que no requiera la descarga o instalación de ningún software adicional.

1.3. Estado del arte

Antes de empezar con el desarrollo de la aplicación que cumpla con los objetivos que se acaban de comentar, se ha hecho una revisión de las soluciones disponibles ya existentes:

- DOMjudge [6], es un software automático preparado para la realización de concursos de programación. Está preparado para procesar las respuestas de forma automática e implementa interfaces tanto para el jurado como para los equipos. Esta opción se descartó porque la configuración resultaba compleja, ya se había intentado con anterioridad y no permite la implementación del sistema anticopia.
- Acepta el reto [7], es otro sistema preparado para la realización de concursos de programación, al igual que DOMjudge. El problema principal de esta plataforma es que no permite subir problemas, en este caso prácticas y exámenes, por lo que el profesorado no podría usarla para corregir sus propios enunciados.
- Mooshak [8] es una herramienta web capaz de evaluar de forma automática la corrección de programas. La razón por la que se descartó Mooshak es que no funciona usando pruebas automáticas, sino que compara “manualmente” la salida estándar del programa con la salida esperada, normalmente, cadenas de caracteres o números enteros.

1.4. Estructura de la memoria

A continuación, se describe brevemente la estructura del resto del documento:

En el capítulo 2, Análisis del problema, se realiza una descripción completa del sistema que se quiere construir a partir de diagramas de casos de uso y de actividad. También

se realiza una descripción detallada de los requisitos funcionales y no funcionales del proyecto. Por último, se explica la metodología y cómo se ha adaptado al caso particular de este proyecto.

En el capítulo 3, Diseño e implementación, se explica cómo se ha desarrollado la aplicación que cumple con los objetivos y requisitos previamente explicados. Se realiza un análisis sobre las herramientas usadas durante el desarrollo y el porqué de su uso. Se explican, de forma detallada, cada una de las partes software que conforman la solución desarrollada. Finalmente, se expone en qué consiste el proceso de despliegue de la aplicación y en qué tecnologías se apoya.

En el capítulo 4, Métricas, se explican las medidas que se han ido tomando a lo largo del desarrollo del proyecto, relativas a tiempos de desarrollo, tiempos de ejecución y calidad del software.

Para finalizar, en el capítulo 5, Conclusiones, se presentan las conclusiones del trabajo realizado, así como sus posibles mejoras y ampliaciones en el futuro y las lecciones aprendidas durante el desarrollo del proyecto.

Capítulo 2

Análisis del problema

Durante este capítulo se intenta obtener un modelo lo más completo posible del problema que se desea resolver. Para ello, primero se introduce el sistema actual que utilizan los docentes para corregir las prácticas, se modela el problema a resolver con ayuda de diagramas UML (*Unified Modeling Language*, o Lenguaje Unificado de Modelado) y también se realiza una recapitulación de aquellos requisitos funcionales y no funcionales que se desea que la solución implemente. Por último, se explica cuál ha sido la metodología seguida durante el desarrollo, cómo se ha adaptado a este caso particular y cuáles han sido los motivos de su elección.

2.1. Descripción del sistema actual

Actualmente el sistema de corrección de la asignatura se basa principalmente en un *script* (secuencia de comandos) bash desarrollado por uno de los docentes de la asignatura. Para entender su funcionamiento se explican los siguientes términos:

- Directorio “Por corregir”: es un directorio que contiene todos aquellos proyectos de los estudiantes que todavía no han sido procesados por el *script*, es decir, sin corregir.
- Directorio “Corregidos”: es un directorio que contiene todos los proyectos de los estudiantes ya procesados.
- Proyecto de referencia: es un proyecto Java a partir del cual se corrigen los proyectos de los estudiantes. No contiene carpeta “src”, pero sí carpeta “test”. Es importante porque así el docente se asegura de que se ejecutan los mismos tests para todos los estudiantes. Gracias al proyecto de referencia, el docente también puede detectar si un estudiante estaba haciendo uso de librerías externas no permitidas en el enunciado de una práctica o examen porque generaría un fallo en la compilación.
- Directorio “Alumno temporal”: es el directorio en el que se descomprimen de uno en uno los ficheros que contienen los proyectos de los estudiantes.
- Directorio “Proyecto compilado”: directorio en el que se copian los ficheros del proyecto de referencia y el código fuente del estudiante. Sobre ese directorio se ejecutan las operaciones de compilación y test.

La Figura 1 muestra el diagrama de actividad del *script*.

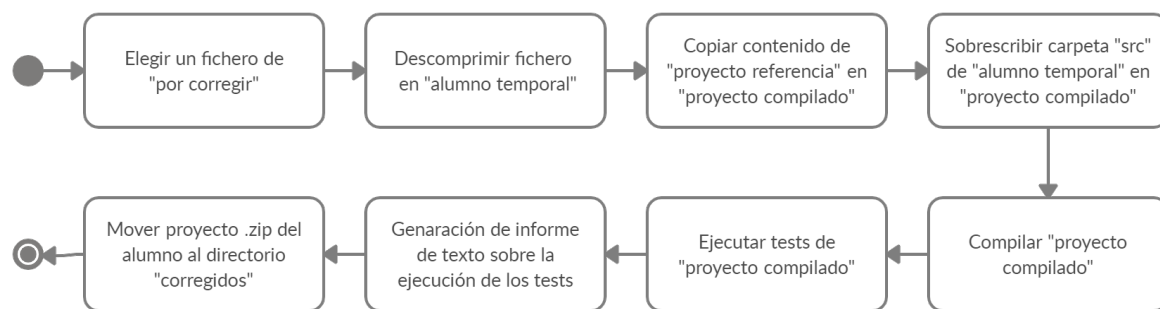


Figura 1: Diagrama de actividad del *script* usado por los profesores

De esta forma, si se quiere corregir un proyecto usando el sistema actual, se ha de abrir una terminal desde la que ejecutar el *script*, una sesión del Aula Virtual desde la que descargar los proyectos de los estudiantes y desde la que puntuar a los estudiantes tras la ejecución del *script*. También se ha de abrir una sesión de Netbeans (o un editor de texto cualquiera) para poder visualizar el código del estudiante. Además todo el sistema depende de que el sistema operativo desde el que se realiza la corrección sea una distribución de Linux [9].

2.2. Modelo del problema a resolver

En esta sección se intenta dar una idea general del modelo del problema mediante el uso de diagramas de casos de uso y diagramas de actividad para cada uno de los casos de uso especificados.

La Figura 2 muestra el diagrama de casos de uso cuando el usuario es un alumno. Como se puede observar, los estudiantes van a tener una única funcionalidad de la aplicación disponible, la autoevaluación de una de sus prácticas.

En cambio, el diagrama de casos de uso cuando el usuario es un docente es mucho más extenso, tal y como se observa en la Figura 3. Esto se debe a que los docentes, a parte de poder realizar la autoevaluación de un conjunto de prácticas, que sería la funcionalidad fundamental del sistema, también necesitan poder realizar tareas de gestión de la aplicación. De ahí que existan los casos de uso “Subir una nueva práctica”, “Actualizar una práctica”, “Borrar una práctica” e “Iniciar Sesión”.

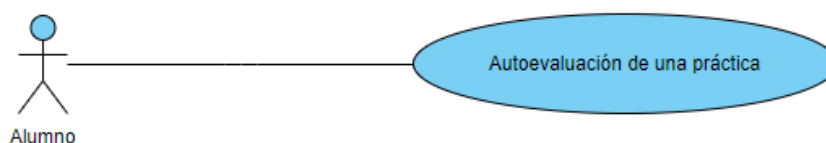


Figura 2: Diagrama de casos de uso para los alumnos

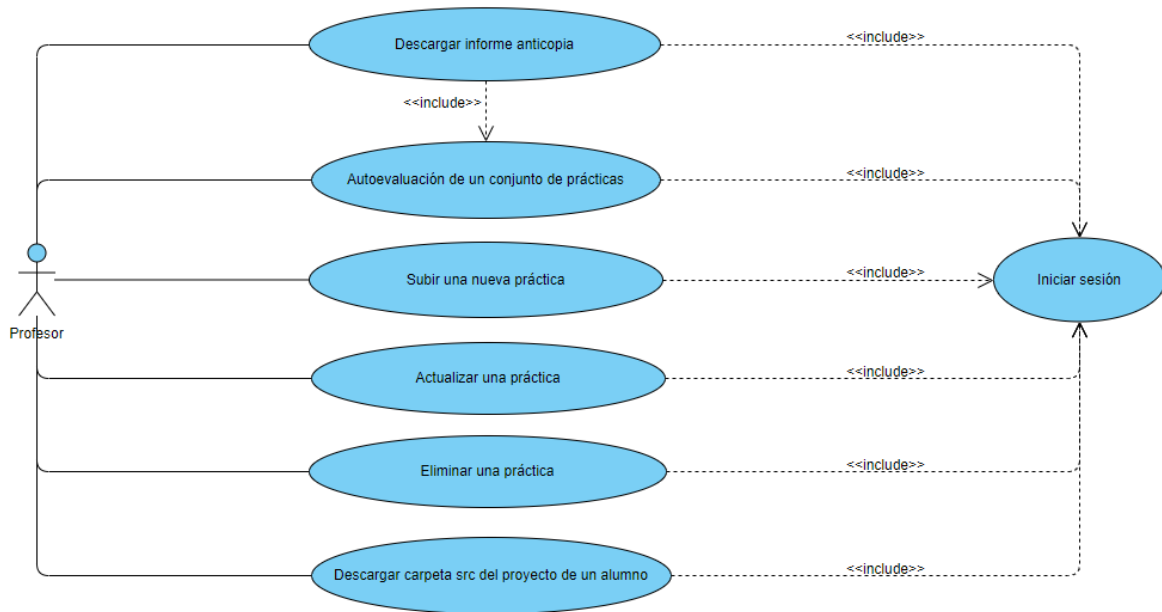


Figura 3: Diagrama de casos de uso para los profesores

De este modo, solo hay dos tipos de actores: estudiante y docente. La forma de diferenciar al tipo de actor durante el uso del sistema será si se realiza un inicio de sesión exitoso o no. Dicho de otra forma, el inicio de sesión es la puerta de acceso al uso de la mayor parte de las funcionalidades del sistema que no están disponibles para los estudiantes.

A continuación, en las Figuras 4, 5 y 6, se exponen una serie de diagramas de actividad que detallan algunos de los casos de uso introducidos previamente. Como se puede observar, siempre se realiza una comprobación del formato de los ficheros que se suben a la aplicación y, en caso de que el formato sea incorrecto, el usuario es redirigido al principio de la interacción.

En el caso de la “Autoevaluación de un conjunto de prácticas”, que hace referencia a la Figura 6, primero se visualizará un informe global, a partir del cual se podrá acceder a los distintos informes individuales de cada uno de los estudiantes.

Por último, matizar que el caso de uso “Descargar informe anticopia” de la Figura 3 depende del caso de uso “Autoevaluación de un conjunto de prácticas”, y se podrá acceder a esa funcionalidad desde el informe global mencionado previamente.

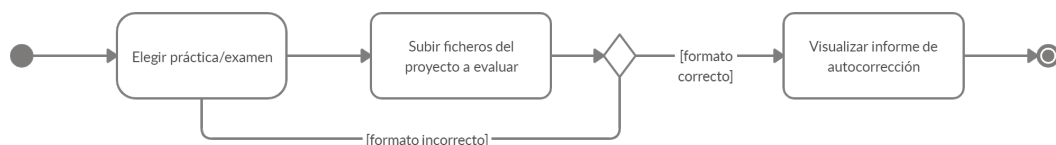


Figura 4: Diagrama de actividad para el caso de uso “Autoevaluación de una práctica”

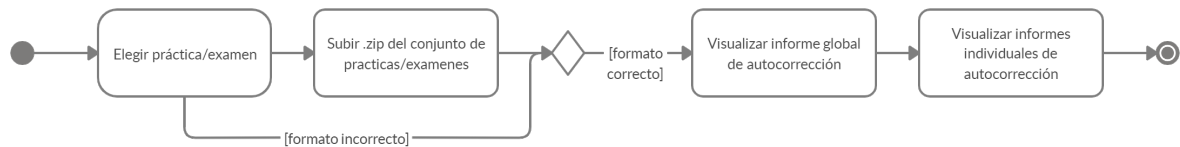


Figura 5: Diagrama de actividad para el caso de uso “Autoevaluación de un conjunto de prácticas”

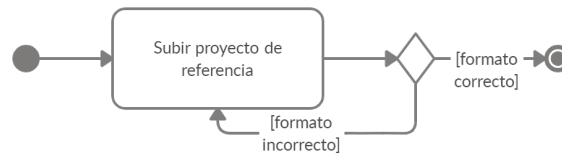


Figura 6: Diagrama de actividad para el caso de uso “subir una nueva práctica”

2.3. Requisitos

En esta sección se van a enumerar y detallar cada uno de los requisitos funcionales y no funcionales recogidos durante la fase de elicitación de requisitos.

2.3.1. Requisitos funcionales

Los requisitos funcionales definen las funciones que se desea que un sistema software incluya. Los requisitos funcionales de esta aplicación son los siguientes:

- El sistema debe permitir validar a un usuario. El usuario del sistema podrá iniciar sesión en la aplicación con su nombre de usuario y contraseña. La aplicación mostrará un mensaje de éxito de validación en caso de que los datos introducidos sean correctos. En caso contrario se debe informar al usuario de que las credenciales introducidas no son correctas.
- El sistema debe permitir realizar la autoevaluación de una práctica. Un usuario no validado debe poder elegir una práctica entre las disponibles en la aplicación y comprobar la corrección de un proyecto de su autoría.
- El sistema debe permitir realizar la autoevaluación de un conjunto de prácticas. Un usuario validado debe poder elegir una práctica entre las disponibles en la aplicación y comprobar la corrección de un conjunto de prácticas de manera simultánea.
- El sistema debe permitir crear una nueva práctica. Un usuario validado debe poder crear una nueva propuesta de práctica a la aplicación y que ésta sea visible para el resto de usuarios.
- El sistema debe permitir actualizar una práctica. Un usuario validado debe poder actualizar el proyecto de referencia de una aplicación ya subida a la aplicación.

- El sistema debe permitir eliminar una práctica. Un usuario validado debe poder eliminar una de las prácticas disponibles y que ésta deje de mostrarse como una opción disponible para el resto de usuarios.
- El sistema debe permitir descargar la carpeta “src” de un proyecto de estudiante. Un usuario validado debe poder descargar el código fuente de un proyecto desde el informe individual de un estudiante.
- El sistema debe permitir descargar un informe anticopia. Un usuario validado debe poder descargar un informe anticopia tras haber realizado la autoevaluación de un conjunto de prácticas.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que no se refieren a la funciones del sistema, sino a sus propiedades y restricciones del sistema en su totalidad. Los requisitos no funcionales del sistema son los siguientes:

- El sistema debe ser accesible a través de un navegador web.
- El sistema debe ser accesible independientemente del sistema operativo que use el usuario.
- El sistema debe contar con una interfaz gráfica amigable.
- El sistema debe poder recuperarse ante fallos. Debe ser posible reiniciar el sistema en caso de que deje de funcionar.
- El sistema debe poder funcionar de forma ininterrumpida durante largos periodos de tiempo.

2.4. Metodología

Para la realización del proyecto se ha seguido una adaptación del modelo en *waterfall* (cascada) descrito por Winston W. Royce en su ensayo de 1987 titulado *Managing the Development of Large Software Systems* [10].

Este modelo consiste en un procedimiento lineal que se caracteriza por dividir el proceso de desarrollo del proyecto en una serie de fases bien diferenciadas. Cada una de estas fases se ejecuta una única vez. Los resultados de cada una de estas fases sirven como punto de partida para la fase siguiente.

Originalmente, Royce propuso un modelo de siete fases, no obstante, hoy en día se suele aplicar una versión del modelo original basado en cinco fases: Análisis, Diseño, Implementación, Verificación y Mantenimiento.

- Análisis. En esta fase se analizan las necesidades de los usuarios finales del producto para determinar las características del software a desarrollar, y se especifica todo lo que debe hacer el sistema sin entrar en detalles técnicos.

- **Diseño.** En esta etapa se describe la estructura interna del software, y las relaciones entre las distintas entidades que lo componen.
- **Implementación.** En esta fase se programan los requisitos especificados previamente, así como las pruebas unitarias que comprueben el correcto funcionamiento del software.
- **Verificación.** Esta fase se centra en probar el funcionamiento la lógica interna del software y las funciones externas del mismo.
- **Mantenimiento.** Esta fase es la que se encarga de actualizar el proyecto cuando éste lo necesite, solucionando fallos, introduciendo nuevas funcionalidades... según las necesidades legales, por petición del cliente, etc.

Esta metodología separa el software a desarrollar de forma que sea posible que un equipo pueda trabajar de forma simultánea sin solaparse en sus actividades. En el caso particular de este proyecto, al haber una única persona encargada de realizar el desarrollo de todas las fases, se ha adaptado la metodología para agilizar el proceso.

La fase de análisis es peculiar porque los usuarios finales de la aplicación son también los tutores del proyecto. De modo que, en las tutorías para hablar del proyecto, en lugar de discutir solamente cómo debía comportarse el sistema, también se habló de cómo enfocar el diseño e implementación de la aplicación. Es decir, la fase de análisis y la fase de diseño se mezclaron en una única fase.

El resto de fases del proyecto se caracterizan por contar con un único programador, de forma que la complejidad del desarrollo del proyecto disminuye bastante. En cuanto a la fase de mantenimiento, comenzará una vez se empiece a utilizar el software. En el apartado 5.3 se comentan posibles mejoras a desarrollar en posteriores proyectos.

Por último, mencionar que, para la gestión del proyecto se ha usado la plataforma GitHub [11]. GitHub es un sistema de gestión de proyectos y control de versiones que permite trabajar en colaboración con otras personas desde cualquier máquina con acceso a Internet. Es una herramienta que permite ver a cualquier colaborador actual o futuro cuál ha sido el desarrollo de un proyecto. Durante la fase de mantenimiento se podrá descargar y modificar el software desde GitHub en caso de necesidad.

Capítulo 3

Diseño e implementación

En este capítulo se describe la solución creada que cumple con los objetivos y requisitos que se han planteado previamente el Capítulo 2.3. Para ello, primero se presentan cuáles han sido las herramientas utilizadas para la construcción de la solución y en qué medida han sido importantes. Después se realiza una descripción detallada de cada una de las partes software que componen la solución. Finalmente, se comenta cómo debe ser el proceso de despliegue de la aplicación.

3.1. Herramientas utilizadas

En esta sección se va a hacer un recorrido por todas las herramientas y tecnologías que se han usado, en mayor o menor medida, durante el desarrollo del proyecto y su posterior puesta en producción.

3.1.1. Java e IntelliJ IDEA

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Actualmente, es uno de los lenguajes de programación más populares [12] ya que permite el desarrollo de todo tipo de aplicaciones. Se eligió Java para este proyecto porque ya estaba familiarizado con el desarrollo de aplicaciones web en este lenguaje.

IntelliJ IDEA [13] es un entorno de desarrollo integrado para el desarrollo de programas informáticos. Está desarrollado por la compañía JetBrains [14]. Es un entorno multi-lenguaje y multiplataforma pero especialmente pensado para desarrollar proyectos en lenguaje Java.

La versión Ultimate incorpora herramientas pensadas específicamente para el desarrollo web en Spring. Además, como las aplicaciones web requieren el uso de diversos lenguajes durante su desarrollo, el editor de texto de IntelliJ IDEA está pensado para soportar HTML, CSS, Javascript [15] e incluso ficheros Docker [16]. Esta versión es de pago, aunque JetBrains concede licencias gratuitas para estudiantes universitarios.

3.1.2. Maven y Spring Boot

Maven [17] es una herramienta software para la gestión y construcción de proyectos en Java. Es una herramienta de código abierto que se creó con el objetivo de simplificar los procesos de compilación y generación de ejecutables a partir del código fuente, similar a Ant. Maven utiliza un fichero XML llamado Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Maven es una parte fundamental del proyecto porque es la base sobre la que trabaja Spring [18].

Spring es un *framework* (estructura conceptual y tecnológica de asistencia definida con artefactos o módulos concretos de software) de desarrollo de aplicaciones empresariales basado en tecnologías Java. Spring Boot [19] es una librería que facilita el desarrollo de aplicaciones con Spring ya que simplifica la configuración del proyecto y por lo tanto acelera el desarrollo. Toda aplicación Spring es también un aplicación Maven, pero no al revés. Spring es el *framework* de Java utilizado para el desarrollo de la aplicación web.

3.1.3. Linux

Linux (o GNU/Linux, más correctamente) es un sistema operativo de tipo Unix [20] multiplataforma, multiusuario y multitarea. Linux es el sistema operativo por excelencia en la red en servidores y supercomputadores a nivel mundial. Es importante para este proyecto porque se quiere ejecutar la solución desarrollada en uno de los servidores web de los que dispone la universidad. Además, la *shell* (intérprete de comandos) de Linux pone a disposición de los desarrolladores una serie de instrucciones que pueden facilitar la gestión de ficheros dentro de la aplicación.

3.1.4. Bootstrap

Bootstrap [21] es un *framework* CSS desarrollado inicialmente en el año 2011 por Twitter [22] que permite dar forma a un sitio web mediante librerías CSS que incluyen tipografías, botones, cuadros, menús y otros elementos que pueden ser utilizados en cualquier sitio web.

Es una herramienta que permite la creación de interfaces web limpias y adaptables de forma rápida y sencilla. Es la herramienta elegida para el desarrollo de las interfaces web de la aplicación.

3.1.5. MySQL

MySQL [23] es un sistema de gestión de bases de datos relacionales de código abierto con un modelo cliente-servidor. Es el sistema elegido para gestionar la base de datos con la que cuenta la aplicación. MySQL es una herramienta de código abierto que cuenta con una gran comunidad de desarrolladores que se da soporte entre sí.

3.1.6. JPlag

JPlag [24] es un software antiplagio multilenguaje de código abierto. Permite detectar similitudes entre una serie de ficheros de código. Al ejecutar JPlag, éste genera un informe global sobre el plagio detectado en todos los proyectos analizados y un conjunto de informes individuales para cada uno de los proyectos analizados remarcando las posibles similitudes con el resto de proyectos analizados.

Permite modificar la sensibilidad de detección y elegir qué directorios específicos se quiere analizar. Además, funciona con varias versiones de Java [25] y otros lenguajes (C [26], C++ [27], C#[28] y Python [29]) y texto plano. Es una herramienta fácil de usar y rápida en su ejecución.

3.1.7. Docker

Docker es una plataforma de software que permite crear, probar e implementar aplicaciones rápidamente.

Docker empaqueta software en unidades estandarizadas llamadas contenedores. Dentro de ellos podemos alojar todas las dependencias que nuestra aplicación necesite para ser ejecutada: empezando por el propio código, las librerías del sistema, el entorno de ejecución o cualquier tipo de configuración. Desde fuera del contenedor solo se requiere el servicio de Docker ejecutando en la máquina local.

Los contenedores son la solución al problema habitual de moverse entre entornos de desarrollo, como puede ser una máquina local o en un entorno real de producción. Se puede probar de forma segura una aplicación sin preocuparse de que el código se comporte de forma distinta en función del entorno.

Los contenedores, además, conforman un sistema completamente aislado del sistema operativo que los contiene, de forma que si se ejecutase algún tipo de *malware* dentro del contenedor bastaría con eliminarlo y volver a ponerlo en marcha. Y dado que para desplegar un contenedor Docker no hace falta arrancar un sistema operativo, como en las máquinas virtuales tradicionales, este proceso tarda de unos pocos minutos a incluso segundos.

El servicio de Docker que se ejecuta en el ordenador se encarga de traducir las peticiones que realiza el contenedor al sistema operativo nativo, de forma que no hace falta un hipervisor (programa que actúa a modo de interfaz entre los sistemas operativos huésped y anfitrión en una máquina virtual) que actúe a modo de interfaz, y tampoco hace falta arrancar un sistema operativo completo dentro de los contenedores. El servicio de Docker también se encarga de reservar más o menos memoria para los contenedores de forma dinámica.

Los contenedores Docker son esenciales para la puesta en producción de la aplicación y evitar que el servidor que albergue la aplicación pueda corromperse por un ataque informático a la aplicación.

3.1.8. JRater

JRater [30] es el nombre que se le ha puesto al proyecto desarrollado para el otro Trabajo de Fin de Grado de mi doble titulación. Es la aplicación que implementa la lógica de este

proyecto.

Esta herramienta, dado un proyecto de referencia y un proyecto de estudiante, genera un informe de compilación y test, en lenguaje JSON [31], para dicho estudiante. Por otra parte, dado un proyecto de referencia y conjunto de proyectos de estudiantes, genera un informe global y un informe individual, también en lenguaje JSON, para cada uno de los estudiantes sobre compilación y ejecución de los tests. También implementa la opción de generar un informe anticopia mediante el uso de JPlag.

Para el caso de corrección de un conjunto de proyectos, JRater utiliza técnicas de programación concurrente, de forma que la corrección de los distintos proyectos se realiza de manera simultánea en distintos hilos de ejecución.

Está diseñado para funcionar en tándem junto con este proyecto.

3.2. Arquitectura del software

A lo largo de esta sección se va a describir cómo ha sido la etapa de diseño e implementación de todas las partes que conforman la solución. Primero, analizando el proyecto desde un punto de vista general para que se entienda cuál es la estructura del mismo y, finalmente, haciendo énfasis en cada una de sus partes.

3.2.1. Estructura del proyecto

Para el desarrollo del proyecto, al tratarse de una aplicación web, se ha usado una arquitectura Modelo-Vista-Controlador, o MVC. La arquitectura MVC se caracteriza por separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos, tal y como se aprecia en la Figura 7 y se explica a continuación:

- El Modelo, que contiene una representación de los datos que maneja el sistema, la lógica de negocio y sus mecanismos de persistencia.
- La Vista, o interfaz de usuario, que se encarga de mostrarle los datos al usuario.
- El Controlador, que actúa a modo de interfaz entre la vista y el modelo. Recibe las ordenes del usuario y se encarga de solicitar datos al modelo y comunicárselos a la vista. Es la lógica de la aplicación.

Este tipo de arquitectura está muy extendida y existen multitud de *frameworks* para ella en los principales lenguajes de programación. En este proyecto se ha elegido el *framework* Spring MVC para el lenguaje Java.

En Spring MVC, los controladores son clases Java anotadas como `@Controller` que responden a las peticiones HTTP [32] que realiza el usuario desde el navegador web. Las Vistas, por otra parte, son plantillas HTML (que se pueden combinar con hojas de estilo CSS) que, junto con un motor de plantillas (en este caso Mustache [33]) muestran la información al usuario. El Modelo consiste en un objeto que contiene tuplas clave-valor. El funcionamiento de Spring sigue el esquema representado en la Figura 8.

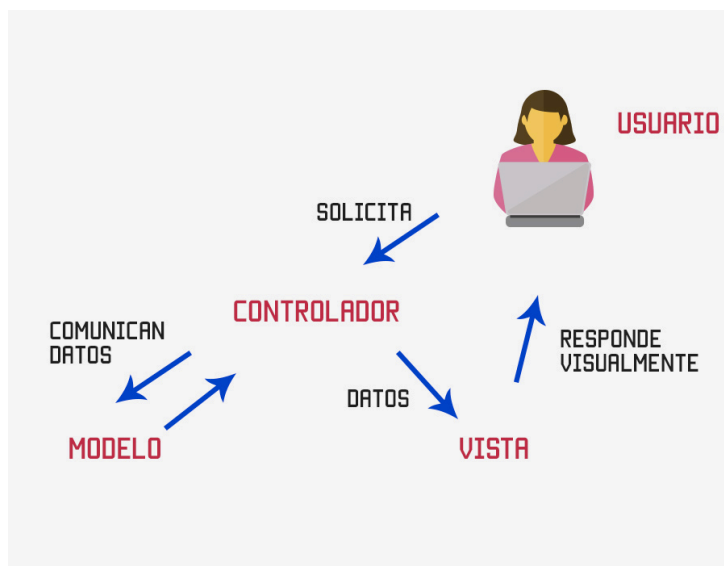


Figura 7: Esquema del funcionamiento de la arquitectura Modelo Vista Controlador

Para mantener el código del proyecto ordenado, se suele tener un Controlador por cada Vista (o plantilla). Además, para mejorar la legibilidad de los métodos de los controladores, la lógica se suele programar en otro tipo de componente software llamado Servicio. Los Servicios son clases Java que comunican la base de datos de la aplicación con los Controladores e implementan su lógica. Suele haber un Servicio por cada Controlador.

La última parte importante del proyecto, a parte de los Controladores, las Vistas y los Servicios, son las Entidades, almacenadas en la base de datos, y los Repositorios.

Las Entidades son clases Java que representan el modelo de datos de la aplicación. Al definir sus atributos hay que especificar también cuál es la relación entre ellas (1:1, 1:N ó N:M).

Por otra parte, los Repositorios son interfaces que definen cómo se realiza el acceso a la base de datos. Se implementan directamente como interfaz y es Spring internamente el que se encarga de inicializarlos de forma que se pueda acceder a los datos a través de ellos.

El diagrama de clases de la aplicación, con los componentes que se acaban de mencionar, aparece representado en la Figura 9.

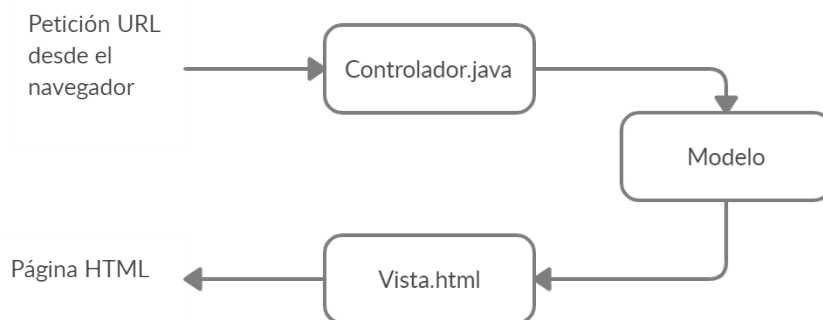


Figura 8: Esquema del funcionamiento de Spring MVC

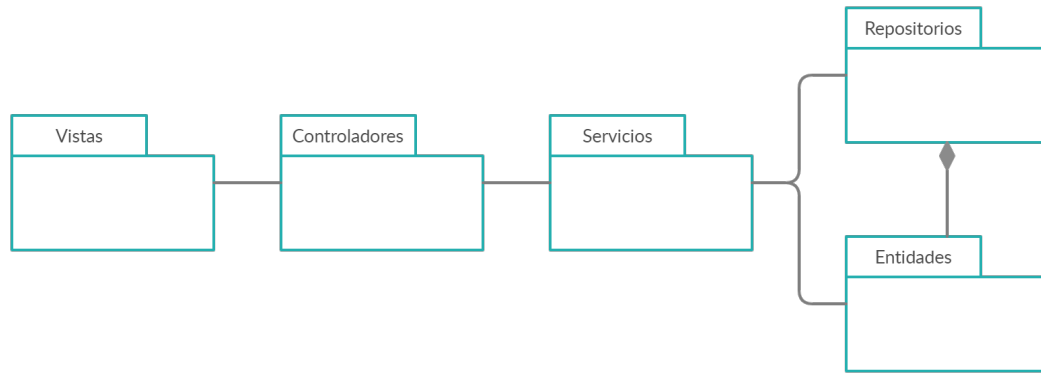


Figura 9: Diagrama de clases general del proyecto

3.2.2. Base de Datos, Entidades y Repositorios

La base de datos del proyecto es una base de datos relacional, definida a través de clases Java usando el estándar JPA [34], que permite interactuar con una base de datos de tipo SQL [35] desde un punto de vista orientado a objetos. JPA se encarga de traducir las clases Java a tablas de la base de datos y los métodos de los Repositorios mencionados previamente en instrucciones en lenguaje SQL.

Cuando se usa JPA, las clases que se quiere que se comporten como entidades en la base de datos deben anotarse como `@Entity` y los repositorios que incorporan los métodos que permitan el acceso a la base de datos deben anotarse como `@Repository`.

Por otra parte, las relaciones entre las distintas entidades debe realizarse en la definición de los atributos de cada clase. Por ejemplo, si se desea que entre la entidad A y la entidad B haya una relación 1:N unidireccional, se debe incluir un atributo de tipo lista en A, anotado como `@OneToMany`.

Las Entidades (o clases anotadas con `@Entity`) de la base de datos del proyecto son las siguientes:

- **Project.** Es el equivalente a un proyecto de referencia que sirve para corregir prácticas o exámenes. Incluye un identificador único, el nombre del proyecto de referencia, una descripción del mismo, una lista de informes individuales, el propio proyecto de referencia y el informe de antiplagio generado por JPlag.
- **Report.** Esta entidad almacena la información correspondiente a un informe individual de corrección de un estudiante. Además de un identificador, incluye información sobre el nombre y la fecha del proyecto, el nombre del estudiante, si hubo éxito durante la fase de compilación y una lista sobre el resultado de la ejecución de cada uno de los test del proyecto de referencia. También guarda los ficheros fuente programados por el estudiante.
- **Test.** Equivale a cada uno de las clases de test de un proyecto de referencia. En cada clase de test pueden incluirse uno o varios tests individuales. Esta entidad, además de un identificador, guarda información sobre el nombre de la clase de test y su ruta dentro del proyecto de referencia, el número de tests individuales y cuántos de dichos tests se han ejecutado correctamente y una lista con el resultado de la ejecución de los distintos test individuales contenidos dentro de la clase de test.

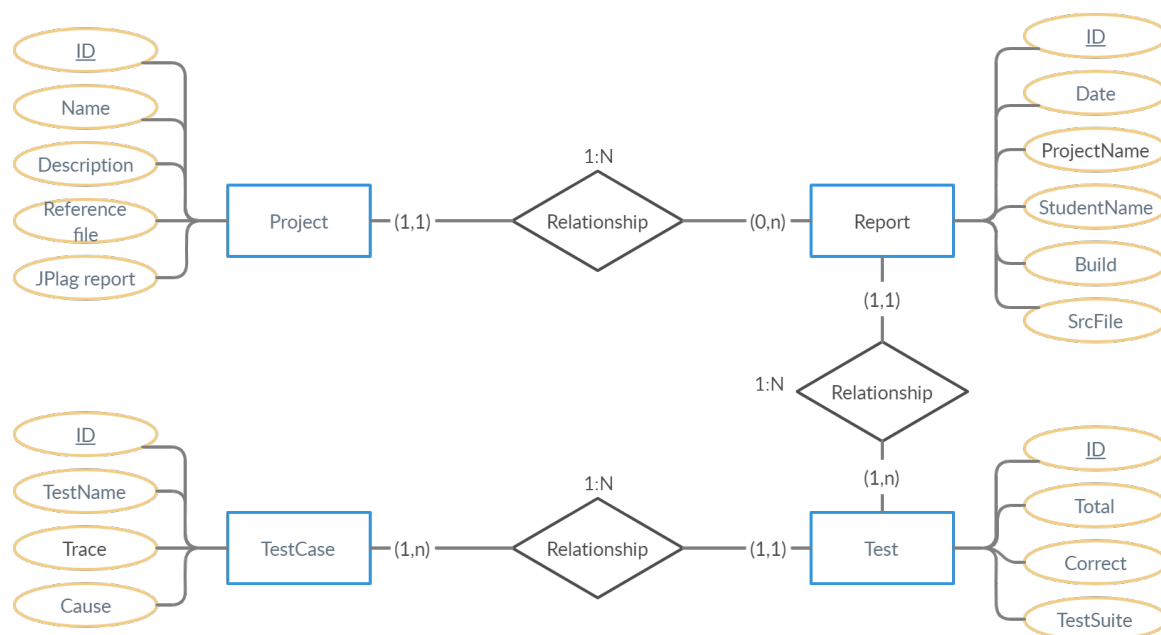


Figura 10: Diagrama Entidad-Relación de la base de datos del proyecto

- **TestCase.** Equivale a cada uno de los test individuales (o métodos anotados con `@Test`) que hay dentro de una clase Java de test. Esta entidad almacena un identificador, el nombre del método o test y, en caso de que la ejecución no fuese correcta, guarda el motivo por el que falló la ejecución y una traza del fallo.

La Figura 10 muestra el diagrama Entidad-Relación de la base de datos del proyecto, tal y como se acaba de definir.

Como se puede observar, se almacenan en la base de datos algunos ficheros y directorios. Por ejemplo, “SrcFile” en el caso de la entidad **Report**, o “ReferenceFile” en el caso de la entidad **Project**. Estos atributos se almacenan como LOB y deben anotarse con la etiqueta `@Lob` según el estándar JPA. Además, también hay que añadir la etiqueta `@Column(length=x)` siendo x la dimensión máxima del fichero en Bytes.

Todas las relaciones entre las entidades de la base de datos se han definido como unidireccionales, ya que realmente no es necesario que, por ejemplo, la entidad **Report** tenga una referencia a la entidad **Project** con la que está relacionada.

Se ha definido eliminación en cascada entre todas las entidades de la base de datos, ya que si un docente está interesado en borrar un proyecto de referencia, se considera que no le interesa guardar los informes de corrección, y por lo tanto tampoco el contenido de los mismos (los **tests** y los **testCases**). Si se borrara una entrada de la tabla **Report**, se borrarían también las entradas de las tablas **Test** y **TestCase** relacionadas con dicha entrada, y así sucesivamente.

Por último, falta mencionar cómo es la estructura de los repositorios. Según el estándar JPA, consisten en una interfaz Java que extiende de `JpaRepository<X, Y>`, siendo X una clase o Entidad e Y, el tipo de dato del identificador, en este caso particular, `long`. Por tanto, es necesaria una interfaz repositorio para cada entidad.

Algoritmo 2 Ejemplo de test unitario

```
public interface ProjectRepository extends JpaRepository<
    Project, Long> {

    Project findById(long id);

    Project findByName(String name);

    List<Project> findAll();
}
```

Las *queries* (peticiones o consultas a la base de datos) se definen como métodos que incorpora la interfaz. No hace falta definir el comportamiento del método, pero sí es necesario ser preciso con el nombre que se le pone, ya que JPA entiende lo que quiere hacer el método en función de su cabecera.

Por ejemplo, si se quiere obtener un objeto de tipo `Project` en función de su identificador, hay que incluir un método en la interfaz del repositorio cuya cabecera sea `Project findById(long id)`. Si se quiere un método que devuelva todos los objetos de tipo `Project` almacenados en la base de datos: `List<Project> findAll()`.

El Algoritmo 2 es un ejemplo del aspecto que tendría un Repositorio, según el estándar JPA.

La Figura 11 representa el diagrama de clases de los paquetes que contienen las entidades y los repositorios.

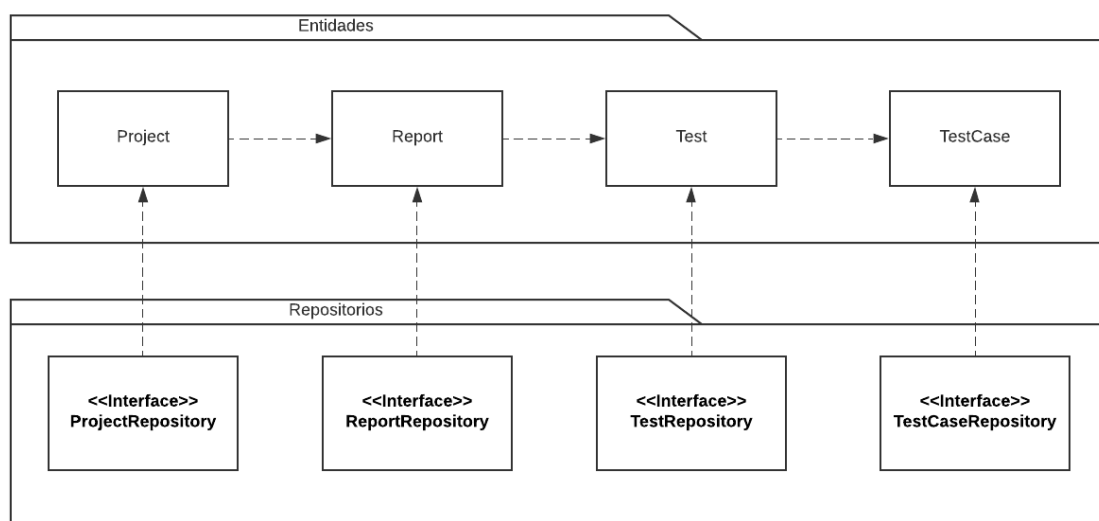


Figura 11: Diagrama de clases de las entidades y repositorios

3.2.3. Interfaz Gráfica y vistas

A continuación se introduce la parte del proyecto relativa a la interfaz gráfica, que en el modelo MVC de Spring corresponde a las Vistas. Para ello, primero se muestran capturas de pantalla de las distintas partes que tiene la aplicación, que ayudará a entender cómo es su funcionamiento interno y cómo están diseñados sus controladores y servicios, de los que se habla en el Apartado 3.2.4.

La Figura 12 se corresponde con la pantalla principal de la aplicación. Es lo que se le muestra al usuario según accede a la web. Desde esta pantalla el usuario tiene acceso a la pantalla de inicio de sesión y a la prácticas que este disponibles en ese momento. Como se vio en los casos de uso para un estudiante, la funcionalidad está bastante limitada.

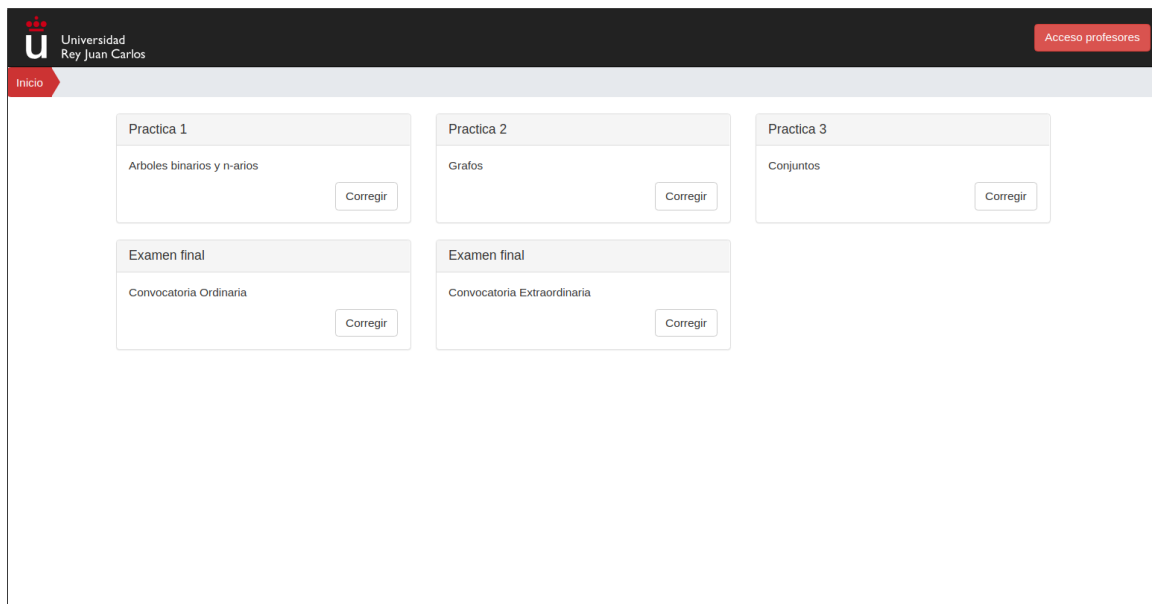


Figura 12: Pantalla principal sin inicio de sesión

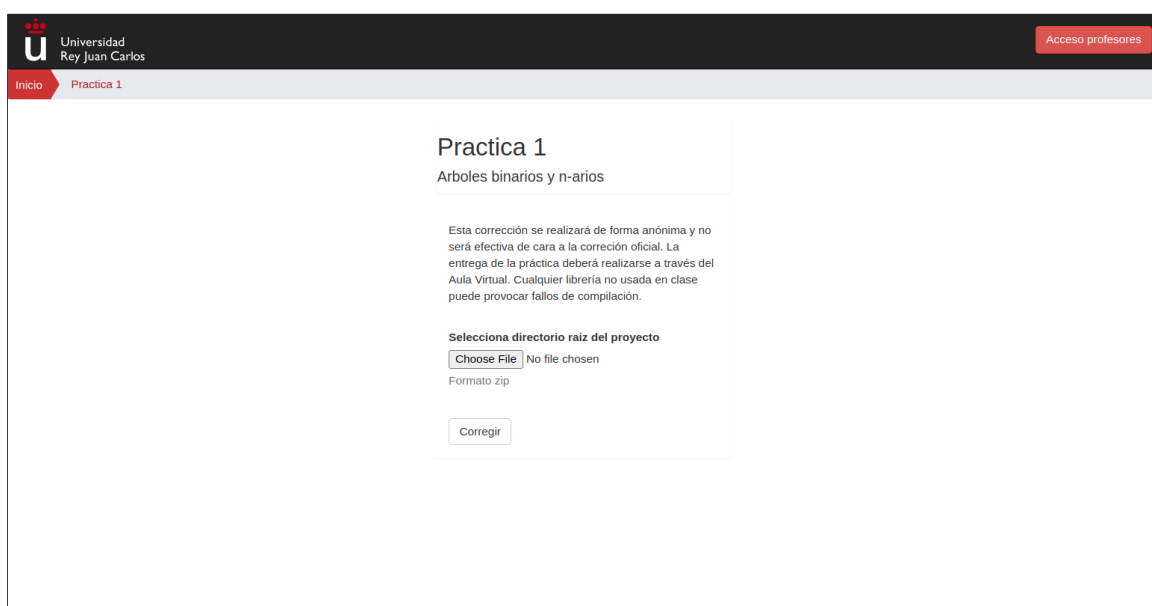


Figura 13: Pantalla de una práctica sin inicio de sesión

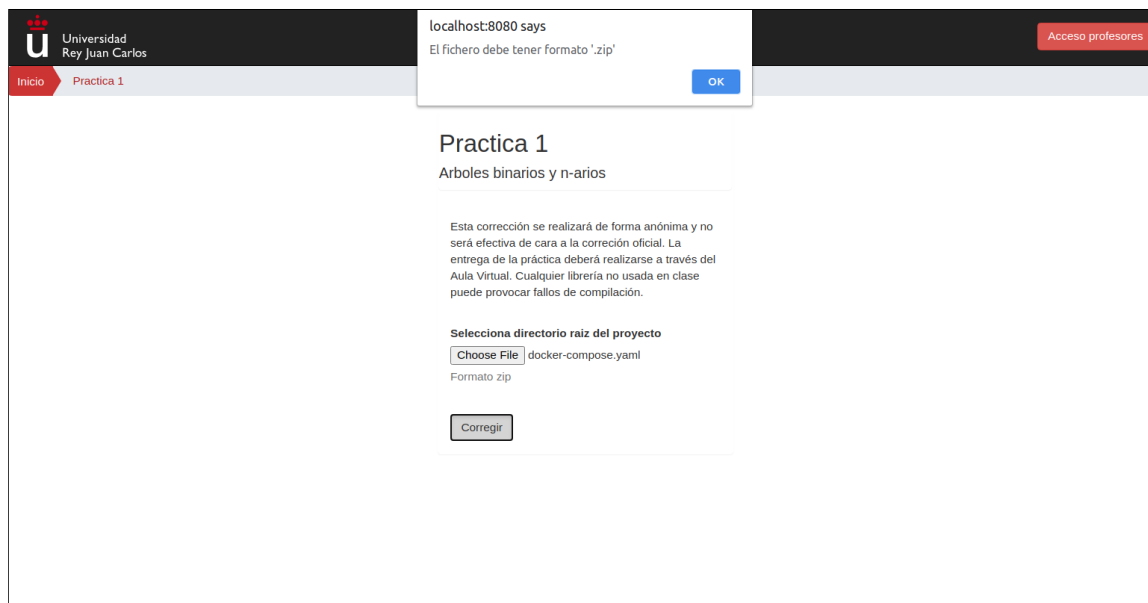


Figura 14: Pantalla de una práctica sin inicio de sesión. Cuadro de diálogo ante formato incorrecto

La pantalla de práctica, correspondiente a la Figura 13, se encarga de informar al usuario de las condiciones de seguir usando la aplicación y también de cuál es el formato correcto para la autoevaluación de una práctica. En caso de que el formato de entrega de una práctica sea incorrecto la aplicación lo detecta y se encarga de informar al usuario y de devolverle a la página de la práctica, tal y como se muestra en la Figura 14.

Una vez que el usuario escoge un proyecto con formato correcto y pulsa en el botón “corregir” la aplicación bloquea la pantalla mientras se realiza la autoevaluación y se informa al usuario de que el proceso puede tardar unos instantes. En la Figura 15 se muestra dicho cuadro de diálogo.

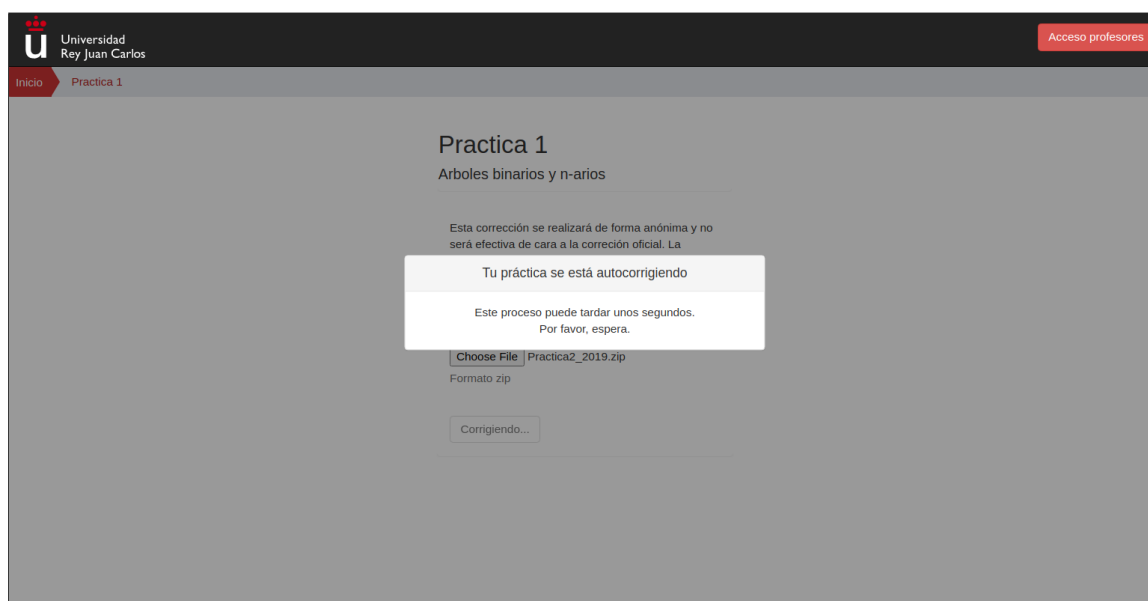


Figura 15: Pantalla de una práctica sin inicio de sesión. Bloqueo mientras se realiza la autoevaluación

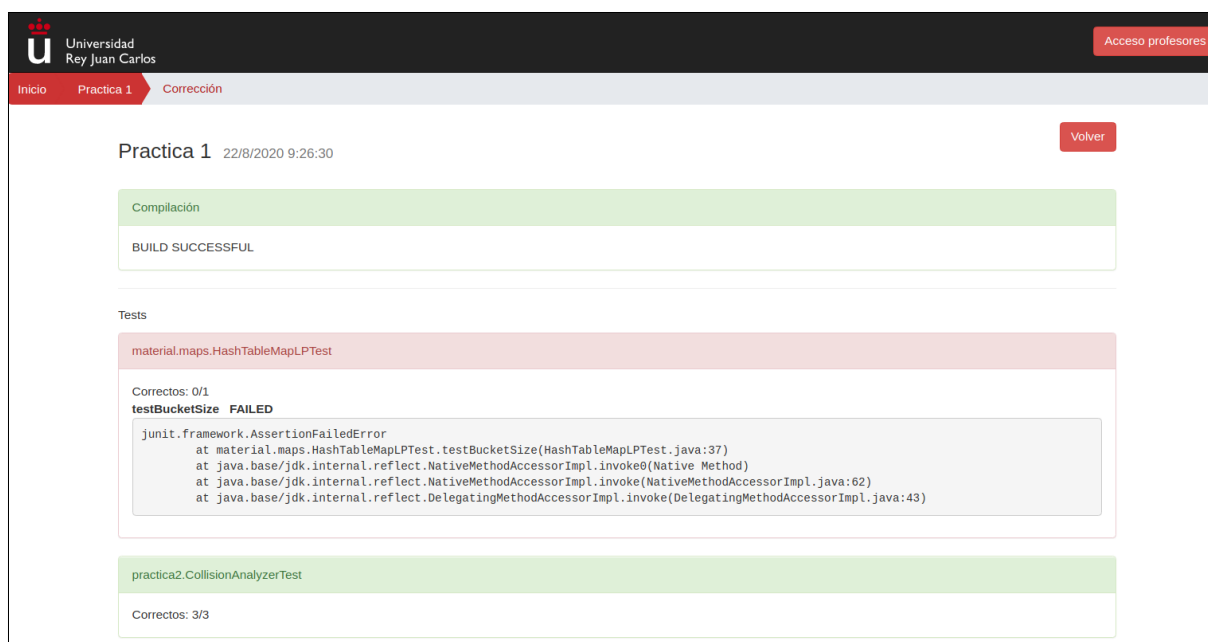


Figura 16: Pantalla de Informe individual sin inicio de sesión

La Figura 16 se corresponde con la pantalla de informe individual a la que los usuarios que no han iniciado sesión tienen acceso tras haber realizado la autoevaluación de una de sus prácticas. Muestra información acerca de las etapas de compilación y tests. En caso de fallo en alguna de las partes de la evaluación se muestra dicha parte en un cuadro con borde rojo y también se muestra información detallada acerca del fallo.

Por otra parte, la Figura 17 muestra la pantalla de inicio de sesión, a la que se accede desde la pantalla principal pulsando el botón “Acceso profesores”.

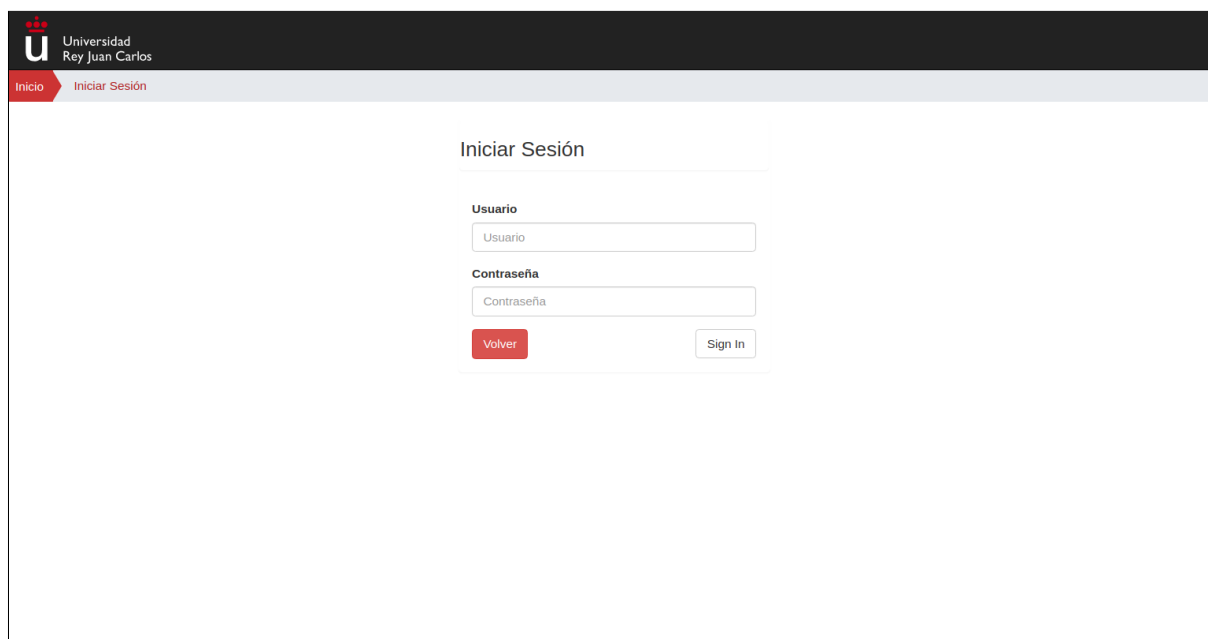


Figura 17: Pantalla de inicio de sesión

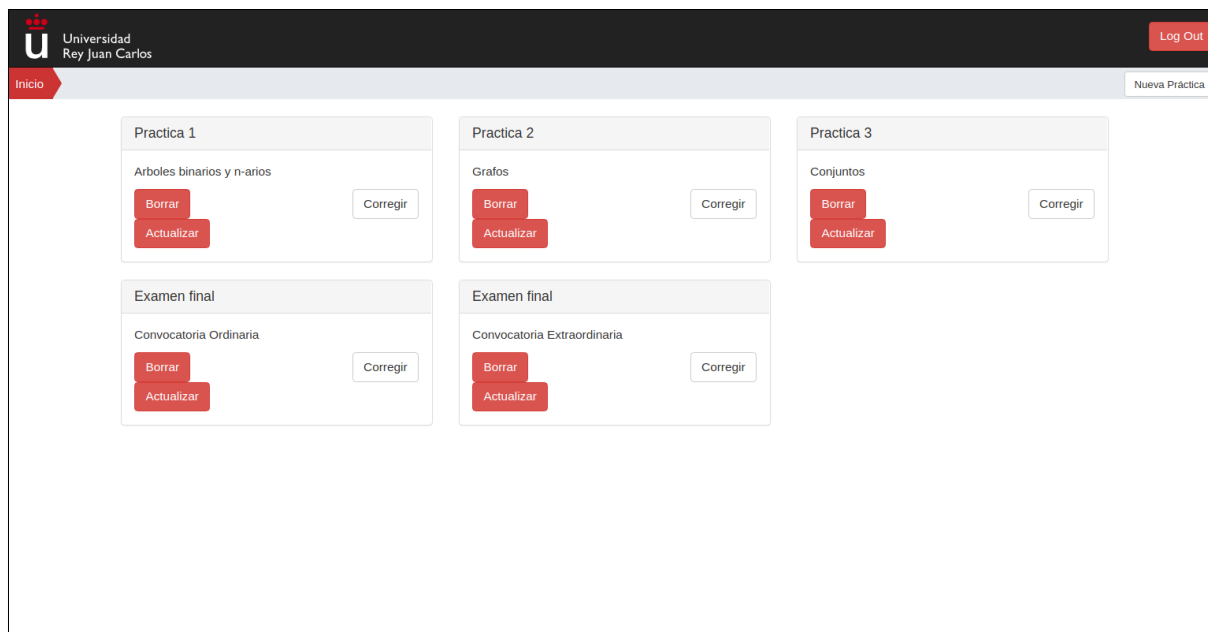


Figura 18: Pantalla principal con inicio de sesión

La Figura 18 muestra la pantalla principal, una vez se ha iniciado sesión. Esta pantalla es muy parecida a la de la Figura 12, pero se añaden las funcionalidades de “Nueva Práctica”, “Actualizar” y “Borrar”.

La Figura 19 muestra el cuadro de diálogo que aparece en la pantalla antes de borrar una de las prácticas para advertir al usuario en caso de que haya clicado el botón “Borrar” sin querer.

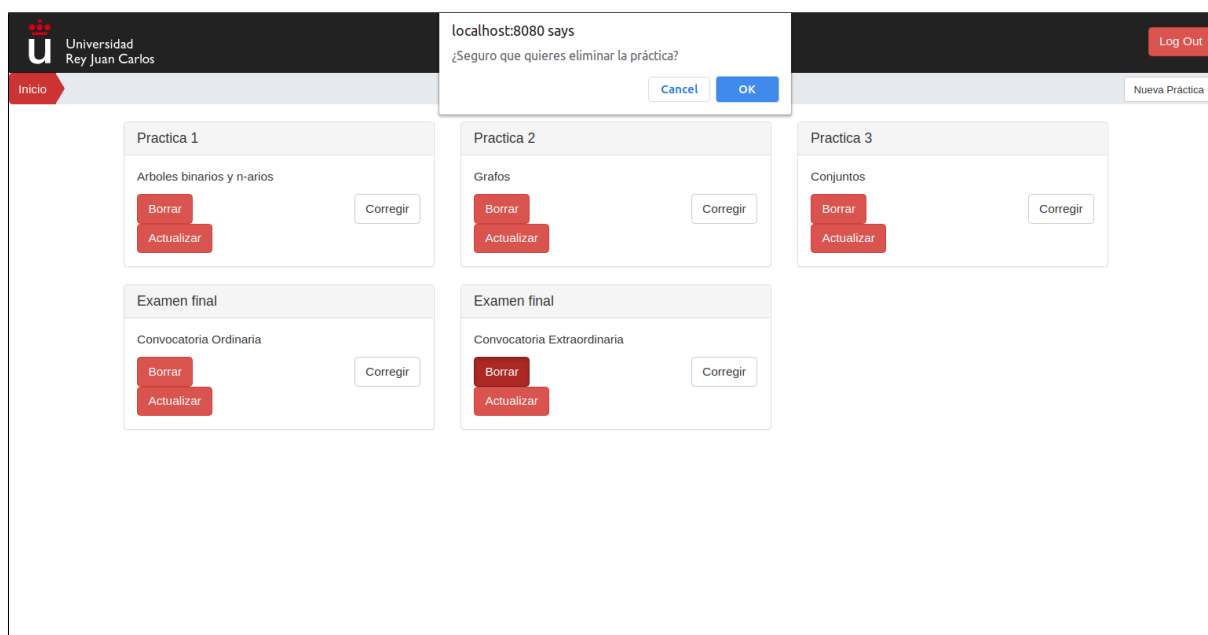


Figura 19: Pantalla principal con inicio de sesión. Cuadro de diálogo para borrar una práctica

The screenshot shows a web interface for updating a practice. At the top, there is a header with the university logo and name, and a 'Log Out' button. Below the header, there is a navigation bar with 'Inicio' and 'Actualizar Práctica'. The main content area is titled 'Actualizar práctica'. It contains a form with the following elements:

- A heading 'Seleccione nuevo proyecto de referencia'.
- A file selection button labeled 'Choose File' and the text 'No file chosen'.
- A label 'Formato zip'.
- An 'Actualizar' button.

Figura 20: Pantalla de actualización de una práctica

Desde la pantalla de la Figura 20 los docentes pueden actualizar el proyecto de referencia de una de las prácticas disponibles de la aplicación.

Por otro lado, la pantalla de la Figura 21 permite a los usuarios crear una nueva práctica que se muestre como disponible en la pantalla principal. Se pide un nombre y una breve descripción, y el proyecto de referencia en formato zip. Si el formato fuese incorrecto, la aplicación informaría al usuario.

The screenshot shows a web interface for creating a new practice. At the top, there is a header with the university logo and name, and a 'Log Out' button. Below the header, there is a navigation bar with 'Inicio' and 'Nueva Práctica'. The main content area is titled 'Nueva Práctica'. It contains a form with the following elements:

- A heading 'Nombre' with a text input field.
- A heading 'Descripción' with a text input field.
- A heading 'Seleccione proyecto de referencia'.
- A file selection button labeled 'Choose File' and the text 'No file chosen'.
- A label 'Formato zip'.
- A 'Subir' button.

Figura 21: Pantalla de Nueva Práctica

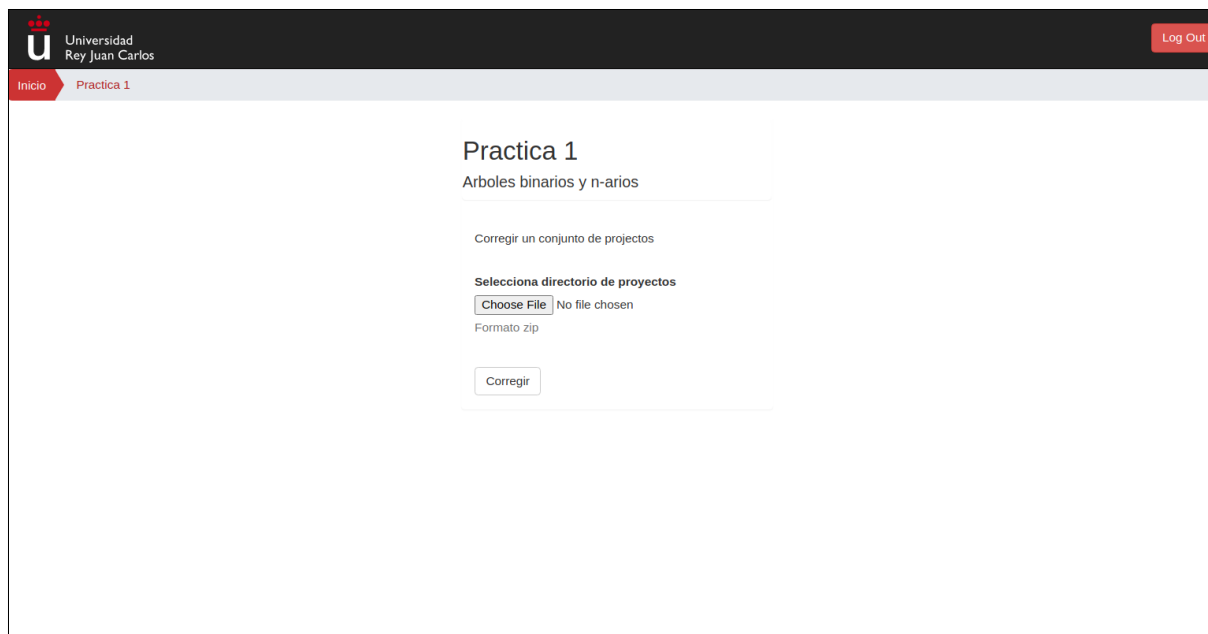


Figura 22: Pantalla de una práctica con inicio de sesión

Desde la pantalla de la Figura 22 los docentes pueden realizar la autoevaluación de un conjunto de prácticas de estudiantes de manera simultánea.

En la Figura 23 se muestra el resultado de la autoevaluación de un conjunto de prácticas de estudiantes. Se muestra una lista con los nombres de los estudiantes y un pequeño resumen del resultado de su evaluación individual. Desde esta pantalla el usuario puede acceder a cada uno de los informes individuales de los estudiantes y también puede descargar el informe anticopia que ha generado JPlag.

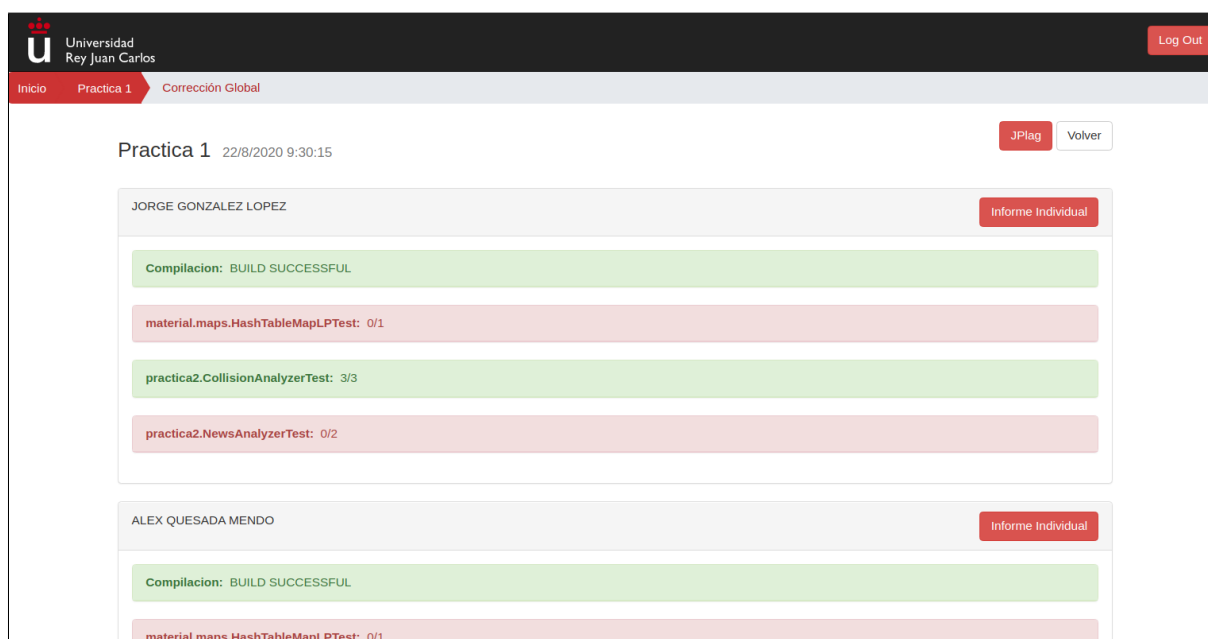


Figura 23: Pantalla de informe global

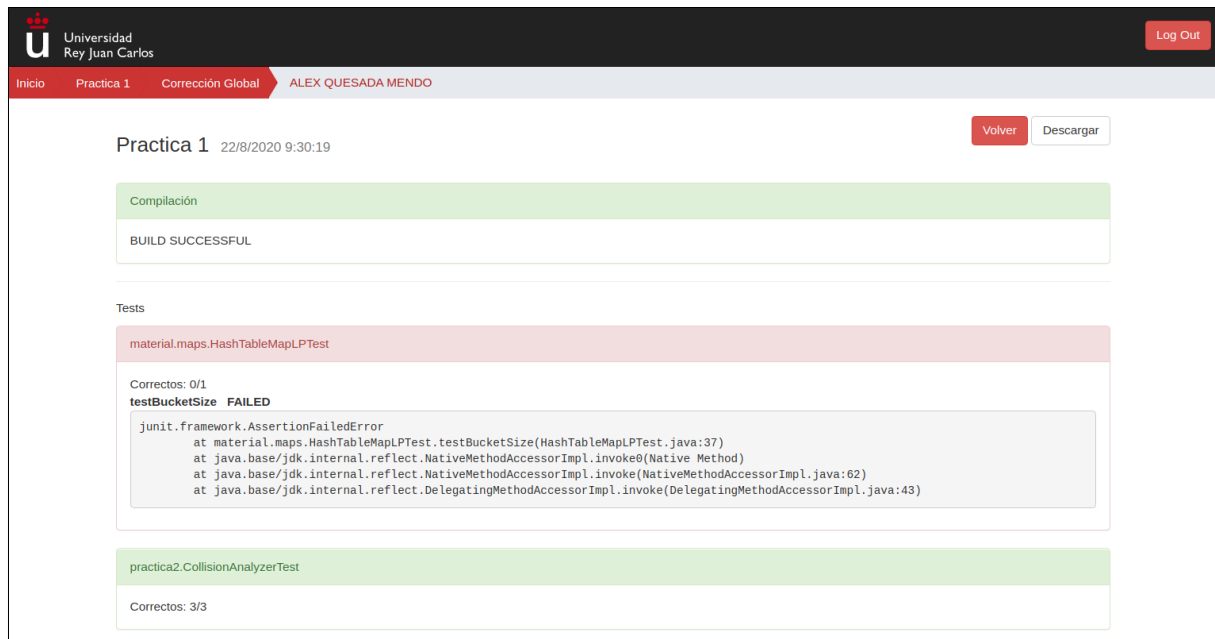


Figura 24: Pantalla de informe individual con inicio de sesión

Desde el informe global se puede acceder a los informes individuales, como el que se muestra en la Figura 24. Es similar al de la Figura 16, pero con la diferencia de que el usuario puede descargar el código fuente del estudiante.

La Figura 25 muestra el diagrama de actividad de la interfaz gráfica de la aplicación.

Este diagrama muestra el posible flujo de trabajo para recorrer las distintas pantallas disponibles en la aplicación. En rojo se muestran aquellas pantallas que solo son accesibles para los usuarios que hayan iniciado sesión. Se puede observar que los estudiantes tienen

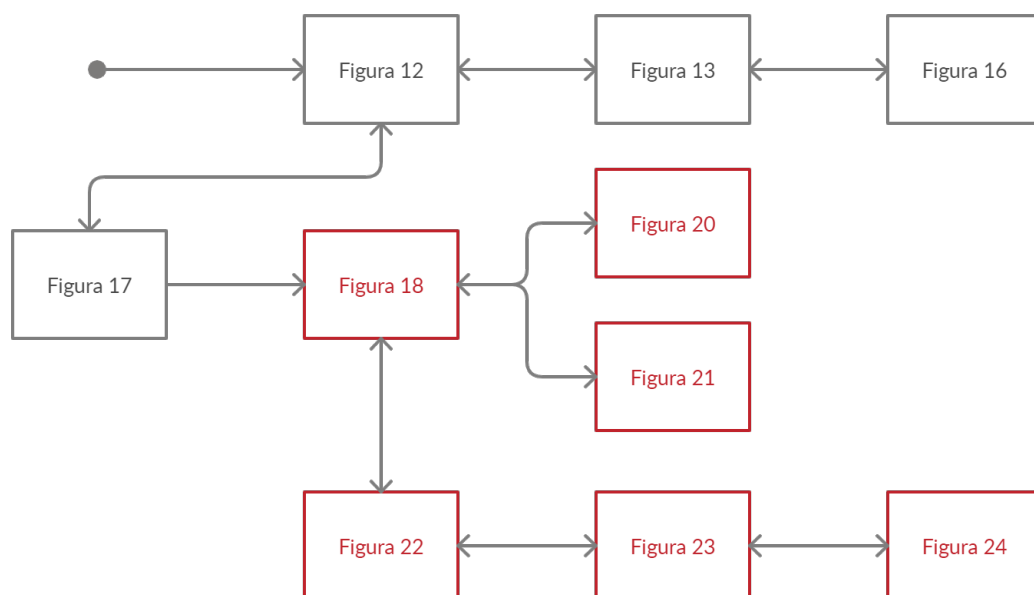


Figura 25: Diagrama de actividad entre las distintas pantallas de la aplicación

acceso a un porcentaje muy pequeño de todo lo que ofrece la aplicación, en comparación con los docentes.

Todas estas pantallas distintas se han generado a partir de dos plantillas HTML. Para conseguir que se muestre la información deseada en cada momento, se le pasa información a la Vista a través del Modelo. La Vista utiliza un motor de plantillas para obtener la información del modelo y generar la pantalla deseada en cada momento. Las dos plantillas utilizadas en la aplicación son:

- `report.html` es la plantilla encargada de mostrar las pantallas que muestran los informes globales e individuales (Figuras 16, 23 y 24).
- `app.html` es la plantilla que se encarga de mostrar el resto de pantallas (Figuras 12, 13, 14, 15, 17, 18, 19, 20, 21 y 22). Al principio también había una plantilla llamada `login.html` que se encargaba de mostrar la pantalla de inicio de sesión, pero posteriormente se integró en `app.html`.

Estas plantillas HTML están enriquecidas con hojas de estilo CSS y funciones Javascript para que el diseño sea amigable para el usuario. Para ello se ha usado Bootstrap, un *framework* CSS que permite diseñar una interfaz web a partir de la introducción de elementos prediseñados. Es una manera rápida y sencilla de crear interfaces de usuario limpias y adaptables.

Bootstrap ha diseñado una infinidad de clases CSS de forma que para añadir un elemento a la web, basta con etiquetar dicho elemento con la clase correspondiente.

Por ejemplo, el Algoritmo 3 muestra cuál es el procedimiento para insertar una barra de navegación anclada a la parte superior de la pantalla, en un fichero HTML.

Para poder usar Bootstrap basta con descargar los ficheros de la página oficial y referenciarlos al comienzo de los ficheros HTML.

3.2.4. Controladores y Servicios

Los controladores son los encargados de reaccionar ante las peticiones HTTP que realicen los usuarios. Los servicios se encargan de implementar la lógica detrás de los controladores. Esta separación no solo permite aislar funcionalidades sino que también aumenta enormemente la legibilidad del código para su futuro mantenimiento.

Se han implementado un total de cuatro controladores y tres servicios, relacionados tal y como muestra el diagrama de clases de la Figura 26.

A continuación se analizan las clases del diagrama una a una.

Algoritmo 3 Barra de navegación en HTML con ayuda de Bootstrap

```
<body>
  <div class="navbar navbar-static-top">
    \% Contenido de la barra de navegación
  </div>
</body>
```

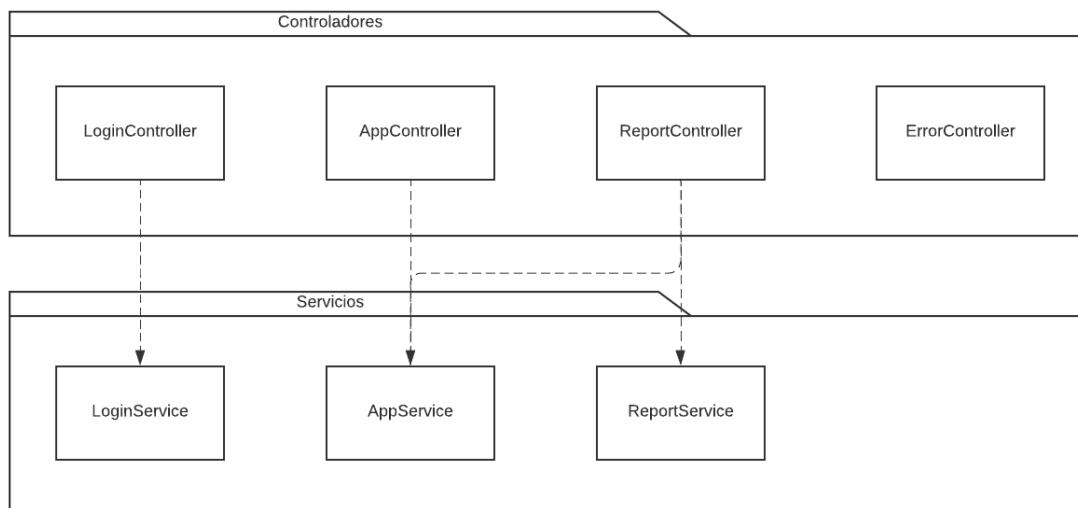


Figura 26: Diagrama de clases de los controladores y servicios

- **ErrorController** es un tipo de controlador especial que permite redireccionar al usuario cuando se produce algún fallo inesperado durante la ejecución de la aplicación. En el caso particular de este proyecto, si se llegase a producir un fallo, este controlador redirige al usuario a la pantalla principal. Es una buena medida de seguridad para que el usuario pueda seguir usando la aplicación con normalidad y, de este modo, se evita que el usuario pueda ver la traza del error que se imprimiría en la página de error que muestra Spring por defecto.
- **LoginController** y **LoginService** se encargan de gestionar la pantalla de inicio de sesión. Las funcionalidades principales de estas clases son: comprobar los credenciales que introducen los usuarios para iniciar sesión y terminar la sesión cuando un usuario sale de la aplicación o pulsa el botón de cerrar sesión.
- **AppController** controla las pantallas de las Figuras 12, 13, 14, 15, 17, 18, 19, 20, 21 y 22. Se encarga de gestionar las peticiones del usuario para la gestión de los datos de la aplicación y redirecciona a **ReportController** cuando el usuario desea realizar una autoevaluación.
- **AppService** se encarga de dar soporte a **AppController**. Incluye métodos que interactúan directamente con los repositorios, es decir, con la base de datos de la aplicación, para guardar, borrar o actualizar información. También se encarga de comprobar la validez del formato de los proyectos que quiere evaluar el usuario.
- **ReportController** es el controlador que se encarga de gestionar las peticiones de autoevaluación, tanto individuales como globales, y también aquellas peticiones que se realicen desde las pantallas de informe (Figuras 16, 23 y 24). Este es el controlador más complejo del proyecto, y se encarga de gestionar las peticiones de autoevaluación, las peticiones de revisión de informes y las peticiones de descarga de informes anticopia y código fuente de los estudiantes. **ReportController** es el encargado de cargar el modelo con los resultados de las autoevaluaciones.

- El **ReportService**, de manera análoga a **AppService**, se encarga de dar soporte a **ReportController**. Este controlador es el encargado de guardar en la base de datos los informes JPlag y el código fuente de los estudiantes con ayuda de los repositorios. También se encarga de ejecutar JRater para que se realicen las autoevaluaciones de las prácticas de los estudiantes y, la funcionalidad más importante, toma los ficheros JSON que resultan de la ejecución de JRater y los procesa para convertirlos en objetos de tipo **Report**, **Test** y **TestCase** que después se añaden al modelo.

3.2.5. Control de Sesión

Para el control de sesión del proyecto se usa una clase Java llamada **UserSession** y los llamados *Scopes* (ciclos de vida de componentes software de un sistema), una herramienta que incorpora Spring de forma nativa. Hay distintos tipos de **Scopes** en función de cómo queramos regular el ciclo de vida de cada componente software. Por ejemplo, si se quiere una única instancia de un componente se usa **Singleton Scope**, por otra parte, si se quiere una instancia por cada nueva petición HTTP, lo más adecuado es el **Request Scope**.

Para controlar la sesión, lo que se usa es el **Session Scope**, que crea una instancia del componente por cada sesión HTTP. Es decir, cada vez que un usuario ingresa en la aplicación se genera un objeto de tipo **UserSession** que almacena los datos de la sesión hasta que se cierra la aplicación; en ese momento Spring se encarga de eliminar dicho objeto.

La clase **UserSession** tiene dos objetivos fundamentales. Por un lado, controlar los inicios de sesión de la web, supuestamente reservados a los docentes. Por otro lado, almacena los datos respectivos a los informes de autoevaluación, tanto globales como individuales.

De esta forma, los usuarios que no han iniciado sesión pueden revisar los informes individuales de las autoevaluaciones que hayan realizado mientras que dure su sesión (ya que los informes individuales de usuarios que no inician sesión no se almacenan en la base de datos). El motivo de guardar los informes globales (y sus respectivos informes individuales) es ahorrar en el número *queries*, ya que es más rápido acceder a un atributo del objeto **UserSession** que a la base de datos.

3.2.6. Paquete de herramientas

Por último, existe un paquete en el proyecto dedicado a almacenar algunas de las funcionalidades que dan soporte al conjunto del proyecto. La clase **Utils** contiene un conjunto de métodos estáticos, es decir, que no necesitan que se instancie un objeto de la clase que los contiene para funcionar. Estos métodos realizan las funciones de compresión y descompresión de ficheros, conversión de un fichero JSON a un objeto de tipo **JSONObject** [36] y creación y borrado de ficheros y directorios.

Este paquete sirve fundamentalmente para mantener organizado el código y facilitar el futuro mantenimiento del proyecto.

3.3. Despliegue del software: Dockerización

La puesta en producción de la aplicación es el último paso de la fase de implementación. Para ello, tal y como se comenta previamente, se utilizan dos contenedores Docker que se comunican entre sí, uno alberga la base de datos y otro la aplicación web.

Los contenedores Docker se construyen a partir de imágenes. Una imagen es una “plantilla” que le indica al servicio de Docker cómo debe ser el contenedor a construir. Prácticamente todas las imágenes utilizan Linux como base para ejecutar las aplicaciones que contienen los contenedores Docker. Existen dos formas de construir un contenedor a partir de una imagen:

- La primera opción, que en este caso particular se usa para desplegar el contenedor que alberga la base de datos, es utilizar una imagen oficial desde los servidores de Docker. Estas imágenes oficiales están listas para contener, por ejemplo, un sistema operativo Linux o una base de datos MySQL sin tener que programar nada, desde el terminal del sistema operativo.
- La segunda opción para construir un contenedor a partir de una imagen es modificándola para adaptarla a las necesidades de cada caso particular. Para ello, hace falta un Dockerfile, un fichero que describe las características que tendrá el contenedor una vez desplegado. Los Dockerfiles se construyen a partir de una imagen oficial y, mediante comandos, se pueden configurar los puertos abiertos del contenedor, qué programas instalar, cómo será el sistema de ficheros. Este método también permite ejecutar comandos una vez finalizada la construcción del contenedor. Este es el método que se ha usado para desplegar el contenedor de la aplicación web.

El Algoritmo 4 es un ejemplo reducido de un Dockerfile que despliega una aplicación web de Spring, abriendo el puerto 8080 del contenedor.

El contenedor de la aplicación web de este proyecto se crea a partir de una imagen oficial de OpenJDK, en particular de la versión 11 (porque JPlag requiere Java 11 para funcionar). Sobre esta imagen base se instala **zip**, **unzip** y **ant**, necesarios para que la aplicación web y JRater funcionen correctamente. Después, se añaden los ficheros de configuración

Algoritmo 4 Ejemplo de fichero Dockerfile para desplegar una aplicación web

```
FROM openjdk:11

WORKDIR /webapp

COPY WebApp.jar /webapp/app.jar

EXPOSE 8080

RUN apt-get update
RUN apt-get install -y zip unzip ant

CMD ["java" , "-jar", "app.jar"]
```

de Netbeans, y los ejecutables (la aplicación web, JRater y JPlag). Finalmente se abre el puerto 8080 y se pone en marcha la aplicación web.

De esta forma ya están definidas los dos contenedores que componen el proyecto al completo. Pero en lugar de tener que lanzar ambos contenedores por separado y configurarlos individualmente, Docker proporciona un servicio llamado Docker-compose. Este servicio permite desplegar y configurar un conjunto de contenedores que vayan a a trabajar de forma conjunta a la vez. La principal ventaja de usar Docker-compose para el despliegue de la aplicación y de la base de datos, es que permite la creación de **Networks**. Las **Networks** son redes internas que la permiten comunicación entre distintos contenedores.

Para desplegar los contenedores usando Docker-compose, hay que crear un fichero `docker-compose.yml`. En este fichero se especifica la versión del servicio que va a usar para construir los contenedores, las posibles redes de interconexión entre los distintos servicios y por último cuáles son dichos servicios.

Algoritmo 5 Ejemplo reducido de fichero Docker-compose que despliega una aplicación web y una base de datos interconectadas

```
version: '2.1'

services:
  mysql:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: 1234
    ports:
      - "3306:3306"
    networks:
      jrater-network:

  spring:
    build: ./App
    ports:
      - "8080:8080"
    depends_on:
      mysql:
    networks:
      jrater-network:

networks:
  jrater-network:
```

Para asegurar que la base de datos está completamente desplegada antes de lanzar la aplicación, se especifica en el Docker-compose el comando que debe ejecutarse internamente en el contenedor, en este caso es un intento de conexión a la base de datos con el usuario y la contraseña de acceso.

El Algoritmo 5 es un ejemplo reducido del fichero Docker-compose que se utiliza para la puesta en producción de este proyecto.

La Figura 27 muestra el diagrama de despliegue del proyecto. Como se puede observar, la comunicación entre los dos servicios se realiza de forma interna dentro del Docker Host. Tanto el servicio que contiene la aplicación, como el servicio que contiene la base de datos se construyen a partir de una imagen base, OpenJDK y MySQL, respectivamente. Ambos servicios cuentan con un volumen cada uno. El volumen del servicio de la aplicación web contiene los directorios y ficheros sobre los que operan JRater y JPlag. Por otro lado, el volumen del servicio de la base de datos contiene los datos almacenados en ésta.

Para desplegar toda esta arquitectura, una vez están escritos los ficheros Dockerfile y Docker-compose, basta con ejecutar una única instrucción desde el terminal del sistema operativo: `docker-compose up --build`.

Automáticamente el servicio de Docker se encargará de descargar e instalar todo lo necesario y, a continuación, de desplegar ambos servicios. Esta facilidad para poner en marcha aplicaciones es otro de los motivos por los que Docker ha sido la herramienta escogida para realizar el despliegue y puesta en producción del proyecto.

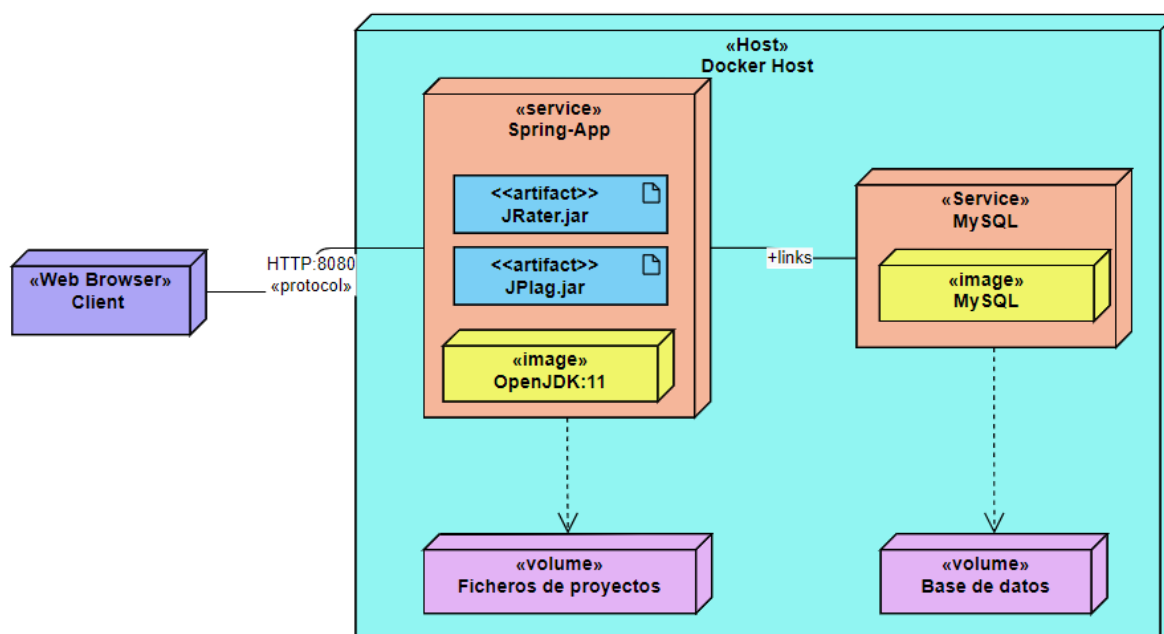


Figura 27: Diagrama de despliegue del proyecto

Capítulo 4

Métricas

Durante este capítulo se explican las distintas medidas que se han acumulado durante el desarrollo del proyecto, relativas a tiempos de ejecución y a tiempos de desarrollo de las distintas fases del proyecto. Por otra parte, también se mencionan métricas sobre la calidad y complejidad del software desarrollado.

4.1. Métricas temporales

Las métricas temporales hacen referencia a los tiempos de ejecución de la aplicación y a los tiempos de desarrollo de la misma.

Los tiempos de ejecución de esta aplicación son esencialmente los mismos que los tiempos de ejecución de JRater. Esto es así porque JRater es la herramienta que se encarga de soportar el peso de la lógica del proyecto, es el encargado de ejecutar los tests de los distintos proyectos que se evalúan y es donde hay más tiempos de espera.

Por otra parte, los tiempos de ejecución son poco representativos, ya que, dependiendo de la máquina en la que se ejecute la aplicación, estos pueden variar notablemente, especialmente en el caso de la corrección de un conjunto de proyectos. Esto se debe a que, como se menciona el apartado 3.1.8, JRater ejecuta la corrección de varios proyectos de manera concurrente en varios hilos de ejecución. Por eso, los tiempos en una máquina cuya CPU tenga, por ejemplo, 4 núcleos y 8 hilos, serán mucho más largos que en un servidor web con varias CPUs y más de 200 hilos.

No obstante, los tiempos de ejecución para los distintos casos de uso en un ordenador portátil con 8 GB de RAM, y un procesador intel i5-8250u (con 4 núcleos y 8 hilos) a 1,60GHz son:

- Autoevaluación de un único proyecto: 5,27 segundos.
- Autoevaluación de un conjunto de 15 proyectos: 12.78 segundos.
- Autoevaluación de un conjunto de 30 proyectos: 16.04 segundos.

El resto de casos de uso tienen tiempos de ejecución despreciables y no se han incluido.

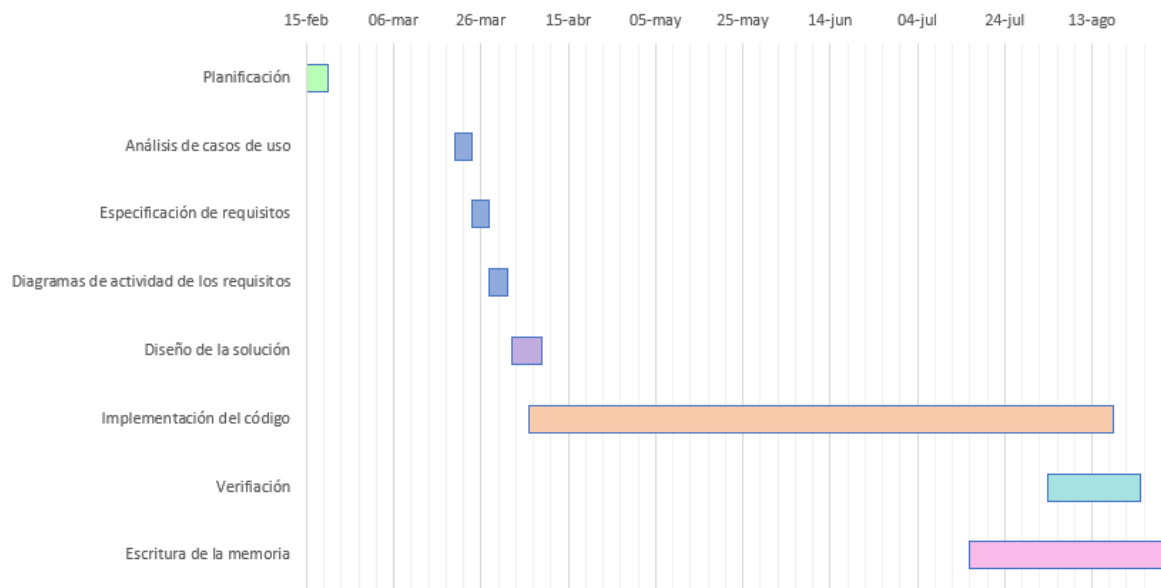


Figura 28: Diagrama de Gantt del proyecto

Por otra parte, están los tiempos de desarrollo del proyecto. La Figura 28 es un diagrama de Gantt que representa la duración de cada una de las distintas actividades en las que se ha dividido el proyecto.

Como se puede observar, desde el final de la fase de planificación hasta que empieza la fase de análisis de casos de uso, hay 30 días en los que no hay ninguna actividad en marcha. Esto se debe a que durante esos 30 días se estaba desarrollando el proyecto de JRater, necesario para que la aplicación web en la que consiste este proyecto funcionase correctamente.

Por último, la Figura 29 muestra la tabla de hitos asociada al diagrama de Gantt de la Figura 28.

Nombre actividad	Fecha inicio	Duración en días	Fecha fin
Planificación	15-feb	5	20-feb
Análisis de casos de uso	20-mar	4	24-mar
Especificación de requisitos	24-mar	4	28-mar
Diagramas de actividad de los requisitos	28-mar	4	01-abr
Diseño de la solución	02-abr	7	09-abr
Implementación del código	06-abr	134	18-ago
Verificación	03-ago	21	24-ago
Escritura de la memoria	16-jul	45	30-ago

Figura 29: Tabla de hitos asociada al diagrama de Gantt

4.2. Métricas sobre calidad del software

Para evaluar la calidad y complejidad del software se ha utilizado una herramienta llamada SonarQube [37]. SonarQube es una plataforma desarrollada en Java que permite realizar análisis de código de manera automatizada. Con esta herramienta se pueden realizar auditorías exhaustivas del código de un proyecto en pocos segundos, obteniendo como resultado una serie de informes que evalúan, desde distintos puntos de vista, la calidad del software desarrollado.

Para el análisis con SonarQube, la propia compañía recomienda utilizar tecnología Docker para poner en marcha un pequeño servidor web, que será el que se encargue de realizar el análisis [38]. Para ello, basta con descargar la imagen oficial de SonarQube mediante el comando `docker pull sonarqube` y, después, poner en marcha el contenedor Docker exponiendo el puerto 9000, desde el que se realiza la conexión al servidor, mediante el comando `docker run -d --name sonarqube -p 9000:9000 sonarqube`.

Una vez el servidor web está en funcionamiento, SonarQube permite realizar el análisis en sí de distintas formas, en función de la naturaleza de cada proyecto. En este caso particular, al ser un proyecto Maven, se ha usado el método específico para proyectos Maven. Según la guía oficial, basta con ir a la raíz del proyecto y ejecutar el siguiente comando: `mvn sonar:sonar -Dsonar.host.url=http://localhost:9000 -Dsonar.login=token`.

Nótese que se accede a la dirección `http://localhost:9000`, esto es porque se han enlazado los puertos 9000 del contenedor Docker y de la máquina local, de forma que accediendo al puerto 9000 de la máquina local, se accede al puerto 9000 del contenedor, donde está esperando peticiones el servidor web de SonarQube. El *token* (ítem) del segundo argumento se genera al crear un proyecto dentro del propio servidor web.

En la Figura 30 se puede obtener una vista general del resultado del análisis, tal y como lo muestra SonarQube.

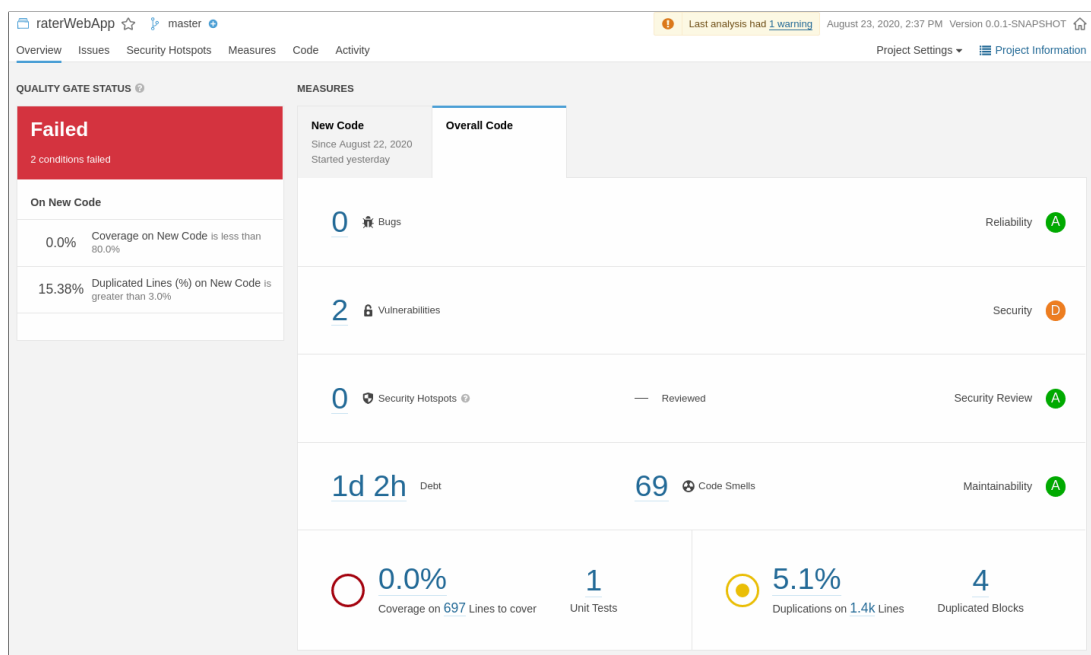


Figura 30: Análisis de calidad con SonarQube

Como se puede observar, se evalúa la calidad del proyecto desde el punto de vista de la fiabilidad o robustez, la seguridad y el futuro mantenimiento.

SonarQube marca como fallido el análisis porque no hay pruebas automáticas para este proyecto. Pero en el resto de apartados, obviando el de vulnerabilidades (ya que no se ha tenido en cuenta la posibilidad de un ciberataque), tienen una buena puntuación. Esto quiere decir que el proyecto resultante es fiable y fácil de mantener para futuros desarrolladores.

Además de puntuar dichos apartados, esta herramienta expone los motivos por lo que considera que hay partes de código que pueden acarrear conflictos y, además, explica cómo solucionar dichos problemas e incluye ejemplos de cómo hacerlo correctamente.

Por eso, se ha seguido un proceso iterativo en el que se iban resolviendo los problemas que esta herramienta encontraba de manera progresiva. Según se iban resolviendo los problemas de fiabilidad y mantenimiento, la puntuación SonarQube otorgaba aumentaba hasta llegar a la situación reflejada en la Figura [30](#).

Capítulo 5

Conclusiones

A lo largo de este capítulo se resumen los principales objetivos cumplidos y lecciones aprendidas durante la realización del proyecto. Finalmente, se propone un conjunto de posibles trabajos para continuar con el desarrollo del proyecto, orientado a futuros desarrolladores.

5.1. Objetivos conseguidos

El proyecto sobre el que trata esta memoria cumple con los objetivos descritos en la introducción de la misma respecto a servir como herramienta de autoevaluación para prácticas de programación, tanto para estudiantes como docentes.

En concreto, el proyecto desarrollado cumple con los objetivos generales planteados:

- Agiliza el proceso de corrección de prácticas o exámenes en asignaturas de programación. Para ello utiliza la herramienta JRater como base para poder corregir de manera automática y concurrente, conjuntos de prácticas o exámenes. Como resultado se obtiene un conjunto de informes que indican a los docentes la calidad del software desarrollado por los estudiantes.
- Ofrece *feedback* a los estudiantes sobre la corrección de sus prácticas, en forma de un informe detallado, antes de entregarlas de forma oficial. Ya que la aplicación web permite a los estudiantes realizar la autoevaluación de sus prácticas de manera anónima.
- Permite obtener un informe anticopia de las prácticas y/o exámenes entregados por los estudiantes. Se usa la herramienta JPlag para la generación de informes anticopia adaptados al tipo de proyectos que se realizan en la asignatura de EDA.
- Ejecuta el código de los proyectos de los estudiantes en un entorno seguro aislado. Para ello, se ha encapsulado la aplicación en un contenedor Docker, que además facilita la puesta en producción y la comunicación con la base de datos del proyecto.
- Permite acceder a la herramienta a través de un navegador web, de forma que no se requiere la descarga ni instalación de ningún software adicional. De esta forma, todo el proceso de corrección se puede realizar desde un navegador web, a elegir por el docente o estudiante.

5.2. Lecciones aprendidas

En esta sección se van a comentar algunas partes del proceso de desarrollo que más tiempo han consumido por falta de información y que puedan ser de utilidad para personas que estén desarrollando proyectos parecidos a este, o que utilicen tecnologías similares.

5.2.1. Netbeans y sus ficheros de configuración

La ejecución de los tests de proyectos Netbeans usando Ant fue una tarea complicada. Sucedió que, usando como ejemplo un proyecto Netbeans que fue facilitado los tutores y el comando `ant test`, desde la raíz del proyecto, no se ejecutaban los tests.

Se probó a crear un pequeño proyecto Netbeans de prueba, y sí que se ejecutaban los tests sin problema. Después se probó a abrir el proyecto de prueba (con el que se estaban teniendo problemas) en Netbeans y ejecutar los tests desde el propio entorno de desarrollo y sí que se ejecutaban correctamente. Así que se probó otra vez desde la consola usando Ant y en este caso sí se ejecutaban los test, sorprendentemente.

A continuación, se comprobó que cambios podía haber sufrido el proyecto tras haberlo abierto desde Netbeans. Lo que se encontró fue un fichero llamado `private.properties` que guardaba la siguiente información `'user.properties.file=/home/mayte/netbeans/8.2/build.properties'`. Esta ruta apuntaba a la localización de los ficheros de compilación de una máquina de la tutora de este proyecto, y cuando Ant buscaba dicho fichero no lo encontraba y, por lo tanto, no ejecutaba los tests.

La solución a este problema fue modificar ese fichero en los proyectos de referencia de la aplicación, apuntando a la ruta `'webapp/build.properties'` dentro del contenedor Docker donde se ejecuta la aplicación.

5.2.2. Usar SonarQube desde el principio del proyecto

SonarQube es una herramienta muy útil para solucionar fallos y malas prácticas no detectadas durante el desarrollo del proyecto. En el caso de este proyecto, se usó únicamente al final del desarrollo para el análisis sobre calidad de software del apartado 4.2. Una vez se realizó en el análisis hubo un proceso iterativo de corrección de fallos y nuevos análisis hasta que la calidad del proyecto fue aceptable.

Si se hubiese usado SonarQube desde el comienzo de la fase de desarrollo, el proceso iterativo para solventar los problemas que detecta la herramienta habría sido mucho más ágil, ya que se hubiera aprendido de las malas prácticas y errores en lugar de repetirlas.

Además, el proyecto contaría con un historial de la calidad del mismo a lo largo del tiempo, que permitiese, en caso de necesidad, volver a una versión en la que la calidad fuera mejor.

5.2.3. Requisitos de la aplicación antes de dockerizar

Por último, una buena práctica que hubiese ahorrado tiempo y varios intentos a la hora de dockerizar la aplicación es apuntar, a lo largo del desarrollo, cuáles eran los requisitos

software que necesitaba la aplicación para funcionar correctamente, y también qué versión de dichos requisitos.

En el caso particular de esta aplicación, se necesitaban los programas zip y unzip de Linux, Ant y Java Runtime Enviroment (JRE) versión 11 o superior, ya que JPlag no se puede ejecutar con una versión de Java inferior.

Al no haber apuntado estos requisitos, las primeras versiones del contenedor fallaban estrepitosamente al intentar ejecutar algunas partes de la aplicación.

5.3. Trabajos futuros

En esta sección se introducen una serie de posibles ampliaciones y mejoras del proyecto para futuros desarrolladores.

5.3.1. Ciberseguridad

Durante la etapa de planificación del proyecto se decidió obviar la implementación de medidas de ciberseguridad para prevenir posibles ataques, ya que la web no va a contar con información sensible, únicamente los proyectos de referencia. Además, se consideró que incorporar dichas medidas era añadir más horas de trabajo y complejidad a un trabajo que ya era extenso en sí mismo.

Algunas posibles medidas de seguridad a implementar son:

- Autenticación de todos los usuarios, no solo de los docentes. Actualmente cualquier persona puede conectarse a la web y usarla, sea o no sea estudiante de la asignatura de EDA del campus de Vicálvaro.

Para solucionar esto, habría que incorporar un servicio de inicio de sesión conectado a una base de datos que contuviese las credenciales de cada uno de los estudiantes que estén cursando la asignatura. De implementar esta medida, una buena práctica sería no guardar las contraseñas como texto plano sino utilizar una función Hash.

- HTTPS [39]. Actualmente la conexión a la aplicación web se realiza por HTTP, un protocolo antiguo que se comunica mandando la información plana, sin cifrar. Esto quiere decir que si alguien se colocase en medio de la conexión entre nuestro equipo y el servidor desde el que se ejecuta la web (por ejemplo, mediante un ataque *Man in the Middle* [40]) podría ver toda la información que se transmite entre ambos extremos.

Para implementar el protocolo HTTPS en proyectos de Spring, el propio *framework* ofrece la herramienta Spring Security que facilita la configuración del mismo. Solo hace falta un certificado (autofirmado o expedido por una Autoridad de Certificación) que contenga la clave que se usará para el protocolo TLS [41], necesario para HTTPS. También es necesario cambiar de puerto al 8443, en lugar del 8080.

5.3.2. Pruebas de Software

El principal motivo de que SonarQube marcara el proyecto como “suspense” era por la ausencia total de tests unitarios. Los tests unitarios, junto con el resto de tipos de pruebas, son una manera, sencilla de implementar, de facilitar y agilizar el mantenimiento de un proyecto.

Las pruebas automáticas son un tipo específico de prueba que requiere la ejecución de un código de pruebas. Sirven para verificar, de manera eficiente (por ser automáticas), el correcto comportamiento del software.

Ya que la mayor parte de la lógica de este proyecto se implementa en JRater, que es otro proyecto separado de este, las posibles pruebas de este proyecto deberían ser pruebas de sistema funcionales. Este tipo de pruebas consisten en automatizar las acciones que realiza un usuario mientras usa la web.

Para ello, la herramienta más extendida es Selenium [42], un *framework* orientado a la automatización de pruebas para aplicaciones web. Permite simular eventos de teclado y ratón, seleccionar partes de la aplicación a partir de sus identificadores... para, posteriormente, verificar si los resultados de una interacción eran los deseados.

5.3.3. Gestión de usuarios

La gestión de usuarios, que ya se ha introducido en el Apartado 5.3.1, podría ser útil para que solo usuarios de la universidad pudieran hacer uso de la aplicación. Por otra parte, si se ampliase la aplicación para que sirviese en otras asignaturas y campus, la gestión de usuarios serviría para que la aplicación “supiese” que proyectos de referencia debe mostrar a cada usuario.

Por otra parte, los docentes de distintas asignaturas podrían tener una interfaz única y distinta para cada asignatura, facilitando el uso de la herramienta.

5.3.4. Soporte para nuevos lenguajes

Actualmente, la aplicación solo está enfocada a trabajar con proyectos Java desarrollados en Netbeans. No obstante, las infraestructuras de este proyecto y de JRater están pensadas para que sean ampliables y puedan trabajar con distintos tipos de proyectos en Java u otros lenguajes.

Esto es posible ya que para la ejecución de los tests, lo que se hace es llamar internamente a Ant para que realice dicha ejecución. Siempre y cuando un lenguaje implemente tests unitarios, es susceptible de ser compatible con JRater y, por lo tanto, con esta aplicación web.

El soporte para nuevos lenguajes sería interesante para que esta aplicación sirviese en otras asignaturas que se desarrollan en otros lenguajes como C++ o Python, o con proyectos Maven, por ejemplo.

5.3.5. Permitir la descarga de los informes

La posibilidad de descargar los informes que se generan en la aplicación podría ser una buena nueva funcionalidad orientada tanto a los estudiantes como a los docentes.

Los estudiantes se aseguran tener una prueba de haber realizado una determinada práctica, por otro lado, los docentes podrían guardar un informe global de los resultados que les permitiese contar con un historial de la trayectoria de la asignatura a lo largo del curso.

5.3.6. Visualización del código de un estudiante junto a su informe de corrección

Actualmente, los docentes cuentan con un botón que les permite descargar la carpeta `src` del proyecto de un estudiante desde la pantalla de un informe individual. No obstante, la posible visualización del código del estudiante al lado de su informe individual permitiría a los docentes no necesitar ninguna herramienta adicional a la hora de corregir las prácticas o exámenes de los estudiantes.

Bibliografía

- [1] *Guías docentes de las asignaturas Programación Orientada a Objetos y Estructuras de Datos Avanzadas del curso 2019/2020*. Abr. de 2020. URL: <https://gestion3.urjc.es/guiasdocentes/> (vid. pág. 1).
- [2] *Web oficial de la Universidad Rey Juan Carlos*. Abr. de 2020. URL: <https://www.urjc.es/> (vid. pág. 1).
- [3] *Web oficial de Java*. Abr. de 2020. URL: <https://www.java.com/> (vid. pág. 1).
- [4] *Web oficial de Netbeans*. Abr. de 2020. URL: <https://netbeans.org/> (vid. pág. 1).
- [5] *Web del Aula Virtual de la Universidad Rey Juan Carlos*. Abr. de 2020. URL: <https://www.aulavirtual.urjc.es/> (vid. pág. 1).
- [6] *Web oficial de DOMjudge*. Abr. de 2020. URL: <https://www.domjudge.org/> (vid. pág. 3).
- [7] *Web oficial de Acepta el reto*. Abr. de 2020. URL: <https://www.aceptaelreto.com/> (vid. pág. 3).
- [8] *Web oficial de Mooshak*. Abr. de 2020. URL: <http://mooshak.inf.um.es/> (vid. pág. 3).
- [9] *Web oficial de Linux*. Mayo de 2020. URL: <https://www.linux.org/> (vid. pág. 6).
- [10] Winston W. Royce. "Managing the Development of Large Software Systems". En: *ICSE '87* (1987), págs. 328-338. DOI: <http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf> (vid. pág. 9).
- [11] *Web oficial de Github*. Mayo de 2020. URL: <https://github.com/> (vid. pág. 10).
- [12] *Los 10 lenguajes de programación más usados según Github*. Mayo de 2020. URL: <https://hardware360.net/2020/07/01/los-10-lenguajes-de-programacion-mas-populares-segun-github/> (vid. pág. 11).
- [13] *Web oficial de IntelliJ IDEA*. Mayo de 2020. URL: <https://www.jetbrains.com/idea/> (vid. pág. 11).
- [14] *Web oficial de JetBrains*. Mayo de 2020. URL: <https://www.jetbrains.com/> (vid. pág. 11).
- [15] *Web oficial de Javascript*. Mayo de 2020. URL: <https://www.javascript.com/> (vid. pág. 11).
- [16] *Web oficial de Docker*. Mayo de 2020. URL: <https://www.docker.com/> (vid. pág. 11).
- [17] *Web Oficial de Maven*. Mayo de 2020. URL: <https://maven.apache.org/> (vid. pág. 12).

- [18] *Web oficial de Spring*. Mayo de 2020. URL: <https://spring.io/> (vid. pág. 12).
- [19] *Web oficial de Spring Boot*. Mayo de 2020. URL: <https://spring.io/projects/spring-boot> (vid. pág. 12).
- [20] *Wikipedia Unix*. Mayo de 2020. URL: <https://es.wikipedia.org/wiki/Unix> (vid. pág. 12).
- [21] *Web oficial de Bootstrap*. Mayo de 2020. URL: <https://getbootstrap.com/> (vid. pág. 12).
- [22] *Web oficial de Twitter*. Mayo de 2020. URL: <https://twitter.com/> (vid. pág. 12).
- [23] *Web oficial de MySQL*. Mayo de 2020. URL: <https://www.mysql.com/> (vid. pág. 12).
- [24] *Repositorio oficial de JPlag*. Mayo de 2020. URL: <https://github.com/jplag/jplag> (vid. pág. 13).
- [25] *Versiones de Java*. Mayo de 2020. URL: <https://www.arquitecturajava.com/las-versiones-de-java/> (vid. pág. 13).
- [26] *Wikipedia del lenguaje C*. Mayo de 2020. URL: [https://es.wikipedia.org/wiki/C_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) (vid. pág. 13).
- [27] *Wikipedia del lenguaje C++*. Mayo de 2020. URL: <https://es.wikipedia.org/wiki/C%2B%2B> (vid. pág. 13).
- [28] *Wikipedia del lenguaje C Sharp*. Mayo de 2020. URL: https://es.wikipedia.org/wiki/C_Sharp (vid. pág. 13).
- [29] *Web oficial de Python*. Mayo de 2020. URL: <https://www.python.org/> (vid. pág. 13).
- [30] Alejandro Quesada Mendo. “Trabajo de Fin de Grado: JRater, aplicación de escritorio para la corrección automática de prácticas de programación”. En: (2020) (vid. pág. 13).
- [31] *Web oficial de JSON*. Mayo de 2020. URL: <https://www.json.org/json-en.html> (vid. pág. 14).
- [32] *Wikipedia del protocolo HTTP*. Jun. de 2020. URL: https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto (vid. pág. 14).
- [33] *Web oficial de Mustache*. Mayo de 2020. URL: <https://mustache.github.io/> (vid. pág. 14).
- [34] *Wikipedia del framework JPA*. Jun. de 2020. URL: https://es.wikipedia.org/wiki/Java_Persistence_API (vid. pág. 16).
- [35] *Wikipedia del lenguaje SQL*. Jun. de 2020. URL: <https://es.wikipedia.org/wiki/SQL> (vid. pág. 16).
- [36] *Web oficial de JSON-Simple*. Jun. de 2020. URL: <https://code.google.com/archive/p/json-simple/> (vid. pág. 28).
- [37] *Web oficial de SonarQube*. Mayo de 2020. URL: <https://docs.sonarqube.org/latest/> (vid. pág. 35).
- [38] *Análisis de código con SonarQube con la ayuda de Docker*. Ago. de 2020. URL: <https://www.federico-toledo.com/analisis-de-codigo-con-sonarqube/> (vid. pág. 35).

- [39] *Wikipedia del protocolo HTTPS*. Ago. de 2020. URL: https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto (vid. pág. 39).
- [40] *Wikidepia del tipo de ataque Man in the Middle*. Ago. de 2020. URL: https://es.wikipedia.org/wiki/Ataque_de_intermediario (vid. pág. 39).
- [41] *Wikipedia del protocolo TLS*. Ago. de 2020. URL: https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte (vid. pág. 39).
- [42] *Web oficial de Selenium*. Ago. de 2020. URL: <https://www.selenium.dev/> (vid. pág. 40).