



Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería del Software

Curso 2024-2025

Trabajo Fin de Grado

**EVOLUCIÓN Y PRUEBAS AUTOMÁTICAS DE LA
HERRAMIENTA EDUCATIVA IUDEX**

Autor: Iván Penedo Ventosa

Tutores: Raúl Martín Santamaría
Isaac Lozano Osorio

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a mi familia y amigos por su cariño y apoyo incondicional, y por brindarme un ambiente tranquilo y propicio para trabajar. Sin su ayuda, este trabajo no habría sido posible.

Agradezco a mis tutores por su orientación y apoyo durante todo el proceso de desarrollo. Sus comentarios y sugerencias fueron invaluable para la finalización de este trabajo.

Finalmente, me gustaría agradecer a todos los profesores y alumnos del grupo de programación competitiva, así como a la asociación de alumnos Dijkstraídos, por descubrirme un nuevo campo educativo al igual que divertido de la informática y que tanto me ha ayudado durante mi paso por la universidad.

Resumen

Con la motivación de avanzar en la implementación de una aplicación que permita la evaluación automática de código, este Trabajo de Fin de Grado tiene como objetivo mantener y mejorar la aplicación en desarrollo **iudex**. Esta aplicación ha sido realizada previamente por otros tres alumnos de la universidad, pero todavía no se encuentra en una versión estable, por lo que este trabajo busca avanzar en su desarrollo.

Las mejoras que se han realizado, se centran en su calidad y su mantenibilidad, mediante el uso de diferentes herramientas como analizadores de código estáticos, y en sus pruebas automatizadas de la aplicación, así como del ciclo ya existente de Integración Continua y Despliegue Continuo. En este apartado se busca reducir la deuda técnica de la aplicación para facilitar el desarrollo de la aplicación a futuros desarrolladores que busquen mejorarlo con nuevas funcionalidades o simplemente mantenerlo.

Además de esto, también busca integrar en la aplicación un sistema de autenticación delegada para poder utilizar el inicio de sesión institucional de la Universidad Rey Juan Carlos evitando la gestión de cuentas de manera individual por parte de la aplicación, y la inclusión de la tecnología de WebSockets para reducir la carga de trabajo de la aplicación en situaciones críticas.

Por otra parte, se analizarán las tareas realizadas con diferentes métodos y herramientas, así como un balance general de todos los resultados obtenidos durante la realización de este trabajo. Asimismo, se validará el correcto funcionamiento de todas las modificaciones realizadas en la aplicación. Finalmente, se propondrán algunas posibles tareas para trabajos futuros para poder continuar en el desarrollo de esta aplicación.

Palabras clave:

- Juez automático
- Pruebas automáticas
- Autenticación delegada
- WebSockets

Siglas y terminología

- **API:** Interfaz de Programación de Aplicaciones, del inglés *Application Programming Interface*, es el contexto por el que se comunican varias aplicaciones, comúnmente entre la interfaz y la lógica de negocio de una aplicación web.
- **Backend:** capa de acceso a datos de un programa software o sitio web que no es visible para los usuarios. Hace referencia a la lógica de negocio de una aplicación.
- **Bugs:** problemas en el código que hacen que un software se comporte de formas que no son esperadas por el usuario ni pretendidas por el desarrollador.
- **CI/CD (Integración Continua/Despliegue Continuo):** enfoque del desarrollo del software que enfatiza en la automatización, colaboración y mejora continua, buscando mejorar los procesos y entregar software de calidad más rápido.
- **Code smells:** signos que permiten medir la limpieza y mantenibilidad del código de una aplicación.
- **Framework:** aquella estructura que sirve de base sobre la que se puede construir una aplicación software.
- **Frontend:** interfaz de usuario de un programa software o sitio web; todo aquello con lo que interactúa el usuario.
- **JSON:** del inglés *JavaScript Object Notation*, formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos que consisten en pares atributo-valor y listas.
- **JWT:** del inglés *JSON Web Token*, estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un objeto JSON.
- **Middleware:** *software* que proporciona funcionalidades y servicios comunes a las aplicaciones, como la gestión de datos, mensajería, autenticación o gestión de API.

- *Mocks*: Objetos que hacen de sustitutos de los objetos de reales con el fin de simplificar, monitorizar o controlar el comportamiento de dependencias durante la realización de las pruebas.
- *Software*: programas e información operativa utilizados por un ordenador, o conjunto de instrucciones que este ejecuta.
- SSO: del inglés *Single Sign-On*, método de autenticación que permite a un usuario acceder a múltiples aplicaciones o sitios web después de iniciar sesión una única vez.
- Tests: en el ámbito de la programación, proceso para evaluar y verificar que un producto software o una aplicación haga lo que se supone que debe hacer, de modo que se puedan detectar fallos para poder corregirlos.
- TFG: Trabajo de Fin de Grado.
- ZIP: formato de archivo comprimido compatible, eficiente y ampliamente utilizado para empaquetar varios archivos y/o directorios en un solo archivo.

Índice de contenidos

Índice de tablas	XI
------------------	----

Índice de figuras	XIII
-------------------	------

1. Introducción	1
1.1. Contexto y alcance	1
1.2. Jueces de código	3
1.3. Estado del arte	6
1.3.1. Arquitectura de la aplicación	6
1.3.2. Desarrollo de la plataforma de la aplicación	8
1.3.3. Implementación de nuevas funcionalidades	9
1.3.4. Interfaz web para la aplicación	11
1.4. Estructura del documento	12
2. Objetivos	15
2.1. Objetivos principales	15
2.1.1. Mejora de la calidad	15
2.1.2. Mejora de las pruebas automatizadas de la aplicación	16
2.1.3. Autenticación delegada y gestión de roles	16
2.1.4. Implementación de <i>WebSockets</i>	16
2.2. Objetivos secundarios	18
2.3. Metodología empleada	18
3. Descripción informática	21
3.1. Herramientas y tecnologías utilizadas	21
3.1.1. Git	21
3.1.2. Spring	22
3.1.3. Single Sign-On (SSO) y Lightweight Directory Access Protocol (LDAP)	22
3.1.4. KeyCloak	23
3.1.5. Mockito	23
3.1.6. SonarCloud	23
3.1.7. Docker	24

3.1.8.	L ^A T _E X	24
3.2.	Implementación	24
3.2.1.	Mejora de métricas de calidad de Sonar	25
3.2.2.	Implementación de pruebas automáticas	28
3.2.3.	Desarrollo de sistema de autenticación	30
3.2.4.	Inclusión de la tecnología de WebSockets	35
4.	Resultados y validación	37
4.1.	Métricas de calidad	37
4.1.1.	Bugs	37
4.1.2.	Vulnerabilidades	38
4.1.3.	Líneas duplicadas	39
4.1.4.	Cobertura del código	39
4.2.	Validación de funcionamiento	41
4.2.1.	Pruebas automatizadas	41
4.2.2.	Autenticación y generación de tokens	42
4.2.3.	Autorización	45
4.2.4.	WebSockets	45
5.	Conclusiones y trabajos futuros	49
5.1.	Conclusiones	49
5.2.	Trabajos futuros	50
5.2.1.	Mejora de la interfaz web	50
5.2.2.	Mejora de la documentación de la API	50
5.2.3.	Realización auditorías y mejorar la seguridad	51
5.2.4.	Añadir generación de informes	51
5.2.5.	Conexión con calificaciones del Aula Virtual	51
	Bibliografía	52

Índice de tablas

1.1. Jueces de código con fines educativos	5
3.1. Valoración de las capas de transporte para la librería Docker-Java	34
4.1. Estadísticas de cobertura por pruebas automáticas de puntos de servicio de la aplicación	40
4.2. Comparativa de métricas de calidad	40
4.3. Comparativa de resultados de las pruebas automáticas	41

Índice de figuras

1.1.	Cronograma de creación de jueces de código con fines educativos .	5
1.2.	Esquema de arquitectura de módulos y servicios de la aplicación. .	7
1.3.	Captura de pantalla de la documentación de la API generada con Swagger	9
1.4.	Ejemplo de tabla de puntuaciones del SWERC 2022	10
1.5.	Vista principal de la interfaz gráfica realizada de la aplicación . .	12
2.1.	Ejemplo de peticiones a una API para comprobar el estado de los envíos	17
3.1.	Diagrama de clases del patrón Simple Factory con las clases DockerContainer	29
3.2.	Diagrama de flujo del proceso de autenticación en <i>iudex</i>	31
3.3.	Diagrama de secuencia del proceso de autenticación con el protocolo OAuth 2.0	32
4.1.	Secuencia de redirecciones de peticiones al solicitar un recurso protegido	42
4.2.	Vista del servicio de autenticación centralizada de la universidad .	43
4.3.	Respuesta del punto de servicio de inicio de sesión	44
4.4.	Respuesta de la petición con token de usuario autenticado	44
4.5.	Roles requeridos en los puntos de servicio de la clase APIContestController	45
4.6.	Repuesta de error al acceder a un recurso compartido sin el rol adecuado	46
4.7.	Repuesta correcta al acceder a un recurso compartido con el rol adecuado	46
4.8.	Validación de funcionamiento de WebSockets	47

1

Introducción

A continuación se va a analizar el contexto de uso de jueces de código automáticos, así como el contexto social que ha motivado la utilización de estos en el ámbito educativo. Posteriormente se analizarán estos jueces de código, se realizará una comparativa entre algunos de ellos, y sus diferentes aplicaciones reales, además de exponer el estado del arte de la aplicación que se va a evolucionar durante este Trabajo de Fin de Grado.

1.1. Contexto y alcance

En las últimas décadas, la sociedad se ha encontrado en un gran proceso constante y significativo de digitalización, donde las tecnologías de la información y comunicación, y más concretamente internet, se han incorporado a nuestra vida cotidiana, lo que ha supuesto un reto y han transformado profundamente las sociedades contemporáneas, afectando a diversos ámbitos como la economía, la política, la cultura y las relaciones sociales. Estos procesos se han acelerado y profundizado con el fenómeno de la globalización, que ha generado una mayor interconexión e interdependencia entre los actores y las regiones del mundo [1].

Con motivo de la reciente pandemia, este mismo proceso ha sufrido un gran incremento, puesto que mientras que en 2008 tan solo el 18 % de los españoles consideraba que internet era “esencial” en su vida. Por otra parte, en 2020 este porcentaje se ha incrementado hasta al 60 % e incluso las recientes estadísticas muestran que un 90 % de la población asegura utilizar internet a diario, así como un tercio afirma estar conectado casi todo el día [2].

Esta transformación digital también ha afectado a la educación, la cual tuvo que sobrevivir un tiempo con el cierre de todos los centros educativos, por lo que se recurrió a soluciones tecnológicas y el uso de internet para poder facilitar el aprendizaje a distancia.

Los cambios que se realizaron durante este período también han llegado a crear un impacto significativo en los estudiantes, puesto que las herramientas utilizadas les han facilitado un aprendizaje diario y en algunos casos la disponibilidad constante. Asimismo, las aplicaciones de computación en la nube permiten aprender en cualquier momento y lugar, disfrutando de las prestaciones elevadas en cuanto a acceso a la información, a la comunicación y a la productividad ofrecidas por el entorno tecnológico [3]. Es por ello que muchas instituciones educativas buscan nuevas maneras de ofrecer comodidades tecnológicas tanto para estudiantes como para docentes.

Los métodos de enseñanzas de carreras técnicas también han sufrido una importante transformación digital que ha provocado cambios en la manera de enseñar. En la actualidad, muchos docentes han implantado el método del aula invertida. Este método se suele utilizar en asignaturas más prácticas, pero que aun así contienen una base teórica, ya que, aprovechando algunas clases grabadas de la época de la pandemia, los profesores ponen a disposición de los alumnos vídeos explicativos de los contenidos teóricos de las asignaturas. Con esto, consiguen que el alumnado estudie la base de la asignatura a su propio ritmo en casa para que, cuando asistan a clase, puedan centrarse en la parte práctica, permitiendo así a los alumnos enfrentarse a un mayor número de ejercicios y, sobre todo, a poder realizarlos en clase con la explicación por parte del docente [4].

Paralelamente, también ha habido una creciente tendencia en cuanto a la intención de reducir el número de pruebas de programación a papel en el ámbito de las carreras técnicas, puesto que se encontraban inconvenientes tanto para docentes como estudiantes. Por una parte, los estudiantes necesitan organizarse antes de empezar a escribir en papel el código de la actividad que estén realizando, porque programar es una acción en la que es muy fácil equivocarse y que se necesite rectificar. Además de esto, si quieren introducir un fragmento de código dentro de otra sección que ya tengan escrita, acaban entregando una solución caótica y que, posteriormente, es difícil de interpretar. Es por esto, que la realización de pruebas a papel también afecta a los docentes, puesto que son ellos quienes tienen luego que revisar las entregas de los alumnos y, al estar realizados de esta manera, esta tarea se hace mucho más tediosa.

Tras buscar una alternativa a la realización de exámenes a papel, se han estado realizando algunos de estos en ordenadores, utilizando un editor de texto o un entorno de desarrollo integrado. De esta manera, el código que escriben los alumnos puede llegar a ser más legible, ya que también se puede permitir que este pueda poner el código en diferentes partes o archivos. Sin embargo, a muchos docentes les preocupa que esto pueda suponer un incremento en cuanto al uso de

métodos no legítimos durante la realización de las pruebas de evaluación, como podría ser la utilización de código que podrían tener ya escrito en los ordenadores. De esta manera, obtendrían una ventaja frente a otros alumnos.

Por otra parte, en la sociedad actual nos encontramos en un punto en el que los sistemas basados en juegos, o ludificados, hacen que las personas se entretengan realizando actividades no recreativas, como podrían ser las tareas educativas, y les motive más. Es por esto que, en numerosas situaciones, se crean aplicaciones que hacen que sea más entretenido y llevadero el proceso de aprendizaje. Uno de los casos más populares es el de *Duolingo*, una aplicación de aprendizaje de idiomas basada en niveles, en la que el usuario va ganando experiencia y progresando por las diferentes secciones y cursos que ofrece. Asimismo, también cuenta con una sección competitiva como las “Ligas” que incentivan a los usuarios a retarse entre ellos para comparar sus progresos semanales [5].

Por estos motivos, en el campo de la informática y la programación, se lleva mucho tiempo queriendo innovar en la manera de enseñar. Se han llevado a cabo diferentes técnicas, aunque no muchas han conseguido destacar. Es por esto, que con la idea de aplicar muchas de las ideas que se han estado proponiendo, están detrás de la idea de los jueces de código automáticos. Estos pretenden ser plataformas o aplicaciones en la que los alumnos puedan realizar, validar y entregar proyectos de programación. A continuación, se explica más en detalle sus características y opciones de uso, así como las ventajas que aporta frente a otros métodos más tradicionales.

1.2. Jueces de código

Los jueces de evaluación automática de código son herramientas que pueden ser de gran ayuda en el contexto de la educación de la programación, pues pueden mejorar significativamente la objetividad, rigurosidad y oportunidad con que se evalúan los códigos realizados por los usuarios. En estas aplicaciones, estos códigos recibidos son compilados y puestos a prueba en un entorno unificado con entradas y salidas predefinidas, lo que permite que se valide la correcta implementación de la tarea solicitada al usuario que ha realizado el ejercicio [6], así como otros parámetros que hacen que se valide si el acercamiento tomado para resolver el ejercicio es el adecuado, como el tiempo que tarda en ejecutarse o la memoria que requiere el programa para ejecutarse.

Estos jueces ya han encontrado su lugar en otros ámbitos como puede ser el laboral, donde se utilizan para automatizar las entrevistas de conocimiento técnico, donde el examinado realiza la prueba, entrega su solución a la aplicación y esta se encarga de validar su funcionamiento [7].

Otro de sus fines es en el ámbito de la programación competitiva, donde su

uso está generalizado debido a su rapidez de retroalimentación, una característica muy relevante en este contexto, puesto que en estos puedes encontrar los problemas a resolver y te permite realizar envíos para obtener si la solución propuesta es la correcta mediante la validación de varios casos de prueba. Existen varios jueces en línea como *Codeforces*¹ o *SPOJ*², los cuales se utilizan como jueces de entrenamiento que se encuentran abiertos al público; y otros como *DomJudge* que permiten crear instancias privadas para concursos propios como el *International Collegiate Programming Contest* (ICPC) en los cuales existen equipos que se enfrentan en un tiempo determinado a los mismos problemas de programación propuestos [8].

El uso de jueces de código en el campo de la programación competitiva es el más extendido, puesto que ya solo en el concurso anteriormente mencionado, el ICPC, entre los años 1999 y 2019 ha sufrido un crecimiento de un 2000 % en el número de asistentes, llegando a alcanzar en ese último año la cifra de 58.963 participantes de 3.406 universidades y 104 países diferentes [9], y estas cifras en la actualidad no han cesado su crecimiento.

Por otra parte, los jueces de evaluación de código también tienen varios usos en el campo de la educación como una herramienta educativa. Mediante esta, los alumnos pueden entregar los códigos realizados para que la aplicación los compile y ejecute, con el fin de proporcionarle una retroalimentación precisa, lo más instantánea posible, que no acepte soluciones incorrectas y que permita al docente tomar acciones en base a las habilidades de los estudiantes mostradas en los diferentes envíos que estos realicen [6].

Un ejemplo de uso en este ámbito podría ser UVa Online Judge³, uno de los sistemas de validación de código más conocidos. Este juez en línea, en sus primeras versiones, fue exclusivo para el uso de estudiantes de clases de algoritmia en la Universidad de Valladolid. Sin embargo, tras un tiempo, fue utilizado también en el ámbito de la programación competitiva, puesto que además de ser una de las primeras plataformas que permitía la realización de envíos al público general, tras 10 años recibió más de 5 millones de envíos realizados por más de 50 mil usuarios de 180 países diferentes [10]. En la tabla 1.1 se puede observar la comparativa de varios jueces de código en línea utilizados con fines educativos a fecha de 2018, donde las celdas en las que un juez “No aplica” es debido a que estos no cuentan con número de problemas específico, ya que se pueden subir los problemas deseados a la aplicación. Por otra parte, en la figura 1.1 se muestra un cronograma con las fechas de creación de cada uno de estos jueces.

Asimismo, ya se han realizado estudios en los que se han utilizado técnicas similares, como la validación y aportación de retroalimentación automática de ejercicios de código enviados por estudiantes mediante sistemas informáticos

¹Codeforces, <https://codeforces.com/>

²Sphere Online Judge, <https://www.spoj.com/>

³Online Judge, <https://onlinejudge.org/>

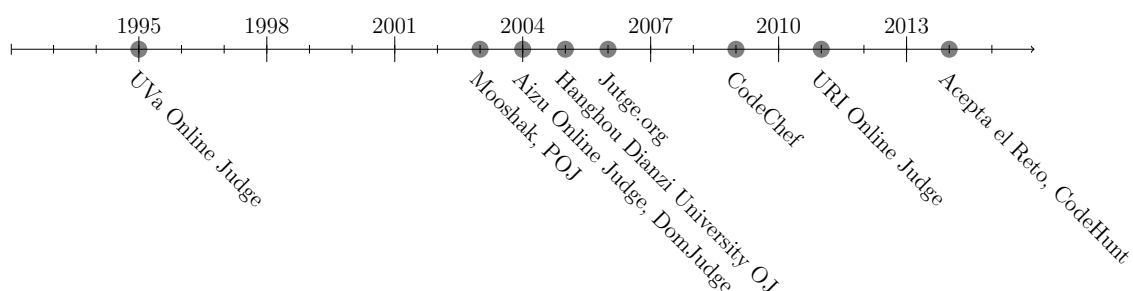


Figura 1.1: Cronograma de creación de jueces de código con fines educativos

Tabla 1.1: Jueces de código con fines educativos

Nombre	Idiomas soportados	# Problemas	Fecha Fundación
UVa Online Judge	Inglés	4300	1995
Mooshak	Inglés, Castellano, Portugués y Árabe	No aplica	2003
POJ	Inglés, Chino	3000	2003
Aizu Online Judge	Inglés y Japonés	3000	2004
DomJudge	Inglés	No aplica	2004
Hangzhou Dianzi University OJ	Inglés, Chino	6000	2005
Jutge.org	Inglés, Castellano, Francés, Catalán y Alemán	4843	2006
CodeChef	Inglés	3000	2009
URI Online Judge	Inglés, Castellano y Portugués	1100	2011
Acepta el Reto	Castellano	708	2014
CodeHunt	Inglés	8300	2014

(*Computer Based Assessment, CBA*). En estos casos, se han encontrado mejoras en la satisfacción de los estudiantes que han utilizado estos métodos como situaciones en las que el 80 % de estos han comentado que se encontraban más motivados a la hora de ponerse a realizar los ejercicios encomendados [11]. Además de estos, se ha comparado la corrección de ejercicios de programación de manera automática mediante alguna aplicación informática y de manera manual por parte de personal docente, y se ha encontrado que no solo los estudiantes estaban satisfechos, ya que la retroalimentación que aportaban era similar, sino que también a los educadores les ahorran mucho tiempo que invertían en las correcciones de estos ejercicios [12].

1.3. Estado del arte

La plataforma de evaluación de código automático *iudex*, del latín *juez*, es una aplicación web de código libre creada para poder utilizarse principalmente en el ámbito educativo, como clases de programación, algoritmos y bases de datos, para que los alumnos puedan realizar en este tanto ejercicios como pruebas prácticas. Esta aplicación ha sido desarrollada por varios alumnos de la universidad y se encuentra ya en un estado avanzado, aunque todavía faltan varios aspectos por cubrir.

En primer lugar, se desarrollaron paralelamente los trabajos explicados en las secciones 1.3.1 y 1.3.2. El primero de ellos se centró en crear la arquitectura de la aplicación mediante el análisis de las diferentes alternativas y el servicio que se encargaría de ejecutar los códigos que eran enviados por los usuarios. Por el contrario, en el segundo de estos trabajos se desarrolló una primera versión de la API con varias funcionalidades, así como los modelos de las bases de datos y los diferentes servicios. Seguidamente, el tercero de estos trabajos 1.3.3, consiguió añadir nuevas funcionalidades y realizar pruebas automáticas de la aplicación, además de crear un ciclo de Integración Continua y Despliegue Continuo. Por último, el cuarto trabajo realizado sobre la aplicación 1.3.4 se basó en la implementación de una interfaz web que los alumnos pudieran utilizar para poder hacer uso del juez.

1.3.1. Arquitectura de la aplicación

En un primer trabajo [13], se llevó a cabo el análisis de la arquitectura que se iba a implementar para la aplicación antes de empezar a construirla. Finalmente, se optó por seguir una implementación basada en servicios, los cuales estarían formados por diferentes módulos. Los servicios se refieren a un conjunto de recursos interrelacionados que proporcionan funcionalidades completas a los usuarios o aplicaciones. Estos recursos pueden incluir módulos, funciones, objetos o algoritmos. La finalidad de un servicio es facilitar el acceso y uso de estas funcionalidades de manera eficiente y efectiva. Por otra parte, los módulos son una serie de acciones y funciones que trabajan juntas para llevar a cabo un objetivo común, teniendo además atributos y datos compartidos, lo que facilita su trabajo y coordinación. Asimismo, cada módulo tiene una descripción abstracta que resume su funcionalidad y cómo trabaja en conjunto con los demás módulos.

Los servicios que se definieron fueron los siguientes:

- Interfaz web: Parte de la aplicación con la que interactúa el usuario para poder utilizarla mediante un esquema de peticiones a una API.
- Juez: Parte de la aplicación que constituye la parte funcional y lógica del

juez. Se encarga también de mantener la base de datos actualizada con las operaciones que realiza.

- RabbitMQ: Servicio de la aplicación que utiliza la tecnología del mismo nombre y se encarga de crear comunicaciones, así como de pasar mensajes entre los servicios del juez y del orquestador de contenedores mediante colas en las que estos mensajes se almacenan hasta que sean procesados.
- Orquestador de contenedores: Parte de la aplicación que se encarga de la correcta ejecución de los códigos que sean enviados por la aplicación. De esta manera, se encarga de recibir los mensajes enviados por el juez mediante RabbitMQ y enviarlos al contenedor correspondiente dependiendo del lenguaje del envío para ejecutarlo. Asimismo, cuando el contenedor termina de ejecutar el código, manda el resultado obtenido de vuelta al juez por el mismo servicio.

En la figura 1.2 se puede observar el esquema de estos servicios y algunos de los módulos más relevantes de cada uno de estos.

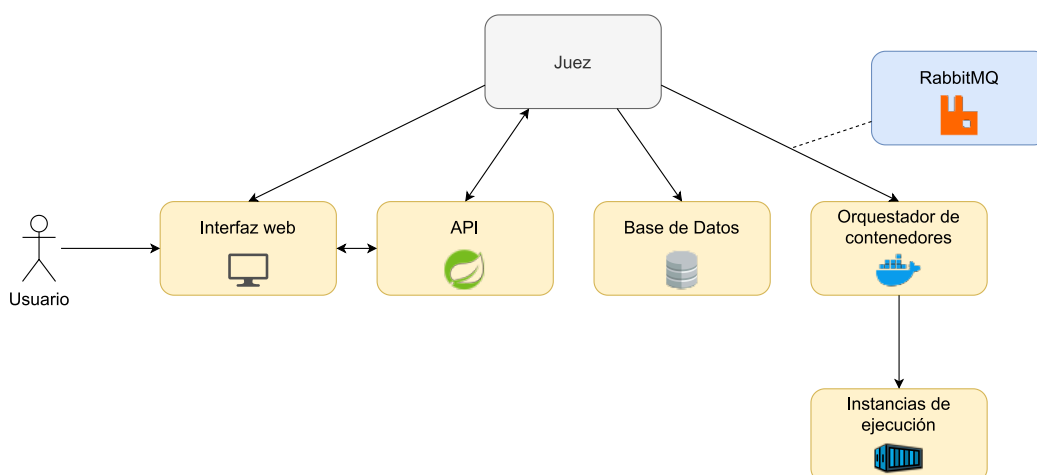


Figura 1.2: Esquema de arquitectura de módulos y servicios de la aplicación.

Además del diseño de la arquitectura, también se llevó a cabo la implementación del orquestador de contenedores y del servicio de RabbitMQ. De esta manera, se especificó y se implementó la conexión entre el servicio del juez y del orquestador, en el cual se envía el código que haya sido enviado por un usuario a la cola de ejecución de RabbitMQ para que el orquestador la pueda revisar. Al recogerlo de la cola, el orquestador crea una instancia en la que se pueda ejecutar, recoge las salidas, las valida y devuelve al juez mediante la cola de revisión el resultado de la ejecución para este pueda actualizar los atributos del envío del usuario.

Por otra parte, se diseñó el orquestador de manera que se pudieran ejecutar diferentes lenguajes de manera simplificada y para que de cara al futuro fuera

más fácil añadir nuevos lenguajes. Por este motivo, se optó por la utilización de contenedores frente a las máquinas virtuales. Uno de los motivos de esta decisión fue que las máquinas virtuales, para la ejecución de los códigos frente al uso de contenedores, implica un consumo menos eficiente de la memoria y era mucho más ineficiente en cuanto a tiempo de arranque, puesto que el uso de contenedores supone una mejora del 1250 % en algunos casos, ya que según análisis realizados, las máquinas virtuales tardaban 25 segundos, mientras que los contenedores tardaban tan solo 2 segundos en arrancar [13]. Además de esto, al utilizar contenedores, se simplifica la adición de nuevos lenguajes, puesto que la mayoría de estos ya cuentan con imágenes publicadas que tan solo requieren de unas pocas modificaciones para poder ser utilizadas en la aplicación, evitando así diferencias entre lenguajes, por ejemplo, compilados como Java e interpretados como Python. Estos dos fueron los únicos lenguajes que permitían ser ejecutados en este primer trabajo. Asimismo, los contenedores era una opción más escalable, ya que como la tarea de levantar contenedores era muy sencilla y que consumía menos recursos, permitía que se pudieran mantener ejecutándose una mayor cantidad de contenedores simultáneamente.

1.3.2. Desarrollo de la plataforma de la aplicación

Una vez escogida la estructura de la aplicación, en un segundo trabajo [14], se comenzó a implementar la sección del juez previamente mencionado. De esta manera, se crearon un sistema de clases y entidades que permitieran la organización y manipulación de información relacionada con concursos, problemas y envíos, entre otros, cuyo esquema también sería útil para construir el esquema de la base de datos, permitiendo así la persistencia de estos.

También se llevó a cabo la implementación de la API para que se pudiera utilizar más fácilmente la aplicación, de modo que se pudieran subir los problemas y recuperar la información de una manera más adecuada, así como la automatización de toda la documentación de esta mediante la librería de *Swagger*. Esta librería permite que, con la misma acción de compilación, se genere toda la documentación de la aplicación mediante el escaneo de todas las clases que contengan los puntos de servicio de la API y, mediante anotaciones, les asigne comentarios, de manera que sea mucho más fácil la generación de la documentación evitando su modificación manual cada vez que se cambie algo en la aplicación. Además de esto, para poder tener una parte más visual, se realizó una pequeña interfaz gráfica para poder realizar las primeras pruebas de la aplicación y tener un producto mínimo viable con el que presentarla, así como mediante el uso de *Swagger*, también se consigue levantar la documentación auto-generada de la API en una ruta del servidor web como se muestra en la figura 1.3, de modo que sea accesible mediante internet y se pueda probar, puesto que desde esta misma se permite la realización de llamadas a la API.

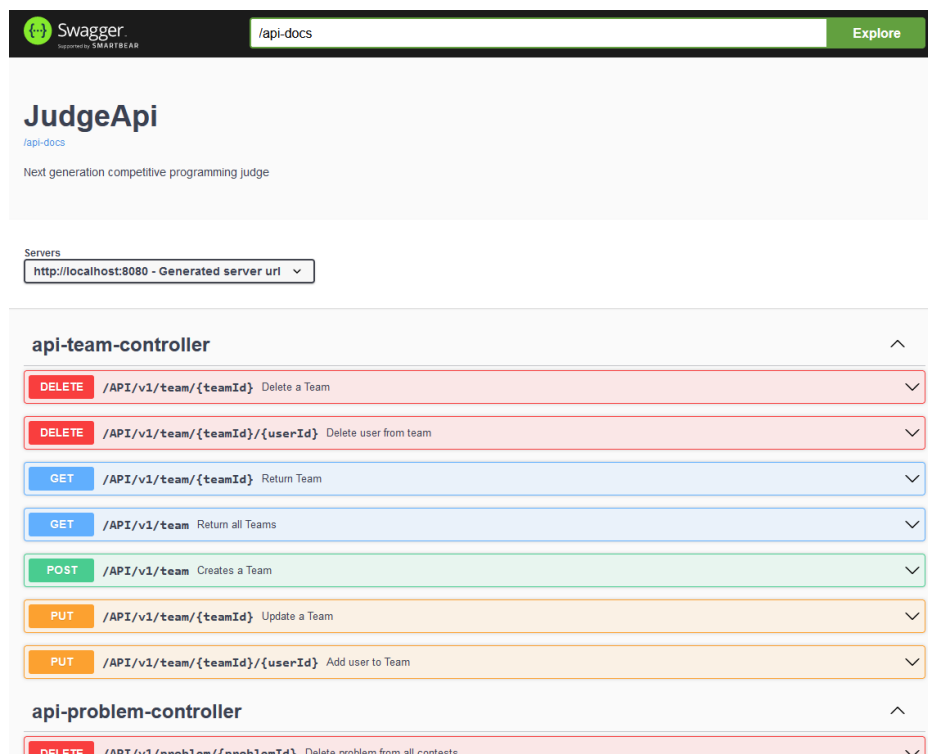


Figura 1.3: Captura de pantalla de la documentación de la API generada con Swagger

Por último, cabe destacar que en este TFG se realizó la implementación de los servicios intermedios, que se encargarían de la lógica de la aplicación que, entre otros, se encargarían de crear las conexiones entre el juez y el orquestador de contenedores mediante expedidores y receptores, revisar los resultados obtenidos de los envíos generados por los usuarios de manera que se asegure de hacer que estos persistan en la base de datos, o generar la funcionalidad de importado de problemas desde un archivo ZIP con un formato estandarizado, de manera que se puedan subir aplicaciones completas para que el orquestador pueda validarlas.

1.3.3. Implementación de nuevas funcionalidades

Tras los dos primeros trabajos que fueron realizados paralelamente por el mismo alumno, se procedió a realizar mejoras generales en partes ya existentes de la aplicación en un tercero [15].

En primer lugar, se buscó añadir funcionalidades nuevas, como sería la adición de los lenguajes C, C++, MySQL o MariaDB a la lista de lenguajes que el juez pueda ejecutar, de manera que la aplicación se pudiera utilizar para dar clases de Introducción a la Programación en grados como el de Ingeniería de Computadores o el de Ingeniería de la Ciberseguridad, donde esta asignatura de primer curso

se imparte en C, o para clases de Bases de Datos, donde se podría utilizar para realizar las prácticas o exámenes correspondientes en varios de los grados de la universidad.

Por otra parte, se consiguió modificar el código ya existente para poder obtener la información necesaria sobre un concurso, de manera que permitiera en un futuro realizar un marcador, en el cual se pudieran encontrar los datos de los diferentes usuarios o equipos que se encontraran participando en un concurso, junto con las puntuaciones globales del concurso e individuales de cada uno de los problemas que se encontraban en estos de una forma similar a la mostrada en la figura 1.4.

















RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K	L
1	 ENS Ulm 1 Ecole Normale Supérieure de Paris	11 1155	11 2 hrs 1 hr	34 1 hr	49 1 hr		183 1 hr	31 1 hr	256 2 hrs	14 1 hr	289 1 hr	112 1 hr	39 1 hr	97 1 hr
2	 gETHyped ETH Zurich	11 1220	5 1 hr	39 1 hr	100 1 hr		159 1 hr	13 1 hr	299 4 hrs	37 1 hr	227 1 hr	129 1 hr	25 3 hrs	87 1 hr
3	 P+P+P Harbour Space University	11 1241	9 1 hr	15 1 hr	26 1 hr	5 hrs	221 3 hrs	18 1 hr	224 6 hrs	40 2 hrs	293 2 hrs	66 2 hrs	33 1 hr	96 1 hr
4	 flag10 Università di Pisa	10 1154	4 1 hr	53 1 hr	55 1 hr		235 4 hrs	16 1 hr	5 hrs	23 1 hr	276 1 hr	189 2 hrs	73 1 hr	150 1 hr
5	 Heroes of the C Universidade do Porto	10 1182	9 1 hr	49 1 hr	80 2 hrs		284 2 hrs	33 1 hr	216 3 hrs	56 1 hr	1 hr	200 1 hr	69 1 hr	126 1 hr
6	 Bestagons Tel Aviv University	10 1189	7 1 hr	58 1 hr	95 1 hr		286 1 hr	17 1 hr	2 hrs	27 2 hrs	297 2 hrs	131 2 hrs	53 1 hr	198 1 hr
7	 dETHerninant ETH Zurich	10 1255	20 1 hr	34 1 hr	65 2 hrs		290 2 hrs	23 2 hrs		25 2 hrs	278 2 hrs	212 2 hrs	74 1 hr	134 2 hrs
8	 TempName Tel Aviv University	10 1285	3 1 hr	17 1 hr	166 7 hrs		236 1 hr	44 3 hrs	281 5 hrs	23 1 hr		111 1 hr	53 2 hrs	91 1 hr
9	 Jokers Tel Aviv University	10 1488	19 1 hr	112 1 hr	136 1 hr		22 2 hrs	295 5 hrs	39 1 hr	294 5 hrs	144 1 hr	41 2 hrs	186 1 hr	
10	 The Outlaws Instituto Superior Técnico	9 943	8 1 hr	26 1 hr	99 2 hrs		246 1 hr	33 1 hr	4 hrs	67 2 hrs		186 1 hr	57 2 hrs	161 1 hr
11	 UPC-I Universitat Politècnica de Catalunya	9 1038	5 1 hr	31 1 hr	122 1 hr		65 1 hr	65 2 hrs	8 hrs	94 2 hrs	182 1 hr	225 1 hr	41 1 hr	233 1 hr
12	 EPFL 1 Ecole Polytechnique Fédérale de Lausanne	9 1054	7 1 hr	36 1 hr	166 2 hrs		257 1 hr	50 2 hrs	1 hr	70 1 hr		140 1 hr	93 1 hr	195 1 hr
13	 I eat PHP for breakfast Harbour Space University	9 1091	6 1 hr	73 1 hr	118 4 hrs		259 2 hrs	11 1 hr	9 hrs	51 1 hr		120 1 hr	88 2 hrs	225 3 hrs
14	 LaStale Blue Università Degli Studi di Milano	9 1397	4 1 hr	44 1 hr	112 4 hrs	1 hr	272 1 hr	63 2 hrs	1 hr	108 1 hr		249 2 hrs	78 1 hr	287 5 hrs
15	 Carovana Fighters Scuola Normale Superiore	9 1459	8 1 hr	102 1 hr	155 2 hrs		28 1 hr	2 hrs	147 4 hrs	298 2 hrs	271 1 hr	55 1 hr	295 1 hr	
16	 x*3 Harbour Space University	8 854	12 1 hr	59 1 hr	91 2 hrs	5 hrs	48 3 hrs		32 1 hr		283 3 hrs	62 1 hr	167 1 hr	

Figura 1.4: Ejemplo de tabla de puntuaciones del SWERC 2022

Esta clasificación está extraída de un concurso de programación competitiva. Los valores de la columna “score” indican el número de problemas resueltos y la puntuación global (con penalizaciones) obtenida por cada equipo, mientras que en cada una de las celdas de los diferentes problemas identificados de A a L, se observa la puntuación individual formada por el tiempo invertido en minutos desde el inicio del concurso más una penalización de 20 minutos por cada envío erróneo.

Además de esto, también se terminó añadiendo nuevas entradas a la API de la aplicación, entre las cuales se incluyeron:

- Puntos de servicio para los *samples*, los cuales serían los datos de prueba de los problemas que se ejecutan y que cuando se añade una nueva entrega, se utilizan para verificar la correcta ejecución del código enviado.
- Puntos de acceso para añadir y borrar equipos participantes de un concurso, para poder así mejorar la experiencia de los usuarios.

- Puntos de servicio para poder especificar los lenguajes aceptados en diferentes concursos, de manera que los creadores de estos pudieran añadir o borrar los lenguajes soportados en un concurso específico, teniendo en cuenta que estos deberían ser lenguajes soportados por el juez, para lo cual se añadió un punto de conexión para conocer los lenguajes disponibles.

Además de las mencionadas funcionalidades nuevas, también se mejoraron aspectos de la aplicación en cuanto a mantenibilidad, adaptabilidad y desarrollo continuo. Algunas de las mejoras fueron las siguientes:

- Refactorización del código y ficheros, puesto que había multitud de código comentado que no se utilizaba y que entorpecía la lectura de este, de manera que se mantuviera en la aplicación un código más limpio y bien formateado, así como la simplificación del código mediante la refactorización de algunas otras clases.
- Mejora y ampliación del sistema de *logs* o registros de acontecimientos ya existente, de manera que estos sean más claros y se puedan aprovechar más para poder detectar y resolver problemas de forma más rápida y eficaz.
- Adición de pruebas automáticas para comprobar el correcto funcionamiento del sistema. Se consiguieron añadir un total de 33 pruebas de diferentes tipos y correctamente ejecutadas, entre las que se encontraban pruebas de integración, pruebas de humo y pruebas del propio sistema.
- Evitar el uso de los objetos nulo y promover el uso de los *Optional* para evitar excepciones del tipo `NullPointerException`.
- Uso del analizador de código estático SonarCloud, sobre el que se profundizará en el capítulo 3, para obtener métricas como la mantenibilidad, número de *bugs*, deuda técnica, etc.

Por último, se consiguieron corregir varios fallos que hacían que la aplicación quedara fuera de disponibilidad o funcionara incorrectamente en algunas situaciones concretas.

1.3.4. Interfaz web para la aplicación

Finalmente, el trabajo más reciente realizado sobre la aplicación [16] trató de realizar una interfaz web más intuitiva que la anterior, la cual solo tenía las partes más relevantes para poder mostrar el correcto funcionamiento de la aplicación.

En primer lugar, se especificaron todas las acciones que deberían ser permitidas para cada tipo de usuario que utilizaría la aplicación (anónimo, alumno y

profesor), así como los requisitos que debería cumplir la aplicación. Después de esto, se realizó la implementación de la interfaz web que se muestra en la figura 1.5. Para esta, se decidió utilizar el *framework* de Angular, ya que facilitaba la creación de esta por su posibilidad para reutilizar componentes.

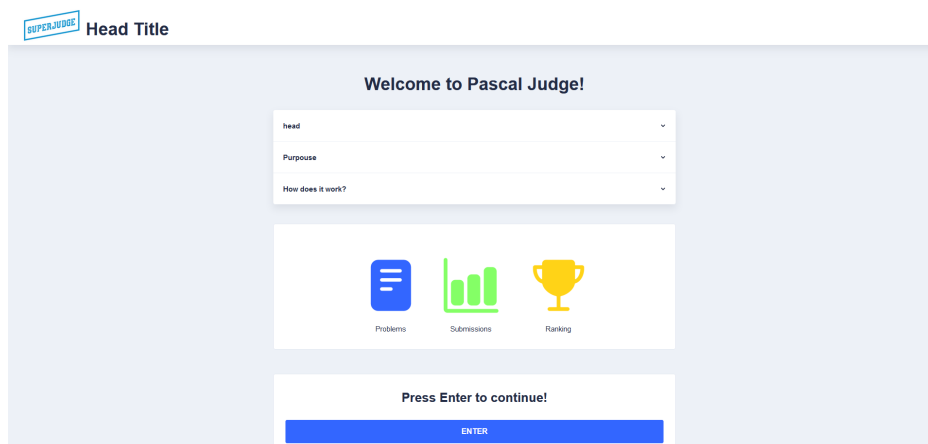


Figura 1.5: Vista principal de la interfaz gráfica realizada de la aplicación

En un inicio, como no se contaba con la versión final de la API, se necesitó simular las respuestas que esta debería dar en su versión final, de manera que se pudiera trabajar en la conexión de la interfaz con la lógica de la aplicación. Cuando la API final ya se encontraba en una versión estable, se eliminaron de los servicios de la interfaz casi todos los usos de la API simulada, exceptuando los que permitían los inicios de sesiones ya que todavía no se contaba con esta funcionalidad en la parte lógica de la aplicación.

Además, se desarrolló la internacionalización de la interfaz. Con esta, se conseguiría que la aplicación se pudiera adaptar a diferentes idiomas y la interfaz web de la aplicación cuente con la opción ser mostrada en cualquiera de estos, así como con dialectos de diferentes regiones, sin necesidad de realizar cambios en el código fuente. De esta manera, para añadir nuevos lenguajes sería tan simple como crear su propio archivo de traducciones y rellenarlo.

Por último, en la memoria de este último trabajo, se dejó también documentado tanto el diagrama de navegación seguido para la implementación de la interfaz como la descripción de la estructura del proyecto escogido, para así facilitar la continuación del trabajo sobre la interfaz.

1.4. Estructura del documento

Tras una breve introducción en este primer capítulo 1, este Trabajo de Fin de Grado tratará explicar las mejoras realizadas en el proyecto *iudex*. En el capítulo

[2](#) se comentan los diferentes objetivos, motivaciones que han promovido la realización de este trabajo y la metodología realizada. Por otra parte, en el capítulo [3](#) se ha explicado la descripción informática de este, incluyendo y las tecnologías utilizadas y las implementaciones y modificaciones realizadas. Seguidamente, en el capítulo [4](#) se comentan los resultados obtenidos tras el trabajo realizado, además una validación del correcto funcionamiento de las modificaciones llevadas a cabo en la aplicación. Finalmente, en el capítulo [5](#) se realizará una retrospectiva y se comentarán los posibles trabajos a realizar en el futuro.

2

Objetivos

Como se ha mencionado previamente, se puede comprobar que aunque el sistema que soporta *iudex* es funcional, todavía necesita algunas funcionalidades, así como otras mejoras para poder estar preparado para utilizarse en un entorno real. Aquellas que van a ser tratadas en este trabajo, quedan especificadas a continuación, así como los objetivos personales y secundarios que cubren las inquietudes y la motivación para la realización de este Trabajo de Fin de Grado, además de la metodología seguida para alcanzarlos

2.1. Objetivos principales

Los objetivos principales del desarrollo de este trabajo, quedan definidos por las funciones y mejoras que se desean desarrollar en la aplicación, las cuales serían las siguientes.

2.1.1. Mejora de la calidad

Con esta tarea, se busca poder familiarizarse tanto con la estructura como con el código de la aplicación para poder desarrollar las siguientes tareas de una manera más cómoda. Mientras tanto, y utilizando la herramienta SonarCloud de la que se hablará más en profundidad en el capítulo 3, se aprovecha la oportunidad para ir comprobando los *bugs*, vulnerabilidades y líneas de código duplicadas, e ir corrigiendo aquellos fallos que se encuentren. De esta manera, se busca aumentar

la mantenibilidad del código de la aplicación, así como aumentar su adaptabilidad y expansibilidad de cara al futuro.

Previo a la realización de este TFG, la aplicación contaba con 8 *bugs*, 6 vulnerabilidades y un 8,4 % de líneas duplicadas.

2.1.2. Mejora de las pruebas automatizadas de la aplicación

Como ya se ha comentado, la API de la aplicación cuenta con varios puntos de servicio, lo cuales varios de ellos ya cuentan con pruebas automáticas que comprueban su correcto funcionamiento. Sin embargo, no cubren todas las necesidades, puesto que no cubren todas las líneas o condiciones de estos, así como algunos de puntos de servicios de la aplicación siguen sin tener pruebas automáticas que los cubran. Por este motivo, se va a realizar la implementación exhaustiva de todos los puntos de servicio de la API de la aplicación, de manera que su funcionamiento pueda ser comprobado durante la ejecución de las pruebas automáticas en el flujo de trabajo ya creado con *GitHub Actions* y mejorar así el ciclo de integración continua y despliegue continuo de la aplicación.

2.1.3. Autenticación delegada y gestión de roles

Uno de los aspectos que se encuentran ausentes en la aplicación sería la seguridad, puesto que con el estado actual, no es posible iniciar sesión, registrar usuarios, ni mucho menos una gestión se permita denegar o permitir el acceso a los diferentes tipos de usuarios a cada uno de los puntos de servicio de la interfaz del *backend* de la aplicación. Es por esto, que se implementará un sistema de autenticación delegada que permita a los usuarios iniciar sesión en la aplicación mediante el mismo inicio de sesión que realizan los alumnos en el Aula Virtual de la universidad, para así facilitar el uso del juez en el ámbito educativo. De esta manera, se evitaría tener que dar de alta a todos los alumnos que fueran a utilizar la aplicación, puesto que se utilizarían los registros institucionales que se otorgan cuando los alumnos se matriculan en la universidad. Asimismo, también se podría facilitar la asignación de calificaciones en el Aula Virtual, puesto que se dispondría de los mismos datos identificativos de los alumnos y sería más cómodo que tener que ir comprobándolo manualmente.

2.1.4. Implementación de *WebSockets*

Por último, con el estado actual de la API, si en la interfaz de aplicación se quisiera notificar a un usuario de que su envío ha sido validado, se tendría

que hacer peticiones periódicas para comprobar si este ha sido terminado de ejecutar. Estas acciones supondrían que, sobre todo cuando haya muchos usuarios utilizando la aplicación, generaría una sobre carga del sistema y podría llegar a sufrir en cuanto a rendimiento. Un ejemplo de esta situación podría ser la del juez de *¡Acepta el Reto!*, observable en la figura 2.1, el cual realiza varias llamadas a su API hasta que el envío que se está comprobando devuelva el resultado final de este, lo que provoca que en momento en los que tenga varios usuarios utilizando su plataforma, esta baje su rendimiento por el gran número de peticiones que tiene que resolver.

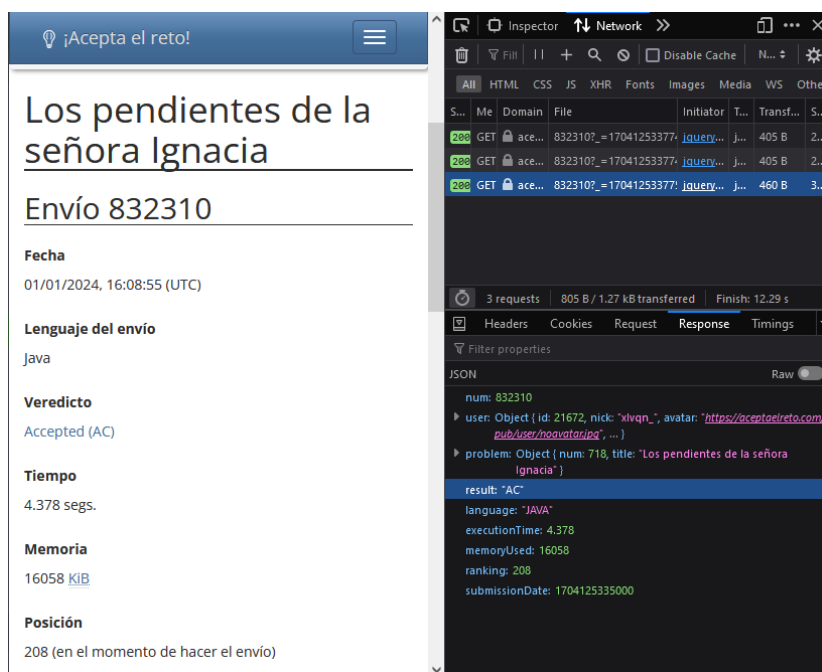


Figura 2.1: Ejemplo de peticiones a una API para comprobar el estado de los envíos

Por este motivo, se han implementado WebSockets en el *backend* de la aplicación. Esta tecnología permite una comunicación a tiempo real, que se consigue mediante la creación de un canal diseñado para permitir un paso de mensajes entre cualquier aplicación cliente/servidor, entre las que se incluye la relación navegador/servidor web que utiliza la aplicación *iindex*. La idea de esta tecnología es la posibilidad de enviar mensajes sin tener que consultar a un servidor para recibir respuesta, como pasa con las API Rest [17].

2.2. Objetivos secundarios

- Estudiar en detalle diferentes tipos y protocolos de autenticación, especialmente *OAuth 2.0*, *OpenID Connect* y *SSO* ¹.
- Adquirir conocimientos sobre los usos y funcionamiento del software de código libre *Keycloak* para la gestión de acceso de usuarios.
- Aprender sobre la generación y uso del estándar de *JWTs* para la gestión de sesiones de usuario.
- Ampliar el conocimiento sobre control de recursos en una API gestionado por roles.
- Profundizar en el entendimiento de la verificación del funcionamiento de una aplicación mediante *tests* de integración.
- Expandir mi visión sobre la Integración Continua y el Despliegue Continuo.
- Desarrollar una mayor comprensión de las métricas de calidad del *software*.
- Aprender sobre la tecnología de virtualización y el uso de contenedores y orquestadores.
- Ampliar los conocimientos sobre *Docker*, su configuración, y la escritura de archivos *Dockerfile* y *Docker Compose*.
- Estudiar en profundidad del *framework Spring Boot* y las tecnologías web relacionadas.
- Descubrir más acerca del mantenimiento y la evolución de una aplicación web.
- Adquirir nuevo conocimiento sobre \LaTeX y Overleaf.

2.3. Metodología empleada

Para la realización de este trabajo, en primer lugar, se plantearon dos tipos de metodologías que se podrían emplear, que son de las cuales derivan las demás. Estas opciones eran las metodologías tradicionales y las metodologías ágiles.

En cuanto a la primera, agrupa aquellas metodologías que pretendían indicar a los profesionales aquellas pautas a seguir para realizar y documentar cada una de las tareas del desarrollo del software, comenzando por un proceso de elicitación de requerimientos, previo al análisis y diseño. De esta manera, se centran en la

¹Single Sign-On

planificación de todo el trabajo a realizar y, una vez quede todo especificado, se comienzan a aplicar el ciclo de desarrollo del producto software. Por esto mismo, se suele utilizar en proyectos grandes con una documentación muy detallada, y uno de los ejemplos más conocidos es la metodología en cascada [18].

Sin embargo, en cuanto a la última opción, nació en una reunión de expertos en Snowbird, Utah, en 2001, entre los que estaban Robert C. Martin, Andrew Hunt y Dave Thomas. Su objetivo con esta nueva idea fue analizar los valores y principios que puedan permitir el desarrollo software de una manera más rápida. De esta manera, surgió *The Agile Alliance* y el concepto de metodologías ágiles. Con estas, los equipos de desarrollo se encuentran en colaboración constante con el cliente, además de centrarse en el software funcional [19].

Finalmente, como la idea del desarrollo de este trabajo es la realización de varias tareas que requieren de verificación por parte de un cliente mediante la realización de varias iteraciones, se ha optado por el uso de una de estas metodologías, y más concretamente por la metodología Scrum.

Durante la realización de este TFG, se han realizado 4 *sprints*², de una duración de 3 semanas cada uno, así como varias reuniones periódicas, cada 2 o 4 días según el desarrollo realizado, con los tutores para poder exponer los avances que se han estado realizando. Mediante esta división, se ha podido realizar las distintas tareas y, tras su finalización, se ha podido recuperar una retroalimentación que ha servido para realizar las modificaciones deseadas lo antes posible, durante el siguiente *sprint*.

Además de esto, se han realizado tanto una planificación como una revisión de los *sprints* al principio y al final de estos, respectivamente. En la primera de estas reuniones, se establecieron los hitos a conseguir en cada uno de los *sprints*, los cuales se establecían partiendo de los objetivos principales mencionados anteriormente en este documento, además de planear las diferentes tareas a realizar recuperadas de la pila de tareas denominada *product backlog*. Por otra parte, en la reunión del *sprint review* se ha presentado, comentado y validado el trabajo realizado en cada *sprint* [20].

Del mismo modo, para seguir los roles de esta metodología, los tutores de este trabajo han estado representando la figura del cliente, por lo que en estas reuniones periódicas, donde se les mostraba el producto obtenido tras la iteración, daban su opinión al respecto, proponían mejoras o nuevas funcionalidades que poder añadir en la siguiente iteración, y probaban aquellas que ya se habían añadido para comprobar el correcto funcionamiento. Esto ha sido bastante relevante porque además ha permitido encontrar errores que, de no ser por su participación en el proceso, no se hubieran encontrado tan fácilmente, además de que se han encontrado nuevas ideas sobre funcionalidades que implementar que han acaba-

²Un *sprint* es un período breve de tiempo fijo en el que el equipo Scrum trabaja para completar el trabajo establecido para dicho período.

do siendo parte del trabajo realizado, como podrían ser la implementación de WebSockets, que en una de las reuniones mantenidas fue propuesta.

3

Descripción informática

Previo al comienzo del trabajo, se organizaron las tareas a realizar y se especificó el camino de desarrollo a seguir, así como las herramientas que se iban a utilizar y el rol que se iba a cumplir.

3.1. Herramientas y tecnologías utilizadas

En la siguiente sección se desarrollan las herramientas tanto tecnológicas como organizativas que se han utilizado para poder llevar a cabo el trabajo realizado.

3.1.1. Git

Git es un sistema de control de versiones moderno, distribuido y seguro que fue desarrollado por Linus Torvalds en 2006, el cual facilita el desarrollo de aplicaciones y código en equipo, puesto que permite confirmar cambios, hacer combinaciones de estos y lanzar publicaciones de software de una manera mucho más sencilla que de otras maneras.

Específicamente, para esta aplicación se está utilizando GitHub como plataforma de alojamiento de código fuente, permitiendo así que varios desarrolladores puedan alojar y revisar su código, así como colaborar con otros. Del mismo modo, GitHub utiliza Git como su sistema de control de versiones, por lo que se beneficia de las ventajas de este. Por otra parte, y para más comodidad con las

tareas cotidianas relacionadas con Git, se ha estado utilizando la herramienta GitKraken Client.

3.1.2. Spring

En la aplicación del juez de código se utiliza Spring como framework [21] [22] para la aplicación web programada con Java. Este se compone de numerosas herramientas, las cuales lo hacen uno de los paquetes más completos dedicados al desarrollo de aplicaciones web.

Por otra parte, y fuera del paquete estándar de Spring, este cuenta con un módulo que facilita toda la implementación de WebSockets, pues este proporciona multitud de clases ya creadas y anotaciones que hacen que la configuración de las clases personalizadas que hacen falta para implementarlos sea mucho más sencilla. Por este motivo, durante este trabajo se ha estado utilizando concretamente esta librería, ya que además de las ventajas mencionadas, al formar parte de un paquete de librerías tan popular como lo es Spring, cuenta con multitud de documentación que ha sido muy útil durante el desarrollo de la tarea.

Asimismo, también se ha utilizado la librería de Spring Security, el cual proporciona una amplia gama de herramientas para la seguridad de los servicios web, como sería la API desarrollada para la aplicación *iudex*. Estas herramientas han sido principalmente utilizadas para la sección de autenticación y autorización [23].

3.1.3. Single Sign-On (SSO) y Lightweight Directory Access Protocol (LDAP)

Para la implementación de la autenticación en la aplicación, se han utilizado los conceptos de SSO y LDAP. En cuanto a SSO, proviene de las siglas en inglés *Single Sign-On*, el cual es un proceso de autenticación que permite a un usuario autenticado acceder a varias aplicaciones con un solo conjunto de credenciales, sin tener que introducirlos cada vez que cambia de aplicación. Siendo así, se simplifica el proceso de inicio de sesión, mejora la seguridad y reduce los costes de gestión de contraseñas, pues se delega en un servicio centralizado. Por otra parte, LDAP, del inglés *Lightweight Directory Access Protocol*, es un protocolo que permite que acceder a bases de datos almacenadas en servidores denominados “de directorio”, que contienen información sobre los usuarios, grupos, recursos y políticas de una organización. Estas se suelen utilizar para consultar y modificar los datos del directorio, así como para poder autenticarlos mediante el SSO. Por lo tanto, ambos procesos facilitan la integración de diferentes aplicaciones y sistemas con una fuente única y centralizada información.

3.1.4. KeyCloak

KeyCloak [24] es una solución de código abierto que se utiliza para la gestión de identidades y acceso a aplicaciones, puesto que permite añadir el servicio de autenticación a estas con un esfuerzo mínimo. Esta tecnología soporta la creación de usuarios y la autenticación fuerte de un usuario, entre otros. También cabe destacar que soporta diferentes protocolos como OpenID Connect, OAuth 2.0 y SAML 2.0.

Entre los principales usos que tiene, se encuentra el de habilitar el inicio de sesión único entre múltiples aplicaciones y servicios, así como integrar con proveedores de identidad externos, como sería el mismo que el que se utiliza actualmente para el Aula Virtual de la Universidad Rey Juan Carlos. Es sobre esta característica en la que se ha centrado parte de este trabajo, puesto que con esta manera conseguimos implementar la autenticación de usuarios en la aplicación mediante los mismos credenciales que se usarían para el resto de servicios de la universidad, centralizando así la administración y el control de acceso de los usuarios y los recursos con la institución [25], evitando la necesidad de crear cuenta de usuario para todos los alumnos de la universidad que vayan a utilizar el juez de código.

3.1.5. Mockito

Mockito [26] es una librería de código abierto para Java utilizada para la realización de pruebas unitarias. Esta permite crear objetos simulados o *mocks* con una API limpia y simple, lo que hace que las pruebas sean legibles. Los objetos simulados que imitan el comportamiento de los objetos reales de forma controlada. En este trabajo se ha utilizado para poder simular las operaciones realizadas por los servicios, de tal manera que se pudiera realizar las pruebas unitarias pertinentes de los puntos de servicio de la aplicación. Con esto, se consigue abstraer la parte de la aplicación que se quiere probar sin necesidad de tener que pasar por numerosos métodos, creando así pruebas pequeñas pero exhaustivas de las partes críticas de la aplicación.

3.1.6. SonarCloud

SonarCloud [27] es una plataforma en línea que analiza la calidad del código fuente de manera estática, sin ejecutarlo, de acuerdo con ciertas reglas de codificación y métricos definidos para los lenguajes de desarrollo más comunes.

Durante la realización de este trabajo, se ha utilizado un canal de integración continua creado en un trabajo anterior, el cual conectaba SonarCloud mediante la herramienta de GitHub Actions. De esta manera, cada vez que se subían cambios

en el repositorio, GitHub se encargaba de ejecutar las pruebas automáticas de la aplicación mediante un flujo de trabajo definido para posteriormente pasar los datos del código compilado a la herramienta de SonarCloud. Posteriormente, estos datos son analizados y sus resultados son accesibles públicamente en la propia página de esta herramienta, https://sonarcloud.io/project/overview?id=URJC-CP_iudex.

Una vez se cuenta con el análisis detallado, se pueden observar los diferentes aspectos a mejorar en el código, para posteriormente revisarlos uno por uno e ir tratando de arreglarlos.

3.1.7. Docker

La aplicación con la que se ha trabajado se encontraba previamente *dockerizada*, puesto que ya contaba con dos archivos de configuración denominados *docker compose* con los que con tan solo un comando se consigue levantar la aplicación y todas sus dependencias, como la base de datos y el servidor de RabbitMQ. Esto se consigue gracias a la tecnología de Docker [28], un proyecto que empezó siendo de código abierto y de la mano de dotCloud en el año 2013. Esta herramienta simplifica la creación y ejecución de aplicaciones en entornos aislados, basándose en la tecnología de contenedores.

3.1.8. L^AT_EX

Finalmente, se ha utilizado L^AT_EX para la redacción de la memoria de este trabajo debido a la calidad tipográfica de los documentos una vez se realizan y la apariencia profesional que aporta. Por este motivo, se ha optado por utilizar la aplicación web Overleaf, ya que al ser un servicio en la nube, facilitaba la colaboración con los tutores, quienes simultáneamente podían revisar y comentar el trabajo realizado.

3.2. Implementación

Para comenzar con el desarrollo de la aplicación fue necesario familiarizarse con el código ya existente. De esta manera, se obtuvo una visión mucho más clara de las diferentes funciones que se encargaban cada una de las clases y paquetes de la aplicación. Se probó a desplegar la aplicación y a probar la interfaz web para comprobar el funcionamiento de esta. Sin embargo, debido a varias dificultades con su uso, se optó por el uso de la API e Insomnia, un cliente que permite realizar peticiones a esta y probar su funcionamiento.

3.2.1. Mejora de métricas de calidad de Sonar

Una vez se había decidido que se usaría la API para probar el funcionamiento del *backend*, se comenzó a revisar la aplicación utilizando el analizador de código SonarCloud. De este modo, se profundizó en la comprensión del código a la vez que se iba mejorando la mantenibilidad de este.

Bugs

Revisando algunas de las incidencias que se encontraban identificadas en la página de SonarCloud, se observó que con las clases de `DockerContainer`, entre las que se incluían también `DockerContainerJava`, `DockerContainerC`, `DockerContainerCPP`, `DockerContainerPython3` y `DockerContainerMySQL`, había varios conflictos, puesto que se encontraron 5 *major bugs*, además de otros de categorías menores. Algunos de estos estaban duplicados por el motivo ya mencionado. Entre los fallos que más destacaban se encuentran:

- Cerrado de recursos abiertos que provocan que la aplicación dedique recursos a mantenerlo abierto aunque ya no se esté utilizando, como es el caso de las clases `CreateContainerCmd` de la librería `Docker-Java`. Estas se utilizan para gestionar los contenedores creados desde la aplicación mediante código en Java. Para arreglarlo se siguieron los pasos proporcionados por SonarCloud y se utilizó dicho recurso dentro de un bloque de código *try-with-resources*, el cual se encarga de cerrar automáticamente aquellas clases que implementen la interfaz *Autocloseable*, tal y como era el caso. Con este cambio, la aplicación ya no sufriría tanto en rendimiento ya que, aunque Java cuenta con un recolector de basura, este no puede eliminar los recursos abiertos en la memoria ¹.
- Durante la comprobación de si había terminado o no la ejecución de código en el contenedor, SonarCloud avisaba de que podría darse el caso en que apareciera una excepción del tipo `NullPointerException`. Para solventarlo, se utilizó el método *Boolean.TRUE.equals* de Java para realizar la comparación, la cual evita estas situaciones, simplificando así la comprensión del código.
- Para un mayor entendimiento de los registros de acontecimientos de la aplicación, se implementaron excepciones personalizadas para indicar cuando se había utilizado algún lenguaje de programación no aceptado por el juez de código.

¹Más información en CERT, FIO04-J

- Para evitar sobrecargar la aplicación mediante la comprobación de si ha finalizado la ejecución de un contenedor, se añadió un retardo de 200 milisegundos a dicha comprobación, rebajando así la carga que soportará la aplicación y ahorrando recursos.

Más adelante, en SonarCloud se observaron dos *bugs* más. El primero de ellos mostraba que había casos en los que en los *logs* de la aplicación se llegaban a mostrar textos que podían haber introducido los usuarios sin que estos hubieran pasado por ningún proceso de tratamiento. Es por esto, que SonarCloud recomendaba la sanitización de estos textos. La manera estudiada fue el utilizar los métodos ya existentes de la clase estática Sanitizer, uno de ellos recibe una cadena de caracteres y otro un objeto de la clase *Optional* conteniendo otra cadena de caracteres. En el primero de estos, se recogería la cadena y se le pasaría una expresión regular, de manera que se eliminaran aquellos caracteres que creaban una vulnerabilidad en la aplicación, mientras que en el segundo, tras comprobar que tiene contenido, se llamaría al primer método. Los caracteres que se tienen en cuenta son los siguientes:

1. `\n` (salto de línea o *line feed*): Provoca que el cursor pase a la siguiente línea. En sistemas operativos basados en Unix, esto también causa que el cursor se mueva al inicio de esa línea.
2. `\r` (retorno de carro o *carriage return*): Hace que el cursor se mueva al comienzo de la línea actual. En sistemas operativos de la misma familia que Windows, se suele utilizar la combinación de este carácter junto con `\n` para realizar un salto de línea completo.
3. `\t` (tabulación o *tab*): Provoca que el cursor se mueva a la posición de la parada de tabulación. Este se establece en intervalos de cuatro u ocho posiciones de caracteres de distancia y es útil para alinear datos en columnas.
4. `\f` (avance de página o *form feed*): Se utiliza generalmente solo con impresoras para hacer que la página actual se alimente para que la siguiente se la página actual.

El motivo por el que se eliminan estos caracteres, es para evitar que se almacenen caracteres que pueden hacer vulnerable partes del código o datos sensibles de los usuarios. Por este motivo, se utilizaron los métodos en las líneas indicadas por SonarCloud en los que se recogían textos introducidos por el usuario, tanto directamente como en los nombres de los diferentes archivos que se pueden subir a la aplicación.

Por otra parte, el otro *bug* con el que contaba en ese momento la aplicación era que, cuando se recibían los mensajes por parte de los contenedores Docker,

no se interrumpía el hilo de ejecución, de manera se dejaban hilos sin terminar de ejecutar. Para solventarlo, cuando se trata con una excepción de la clase **InterruptedException**, se interrumpe la ejecución del hilo actual para posteriormente relanzar la traza de error, pero esta vez con una excepción en tiempo de ejecución para que la aplicación pueda seguir ejecutándose.

Fallos de código y lógica

Mientras se arreglaba la parte de los contenedores de la aplicación, se encontraron también dos fallos en el código fuera de los encontrados con la herramienta SonarCloud, que hacían que no funcionara correctamente la API. Esto se debía a que en trabajos anteriores se habían modificado las relaciones entre entidades, y los repositorios, que son las representaciones de las tablas de la base de datos en tiempo de ejecución en el código Java y estas no se encontraban, de modo que no se llegaban a actualizar las tablas relación entre las entidades de *Contest* y *Problem*. Por otra parte, tampoco se conseguían editar satisfactoriamente estas relaciones, puesto que en el código de Java al tratar con conjuntos de los objetos relación *ContestProblem*, se requería la implementación del método *equals*, el cual al no estar sobrescrito, tan solo comprobaba la referencia en memoria en los objetos, lo que interpretaba objetos con todos los atributos idénticos como diferentes.

Por otra parte, se observaba un 8,4 % de código duplicado en toda la aplicación, donde aproximadamente la mitad se encontraba en las clases *DockerContainer*. Esto era debido a que el código de estas había sido copiado y pegado entre las diferentes clases, ya que la mayoría habían sido duplicadas y personalizadas para cada lenguaje.

Como solución temporal, y a la espera de tener más información sobre la aplicación, los contenedores de todos los lenguajes, exceptuando el correspondiente a MySQL, se juntaron todas las demás clases en una sola, que acabaría siendo la denominada *DockerContainer*, donde se irían generando dinámicamente las diferentes opciones dependiendo del lenguaje. De esta manera, se reducirían el número de líneas y mientras se trabajaba en otra solución, se consiguió eliminar los *bugs* existentes. Más adelante, esta solución se sustituyó por la aplicación del patrón de diseño *Simple factory* que se comenta en la sección 3.2.1.

Finalmente, se intentaron ejecutar algunas pruebas automáticas que se encontraban desactivadas. Algunas de estas fallaron porque no contaban con las últimas entidades añadidas en trabajos recientes. De esta manera, no se podían probar satisfactoriamente. Entre las clases que no se habían tenido en cuenta se encuentran *TeamUser* y *ContestProblem*. Con la inclusión de estas nuevas clases en los métodos donde eran necesarias, se consiguieron arreglar varias de las pruebas que no se conseguían ejecutar.

Patrón de diseño simple factory

Los patrones de diseño son, generalmente, soluciones reutilizables para problemas comunes en cuanto a diseño, los cuales buscan solucionar problemas de tipos semejantes con soluciones que también son similares. Además de esto, son soluciones consolidadas, pues estas han sido aplicadas a lo largo de mucho tiempo. Por otra parte, también aportan una guía sobre como se puede solucionar un problema y como puede ser utilizado en diferentes situaciones, donde al mismo tiempo ayudan a alcanzar el mejor diseño posible de una manera mucho más rápida [29].

Por otra parte, el patrón de diseño *Simple factory* busca la simplificación de la creación de un objeto sin exponer la lógica de creación al cliente. En la programación orientada a objetos, una factoría es un objeto que crea otros objetos, y esta puede ser invocada de diferentes maneras, siendo una de ellas la existencia de un método que pueda devolver objetos que partan de diferentes clases pero todas tengan la misma clase padre. Finalmente, en el diagrama 3.1 se puede observar el estado final de las modificaciones realizadas a las clases `DockerContainer`. Asimismo, en la clase `DockerContainerMySQL` ha vuelto a añadir el método *ejecutar* para especificar que la única clase que tiene sobrescrito dicho método por varias dificultades para abstraerlo con la implementación de la clase padre.

De esta manera, y en comparación con el estado inicial, se consiguió reducir el número de líneas duplicadas en el proyecto más de un 33 %, hasta un 5,6 %, se arreglaron los fallos en el código que hacían que este fuera vulnerable y, además, se consiguió implementar una solución que facilitaba la adición de nuevos lenguajes, ya que parte de la implementación permanecería oculta y solo sería necesario implementar una la nueva clase en la que se codifiquen aquellos métodos que sean necesarios para cargar la configuración de los contenedores. Por otra parte, se requeriría también modificar la clase factoría `DockerContainerFactory` para que el método *createContainer* devuelva el tipo de clase hija de `DockerContainer` correspondiente. Aunque es posible que otras soluciones hubieran reducido más las líneas duplicadas en estas clases, esta es una solución que facilita la comprensión del uso de estas clases.

3.2.2. Implementación de pruebas automáticas

Intentando ejecutar las pruebas automáticas en local, desde IntelliJ, se encontraron numerosos errores, debido a dependencias faltantes, cuando estas se encontraban incluidas en el archivo *pom.xml* del proyecto. En un intento de solucionarlo, se probó borrando la caché de la aplicación, pero no tuvo éxito. Por otra parte, también se optó la opción de cambiar el ámbito del uso de la librería de JUnit de *test* a *compile*, de manera que se utilizara siempre. Esto solucionó el error, pero no era lo adecuado, así que se probó con la solución de borrar la caché

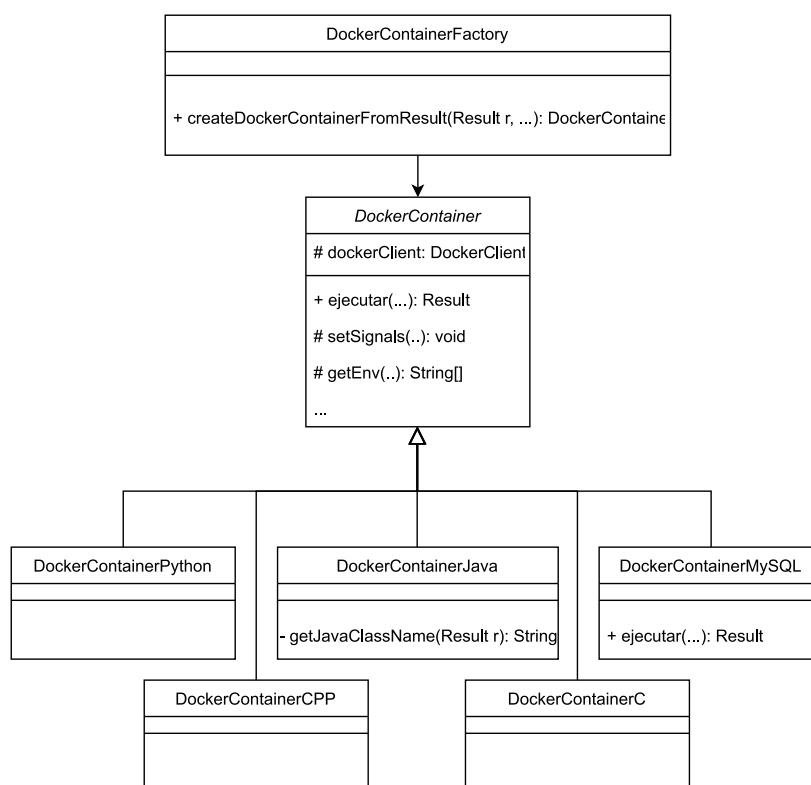


Figura 3.1: Diagrama de clases del patrón Simple Factory con las clases Docker-Container

de Maven del directorio `~/m2`, lo cual arregló el problema momentáneamente. Sin embargo, como más adelante volvió a dar el mismo error, se creó un nuevo fichero **docker-compose-tests.yml** que se usaría para poder ejecutar los *tests* en un entorno aislado. Asimismo, este nuevo fichero con algunas modificaciones posteriores es el que se utilizó para el flujo de integración continua y despliegue continuo de *GitHub Actions*.

Revisando las pruebas implementadas en anteriores trabajos, se observó que varias se encontraban ya realizadas, pero estaban desactivadas. Tras probar a activarlas, algunas de estas finalizaban su ejecución, pero muchas otras no. Algunos de los *tests* que daban error fue por el uso de un objeto `MultipartFile` que se encontraba simulado mediante un *mock*. Por este mismo motivo, cuando se simulaba uno de los métodos de los diferentes servicios que utilizaban este tipo de objetos, no realizaban las comprobaciones correctas, ya que no identificaban el objeto `InputStream` como si fuera el mismo. El motivo por el que ocurría esto, era que el método `MultipartFile.getInputStream` devolvía, según la documentación oficial de Spring ², un nuevo flujo de datos para los datos que se estaban leyendo de un mismo *socket*, pero cuando el servicio `ProblemService` de la aplica-

²<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/core/io/InputStreamSource.html>

ción consumía un objeto de la clase `InputStream`, provocaba que cuando se leyera de un socket diferente, generaba un socket diferente al que utilizaba el sustituto de la clase `MultipartFile`. Para solucionarlo se modificaron las cabeceras de los métodos para que estos, en vez de utilizar este tipo de objetos, utilizaran los correspondientes `InputStream`. Con esto se consiguió dejar de sustituir el uso de los `MultipartFile` por objetos de esta última clase.

Una vez corregidas las pruebas que ya se encontraban implementadas, se comenzó con la implementación de las restantes. Para poder proceder, se hizo uso de la herramienta SonarCloud, ya que dispone de una opción para observar cuáles son las líneas de código que no se encuentran cubiertas por pruebas, así como aquellas condiciones de una comprobación que no se han tenido en cuenta en alguna de las pruebas implementadas. Al finalizar, se procedió a cerrar otro gran bloque de la parte de implementación del trabajo.

3.2.3. Desarrollo de sistema de autenticación

Para comenzar con el bloque más denso, el correspondiente a la autenticación y autorización, se requirió realizar una valoración previa, pues la versión de Spring que se estaba utilizando es la 2.6.5, cuando las más actuales eran las correspondientes a la 3. Revisando los cambios entre las versiones, se observó que entre estas había una diferencia significativa, ya que se había modificado toda la librería de Spring para que trabajara con las dependencias de **jakarta** en vez de con las del paquete de **javax**. Esto, sobre todo en la dependencia que se iba a agregar de Spring Security suponía un gran cambio en toda la aplicación. Sin embargo, por comodidad ya que todo estaba establecido para la versión 2 de la librería, se decidió en un principio dejar la versión actual.

Diseño e implementación

Una vez tomada la decisión, se comenzó a diseñar el camino que seguiría la implementación de autenticación en la aplicación. Teniendo en cuenta que la aplicación no tendría la posibilidad de crear cuentas de cero, puesto que se contaría con el servidor LDAP de la universidad, el esquema que se diseñó para esta implementación es el que se puede observar en el diagrama de flujo de la figura 3.2. De esta manera, los usuarios, al querer iniciar su sesión en la aplicación, necesitarían iniciar sesión en el servidor de la universidad, de donde se recogería su información personal entre la que se incluiría el correo electrónico como identificador inicial y su nombre completo. Con esta información se comprobaría si ya tiene datos en *iudex* y, si no los tiene, se crea un usuario para almacenar su información en la base de datos de la aplicación y finalmente se le devuelven los códigos de autenticación para que pueda autenticarse con ellos.

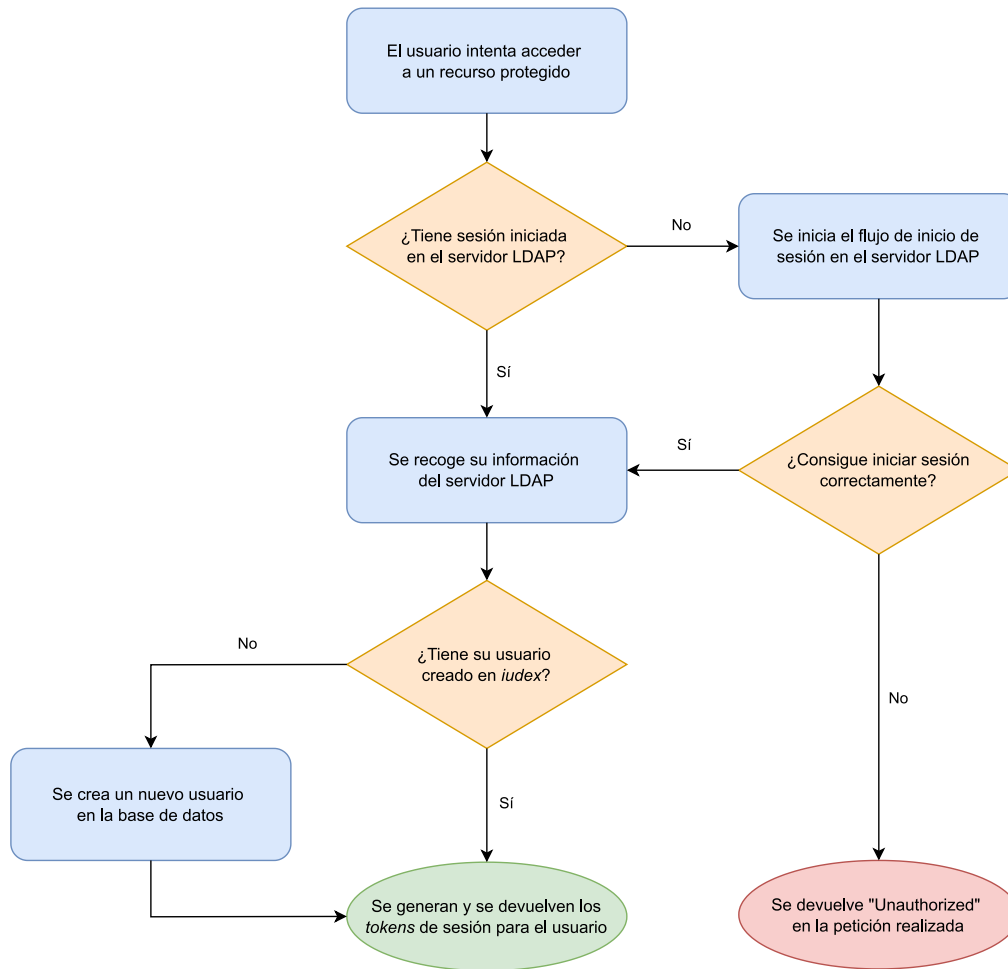


Figura 3.2: Diagrama de flujo del proceso de autenticación en *iudex*

Por otra parte, para la obtención de los datos, se ha utilizado el proceso de autenticación Single Sign-On, siguiendo el protocolo OAuth 2.0 [30]. Este protocolo define el proceso por el que se permite el acceso realizado por una aplicación concreta a un recurso que pertenece a un usuario, aunque no sea necesaria la presencia de este en el proceso. Por lo tanto, el objetivo de este protocolo es la autorización de aplicaciones de terceros a recursos de protegidos. En cuanto a los roles en la especificación de OAuth 2.0 encontramos los siguientes cuatro:

- *Resource owner*, o propietario del recurso: Entidad a la que pertenece un recurso que se encuentra almacenado en una localización remota. Este puede ser un usuario humano o no, pero siempre debe ser el usuario final.
- *Resource server*, o servidor del recurso: Entidad que almacena los recursos de los propietarios, permitiendo el acceso a estos mediante la aceptación de *tokens*, denominados generalmente *access tokens* o *tokens* de acceso.
- Cliente: Aplicación que pretende obtener permiso del usuario propietario

para poder acceder a los recursos almacenados por el servidor. Esta debe probar que tiene permiso mediante la posesión de un *token* de acceso.

- *Authorization server*, o servidor de autorización: Servidor central que expide *tokens* válidos a los clientes. Estos solo pueden ser expedidos si el propietario de un recurso lo ha permitido y dependiendo del protocolo utilizado, los tipos de *tokens* expedidos pueden variar.

En la figura 3.3 se puede observar el proceso seguido para la implementación, la autenticación de los usuarios de la aplicación *iudex* siguiendo dicho protocolo. En primer lugar, el cliente intentará acceder a un recurso protegido, por lo que la API, o servidor del recurso, le redirigirá a la página de inicio de sesión de la universidad, quien sería el servidor de autenticación. Este mostraría la página de inicio institucional para que el usuario pudiera iniciar su sesión. Tras esto, el servidor de autenticación validaría las credenciales y si el usuario inicia sesión correctamente, redirige a la aplicación la respuesta con los *tokens* del usuario. Una vez verificada la autenticación del usuario, se cargaría su información de la aplicación para, posteriormente, generar los tokens de acceso concretos de *iudex* que tendría que usar el cliente para poder acceder a los recursos del usuario.

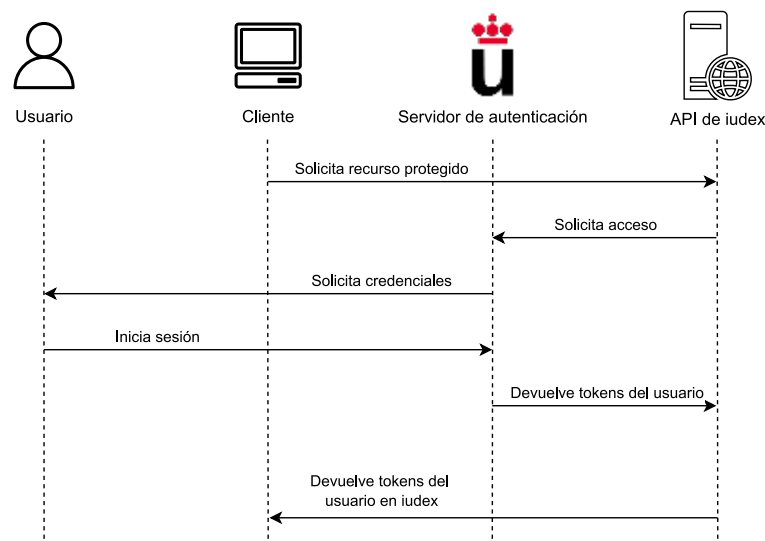


Figura 3.3: Diagrama de secuencia del proceso de autenticación con el protocolo OAuth 2.0

Mediante el uso de KeyCloak se facilita el trabajo de implementación de este proceso, ya que se encarga de toda la configuración de las comunicaciones entre el servidor de autenticación y la aplicación. Además de esto, KeyCloak cuenta con una consola de administración que agiliza varias tareas, pues de manera intuitiva y con una interfaz gráfica, se permite la configuración de los diferentes servidores de autenticación. Por otra parte, mediante el uso de la librería de Spring Security, se utiliza la abstracción implementada en esta para poder realizar la

autenticación mediante el protocolo OAuth 2.0. Mediante esta librería, se realizan los intercambios de peticiones entre el servidor de KeyCloak y el de la aplicación, así como contar con clases abstractas para las implementaciones concretas de la aplicación.

Para poder terminar de hacer funcionar la autenticación, fue necesaria la implementación de las clases que se encargarían de generar, filtrar y validar los *tokens* de la aplicación. Por otra parte, los *tokens* utilizados fueron basados en el estándar de los *JSON Web Tokens*, o *JWT*, puesto que este tipo de tokens no almacenan ninguna información sobre el historial de comportamiento del usuario y es uno de los estándares más utilizados y consolidados. Mediante la utilización de este tipo de *tokens*, se es capaz de cifrar la información esencial del usuario, de manera que no sea necesaria su autenticación repetidamente y, cuando el *token* que se esté utilizando caduque, se podrá solicitar otro mediante un *token* de refresco.

Roles de usuario

Una vez finalizada la autenticación en la aplicación, se comenzó a tratar con la autorización, y para ello se utilizarían tres roles diferentes:

- Usuario o alumno: Sería aquel tipo de usuario común, que podría realizar acciones simples como crear un equipo, ver los concursos, acceder a los problemas y datos, un concurso al que se haya apuntado, ver los datos de un problema, realizar envíos a un problema de un concurso al que tenga acceso, etc.
- Juez o docente: Sería aquel tipo de usuario que podría gestionar los problemas y concursos, así como los usuarios que participan en estos, entre otras acciones.
- Administrador: Sería aquel tipo de usuario con un nivel de permisos superior, pues tendría los permisos suficientes para modificar los permisos de todos los usuarios de la aplicación.

Una vez especificados los roles, se añadieron de manera orientativa los permisos requeridos en cada uno de los puntos de servicio de la aplicación para limitar el acceso.

Actualización de Spring

Para poder finalizar este bloque de autenticación y autorización, se probó a ejecutar las pruebas automáticas de la aplicación para poder así validar que todo

	Apache HttpClient 5	Zerodep	OkHttp	Netty	Jersey
Estabilidad	3	3	2	2	1
Soporte a largo plazo	Sí	Sí	?	No	No
Soporte para sockets de Unix	Sí	Sí	Sí	Sí	Sí
Soporte para Npipe de Windows	Sí	Sí	Sí	No	No
Soporte de acoplamiento de entrada estándar	Sí	Sí	Sí	Sí	No

Tabla 3.1: Valoración de las capas de transporte para la librería Docker-Java

funcionaba correctamente. Sin embargo, no todas las pruebas finalizaban correctamente. Tras investigar, se concluyó que los *tests* implementados no estaban adecuados para la versión de Spring Security utilizada y, por este motivo, se decidió actualizar la versión de Spring a su versión 3 y así poder actualizar también la versión de todos los paquetes de Spring utilizados, incluyendo la versión de Spring Security.

En cuanto a la actualización de la versión de Spring, el cambio más relevante fue el cambio en la librería que se utilizaba, puesto que en la versión 2 se utilizaba la librería de **javax** y, tras actualizar, se tuvo que modificar todas las importaciones de clases de esta librería para que importaran las del paquete de **jakarta**. Asimismo, en cuanto a la configuración de los filtros de seguridad se encontraron varios métodos deprecados, por lo que se tuvo que sustituir por los métodos actualizados correspondientes.

Además de estos cambios, la librería de **Docker-Java** también requería modificaciones. Esto se debía a que, por defecto, esta librería utilizaba los paquetes de **jersey** para la capa de transporte para las conexiones entre la aplicación y los contenedores que gestionaba, pero tras actualizar la versión, se recomienda el uso de otras librerías, puesto que se había dejado de dar soporte a esta. En la tabla 3.1 se puede observar una comparativa de las diferentes alternativas propuestas por los desarrolladores de la librería [31]. En esta tabla la estabilidad se mide con valores numéricos del 1 al 3, donde 1 significa un valor inferior y 3 un valor mayor, y donde “?” significa que está pendiente de valoración por parte de los desarrolladores. Finalmente, siguiendo estas recomendaciones, se optó por utilizar la librería Apache HttpClient 5.

Tras esto, se modificaron ligeramente las pruebas automáticas para que, en

aquellos que hacía falta la autenticación de un usuario, se utilizara uno simulado, de manera que se pudiera comprobar la funcionalidad de estos en vez de la parte de autorización. Asimismo, para poder probar esto, se crearon unos tests específicos con los que se comprobaban que diferentes usuarios simulados, con diferentes roles, pudieran o no acceder a recursos que requerían permisos específicos. Para poder ejecutar las pruebas de integración, se requería también un servidor de KeyCloak al que poder conectarse para probar las conexiones, por lo que, para simplificar este proceso, se añadió un nuevo contenedor al archivo de configuración **docker-compose-tests.yml**. Con todo esto, se consiguió finalizar la parte de autenticación y autorización de la aplicación y, con la intención de hacer la aplicación más mantenible de cara al futuro, se aprovechó para actualizar la mayoría de sus dependencias.

3.2.4. Inclusión de la tecnología de WebSockets

Mientras se trabajaba en el anterior bloque, en una de las reuniones mantenidas con los tutores, se propuso la idea de implementar *WebSockets*, lo que permitiría reducir la carga que soportaría la aplicación cuando esta contara con varios usuarios simultáneos. Esta era una idea que ya fue planteada para trabajos anteriores, pero que no se había llegado a realizar, por lo que se comenzó a trabajar en ella. Para poder implementarlos más fácilmente, se decidió utilizar otra librería de la familia de Spring, la cual cuenta ya con mucha de la configuración necesaria ya implementada.

Tipos de mensajes

En primer lugar, se especificaron los tipos de mensajes que se iban a enviar. Como todavía no se tenía claro, puesto que la parte de la interfaz todavía estaba en desarrollo, se pensó en dejar implementados dos tipos de mensajes. El primero de ellos serían generales y servirían para que los administradores pudieran enviar avisos a todos los usuarios que se encuentren utilizando la aplicación en ese momento, los cuales serían fácilmente configurables ya que permitirían al administrador personalizar el mensaje que se quiera enviar. Por otra parte, los otros mensajes serían más específicos, y con el fin de dejar algún claro ejemplo para trabajos futuros, se decidió hacerlo sobre los envíos, en los que avisaría del equipo, a qué concurso y a qué problema lo ha realizado.

Publicadores

Tras la implementación de los mensajes, se especificaron los publicadores que enviarían dichos mensajes por los diferentes canales o *sockets*. Los cuales serían

los siguientes cuatro:

- Simple: Permitiría la recepción de los mensajes generales. Para poder hacer uso de estos mediante la API, se podría implementar un punto de servicio al que solo los administradores pudieran acceder para mandar el texto que quieran y que luego se envíe por este canal.
- Envíos: Estos serían para los tipos de mensajes específicos mencionados, para los que se han implementado tres canales diferentes:
 - Globales: Este canal serviría para recibir mensajes de todas los envíos correctos de concursos públicos, de manera que aquellos usuarios que quieran estar informados, puedan hacerlo mediante este canal. Sin embargo, aunque este canal se encuentra implementado, de momento no se envía ningún mensaje por este.
 - Por concurso: Este canal permitiría que se recibieran mensajes de los envíos correctos de un concurso en concreto. De esta manera, se podría mantener informados a todos los participantes de un concurso mediante la suscripción a este canal y así estar más pendientes del progreso de este.
 - Por concurso y equipo: Mediante este canal se recibirían todos los mensajes de los envíos **no correctos** de un equipo a al concurso en el que estén participando. De esta manera, no necesitarían recargar constantemente la página tras realizar un envío, ya que en el momento en el que la validación de este terminara, sería notificado. Por otra parte, se ha dejado que por este canal solo enviaran aquellos envíos no correctos para que los equipos no recibieran avisos duplicados, ya que la idea sería que estuvieran suscritos tanto a este canal como al anterior.

4

Resultados y validación

A lo largo de esta sección, se van analizar las métricas de las mejoras de calidad de la aplicación realizadas mediante el uso de la herramienta SonarCloud, así como la validación de que aquellas tareas que han sido realizadas. De esta manera, se observará el progreso de la aplicación y el correcto funcionamiento de esta tras la finalización de este trabajo.

4.1. Métricas de calidad

Como ya se ha mencionado, para poder realizar una comparativa de las métricas de calidad del software se va a utilizar la herramienta SonarCloud pues, además de realizar un análisis del código de la aplicación, también permite la generación de gráficos para poder realizar comparativas con anteriores versiones de la aplicación. A continuación, se va a desarrollar el progreso en cada una de las métricas que han sido tratadas en el alcance de este trabajo.

4.1.1. Bugs

En primer lugar, se priorizó la corrección de fallos en el código. En cuanto a los análisis de SonarCloud, antes de introducir nuevos cambios en la rama principal de la aplicación, se encontraron 8 bugs.

En la Pull Request **#135**¹, que son de cambios realizados, específicamente la que implementaba una solución temporal a la duplicación de clases `DockerContainer`, se arreglaron 5 de los bugs, puesto que se repetían en las 5 clases. Este fue arreglado en el cambio **04cb5d3** después de que todas las clases `DockerContainer` fueran juntadas en una sola clase.

Por otra parte, otros 2 bugs fueron arreglados en los commits **836e021** y **a2f80f7**, los cuales se encontraban el tratamiento de algunas excepciones, mientras que el último se arregló en la Pull Request **#153**².

De esta manera, en SonarCloud podemos comprobar la evolución del número de bugs en la aplicación durante el desarrollo de este trabajo en la tabla 4.2. Como se puede observar, desde junio que se realizó el último cambio en la rama principal de la aplicación antes de comenzar, hasta mediados de julio donde se introdujeron los cambios mencionados, se observa un decremento en el número de bugs, de 8 a 0. Asimismo, a lo largo del resto de las tareas que se han realizado, se ha conseguido mantener el número de bugs abiertos en la aplicación a 0.

4.1.2. Vulnerabilidades

Además de arreglar los bugs encontrados en los análisis, también se estuvieron analizando las vulnerabilidades. Estas, en vez de ser fallos que impiden que la aplicación funcione correctamente, son fallos encontrados en el sistema software que, al tratarse de una aplicación, pueden ser explotados por un atacante para comprometer la seguridad de este y permitir la materialización de una amenaza.

En un primer lugar, cuando se comenzó con el desarrollo de la aplicación, en el análisis realizado por SonarCloud se pudo observar que la aplicación contaba con 6 vulnerabilidades. Estas se encontraban en secciones del código en las que se escribían en los registros de actividad de la aplicación datos que podían ser introducidos por usuarios. De esta manera, los usuarios podían llegar a utilizar estos registros para obtener información al introducir cadenas de caracteres maliciosas³.

Tras conseguir solventar estas incidencias, se consiguió reducir el número de vulnerabilidades a 0, tal como se puede observar en la tabla 4.2. Asimismo, desde este momento, se consiguió mantener el número de vulnerabilidades en el código de la aplicación en esta misma cifra.

¹<https://github.com/URJC-CP/iudex/pull/135>

²<https://github.com/URJC-CP/iudex/pull/153>

³CWE-20, CWE-117

4.1.3. Líneas duplicadas

Por otra parte, se trataron de reducir las líneas duplicadas en el código de la aplicación. Revisando los valores de los análisis de SonarCloud, se observa que antes de comenzar con la realización de este trabajo la aplicación contaba con 694 líneas duplicadas, lo que suponía un 8,4 % de esta con código repetido.

Analizando las métricas proporcionadas, se observaron que la mayoría de estas líneas se encontraban en las clases `DockerContainer` de los diferentes lenguajes pues, como se ha mencionado anteriormente, tanto las clases `DockerContainerC`, `DockerContainerCPP`, `DockerContainerJava`, `DockerContainerMySQL` y `DockerContainerPython` se encontraban prácticamente duplicadas. Tras leer el código, se comprobó que las únicas de las diferencias entre estas clases eran los diferentes valores de las variables de entorno que se pasan a los contenedores que se encargan de ejecutar el código.

Una vez ya implementado el patrón de diseño `Simple Factory`⁴, utilizando las métricas proporcionadas se puede apreciar que se consiguió reducir la cantidad de líneas duplicadas a 521, lo que supone un 5,6 % del código de la aplicación y con una mejora del 25 % frente a los valores anteriores. Estos datos se pueden ver reflejados en la tabla 4.2.

4.1.4. Cobertura del código

Tras mejorar las pruebas automáticas de la aplicación, se buscó aumentar la cobertura del código por estas, de manera que se pudiera comprobar el correcto funcionamiento de la aplicación antes de introducir cambios nuevos en la rama principal de manera automatizada. En concreto, se propuso aumentar la cobertura del código de los puntos de servicio de la aplicación, los cuales se encontraban el paquete `es.urjc.etsii.grafo.iudex.api.v1`. Como se puede observar en la tabla 4.1, se consiguió que la mayoría de los puntos de servicio alcanzaran un 100 % de cobertura por pruebas automáticas, teniendo un total de 97,4 % de cobertura en todo el paquete.

Aquellos que no alcanzado dicha cifra son por situaciones específicas que tenían un grado alto de complejidad, ya que estaban relacionados con el ámbito de la autenticación de usuarios y dependían de respuestas de servidores externos.

De una manera global, tal como se muestra en la tabla 4.2, se puede observar en toda la aplicación una cobertura del 16,5 % en el análisis previo al comienzo del trabajo. Finalmente, cuando se terminaron de realizar todas las implementaciones

⁴Pull Request #165: <https://github.com/URJC-CP/iudex/pull/165>

Clase	Cobertura	Líneas sin cubrir	Condiciones sin cubrir
api/v1	97,4 %	6	7
APIUserController	100 %	0	0
APITeamController	100 %	0	0
APISubmissionController	100 %	0	0
APILanguageController	100 %	0	0
APIContestController	100 %	0	0
APIAdminController	100 %	0	0
APIProblemController	97,7 %	1	2
APIErrorController	50,0 %	2	3
APIOAuthController	16,7 %	3	2

Tabla 4.1: Estadísticas de cobertura por pruebas automáticas de puntos de servicio de la aplicación

de las pruebas automáticas⁵⁶, esta cifra alcanzó un 32 % de cobertura, lo cual implica un incremento de prácticamente un 100 % sobre el valor inicial.

A continuación se puede observar en la tabla 4.2 un resumen de la evolución de todas las métricas mencionadas, de manera que se pueden observar los valores antes del comienzo de este trabajo y los resultados del análisis una vez terminado.

Métrica	Antes	Después
Bugs	8	0
Vulnerabilidades	6	0
Líneas duplicadas	694	521
Líneas duplicadas (%)	8,4 %	5,6 %
Cobertura (%)	16,5 %	32 %

Tabla 4.2: Comparativa de métricas de calidad

⁵Pull Request #136: <https://github.com/URJC-CP/iudex/pull/136>

⁶Pull Request #143: <https://github.com/URJC-CP/iudex/pull/143>

4.2. Validación de funcionamiento

A continuación se va a comentar cómo se ha validado el correcto de las nuevas funcionalidades implementadas en la aplicación a lo largo de este trabajo.

4.2.1. Pruebas automatizadas

Para la validación de las pruebas automáticas, una vez estuvieron todas implementadas, se pueden ejecutar todas mediante el comando **docker compose -f docker-compose-tests.yml up**. De esta manera, se obtiene un resumen con el número de pruebas que se han ejecutado satisfactoriamente, cuántas se han saltado porque tengan una anotación concreta, y cuántas se han encontrado una excepción en su ejecución.

Con el estado de la aplicación, y ejecutando el comando **mvn clean test**, ya que antes el archivo **docker-compose-tests.yml** no existía, se obtiene un resumen incluido en la tabla 4.3 en el que indica que se han ejecutado satisfactoriamente 44 pruebas, otras 30 se han saltado y 0 han fallado, aunque en numerosas ocasiones saltan excepciones al no lograr conectar con la base de datos.

Por otra parte, ejecutando el comando inicial utilizando el fichero de **docker compose** una vez finalizada las implementaciones descritas en la sección 3.2.2 de este trabajo, se obtienen unos resultados diferentes. En este resumen de resultados añadido en la tabla 4.3 se puede apreciar que el número de pruebas que se han ejecutado satisfactoriamente es 61, se han saltado 11 pruebas y en ninguna de ellas se han encontrado algún error.

Tests ejecutados	Antes	Después
Correctamente	44	61
Fallidos	0	0
Con excepciones	0	0
Omitidos	30	11
Tiempo de ejecución	34,721 s	58,006 s

Tabla 4.3: Comparativa de resultados de las pruebas automáticas

De esta manera, cuando se crea alguna *Pull Request* en el repositorio de la aplicación en GitHub, estas pruebas se ejecutarán automáticamente por lo especificado en el flujo de trabajo definido⁷ y, según se va trabajando en ella, GitHub

⁷<https://github.com/URJC-CP/iudex/blob/master/.github/workflows/build.yml>

te indica si se han ejecutado todas las pruebas correctamente para comprobar si se puede juntar con la rama principal.

4.2.2. Autenticación y generación de tokens

En cuanto a la autenticación y generación de tokens JWT⁸, se probó el punto de servicio `/API/v1/oauth/login`, el cual se encarga de guiar al usuario que está utilizando la aplicación durante el proceso de inicio de sesión explicado en la sección 3.2.3. Para probarlo adecuadamente, se ha utilizado el navegador web Firefox. Este punto de servicio sería el que se encargaría de crear el usuario nuevo en la aplicación dada la información del servidor de autenticación en el caso de que no existiera previamente en la aplicación.

En primer lugar, probando a acceder al recurso protegido `/API/v1/contest` nos inicia el flujo de autenticación delegada, tal como se puede observar en la figura 4.1. Todas estas redirecciones son realizadas por el servidor de KeyCloak con dominio `idp.numa.host`, el servidor de autenticación de la universidad y derivados. Asimismo, se puede observar como la vista para iniciar sesión se obtiene de la última petición, con dominio `identifica.urjc.es`.

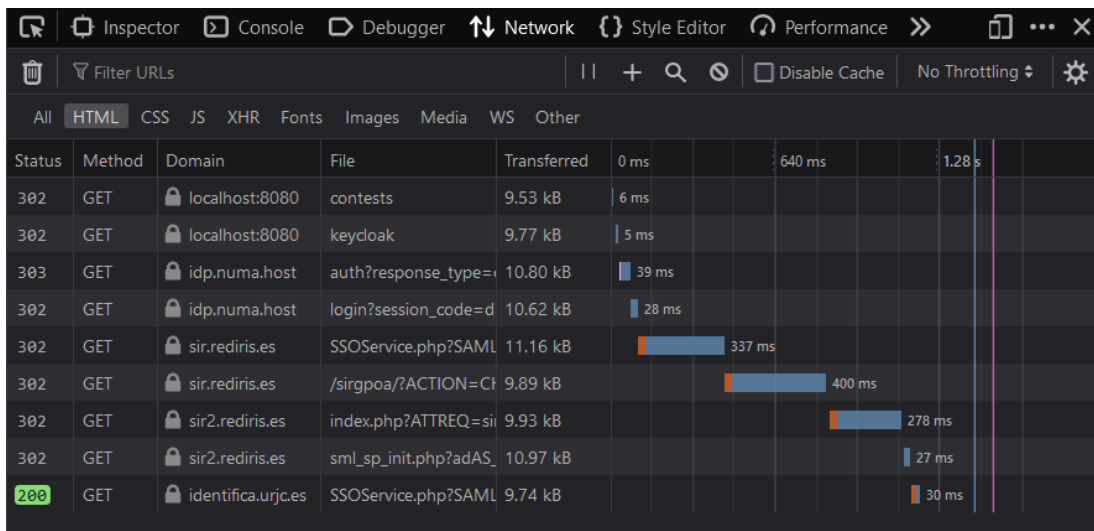


Figura 4.1: Secuencia de redirecciones de peticiones al solicitar un recurso protegido

Además de intentando acceder a un recurso protegido, se puede iniciar el flujo de autenticación mediante el punto de servicio introduciendo la URL completa de la aplicación para el registro e inicio de sesión. Al no tener una sesión iniciada, el servidor redirige al servicio de autenticación, el cual al estar utilizando el entorno

⁸Pull Request #151: <https://github.com/URJC-CP/iudex/pull/151>

de producción sería el servidor de la universidad de la figura 4.2, pero si, por el contrario, se utiliza el entorno de desarrollo redirigiría al servidor de KeyCloak proporcionado. Al conseguir iniciar sesión correctamente, se redirige al mismo punto de servicio de la aplicación para que esta pueda reconocer la sesión del usuario y comenzar a utilizar las cuentas almacenadas en la base de datos de la aplicación. Tras obtener el usuario de esta, genera los tokens y en la respuesta del flujo de peticiones se obtendría un objeto en formato JSON como se muestra en la figura 4.3 con la siguiente información:

- **status:** Almacena el valor *SUCCESS* o *FAILURE*, para indicar si ha finalizado el inicio de sesión satisfactoriamente o ha ocurrido algún error en el proceso, respectivamente.
- **message:** Devuelve el mensaje correspondiente al estado. Si ha ocurrido algún error en flujo de inicio de sesión, indica cuál ha sido.
- **error:** Indica el error de la excepción que ha ocurrido en la aplicación.
- **accessToken:** Incluye el valor del token que debe utilizar el usuario para autenticar todas las peticiones que se realicen con la aplicación.
- **refreshToken:** Almacena el token que debe utilizar el usuario para que, cuando le caduque el *accessToken*, pueda solicitar otro sin tener que pasar por todo el proceso de inicio de sesión.

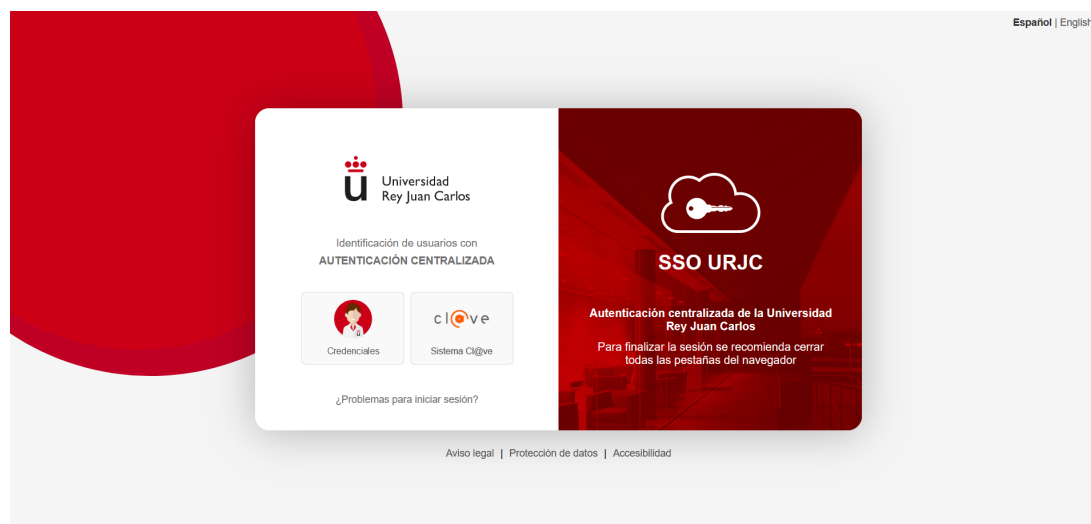


Figura 4.2: Vista del servicio de autenticación centralizada de la universidad

De esta manera, utilizando el cliente Insomnia, se puede acceder a un recurso protegido de la API especificando en la autenticación el *accessToken* devuelto en el anterior flujo y estableciendo el tipo a *Bearer*, tal como se muestra en la figura 4.4.

4.2.3. Autorización

Una vez se cuenta con el token de acceso a la aplicación, los diferentes recursos proporcionados por los puntos de servicios, requieren que los usuarios cuenten con un rol específico. De esta manera, por ejemplo y como se puede observar en la figura 4.5, los usuarios con rol **USER** podrían acceder a la información de todos los concursos desde el punto de servicio **GET /API/v1/contest**, pero solo aquellos que tengan el rol **JUDGE** podrían crear concursos nuevos desde el punto de servicio **POST /API/v1/contest**.

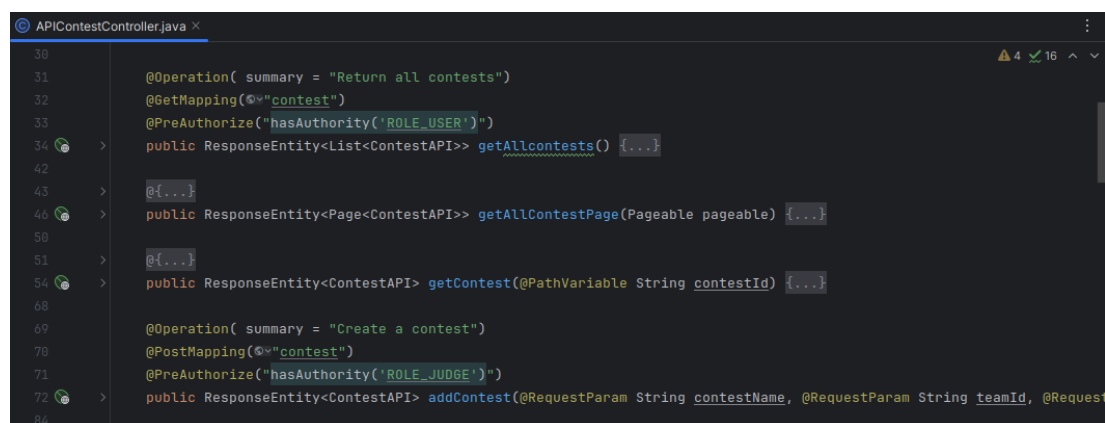


Figura 4.5: Roles requeridos en los puntos de servicio de la clase **APIContestController**

De esta manera, cuando un usuario con solo el rol **USER** intenta añadir un nuevo problema desde el punto de servicio **/API/v1/problem/fromZip**. Sin embargo, tal como se observa en la figura 4.6, la API le devuelve el código de error **401 Unauthorized**. Por otra parte, tras añadirle el rol **JUDGE**, en la figura 4.7 se verifica que el servidor ha realizado la acción solicitada, puesto que ahora el usuario cuenta con los roles requeridos.

4.2.4. WebSockets

Finalmente, se procedió a la validación de la implementación de los WebSockets⁹. Para ello, se ha desarrollado una interfaz web en la que poder probarlos cómodamente¹⁰. Una vez levantada la aplicación, esta interfaz es accesible desde la ruta **/test.html**.

Con esta interfaz, se puede suscribir a varios WebSockets y, cuando se reciba un mensaje por estos, se mostrará como una notificación en la esquina superior

⁹Pull Request #160: <https://github.com/URJC-CP/iudex/pull/160>

¹⁰Pull Request #166: <https://github.com/URJC-CP/iudex/pull/166>

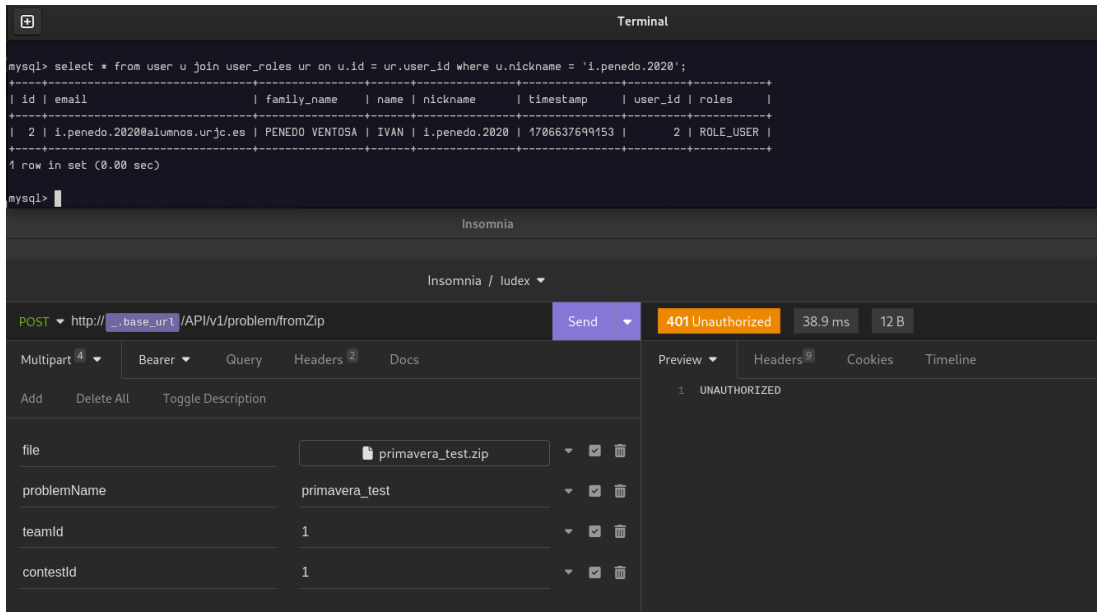


Figura 4.6: Respuesta de error al acceder a un recurso compartido sin el rol adecuado

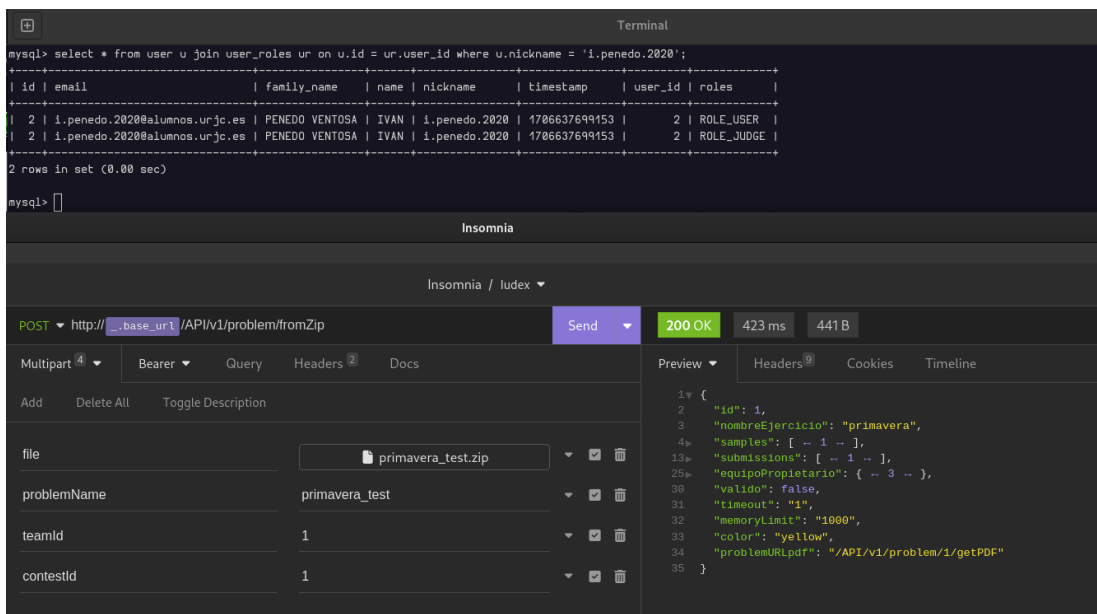


Figura 4.7: Respuesta correcta al acceder a un recurso compartido con el rol adecuado

izquierda. Esta interfaz también podrá servir de ejemplo para incorporar esta tecnología en el frontend de la aplicación en un futuro. Asimismo, ya se encuentra suscrito al canal por el que se envían los avisos generales de la aplicación en forma de mensajes simples.

Para validar el funcionamiento de los diferentes WebSockets, se realizaron las siguientes pruebas:

- **Simple:** Al ya estar suscrito a este canal, tan solo haría falta redactar un nuevo mensaje de aviso en el cuadro de texto correspondiente y pulsar el botón de **Broadcast**.
- **Envíos:** Estos serían para los tipos de mensajes específicos existentes, hará falta incluir en los cuadros de texto los correspondientes ID de un concurso e ID de un equipo que sean necesarios para después conectarse al canal deseado mediante el canal pertinente. Después de esto, es necesario que se realice un envío que se ajuste al WebSocket al que se ha suscrito para comprobar que llega la notificación adecuada según lo expuesto en la sección 3.2.4.

De esta manera, en la figura 4.8 se pueden observar diferentes notificaciones enviadas mediante diferentes WebSockets.

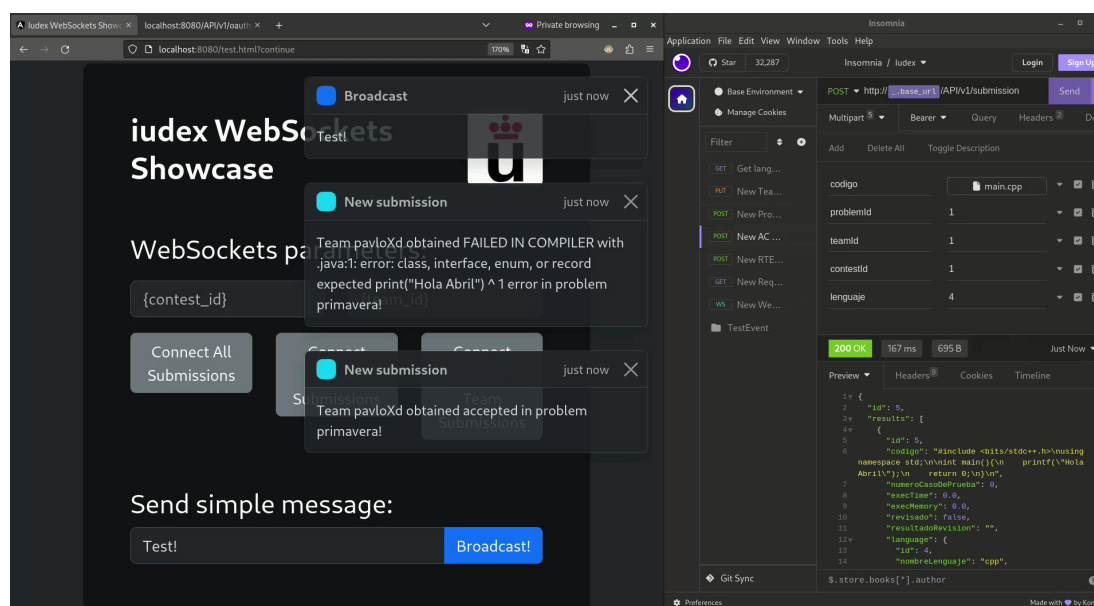


Figura 4.8: Validación de funcionamiento de WebSockets

Finalmente, en el histórico de cambios del repositorio de la aplicación¹¹ se

¹¹<https://github.com/URJC-CP/iudex/commits/master/?author=xIvqn>

pueden observar todos los cambios introducidos en esta durante la realización de este Trabajo de Fin de Grado.

5

Conclusiones y trabajos futuros

A continuación se desarrollarán las conclusiones alcanzadas durante el desarrollo de este trabajo. Además de esto, se comentan algunas propuestas para trabajos futuros que se han ido encontrando para que otros alumnos o desarrolladores puedan aportar a esta aplicación.

5.1. Conclusiones

Este Trabajo de Fin de Grado ha consistido en la evolución de un juez de evaluación de código automático para que se pueda utilizar en asignaturas de programación de la Universidad Rey Juan Carlos. Con este trabajo, se ha mejorado la aplicación en desarrollo *iudex* sobre la que ya se habían realizado otros cuatro trabajos por parte de antiguos alumnos de la universidad. Cuando se encuentre en una versión estable, esta aplicación podría ser utilizada por los docentes de varias asignaturas de programación para evaluar a los alumnos.

Asimismo, al ser una aplicación de código abierto, otras universidades, asociaciones o usuarios podrían utilizar la aplicación, o incluso modificarla para ajustarla a sus necesidades. De esta manera, se podría utilizar tanto para clases de programación, como por ejemplo para concursos de programación, el cual es otro de los usos comunes de los jueces de código automáticos observados.

Durante la realización del Trabajo de Fin de Grado, se ha adquirido un conocimiento amplio sobre autenticación en aplicaciones, autenticación delegada y

tokens JWT, así como sobre diferentes métodos de “sign-on” y aplicaciones que facilitan la implementación de estas como KeyCloak. Por otra parte, se ha profundizado en conocimientos sobre calidad del código, mantenibilidad y evolución de productos software, además de temas como la Integración Continua y Despliegue Continuo, con lo que no había llegado a trabajar de manera práctica durante la carrera. Por último, se ha aprendido a trabajar con L^AT_EX para la redacción de este documento, y con la tecnología de virtualización de contenedores y Docker.

En cuanto a los objetivos propuestos, tal como se ha desarrollado en el capítulo 3, las tareas realizadas han consistido en la mejora de la aplicación en cuanto a calidad del código, métricas y mantenimiento, mejora de pruebas automáticas que permitan la comprobación del correcto funcionamiento de la aplicación, implementación de un sistema de autenticación en la aplicación, y más concretamente de autenticación delegada mediante los servidores de la Universidad Rey Juan Carlos, y la implementación de la tecnología de WebSockets para reducir la carga de trabajo de la aplicación. Finalmente, se puede comprobar mediante la validación realizada expuesta en el capítulo 4 que todos estos objetivos han sido alcanzados en su completitud.

5.2. Trabajos futuros

Asimismo, durante la realización de este trabajo, se han observado algunas posibles tareas para mejorar la aplicación o para incluir nuevas funcionalidades. A continuación se comentarán varios aspectos de mejora encontrados.

5.2.1. Mejora de la interfaz web

La versión actual de la interfaz web cuenta con pocas funcionalidades de las que la API de la aplicación aporta. Por este motivo, haría falta incluir aquellas funcionalidades que falten para obtener una interfaz utilizable por los alumnos durante las asignaturas. Además de esto, serían necesarias varias modificaciones que hagan de esta interfaz un producto más fácil de utilizar, ya que muchas de las opciones que permite, no son nada intuitivas, y a un usuario podría serle difícil saber dónde tiene que pulsar para hacer alguna acción. Por lo tanto, se propone la realización de un análisis de usabilidad y accesibilidad, para permitir que un mayor número de usuarios pueda utilizar más cómodamente la aplicación.

5.2.2. Mejora de la documentación de la API

Actualmente, el juez cuenta con documentación para la API, pero esta es autogenerada y cuenta con escasa información útil. Aprovechando que ya se uti-

lizan las librerías de Swagger y OpenAPI, se podría aumentar la información de utilización de los puntos de servicio mediante el uso de anotaciones. De esta manera, aquellas personas que quieran trabajar en la mejora de la aplicación, podrían tener una documentación más exhaustiva para incentivarles a trabajar en su desarrollo.

5.2.3. Realización auditorías y mejorar la seguridad

Durante las diferentes implementaciones que se han realizado y, en específico, durante las mejoras de calidad de la aplicación, se han observado algunos fallos de seguridad. Estos podrían provocar la exposición de datos a los que los usuarios no deberían tener acceso, como los encontrados durante la realización de este trabajo y por lo que se han modificado algunos registros de ejecución de la aplicación. Sin embargo, es probable que el juez cuente con otros fallos de seguridad.

Por este motivo, se propone la realización de auditorías de seguridad, las cuales son procesos por lo que el auditor encuentra fallos de seguridad en una aplicación y los expone en un documento formal para que estos puedan ser solventados por los desarrolladores. Una vez finalizado este proceso, se obtendría una aplicación mucho más segura y con una mayor integridad.

5.2.4. Añadir generación de informes

En cuanto a nuevas funcionalidades, se propone la implementación de un servicio para generar informes. Este permitiría a los profesores que, una vez terminara un concurso, puedan solicitar un documento que se genere automáticamente sobre el que observar las estadísticas y el rendimiento de los alumnos o participantes del concurso. Este documento sería de gran utilidad para poder realizar algunos trabajos analizando los datos obtenidos mediante la utilización del juez y poder compararlo con los resultados obtenidos mediante el uso de otros métodos de evaluación. Esto permitiría seguir mejorando el juez y perfeccionando diferentes métodos de evaluación.

5.2.5. Conexión con calificaciones del Aula Virtual

Profundizando en la anterior propuesta, también se podría analizar la integración de la aplicación de evaluación de código automática con la página del Aula Virtual de la Universidad Rey Juan Carlos. De este modo, se podría intentar conectar los concursos que se realicen en la aplicación para que, cuando estos finalicen, dependiendo de diferentes parámetros como podrían los resultados por tiempo utilizados en los concursos convencionales de programación competitiva o

el número de problemas resueltos, se pudieran asignar unas notas determinadas a todos los alumnos que hayan participado en estos.

Con esta conexión se incentivaría el uso de esta aplicación por parte de los profesores, ya que les facilitaría la calificación de los estudiantes. Por otra parte, los alumnos podrían obtener de forma más inmediata de sus calificaciones obtenidas en pruebas de programación.

Bibliografía

- [1] A. Rivoir, *Globalización y digitalización.: Reflexiones en torno a las consecuencias de la pandemia por la pandemia por COVID-19 iniciada en 2020*. CLACSO, 2022, pp. 39–48. [Online]. Available: <http://www.jstor.org/stable/j.ctv3142tw7.6>
- [2] “El proceso de digitalización de la sociedad española se consolida y se amplía en el marco de la pandemia con nuevos usos como el teletrabajo - Fundación BBVA — fbbva.es,” <https://www.fbbva.es/noticias/el-proceso-de-digitalizacion-de-la-sociedad-espanola-se-consolida-y-se-amplia-en-el-marco-de-la-pandemia-con-nuevos-usos-como-el-teletrabajo/>, [Accessed 04-01-2024].
- [3] X. M. Garcia, “Transformación digital y pandemia: siete tendencias educativas para la era pos-covid-19,” *Blog del eLearning Innovation Center*, 2020. [Online]. Available: <https://blogs.uoc.edu/elearning-innovation-center/es/transformacion-digital-y-pandemia-siete-tendencias-educativas-para-la-era-pos-covid-19/>
- [4] L. Fernández-Jambrina, “Experiencia de aula invertida en matemáticas para la ingeniería durante la pandemia,” *Revista educación, investigación, innovación y transferencia*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:254717199>
- [5] J. A. Miller and S. Cooper, “Case studies in game-based complex learning,” *Multimodal Technologies and Interaction*, vol. 5, no. 12, p. 72, Nov 2021. [Online]. Available: <http://dx.doi.org/10.3390/mti5120072>
- [6] M. M. Rahman, Y. Watanobe, A. Shirafuji, and M. Hamada, “Exploring automated code evaluation systems and resources for code analysis: A comprehensive survey,” 2023.
- [7] R. Mello, “Predicting the performance of job applicants in coding tests,” in *Bachelor of Science Thesis*, 2017. [Online]. Available: <http://hdl.handle.net/2077/52659>
- [8] J. Eldering, G. Nicky, K. Johnson, T. Kinkhorst, M. Pluijmaekers, M. Vasseur, and T. Werth, *About DOMJudge*, October 2023, version 8.2. [Online]. Available: <https://www.domjudge.org/about>
- [9] I. C. P. Contest, “Icpc 2021 fact sheet,” 2021. [Online]. Available: <https://icpc.global/community/history/world-finals-2020-factsheet.pdf>
- [10] M. A. Revilla, S. Manzoor, and R. Liu, “Competitive learning in informatics: The uva online judge experience,” *Olympiads in Informatics*, vol. 2, no. 10, pp. 131–148, 2008.
- [11] C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas, “Automated assessment and experiences of teaching programming,” *J. Educ. Resour. Comput.*, vol. 5, no. 3, p. 5–es, sep 2005. [Online]. Available: <https://doi.org/10.1145/1163405.1163410>
- [12] Y. Supriyati, D. Susanti, and S. Maulana, “Computer-based application for high school physics exams using irt model 1p,” in *AIP Conference Proceedings*, vol. 2320, no. 1. AIP Publishing, 2021.

-
- [13] P. L. Parrilla, J. S.-O. Calvo, and I. L. Osorio, “Diseño e implementación de una arquitectura basada en contenedores para la automatización de la evaluación de código,” Trabajo de Fin de Grado en Ingeniería de Computadores, 2020/2021.
- [14] —, “Desarrollo de una plataforma de análisis automático de código,” Trabajo de Fin de Grado en Ingeniería Informática, 2020/2021.
- [15] S. Giri, J. M. C. Verdugo, and R. M. Santamaría, “Nuevas funciones en juez de código online,” Trabajo de Fin de Grado en Ingeniería Informática, 2020/2021.
- [16] M. S. Alonso, J. M. C. Verdugo, and R. M. Santamaría, “Diseño e implementación de una arquitectura basada en contenedores para la automatización de la evaluación de código,” Trabajo de Fin de Grado en Ingeniería Informática, 2021/2022.
- [17] M. Corporation, “WebSockets - Referencia de la API Web — MDN — developer.mozilla.org,” https://developer.mozilla.org/es/docs/Web/API/Websockets_API, [Accessed 18-11-2023].
- [18] D. López-Mora, M. Villamar-Coloma, Á. Bravo-Pino, and E. Lozano-Rodríguez, “El uso de las metodologías ágiles y su importancia para el desarrollo de software,” *Killkana Técnica*, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:204160475>
- [19] G. G. V. Camacho, “Metodologías ágiles en tfgs,” 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:170248490>
- [20] A. Akhtar, B. Bakhtawar, and S. Akhtar, “Extreme programming vs scrum: A comparison of agile models,” *International Journal of Technology, Innovation and Management (IJTIM)*, vol. 2, no. 2, p. 80–96, Oct. 2022. [Online]. Available: <https://journals.gaftim.com/index.php/ijtim/article/view/77>
- [21] M. Mythily, A. Samson Arun Raj, and I. Thanakumar Joseph, “An analysis of the significance of spring boot in the market,” in *2022 International Conference on Inventive Computation Technologies (ICICT)*, 2022, pp. 1277–1281.
- [22] “Spring Framework — spring.io,” <https://spring.io/projects/spring-framework/>, [Accessed 02-01-2024].
- [23] A. Bucko, K. Vishi, B. Krasniqi, and B. Rexha, “Enhancing jwt authentication and authorization in web applications based on user behavior history,” *Computers*, vol. 12, no. 4, 2023. [Online]. Available: <https://www.mdpi.com/2073-431X/12/4/78>
- [24] Keycloak, “Keycloak documentation 23.0.3.” [Online]. Available: <https://www.keycloak.org/documentation>
- [25] A. Chatterjee and A. Prinz, “Applying spring security framework with keycloak-based oauth2 to protect microservice architecture apis: A case study,” *Sensors*, vol. 22, no. 5, p. 1703, Feb. 2022. [Online]. Available: <http://dx.doi.org/10.3390/s22051703>
- [26] “Mockito framework site — site.mockito.org,” <https://site.mockito.org/>, [Accessed 02-01-2024].
- [27] “SonarCloud Online Code Review as a Service Tool — sonarsource.com,” <https://www.sonarsource.com/products/sonarcloud/>, [Accessed 02-01-2024].
- [28] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, p. 2, 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:62479797>
- [29] V. Sarcar, *Design Patterns in C#: A Hands-on Guide with Real-World Examples*. Apress, 2018. [Online]. Available: <http://dx.doi.org/10.1007/978-1-4842-3640-6>

BIBLIOGRAFÍA

- [30] K. Dodanduwa and I. Kaluthanthri, “Role of trust in oauth 2.0 and openid connect,” in *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*, 2018, pp. 1–4.
- [31] “Available transports, docker-java documentation,” <https://github.com/docker-java/docker-java/blob/main/docs/transport.md>, [Accessed 05-01-2024].

