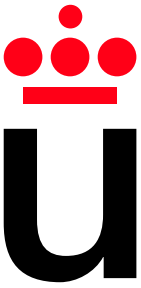

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería Informática

Curso 2024-2025

Trabajo Fin de Grado

ALCALAPP: UNA APLICACIÓN WEB PARA LA GESTIÓN DE TAREAS E INCIDENCIAS EN EQUIPOS DE DESAROLLO

Autor: Elisa Sofía Alcalá Guerrero
Tutor: Michel Maes Bermejo

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia, en especial a mis padres, los cuales han dedicado todo su amor y esfuerzo en mi desarrollo personal y académico, siendo un vivo ejemplo de trabajo, perseverancia y dedicación.

Agradecer también a Alex, por su apoyo y amor incondicional. Gracias por darme fuerzas cuando más las necesito y celebrar mis logros como tuyos.

Por último, me gustaría agradecer a mi tutor Michel Maes, por servirme de guía y apoyo durante todo este proceso, por su gran compromiso, disposición e implicación desde el primer momento.

Resumen

Este proyecto se centra en el desarrollo de una aplicación web robusta y funcional diseñada para optimizar la gestión de tareas y mejorar la dinámica de trabajo en equipos de desarrollo tecnológico. La herramienta ofrece funcionalidades que simplifican la visualización, asignación, seguimiento y resolución de tareas, fomentando la transparencia y la colaboración entre los miembros del equipo. Para lograr estos objetivos, se implementa un sistema de tickets que centraliza la gestión de releases, proyectos e incidencias, facilitando su organización y trazabilidad en todo momento.

Se ha implementado una arquitectura de tres capas (Presentación, Aplicación y Persistencia) para organizar las responsabilidades del sistema, facilitar su mantenimiento y optimizar su rendimiento. La capa de Presentación gestiona la interfaz de usuario con tecnologías como JavaScript, HTML y CSS; la de Aplicación coordina la lógica de negocio con tecnologías como Java, Spring Boot, Maven; y la de Persistencia asegura una comunicación eficiente con la base de datos MySQL mediante Hibernate y Spring Data JPA.

Se ha implementado un enfoque de pruebas exhaustivo que incluye pruebas unitarias y de sistema, utilizando herramientas como JUnit 5, Mockito y Selenium. Este enfoque garantiza que la lógica de negocio se valide adecuadamente y que las funcionalidades se comporten como se espera.

La metodología Git Flow se ha adoptado para gestionar el flujo de trabajo, permitiendo una organización eficiente del desarrollo mediante ramas específicas para cada etapa, como desarrollo, producción y características. También se han adoptado principios de DevOps con la automatización del proceso de integración y despliegue continuo mediante GitHub Actions. Este ejecuta pruebas automáticas, verifica la cobertura de código, gestiona el versionado del proyecto y facilita el despliegue en Microsoft Azure, a través de tecnologías como Docker y Azure Container Instances.

Finalmente, se realiza una reflexión sobre los resultados obtenidos en el apartado de conclusiones y se exponen algunas mejoras para posibles trabajos futuros.

Palabras clave:

- Java
- Spring Boot
- Maven
- MySQL
- Hibernate
- Docker
- Microsoft Azure
- Git
- DevOps

Índice de contenidos

Índice de figuras

Índice de códigos

1 Introducción

En este capítulo se expondrá el contexto y el alcance de la aplicación, su justificación, y una pequeña introducción de esta, además de la estructura general del documento.

1.1. Contexto y alcance

En la era digital actual, la tecnología ha transformado profundamente el modo en que las empresas operan y gestionan sus procesos internos. La eficiencia y la agilidad se han convertido en factores críticos para el éxito, especialmente en el ámbito empresarial donde los equipos de desarrollo tecnológico desempeñan un papel crucial. La gestión de tareas e incidencias es una de las áreas que más impacto tiene en la productividad y en la calidad del trabajo, ya que permite organizar, priorizar y solucionar problemas de manera efectiva.

La creciente complejidad de los proyectos tecnológicos, junto con la necesidad de una colaboración fluida entre diferentes miembros del equipo, ha generado una demanda significativa de herramientas que faciliten esta gestión. Las aplicaciones web han demostrado ser soluciones eficaces debido a su accesibilidad, escalabilidad y capacidad para integrar diversas funcionalidades en una plataforma unificada.

En este contexto, surge la necesidad de desarrollar una aplicación web específicamente diseñada para el control y la gestión de tareas e incidencias dentro de equipos de desarrollo tecnológico. Esta aplicación no solo debe permitir el seguimiento detallado de cada tarea e incidencia, sino también ofrecer herramientas de colaboración que mejoren la comunicación y la coordinación entre los miembros

de diferentes equipos.

1.1.1. Contexto empresarial. Estructuras Organizativas

Las empresas tecnológicas suelen tener estructuras organizativas complejas con múltiples equipos de desarrollo trabajando en paralelo. Estas estructuras a menudo incluyen:

1. Equipos Multifuncionales. Estos equipos están compuestos por desarrolladores, diseñadores, ingenieros de calidad (QA), gerentes de producto y otros roles clave. La colaboración entre diferentes disciplinas es esencial para el éxito de los proyectos.
2. Metodologías Ágiles. La mayoría de las empresas tecnológicas utilizan metodologías ágiles, como Scrum o Kanban, para gestionar sus proyectos. Estas metodologías permiten una mayor flexibilidad y adaptación rápida a los cambios, facilitando la entrega continua de valor.
3. Silos de Especialización. Dentro de una organización, pueden existir varios equipos especializados en diferentes áreas tecnológicas, como front-end, back-end, inteligencia artificial, seguridad, entre otros. Cada equipo se enfoca en un aspecto específico del desarrollo, pero la coordinación entre equipos es crucial para el éxito del proyecto global.
4. Herramientas de Colaboración. Para gestionar la complejidad y la distribución geográfica de los equipos, las empresas tecnológicas emplean una variedad de herramientas de colaboración y gestión de proyectos, como Jira, Confluence, Slack, y GitHub. Estas herramientas facilitan la comunicación, la gestión de tareas y el seguimiento de incidencias.
5. Cultura de DevOps. Muchas empresas han adoptado la cultura DevOps, que promueve la colaboración entre los equipos de desarrollo y operaciones para acelerar el ciclo de vida del desarrollo y la entrega continua de software de alta calidad.

La falta de herramientas adecuadas para la gestión de tareas e incidencias puede ocasionar múltiples problemas en los equipos de desarrollo tecnológico. Uno de los principales inconvenientes es el retraso en los plazos impuestos, ya que sin una gestión eficiente, las tareas pueden acumularse y desbordar el cronograma del proyecto.

Otras de las cuestiones más problemáticas que pueden surgir es la falta de visibilidad y trasparencia en el seguimiento de las actividades. Esta puede complicar la obtención de una visión clara del progreso, dificultando así la toma de decisiones informadas.

Por otra parte, la ausencia de una plataforma centralizada puede provocar una comunicación fragmentada, lo que afecta negativamente la coordinación entre los miembros del equipo.

1.1.2. Justificación del proyecto

La justificación de este proyecto se basa en varios aspectos clave que resaltan su relevancia y la necesidad de su implementación en el ámbito empresarial.

1. Mejora de la Productividad y Eficiencia.

   La gestión eficaz de tareas e incidencias permite a los equipos de desarrollo tecnológico optimizar su tiempo y recursos. Al disponer de una herramienta que centralice toda la información relevante, los equipos pueden priorizar y resolver problemas de manera más rápida y eficiente, reduciendo así el tiempo dedicado a actividades administrativas y aumentando el tiempo disponible para el desarrollo.

2. Facilitación de la Colaboración.

   En los equipos de desarrollo, la colaboración y la comunicación efectiva son esenciales. Una aplicación web que permita la asignación de tareas, el seguimiento del progreso y la actualización en tiempo real de incidencias fomenta un entorno de trabajo colaborativo. Esto es particularmente importante en proyectos de gran envergadura o en equipos distribuidos geográficamente, donde la coordinación puede ser un desafío.

3. Transparencia y Responsabilidad.

   Al centralizar la gestión de tareas e incidencias, se promueve una mayor transparencia dentro del equipo. Todos los miembros pueden ver quién es responsable de cada tarea, cuáles son los plazos y qué progresos se han realizado. Esto no solo mejora la rendición de cuentas, sino que también ayuda a identificar rápidamente cualquier problema o retraso que pueda surgir.

4. Adaptabilidad y Escalabilidad.

   La naturaleza de los proyectos tecnológicos puede variar significativamente en términos de tamaño y complejidad. Una aplicación web bien diseñada puede adaptarse a diferentes necesidades y escalar según sea necesario, proporcionando una solución flexible que puede crecer junto con la empresa.

5. Reducción de Errores y Retrabajos.

   Una gestión eficaz de tareas e incidencias contribuye a la reducción de errores y la necesidad de retrabajar tareas. Al tener un sistema que documenta

y sigue cada incidencia, se minimiza el riesgo de pasar por alto problemas y se asegura que cada tarea se complete correctamente desde el principio.

6. Toma de Decisiones Basada en Datos.

   La aplicación propuesta puede facilitar la obtención informes y análisis basados en los datos de tareas e incidencias registradas. Esto proporciona una base sólida para la toma de decisiones, permitiendo a los gerentes identificar tendencias, evaluar el rendimiento del equipo y tomar medidas proactivas para mejorar la eficiencia y la calidad del trabajo.

1.2. Estructura del documento

La estuctura que se sigue para la realización de la memoria es la siguiente:

1. Introducción.
2. Objetivos.
3. Tecnologías, Herramientas y Metodologías.
4. Descripción informática.
5. Conclusiones y trabajos futuros.
6. Bibliografía.
7. Apéndices.

2

Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación web robusta y funcional para la gestión de tareas e incidencias en equipos de desarrollo tecnológico. Los objetivos específicos podemos dividirlos en dos categorías: objetivos técnicos y objetivos funcionales.

2.1. Objetivos Funcionales

1. Facilitar la gestión de tareas e incidencias.

   Proveer herramientas que permitan asignar, seguir y resolver tareas e incidencias de manera eficiente, promoviendo la colaboración y la comunicación tanto interna como externa de los diferentes equipos.

2. Mejorar la eficiencia y productividad del equipo.

   Implementar funcionalidades que automaticen tareas rutinarias y proporcionen información en tiempo real, permitiendo a los desarrolladores centrarse en actividades de mayor valor.

3. Promover la transparencia y el control.

   Ofrecer una visibilidad clara del progreso del proyecto y la resolución de incidencias, facilitando la toma de decisiones informadas y la gestión proactiva.

2.2. Objetivos Técnicos

1. Garantizar la seguridad de la aplicación.

   Incorporar medidas de seguridad asegurando que la aplicación proteja los datos sensibles de los usuarios y cumpla con las prácticas de seguridad informática.

2. Fomentar la utilización de buenas prácticas.

   Diseñar la aplicación siguiendo principios y patrones de arquitectura que permitan su fácil escalabilidad y calidad del código, facilitando su mantenimiento. Esto incluirá el uso de patrones de diseño, así como la aplicación de principios de código limpio y sostenible.

3. Desarrollar una aplicación web intuitiva y accesible.

   Crear una plataforma que sea fácil de usar para los desarrolladores, con una interfaz amigable y funcionalidades claras y útiles.

4. Uso de Metodologías de Integración Continua.

   Asegurar que el desarrollo de la aplicación siga procedimientos de Integración Continua (CI), permitiendo un flujo constante de integración de cambios y reduciendo la posibilidad de errores durante el proceso de desarrollo.

3

Tecnologías, Herramientas y Metodologías

En este capítulo se listarán las tecnologías, herramientas y metodologías usadas en la aplicación.

3.1. Tecnologías

3.1.1. Java

Java [1] es el lenguaje de programación utilizado en el backend. Se ha seleccionado debido a mi amplia experiencia en su uso y a la abundancia de bibliotecas y recursos disponibles. El proyecto está definido con la versión 17 de Java.

3.1.2. Spring

Spring [2] es un framework modular que facilita la creación de aplicaciones web. Se ha elegido por su amplia adopción en la industria y la abundancia de documentación y recursos disponibles.

3.1.3. Spring Boot

Spring Boot [3] es una herramienta que permite desarrollar aplicaciones Spring de manera rápida y sencilla, facilitando la configuración y el despliegue de proyectos. Su enfoque en la simplicidad y la reducción de la cantidad de código de configuración necesario lo convierte en una opción altamente recomendable. El proyecto utiliza la versión 3.3.0 de Spring Boot.

3.1.4. Maven

Maven [4] es una herramienta de gestión de proyectos y construcción que se ha utilizado para gestionar las dependencias y la configuración del proyecto, facilitando la integración y el despliegue continuo.

3.1.5. MySQL

MySQL [5] es un sistema de gestión de bases de datos relacional que se ha utilizado en el proyecto. Es una base de datos robusta y ampliamente utilizada, lo que asegura un buen rendimiento y confiabilidad. Se ha usado el conector MySQL en la versión 8.0.19.

3.1.6. Hibernate

Hibernate [6] es un framework de mapeo objeto-relacional (ORM) muy popular en el ecosistema Java. Implementa la especificación JPA y proporciona características adicionales.

3.1.7. Apache Tomcat

Apache Tomcat [7] es el contenedor de servlets utilizado para desplegar la aplicación web. Su integración con Spring Boot y su popularidad lo hacen una elección natural para el desarrollo y despliegue de aplicaciones web Java.

3.1.8. Lombok

Lombok [8] es una biblioteca que simplifica la gestión de código repetitivo en Java. En lugar de escribir manualmente getters, setters y constructores, Lombok permite generar estos métodos automáticamente mediante el uso de anotaciones.

Esto no solo reduce la cantidad de código, sino que también mejora la legibilidad y el mantenimiento de las clases.

3.1.9. Spring Security

Spring Security [9] es un framework que gestiona, entre otras cosas, la autenticación y autorización en una aplicación Spring. Se ha configurado para la autenticación mediante un proveedor basado en usuario y contraseña, utilizando BCrypt para el cifrado de contraseñas. Además, en este proyecto se ha utilizado:

1. **Spring Security Taglibs** [10] es un módulo de Spring Security que proporciona un conjunto de etiquetas personalizadas diseñadas para facilitar la implementación de controles de seguridad en aplicaciones web basadas en JavaServer Pages (JSP). En este proyecto, se ha utilizado la versión 6.0.0.

3.1.10. Apache HttpClient y HttpCore5

Apache HttpClient [11] y HttpCore5 [12] son unas bibliotecas utilizadas para realizar y manejar solicitudes HTTP. Es útil para la comunicación entre servicios y para probar las APIs de la aplicación. En este proyecto se utilizan las versiones 5.4-alpha2 y 5.3-alpha2 respectivamente.

3.1.11. JUnit 5

JUnit 5 (Jupiter) [13] es el framework de pruebas utilizado para los tests unitarios del backend. Permite crear tests parametrizables de manera sencilla y ofrece métodos muy útiles para las pruebas unitarias.

3.1.12. Mockito

Mockito [14] es un framework que permite crear dobles en los test unitarios de manera sencilla. A la hora de crear test unitarios es imprescindible crear dobles de las dependencias externas para facilitar probar distintos casos y agilizar el test. Ha sido usado en los test unitarios del Backend ampliamente.

3.1.13. Selenium

Selenium [15] es un conjunto de herramientas y bibliotecas de código abierto utilizadas principalmente para la automatización de pruebas de aplicaciones web.

En este proyecto se ha utilizado Selenium WebDriver en la versión 4.21.0.

3.1.14. WebDriverManager

WebDriverManager [16] es una biblioteca que simplifica la gestión de controladores para pruebas de automatización de Selenium. En este proyecto, se utiliza para descargar y configurar automáticamente los controladores necesarios para los navegadores, lo que evita la necesidad de gestionar manualmente estas dependencias. Se ha utilizado la versión 5.8.0.

3.1.15. JaCoCo

JaCoCo [17] es una herramienta de análisis de cobertura de código para Java. Se utiliza para medir qué partes del código han sido ejecutadas durante las pruebas, proporcionando métricas valiosas sobre la efectividad de las pruebas unitarias y ayudando a identificar áreas que necesitan más pruebas. Se ha añadido como plugin en la versión 0.8.8.

3.1.16. Docker

Docker [18] es una plataforma que permite crear, desplegar y gestionar contenedores de software. Estos contenedores encapsulan aplicaciones y sus dependencias en un entorno aislado, lo que asegura que funcionen de manera consistente en diferentes entornos. Docker se ha convertido en un estándar en la industria debido a su capacidad para simplificar el despliegue y la escalabilidad de aplicaciones.

3.1.17. JavaScript

JavaScript [19] es un lenguaje interpretado que permite construir aplicaciones web interactivas y dinámicas. Su versatilidad y amplio soporte en todos los navegadores lo convierten en una herramienta esencial para el desarrollo moderno. En este proyecto se han utilizado varios frameworks y bibliotecas JavaScript, entre ellas:

1. **Bootstrap** [20] para proporcionar componentes interactivos y un diseño responsivo utilizando su JavaScript bundle. Se ha utilizado la versión 5.3.0.

2. **jQuery** [21] para simplificar el recorrido y manipulación del documento HTML, el manejo de eventos, la animación y las interacciones. Se ha utilizado la versión 3.7.1.

3. **DataTables** [22] para agregar funcionalidades avanzadas a las tablas HTML como paginación, filtrado y ordenamiento, con integración específica para Bootstrap. Se ha utilizado la versión 2.0.3.

4. **Chart.js** [23] para crear gráficos interactivos y flexibles en las páginas web. Se ha utilizado la versión 2.7.2.

5. **Flatpickr** [24] proporciona un selector de fecha y hora para aplicaciones web.

3.1.18. HTML

HTML [25] es un lenguaje de marcado que permite crear páginas web. Para la maquetación de las distintas páginas web es imprescindible usar HTML.

3.1.19. CSS

CSS [26] es un lenguaje de estilo utilizado para definir la presentación de documentos HTML. Se utiliza para estilizar y dar formato a las páginas web, mejorando la apariencia y la experiencia del usuario. En este proyecto, se ha utilizado:

1. **FontAwesome** [27] es una biblioteca que proporciona iconos vectoriales escalables para su uso en aplicaciones web, mejorando la estética y la usabilidad de la interfaz.

3.1.20. JSP

JSP [28] (JavaServer Pages) es una tecnología que permite la creación de páginas web dinámicas utilizando Java. Se utiliza para generar contenido HTML de forma dinámica en el servidor, integrando Java en páginas web. Además, se ha utilizado la siguiente biblioteca:

1. **JSTL** (JavaServer Pages Standard Tag Library) [29] proporciona un conjunto de etiquetas personalizadas que simplifican el desarrollo. Permite realizar operaciones como control de flujo, iteración, manipulación de variables y más, facilitando la creación de páginas web dinámicas en Java. En este proyecto se usan las versiones 3.0.1 y 3.0.0.

3.1.21. JSON

JSON [30] es un formato de representación de datos que se usa para la comunicación entre distintos sistemas. Es un estándar en la industria y permite la comunicación entre Frontend y Backend.

3.1.22. FasterXML

FasterXML [31] es la herramienta que usa Spring para parsear los objetos JSON utilizados en la aplicación. Se utiliza la biblioteca jackson-databind de FasterXML.

3.1.23. JAXB

La biblioteca JAXB [32] proporciona una API para la conversión entre objetos Java y su representación XML. Se añade la dependencia javax.xml.bind con la versión 2.3.1.

3.1.24. Spring Boot DevTools

Spring Boot DevTools [33] es una herramienta que mejora la experiencia de desarrollo en aplicaciones Spring Boot. Facilita el proceso de desarrollo al habilitar características como la recarga automática de aplicaciones, lo que permite ver los cambios en tiempo real sin necesidad de reiniciar el servidor.

3.2. Herramientas

3.2.1. Spring Tools Suite

Spring Tools Suite [34] es un entorno de desarrollo integrado (IDE) basado en Eclipse para el desarrollo de aplicaciones basadas en el framework Spring para Java. Proporciona herramientas poderosas para desarrollar, depurar y desplegar aplicaciones Spring de manera eficiente.

3.2.2. Visual Studio Code

Visual Studio Code [35] es un editor de código altamente versátil, compatible con una amplia gama de lenguajes de programación. Lo he requerido para poder

visualizar y editar lo respectivo al front-end, en especial los archivos JSP.

3.2.3. Git

Git [36] es una herramienta de control de versiones fundamental en el desarrollo de software. Es esencial utilizar un sistema de control de versiones para crear y mantener proyectos de software. Git es ampliamente utilizado en la actualidad y ofrece potentes capacidades de control de versiones.

3.2.4. GitHub

GitHub [37] es una plataforma colaborativa para alojar proyectos de software. He optado por utilizarla debido a su popularidad, su opción de espacio gratuito para proyectos iniciales y la integración con GitHub Actions para sistemas de integración continua (CI).

3.2.5. MySQL Workbench

MySQL Workbench [38] es una herramienta gráfica de diseño y administración de bases de datos MySQL. Permite modelar, visualizar, generar scripts SQL, realizar tareas de administración y mucho más, lo que facilita el trabajo con bases de datos MySQL.

3.2.6. Docker Hub

Docker Hub [39] es un servicio en línea que actúa como un registro de imágenes de Docker. Te permite almacenar, compartir y distribuir imágenes de contenedores.

3.2.7. Azure

Azure [40] es una plataforma de servicios en la nube desarrollada por Microsoft que proporciona una amplia gama de servicios para el desarrollo, implementación y gestión de aplicaciones y servicios a través de una red global de centros de datos. En este caso, se ha utilizado para el despliegue del proyecto y de la base de datos. Los servicios utlizados son:

1. **Azure Container Instances** [41] permite ejecutar aplicaciones en contenedores sin tener que gestionar servidores o configurar una infraestructura de contenedores compleja.

2. **Azure Database for MySQL - Flexible Server** [42] permite desplegar, gestionar y escalar bases de datos MySQL con una gran flexibilidad en la configuración y optimización.

3.3. Metodologías

El desarrollo de la página web ha seguido un enfoque que combina varias metodologías para optimizar la gestión del código, la integración de nuevas funcionalidades y el despliegue continuo de versiones en producción. Las metodologías empleadas se describen a continuación.

3.3.1. Git Flow

Se ha utilizado la metodología Git Flow [43] como base para la gestión de las ramas del proyecto, estableciendo un flujo de trabajo estructurado que facilita el desarrollo y despliegue. Esta metodología define ramas específicas para distintos propósitos: develop para el desarrollo, master para producción, feature para nuevas funcionalidades, y hotfix para correcciones urgentes.

Cada nueva funcionalidad o mejora se implementa en una rama feature, la cual se crea a partir de develop. Al completarse, se fusiona en develop, donde se ejecutan pruebas automáticas para asegurar la calidad y estabilidad del código. La rama develop funciona como un entorno de integración continua, en el que se consolidan y verifican todos los cambios en desarrollo.

Cuando una versión de develop está lista para producción, se realiza un merge a master, lo cual activa automáticamente el proceso de despliegue en producción mediante un flujo de CI/CD. Esto garantiza que solo el código probado y aprobado se despliegue en producción.

Además, para correcciones urgentes en producción, se crean ramas hotfix a partir de master. Una vez completadas y probadas, las ramas hotfix se fusionan tanto en master como en develop, asegurando que la corrección esté disponible en producción y se mantenga en la versión de desarrollo.

En la Figura 3.1 se muestra un esquema representativo del flujo de trabajo en Git siguiendo la metodología Git Flow.

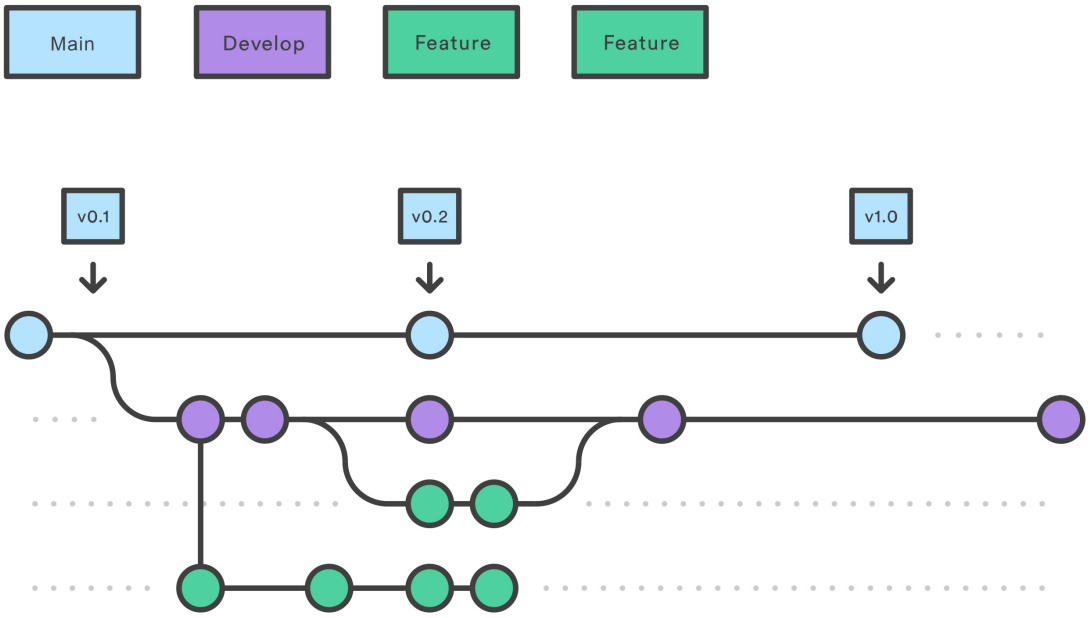

Figura 3.1: Ejemplo de flujo de trabajo de Git Flow

3.3.2. Metodologías Ágiles

Aunque no se ha implementado un marco ágil formal como Scrum [44] con ciclos de trabajo o "sprints" definidos, la creación de ramas feature para cada tarea específica sugiere un enfoque ágil implícito. Esto se debe a la propia naturaleza del proyecto, es decir, el desarrollo ha sido realizado y guiado de manera individual, sin la necesidad de coordinar un equipo, lo que ha hecho innecesario establecer un marco formal de trabajo colaborativo. Aun así, la entrega incremental de nuevas funcionalidades, así como la flexibilidad para adaptarse a cambios o nuevas necesidades, son características claves de las metodologías ágiles que se reflejan en el desarrollo del proyecto.

Este enfoque ágil ha permitido trabajar de manera continua en mejoras y nuevas funcionalidades, manteniendo un ritmo constante de entregas sin comprometer la calidad del desarrollo..

3.3.3. DevOps

El proyecto también adopta principios fundamentales de DevOps a través de la automatización de procesos clave (Figura 3.2). Se ha implementado un sistema de Integración Continua (CI) y Despliegue Continuo (CD) utilizando GitHub Actions [45].

Cada vez que se realizan cambios en una rama de características, estos se

suben primero a la rama develop, donde se ejecutan automáticamente las pruebas unitarias para garantizar que todo funcione correctamente. Posteriormente, se crea un Pull Request en el cual se verifica que los cambios sean adecuados y no afecten negativamente al código existente. Solo después de pasar esta revisión y asegurarse de que todo esté funcionando correctamente, se aprueba el Pull Request y los cambios se fusionan con la rama master, lo que desencadena el proceso de despliegue a producción.

El despliegue continuo permite que las nuevas versiones se lancen de manera automática, minimizando la intervención manual y garantizando que la aplicación esté siempre actualizada en el entorno de producción. Además, el uso de Docker para la generación de imágenes y el despliegue en Azure Container Instances asegura que las versiones sean consistentes y fácilmente replicables.

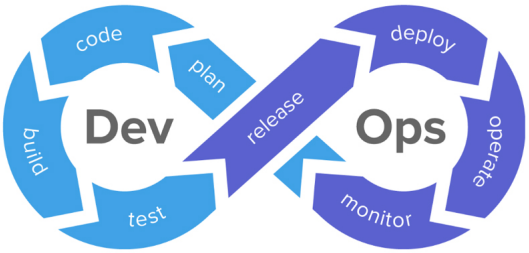

Figura 3.2: Ciclo de DevOps

4

Descripción informática

En este capítulo se proporciona una descripción exhaustiva del proyecto de software realizado, detallando cada aspecto crítico de su desarrollo desde los requisitos iniciales hasta la distribución final. El objetivo es ofrecer una visión completa y detallada de cómo se ha llevado a cabo el proyecto, abarcando tanto la planificación y el diseño como la implementación y las pruebas. A lo largo de este capítulo, se presentan los elementos clave y las decisiones técnicas que han guiado el desarrollo del software, proporcionando una base sólida para entender el producto final.

4.1. Requisitos

En esta sección, se detallan los requisitos funcionales y no funcionales de la aplicación, los cuales fueron definidos antes del inicio del desarrollo. Los requisitos se presentan como una lista detallada, proporcionando una comprensión clara de las funcionalidades que debe implementar la aplicación.

4.1.1. Requisitos Funcionales

Los requisitos funcionales especifican las funciones y características que el sistema debe tener para cumplir con las expectativas de los usuarios y las necesidades del negocio. Estos requisitos describen las acciones que los usuarios pueden realizar dentro del sistema y las capacidades que este debe ofrecer.

El sistema debe permitir realizar las siguientes acciones para asegurar un uso completo y eficaz de la aplicación:

Rol USER

- **Gestión de Usuarios:**
  - RF1: Los usuarios podrán iniciar sesión en el sistema.
  - RF2: Los usuarios podrán cerrar sesión en el sistema.
  - RF3: Los usuarios podrán visualizar su perfil.
  - RF4: Los usuarios podrán visualizar todas sus tareas individuales.
  - RF5: Los usuarios podrán visualizar estadísticas personalizadas que reflejan su rendimiento.
- **Gestión de Equipos:**
  - RF6: Los usuarios podrán visualizar todas las tareas asignadas a su equipo.
  - RF7: Los usuarios podrán visualizar gráficos y estadísticas de desempeño de su equipo.
  - RF8: Los usuarios podrán visualizar la carga de trabajo de cada miembro de su equipo.
- **Gestión de Incidencias:**
  - RF9: Los usuarios podrán crear incidencias.
  - RF10: Los usuarios podrán visualizar las incidencias.
  - RF11: Los usuarios podrán clonar las incidencias.
  - RF12: Los usuarios podrán editar incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF13: Los usuarios podrán cambiar el estado de las incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF14: Los usuarios podrán asignarse y desasignarse las incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF15: Los managers podrán asignar y desasignar a usuarios de su mismo equipo las incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF16: Los usuarios podrán traspasar incidencias a otros equipos (solo si pertenecen al equipo asignado a la incidencia).
  - RF17: Los usuarios podrán visualizar un seguimiento de los cambios realizados en cada incidencia.

- RF18: Los usuarios podrán filtrar las incidencias por diferentes campos.

- **Gestión de Proyectos:**
  - RF19: Los usuarios podrán visualizar los proyectos.
  - RF20: Los usuarios podrán editar los proyectos (solo si pertenecen al equipo asignado al proyecto).
  - RF21: Los usuarios podrán cambiar el estado de los proyectos (solo si pertenecen al equipo asignado al proyecto).
  - RF22: Los usuarios podrán asignarse y desasignarse a sí mismos a los proyectos (solo si pertenecen al equipo asignado al proyecto).
  - RF23: Los usuarios podrán filtrar los proyectos por diferentes campos.
- **Gestión de Releases:**
  - RF24: Los usuarios podrán visualizar las releases.
  - RF25: Los usuarios podrán filtrar las releases por diferentes campos.

Rol MANAGER

- **Gestión de Usuarios:**
  - RF26: Los managers podrán iniciar sesión en el sistema.
  - RF27: Los managers podrán cerrar sesión en el sistema.
  - RF28: Los managers podrán visualizar su perfil.
  - RF29: Los managers podrán visualizar todas sus tareas individuales.
  - RF30: Los managers podrán visualizar estadísticas personalizadas que reflejan su rendimiento.
  - RF31: Los managers podrán visualizar el perfil de todos los usuarios de su mismo equipo.
  - RF32: Los managers podrán editar el puesto de trabajo y el equipo perteneciente de todos los usuarios de su mismo equipo.
- **Gestión de Equipos:**
  - RF33: Los managers podrán visualizar el equipo al que pertenecen.
  - RF34: Los managers podrán editar el equipo al que pertenecen.
  - RF35: Los managers podrán visualizar todas las tareas asignadas a su equipo.
  - RF36: Los managers podrán visualizar gráficos y estadísticas de desempeño de su equipo.

- RF37: Los managers podrán visualizar la carga de trabajo de cada miembro de su equipo.

- **Gestión de Incidencias:**
  - RF38: Los managers podrán crear incidencias.
  - RF39: Los managers podrán visualizar las incidencias.
  - RF40: Los managers podrán clonar las incidencias.
  - RF41: Los managers podrán editar incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF42: Los managers podrán cambiar el estado de las incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF43: Los managers podrán asignarse y desasignarse las incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF44: Los managers podrán asignar y desasignar a usuarios de su mismo equipo las incidencias (solo si pertenecen al equipo asignado a la incidencia).
  - RF45: Los managers podrán traspasar incidencias a otros equipos (solo si pertenecen al equipo asignado a la incidencia).
  - RF46: Los managers podrán visualizar un seguimiento de los cambios realizados en cada incidencia.
  - RF47: Los managers podrán filtrar las incidencias por diferentes campos.
  - RF48: Los managers podrán eliminar las incidencias que estén asignadas a su equipo.

- **Gestión de Proyectos:**
  - RF49: Los managers podrán crear proyectos asignados a su equipo.
  - RF50: Los managers podrán visualizar los proyectos.
  - RF51: Los managers podrán clonar los proyectos asignados a su equipo.
  - RF52: Los managers podrán editar los proyectos (solo si pertenecen al equipo asignado al proyecto).
  - RF53: Los managers podrán cambiar el estado de los proyectos (solo si pertenecen al equipo asignado al proyecto).
  - RF54: Los managers podrán asignarse y desasignarse los proyectos (solo si pertenecen al equipo asignado al proyecto).
  - RF55: Los managers podrán asignar y desasignar a usuarios de su mismo equipo a los proyectos (solo si pertenecen al equipo asignado al proyecto).

  - RF56: Los managers podrán filtrar los proyectos por diferentes campos.

- **Gestión de Releases:**
  - RF57: Los managers podrán crear releases.
  - RF58: Los managers podrán editar las releases.
  - RF59: Los managers podrán visualizar las releases.
  - RF60: Los managers podrán añadir a las releases proyectos asignados a su equipo.
  - RF61: Los managers podrán filtrar las releases por diferentes campos.

Rol ADMIN

- **Gestión de Usuarios:**
  - RF62: Los administradores podrán iniciar sesión en el sistema.
  - RF63: Los administradores podrán cerrar sesión en el sistema.
  - RF64: Los administradores podrán crear nuevos usuarios.
  - RF65: Los administradores podrán eliminar usuarios.
  - RF66: Los administradores podrán editar todos los usuarios.
  - RF67: Los administradores podrán visualizar todos los usuarios.
  - RF68: Los administradores podrán visualizar todas sus tareas individuales.
  - RF69: Los administradores podrán visualizar estadísticas personalizadas que reflejan su rendimiento.
- **Gestión de Equipos:**
  - RF70: Los administradores podrán crear nuevos equipos.
  - RF71: Los administradores podrán eliminar todos los equipos.
  - RF72: Los administradores podrán editar todos los equipos.
  - RF73: Los administradores podrán visualizar todos los equipos.
  - RF74: Los administradores podrán visualizar todas las tareas asignadas a su equipo.
  - RF75: Los administradores podrán visualizar gráficos y estadísticas de desempeño de su equipo.
  - RF76: Los administradores podrán visualizar la carga de trabajo de cada miembro de su equipo.
- **Gestión de Incidencias:**

- RF77: Los administradores podrán crear incidencias.
- RF78: Los administradores podrán eliminar incidencias.
- RF79: Los administradores podrán visualizar las incidencias.
- RF80: Los administradores podrán clonar las incidencias.
- RF81: Los administradores podrán editar incidencias.
- RF82: Los administradores podrán cambiar el estado de las incidencias.
- RF83: Los administradores podrán asignar y desasignar a cualquier usuario perteneciente al equipo asignado a la incidencia.
- RF84: Los administradores podrán traspasar incidencias a otros equipos.
- RF85: Los administradores podrán visualizar un seguimiento de los cambios realizados en cada incidencia.
- RF86: Los administradores podrán filtrar las incidencias por diferentes campos.

- **Gestión de Proyectos:**
  - RF87: Los administradores podrán crear proyectos asignados a cualquier equipo.
  - RF88: Los administradores podrán eliminar proyectos asignados a cualquier equipo.
  - RF89: Los administradores podrán visualizar los proyectos.
  - RF90: Los administradores podrán clonar los proyectos asignados a cualquier equipo.
  - RF91: Los administradores podrán editar los proyectos.
  - RF92: Los administradores podrán cambiar el estado de los proyectos.
  - RF93: Los administradores podrán asignar y desasignar a cualquier usuario perteneciente al equipo asignado al proyecto.
  - RF94: Los administradores podrán filtrar los proyectos por diferentes campos.

- **Gestión de Releases:**
  - RF95: Los administradores podrán crear releases.
  - RF96: Los administradores podrán eliminar releases.
  - RF97: Los administradores podrán editar releases.
  - RF98: Los administradores podrán visualizar las releases.
  - RF99: Los administradores podrán añadir proyectos a las releases.
  - RF100: Los administradores podrán filtrar las releases por diferentes campos.

4.1.2. Requisitos No Funcionales

Los requisitos no funcionales describen los criterios que pueden ser utilizados para juzgar el funcionamiento de un sistema, en lugar de comportamientos específicos.

1. RNF1: El sistema debe ser intuitivo y fácil de utilizar.

2. RNF2: El sistema debe adaptarse a diferentes tamaños de pantalla de dispositivos de escritorio

3. RNF3: El sistema debe permitir la gestión de diferentes roles, asignando acciones específicas según el rol.

4. RNF4: El sistema debe garantizar el almacenamiento de datos en una base de datos relacional.

5. RNF5: El sistema incluirá pruebas exhaustivas para asegurar la integridad y el correcto funcionamiento del sistema.

6. RNF6: El sistema debe incluir herramientas para confirmar acciones importantes antes de su ejecución.

4.2. Arquitectura y Análisis

En este apartado se detallará la arquitectura y el análisis del sistema desarrollado para la gestión integral de usuarios, releases, proyectos e incidencias. Como se puede observar en la Figura 4.1, la arquitectura de la aplicación web ha sido diseñada siguiendo el modelo de n-capas, adoptando específicamente una estructura de 3 capas: Capa de Presentación (Frontend), Capa de Aplicación (Backend) y Capa de Persistencia (Base de Datos). Esta arquitectura permite una separación clara de responsabilidades, facilitando el mantenimiento, la escalabilidad y la robustez del sistema.

Además, para el despliegue de la aplicación se han utilizado tecnologías como Azure Container Instances, Azure Databases MySQL Server y Docker. A continuación se describe en mas detalle cada una de estas capas.

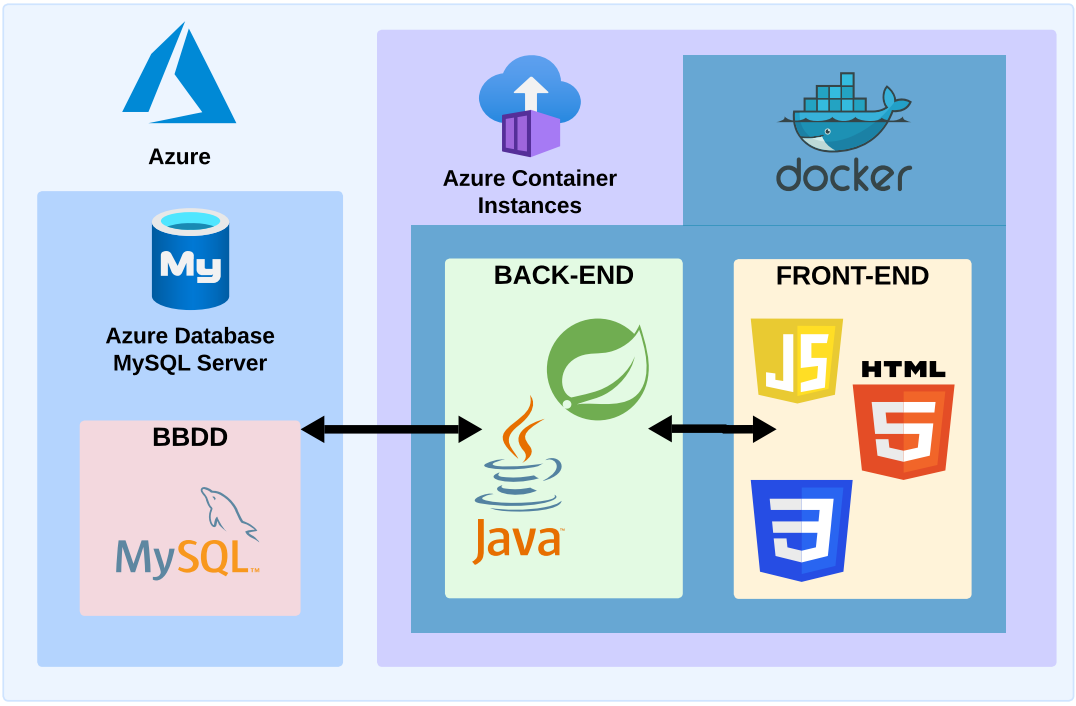

Figura 4.1: Arquitectura general de AlcalApp

4.2.1. Capa de Presentación (Frontend)

La Capa de Presentación es fundamental para la interacción del usuario con la aplicación web. Su principal función es manejar la entrada y salida de datos, proporcionando una interfaz intuitiva y visualmente atractiva que facilita a los usuarios interactuar con el sistema de manera efectiva.

Funciones Principales:

1. **Interacción del Usuario**:

   La Capa de Presentación es responsable de capturar las entradas del usuario a través de diversos elementos de la interfaz, como formularios, botones, menús, y otros componentes interactivos.

   Incluye paneles de control personalizados que muestran información relevante según el rol y las necesidades del usuario. Esto garantiza una experiencia de usuario optimizada y centrada en sus tareas y responsabilidades.

2. **Generación Dinámica de Contenido**:

   Utilizando JSP (JavaServer Pages), la capa de presentación puede generar contenido HTML de manera dinámica en el servidor. Esto permite que las

vistas se actualicen en tiempo real, reflejando el estado más reciente de la aplicación y los datos asociados.

La integración de JSTL (JavaServer Pages Standard Tag Library) facilita la creación de páginas web dinámicas, permitiendo operaciones comunes como iteraciones, manipulación de variables y control de flujo directamente en las páginas JSP.

3. **Visualización y Manipulación de Datos**:

   La Capa de Presentación maneja la visualización de datos complejos de manera simplificada. Utiliza componentes interactivos para mostrar información de manera clara y accesible, como tablas y gráficos. Estos componentes permiten funcionalidades avanzadas como paginación, filtrado y ordenamiento, mejorando la experiencia del usuario al interactuar con grandes volúmenes de datos.

   La integración con bibliotecas de gráficos permite la representación visual de datos, facilitando el análisis y la toma de decisiones por parte del usuario.

4. **Actualización en Tiempo Real**:

   A través de técnicas como Fetch (una API nativa de JavaScript para realizar solicitudes asíncronas), la Capa de Presentación puede actualizar partes de la interfaz de usuario sin necesidad de recargar la página completa. Esto mejora significativamente la eficiencia y la fluidez de la interacción del usuario con la aplicación.

5. **Diseño Responsivo**:

   El diseño responsivo asegura que la interfaz de usuario se adapte a diferentes tamaños y resoluciones de pantalla, proporcionando una experiencia de usuario consistente y accesible desde distintos dispositivos.

4.2.2. Capa de Aplicación (Backend)

La Capa de Aplicación, también conocida como backend, es la responsable de gestionar la lógica de negocio, procesar las solicitudes del usuario, interactuar con la base de datos, y asegurar la correcta operación del sistema. Esta capa actúa como intermediaria entre la interfaz de usuario y los datos almacenados, proporcionando funcionalidades esenciales para el funcionamiento de la aplicación.

Funciones Principales:

1. **Procesamiento de Solicitudes:**

   El backend recibe y procesa las solicitudes del frontend, aplica la lógica de negocio necesaria y responde con los datos apropiados. Para esto, se utiliza el patrón de diseño Spring MVC [46] (modelo, vista y controlador).

2. **Interacción con la Base de Datos:**

   Gestiona todas las operaciones relacionadas con la base de datos, incluyendo la creación, lectura, actualización y eliminación (CRUD) de registros. Se utilizan Hibernate y Spring Data JPA para el mapeo objeto-relacional (ORM).

3. **Gestión de Seguridad:**

   Incluye la implementación de medidas de seguridad como la autenticación de usuarios y la autorización de acceso a recursos específicos. Esto se logra utilizando Spring Security para la autenticación basada en roles.

4. **Implementación de la Lógica de Negocio:**

   Contiene las reglas y procedimientos que definen cómo la aplicación debe comportarse y cómo deben procesarse los datos. Esta lógica se implementa en Java (Figura 4.2) y se organiza y encapsula en servicios y controladores.

Figura 4.2: Lenguaje utilizado para la realización del Back-end

4.2.3. Capa de Persistencia (Base de Datos)

La Capa de Persistencia es fundamental para almacenar y recuperar datos de manera eficiente y segura, sirviendo como el núcleo donde se gestionan los datos críticos de la aplicación. Esta capa maneja diversas entidades y sus relaciones para asegurar la integridad y consistencia de los datos a lo largo del ciclo de vida de la aplicación.

Funciones Principales:

1. **Almacenamiento de Datos:**

   La principal función de la capa de persistencia es el almacenamiento de datos. Utiliza un sistema de gestión de bases de datos relacional como MySQL (Figura 4.3) para definir y gestionar tablas que contienen la información esencial de la aplicación, como usuarios, roles, releases, proyectos, incidencias y equipos.

2. **Recuperación de Datos:**

   Proporciona mecanismos eficientes para recuperar datos cuando son necesarios. Esto incluye la realización de consultas complejas que pueden involucrar múltiples tablas y relaciones, asegurando que la información relevante esté disponible para las operaciones de la capa de aplicación.

3. **Operaciones CRUD:**

   Maneja las operaciones básicas de creación, lectura, actualización y eliminación (CRUD) de registros. Estas operaciones son fundamentales para mantener la base de datos actualizada y reflejar los cambios en la aplicación en tiempo real.

4. **Mapeo Objeto-Relacional (ORM):**

   Utiliza Spring Data JPA para mapear entidades Java a tablas de la base de datos, eliminando la necesidad de escribir SQL manualmente. Spring Data JPA simplifica la gestión de las entidades y sus relaciones, proporcionando una abstracción que facilita la creación de consultas derivadas a partir de los nombres de los métodos de los repositorios. Además, permite la definición de consultas personalizadas mediante anotaciones y gestiona automáticamente las transacciones, optimizando las operaciones de base de datos.

Figura 4.3: Sistema de gestión de datos utilizado

4.2.4. Comunicación entre Capas

La comunicación entre las capas de una aplicación n-capas es esencial para garantizar que los datos fluyan correctamente desde la capa de presentación hasta la base de datos y viceversa. Cada capa interactúa con la siguiente a través de interfaces bien definidas y protocolos de comunicación. En la Figura 4.4 se muestra un esquema sobre cómo se comunican las capas de tu aplicación, que se explica a continuación:

1. **Comunicación entre la Capa de Presentación y la Capa de Aplicación**

La comunicación entre la capa de presentación (Frontend) y la capa de aplicación (Backend) se realiza principalmente a través de solicitudes HTTP. Estas solicitudes pueden ser de tipo GET, POST, PUT, DELETE, entre otras, dependiendo de la operación que se necesite realizar.

El frontend envía solicitudes HTTP a la API del backend. Las solicitudes pueden ser AJAX (Asynchronous JavaScript and XML) para operaciones asincrónicas, utilizando jQuery o el API Fetch de JavaScript.

Las respuestas HTTP del backend suelen estar en formato JSON, que es fácil de procesar en JavaScript.

2. **Comunicación entre la Capa de Aplicación y la Capa de Persistencia**

   La comunicación entre la capa de aplicación (Backend) y la capa de persistencia (Base de Datos) se realiza a través de conexiones TCP, utilizando un ORM (Object-Relational Mapping), en este caso, Hibernate, para gestionar las operaciones de base de datos.

   El backend establece una conexión TCP con el servidor de la base de datos MySQL para permitir el intercambio de datos. Esta conexión se configura en los archivos de configuración de Hibernate y se mantiene activa durante las operaciones de base de datos.

   Hibernate mapea las entidades Java a tablas de la base de datos y traduce las operaciones en objetos Java a consultas SQL que se ejecutan a través de la conexión TCP con la base de datos.

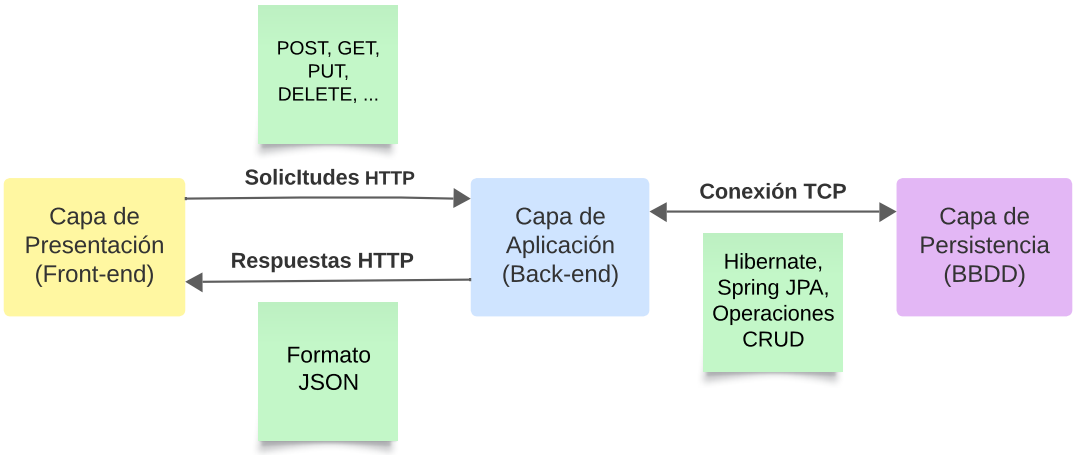

Figura 4.4: Esquema de comunicación entre capas

4.3. Diseño e Implementación

4.3.1. Back-End

Diseño

Para el diseño del proyecto se han seguido varios estándares de programación orientada a objetos que aseguran la mantenibilidad, extensibilidad y testabilidad del código. Entre otros, destacan los siguientes:

- **Modelo-Vista-Controlador (MVC) y Principio de Responsabilidad Única:**

  Este principio asegura que cada clase tiene una única tarea o responsabilidad. Este enfoque se observa claramente en la separación entre los controladores, los servicios y los repositorios, siguiendo el patrón arquitectónico Modelo-Vista-Controlador (MVC) como se muestra en la Figura 4.5. Los controladores gestionan las peticiones HTTP y actúan como intermediarios entre el modelo y la vista, los servicios encapsulan la lógica de negocio y los repositorios se encargan de interactuar con la base de datos.

- **Inversión de Control:**

  La Inversión de Control (IoC), implementada a través de la Inyección de Dependencias, complementa este diseño al externalizar la gestión de dependencias, permitiendo que las clases no se ocupen de instanciar sus propias dependencias. En su lugar, estas son inyectadas automáticamente por el contenedor de Spring. Esto no solo reduce el acoplamiento entre componentes, sino que también favorece un diseño modular donde cada componente del sistema puede desarrollarse, probarse y mantenerse de manera independiente. En la Figura 4.5 se puede observar esto con la inclusión de la anotación `@Autowired`.

```java
@Controller
@RequestMapping("/tickets")
public class TicketController {
    @Autowired
    private TicketService ticketService;
    @Autowired
    private AlcalappService alcalappService;

    @GetMapping("")
    public String ticketPage(Model model) {

        List<Ticket> tickets = ticketService.findAll();

        model.addAttribute("tickets", tickets);
        model.addAttribute("page", "INCIDENCIAS");
        return "tickets";
    }
```

```java
@Service
public class TicketServiceImpl implements TicketService {
    @Autowired
    private TicketRepository ticketRepository;

    public List<Ticket> findAll() {
        return ticketRepository.findAll();
    }
```

```java
public interface TicketRepository extends JpaRepository<Ticket, Long>{

    List<Ticket> findAll();
```

Figura 4.5: Ejemplo de Responsabilidad Única e Inversión de Control para el tratamiento de los tickets

- **Interfaz y Implementación:**

  Los servicios están divididos en interfaces (`EmployeeService`, `ProjectService`) y sus implementaciones (`EmployeeServiceImpl`, `ProjectServiceImpl`), lo cual es una aplicación del Principio de Segregación de Interfaces, como se muestra en la Figura 4.6. Este patrón también facilita el uso de testing y mocking, ya que puedes sustituir implementaciones en los tests.

```java
public interface MessageService {

    Message messageCreation(Timestamp date, String username, String teamName);
```

```java
@Service
public class MessageServiceImpl implements MessageService{

    @Autowired
    MessageRepository messageRepository;

    public Message messageCreation(Timestamp date, String username, String teamName) {
        Message message = new Message();
        LocalDateTime truncatedDateTime = date.toLocalDateTime().withSecond(0).withNano(0);
        Timestamp truncatedTimestamp = Timestamp.valueOf(truncatedDateTime);
        message.setDateRecord(truncatedTimestamp);
        message.setUserName(username);
        message.setText(Constants.CREATED_BY + teamName);
        return messageRepository.save(message);
    }
```

Figura 4.6: Ejemplo de Interfaz - Implementación para el tratamiento de los mensajes

- **Reutilización de Código:**

  La inclusión de clases utilitarias ha sido un paso importante para optimizar el código y mantener la coherencia en todo el sistema. Centralizar la lógica común en estas clases no solo ha permitido reducir la duplicación de código, sino que también ha facilitado el mantenimiento y la evolución del proyecto. Un ejemplo de esto se puede ver en la Figura 4.7, donde se muestra la implementación de la clase `Constants.java`, que contiene valores constantes utilizados en varias partes del sistema.

```java
public class Constants {
    //DATE FORMAT
    public static final String DATEFORMAT = "yyyy-MM-dd'T'HH:mm:ss'Z'";
    public static final String DATEFORMAT_YYYY_MM = "yyyy-MM";

    //LENGUAGES
    public static final String LANGUAGE_ES = "es";
    public static final String LANGUAGE_ES_UPPER = "ES";

    //NAMES
    public static final String NAME_PRJ = "PRJ ";
    public static final String NAME_R = "R";
    public static final String NAME_TCK = "TCK ";
```

Figura 4.7: Ejemplos de atributos de la clase Constants.java

Implementación

- **Controladores**

1. **AlcalAppController**

   El controlador `AlcalAppController` centraliza la lógica de autenticación, gestión de vistas principales y operaciones administrativas de la aplicación. Su objetivo es coordinar servicios para ofrecer funcionalidades integradas a los usuarios. Utiliza inyección de dependencias para acceder a los servicios de `EmployeeService`, `ProjectService`, `TeamService`, `TicketService`, `ReleaseService` y `AlcalappService`. Las principales funcionalidades del controlador incluyen:

   - **Gestión de autenticación:**
     a) *Inicio de sesión:* La vista de inicio de sesión (`loginPage`) con manejo de errores en credenciales.
     b) *Cierre de sesión:* Implementa el proceso de logout seguro con redirección a la página de inicio de sesión.
   - **Páginas principales:**
     a) *Trabajo diario:* La página `dailyWorkPage` muestra métricas y datos relacionados con el desempeño del equipo, incluyendo:
        - Tickets en progreso, bloqueados y completados.
        - Proyectos por equipo, listos para desplegar y completados.
        - Carga de trabajo y estadísticas mensuales de tickets y proyectos.
     b) *Trabajo personal:* La página `myWorkPage` presenta al usuario información sobre:
        - Sus tickets y proyectos en diferentes estados (en progreso, listos y completados).
        - Gráficos con estadísticas de resolución de tareas y creación de tickets.
   - **Gestión de usuarios y equipos:**
     a) *Gestión de usuarios:* Implementa funcionalidades para la gestión de usuarios como crear (`createUser`), editar (`updateUser`) y eliminar usuarios (`deleteUser`).
     b) *Gestión de equipos:* Implementa funcionalidades para la gestión de equipos como crear (`createTeam`), editar (`updateTeam`) y eliminar equipos (`deleteTeam`).
   - **Gestión de errores:** Proporciona una página genérica de error mediante el método `handleError`.

   Este controlador actúa como un punto central para integrar servicios y gestionar las vistas esenciales de la aplicación, brindando una experiencia cohesiva y eficiente al usuario.

2. **TicketController**

   El controlador `TicketController` permite la gestión integral de tickets. Este controlador tiene inyectados los servicios `TicketService` y

`AlcalappService`. Las principales funcionalidades de este controlador son la visualización, la creación, la edición y la eliminación de tickets. Además, se cuenta con las siguientes funciones:

- Reasignación y transferencia de tickets: Permite la transferencia de tickets entre diferentes equipos o empleados, y la actualización de su estado para reflejar el avance del trabajo en curso o el bloqueo.

3. **ProjectController**

   El controlador `ProjectController` gestiona la administración de proyectos en la aplicación. Se le inyectan los servicios `ProjectService` y `AlcalappService`. Las principales funcionalidades de este controlador son la visualización, la creación, la edición y la eliminación de proyectos. Además, se cuenta con las siguientes funciones:

   - Asignación de empleados y equipos: Permite reasignar empleados y equipos a proyectos específicos, mejorando la flexibilidad dentro de la aplicación.

4. **ReleaseController**

   El controlador `ReleaseController` organiza la planificación y despliegue de releases, gestionando tanto su creación como su seguimiento. Tiene inyectados los servicios `ReleaseService` y `AlcalappService`. Las principales funcionalidades de este controlador son la visualización, la creación, la edición y la eliminación de releases.

- **Servicios**

1. **AlcalappService:** Este servicio es fundamental para gestionar las operaciones del `AlcalappController`, pero su función es más amplia. No solo proporciona las funciones necesarias para este controlador, sino que también centraliza la lógica de negocio que requiere la colaboración de múltiples servicios y la interacción con varias tablas en la base de datos.

   El `AlcalappService` integra operaciones que son complejas y no pueden ser manejadas por un único servicio. Por ejemplo, puede manejar la creación de proyectos y su asignación a empleados, asegurando que todas las relaciones entre datos se gestionen de manera adecuada. Al inyectar todos los servicios relevantes, junto con `UserRepository` y `PasswordEncoder`, el `AlcalappService` actúa como un punto central para coordinar estas operaciones. Esto simplifica el código, ya que evita que los controladores necesiten interactuar directamente con varios servicios, haciendo que el sistema sea más eficiente y fácil de mantener. En resumen, el `AlcalappService` juega un papel clave en la orquestación de procesos que implican múltiples servicios y cruces de datos, mejorando la funcionalidad general de la aplicación.

2. **TicketService:** Este servicio está diseñado para manejar todas las interacciones asociadas a los tickets dentro de la aplicación. Proporciona las funciones requeridas por el `TicketController` para crear, modificar y eliminar tickets, así como gestionar su estado y asignaciones.
3. **ProjectService:** Este servicio se encarga de todas las operaciones relacionadas con la gestión de proyectos, proporcionando las funcionalidades necesarias para que el `ProjectController` realice tareas como la creación, actualización y eliminación de proyectos, así como la asignación de empleados y equipos.
4. **ReleaseService:** Este servicio implementa todas las funcionalidades del `ReleaseController`, gestionando el ciclo de vida de las releases en la aplicación. Facilita la creación, edición y eliminación de releases, asegurando una planificación adecuada de los despliegues.
5. **EmployeeService:** Este servicio se ocupa de gestionar la información de los empleados, permitiendo realizar operaciones CRUD sobre sus datos. Es esencial para vincular a los empleados con proyectos, tickets y equipos, garantizando que la información esté siempre actualizada y accesible.
6. **TeamService:** Encargado de gestionar los equipos dentro de la aplicación, este servicio se ocupa de la asignación de equipos a tickets y proyectos, así como de la administración de los empleados dentro de esos equipos, optimizando así la colaboración y el flujo de trabajo.
7. **MessageService:** Este servicio genera los mensajes asociados a la línea de tiempo de los tickets, facilitando el seguimiento de su estado y asegurando que los usuarios estén informados sobre las actualizaciones y cambios relevantes.
8. **UserDetailsServiceImpl:** Fundamental para la autenticación de usuarios, este servicio implementa la interfaz `UserDetailsService` de Spring Security. Su función principal es cargar los detalles del usuario a partir del nombre de usuario, verificando su existencia en el repositorio y asignando los roles y permisos necesarios para asegurar un control de acceso efectivo en la aplicación.

- **Repositorios y Entidades:**

  La interacción con la base de datos en la aplicación se realiza a través de repositorios que son responsables de manejar las operaciones CRUD para cada entidad. Por lo que en total, se cuenta con un repositorio por cada entidad: `EmployeeRepository`, `MessageRepository`, `ProjectRepository`, `ReleaseRepository`, `TeamRepository`, `TicketRepository` y `UserRepository`.

- **Modelos:**

  Los modelos desempeñan un papel crucial en la comunicación entre el back-end y el front-end de la aplicación, actuando como Data Transfer Objects

(DTO). Estas clases se encuentran en el paquete `web/model`, donde se definen estructuras que facilitan la transferencia de datos entre las diferentes capas de la aplicación. Están diseñadas para encapsular información relevante, permitiendo que el front-end acceda fácilmente a los datos necesarios para su visualización. Los objetos creados son los siguientes: `EmployeeDTO`, `EmployeePerTeam`, `ProjectTable`, `TablePerEmployee`, `TableTeam`, `TeamDTO`, `UserAndEmployeeDTO`, `WorkLoad` y `WorkPerEmployee`.

- **Otras clases:**

  En el paquete `configuration`, la clase `SecurityConfiguration` gestiona la seguridad de la aplicación. Esta clase define las políticas de acceso según roles, estableciendo quién puede acceder a cada endpoint. Además, configura el proceso de autenticación mediante `UserDetailsService` y asegura el cifrado de contraseñas con `PasswordEncoder`.

  La clase `Constants`, ubicada en el paquete `utils`, centraliza una serie de valores constantes y configuraciones predeterminadas que se utilizan en toda la aplicación. Esto incluye formatos de fecha, códigos de idioma, así como una variedad de estados y mensajes comunes.

4.3.2. Front-End

Diseño

El diseño del front-end del proyecto ha sido cuidadosamente estructurado siguiendo estándares y buenas prácticas para garantizar la mantenibilidad, extensibilidad y usabilidad del código:

- **Organización de Archivos:**

  La organización de archivos en el proyecto se ha llevado a cabo de manera lógica y coherente dentro de la carpeta `src/main/resources/static`, facilitando la gestión de los distintos recursos de la aplicación (Ver Figura 4.8).

  La carpeta `css` alberga hojas de estilo como `style.css` y `timeline.min.css`, que son responsables de la apariencia visual de la interfaz. En la carpeta `images` se almacenan iconos y gráficos, como `arrow-left.svg` y `arrow-right.svg`.

  Por último, la carpeta `js` contiene scripts JavaScript que manejan la lógica del front-end, como `dailywork.js`, `profile.js` y `project.js`, cada uno enfocado en funcionalidades específicas. Esto no solo mejora la legibilidad del código, sino que también facilita el mantenimiento y la actualización de los scripts según sea necesario.

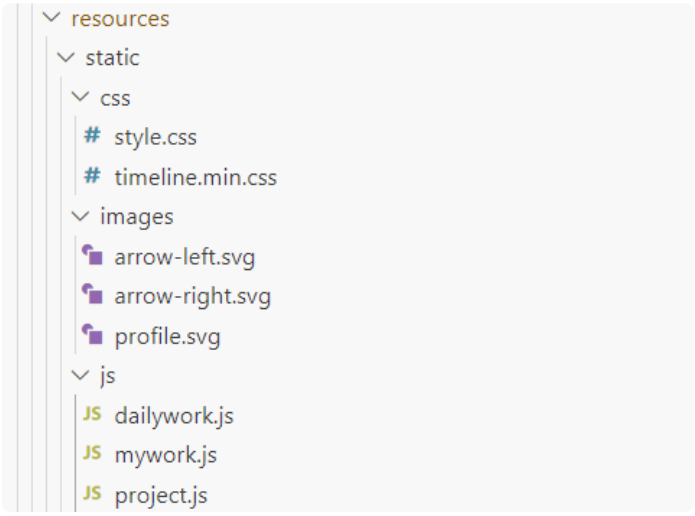

Figura 4.8: Organización de las carpetas de los estilos, imágenes y scripts

- **Estructura de Plantillas JSP:**

  La carpeta `WEB-INF/jsp` contiene los archivos JSP que actúan como plantillas para la presentación de las vistas de la aplicación (Ver Figura 4.9). Esta es la práctica más común y recomendada para este tipo de plantillas.

  Al colocar los archivos JSP dentro de `WEB-INF`, se asegura que no sean accesibles directamente a través de la URL, evitando que los usuarios accedan a ellos directamente en el navegador.

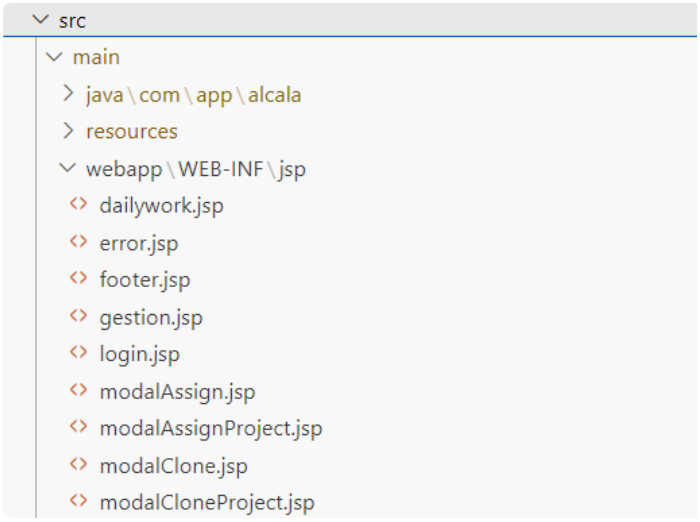

Figura 4.9: Organización de las archivos JSP

- **Componentes Reutilizables:**

Archivos como `navbar.jsp` y `footer.jsp` se utilizan en múltiples vistas, lo que garantiza una apariencia consistente en toda la aplicación. Esta estrategia no solo optimiza el diseño, sino que también facilita las actualizaciones, ya que cualquier cambio en estos componentes se refleja automáticamente en todas las páginas que los incluyen. En la Figura 4.10 se muestra como se incluyen estos componentes dentro de las plantillas JSP.

```
<body>
  <div class="container" id="dailywork">

    <!-- NavBar  -->
    <%@ include file="navbar.jsp" %>

    <main  class="d-flex">

      <!-- SideBar -->
      <%@ include file="sidebar.jsp" %>

      <section class="container-fluid">···
      </section>
      <%@ include file="modalNewUser.jsp" %>
      <%@ include file="modalCreate.jsp" %>
      <%@ include file="modalLogout.jsp" %>
      <%@ include file="modalCreateRelease.jsp" %>

    </main>
  </div>
</body>
```

Figura 4.10: Ejemplo de componentes reutilizables para el archivo dailywork.jsp

Implementación

El front-end de la aplicación está compuesto por un total de 38 plantillas JSP, cada una diseñada para atender diferentes funcionalidades y necesidades del usuario. Las vistas más destacadas son `dailywork.jsp` y `mywork.jsp`.

El archivo `dailywork.jsp` (Figura 4.11) es la página principal de la aplicación, donde los usuarios pueden gestionar sus actividades diarias. Esta vista proporciona una interfaz clara y accesible que permite a los usuarios visualizar sus tareas programadas, establecer prioridades y realizar un seguimiento del progreso. Además, cuenta con funcionalidades interactivas que facilitan la actualización de estados y la asignación de tareas en tiempo real.

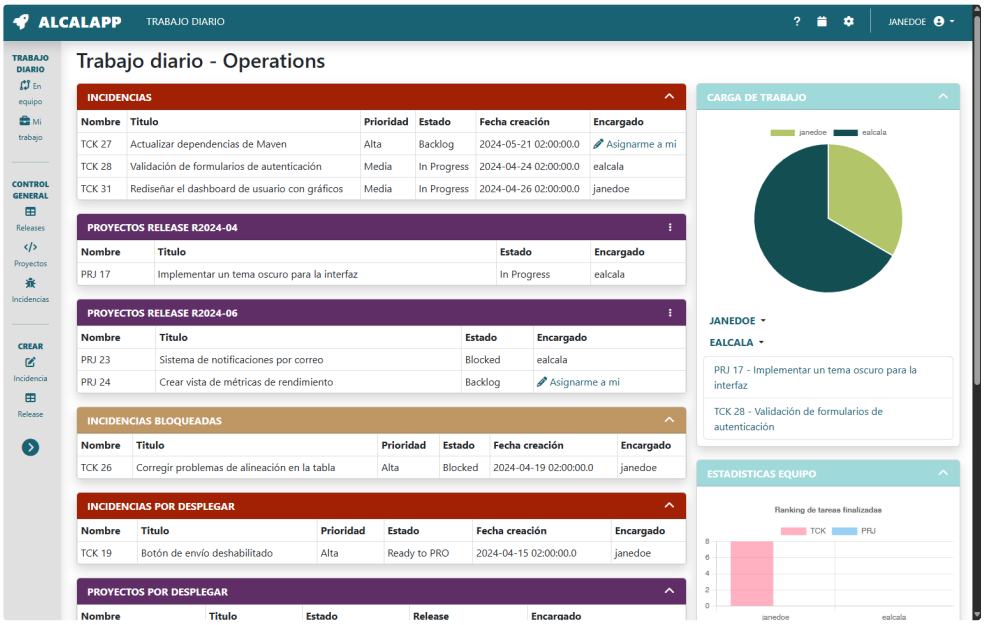

Figura 4.11: Vista de dailywork.jsp

La plantilla `mywork.jsp` (Figura 4.12) ofrece a los usuarios una visión más detallada de sus tareas y responsabilidades asignadas. Aquí, los usuarios pueden ver un resumen de las actividades pendientes, completadas y en progreso, lo que les permite organizar su trabajo de manera eficiente. Esta vista también incluye opciones de filtrado y búsqueda, ayudando a los usuarios a localizar rápidamente tareas específicas y a priorizar su carga de trabajo.

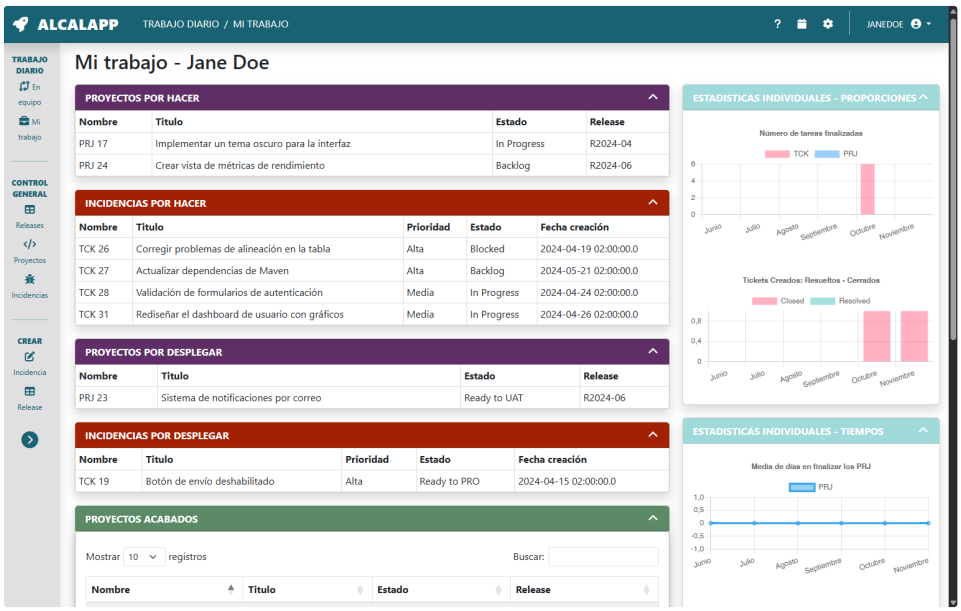

Figura 4.12: Vista de mywork.jsp

Las vistas `projects.jsp`, `releases.jsp` y `tickets.jsp` (Figura 4.13) están diseñadas para facilitar la visualización de todos los proyectos, releases e incidencias respectivamente. Estas vistas presentan una lista organizada de elementos, permitiendo a los usuarios consultar de manera rápida el estado, fechas y detalles de cada una de ellas. Además, integran opciones de filtrado que simplifican el manejo de grandes volúmenes de datos, mejorando así su seguimiento.

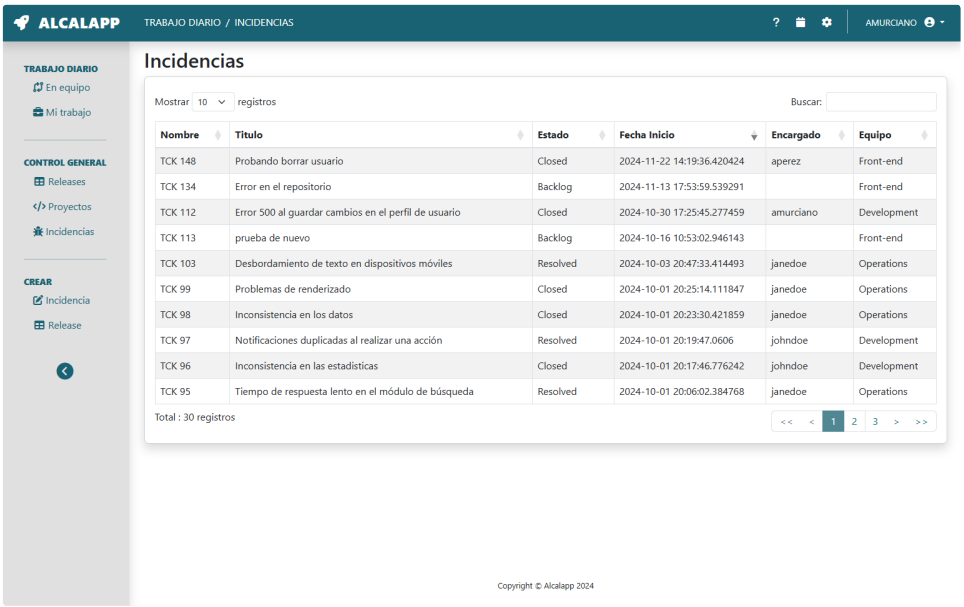

Figura 4.13: Vista de tickets.jsp

Asimismo, se incorporan las vistas`ticket.jsp` (Figura 4.14), `project.jsp` (Figura 4.15) y `release.jsp`, las cuales están enfocadas en la visualización de elementos individuales. Estas vistas muestran en detalle todas las características y atributos de un proyecto, release o ticket específico, permitiendo al usuario revisar información detallada y realizar acciones sobre elementos particulares de forma precisa y directa, como la edición o el borrado.

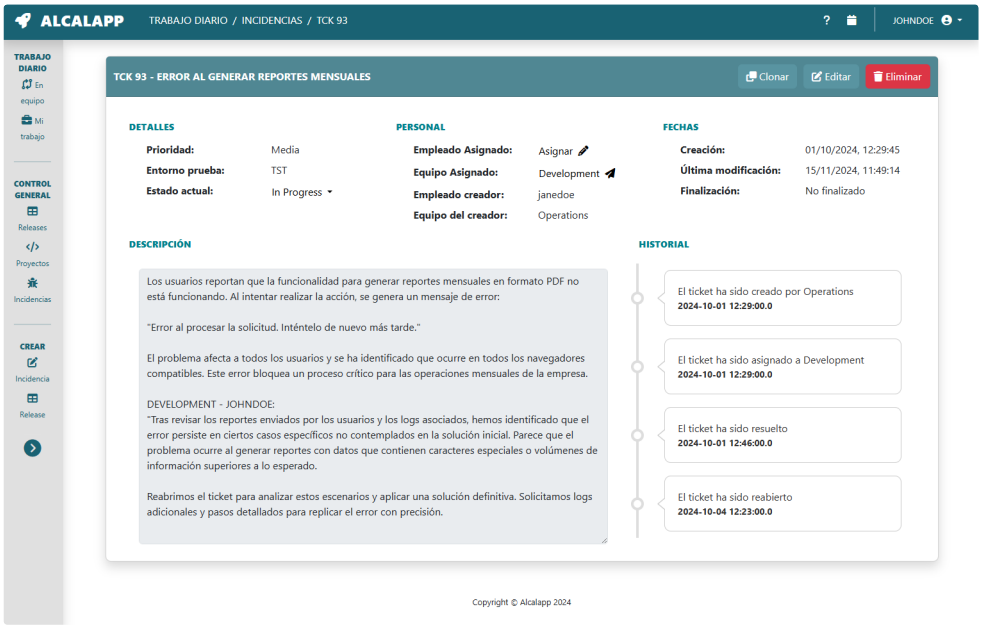

Figura 4.14: Vista de ticket.jsp

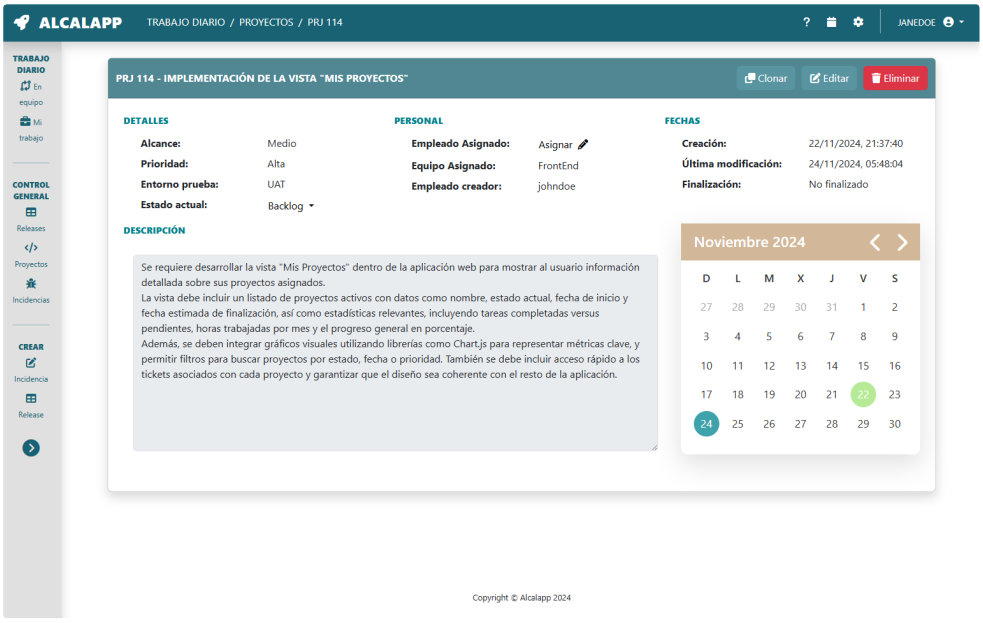

Figura 4.15: Vista de project.jsp

Las gestiones administrativas de usuarios y equipo se llevan a cabo en vistas como `gestion.jsp` (Figura 4.16), `user.jsp` y `team.jsp`. El acceso a estas vistas están limitadas según el rol del usuario.

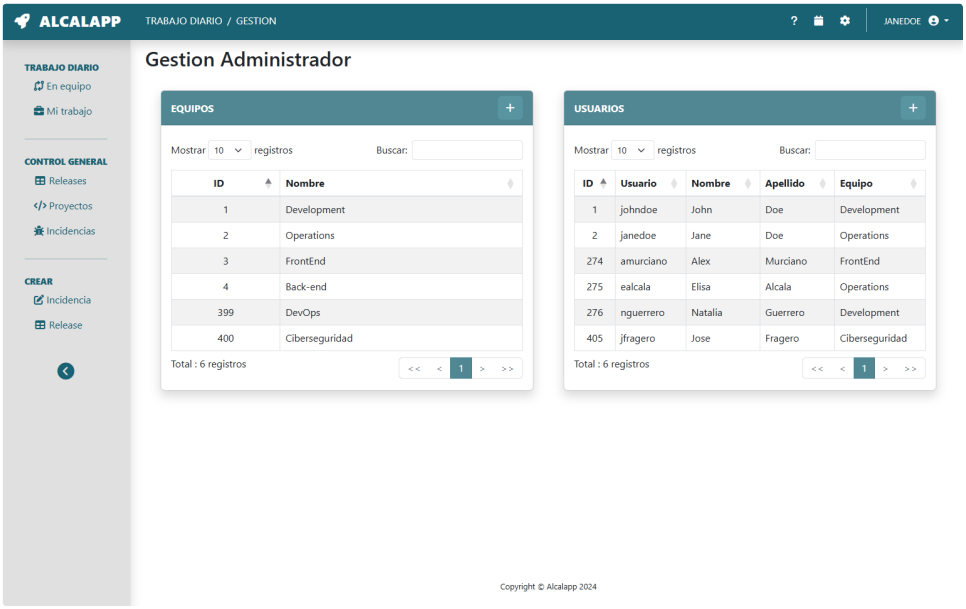

Figura 4.16: Vista de gestion.jsp

Para realizar operaciones más complejas, como la creación, eliminación o clonación de elementos, la aplicación integra numerosos modales que permiten llevar a cabo estas acciones de manera eficiente y fluida, como por ejemplo `modalCreate.jsp`, `modalClone.jsp`, `modalDelete.jsp`, ... En total se incluyen 21 modales para la gestión de la aplicación como el que se muestra en la Figura 4.17.

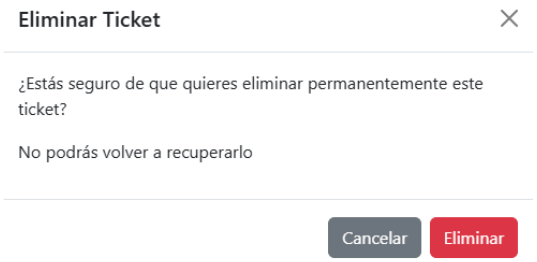

Figura 4.17: Vista de ModalDeleteTicket.jsp

Asimismo, la seguridad del acceso se garantiza a través de la vista `login.jsp`, que permite a los usuarios autenticarse, y el `modalLogout.jsp`, que facilita la desconexión del sistema.

Completando estas vistas, se incluye una página de error propia en el archivo `error.jsp`.

Cada plantilla JSP cuenta con un archivo JavaScript asociado que maneja la

lógica específica de esa vista. Los archivos principales incluyen `dailywork.js`, `mywork.js`, `team.js`, `project.js`, `release.js`, `ticket.js`, `user.js` y `timeline.js`. Cada uno de estos scripts está diseñado para interactuar directamente con los elementos de su plantilla correspondiente, proporcionando funcionalidades dinámicas y mejorando la interactividad de la interfaz de usuario.

Por otro lado, el archivo `scripts.js` contiene el código JavaScript que se emplea en todas las pantallas de la aplicación, como las acciones relacionadas con las barras lateral y superior. Esto centraliza y reutiliza funciones comunes, optimizando el código y facilitando el mantenimiento de la lógica compartida en toda la aplicación.

La mayoría de estilos están albergados en el archivo `style.css` que combinado con las librerías se encargan de darle un diseño coherente y atractivo a la interfaz de usuario.

Integrar las plantillas JSP con las diversas librerías y frameworks de frontend ha sido un desafío considerable. La combinación de librerías de componentes como Chart.js y DataTables, junto con frameworks como Bootstrap, ha requerido un ajuste cuidadoso para asegurar su compatibilidad y una correcta visualización en la interfaz. A menudo, surgieron problemas de compatibilidad entre versiones y conflictos en la carga de estilos, que podían llevar a resultados inesperados en el diseño y la disposición de los elementos en pantalla. Estos inconvenientes hicieron el proceso de desarrollo más complejo, ya que fue necesario probar y ajustar constantemente el código para evitar sobrescrituras de estilos no deseadas y asegurar una integración fluida entre las plantillas JSP y las librerías utilizadas. Este esfuerzo adicional fue clave para lograr una interfaz visualmente coherente y funcional, aunque implicó una gestión detallada y precisa de cada elemento del frontend.

Además de los desafíos de integración entre las plantillas JSP y las diversas librerías de frontend, otro de los mayores retos durante el desarrollo ha sido garantizar que la página web sea responsiva, permitiendo que la interfaz de usuario se adapte adecuadamente a diferentes tamaños de pantalla de escritorio.

Para lograr esta adaptabilidad, se implementó un enfoque basado en media queries dentro de las hojas de estilo CSS. Las media queries permiten aplicar estilos específicos según las características del dispositivo, como el ancho y la altura de la pantalla. Se definieron reglas específicas para ajustar el diseño y la disposición de los elementos en diferentes tamaños de pantallas, como se muestra en el Código 4.1, garantizando que toda la información se mantenga clara y accesible.

Esta estrategia ha sido fundamental para asegurar una experiencia de usuario consistente y funcional en dispositivos de diferentes resoluciones, aunque implicó un esfuerzo adicional para gestionar correctamente las interacciones entre los

estilos personalizados y las librerías de componentes y frameworks utilizados.

Código 4.1: Ejemplo de media queries del archivo style.css

```
.custom-card {
  width: 100%;
  max-width: 1300px;
  margin: 20px auto;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.25);
  border-radius: 8px;
  padding: 0;
}
@media (min-width: 1024px) {
  .custom-card {
    margin: 25px auto;
  }
}
@media (min-width: 1600px) {
  .custom-card {
    max-width: 1590px;
  }
}
```

4.3.3. Base de Datos

Diseño

El diseño de la base de datos sigue un enfoque relacional, y su programación está basada en la utilización de entidades a través del framework JPA (Java Persistence API) junto con Hibernate, lo que permite mapear las tablas de la base de datos a clases Java. Esto asegura una persistencia de datos eficiente y facilita la interacción con la base de datos, eliminando la necesidad de escribir consultas SQL manuales para las operaciones CRUD (Crear, Leer, Actualizar y Eliminar). A continuación, se detallan algunos aspectos clave de la implementación de la base de datos utilizando entidades:

- **Mapeo de Entidades:** Cada entidad representa una tabla en la base de datos, y sus atributos corresponden a las columnas de dicha tabla. Por ejemplo, la entidad `Employee`, que representa la tabla `EMPLOYEE`, tiene atributos como `employeeId`, `employeeName` y `employeeDni`, que son mapeados a las columnas correspondientes en la base de datos a través de anotaciones como `@Column` y `@Id`. En la Figura 4.18 se puede observar como se mapea la entidad con su tabla correspondiente.

- **Relaciones entre Entidades:** El sistema implementa varios tipos de relaciones entre entidades que reflejan las relaciones entre las tablas de la

base de datos. Por ejemplo, en la clase `Employee`, un empleado pertenece a un equipo (`Team`) mediante una relación `@ManyToOne`, y un empleado puede estar asignado a varios proyectos y tickets a través de relaciones `@OneToMany` utilizando mapas de proyectos y tickets, representados por `Map<Long, Project>` y `Map<Long, Ticket>`.

- **Automatización de Claves Primarias:** El atributo `employeeId` en la entidad `Employee` está marcado con la anotación `@Id` y `@GeneratedValue(strategy = GenerationType.IDENTITY)`, lo que indica que su valor se genera automáticamente por la base de datos. Esto asegura que cada registro tenga una clave primaria única, sin necesidad de gestionar manualmente los valores de las claves.

- **Gestión de Mapeos:** La entidad `Employee` contiene atributos como `birthDate` y `hireDate` que son mapeados a columnas de tipo `TIMESTAMP` en la base de datos SQL. Estas fechas se manejan en formato `java.sql.Timestamp`, asegurando una correcta compatibilidad con los sistemas de gestión de bases de datos relacionales.

  Además, para almacenar grandes cantidades de datos en ciertos campos, como descripciones detalladas, se utiliza la anotación `@Lob` (Large Object). Por ejemplo, en la entidad `Project`, el atributo `descriptionProject` está anotado con `@Lob` y mapeado a una columna de tipo `LONGTEXT`, lo que permite almacenar descripciones extensas de los proyectos sin las limitaciones de los tipos de datos tradicionales. Esto asegura que la base de datos pueda manejar grandes volúmenes de texto o datos binarios de forma eficiente.

```java
@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "RECORD")
public class Message implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_record")
    private Long idRecord;
    @Column(name = "user_name")
    private String userName;
    @Column(name = "text")
    private String text;
    @Column(name = "date_record")
    private Timestamp dateRecord;

}
```

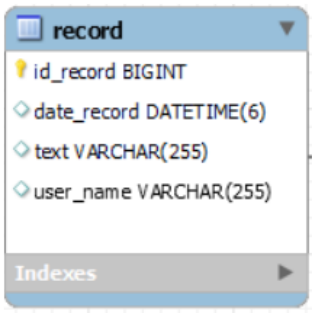

Figura 4.18: Mapeo de la entidad Message a la tabla Record

Implementación

La aplicación organiza sus elementos clave mediante clases interrelacionadas que reflejan la estructura y asignación de tareas en el sistema. La Figura 4.19 muestra un diagrama de entidad-relación que representa las relaciones entre algunas de las clases de AlcalApp que se describen a continuación.

La clase `Employee` representa a los empleados de la organización y está conectada con entidades fundamentales como `Team`, `Project` y `Ticket`, reflejando su pertenencia a un equipo y su participación en proyectos y tareas específicas. La clase `Team` agrupa a los empleados bajo un líder y se relaciona con las clases `Project` y `Ticket` para asignar tareas y proyectos a nivel de equipo.

Cada `Project` y `Ticket` se asigna a un equipo concreto y, dentro de este, a un empleado del equipo encargado de llevar a cabo la tarea. La clase `Ticket` representa las incidencias o tareas individuales dentro del desarrollo de proyectos. Además, cada `Ticket` incluye un sistema de `Messages` que sirve para registrar eventos clave como su creación, asignación, transferencia a otro equipo o empleado, y otros cambios de estado. Este registro facilita el seguimiento histórico de cada incidencia, permitiendo que los equipos consulten detalles sobre la asignación y evolución de las tareas de manera estructurada.

La clase `Release` organiza el ciclo de vida de los lanzamientos en el sistema, conectándose con `Project` para agrupar proyectos bajo una misma versión. `Release` también almacena al usuario que creó el lanzamiento, proporcionando una referencia clara de quién lo ha iniciado dentro del sistema.

Esta estructura permite asignar de forma clara responsabilidades, desde el nivel de equipo hasta el empleado individual encargado de cada tarea o proyecto.

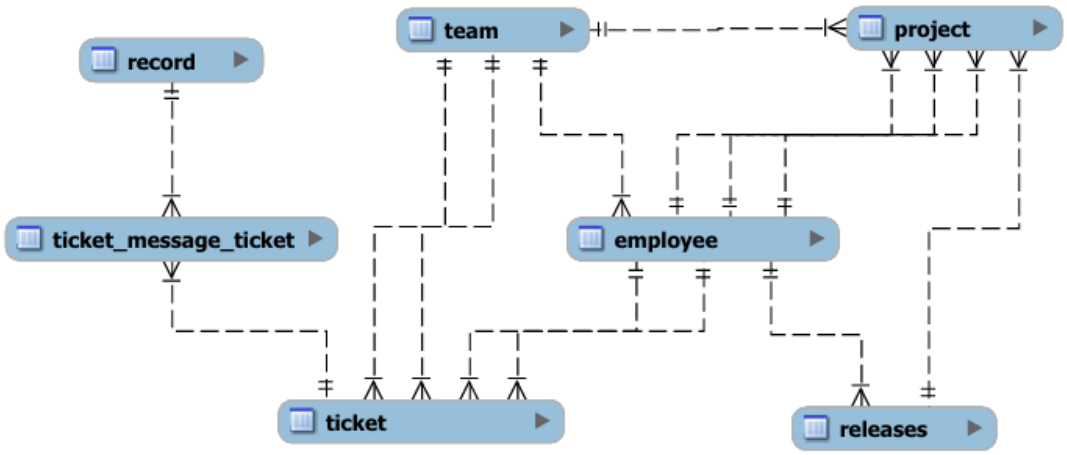

Figura 4.19: Diagrama de Entidad - Relación

Por otra parte, la clase `User` se ha diseñado exclusivamente para gestionar la autenticación y el inicio de sesión en el sistema. En la Figura 4.20 se muestra el diagrama de entidad-relación que se genera de dicha clase.

Esta entidad encapsula la información necesaria para la autenticación de cada usuario, incluyendo un identificador único (`id`), un nombre de usuario (`name`) y una contraseña cifrada (`encodedPassword`). Adicionalmente, la clase incorpora una lista de roles (`roles`), lo que permite asignar distintos niveles de permisos y accesos de manera sencilla y adaptable, alineándose con los principios de seguridad y gestión de accesos en el sistema.

Aunque `User` y `Employee` comparten el mismo nombre de usuario, lo que permite identificar a cada empleado mediante su nombre de inicio de sesión, ambos cumplen propósitos separados. `User` se orienta exclusivamente a la autenticación, mientras que `Employee` representa los datos y las responsabilidades del personal dentro de la organización, como la asignación de tareas y la pertenencia a equipos. En este caso, se evita la sobrecarga de la clase `User` con datos adicionales, facilitando una implementación más ligera y modular en la gestión de usuarios.

Este diseño simplifica la autenticación al mantener la clase `User` aislada de los detalles operativos de `Employee`. Además, asegura que cualquier cambio en la estructura de datos de los empleados no afecte la seguridad o la lógica de inicio de sesión, logrando una separación de responsabilidades adecuada y mejorando la flexibilidad del sistema.

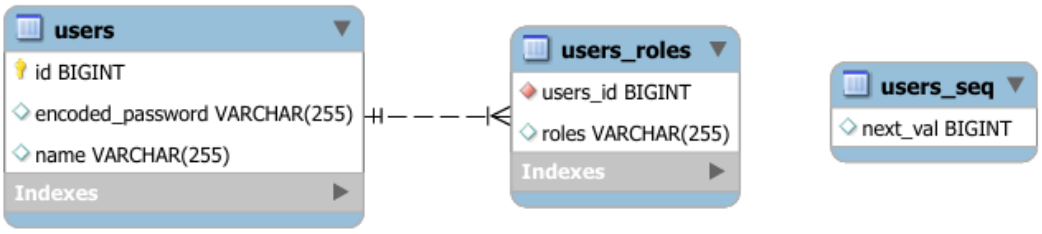

Figura 4.20: Diagrama de Entidad - Relación de la clase User.java

Uno de los mayores desafíos durante el desarollo de la base de datos, fue la optimización del rendimiento al cargar entidades y sus relaciones. Un aspecto clave fue la correcta elección entre las estrategias de carga de datos que Hibernate proporciona: `FetchType.EAGER` y `FetchType.LAZY`. Este punto se convirtió en una problemática significativa, ya que un mal uso de estas estrategias puede afectar negativamente el rendimiento de la aplicación.

Al principio, se empleó la estrategia `EAGER` para cargar las relaciones entre entidades, como en el caso de un empleado (`Employee`) y los proyectos (`Project`) y tickets (`Ticket`) asociados a él. La idea era que, al recuperar un empleado de la base de datos, también se cargaran de inmediato todos los proyectos y tickets

relacionados. Aunque en teoría parecía una solución eficiente para evitar consultas adicionales, rápidamente surgieron problemas de rendimiento y sobrecarga de memoria.

Para mitigar este problema, se optó por cambiar la estrategia a `FetchType.LAZY`, que permite cargar las relaciones de manera diferida. Con esta estrategia, los datos relacionados (como los proyectos y tickets de un empleado) solo se cargan cuando realmente son necesarios. Esto resultó en una mejora notable en el rendimiento, ya que evitaba la carga innecesaria de grandes volúmenes de datos hasta que se requirieran explícitamente. Sin embargo, se tuvo que encontrar un equilibrio entre ambas estrategias, ya que optar exclusivamente por `LAZY` generaba numerosos problemas en la carga del contexto de los tests, como excepciones de tipo `LazyInitializationException`.

Además de los problemas de rendimiento, surgió otra complicación relacionada con la serialización de las entidades en las respuestas JSON. Dado que algunas relaciones entre entidades son bidireccionales (por ejemplo, un `Employee` tiene proyectos y cada `Project` tiene un `Employee` asignado), esto puede generar bucles infinitos durante la serialización de los datos al formato JSON, lo que lleva a errores en la API. Para resolver esta problemática, fue necesario aplicar la anotación `@JsonIgnore` en aquellas relaciones que no debían ser serializadas en las respuestas JSON, evitando así que la información circular causara problemas.

4.4. Pruebas

En el desarrollo de software, las pruebas son esenciales para garantizar la calidad, la estabilidad y el correcto funcionamiento del sistema. En este proyecto se implementan dos tipos principales de pruebas: pruebas unitarias y pruebas de sistema. La Figura 4.21 muestra todas las pruebas de AlcalApp que se describen a continuación.

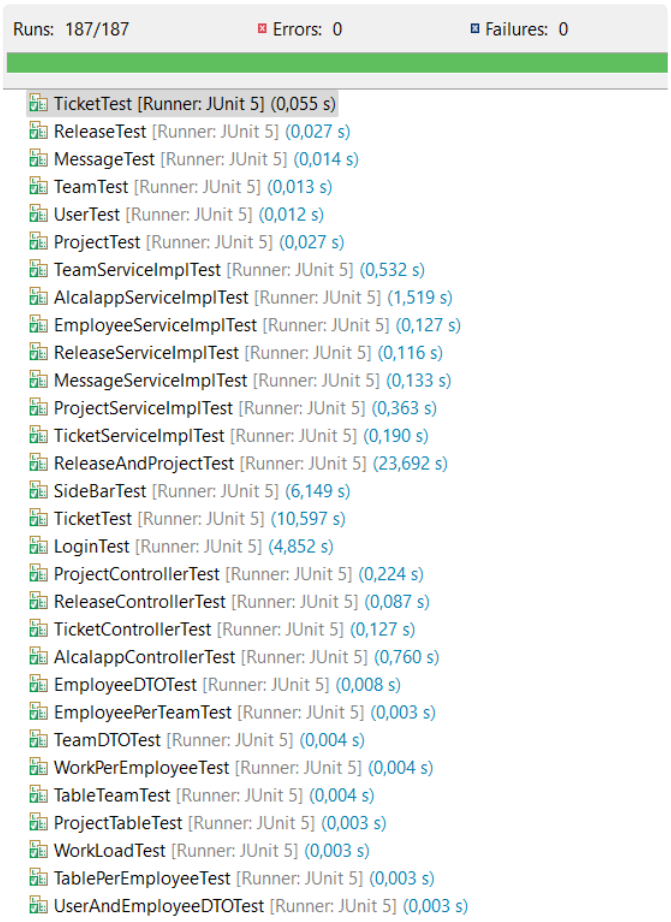

Figura 4.21: Pruebas unitarias y de sistema de AlcalApp

4.4.1. Pruebas Unitarias

Las pruebas unitarias en este proyecto tienen como objetivo validar funciones, métodos o clases individuales de manera aislada para asegurar que su comportamiento sea el esperado. Estas pruebas son cruciales para verificar la lógica interna de los servicios y controladores de la aplicación. Para completar la cobertura de la aplicación se incluyen además pruebas unitarias de las entidades y los modales.

Para la ejecución de pruebas unitarias en Java, se utiliza JUnit 5, un framework robusto y flexible que facilita la creación y ejecución de pruebas. Mockito se emplea para simular dependencias externas, lo que permite inyectar estas simulaciones en las pruebas y validar la lógica del código sin interferencias externas. Spring Test ofrece herramientas para integrar el contexto de Spring en las pruebas unitarias, asegurando que los componentes que dependen del framework funcionen correctamente.

En el proyecto, se realiza una amplia cobertura con 183 pruebas unitarias que

abordan casi el 100 % de la lógica de negocio, tanto de los servicios como de los controladores. Los ejemplos incluyen:

- **Pruebas de Controladores:** Para verificar el correcto funcionamiento de los endpoints, se utiliza MockMvc, una herramienta que permite realizar peticiones HTTP simuladas y validar tanto las respuestas como el comportamiento esperado del controlador. Por ejemplo, en el test `testGestion` del archivo `AlcalappControllerTest.java` (Código 4.2), se comprueba el acceso al endpoint `/gestion` y su integración con los servicios relacionados.

  En este test, se crea un entorno simulado que incluye objetos como `Employee` y `Team`, que son necesarios para la sesión. Además, se utilizan mocks de servicios como `alcalappService` y `teamService`, configurados con `Mockito` mediante el método `when(...).thenReturn(...)` para emular el comportamiento esperado de las dependencias externas.

  Código 4.2: Test Unitario testGestion de AlcalappControllerTest.java

```
@Test
@WithMockUser(username = "admin", roles = "ADMIN")
public void testGestion() throws Exception {
    List<Team> teams = Arrays.asList(new Team());
    List<Employee> employees = Arrays.asList(new Employee()
        );

    when(teamService.findAll()).thenReturn(teams);
    when(employeeService.findAll()).thenReturn(employees);

    mockMvc.perform(get("/gestion"))
        .andExpect(status().isOk())
        .andExpect(view().name("gestion"))
        .andExpect(model().attribute("teams", teams))
        .andExpect(model().attribute("employees", employees
            ))
        .andExpect(model().attribute("page", "GESTION"));
}
```

- **Pruebas de Servicios:** Las pruebas de servicios se centran en validar la lógica de negocio implementada en clases como `AlcalappServiceImpl`. Para ello, se utiliza Mockito, que permite simular el comportamiento de dependencias externas y así verificar el funcionamiento interno de los métodos.

  Por ejemplo, en el test `testAssignTicket` de la clase `AlcalappServiceImplTest` (Código 4.3), se evalúa el método `assignTicket`, que se encarga de asignar un ticket a un empleado.

Código 4.3: Test Unitario testAssignTicket de AlcalappServiceImplTest.java

```
@Test
void testAssignTicket() {
```

```
3       Ticket ticket = new Ticket();
4       ticket.setIdTicket(1L);
5
6       Employee employee = new Employee();
7       employee.setUserEmployee("User 1");
8       employee.setTicketMapEmployee(new HashMap<>());
9
10      when(ticketService.findById(anyLong()))
11          .thenReturn(ticket);
12      when(employeeService.findByUserEmployee(anyString()))
13          .thenReturn(employee);
14      when(ticketService.save(any(Ticket.class)))
15          .thenReturn(ticket);
16
17      Ticket result = alcalappService.assignTicket(1L, "\"
            User 1\"");
18
19      assertNotNull(result);
20      assertEquals("User 1", result.getEmployeeUserAssign());
21      assertEquals(employee, result.getEmployeeAssign());
22      assertTrue(employee.getTicketMapEmployee().containsKey
            (1L));
23      assertEquals(ticket, employee.getTicketMapEmployee().
            get(1L));
24  }
```

- **Pruebas de Entidades y Modales:** En estas pruebas, se comprueba la funcionalidad de los métodos generados por `@Data`, como `getters`, `setters`, `equals`, `hashCode`, y `toString`. También se verifican los constructores generados por `@AllArgsConstructor` y `@NoArgsConstructor`, asegurando que las instancias de las entidades pueden crearse con o sin argumentos, según se necesite. En el Código 4.4 se muestra un ejemplo de uno de estos métodos.

Código 4.4: Test Unitario testGettersSetters de UserTest.java

```
1  @Test
2  public void testGettersSetters() {
3      User newUser = new User();
4
5      newUser.setName("user");
6      newUser.setEncodedPassword("newEncodedPassword");
7      newUser.setRoles(List.of("ROLE_USER"));
8
9      assertThat(newUser.getName()).isEqualTo("user");
10     assertThat(newUser.getEncodedPassword()).isEqualTo("
           newEncodedPassword");
11     assertThat(newUser.getRoles()).containsExactly("ROLE_USER")
           ;
12 }
```

4.4.2. Pruebas de Sistema (E2E)

En este proyecto, se utilizan pruebas de sistema para verificar que las distintas partes de la aplicación funcionen conjuntamente como un todo cohesivo. Estas pruebas son esenciales para garantizar que la integración entre componentes y servicios no introduzca errores y que el sistema en su conjunto cumpla con los requisitos funcionales esperados.

Selenium WebDriver, junto con JUnit 5, se emplea para realizar pruebas de interfaz de usuario. Esta combinación permite interactuar con el navegador y simular acciones de usuario para verificar el comportamiento de la aplicación desde la perspectiva del usuario final. Mientras que JUnit 5 coordina la ejecución de pruebas y la gestión de escenarios de prueba, Selenium se encarga de automatizar las interacciones con la UI, lo que permite detectar problemas de integración y funcionalidad que no serían evidentes con pruebas unitarias aisladas.

La configuración inicial de las pruebas automatizadas con Selenium es esencial para garantizar su correcta ejecución. En este proyecto, el método `setUpTest`, mostrado en el Código 4.5, se encarga de preparar el entorno necesario antes de cada prueba.

Código 4.5: Metodo setUpTest de las pruebas Selenium

```
@BeforeEach
public void setUpTest() {
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--headless");
    options.addArguments("--allow-insecure-localhost");
    options.addArguments("--disable-gpu");

    driver = new ChromeDriver(options);
    wait = new WebDriverWait(driver, Duration.ofSeconds(10));
}
```

En el contexto específico de este proyecto, las pruebas automatizadas de Selenium incluyen varios escenarios clave:

- **LoginTest:** Verifica la funcionalidad de inicio de sesión, tanto con credenciales válidas como inválidas (Código 4.6). Asegura que los usuarios puedan iniciar sesión correctamente y que el sistema maneje adecuadamente los intentos de inicio de sesión fallidos.

Código 4.6: Método testLoginWithInvalidCredentials de LoginTest.java

```
@Test
public void testLoginWithInvalidCredentials() {

    driver.get("https://localhost:" + this.port + "/login")
        ;
```

```
5       WebElement usernameField = driver.findElement(By.id("
            username"));
6       WebElement passwordField = driver.findElement(By.id("
            password"));
7       WebElement loginButton = driver.findElement(By.id("
            loginButton"));
8
9       usernameField.sendKeys("invalidUsername");
10      passwordField.sendKeys("invalidPassword");
11      loginButton.click();
12
13      assertTrue(driver.getCurrentUrl().endsWith("/login"));
14  }
```

- **SideBarTest:** Comprueba la correcta navegación y accesibilidad de diferentes secciones de la aplicación a través del menú de la barra lateral. Garantiza que los enlaces de navegación dirijan a las páginas correctas y que la interfaz sea coherente con la funcionalidad esperada.

- **TicketTest:** Simula la creación, asignación, edición, y eliminación de tickets. Verifica que las operaciones sobre tickets, como la asignación a usuarios y el cambio de estado, funcionen correctamente y que los datos se actualicen como se espera.

- **ReleaseAndProjectTest:** Asegura que las funcionalidades relacionadas con la gestión de lanzamientos y proyectos se comporten correctamente. Verifica la capacidad de crear, editar, y eliminar lanzamientos y proyectos, y que los datos se gestionen adecuadamente en la interfaz de usuario.

4.4.3. Cobertura

La cobertura de pruebas es una métrica crucial que mide el porcentaje del código fuente que se ejecuta al menos una vez durante la ejecución de las pruebas. Esta métrica proporciona una visión general de la efectividad de las pruebas al mostrar qué tan exhaustivamente se ha probado el código.

En el proyecto se cuenta con un total de 187 tests que se encargan de cubrir el 88,6 % de la lógica de la aplicación cubriendo así 21.261 instrucciones (Figura 4.22).

En el contexto de este proyecto, se utiliza la herramienta JaCoCo para generar informes detallados de cobertura de código. JaCoCo analiza el código ejecutado durante las pruebas y reporta qué líneas, ramas y métodos han sido cubiertos. Estos informes ayudan a identificar áreas del código que no están suficientemente probadas, permitiendo enfocar los esfuerzos de prueba en esas áreas.

Además, como parte del proceso de integración continua, se ha establecido que la cobertura de código no puede bajar del 80 %. Esta medida asegura que el código no empeore con las nuevas funcionalidades o cambios, manteniendo un nivel mínimo de calidad en el software y evitando que se introduzcan áreas no probadas en el código base.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
alcalapp	95,3 %	21.261	1.050	22.311
src/main/java	88,6 %	8.066	1.038	9.104
com.app.alcala	37,5 %	3	5	8
com.app.alcala.configuration	100,0 %	107	0	107
com.app.alcala.entities	89,6 %	3.030	353	3.383
com.app.alcala.service.impl	88,0 %	2.634	360	2.994
com.app.alcala.utils	98,2 %	161	3	164
com.app.alcala.web.controller	90,5 %	984	103	1.087
com.app.alcala.web.model	84,3 %	1.147	214	1.361
src/test/java	99,9 %	13.195	12	13.207

Figura 4.22: Cobertura de los test de AlcalApp

4.5. Distribución y Despliegue

4.5.1. Docker

Docker es una plataforma de código abierto que automatiza la implementación de aplicaciones dentro de contenedores. Los contenedores son entornos ligeros y aislados que empaquetan la aplicación y todas sus dependencias, lo que garantiza que el software se ejecute de manera consistente en cualquier entorno.

En este proyecto, Docker se utiliza para crear imágenes de la aplicación, lo que facilita su despliegue en diversos entornos. En particular, se crea una imagen Docker del proyecto bajo el repositorio `elisaalcala/alcalapp`, que incluye la aplicación y todas sus dependencias necesarias para su correcta ejecución.

Además, para las pruebas, se crea una imagen de la base de datos, específicamente diseñada para contener los datos y el esquema necesarios para la ejecución de los tests. Esta imagen se encuentra en el repositorio `elisaalcala/mydb`, permitiendo que la base de datos se ejecute en un contenedor independiente, lo que facilita la integración de la aplicación con una base de datos preconfigurada y lista para pruebas sin la necesidad de un servidor de base de datos externo.

4.5.2. Microsoft Azure

Azure es la plataforma de nube de Microsoft que ofrece una amplia gama de servicios para desarrollar, alojar y administrar aplicaciones. Es conocida por su flexibilidad, escalabilidad y capacidades de integración con herramientas DevOps. Para este proyecto, Azure proporciona una infraestructura confiable y económica para ejecutar aplicaciones web modernas.

Antes de crear los recursos necesarios para el despliegue de la aplicación, se realizó la configuración inicial de Azure creando un grupo de recursos y una identidad administrada como muestra la Figura 4.23

Nombre ↑		Tipo	Ubicación	Grupo de recursos
alcalapp	···	Instancias de contenedor	West Europe	alcalapp
db-alcalapp	···	Servidor flexible de Azure Database for MySQL	Spain Central	alcalapp
alcalapp	···	Grupo de recursos		alcalapp
elisaalcala	···	Identidad administrada	West Europe	alcalapp

Figura 4.23: Recursos creados para el despliegue en Microsoft Azure

Grupo de Recursos e Identidad Administrada

Un grupo de recursos en Azure es un contenedor lógico que organiza todos los recursos relacionados con un proyecto. En este caso, se creó un grupo de recursos específico llamado “alcalapp” para agrupar todos los componentes del proyecto, como los contenedores, bases de datos y otros servicios necesarios. Esta organización facilita la gestión, supervisión y mantenimiento de los recursos, además de permitir una administración centralizada de los mismos.

Por otro lado, la identidad administrada en Azure es un servicio que permite la autenticación sin necesidad de gestionar credenciales de manera manual. Al asignar una identidad administrada a los servicios que requieren acceso entre sí, como la aplicación en contenedores y la base de datos, se garantiza un acceso seguro y automatizado entre los componentes, evitando la exposición de credenciales sensibles.

Azure Container Instances (ACI)

Azure Container Instances (ACI) es un servicio que permite ejecutar contenedores de Docker directamente en la nube sin necesidad de gestionar infraestructura subyacente como máquinas virtuales o clústeres de Kubernetes. Este enfoque

reduce la complejidad y los tiempos de despliegue, asegurando un rendimiento óptimo para aplicaciones basadas en contenedores.

Para este proyecto, se utiliza ACI para desplegar la aplicación, que se encuentra contenida en una imagen Docker almacenada en DockerHub. La configuración incluye la definición de variables de entorno esenciales para la integración con otros servicios, como la base de datos.

Azure MySQL Flexible Server

Azure MySQL Flexible Server es un servicio administrado que proporciona una solución segura para bases de datos MySQL. En este proyecto, la base de datos MySQL Flexible Server se utiliza para almacenar los datos de la aplicación, como usuarios, proyectos e incidencias.

Al igual que con ACI, para crear el servidor MySQL fue necesario asignarlo al mismo grupo de recursos. Gracias a la identidad administrada, la aplicación en contenedores puede autenticar de forma segura sus conexiones a la base de datos sin necesidad de gestionar credenciales manualmente.

4.5.3. Git Hub Actions

Para el despliegue y distribución del proyecto, se ha adoptado un enfoque ágil utilizando la metodología Git Flow, que facilita la gestión de ramas y la organización del trabajo. Se han creado ramas específicas para desarrollo (develop), producción (master), características (feature) y correcciones urgentes (hotfix), asegurando un flujo de trabajo estructurado y claro.

Los cambios se desarrollan en ramas feature según las tareas propuestas y, una vez completados, se fusionan en la rama develop, donde se realizan pruebas automáticas. Cuando una versión está lista para ser liberada, se fusiona con la rama master, que representa el estado de producción, activando el proceso de despliegue.

En caso de que sea necesario realizar cambios urgentes en producción, como correcciones críticas, se crean ramas hotfix a partir de master. Una vez solucionado el problema, estos cambios se fusionan tanto en la rama master como en la rama develop, asegurando que la corrección esté disponible en producción y en la línea de desarrollo para futuras versiones.

El despliegue y distribución del proyecto se gestiona de manera automatizada utilizando tres workflows clave en GitHub Actions: Push to Develop, Pull Request to Master, y Push to Master. Cada uno desempeña un papel específico dentro de la metodología Git Flow para garantizar un flujo de trabajo continuo, eficiente

y sin interrupciones. A continuación, se explica el propósito y funcionamiento de cada workflow.

Push to Develop

Este workflow se activa cada vez que se realiza un push a la rama develop. Su objetivo principal es garantizar que el código en desarrollo sea válido y cumpla con los estándares establecidos antes de avanzar en el flujo. Las tareas realizadas incluyen:

1. **Configuración de servicios:** Se utiliza una imagen de MySQL con la configuración específica del proyecto como se muestra en el Código 4.7.

   Código 4.7: Configuración del workflow Push_Develop.yaml

```yaml
services:
  mysql:
    image: elisaalcala/mydb:latest
    env:
      MYSQL_ROOT_PASSWORD: 3301
      MYSQL_DATABASE: alcalapp_db
    ports:
      - 3306:3306
    options: --health-cmd="mysqladmin ping -h localhost" --health-interval
       =10s --health-timeout=5s --health-retries=3
```

2. **Compilación y pruebas:** Se compila el proyecto con Maven y se ejecutan sólo las pruebas unitarias, como se muestra en el Código 4.8, ya que son ligeras y rápidas, permitiendo un ciclo de retroalimentación ágil.

   Código 4.8: Steps para compilar la aplicación de Push_Develop.yaml

```yaml
name: Checkout repository
uses: actions/checkout@v3

name: Set up JDK 17
uses: actions/setup-java@v3
with:
  distribution: 'temurin'
  java-version: '17'

name: Build with Maven
run: mvn clean install --file pom.xml

name: Run unit tests
run: mvn test -Dtest="com.app.alcala.service.impl.*Test,
       com.app.alcala.web.controller.*Test,com.app.alcala.web.model.*Test,
           com.app.alcala.entities.*Test"
```

3. **Versionado Automático:** Como se muestra en el Código 4.9, dependiendo del tipo de commit (fix, feat o !), se incrementa automáticamente el número de versión del proyecto en los componentes de patch, minor o major:

   a) Componente Patch: Este componente se incrementa cuando se realizan correcciones de errores que no afectan a la funcionalidad existente.

Por ejemplo, un commit con un mensaje que empieza con “fix” indica que se ha resuelto un problema en el código. Un incremento en este componente resulta en una nueva versión del tipo 0.0.X, donde X es el nuevo número de parche.

b) Componente Minor: Cuando se añaden nuevas funcionalidades de manera retrocompatible, el componente minor se incrementa. Un commit que comienza con “feat” señala la inclusión de una nueva característica. Esto resulta en una versión como 0.X.0, donde X es el nuevo número menor.

c) Componente Major: Si se introducen cambios incompatibles que rompen la funcionalidad existente, el componente mayor se incrementa. Un mensaje de commit que comience con “!” indicará que se ha realizado un cambio significativo que afecta la API o la manera en que se utiliza el software. Esto produce una versión del tipo X.0.0, donde X es el nuevo número mayor.

Código 4.9: Step para realizar el versionado automático en Push_Develop.yaml

```
name: Determine Version Increment
id: determine_increment
run: |
  COMMIT_MSG="${{ github.event.head_commit.message }}"
  CURRENT_VERSION=${{ env.CURRENT_VERSION }}

  if [[ "$COMMIT_MSG" == *"fix:"* ]]; then
    # Incrementar patch
    NEW_VERSION=$(echo $CURRENT_VERSION | awk -F. -v OFS=. '{$NF++; print }')
  elif [[ "$COMMIT_MSG" == *"feat:"* ]]; then
    # Incrementar minor y resetear patch
    NEW_VERSION=$(echo $CURRENT_VERSION | awk -F. -v OFS=. '{$(NF-1)++; $NF=0; print}')
  elif [[ "$COMMIT_MSG" == *"!:"* ]]; then
    # Incrementar major y resetear minor y patch
    NEW_VERSION=$(echo $CURRENT_VERSION | awk -F. -v OFS=. '{$1++; $(NF-1)=0; $NF=0; print}')
  else
    echo "No se encontro ningun patron de incremento en el mensaje de commit. No se realizaran cambios."
    exit 0
  fi

  echo "NEW_VERSION=${NEW_VERSION}-SNAPSHOT" >> $GITHUB_ENV
```

Pull Request to Master

Este workflow se ejecuta al abrir un pull request hacia la rama master, y su propósito es validar que el código en desarrollo esté listo para producción. Este paso es crítico para evitar la promoción de código defectuoso al entorno de producción. Incluye las siguientes tareas:

1. **Pruebas automáticas:** Esta vez se ejecutan todos los test disponibles, incluyendo las pruebas E2E.

2. **Verificación de cobertura de código:** Usando JaCoCo, se genera un informe de cobertura y se verifica que al menos el 80 % del código esté cubierto por las pruebas. Si este umbral no se cumple, el workflow falla, asegurando que no se comprometa la calidad del código. En el Código 4.10 se puede observar como se realiza esta función.

Código 4.10: Step para comprobar la cobertura de Pull_Master.yaml

```
name: Check code coverage
run: |
  mvn jacoco:report

  COVERED=$(grep -oP '<counter type="INSTRUCTION" missed="\d+" covered="\d+"' target/site/jacoco/jacoco.xml | \
      awk -F'covered="' '{sum_covered += $2} END {print sum_covered}')

  MISSED=$(grep -oP '<counter type="INSTRUCTION" missed="\d+" covered="\d+"' target/site/jacoco/jacoco.xml | \
      awk -F'missed="' '{sum_missed += $2} END {print sum_missed}')

  TOTAL=$(( COVERED + MISSED ))

  if [ "$TOTAL" -eq 0 ]; then
    echo "No coverage data found. Failing the build."
    exit 1
  fi

  COVERAGE_PERCENT=$(( 100 * COVERED / TOTAL ))
  echo "coverage: $COVERED%"
  echo "Code total: $TOTAL%"
  echo "Code coverage percentage: $COVERAGE_PERCENT%"

  if [ "$COVERAGE_PERCENT" -lt 80 ]; then
    echo "Code coverage is below 80% ($COVERAGE_PERCENT%). Failing the build."
    exit 1
  fi
```

Push to Master

El último workflow se activa cuando se realiza un push a la rama master, señalando la liberación de una nueva versión en producción. Las tareas realizadas son:

1. **Eliminación del sufijo -SNAPSHOT:** Se convierte la versión actual en una versión de producción, eliminando el sufijo -SNAPSHOT como se muestra en el Código 4.11.

Código 4.11: Steps para realizar el versionado automático de Push_Master.yaml

```
name: Get Current Version
id: get_version
run: |
```

```
  CURRENT_VERSION=$(mvn help:evaluate -Dexpression=project.version -q -
      DforceStdout)
  echo "CURRENT_VERSION=${CURRENT_VERSION}" >> $GITHUB_ENV

name: Remove SNAPSHOT from Version
run: |
  RELEASE_VERSION=$(echo ${{ env.CURRENT_VERSION }} | sed 's/-SNAPSHOT//')
  mvn versions:set -DnewVersion=$RELEASE_VERSION
  echo "RELEASE_VERSION=${RELEASE_VERSION}" >> $GITHUB_ENV
```

2. **Generación de un tag:** La nueva versión se registra en un commit y se crea un tag en GitHub con el número de versión. En el Código 4.12 se muestra es step para crear el nuevo tag.

Código 4.12: Step para crear la nueva etiqueta en Git de Push_Master.yaml

```
name: Create new tag for release
run: |
  git tag -a "v${RELEASE_VERSION}" -m "Release version ${RELEASE_VERSION}"
  git push origin "v${RELEASE_VERSION}"
env:
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

3. **Construcción y publicación del contenedor Docker:** Se genera una imagen Docker como se muestra en el Código 4.13 y se etiqueta con el número de versión. La imagen se publica automáticamente en DockerHub.

Código 4.13: Step para crear y publicar la imagen en Docker de Push_Master.yaml

```
name: Generate Docker image
run: mvn spring-boot:build-image -DskipTests -Dspring-boot.build-image.
    imageName=${{ secrets.DOCKERHUB_USERNAME }}/alcalapp:${{ steps.project.
    outputs.tag }}

name: Login to DockerHub
run: echo "${{ secrets.DOCKERHUB_TOKEN }}" | docker login -u "${{ secrets.
    DOCKERHUB_USERNAME }}" --password-stdin

name: Push image to DockerHub
run: docker push ${{ secrets.DOCKERHUB_USERNAME }}/alcalapp:${{ steps.
    project.outputs.tag }}
```

4. **Despliegue en Azure:** Se utiliza Azure Container Instances para desplegar la imagen en un entorno de producción. Se configuran variables de entorno necesarias, como las credenciales de base de datos y la URL de conexión.

Código 4.14: Job para desplegar la aplicación en Azure de Push_Master.yaml

```
deploy:
runs-on: ubuntu-latest
needs: publish

steps:
  - name: 'Az CLI login'
    uses: azure/login@v2
    with:
      client-id: ${{ secrets.AZURE_CLIENT_ID }}
      tenant-id: ${{ secrets.AZURE_TENANT_ID }}
      subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
```

```
12
13
14  - name: 'Run az commands to deploy container'
15    run: |
16      az container create \
17        --resource-group ${{ secrets.AZURE_GROUP }} \
18        --name alcalapp \
19        --image ${{ secrets.DOCKERHUB_USERNAME }}/alcalapp:${{ needs.
              publish.outputs.tag }} \
20        --dns-name-label elisaalcala-alcalapp \
21        --registry-login-server index.docker.io \
22        --registry-username ${{ secrets.DOCKERHUB_USERNAME }} \
23        --registry-password ${{ secrets.DOCKERHUB_READ_TOKEN }} \
24        --ports 8443 \
25        --environment-variables \
26          DB_URL="jdbc:mysql://db-alcalapp.mysql.database.azure.com:3306/
                alcalapp_db?serverTimezone=UTC&useSSL=true" \
27          DB_USERNAME="${{ secrets.DOCKERHUB_USERNAME }}" \
28          DB_PASSWORD="${{ secrets.DB_PRO_PASSWORD }}"
```

5

Conclusiones y trabajos futuros

En este capítulo se detallan las conclusiones derivadas del TFG y la propuesta de posibles trabajos futuros.

5.1. Conclusión

El desarrollo de esta aplicación web para la gestión de tareas e incidencias ha sido una experiencia enriquecedora que me ha permitido profundizar en diversas áreas del desarrollo de software. A lo largo del proceso, no solo he aplicado conocimientos técnicos adquiridos durante mi formación, sino que también he desarrollado nuevas habilidades.

Durante el desarrollo, apliqué mis conocimientos en Java y Spring, tecnologías que ya había utilizado anteriormente y que, aunque no me resultaron del todo novedosas, me brindaron un sólido punto de partida. A lo largo del proyecto, logré implementar funcionalidades clave como la gestión de tareas e incidencias, lo que permitió cumplir con uno de los objetivos principales desde el inicio.

Sin embargo, me enfrenté a mayores desafíos con el front-end. Mis conocimientos en este ámbito eran más limitados, ya que nunca había utilizado plantillas JSP ni trabajado con Bootstrap u otras librerías CSS. No obstante, al integrar estas herramientas en el proyecto, descubrí su gran utilidad para simplificar el diseño y mejorar la experiencia del usuario, facilitando la creación de una interfaz más atractiva y funcional. A medida que avanzaba en el desarrollo, noté una mejora

significativa en mi capacidad para organizar y construir la interfaz de usuario con mayor rapidez y eficacia, lo que también contribuyó al cumplimiento de mis objetivos de diseño.

Uno de los aspectos que representó un importante desafío fue el despliegue de la aplicación en Azure, tecnología que no había utilizado previamente en mi formación académica. Durante este proceso, uno de los retos más significativos fue la implementación del servicio de base de datos MySQL en la nube. La falta de experiencia en el uso de servicios de base de datos en la nube me obligó a investigar a fondo cómo configurar adecuadamente MySQL en Azure y cómo integrarlo con la aplicación. Este proceso requirió que me familiarizara con aspectos técnicos esenciales, como la configuración de redes para asegurar la conectividad, las medidas de seguridad para proteger los datos y la gestión de credenciales, demandando una adaptación cuidadosa para cumplir con los requisitos de la plataforma. A pesar de las dificultades iniciales, esta experiencia me brindó la oportunidad de adquirir nuevas habilidades y conocimientos en el uso de bases de datos en la nube, enriqueciendo mi experiencia en el despliegue de aplicaciones en entornos de producción.

En proyectos grandes, tener una metodología clara y bien definida es esencial, ya que, sin ella, es fácil que el desarrollo se demore considerablemente. La falta de un enfoque estructurado puede derivar en caos y pérdida de tiempo, incluso en tareas sencillas como, por ejemplo, la redacción de mensajes de commit, que si no se realizan correctamente, pueden generar importantes desventajas a largo plazo. Al inicio de este proyecto, enfrenté múltiples dificultades debido a la falta de organización, lo que complicó la revisión de errores y el seguimiento del progreso. Para superar estos obstáculos, establecí numerosas normas, incluyendo la creación de un sistema de versionado automático y la definición de un flujo de trabajo más riguroso. Estas medidas fueron clave no solo para avanzar sin retrasos innecesarios, sino también para garantizar que se cumplieran los objetivos del proyecto.

Aunque este proyecto no implicó trabajo en equipo, lo que me impidió implementar las metodologías ágiles de manera estricta, fue una oportunidad valiosa para familiarizarme con ellas y apreciar su relevancia. La integración de ciertos conceptos ágiles, como la adaptación continua y la retroalimentación constante, resultó esencial para el desarrollo. Estoy segura de que, en futuros proyectos colaborativos, podré aprovechar estos conocimientos de manera más efectiva y contribuir de manera más significativa al éxito del equipo.

5.2. Trabajos futuros

En cuanto a los trabajos futuros, considero que hay varias áreas donde podría expandir y mejorar la aplicación.

Una de las primeras mejoras sería establecer un sistema de roles más detallado que refleje con mayor precisión la estructura organizativa de una empresa real. Esto incluiría definir roles adicionales cada uno con permisos y responsabilidades claramente especificados. Esta funcionalidad permitiría asignar accesos de manera más realista y alineada con los flujos de trabajo de una organización actual, mejorando tanto la seguridad como la eficiencia en la gestión de proyectos.

Otra posible mejora sería la implementación de un sistema de mensajería interna y notificaciones en tiempo real. Este sistema permitiría que cada usuario recibiera alertas sobre actualizaciones en las incidencias que ha abierto o en las que está participando. Esto mejoraría la interacción y aseguraría que los usuarios estén siempre al tanto del estado de las tareas sin tener que acceder manualmente al sistema.

Además, sería útil añadir la posibilidad de dejar comentarios en las incidencias. Esto facilitaría la comunicación entre los miembros del equipo, permitiendo un intercambio más fluido de información y seguimiento dentro de cada incidencia. Con esta característica, los usuarios podrían discutir detalles específicos de las incidencias, adjuntar información relevante y resolver problemas de manera más eficiente.

Por último, se podrían implementar métricas que comparen el rendimiento entre diferentes equipos, lo que facilitaría la identificación de buenas prácticas y áreas de mejora. Por ejemplo, métricas como la tasa de resolución de incidencias por equipo, el tiempo medio de respuesta entre equipos, o el volumen de incidencias abiertas y cerradas en un periodo determinado permitirían evaluar el rendimiento colectivo. Estas métricas globales no solo ofrecerían una perspectiva más amplia del desempeño organizacional, sino que también fomentarían una sana competencia entre equipos, impulsando así la mejora continua en toda la organización.

Bibliografía

[1] O. Corporation, "Java," Online, n.d. [Online]. Available: https://www.oracle.com/java/

[2] I. Pivotal Software, "Spring framework," Online, n.d. [Online]. Available: https://spring.io/projects/spring-framework

[3] ——, "Spring boot," Online, n.d. [Online]. Available: https://spring.io/projects/spring-boot

[4] A. S. Foundation, "Apache maven," Online, n.d. [Online]. Available: https://maven.apache.org/

[5] O. Corporation, "Mysql," Online, n.d. [Online]. Available: https://www.mysql.com/

[6] I. Red Hat, "Hibernate orm," Online, n.d. [Online]. Available: https://hibernate.org/orm/

[7] A. S. Foundation, "Apache tomcat," Online, n.d. [Online]. Available: https://tomcat.apache.org/

[8] P. Lombok, "Project lombok," Online, n.d. [Online]. Available: https://projectlombok.org/

[9] I. Pivotal Software, "Spring security," Online, n.d. [Online]. Available: https://spring.io/projects/spring-security

[10] S. S. Team, "Spring security taglibs," Online, n.d. [Online]. Available: https://docs.spring.io/spring-security/reference/servlet/integrations/jsp-taglibs.html

[11] A. S. Foundation, "Apache httpclient," Online, n.d. [Online]. Available: https://hc.apache.org/httpcomponents-client-5.1.x/

[12] ——, "Apache httpcore," Online, n.d. [Online]. Available: https://hc.apache.org/httpcomponents-core-5.1.x/

[13] J. Team, "Junit 5," Online, n.d. [Online]. Available: https://junit.org/junit5/

[14] M. Team, "Mockito," Online, n.d. [Online]. Available: https://site.mockito.org/

[15] S. Project, "Selenium," Online, n.d. [Online]. Available: https://www.selenium.dev/

[16] G. Sironi, "Webdrivermanager," Online, n.d. [Online]. Available: https://github.com/bonigarcia/webdrivermanager

[17] J. Team, "Jacoco," Online, n.d. [Online]. Available: https://www.jacoco.org/jacoco/trunk/doc/

[18] I. Docker, "Docker," Online, n.d. [Online]. Available: https://www.docker.com/

[19] M. Foundation, "Javascript," Online, n.d. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[20] I. Twitter, "Bootstrap," Online, n.d. [Online]. Available: https://getbootstrap.com/

[21] jQuery Foundation, "jquery," Online, n.d. [Online]. Available: https://jquery.com/

[22] DataTables, “Datatables,” Online, n.d. [Online]. Available: https://datatables.net/

[23] Chart.js, “Chart.js,” Online, n.d. [Online]. Available: https://www.chartjs.org/

[24] flatpickr, “flatpickr,” Online, n.d. [Online]. Available: https://flatpickr.js.org/

[25] W. W. W. C. (W3C), “Html,” Online, n.d. [Online]. Available: https://www.w3.org/TR/html52/

[26] ——, “Css,” Online, n.d. [Online]. Available: https://www.w3.org/Style/CSS/

[27] FontAwesome, “Fontawesome: Icons for everyone,” Online, n.d. [Online]. Available: https://fontawesome.com/

[28] O. Corporation, “Javaserver pages,” Online, n.d. [Online]. Available: https://www.oracle.com/java/technologies/java-server-pages.html

[29] ——, “Javaserver pages standard tag library,” Online, n.d. [Online]. Available: https://javaee.github.io/jstl-api/

[30] JSON.org, “Json,” Online, n.d. [Online]. Available: https://www.json.org/json-en.html

[31] FasterXML, “Jackson,” Online, n.d. [Online]. Available: https://github.com/FasterXML/jackson

[32] O. Corporation, “Java architecture for xml binding (jaxb),” Online, n.d. [Online]. Available: https://jaxb.java.net/

[33] I. Pivotal Software, “Spring boot devtools,” Online, n.d. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/using.html#using-devtools

[34] ——, “Spring tools suite,” Online, n.d. [Online]. Available: https://spring.io/tools

[35] M. Corporation, “Visual studio code,” Online, n.d. [Online]. Available: https://code.visualstudio.com/

[36] L. Torvalds, “Git,” Online, n.d. [Online]. Available: https://git-scm.com/

[37] I. GitHub, “Github,” Online, n.d. [Online]. Available: https://github.com/

[38] O. Corporation, “Mysql workbench,” Online, n.d. [Online]. Available: https://www.mysql.com/products/workbench/

[39] I. Docker, “Docker hub,” Online, n.d. [Online]. Available: https://hub.docker.com/

[40] M. Corporation, “Microsoft azure,” Online, n.d. [Online]. Available: https://azure.microsoft.com/

[41] M. Azure, “Azure container instances,” Online, n.d. [Online]. Available: https://learn.microsoft.com/en-us/azure/container-instances/

[42] ——, “Azure database for mysql - flexible server,” Online, n.d. [Online]. Available: https://learn.microsoft.com/en-us/azure/mysql/flexible-server/overview

[43] I. GitHub, “Github flow,” Online, n.d. [Online]. Available: https://docs.github.com/en/get-started/using-github/github-flow

[44] K. Schwaber and J. Sutherland, “The scrum guide,” Online, 2020. [Online]. Available: https://scrumguides.org

[45] I. GitHub, “Github actions,” Online, n.d. [Online]. Available: https://docs.github.com/en/actions

[46] Spring Team, “Spring web mvc,” Online, n.d. [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webmvc.html

Apéndices

A

Repositorio GitHub

A.1. Enlace al repositorio

El proyecto ha sido alojado en la plataforma Github.

El enlace es el siguiente: https://github.com/elisaalcala/alcalapp