Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Aplicación Web para la Gestión de Proyectos

Estudante: Jose Antonio Dosil Toja
Dirección: Jose Losada Perez

A Coruña, febreiro de 2020.

A mi familia, por estar ahí siempre a mi lado apoyándome

Agradecimientos

A mi familia porque siempre han estado ahí apoyándome y motivándome para seguir adelante durante todos estos años, y en especial a ti, Macu, que aunque ya no estés, siempre estaré agradecido por todo lo que has hecho por mí.

A todos los compañeros y amigos de la carrera con los que he compartido estrés y risas y me han hecho mas ameno el camino.

Resumen

El objetivo de este proyecto es el desarrollo de una aplicación web para la Gestión de Proyectos con una arquitectura Modelo-Vista-Controlador utilizando tecnologías Java. La aplicación permitirá la gestión de los diferentes elementos de un proyecto, siendo estés las tareas que lo componen y los recursos humanos que las realizaran. Para ello, permitirá la creación, modificación y eliminación de proyectos, tareas y usuarios, así como la asignación de tareas a usuarios para determinar el responsable de realizarlas. También se contará con las funcionalidad de buscar proyectos, tareas y usuarios usando una variedad de filtros de búsqueda.

La aplicación permitirá además añadir relaciones entre las tareas y la creación de ciclos de desarrollo con los que agruparlas, lo cual puede ayudar en su gestión y organización, así como añadir estimaciones a las tareas y registrar el tiempo dedicado, lo cual puede ser una ayuda para la planificación y seguimiento de un proyecto.

Por último, también se podrá conectar con una cuenta de GitLab para asignar un repositorio a un proyecto, asignar una branch de git a un ciclo de desarrollo, o localizar los commits que estén asociados con una tarea concreta del proyecto.

Todo este proceso se llevará a cabo utilizando principalmente Proceso Unificado de Desarrollo, en conjunto con elementos de otras metodologías como Scrum o Kanban.

La aplicación se ha llamado ***GesTa*** que proviene de las palabras **GES**tión y **TA**rea.

Abstract

The objetive of this project is the design and development of a Project Managment web application using MVC architecture and Java technologies. The application will allow the management of diferent elements of a project: the tasks that compose it and the human resource that will perform them. To allow this, users will be able to create, modify and remove projects, tasks and users, as well as assign tasks to users. It will also be posible to search projects.

Users will be able to link tasks and create development cycles to group tasks, which will help with task organization and management, as well as add estimates and log time, which can help with project planification and tracking.

Finally, the application will offer the option to connect to a GitLab account to assign a repository to a project, assign a git branch to a development cycle or search the commits linked to a task.

The methodology that will be used to create the application is Unified Software Development Process, along with elements from other methodologies like Scrum or Kanban.

The name of the application is ***GesTa*** , that comes from the spanish words **GES**tión (managment) and **TA**rea (task).

Palabras clave:

- √ Sencillez
- √ Gestión de Proyectos
- √ Gestión de Tareas
- √ Software
- √ Metodologías
- √ Java
- √ Spring
- √ Thymeleaf
- √ Bootstrap

Keywords:

- √ Simplicity
- √ Project Management
- √ Task Management
- √ Software
- √ Methodologies
- √ Java
- √ Spring
- √ Thymeleaf
- √ Bootstrap

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Título del proyecto y datos principales

El Trabajo de fin de grado aquí expuesto lleva por titulo "**Aplicación Web para la Gestión de Proyectos**"

Tipo: Proyecto clásico de Ingeniería.

Especialización: Este proyecto se enmarca dentro del Grado en Ingeniería Informática en la mención de Ingeniería de Software por la Facultad de Informática de A Coruña.

1.2 Contexto y motivación

Un gestor de proyectos es una aplicación utilizada tanto por empresas como por equipos independientes para gestionar los diferentes elementos y etapas por los que puede pasar un proyecto. Estos ayudan a declarar y visualizar por que fases ha de pasar un proyecto, que tareas han de realizarse y quien ha de realizarlas.

Los gestores de proyectos son muy utilizados en el ámbito del desarrollo software permitiendo aplicar de forma mas eficiente una metodología y ayudando a mantener el control sobre los diferentes elementos de un proyecto Software. Principalmente por esta razón, que sean muy utilizados en software se ha decidido realizar este proyecto, para poder comprender desde otra perspectiva las necesidades que un gestor de proyectos ha de cumplir para ser sencillo, fácil de utilizar, pero a la vez totalmente útil en diferentes entornos de desarrollo.

1.3 Estado de la cuestión

En el mundo existen diversas aplicaciones que sirven para la gestión de proyectos y tareas, cada una con diferentes enfoques y existen tanto gratuitas, comerciales y proyectos open source. Aun en las diferencias, siempre se enfocan en la creación, planificación y seguimiento de tareas.

Entre ellas se encuentra **Redmine**[1]. Redmine es una aplicación open source desarrollada en Ruby on Rails. Entre otros, nos permite hacer una gestión de proyectos y tareas con: horas estimadas para la realización de una tarea; establecimiento de fecha inicio y fin; estado de las tareas configurables; porcentaje de realización de cada tarea; registro de horas dedicadas, así como otras herramientas para la visualización del estado de los proyectos como los Diagramas Gantt. También se pueden añadir mas herramientas desarrolladas por terceros, permitiéndole cubrir muchas de las necesidades que conlleva gestionar un proyecto.

Otras de las herramientas es **Jira**. Jira es una aplicación comercial desarrollada por la empresa Attlassian. Si bien nos permite realizar las mismas acciones que Redmine, estas están mas refinadas y orientadas a grandes entornos organizativos. También permite extensiones y personalizar para enfocar cada proyecto de la forma mas adecuada, así como conectividad con herramientas externas, para ampliar las posibilidades de la aplicación. Entre otros, también cuenta con registro de versiones, las cuales conectándolas a herramientas como Bamboo, software de Atlassian para la integración continua, es posible compilar y generar versiones entrégales. Estas versiones también se pueden asociar a las tareas para ayudar a su seguimiento.

Estos son sólo dos ejemplos de la gran variedad de herramientas que existen en el mercado. De forma general todas las herramientas nos permiten la creación, planificación y seguimiento de tareas y proyectos, variando en cada una de ellas los enfoques concretos de implementación, enfocándose algunos en facilitar el desarrollo en ciertas metodologías, siendo frecuente el enfoque en metodologías como Scrum o Kanban, mientras que otros generalizan para que pueda ser utilizado por cualquier clase de equipo o proyecto.

En el caso del enfoque de GesTa, se ha basado en la experiencia en el uso de Jira y Redmine, no para crear un sustituto, si no para analizar desde otra perspectiva las necesidades de un equipo de desarrollo.

1.4 Objetivos

Los objetivos a abordar en este proyecto son los siguientes:

- Diseñar una aplicación web con arquitectura MVC.
- Implementar dicha aplicación utilizando Java y otras tecnologías relacionadas.

- Trabajar con un enfoque de desarrollo iterativo.
- Gestionar los usuarios de la aplicación, habiendo tres tipos: usuario registrado, lideres o jefes de proyecto y Administradores.
- Limitar el acceso a diferentes partes de la aplicación atendiendo a los diferentes tipos de usuario definidos.
- Implementar páginas para ver los detalles de los Proyectos, Ciclos de Desarrollo y Tareas, teniendo en cuenta las limitaciones de acceso de cada tipo de usuario.
- Implementar buscadores para encontrar tareas y proyectos, teniendo en cuenta las limitaciones de cada tipo de usuario.
- Conectarse a la API de GitLab para poder asociar o buscar información relacionada con los diferentes elementos de la aplicaciones (proyectos, ciclos y tareas).
- Soportar internacionalización de los idiomas Gallego, Castellano e Inglés.
- Utilizar Bootstrap y JQuery para para crear una interfaz de usuario dinámica y responsive.
- Utilizar Spring MVC y Thymeleaf para el subsistema controlador.
- Utilizar hibernate para el subsistema del modelo.

1.5 Estructura de la memoria

La memoria se ha estructurado en los siguientes cuatro capítulos:

- **Capitulo 1 - Introducción**: Breve descripción del proyecto, de los motivos que llevaron a su desarrollo y de los objetivos.
- **Capitulo 2 - Tecnologías y Herramientas**: Listado y explicación de las diferentes herramientas utilizadas en el proyecto.
- **Capitulo 3 - Planificación, Diseño y Desarrollo**: Explicación de la metodología utilizada, así como de los detalles de la planificación, diseño y desarrollo del proyecto.
- **Capitulo 4 - Validación**: Detalles de las pruebas realizadas.
- **Capitulo 5. Manual de Usuario**: Instrucciones sobre el uso de la herramienta.
- **Capítulo 6. Conclusiones**: Evaluación final, conclusiones a las que se han llegado al finalizar el desarrollo y lineas de trabajo futuras.

Capítulo 2

Tecnologías y Herramientas

En este apartado describiremos las diferentes tecnologías y herramientas que se han utilizado en el desarrollo de la aplicación.

2.1 Git

Git[2] es un software de control de versiones open source diseñado por Linus Torvalds. Su diseño se basa en la experiencia de Linus Torvalds en el desarrollo y mantenimiento de grandes cantidades de código para el Kernel de Linux, en la que participa mucha gente. Incide en el rendimiento y en la facilidad de desarrollo colaborativo al permitir de una forma eficiente el desarrollo en paralelo.

2.2 GitLab

GitLab [3] es una aplicación web de control de versiones y desarrollo de software basado en Git. Además nos ofrece una suite de herramientas para realizar Integración Continua y Entrega Continua entre otros, lo que nos permite, en una sola aplicación realizar una gran variedad de acciones que proporcionan ayuda y soporte en un desarrollo software.

Si bien empezó como un proyecto de Open Source, su desarrollo se separó entre GitLab Comunity Edition, la cual siguió siendo Open Source bajo licencia MIT, y GitLab Enterprise Edition, privativo, y con funcionalidades exclusivas no disponibles en Comunity.

2.3 Sourcetree

Sourcetree [4] es una aplicación desarrollada por Attlasian que nos brinda una interfaz gráfica para una utilización mas cómoda de Git. Cuenta con diferentes elementos para realizar

todas las acciones de Git de una forma fácil y rápida, así como otras herramientas adicionales, como la integración de GitFlow.

2.4 Bootstrap

Bootstrap [5] es un framework para el diseño de interfaces de aplicaciones web. Contiene las herramientas necesarias para desarrollar un frontend en HTML, CSS y javascript con capacidad para adaptarse correctamente a diferentes dispositivos, sin necesidad de tener mucho conocimiento en tecnologías de desarrollo de interfaces web.

Diseñado en sus origines por Mark Otto y Jacob Thornton para su utilización en el diseño de la interfaz de Twitter, es ahora una herramienta open source desarrollada principalmente por **Bootstrap Team**

2.5 JQuery

JQuery [6] es una librería de JavaScript de software libre, desarrollada bajo la licencia de MIT y GNU y creada inicialmente por John Resig en el año 2006. Simplifica la manipulación de los elementos de HTML, el manejo de eventos y el uso de AJAX, con un API fácil de importar y utilizar.

2.6 Java

Java [7] es un lenguaje de programación comercializado por primera vez por Suns Microsystems en 1995. Es uno de los lenguajes mas utilizados.

Si bien es un lenguaje que es necesario compilar antes de su utilización, el resultado produce un Bytecode, que puede ser ejecutado sobre una Maquina Virtual de Java (JVM), Esto le permite ser un lenguaje ideal para el desarrollo multiplataforma, dado que el mismo bytecode podrá ejecutarse en cualquier plataforma con una implementación de la JVM disponible.

2.7 Spring Boot

Spring Boot [8] es un framework que permite facilitar el desarrollo de aplicaciones en java. Ofrece muchas facilidades para integrar diferentes elementos de Spring de una forma fácil y sin necesitar demasiada configuración por parte de los desarrolladores, pero permitiendo que igualmente se configure lo que se considere necesario.

De entre todos los componentes de Spring, en esta aplicación se ha utilizado la gestión de dependencias basado en anotaciones, Spring Security para gestionar la autenticación de

usuarios, Spring MVC para la gestión de las peticiones HTTP y Spring Data JPA para los elementos de la capa de acceso a datos.

2.8 IntelliJ IDEA

IntelliJ IDEA [9] es un IDE para el desarrollo de aplicaciones en Java, aunque también permite desarrollo para otro lenguajes. Esta desarrollado principalmente por la empresa JetBrains.

El IDE cuenta con una versión Comunity, siendo esta gratis y Open Source, y una versión Enterprise. En este proyecto se ha utilizado la versión Enterprise.

Cuenta con todo lo necesario para el desarrollo de aplicaciones en java.

2.9 Yed Graph Editor

Yed Graph Editor [10] es un editor de diagramas gratuito desarrollado por la empresa **yWorks**. Permite crear diferentes tipos de diagramas como de Entidad Relación o de casos de uso con elementos propios de dichos diagramas, e incluso modificar o crear nuevos tipos de diagrama o elementos para que se adapte mejor a las necesidades del proyecto.

Se ha usado en las fases iniciales para realizar el diseño de la aplicación.

2.10 PlantUML

PlantUML [11] es una herramienta de código abierto que nos permite crear diagramas UML a partir de un lenguaje simple de texto plano de forma automatizada. Se encarga de la distribución y dibujo de los elementos, intentando que queden claros y fáciles de comprender.

En este proyecto se ha utilizado para la creación de los diagramas finales para incluir en la documentación.

2.11 Taiga

Taiga [12] es una herramienta para la gestión de proyectos, centrada principalmente en el uso de metodologías de desarrollo ágiles como Scrum o Kanban.

La aplicación se desarrolla siguiendo un modelo open source, y es de uso gratuito, aunque limitado a un solo proyecto privado de hasta tres miembros.

En este proyecto se ha utilizado en la fase de desarrollo para realizar la gestión de los ciclos, y para registrar las incidencias.

Capítulo 3

Planificación, Diseño y Desarrollo

En este capítulo se muestra el proceso realizado para la realización del proyecto, detallando las fases de planificación, análisis, diseño y desarrollo, así como detalles de la metodología utilizada.

3.1 Estudio de Viabilidad

En este punto se expondrá la planificación, el análisis de los riesgos, así como el seguimiento y análisis de costes realizados durante el desarrollo del proyecto.

3.1.1 Planificación

La planificación es una fase muy importante a la hora de abordar un proyecto, nos permite de forma temprana hacer una descripción del proyecto, localizar posibles riesgos y los elementos de mayor importancia que el producto final ha de cumplir.

En este caso, se ha intentando hacer una planificación lo mas acertada posible.

El proyecto inicia su planificación entre mediados y finales del segundo cuatrimestre del curso 2018/2019, estableciendo como fecha final aproximada mediados de Noviembre para así presentarlo a la convocatoria de defensa de Diciembre de 2019, siendo el 30 de diciembre la fecha de finalización Real.

La planificación ha constado de la siguientes fases.

1. Investigación de las herramientas actuales y obtención posible funcionalidades.

2. Especificación de Requisitos y casos de uso.

3. Diseño inicial y elección de herramientas.

4. Desarrollo iterativo e incremental del producto. Cada iteración contara con una primera elección de los requisitos a implementar intentando mantener la misma cantidad de trabajo para cada iteración. Luego se desarrollara cada requisito atendiendo a:

   - Análisis y Diseño a bajo nivel del requisito.
   - Establecimiento de las tareas.
   - Implementación.
   - Pruebas y Validación.

5. Realización de memoria y conclusiones.

3.1.2 Estimación de Tiempo y Costes

Para la realización de este proyecto se ha contado con un trabajador a media jornada que realiza las labores de analista, diseñador y desarrollador en el proyecto. El director de proyecto hace la labor de jefe de proyecto. Los costes asignados a cada rol pueden verse en la tabla 3.1. El coste final estimado del proyecto es **8790€** 3.2

Rol	Coste por hora	Horas	Coste
Jefe de proyecto	45€	10 horas	450€
Analista	35€	52 horas	1820€
Diseñador	25€	88 horas	2200€
Desarrollador	20€	216 horas	4320€

Tabla 3.1: Tabla de costes de recursos

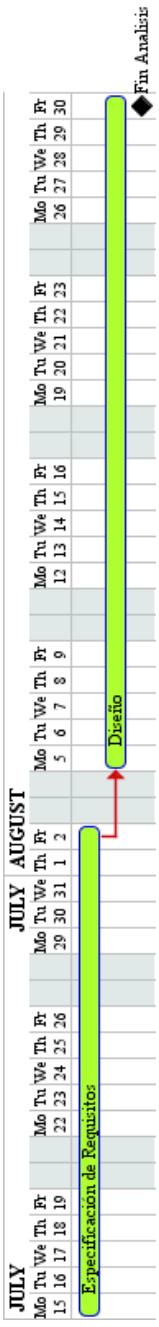
Inicio	Fin	Coste Total
15 de Julio de 2019	15 de Noviembre de 2019	8790€

Tabla 3.2: Estimación de costes del proyecto.

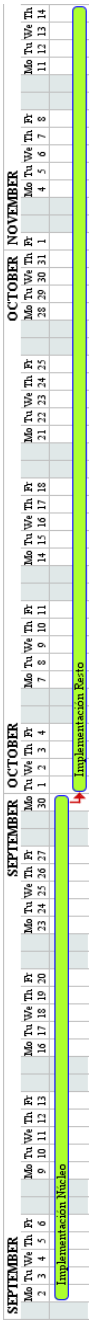
En la figura 3.1 podemos ver diagramas Gantt con la estimación de tiempo.

3.1.3 Seguimiento

Durante el desarrollo se ha seguido una planificación flexible, atendiendo primero a crear una aplicación prototipo con los elementos más importantes que permitiese hacer las labores de creación y seguimiento de proyectos y tares, y dejando para mas adelante el refinamiento de la aplicación para permitir realizar acciones mas complejas. Para el seguimiento se ha utilizado la aplicación web Taiga, que da la posibilidad de registrar tareas, crear ciclos de desarrollo para identificar las diferentes iteraciones y llevar una gestión visual gracias a la utilización



(a) Análisis de Requisitos y Diseño



(b) Implementación

Figura 3.1: Gantt Estimación Duración de proyecto.

de tableros Kanban. Con esta herramienta se pretendía realizar un registro y control de las iteraciones y los requisitos que se implementan en cada una, en lugar de un control del estado del proyecto en general, lo cual siempre se podía ver observando el numero de requisitos pendientes de implementar.

El proyecto se divide en 4 grandes fases: **obtención de funcionalidades**, **especificación de requisitos**, **diseño** y **desarrollo**.

En la **obtención de funcionalidades** se investigaron otras herramientas como Jira o Redmine para ver las funcionalidades que implementan, así como las formas que estas herramientas utilizan para representar e introducir datos. Con la experiencia que se obtuvo con el trabajo de estas herramientas se redactó una lista de posibles funcionalidades a implementar la cual se compartió con el director para su revisión y aprobación. Utilizando dicha lista una vez aprobada, se dio comienzo a la **especificación de requisitos**, identificando para cada requisito sus pasos generales, pasos excepcionales, los actores que participarían en cada caso y las relaciones entre los distintos requisitos, usando como ayuda diagramas de caso de uso. El análisis, redacción y revisión de los requisitos se realizó en la segunda mitad del mes de Julio de 2019.

Conforme a lo planificado, a principios de agosto se dio comienzo a al **diseño**, en donde, a partir de los elementos resultado de la fase anterior, se realizó un diseño a alto nivel, se definió la arquitectura a utilizar y se identificaron los elementos mas importantes que compondrían la aplicación mediante diagramas de entidad-relación y de clases. También se seleccionaron las herramientas y tecnologías que se utilizarían durante desarrollo. Esta fase se dio por finalizada el 24 de agosto, 7 días antes de lo planificado, dando comienzo al **desarrollo**. Este se intentó llevar a cabo en iteraciones o ciclos de una semana. En cada ciclo primero se establecen los requisitos a implementar, atendiendo a su importancia y al tiempo estimado para llevarlos a cabo, luego se realiza la implementación de cada caso de uso, y por último se realizan pruebas y validaciones finales. Así mismo, dentro del **desarrollo**, podemos agrupar las iteraciones en dos grandes fases:

1. **Desarrollo del Núcleo de la Aplicación**. Al comienzo de desarrollo se marcó como objetivo realizar los elementos necesarios para crear una aplicación con funcionalidad mínima, pero que cumpliese con los objetivos mínimos esperados de un gestor de proyectos. Estos requisitos fueron: creación de proyectos; creación de tareas en dichos proyectos; creación y login de usuarios y asignación de proyectos y tareas a usuarios. Esta parte se estimo para realizar en el primer mes de desarrollo, siendo este todo septiembre en la planificación. Finalmente se comenzó el 24 de agosto y terminó el 29 de septiembre.

2. **Desarrollo del resto de requisitos**. En esta fase, si bien se seguía teniendo en cuenta

para la planificación, dejo de ser tan prioritario la importancia del requisito y entro en juego el factor de la duración estimada. Esto no significa que se intentase hacer lo que se esperaba que fuese mas corto, si no que dependiendo de los acontecimientos ajenos al proyecto esperados para una semana, se planificaban los requisitos a realizar en dicho ciclo. Además en esta fase también se llevaron a cabo ciclos de refactorización, para mejorar el código de requisitos ya realizados, los cuales se llevaron a cabo en semanas en las que, debido a la dificultad en la previsión de acontecimientos ajenos al proyecto, era complicado establecer una planificación adecuada del proyecto que se pudiera cumplir en plazo. Al comienzo de esta fase se observó la dificultad de completar el proyecto para mediados de Noviembre, pensándose como nueva fecha de fin principios de diciembre. Finalmente, esta fecha tampoco se cumplió, completándose las funcionalidades establecidas en los requisitos el 20 de diciembre de 2019.

En este punto ya se ve una desviación en la estimación. Si bien el desarrollo se pudo iniciar antes de lo planificado, el proyecto termino más de un mes después del fin planificado (Tabla de estimación 3.2).

A continuación se comentan más en detalle ciclos realizados durante la fase de Desarrollo.

El primer ciclo se planificó del 24 al 31 de agosto, retrasándose finalmente un día para terminarse el 1 de septiembre. En el primer día se decidió comenzar con la administración básica de usuario, siendo la pila de tareas la siguiente: Creación de Usuarios (dos días), Autenticación de Usuarios (dos días), Ver perfil de usuario (un día), Modificar Contraseña (un día).

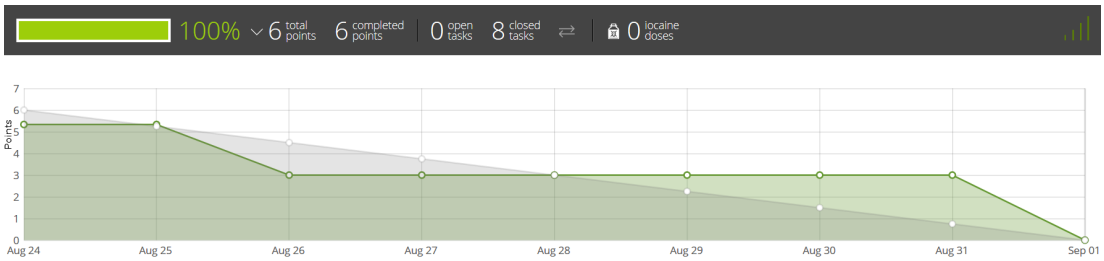


Figura 3.2: Gráfico Burndown del CICLO 1

En el gráfico 3.2 es posible observar como en este ciclo no se cumplió lo estimado, y tampoco se llevó el ritmo de trabajo esperado. Típico de las primeros ciclos en nuevos proyectos, sobretodo cuando en la organización no hay experiencia en el desarrollo y estimación de otros proyectos.

El segundo ciclo (Figura 3.3) se planificó del 7 de septiembre al 14 de septiembre. En este se completo la parte de gestión de usuarios y se comenzó la gestión de tareas. Los casos de uso que se decidió llevar a cabo en el ciclo fueron: Buscar Usuario (dos días), Crear Proyecto (un día), Buscar Proyecto (dos días), ver detalles de proyecto (un día).

En el gráfico 3.3 se aprecia que se ha seguido mejor la progresión esperada.

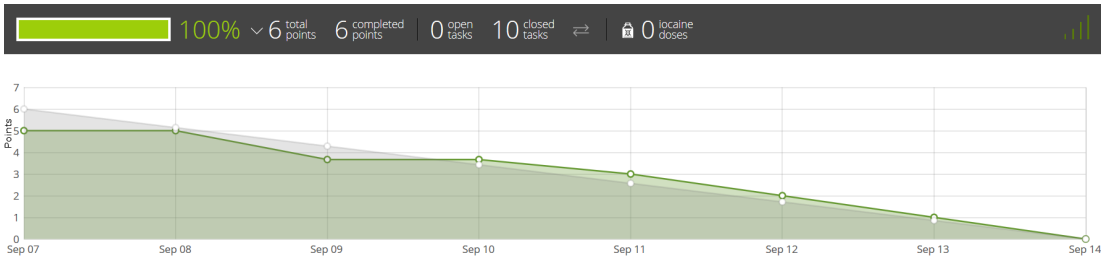


Figura 3.3: Gráfico Burndown del CICLO 2

El tercer ciclo (Figura 3.4) se planificó del 15 al 22 de septiembre. En este ciclo se completo la gestión básica de proyectos y se comenzó la gestión de tareas. Los casos de uso implementados fueron: Eliminar proyecto (un día), Editar proyecto (un día), Asignar proyectos (tres días), Crear tarea desde detalles del proyecto (un día y medio).

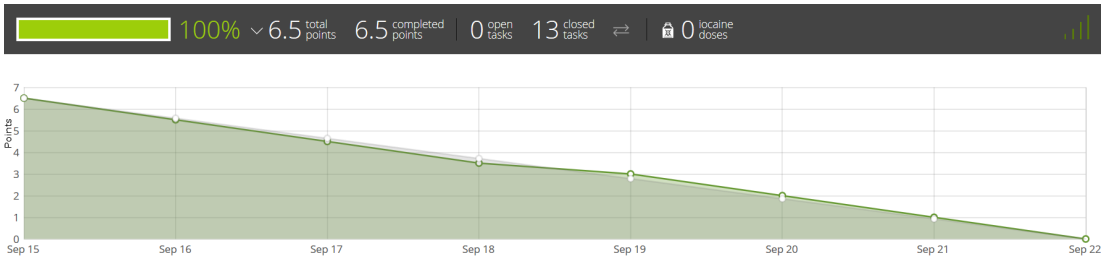


Figura 3.4: Gráfico Burndown del CICLO 3

El cuarto ciclo (Figura 3.5) se planificó del 22 al 29 de septiembre. En este ciclo se completo la gestión de tareas. Para realizar ciclos de la misma duración, se añadieron otros casos de uso que no se consideraron parte del mínimo de funcionalidades. Los casos de uso implementados fueron: Crear tareas desde Cualquier punto de la aplicación (un día), Ver Detalles de Tarea (un día), editar tarea (un día), Buscar Tareas (un día), Comentar Tarea (un día), Añadir Observador (un día).

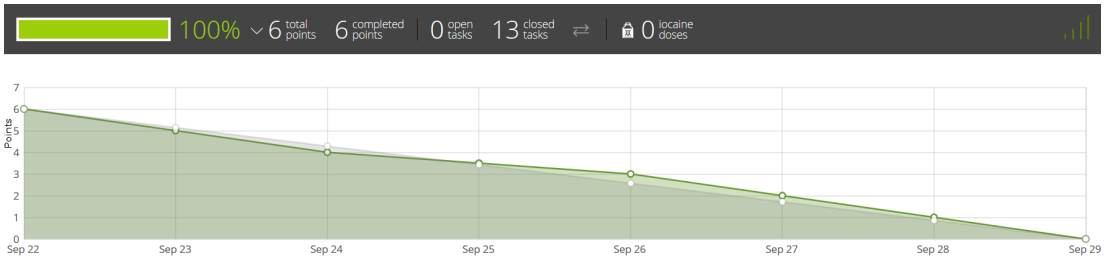


Figura 3.5: Gráfico Burndown del CICLO 4

A partir de este punto se dio por finalizada la primera fase del desarrollo, con una duración

total de 37 días, comenzando el 24 de agosto y finalizando el 29 de septiembre de 2019 y contando con un total de 4 ciclos.

El ciclo 5 (Figura 3.6) se planificó del 30 de septiembre al 6 de octubre. En este se comienzan las funcionalidades añadidas para completar el proyecto, siendo estas las siguientes: Registrar tiempo dedicado (un día), Ver tiempos registrados (dos días), Ver observadores (medio día), Eliminar observador (medio día), Crear ciclo de desarrollo (un día).

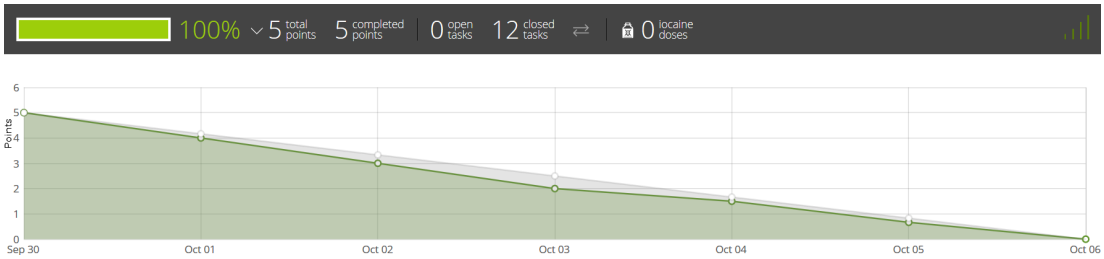


Figura 3.6: Gráfico Burndown del CICLO 5

El ciclo 6 (Figura 3.7) se planificó del 14 al 20 de octubre. En este ciclo se llevaron a cabo: Buscar ciclo de desarrollo (un día), Asignar tarea a un ciclo de desarrollo (un día), Añadir ciclo de desarrollo a la búsqueda de tareas (medio día), Ver detalles de ciclo de desarrollo (medio día), Crear sistema de permisos para proyectos (dos días). Con la última tarea comienza el modelado para hacer un sistema mas complejo de gestión de usuarios, basado en permisos.

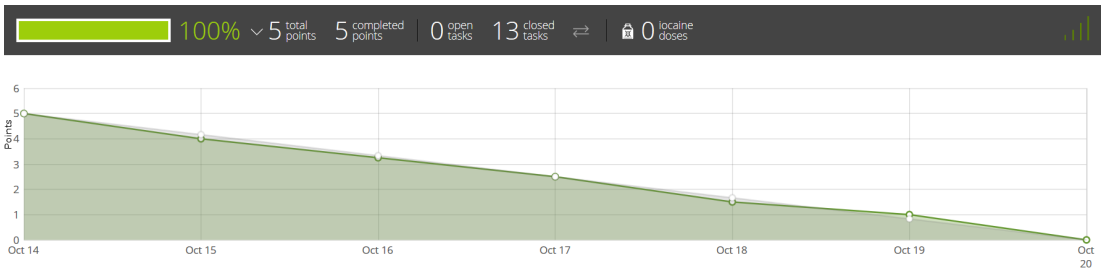


Figura 3.7: Gráfico Burndown del CICLO 6

El ciclo 7 (Figura 3.8) se planificó del 21 al 28 de octubre. Se completo el desarrollo del sistema de permisos de usuarios, y se comenzó a realizar integraciones del sistema para conectarlo con GitLab para asociar repositorios a proyectos y obtener información de los mismos. Lo implementado en este ciclo fue: Añadir permisos y niveles de permiso de acceso (un día), Modificar los permisos de un Usuario (dos días), Asignar un Repositorio GitLab a un proyecto (un día) y Buscar Branches de un repositorio gitlab (dos días).

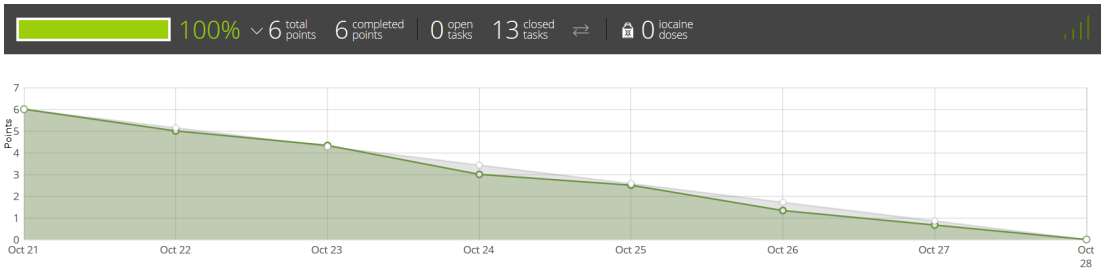


Figura 3.8: Gráfico Burndown del CICLO 7

El ciclo 8 (Figura 3.9) se planificó del 11 al 17 de noviembre. En este ciclo se incorporaron cambios para mejorar el control del flujo de elementos de un proyecto, así como que se continuó con la integración de GitLab. Las tareas implementadas en este ciclo fueron: Buscar Repositorio en GitLab (un día), Crear estado de Tareas (un día), Añadir Tareas relacionadas (un día), Buscar commits realizados asociados a una tarea (dos días).

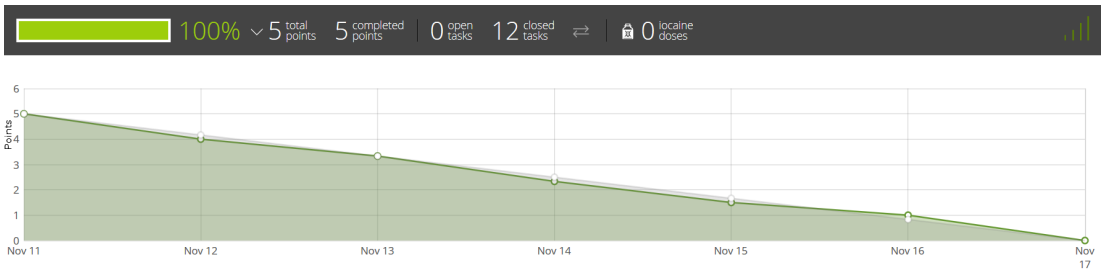


Figura 3.9: Gráfico Burndown del CICLO 8

El ciclo 9 (Figura 3.10) se planificó del 18 al 24 de noviembre. En este ciclo se continua con la integración con GitLab, así como mas mejoras. Lo implementado en este ciclo fue: Mostrar Tareas Relacionadas (un día), Asociar branch a Ciclo de desarrollo (un día), Asignar Repositorio a un proyecto usando la búsqueda de repositorios (un día), Buscar commits asociados a un ciclo de desarrollo (un día), Crear una página resumen como página inicial.

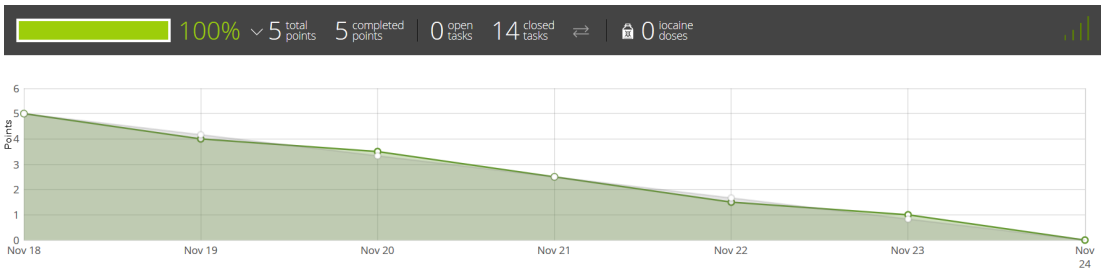


Figura 3.10: Gráfico Burndown del CICLO 9

Finalmente el ciclo 10 (Figura 3.11) se llevó a cabo del 2 al 8 de diciembre. En este ciclo se crea un panel de control para facilitar el proceso de administración, así como mejoras en la

búsquedas. En concreto, lo implementado en este ciclo fue: Panel de control de administración (dos días), Exclusiones en la búsqueda de Usuarios (un día), Exclusiones en la búsqueda de tareas (un día), exclusiones en la búsqueda de proyectos (un día).

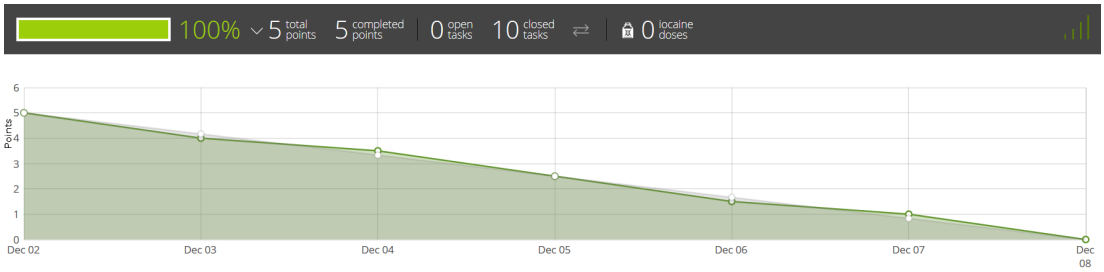


Figura 3.11: Gráfico Burndown del CICLO 10

Fuera de ciclos se implementó Añadir adjuntos a una Tarea. La razón de ello fue que no se consideró oportuno crear un ciclo solo para la implementación de dicha tarea.

3.2 Costes Finales del Proyecto

Viendo lo descrito en la tabla de costes de recursos 3.1, podemos hacer una aproximación a los costes reales del proyecto.

Las primeras fases sucedieron de forma similar a lo planificado. El trabajo del analista cumplió con el medio mes establecido en planificación y siendo su coste final el siguiente: **1820€**.

Para el caso de la fase de diseño se planificó que duraría todo el mes de agosto, sin embargo se dio por terminado el diseño el día 24, estando ya listo para ser implementado. Su coste final fue **1800€**.

Por último el desarrollo, siendo un desarrollo de ciclos de 7 días en los cuales se trabajo a media jornada, y teniendo 10 ciclos establecer el coste del mismo se hace de forma fácil, siendo este **5600€**.

Con esto podemos determinar que el coste final del proyecto, teniendo en cuenta que la parte del jefe de proyecto permanece sin cambios, fue de **9670€**, **880** por encima, siendo esto aproximadamente un **10%** mas caro de lo estimado. Podemos determinar además que donde mas se sufrió un desvió tanto en planificación de tiempo como de coste fue en el desarrollo.

3.3 Metodología

Para la realización de este proyecto se ha optado por utilizar la metodología Proceso Unificado de Desarrollo (PUDS) [13], aunque , como iremos viendo en este apartado, no se ha

aplicado siguiendo todos los elementos de dicha metodología, si no que se ha adaptado y escogido los elementos que se han visto oportunos para un desarrollo realizado solo por una persona, combinándolos con técnicas de otras metodologías, como Scrum o Kanban.

PUDS es una metodología y un marco de trabajo iterativo e incremental. Para PUDS un incremento es cualquier parte del software que se complete en cada iteración, pudiendo ser eso software funcionando como documentación, siendo típico de proyectos que emplean PUDS que las primeras iteraciones los incrementos sean principalmente Especificación de requisitos, diagramas de casos de uso y demás documentación que sirvan para describir y definir el proyecto. Esto lo diferencia de metodologías ágiles incrementales como Scrum que consideran un incremento como un producto software entregable.

En PUDS se divide principalmente en cuatro fases: Inicio, Elaboración, Construcción y Transición. Estas fases, a su vez pueden ser divididas en diferentes iteraciones

En el inicio es donde se define el producto, las metas que debe cumplir, análisis de viabilidad y plazos. En en el caso de nuestro proyecto, esta fase coincidiría con la primera de nuestro proyecto, la **Obtención de posibles funcionalidades**.

En la elaboración se define a mas bajo nivel el proyecto, se establecen los casos de uso a realizar y se empieza a crear el núcleo de la aplicación. En este proyecto esta fase correspondería con la **Especificación de Requisitos** y con la primera etapa del **Desarrollo e Implementación**, en la que se implementó los factores mas importantes que definen a la aplicación.

En la Construcción es donde se implementa el resto del producto ajeno al núcleo, incluyendo los requisitos menos importantes. Al final de esta fase se obtiene el producto completo. En el caso de nuestro proyecto esto correspondería a la segunda etapa explicada en **Desarrollo e Implementación**.

Y por ultimo, en la transición es donde el producto debe estar listo para ser probado e instalado por el cliente, es decir, sería la fase en la cual podríamos dar por finalizado el producto. En el caso de nuestro proyecto, nos encontraríamos en esta fase, realizando la documentación final de entrega del producto, entre los que se encontraría esta memoria.

Como se comentó al comienzo de esta sección, también se tomaron elementos de otras metodologías como Scrum o Kanban, principalmente en la fase de Elaboración y Construcción, para ayudar a gestionar la implementación.

De Kanban, principalmente se utilizó uno de sus prácticas mas reconocidas, el uso de un tablero en el cual hacer más visible el estado del proyecto.

De Scrum se aplicó la forma de trabajar durante el desarrollo. Si bien las iteraciones también forman parte de PUDS, las realizadas durante el desarrollo tenían cierto parecido a lo conocido como Sprints en Scrum, excluyendo los elementos de reuniones realizados como el daily meeting o la reunión de revisión del sprint. En concreto la forma utilizada de trabajar

es, elegir los elementos de la pila de producto a realizar en el sprint y al comienzo del mismo hacer una subdivisión de los mismos en tareas.

3.4 Análisis

En esta sección trataremos todo lo relacionado con la obtención, análisis y especificación de requisitos.

3.4.1 Análisis de Requisitos

La fase de análisis de requisitos comenzó con una reunión en la cual se redactaron posibles funcionalidades a implementar. Los requisitos funcionales explicitados en dicha reunión fueron:

- **Gestión de Usuarios**. Esto incluye la creación y modificación de usuarios, el proceso de autenticación, la administración de permisos de acceso a proyectos y la búsqueda de usuarios.

- **Gestión de Proyectos**. Esto incluye la creación, búsqueda, modificación y eliminación de proyectos.

- **Gestión de Tareas**. Esto incluiría la creación, búsqueda y modificación de tareas. También incluiría todas las operaciones a realizar relacionadas con tareas, como su asignación a un usuario, agruparlas en ciclos de desarrollo,visualizar su progreso, añadir comentarios y adjuntar archivos.

- **Gestión del tiempo dedicado**. Esto incluirá el propio registro del tiempo dedicado en una tarea, así como la búsqueda de tiempos dedicados, tanto en una sola tarea como en un proyecto.

- **Integración con GitLab**. Esto incluye buscar commits asociados a una tarea o proyecto y asignar una branch a un ciclo de desarrollo.

Además también se consideró añadir otras funcionalidades, como la representación de estadísticas o métricas útiles para un proyecto, tarea o ciclo de desarrollo, como por ejemplo la implementación de diagramas Gant o gráficos Burndown, pero no se vieron necesarias, es decir, son funcionalidades que no son criticas para el proyecto, aunque podrían añadirse a mayores una vez el proyecto estuviese completado.

3.4.2 Casos de Uso

Una vez establecidos los requisitos a implementar, se procedió a la extracción de los casos de Uso.

Los casos de uso se agruparon atendiendo al subsistema o entidades a las que hacen referencia. Si bien están numerados, el orden de los mismos no denota ninguna clase de prioridad o importancia.

Gestión de Usuarios

- **CU-01 Autenticación de Usuarios:** El sistema permitirá a los usuarios registrados en la aplicación autenticarse para obtener acceso. *Actores:* Usuario .
- **CU-02 Crear Usuario:** Los administradores podrán crear nuevos usuarios. *Actores:* Administrador .
- **CU-03 Ver perfil de usuario:** Los usuarios podrán acceder a una página con los detalles de un usuario, o de uno mismo. *Actores:* Usuario.
- **CU-04 Cambiar Contraseña:** Los usuarios podrán cambiar su contraseña, accediendo antes a su perfil. *Actores:* Usuario.
- **CU-05 Buscar Usuario:** Los usuarios podrán realizar búsquedas de otros usuarios, pudiendo utilizar como filtro el nombre y apellidos, nombre de usuario o proyecto asignado. *Actores:* Usuario.
- **CU-06 Modificar Permisos de usuario:** Los administradores podrán modificar los permisos de acceso a un usuario. Esto es, modificar las acciones que un usuario puede hacer en los diferentes proyectos que tiene la aplicación registrados. *Actores:* Administrador.

Gestión de Tareas

- **CU-07 Crear Tarea:** Los usuarios podrán crear tareas en aquellos proyectos en los que tengan permiso de escritura. *Actores:* Usuario.
- **CU-08 Ver Tarea:** Los usuarios podrán ver los detalles de las tareas pertenecientes a los proyectos en los cuales tengan permiso de lectura. *Actores:* Usuario.
- **CU-09 Editar Tarea:** Los usuarios podrán editar los detalles de una tarea accediendo a través de **Ver Tarea**, en aquellas tareas donde tengan permiso. *Actores:* Usuario.
- **CU-10 Buscar Tareas:** Los usuarios podrán buscar tareas pertenecientes a proyectos en los que tengan acceso, pudiendo filtrar por nombre o identificador de Tareas, pro-

yecto, usuario responsable de hacer la tarea, estado de la tarea, y fecha de inicio y fin. *Actores:* Usuario.

- **CU-11 Añadir Adjunto a una Tarea:** Los usuarios podrán añadir adjuntos en aquellas tareas que estén asociadas a proyectos para los cuales tengan permisos. *Actores:* Usuario.
- **CU-12 Comentar Tarea:** Los usuarios podrán añadir comentarios en las tareas asociadas a aquellos proyectos en los que tenga permisos. *Actores:* Usuario.
- **CU-13 Ver Cambios de Una Tarea:** Los usuarios podrán ver los cambios de las tareas asociadas a proyectos en los que tengan permisos. *Actores:* Usuario.
- **CU-14 Añadir Observador:** Los usuarios pueden añadir observadores a las tareas de los proyectos en los cuales tengan permisos. *Actores:* Usuario.
- **CU-15 Ver Observadores:** Los usuarios podrán ver los observadores en aquellas tareas asociadas a proyectos en los que tengan permisos. *Actores:* Usuario .
- **CU-16 Eliminar Observadores:** Los usuarios podrán eliminar observadores en aquellas tareas asociadas a proyectos en los que tengan permisos. *Actores:* Usuario.
- **CU-17 Copiar Tareas:** Los administradores podrán copiar tareas de un proyecto a otro. *Actores:* Administrador.

Gestión de Proyectos

- **CU-18 Crear Proyecto:** Los administradores podrán crear nuevos proyectos. *Actores:* Administrador.
- **CU-19 Buscar Proyecto:** El sistema permitirá a os usuarios buscar proyectos, usando para filtrar el nombre o identificador del proyecto, el líder de proyecto, el estado del mismo y las fechas de inicio y fin. *Actores:* Usuario.
- **CU-20 Ver Detalles de Proyecto:** Los usuarios podrán ver los detalles de un proyecto para el cual tengan permisos. *Actores:* Usuarios.
- **CU-21 Editar Proyecto:** Los administradores podrán editar los detalles de un proyecto, accediendo a través de **Ver detalles de Proyecto**. *Actores:* Administrador.
- **CU-22 Eliminar Proyecto:** Los administradores podrán eliminar proyectos. *Actores:* Administrador.
- **CU-23 Asignar Proyectos:** Los administradores podrán asignar proyectos a usuarios, dándoles permisos de lectura y escritura. *Actores:* Administrador.

Gestión y registro de Horas dedicadas

- **CU-24 Ver Tiempos Registrados:** Los usuarios podrán ver los tiempos que hayan registrado. permisos.*Actores:* Usuario.
- **CU-25 Ver Tiempos Registrados en Proyecto:** Los responsables de proyecto podrán ver los tiempos que han registrado los usuarios con permisos en los proyectos de los que se encarga.*Actores:* Responsable de Proyecto.
- **CU-26 Registrar Tiempo:** Los usuarios pueden registrar el tiempo que dedican a realizar las tareas.*Actores:* Usuario.

Gestión de Ciclos de Desarrollo

- **CU-27 Crear Ciclo de Desarrollo:** Los usuarios podrán crear ciclos de desarrollo con los cuales agrupar las tareas de un proyecto concreto, siempre que tenga permisos. *Actores:* Usuario.
- **CU-28 Buscar Ciclos de Desarrollo:** Los usuarios podrán buscar los ciclos de desarrollo de un proyecto en el cual tienen permisos. *Actores:* Usuario.
- **CU-29 Ver Detalles de Ciclo de Desarrollo:** Los usuarios podrán ver los detalles de un ciclo de desarrollo. *Actores:* Usuario.

Conexión con GitLab

- **CU-30 Buscar Commits:** Los usuarios podrán buscar los Commits de Proyectos o Tareas. *Actores:* Usuario.
- **CU-31 Buscar Branchs:** Los usuarios podrán buscar las ramas de un repositorio. *Actores:* Usuario.
- **CU-32 Buscar Repositorios:** Los administradores podrán buscar los repositorios de una cuenta de GitLab. *Actores:* Administrador.
- **CU-33 Asignar Repositorio a un Proyecto:** La aplicación permitirá asignar un repositorio a un proyecto si la aplicación esta asociada a una cuenta de GitLab. *Actores:* Administrador.
- **CU-34 Asignar Branch a Ciclo de Desarrollo:** Los usuarios podrán asociar a un ciclo una branch siempre y cuando el proyecto tiene un repositorio asociado. *Actores:* Usuarios.

De entre estos casos de uso, finalmente se descartaron los casos de uso **Ver Cambios de Una Tarea** y **Copiar Tareas**. La razón es que son requisitos que se extrapolaron de los requerimientos como posibles ayudas a la gestión de proyectos, pero que realmente nunca

se especificaron como una necesidad. Igualmente esto no implica que se descarten de forma definitiva, pudiendo ser implementados como mejoras futuras.

En la figura 3.12 se muestra el diagrama completo de casos de uso.

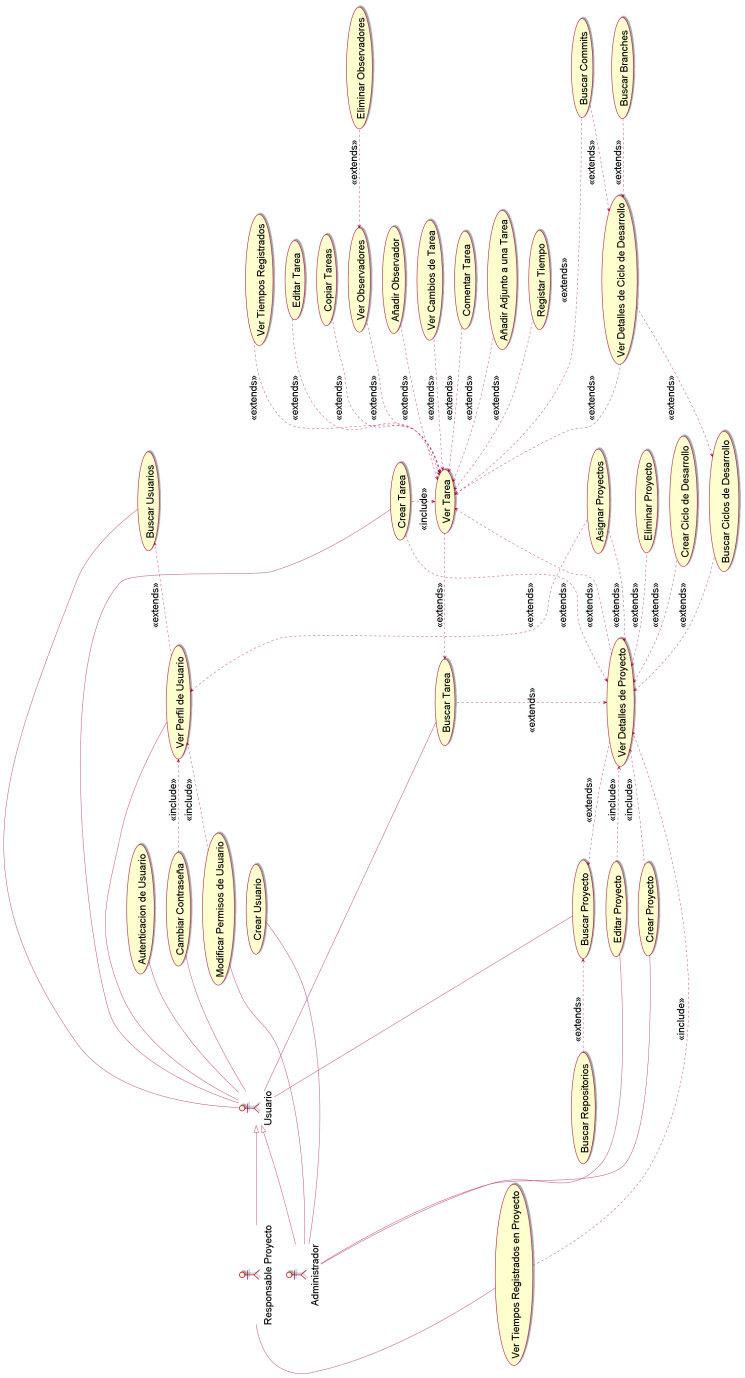


Figura 3.12: Diagrama completo de Casos de Uso

3.5 Arquitectura y Diseño

En esta sección se tratará la elección de la arquitectura de la aplicación y el diseño previo a la implementación.

3.5.1 Arquitectura

Para el desarrollo de ***GesTa*** se escogió como arquitectura Modelo-Vista-Controlador (MVC) sobre una de cliente servidor.

Esta arquitectura esta conformada por tres componentes con acciones o responsabilidades bien diferenciadas, separando de esta forma el modelo del dominio de la aplicación, la forma en la que dicho modelo se le muestra al usuario y como el usuario interactúa con ello. Con esto se aísla dichas funcionalidades facilitando el entendimiento y la modificación de cada componente por separado.

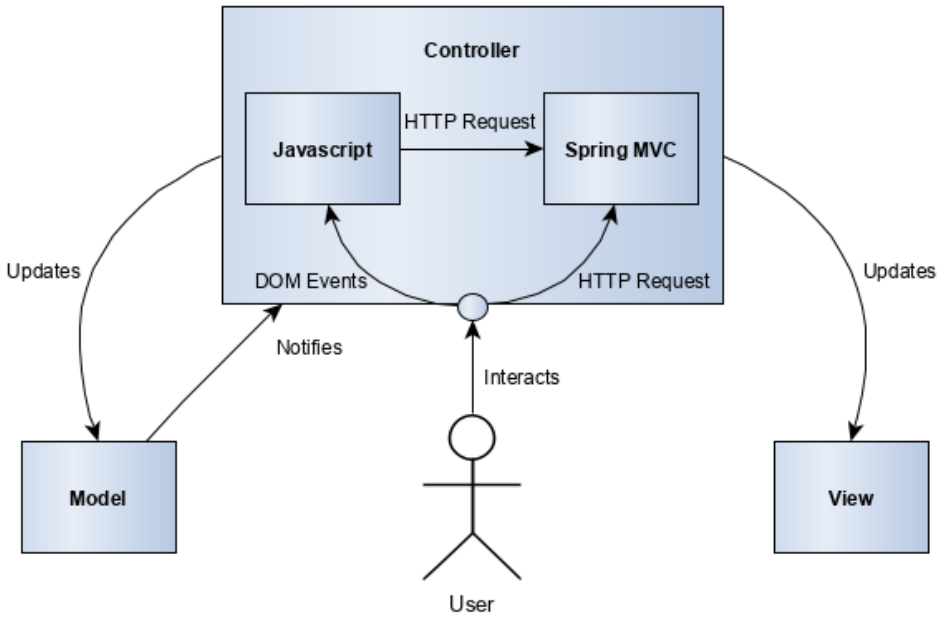


Figura 3.13: Arquitectura MVC en ***GesTa***

Los componentes de MVC en ***GesTa*** son los siguientes:

- **Modelo**

  El modelo es la capa de acceso a datos, compuesta por clases que representan las Entidades de la base de datos, así como DAOs para recuperar, crear o modificar los datos. El modelo contendrá una serie de servicios, los cuales serán los encargados de interactuar con los DAOs, y exponer la lógica de negocio para su uso por los otros componentes. En

la implementación, se ha usado **Hibernate**, **JPA** y **Spring Data JPA** para la descripción de Entidades y para la declaración y creación de los DAO.

- **Vista**

  La vista esta compuesta por un conjunto de páginas HTML las cuales sirven para mostrar los datos del modelo, o para mostrar los elementos necesarios para la modificación de los datos en el navegador del cliente. Para el desarrollo de la vista se usó **Thymeleaf** para crear templates, las cuales, en tiempo de ejecución, pasarán por un preprocesado para cargar contenido de forma dinámica antes de enviarlas al cliente.

- **Controlador**

  El controlador contiene los componentes que recibirán y responderán a los eventos desatados por el usuario, comunicándose con el **Modelo** para interactuar con los datos, y modificando la **Vista** de ser necesario. En este proyecto, el **Controlador** estará compuesto de clases marcadas con la anotación **@Controller** de **Spring MVC**, las cuales responderán a peticiones HTTP ejecutadas por el cliente al interactuar con elementos de la vista, y **Javascript** y **JQuery** para gestionar eventos en el navegador del usuario.

3.5.2 Diseño

Una vez se especificaron los requisitos y la arquitectura a utilizar, se pasó a realizar el diseño preliminar de la aplicación.

Se comenzó definiendo el modelo de datos de la aplicación a través de la creación de un diagrama de Entidad-Relación (figura 3.14), que identifica cada Entidad, sus atributos y relaciones.

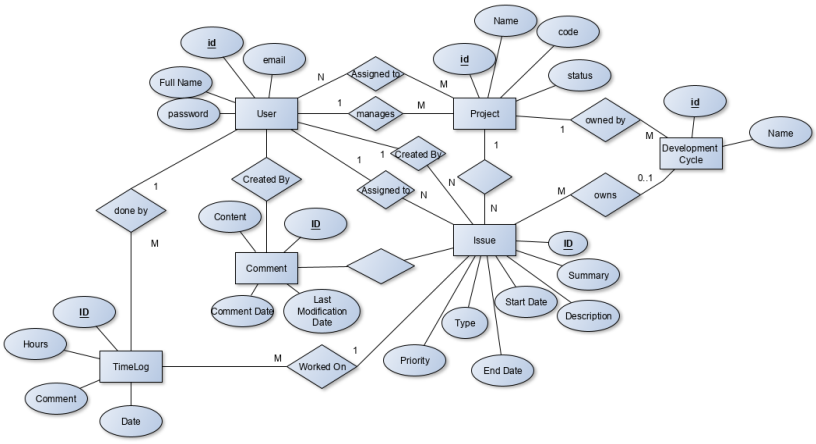


Figura 3.14: Diagrama Entidad-Relación

También se realizó un diagrama de clases de las entidades (Figura 3.15), en el cual además de mostrar relaciones y atributos, también se entra mas en detalle de los tipos de datos que estos últimos tendrán en la aplicación.

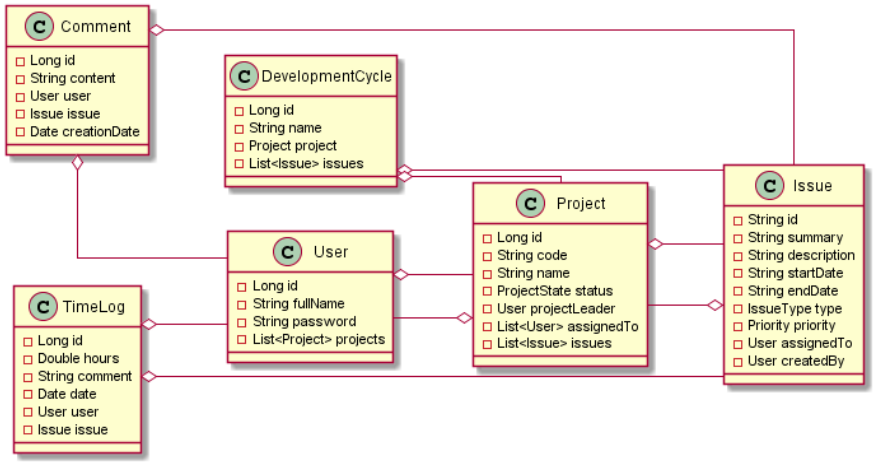


Figura 3.15: Diagrama de clase de Entidades

Finalizado este punto, las entidades planificadas para desarrollar fueron:

- **Issue** para representar las Tareas con los siguientes atributos: ID de tarea, nombre de tarea, descripción, fecha inicio, fecha fin, tipo de tarea, prioridad, usuario responsable y usuario que crea la tarea.
- **Project** para representar los proyectos con los siguientes atributos: ID de proyecto, Código de proyecto, nombre, estado, líder de proyecto, usuarios miembros del proyecto y lista de tareas.
- **User** para representar a los usuarios con los siguientes atributos: ID de usuario, nombre completo, email y contraseña.
- **DevelopmentCycle** para representar los ciclos de desarrollo con los siguientes atributos: ID del ciclo, nombre, proyecto al que pertenece y lista de tareas.
- **Comment** para representar los comentarios de una tarea con los siguientes atributos: ID de comentario, contenido, usuario que crea el comentario, tarea a la que se añade y fecha de creación.
- **TimeLog** para representar los tiempos registrados en una tarea con los siguientes atributos: ID de registro, horas dedicadas, comentario, fecha, usuario que realiza el registro y tarea.

Para la comunicación entre la aplicación java y la base de datos se decidió utilizar el patrón **DAO**. Este se basa en la creación de componentes que definirán una API abstracta para separar

la BD y la implementación concreta de acceso a datos de los otros elementos de la aplicación. Esto permite aislarlos y que cualquier otra capa de la aplicación evolucione por su cuenta sin la necesidad de conocer los detalles del modelo de datos. En nuestro caso, se decidió crear una interfaz DAO para cada una de las entidades definidas anteriormente, y usar **Spring Data JPA** el cual genera de forma automatizada una implementación en el arranque de la aplicación, basándose principalmente en el nombre de los métodos declarados. De esta forma nos evita tener que realizar una implementación para diferentes bases de datos, delegando en Spring para que se generen los elementos necesarios para un gestor concreto. Las interfaces que se definieron en este punto son:

- **UserRepository**
- **ProjectRepository**
- **IssueRepository**
- **DevelopmentCycleRepository**
- **CommentRepository**
- **TimeLogRepository**

En la Figura 3.16 se puede apreciar los DAO creados en ***GesTa*** y las entidades a las cuales están relacionados.

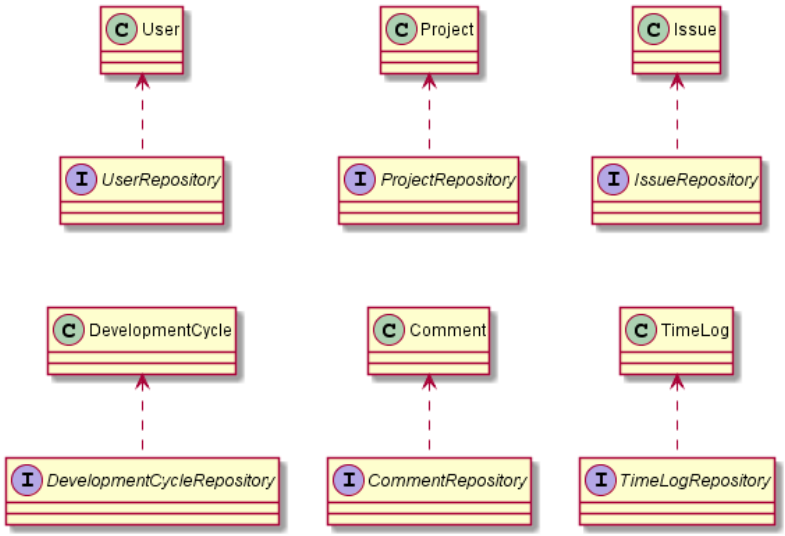


Figura 3.16: Diagrama de clase de DAOs

Para exponer el modelo se decidió utilizar el patrón **Fachada**. Este se basa principalmente en ocultar la complejidad de un conjunto de subsistemas mediante la implementación de una interfaz, la cual se encargará de interactuar con los diferentes elementos que sean necesarios

para completar una operación. En nuestro caso, estas fachadas, , conocidas como Servicios, implementarán la lógica de negocio, se comunicaran con la BD a través de los DAO y realizaran las comprobaciones necesarias para cada operación. Además y servirán como una interfaz simple contra la que el resto de elementos de la arquitectura MVC pueden interactuar para responder de forma adecuada a las acciones de los usuarios.

Se decidió crear servicios que agrupen funcionalidades relacionadas con elementos comunes, en lugar de crear un servicio por entidad. Para ello se pensó en mantener la división en grupos que se hizo cuando se especificaron los casos de uso y que podemos ver en la sección 3.4.2, a excepción del subgrupo de Ciclos de Desarrollo cuyas funcionalidades están muy ligadas a otras entidades y por lo tanto se incluirían en otros servicios. Los servicios a implementar, con nombres provisionales, fueron: **GitLabService**, **IssueService**, **ProjectService**, **TimeLogService** y **UserService**. También se pensó en sus posibles métodos, basándose para ello en las necesidades que exponen los casos de uso, completando así la definición de los elementos que componen el **Modelo**. En la figura 3.17 podemos ver los servicios y su relación con los DAO.

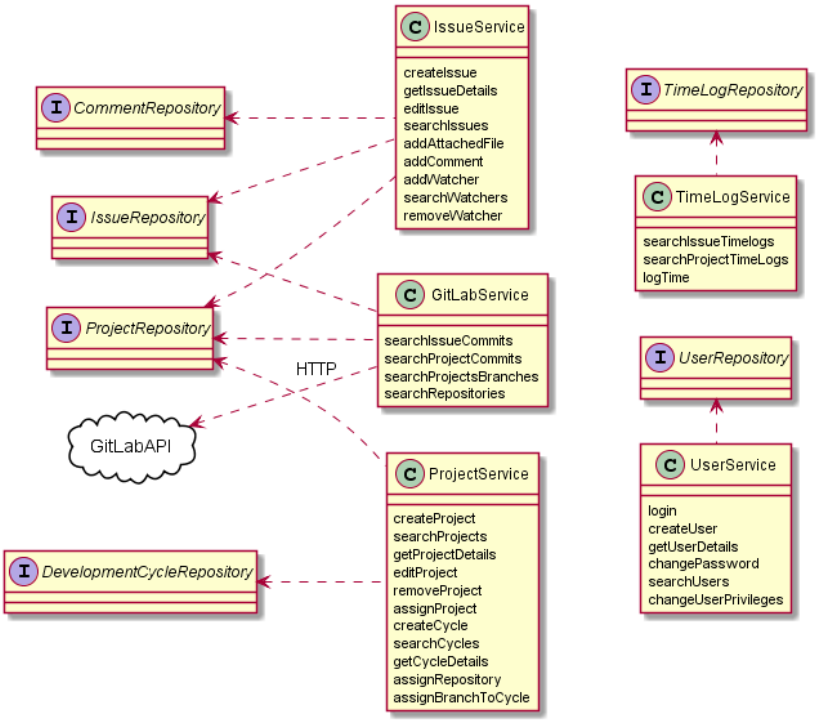


Figura 3.17: Diagrama de clases de los servicios definidos y relación con los DAO

Por último se definió y decidió los elementos y la secuencia de acciones que comprenderían los elementos del **Controlador** y **Vista**.

El controlador es el encargado de recibir acciones del cliente, ya sean estas peticiones

HTTP o eventos DOM, y responder a estas con el contenido adecuado. Para ello, en los casos en los que sea necesario, interactuará con el modelo a través de los servicios y responderá cambiando los elementos en la vista. En este caso, se pensó en utilizar los componentes **@Controller** de **Spring MVC** para responder ante petición HTTP, y javascript para responder directamente en el navegador del cliente a eventos DOM. También podríamos incluir en el controlador la librería que preprocesa las páginas de Thymeleaf antes de enviárselas al cliente. En este punto del diseño no se estableció con exactitud las clases a implementar ni lo que se desarrollaría en javascript, dejando estos detalles para el diseño a bajo nivel realizado durante el desarrollo, pero si se decidió que se intentaría crear una clase **@Controller** por cada caso de uso, en lugar de agruparlos en pocos controladores, al contrario de como se hizo en el caso de los servicios del modelo. Aunque de esta forma se aumentaría de forma significativa el número de clases, también se cumpliría con el **Principio de Responsabilidad Única** al hacer que una clase solo responda ante las peticiones relacionadas con un caso de uso.

Por último, se concretó que los elementos de la vista a implementar serían templates de Thymeleaf combinadas con Bootstrap, lo que permitirá crear páginas con un estilo adecuado y funcionales de forma fácil. Al igual que con los controladores, tampoco se definió en este punto lo que se implementaría en concreto, aunque en principio sería, como mínimo, una vista por caso de uso o controlador.

Una vez definidos los elementos que compondrían cada componente de MVC, se pudo dar por finalizado el diseño, estableciendo como se comunicarían entre sí los componentes de la arquitectura. Esto se hizo de forma genérica realizando el diagrama de clase de la figura 3.18 y el diagrama de flujo 3.19, y a bajo nivel en diagramas de secuencia como el de la figura 3.20, los cuales fueron corregidos y refinados durante el desarrollo.

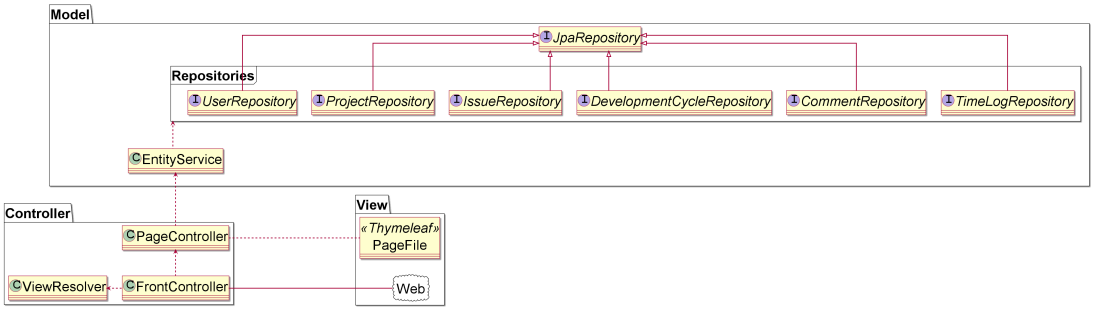


Figura 3.18: Diagrama de clase de los componentes MVC

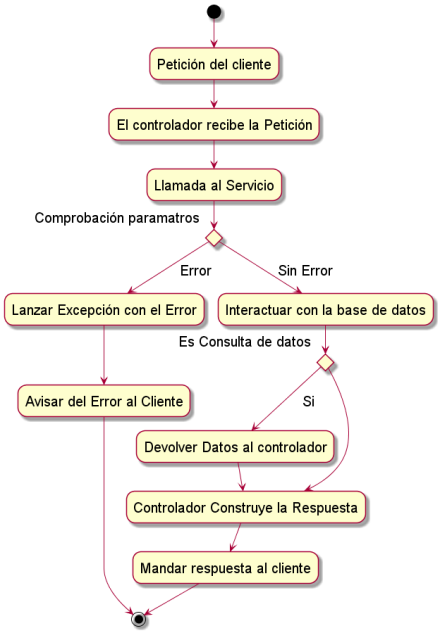


Figura 3.19: Diagrama de flujo general de la aplicación

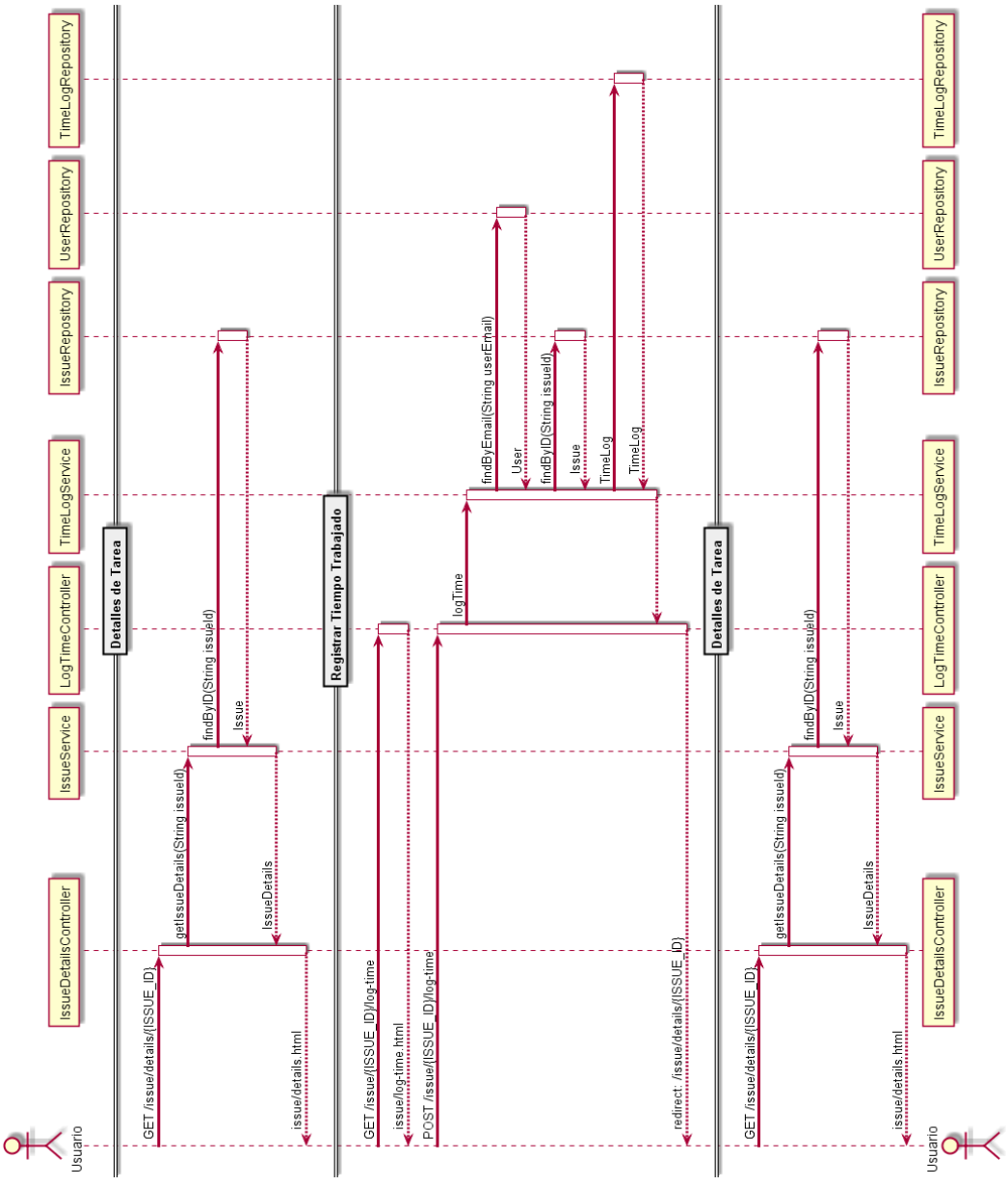


Figura 3.20: Diagrama de Secuencia del caso de uso **Registrar Tiempo**

3.6 Implementación

En esta sección se detallará como se realizó la implementación, llegando mas a bajo nivel de lo presentado hasta ahora.

Debido a que en la fase de desarrollo se siguieron ciertos principios de Scrum como hacer iteraciones siendo el resultado de estas productos entregables con funcionalidad, este no siguió el mismo orden de implementación que el orden de las capas explicadas en el modelo de arquitectura ni el mismo orden que se siguió en el diseño, es decir, no se desarrollo una capa de MVC de forma completa antes que ninguna otra, si no que se implementó de forma completa todo el flujo correspondiente a una funcionalidad. Si una funcionalidad es muy compleja, es posible que se divida y se implemente en varias tareas o ciclos.

Para una funcionalidad completa normalmente se siguen los siguientes pasos:

- **Modelo**
  - Implementación o Modificación de Entidades.
  - Implementación o Modificación de DAO.
  - Implementación o Modificación de Servicios del modelo.
- **Controlador y Vista**
  - Implementación o modificación del Controlador.
  - Creación o modificación de página HTML.

Para desarrollar una funcionalidad se intento seguir ese orden de implementación. Cabe destacar que no todos los desarrollos requieren realizar todos los pasos, habiendo casos en los que lo implementado para otra funcionalidad se puede utilizar para la que se este implementando, o por ejemplo cuando se esta desarrollando solo parte de una funcionalidad que se dividió por complejidad.

A continuación se comentará con mas detalle la implementación y el desarrollo realizado en cada una de las capas.

3.6.1 Desarrollo del Modelo

Lo primero es crear la entidad. Para ello se crea una clase Java marcada con la anotación **@Entity**. En esta entidad se añaden los atributos que se identificaron al realizar el diagrama Entidad-Relación en las primeras fases del proyecto. Dependiendo del tipo de atributo, será necesario marcarlo con anotaciones propias de Hibernate y JPA. Las más usadas en el proyecto son las siguientes:

- **@Id** para identificar la clave primaria.

- **@GeneratedValue** para los casos en los que la clave primaria sea generada automáticamente al persistir la entidad.
- **@OneToOne** Para atributos que representan el tipo de relación uno a uno (1:1).
- **@OneToMany** Para atributos que representan el tipo de relación uno a muchos (1:N)
- **@ManyToMany** Para atributos que representan el tipo de relación muchos a muchos (N:M).
- **@ManyToOne** Para atributos que representan el tipo de relación muchos a uno (N:1)
- **@JoinColumn** Es usado para especificar la columna que servirá para establecer la relación en base de datos. Acompaña a las anotaciones **@ManyToOne** y **@OneToOne**, donde también indica que el *"dueño"* de la relación es la otra entidad.
- **@Column** Usado para marcar el nombre de la columna en BD.

En el código fuente 3.1 se puede ver una de las entidades implementadas.

```
@Entity
public class User implements Serializable {

    private static final long serialVersionUID =
    -3210594041102407826L;
    @Id
    @GeneratedValue
    @Column(name= "user_id")
    private Long id;

    @Column(nullable = false, unique = true)
    private String email;

    private String fullName;
    private String password;
    private String role = "ROLE_USER";
    private Instant created;

    // Getters and Setters
}
```

Código Fuente 3.1: Ejemplo Entidad

Cabe destacar que también se ha utilizado el plugin de maven **maven-processor-plugin** para ejecutar en tiempo de compilación el preprocesado de hibernate **JPAMetaModelEntityProcessor**. Esto generará clases que contienen metainformación, como el nombre de cada atributo y su tipo, de de cada entidad, lo cual se usará al implementar **Specifications**, como veremos mas adelante. Estas clases autogeneradas se llamarán de la forma **Entity_**

Para cada entidad se crea una interfaz que servirá como definición del DAO. Por convección de nombrado, la interfaz tendrá un nombre como **EntityRepository**, siendo Entity el nombre de la entidad. Esta clase estará marcada con la anotación **@Repository**. De estas interfaces no hará falta realizar ninguna implementación dado que al iniciar la aplicación Spring Data JPA las creará automáticamente, adaptada a la base de datos que se haya especificado en la configuración de Spring Boot.

Además las interfaces extenderán a **JpaRepository**, la cual contiene definiciones de métodos básicos como las operaciones CRUD, y **JpaSpecificationExecutor** la cual proporciona métodos que permiten ejecutar consultas con filtros complejos y dinámicos, usando para ello implementaciones de la interfaz **Specifications**. En el Código Fuente 3.2 se puede ver un ejemplo de una definición de un DAO.

```
@Repository
public interface UserRepository
        extends JpaRepository<User, Long>,
        JpaSpecificationExecutor<User>,
        PagingAndSortingRepository<User, Long> {

    User findByEmail(String email);

    Optional<User> findUserByEmail(String email);

    @Query("SELECT DISTINCT u " +
            "FROM User u " +
            "INNER JOIN TimeLog t ON t.user = u " +
            "INNER JOIN Issue i ON t.issue = i " +
            "WHERE i.project.code = ?1")
    Set<User> findUsersWithTimeLogsInProject(String projectId);
}
```

Código Fuente 3.2: Ejemplo DAO

Al finalizar el desarrollo de la aplicación, las entidades y DAOs implementados son los siguientes:

- **AttachedFile** y **AttachedFileRepository**
- **Comment** y **CommentRepository**
- **DevelopmentCycle** y **DevelopmentCycleRepository**
- **Issue** y **IssueRepository**
- **ProjectPrivileges** y **ProjectPrivilegesRepository**
- **Project** y **ProjectRepository**
- **GitLabRepo** y **GitLabRepoRepository**
- **TimeLog** y **TimeLogRepository**

- **User** y **UserRepository**

Para los casos en los que fuera necesario implementar filtros complejos, se utilizó, como se mencionó anteriormente, implementaciones de la interfaz **Specifications** definida en **Spring Data**. Estas implementaciones hacen uso de **JPA Criteria API** para crear predicados reutilizables, permitiendo crear queries con filtros complejos y dinámicos. Las implementaciones se crearon en métodos **static** declarados en clases helper, utilizando para ello expresiones Lambda, tal como muestra el código 3.3.

```
1     public static Specification<Issue> issueTypeEqualsTo(
2             IssueType issueType){
3         return (Specification<Issue>)
4             (root, criteriaQuery, builder) ->
5                 builder.equal(root.get(Issue_.type), issueType);
6     }
7
8     public static Specification<Issue> issueFromProject(
9             String projectId){
10        return (Specification<Issue>)
11            (root, criteriaQuery, builder) -> {
12                Join<Issue, Project> projectJoin = root.
13                    join(Issue_.project);
14                return builder.equal(
15                        projectJoin.get(Project_.code),
16                        projectId);
17        };
18    }
```

Código Fuente 3.3: Ejemplo Specification para búsqueda de tareas

Podemos crear filtros complejos combinándolos mediante el uso de métodos definidos en las Specifications que representan los diferentes operadores lógicos, por ejemplo de la forma **filtro1 .or(filtro2)**. Estos métodos devuelven otra instancia de Specification, permitiendo que se pueda volver a combinar.

En ese ejemplo se puede ver también el uso de las clases con metainformación de las entidades (**Issue_** y **Project_**), las cuales le indican a **JPA Criteria API** el nombre y tipo de los atributos de la entidad.

El siguiente paso es la creación (o modificación) del servicio. Estos se marcaron con la anotación de Spring **@Service**, lo cual hará que en tiempo de ejecución se instancie un bean de dicha clase. En estos se crearán los métodos necesarios para realizar la funcionalidad del caso de uso, es decir, contendrán la lógica de negocio. Para ello obtendrán o modificaran los datos de BD utilizando los DAO, los cuales se insertarán en un atributo del servicio de forma automática al iniciar la aplicación usando la anotación **@Autowired**.

Los servicios que se implementaron en la aplicación son los siguientes:

- **GitLabRepoService**

- **IssueService**
- **ProjectService**
- **TimeLogService**
- **UserService**

```
@Service
public class TimeLogService {
    @Autowired
    private TimeLogRepository timeLogRepository;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private IssueRepository issueRepository;

    @Transactional
    public void logTime(String userEmail, @Valid TimeLogForm
        timeLogForm) throws UserNotFoundException,
            IssueNotFoundException {
        User user = userRepository.findByEmail(userEmail);
        if(user == null){
            throw new UserNotFoundException(
                new HashMap<String, Object>(){{
                    put("email", userEmail);
                }});
        }
        Issue issue = issueRepository
            .findById(
                timeLogForm.getIssueId()).orElseThrow(() ->
                    new IssueNotFoundException(
                        timeLogForm.getIssueId()));
        TimeLog timeLog = new TimeLog();
        timeLog.setUser(user);
        timeLog.setIssue(issue);
        timeLog.setComment(timeLogForm.getComment());
        timeLog.setDate(timeLogForm.getDate());
        timeLog.setHours(
            BigDecimal.valueOf(timeLogForm.getHours()));
        timeLogRepository.save(timeLog);
    }
    //other methods
}
```

Código Fuente 3.4: Ejemplo Servicio

Entre esos servicios, cabe mencionar que a diferencia de los otros **GitLabRepoService** realiza llamadas externas a la API de GitLab para recuperar los datos de los repositorios Git.

También es importante destacar que para separar totalmente la lógica del modelo del resto de la aplicación, se implementaron clases DTO para utilizar como respuesta o para agrupar parámetros en los métodos del servicio.

3.6.2 Implementación de Controlador

Para el controlador, lo primero es crear una clase en la cual se añadirá la anotación **@Controller**, lo cual permitirá que Spring la instancie en el arranque y que Spring MVC la identifique como una clase que contendrá los métodos que atenderán a las peticiones HTTP. Por convención, estas clases se llamarán de la forma **PageController**, siendo Page un nombre por el cual podamos identificar la página o acciones que maneja el controlador.

En estas clases, si bien es posible que tengan comprobaciones mínimas de datos o tratamiento de excepciones, no deben contener lógica de negocio, si no que delegaran esto en los Servicios. Para ello, se añadirá en cada controlador un atributo por cada servicio que precise utilizar anotado con **@Autowired** para que Spring inyecte los beans cuando se inicie la aplicación.

Luego se crearan los métodos que atenderán las peticiones que una funcionalidad necesite, en los cuales se añadirán las anotaciones **@GetMapping** o **@PostMapping**, dependiendo de si el método se debe ocupar de una petición HTTP GET o POST. Como ya se menciono, estos métodos delegaran en los servicios del modelo, conteniendo solo llamadas a estos, comprobaciones simples o captura de excepciones para devolver páginas o mensajes de error al cliente.

Las clases pertenecientes al controlador una vez finalizada la aplicación son las siguientes:

- **HomeController** para la página principal.
- **SignInController** para gestionar la autenticación.
- **AdminControlPanelController** para el panel de control.
- **UserController** para el perfil de usuario.
- **SearchUserController** para gestionar la búsqueda de usuario.
- **CreateUserController** para la página de crear usuario.
- **ChangePasswordController** para gestionar la página de cambio de contraseña.
- **AssignPrivilegesController** para asignar privilegios a los usuarios.
- **AssignProjectController** para asignar el proyecto a un usuario.
- **CreateCycleController** para crear un nuevo ciclo de desarrollo en el proyecto.
- **CreateProjectController** para crear un nuevo proyecto.
- **EditProjectController** para la edición de proyectos.

- **ProjectController** para mostrar los detalles de un proyecto.
- **RemoveProjectController** para las gestiones del borrado de los proyectos.
- **SearchCycleController** para la búsqueda de ciclos de desarrollo de un proyecto.
- **SearchProjectController** para la búsqueda de proyectos.
- **AssignGitRepositoryController** para gestionar la asignación de repositorios git a un proyecto.
- **CreateIssueController** para crear tareas,
- **IssueDetailsController** para mostrar los detalles de las tareas.
- **EditIssueController** para editar una tarea.
- **SearchIssuesController** para la búsqueda de tareas.
- **ChangeStatusController** para cambiarle el estado a una tarea.
- **AttachFileController** para adjuntar ficheros a una tarea.
- **LinkIssuesController** para asociar tareas.
- **ViewWatchersController** para ver los observadores de una tarea.
- **AddWatcherController** para añadir los observadores de una tarea.
- **AssignBranchController** para asignar una rama de git a un ciclo de desarrollo.
- **BranchesController** para ver y recuperar las ramas de un repositorio git asociado a un proyecto.
- **CycleCommitsController** para visualizar los commits realizados en una rama asociada a un ciclo de desarrollo.
- **GitRepositoriesController** para visualizar los repositorios git de la cuenta asociada a la aplicación.
- **IssueCommitsController** para visualizar commits recientes que estén relacionados con una tarea.
- **CycleDetailsController** para ver los detalles de un ciclo de desarrollo.
- **LogTimeController** para ver registrar el tiempo dedicado en una tarea.

- **ViewTimeLogsController** para visualizar, a diferentes niveles, los tiempos registrados.

- **CustomErrorController** página común para mostrar los diferentes errores.

```
@Controller
public class ProjectController {

    private static final String PROJECT_DETAILS_VIEW = //
        "project/details";

    @Autowired
    private ProjectService projectService;
    @Autowired
    private IssueService issueService;

    @GetMapping("/project/details/{projectID}")
    public String getProjectDetails(Model model, @PathVariable
        String projectID) throws OperationForbiddenException {

        ProjectDetails projectDetails = null;
        try {
            projectDetails = projectService
                .getProjectDetails(projectID, true);
            model
                .addAttribute("projectIssues",
                    projectDetails.getIssueDetails());
        } catch (EntityNotFoundException e) {
            return "redirect:/error";
        }
        model.addAttribute("projectDetails", projectDetails);

        return PROJECT_DETAILS_VIEW;
    }

}
```

Código Fuente 3.5: Ejemplo Controlador

Tal como se comentó en la sección de Diseño, cada **@Controller** gestionará las peticiones de una sola página, permitiendo separar y abstraer de una mejor manera las funcionalidades y cumpliendo de esta forma el **Principio de Responsabilidad Única**.

Por último, cabe destacar el uso de JavaScript y JQuery. Estos nos permiten capturar los eventos y ejecutar acciones en el cliente, así como cargar contenido dinámico haciendo llamadas AJAX al servidor.

3.6.3 Implementación de la Vista

Para la implementación de la vista se ha utilizado principalmente HTML y Thymeleaf. Con este último se podrá crear *"templates"* de páginas HTML, permitiendo de esa forma en tiempo de ejecución crear páginas con un contenido dinámico dependiendo de la ejecución concreta. Esas templates serán las que los controladores manden como respuesta, pasando antes de llegar al cliente por un preprocesado para convertirlas en HTML. También se ha usado bootstrap para crear páginas con un estilo adecuado de una forma fácil, sin necesidad de entrar en la edición o creación de estilos propios en CSS, así como para crear fácilmente las ventanas modales.

Dado que ambos están muy relacionados, la vista y el controlador de una funcionalidad se implementaron en la mayoría de los casos en paralelo. De esta forma se podía probar que todo fuese correcto mientras se desarrollaban.

Al principio del desarrollo se aprovecho para diseñar un estilo común de página, basado principalmente en el uso de una barra de menú superior y en el uso de la clase **Card** definida en bootstrap.

La barra de menú se implementó como un elemento **<nav>** de HTML en el cual se añadieron enlaces para acceder a las páginas de búsqueda de proyectos y búsqueda de tareas así como un submenú para acciones relacionadas con el usuario. Se implementó una única vez en un fichero llamado **header.html** y se marcó para poder utilizarse en otros ficheros con el atributo de thymelaf **th:fragment** dándole como valor **header**. Para añadirlo en todas las páginas, al crear su fichero html al comienzo del **<body>** se añade siempre el siguiente código:
`<div th:replace="fragments/header :: header"></div>`

Para el contenido de las páginas se utilizó la clase **Card** proporcionada por Bootstrap, con la cual podemos crear de forma fácil un estilo común. Podemos ver unn ejemplo en el Código Fuente 3.6

```
1   <div class="container">
2       <div class="card">
3           <div class="card-header">
4               <h5 th:text="#{title}"></h5>
5           </div>
6           <div class="card-body">
7               <th:block th:fragment="thisPageFragment">
8                   <!-- CONTENIDO -->
9               </th:block>
10          </div>
11      </div>
12  <div>
```

Código Fuente 3.6: Ejemplo contenido de página HTML

En dicho ejemplo se puede ver el uso de varios elementos y atributos proporcionados por

thymeleaf.

El primero es el atributo **th:text**. Este define el texto que se mostrará dentro del tag en el que se haya definido, pudiendo ser una variable, si va marcado de la forma **${variable}**, o un mensaje de internacionalización definido en la aplicación si va marcado como **#{mensaje}**, como en el ejemplo.

El otro es el elemento **<th:block>**. Este sirve principalmente para marcar bloques del html con otros atributos de thymeleaf. Una vez se procese el fichero ni las etiquetas ni los atributos de la misma aparecerán en el HTML final, pero si su contenido. En el ejemplo lo usamos para marcar el contenido de **card-body** con el atributo **th:fragment**, lo cual se hizo en prácticamente todas las páginas para permitir recuperar solo el contenido sin otros elementos de la página como el menú superior o el **card-header** y así poder usarlo, por ejemplo, en ventanas modales de otras funcionalidades.

Además de esos en el proyecto se usan con frecuencia el atributo **th:replace**, para el insertar el menú superior en todas las páginas u otros elementos declarados en ficheros comunes que se usen en varias páginas, y el **th:each** el cual se usa para iterar por una lista o colección de elementos, creando para cada uno de ellos el tag en el que se introduce el atributo y su contenido.

```
1     <tr th:each="issue : ${issues}">
2         <td class="col-auto">
3             <a th:text="${issue.issueID}"
4                 th:href="@{/issue/details/{id}(id =
    ${issue.issueID})}">
5             </a>
6         </td>
7         <td class="col-6" th:text="${issue.summary}"
8             style="word-break: break-all;"></td>
9         <td class="col-auto d-none d-sm-table-cell"
10            th:text="${issue.issueType}"></td>
11        <td class="col-auto">
12            <a th:text="${issue.assignedUser.fullName}"
13               th:href="@{/profile/{id}(id =
    ${issue.assignedUser.id})}"></a>
14        </td>
15    </tr>
```

Código Fuente 3.7: Ejemplo de página HTML

En el Código Fuente 3.7 se puede ver un ejemplo de **th:each** y también como Thymeleaf permite establecer valores dinámicos para otros atributos propios de HTML añadiéndoles **th:**. En este caso, se establece la URL de un enlace de forma dinámica en el atributo **th:href**, el cual al procesarse se cambiará por el atributo de HTML **href**.

Las páginas de Thymeleaf creadas a finalizar del desarrollo son:

- **signin** : Página para autenticarse en la aplicación. Como la aplicación es *cerrada*, es

decir solo para usuarios con cuentas creadas por administradores, también es la página a la cual se redirige por defecto si no se esta autenticado.

- **index** : Página principal de bienvenida de la aplicación. Incluye un mensaje de bienvenida y un pequeño resumen con los proyectos y tareas asignadas.
- **admin**
  - **panel** : Panel de control de administración. Contiene enlaces a otras páginas para facilitar el trabajo de un administrador.
- **user**
  - **user-profile** : Página que contiene los detalles de un usuario, junto con sus proyectos asignados.
  - **change-password** : Página que puede ser usada por los usuarios para cambiar la contraseña de su cuenta.
  - **assign-privileges** : Página accesible por los administradores para gestionar los permisos que un usuario tiene en un proyecto concreto.
  - **create** : Página usada por los administradores para crear nuevos usuarios.
  - **search** : Página para buscar usuarios utilizando varios criterios de búsqueda.
- **project**
  - **details** : Página que muestra los detalles de un proyecto. También sirve de acceso para otras páginas para gestionar proyectos.
  - **create** : Página usada por administradores para crear nuevos proyectos.
  - **edit** : Página usada por administradores para modificar proyectos.
  - **remove** : Página usada por administradores para eliminar proyectos.
  - **search** : Página para buscar proyectos.
  - **assign** : Página usada para asignar proyectos.
  - **assign-git** : Página usada por los administradores para asignar un repositorio git a un proyecto.
  - **create-cycle** : Página para crear un nuevo ciclo de desarrollo.
  - **search-cycle** : Página para buscar un ciclo de desarrollo.
- **issue**
  - **add-watcher** : Página utilizada para añadir nuevos observadores a una tarea.

  - **attach** : Página utilizada para adjuntar elementos a una tarea.
  - **change-status** : Página para cambiar el estado de una tarea.
  - **create** : Página para crear una nueva tarea en un proyecto.
  - **details** : Página para mostrar los detalles de una tarea.
  - **edit** : Página para editar los detalles de una tarea.
  - **link-issues** : Página para asociar tareas.
  - **log-time** : Página usada para añadir tiempo realizado a una tarea.
  - **related-issues** : Página para ver las tareas relacionadas.
  - **search** : Página para buscar tareas.
  - **view-watchers** : Página para ver los observadores de una tarea.
- **time-logs** : Página para visualizar los tiempos registrados en una tarea.
- **cycle**
  - **details** : Página para visualizar los detalles de un ciclo de desarrollo, incluyendo las tareas que pertenecen al mismo.
- **git**
  - **assign-branch** : Página para asignar una rama a un ciclo de desarrollo.
  - **branches** : Página para ver y buscar las ramas de un repositorio git de un proyecto.
  - **cycle-commits** : Página para ver los últimos commits asociados a tareas que pertenecen a un ciclo de desarrollo.
  - **issue-commits** : Página para ver los últimos commits asociados a una tarea.
  - **repositories** : Página para ver los repositorios de git para la cuenta que se haya asociado a la aplicación.

Cabe destacar que conforme se avanzó en el desarrollo, se fueron encontrando varios elementos que se podrían reusar en diferentes páginas. En algunos casos esos elementos se movieron a los siguientes ficheros:

- **header.html** contiene el menú superior.
- **resultTables.html** contiene los elementos necesarios para mostrar en formato de tabla una lista con datos de proyectos, tareas y usuarios, así como los componentes necesarios para aplicar la paginación.
- **components.html** contiene componentes comunes como la implementación de la ventana modal, mensajes de alerta o de error y campos para formularios.

Capítulo 4

Validación

PARA un proyecto software la validación es una parte importante del proceso. Es el momento en el cual se comprueba que los elementos creados durante el proceso de desarrollo no dan error, funcionan correctamente y dando el resultado esperado.

En este capitulo hablaremos de como se ha aplicado la validación en este proyecto.

4.1 Pruebas de Unidad

El propósito de las pruebas de unidad es comprobar que cada elemento unitario de la aplicación funciona como se espera. Con estas se intenta comprobar un solo componente , a poder ser de forma aislada del resto, para observar si este cumple con su cometido para una o varias entradas controladas. Por esto en la aplicación se han usado Mocks para controlar completamente las respuestas de otros otros componentes.

Debido a la naturaleza del proyecto, así como a la tecnología utilizada, las pruebas de unidad que se han implementado afectan principalmente a los métodos de los servicios dado que son estos los que contienen la lógica de negocio. Si bien esto podría afectar a los porcentajes de cobertura de estos tests, viendo la tecnología utilizada podría entenderse el porque se ha realizado de esta forma. Para empezar, debido a que Spring Data JPA crea de forma automática los repositorios, no se vio la necesidad de crearles pruebas. Para el caso de los controladores de Spring MVC, si bien si contienen cierta funcionalidad, esta no es de una gran complejidad, pudiendo ser probada directamente por el desarrollador mientras implementa los elementos de la vista, o en los tests de integración.

4.2 Pruebas de Integración

El propósito de las pruebas de integración es comprobar que los diferentes elementos o módulos del sistema se instancian, funcionan y se comunican correctamente entre si.

En este proyecto, las pruebas de integración se han enfocado en comprobar que los componentes se comunican correctamente, desde la entrada de una petición en los controladores de Spring MVC hasta la modificación o consulta de la base de datos. Para esto, se ha utilizado la clase proporcionada por Spring MVC **MockMvc** para simular llamadas HTTP contra los controladores, los cuales después se comunicaran con el modelo a través de los servicios y estos contra la base de datos a través de los DAO.

Cabe destacar que, para controlar los datos manipulados en los test, así como para evitar que cualquier dato preexistente pueda influenciar en el resultado de los test, se ha creado una configuración especial que solo se carga en los tests para que se cree y se utilice una base de datos en memoria, la cual no se guarda de forma permanente. También se han marcado las clases con la anotación **@Transactional**, lo cual hace que todo lo que ocurre en cada test sea dentro de una sola transacción en la cual Spring hará rollback de forma automatizada, por lo que incluso cuando en uno se manipulen los datos, nunca afectará a los otros test.

4.3 Pruebas no Automatizadas

Para asegurar el correcto funcionamiento de todos los elementos de la aplicación, se han realizado pruebas no automatizadas realizadas por el desarrollador. Estas, al igual que el resto, no se realizan una única que vez si no que se repiten varias veces durante el desarrollo.

Principalmente estas comienzan una vez finalizado el desarrollo o cambio de una funcionalidad, probando todos los elementos de la misma incluso cuando no se ha cambiado por completo. Además, dado que para ciertos desarrollos varias funcionalidades pueden tener elementos comunes y afectarse entre sí, también se comprueban al finalizar un ciclo, para confirmar que todas las funcionalidades del mismo se entregan correctamente.

También fuera de ciclos o desarrollos se han vuelto comprobar los diferentes elementos de la aplicación, incluidos los implementados en fases iniciales, para asegurarse de que siguen funcionando correctamente.

Capítulo 5

Manual de Usuario

En este capitulo se mostrará desde la perspectiva de un usuario las distintas funcionalidades de GesTa, mostrando paso a paso las diferentes formas de interactuar con ella.

Cabe destacar que todos los elementos de la aplicación están reservados a usuarios registrados. Además ciertas funcionalidades solo estarán disponibles para algunos roles de usuario. La aplicación esta pensada principalmente para su despliegue en entornos de una organización cerrada, por lo que una persona no podrá registrarse libremente para poder acceder. En su primera ejecución la aplicación crea un usuario para el administrador con una contraseña predefinida, con el cual se podrán crear nuevos usuarios.

5.1 Explicación de Términos, Tipos y Estados

En la aplicación, así como en este manual, se hacen uso de diversos términos o se mencionan tipos o estados. En esta sección se identificaran los principales términos que se podrán emplear en los elementos de la aplicación, así como los distintos valores que puedan tomar.

- **Estado de Proyecto**
  - **Abierto** : proyectos en los cuales se esta trabajando de forma activa.
  - **Cerrado** : proyectos en los que no se trabaja actualmente. Solo para datos históricos
- **Estado de Tarea**
  - **Abierta** : La tarea se ha registrado pero todavía no se ha comenzado a trabajar en ella.
  - **En Progreso** : Se ha comenzado a trabajar en la tarea.
  - **Cerrada** : La tarea se ha completado.
- **Tipo de Tarea**

- **Tarea** : Tipo genérico, para tareas normales.
- **Mejora** : Para marcar las tareas en las cuales se pretende hacer una mejora sobre el producto.
- **Incidencia** : Para marcar las incidencias o errores del producto que se deben corregir.

5.2 Aclaración de Ventanas Modales

En la aplicación se hace uso de ventanas modales las cuales se abren a través de enlaces o botones presentes en ciertas páginas. En algunos casos, estas ventanas modales también podrán usarse como páginas completamente independientes. Para estos casos, se marcará con una referencia a la sección de dicha página, en lugar de explicar su funcionamiento allí donde se use.

5.3 Pagina de Login

Para el uso de la aplicación es necesario estar autenticado. Por ello al acceder a la aplicación se redirigirá automáticamente a la página de Login. Esta es una página simple en donde el usuario podrá introducir sus datos de acceso.

En la figura 5.1 se puede ver la página de inicio de sesión.

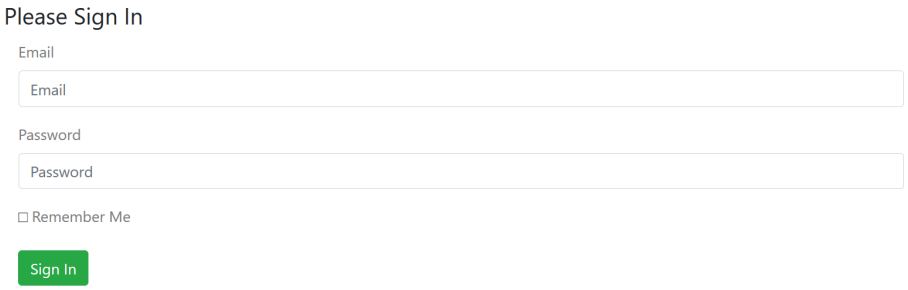


Figura 5.1: Página de Inicio de Sesion

5.4 Barra de Menú

La barra de menú superior es accesible desde todas las páginas de la aplicación y ofrece enlaces a búsqueda de tareas y búsqueda de proyectos. A la derecha de la barra aparece un enlace para acceder a las tareas que el usuario tiene asignadas y un submenú con enlaces para acceso a la página de perfil de usuario, al panel de administración en caso de ser administrador

y uno para desconectarse. Por último, la barra también contiene un submenú para seleccionar el idioma de la aplicación.

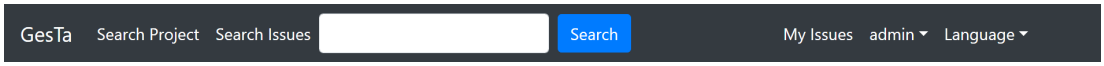


Figura 5.2: Barra de Menú

5.5 Página de Inicio

La página de inicio de la aplicación ofrece un resumen útil para el usuario, incluyendo un listado con los proyectos asignados, un listado con las tareas asignadas y también un listado con las tareas que el usuario está observando. Estos listados contendrán la misma información que las tablas resultado mostradas en la página de búsqueda de proyectos (5.6) para el caso de los proyectos asignados, y de la página de búsqueda de tareas (5.7) para las tareas asignadas y observadas.

En la figura 5.3 se puede ver un ejemplo de la página de inicio.

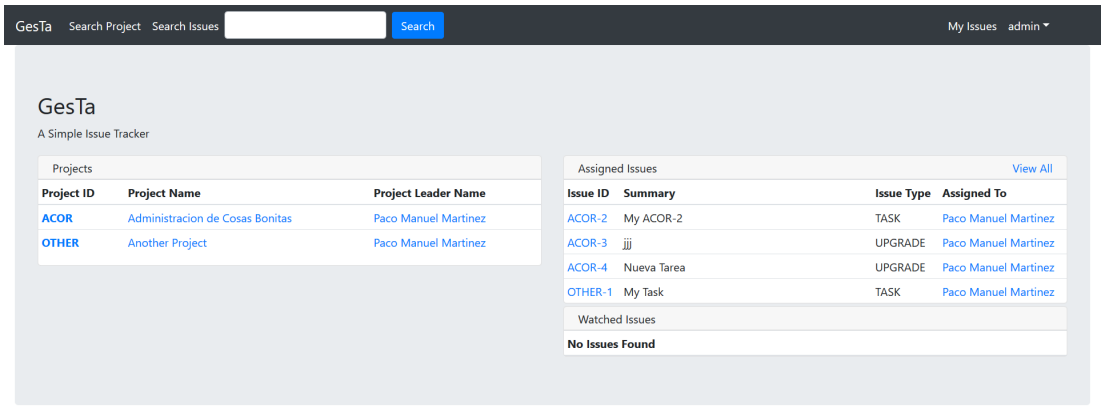


Figura 5.3: Página de Inicio

5.6 Búsqueda de proyectos

En la página de búsqueda de proyectos veremos una lista de los proyectos a los cuales el usuario tiene acceso. En caso de ser un administrador le mostrará todos los proyectos. Además también se mostrará unos filtros por los cuales el usuario puede filtrar los proyectos que se muestran. Estos filtros son:

- Identificador o nombre del proyecto.

- Nombre o email del Jefe de proyecto.

La lista de proyectos resultados estarán formadas por el identificador y el nombre del proyecto, los cuales serán a su vez enlaces para acceder a la página de detalles de proyecto (5.12), y el nombre del líder del proyecto, que será a su vez un enlace para la página de detalles de usuario (5.9).

En la figura 5.4 se muestra la página de búsqueda de proyectos.

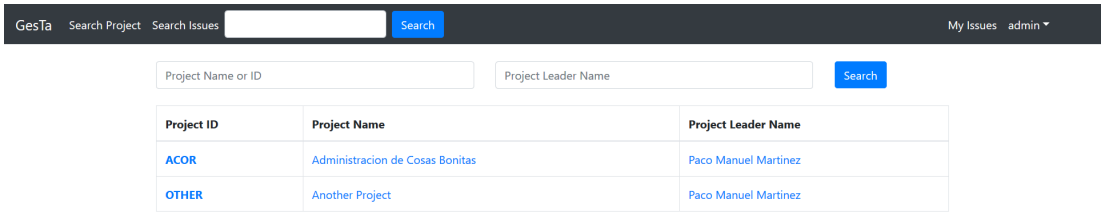


Figura 5.4: Buscar Proyecto

5.7 Búsqueda de Tareas

En la página de búsqueda de tareas se mostrará una lista de tareas pertenecientes a los proyectos a los cuales el usuario tiene acceso. En caso de ser un administrador se le mostrarán todas las tareas de todos los proyectos. Además también se mostrarán los siguientes filtros:

- Identificador de Tarea.
- Nombre o descripción breve de la tarea.
- Responsable de realizar la tarea.
- Tipo de Tarea.

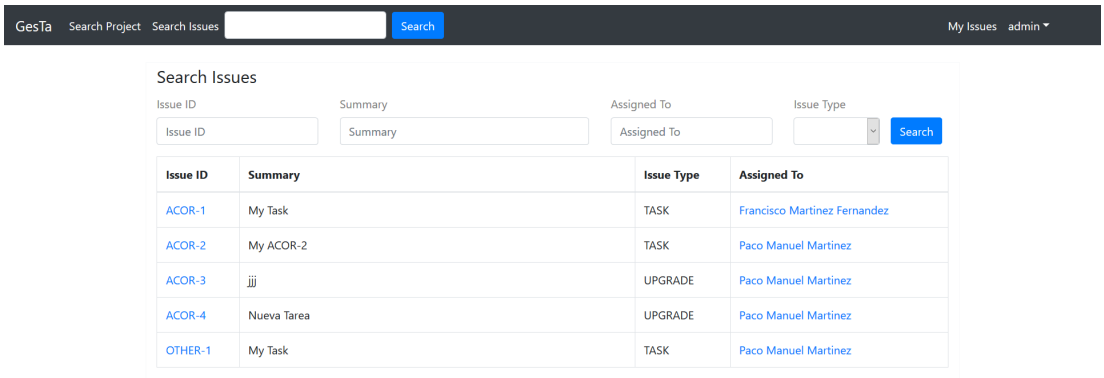


Figura 5.5: Buscar Tareas

Para cada tarea que cumpla los filtros se mostrarán los siguientes datos:

- Identificador de tarea, con un enlace para la página de detalles de tarea (5.14).
- Nombre o descripción breve de la tarea.
- Tipo de Tarea.
- Responsable de realizar la tarea, con un enlace a la página de detalles de usuario (5.9)

En la figura 5.5 se puede ver un ejemplo de esta página.

5.8 Búsqueda de Usuarios

Esta página está compuesta por una lista de usuarios y por lo siguientes filtros:

- Email de usuario.
- Nombre de Usuario.
- Identificador de Proyecto asignado.

Para cada usuario que cumpla los filtros se mostrará su email y su nombre completo, siendo este ultimo además un enlace a la página de perfil de usuario (5.9).

En la figura 5.6 se puede ver un ejemplo de la ejecución de esta página.

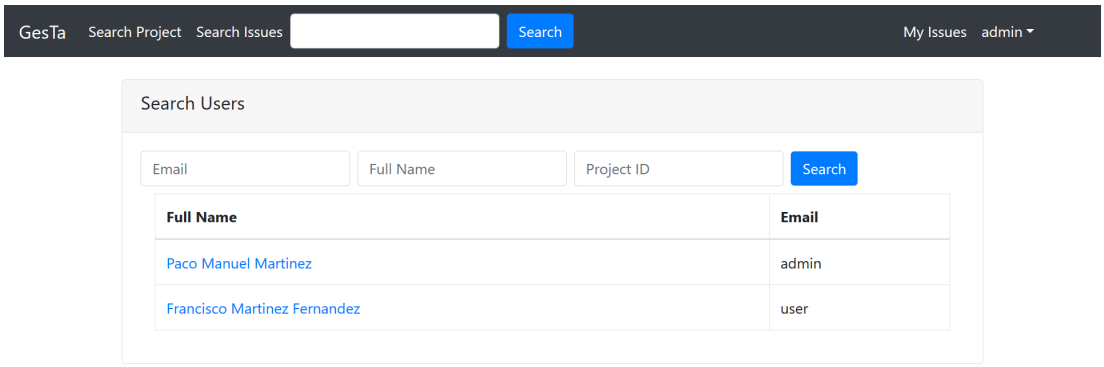


Figura 5.6: Buscar Usuarios

5.9 Perfil de Usuario

En esta página se muestran varios detalles de un usuario en dos secciones diferenciadas.

En la primera se muestran los datos del usuario, su nombre completo y email. Si el usuario que entra en esta página es el mismo al cual pertenece el perfil, se mostrará un enlace para abrir una modal para cambiar la contraseña. Si es un administrador, se mostrará un enlace que abrirá la página de gestión de privilegios en proyectos (5.20).

En la segunda sección se muestra una lista de los proyectos que el usuario tiene asignado, de una forma similar a como se muestra el resultado de una búsqueda de proyectos (5.6). Si el

usuario que accede a la página del perfil es un administrador, se mostrará también un enlace para acceder a la página de Asignar Proyectos (5.11).

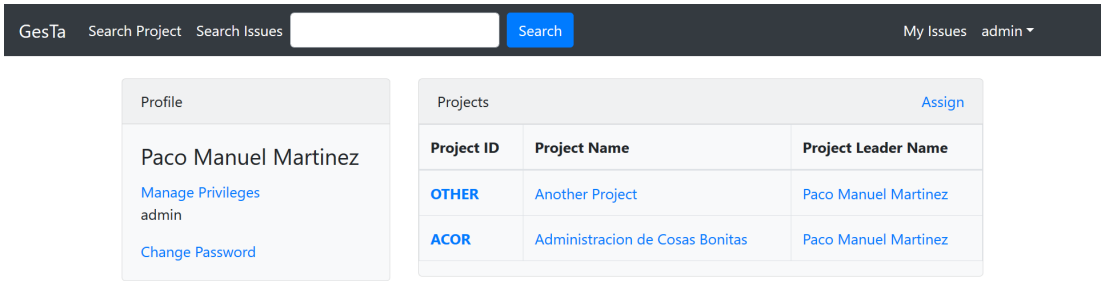
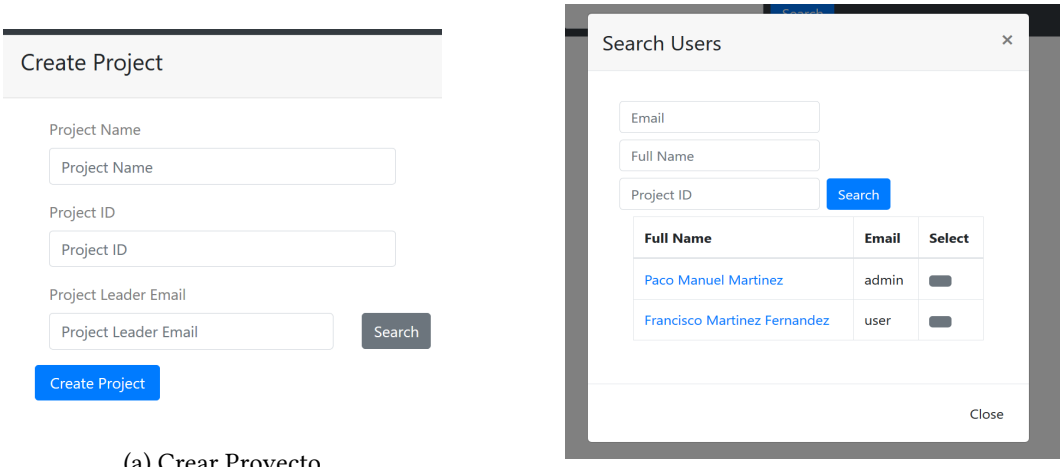


Figura 5.7: Perfil de Usuario

5.10 Crear Proyecto

A través de esta página los administradores podrán crear nuevos proyectos.



(a) Crear Proyecto

(b) Búsqueda de Usuarios

Figura 5.8: Imágenes de Creación de Proyectos

Para ello deberán introducir los siguientes datos:

- Nombre de Proyecto.
- Identificador de Proyecto.
- Email del líder de proyecto.

Para ayudar en la selección del líder del proyecto, tendremos al lado del campo del email un botón que abrirá con el contenido de la página de búsqueda de usuario (5.8) con un boton al lado de cada resultado para seleccionarlo.

En las figura 5.8 se muestra la página de creación de proyecto y la modal de búsqueda de usuarios.

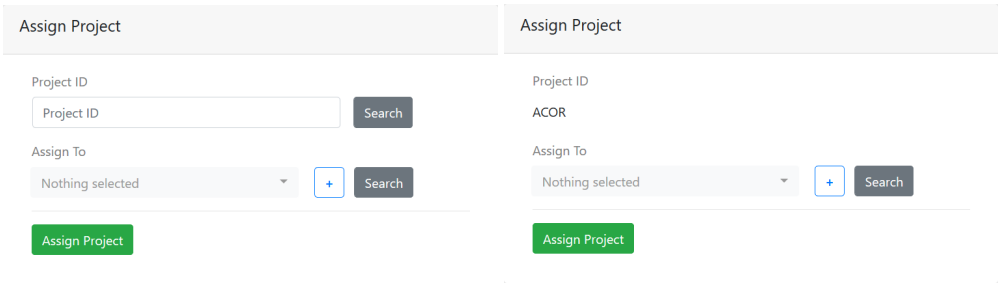
5.11 Asignar Proyectos

Esta página permitirá asignar proyectos a uno o mas usuarios. Se podrá acceder desde varias páginas de la aplicación, y su funcionamiento puede variar dependiendo desde donde se acceda.

Desde los detalles de usuario podremos acceder a una versión visible solo para administradores (Figura 5.9a). En esta se pide que se introduzca el identificador del proyecto a asociar, así como un botón que abre una modal con el contenido de la página de búsqueda de proyectos (5.6) para hacer más fácil la selección de proyectos.

Tras esto contiene un buscador simple por el nombre completo de un usuario con un botón para añadir al usuario seleccionado y un botón con el cual se abre una ventana modal con el contenido de la página de búsqueda de usuarios (5.8) para hacer una búsqueda avanzada. Al seleccionar usuarios por cualquiera de estos métodos se añadirán a una lista. Si se accedió desde la página de detalles de un usuario, la lista ya contendrá a dicho usuario.

Desde la página de detalles de proyecto los administradores y el líder del proyecto podrán a otra versión (Figura 5.9b). En esta, el proyecto viene prefijado, y no se permite su modificación.



(a) Página General (b) Página desde Proyecto

Figura 5.9: Asignar Proyecto

5.12 Detalles de Proyecto

En está página se muestra una gran variedad de datos de un proyecto.

En la parte superior se muestra el estado del proyecto, si esta abierto o cerrado, su identificador y el nombre del proyecto. Si el usuario es un administrador también se mostrarán dos botones para editar y eliminar el proyecto.

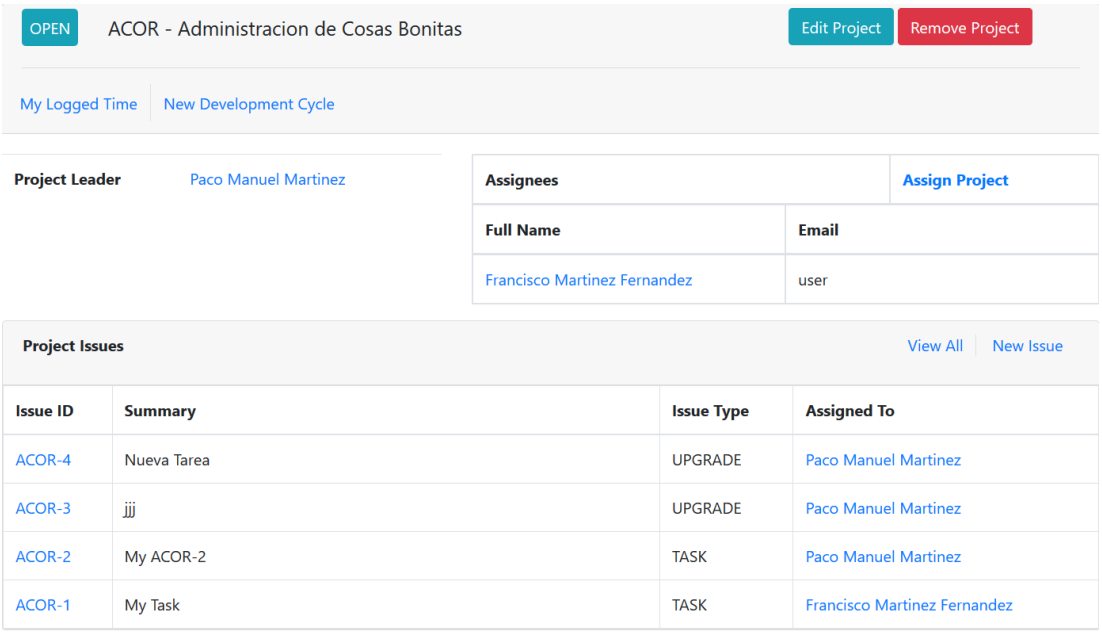


Figura 5.10: Detalles de Proyecto

Debajo de la información del proyecto habrá un enlace para acceder a la página de ver tiempos registrados (5.19) prefiltrando por el identificador del proyecto, y otro para la creación de un nuevo ciclo de desarrollo. Este último abrirá una ventana modal, tal como se muestra en la figura 5.11, en la que se le pedirá al usuario que introduzca un nombre para el ciclo.

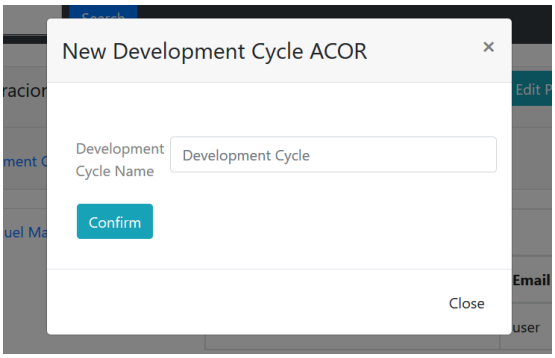


Figura 5.11: Nuevo Ciclo de Desarrollo

La siguiente sección de la página contiene datos sobre los integrantes del proyecto. Por un lado tenemos el nombre del jefe de proyecto, y a su lado una tabla con los nombres y emails de los usuarios con el proyecto asignado. En ambos, el nombre completo es a su vez un enlace para acceder al perfil de dicho usuario (5.9). Además, si el usuario que accede a la página es un administrador o el líder del proyecto, en la parte superior de la tabla aparecerá un enlace para asignar el proyecto a mas usuarios (5.11).

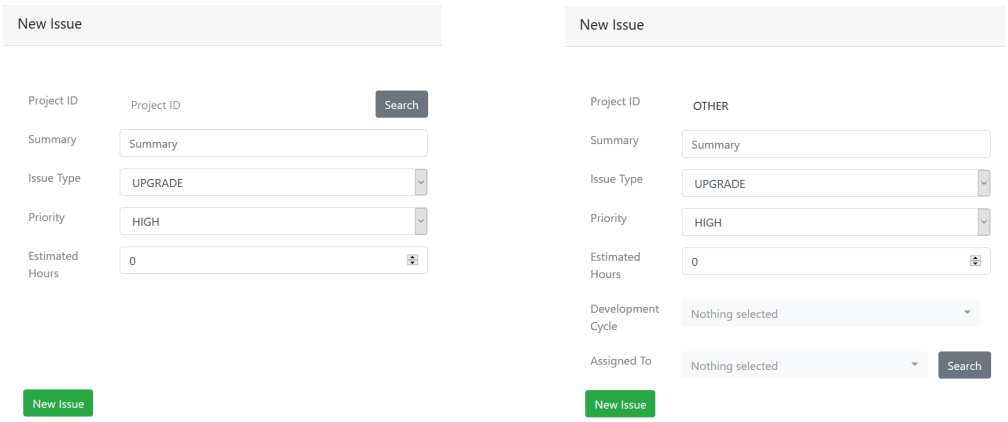
Por último se muestra una tabla con las tareas mas recientes del proyecto. En la parte superior de esta tabla, hay dos enlaces, uno abrirá la página de buscar tarea (5.7) con el filtro de identificador de proyecto preestablecido y el otro abrirá la página de crear nueva tarea (5.13).

En la figura 5.10 podemos ver un ejemplo de la página de detalles.

5.13 Crear Tarea

Los usuarios podrán crear tareas en los proyectos que tengan asignados accediendo a esta página desde la detalles de proyecto (5.12). El sistema le pedirá al usuario que introduzca los siguientes datos de un proyecto:

- Identificador del proyecto.
- Nombre o Descripción Breve.
- Tipo de Tarea.
- Prioridad.
- Horas Estimadas.



(a) (b)

Figura 5.12: Nueva Tarea

Para introducir el proyecto, los usuarios han de hacer click sobre el botón de buscar, el cual abrirá una ventana modal con el contenido de la búsqueda de proyectos (5.6) desde la que el usuario podrá seleccionarlo. Una vez seleccionado, se hará visible el selector de ciclos de desarrollo y el buscador de usuarios para seleccionar a la persona responsable de realizar la tarea.

La figura 5.12a es un ejemplo de este caso.

Para cuando se accede a esta página desde la de detalles de proyecto (5.12), la página se muestra con algunas diferencias. En primer lugar no aparece el botón para buscar proyecto, si

no que ya viene prefijado con el identificador de proyecto. Además el buscador de usuarios y el selector de ciclos aparecen por defecto. La figura 5.12b es un ejemplo de este segundo caso.

5.14 Detalles de Tarea

En está página se muestran diversos datos referentes a una tarea, siendo accesible para todos los usuarios con permisos en el proyecto al cual pertenezca.

En la parte superior de la página se muestra el Identificador del proyecto que es a su vez un enlace a su página del proyecto (5.12). Bajo este se muestra el Identificador y el nombre de la tarea, así como su estado. En esta sección podremos ver además los siguientes enlaces:

- Editar Tarea (5.15).
- Registrar Tiempo (5.18).
- Ver Tiempos Registrados (5.19).
- Cambiar Estado de la Tarea.

Estos enlaces abrirán una ventana modal con el contenido de dichas páginas.

En la siguiente sección se muestran los siguientes detalles de una tarea:

- Tipo de Tarea.
- Prioridad.
- Tiempo Estimado.
- Ciclo de Desarrollo.
- Fecha de Inicio.
- Fecha de fin.
- Responsable de realizar la tarea.
- Usuario que ha creado la tarea.

Tanto el responsable de la tarea como el creador son además enlaces a los perfiles de dichos usuarios (5.9)

Lo siguiente que se muestra son dos enlaces que abren modales, uno para añadir observadores a la tarea (5.16) y el otro para ver los observadores (5.17), seguidos de la descripción de la tarea.

Por último tenemos la sección de ficheros adjuntos, en la cual se muestra un listado de enlaces a los ficheros adjuntos a la tarea, y la sección de comentarios, que nos permitirá ver y añadir comentarios a la tarea.

En la figura 5.13 tenemos un ejemplo de una página de detalles de tarea.

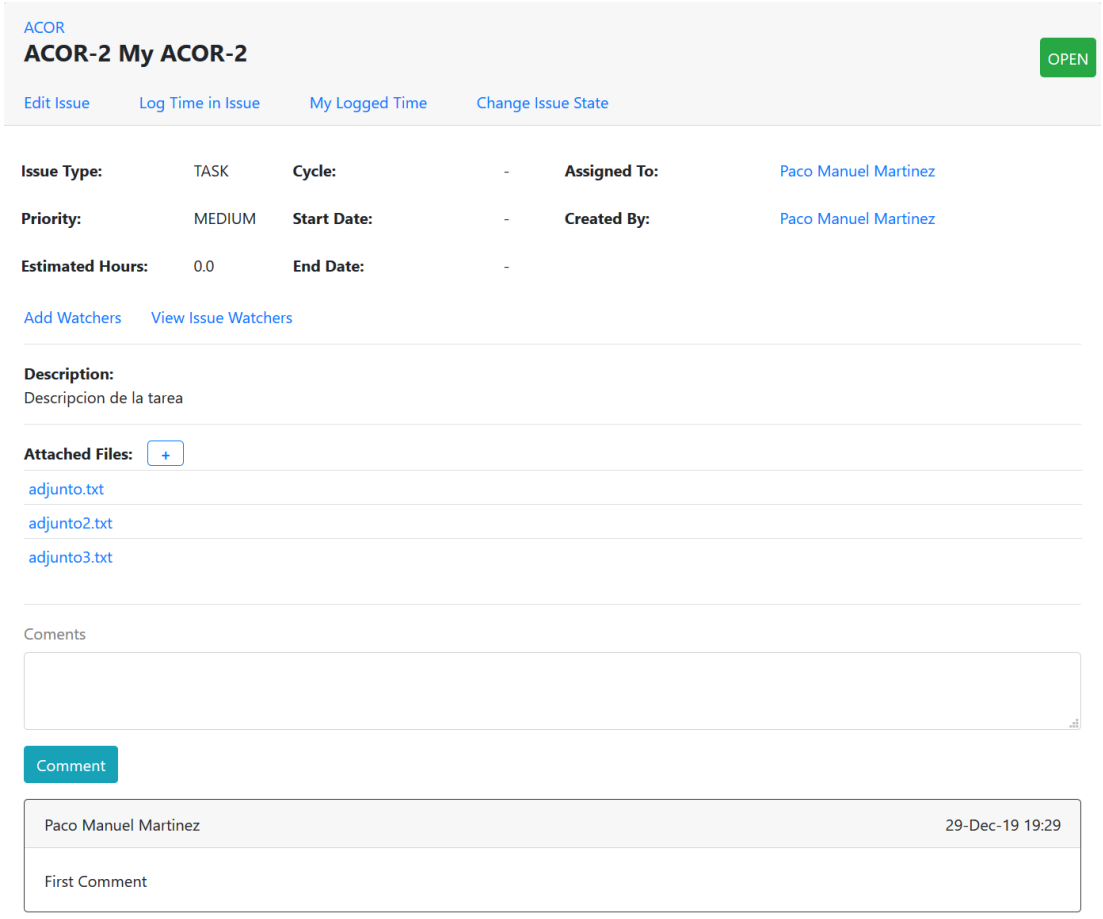


Figura 5.13: Página de Detalles de una Tarea

5.15 Editar Tarea

En esta página se podrán editar los siguientes datos de una tarea:

- Nombre o descripción breve de la tarea.
- Tipo de Tarea.
- Prioridad.
- Ciclo de Desarrollo
- Tiempo Estimado.
- Usuario Responsable.
- Descripción.

En dichos campos se mostrarán los valores actuales de la tarea.

Si el usuario no pone algún valor valido para alguno de los campos el sistema marcará el error y no editará la tarea. Estos valores no validos pueden ser nombre de la tarea vacío, tiempo estimado negativo o no seleccionar a ningún responsable.

En la figura 5.14 se puede ver un ejemplo de la página de editar tarea.

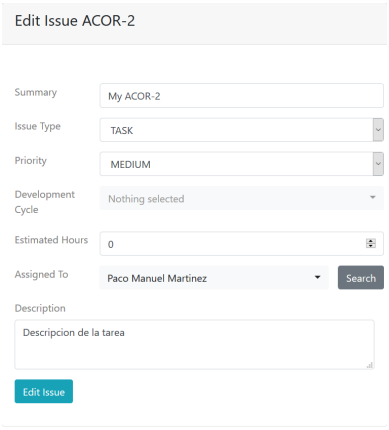


Figura 5.14: Editar Tarea

5.16 Añadir Observador

En esta página los usuarios podrán añadir observadores a una tarea y se accede desde la página de detalles de tarea (5.14). La página se puede encargar de registrar varios observadores a la vez.

Para seleccionar los observadores hay dos formas. La primera es haciendo click al botón buscar. Esta abrirá otra ventana modal con el contenido de la página de buscar usuario (5.8) en la que se puede hacer una búsqueda avanzada y seleccionar el usuario a añadir. La segunda forma es mediante un buscador simple por nombre.

Una vez seleccionado algún usuario, estos se irán mostrando como una lista bajo las tareas.

Por último, para añadir de forma definitiva los observadores, es necesario hacer click sobre el botón de confirmar localizado en la parte superior de la página.

En la figura 5.15 podemos ver un ejemplo.

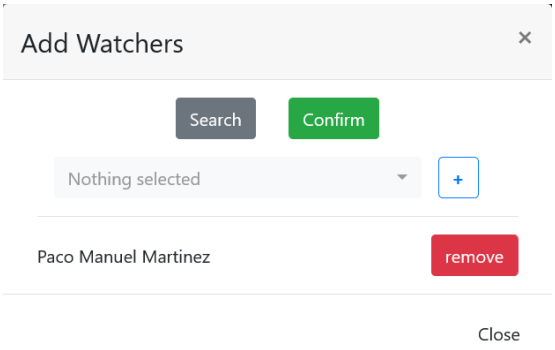


Figura 5.15: Añadir Observador

5.17 Ver Observadores

En esta página, accesible desde la página de detalles de tarea (5.14), se pueden ver los observadores de una tarea.

En ella se muestra una tabla con el nombre y el email de cada observador. Además, si el usuario que abre esta ventana es observador de la tarea, se mostrará junto a sus datos un botón para eliminarse de la lista. En caso de ser administrador el botón se mostrará para todos los usuarios.

En la figura 5.16 podemos ver un ejemplo de esta página.

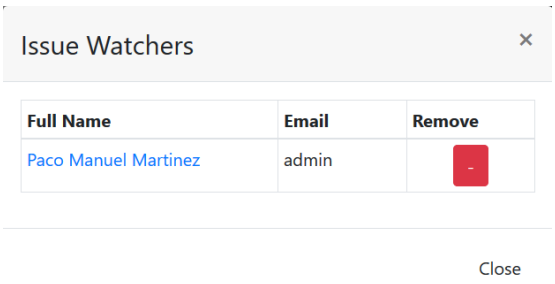


Figura 5.16: Ver Observadores

5.18 Registrar Tiempo

En esta página un usuario puede registrar el tiempo que ha dedicado a una tarea en un día concreto accediendo desde la página de detalles de tarea (5.14) como una ventana modal.

La aplicación le pide al usuario que introduzca los siguientes datos:

- Fecha en la que se hizo el trabajo.
- Horas dedicadas.
- Comentario (Opcional).

Una vez introducidos los datos, se comprueba su validez, marcando como error si no se ha introducido la fecha u hora, o si se introduce una hora negativa.

En la figura 5.17 podemos ver la página para registrar tiempo.

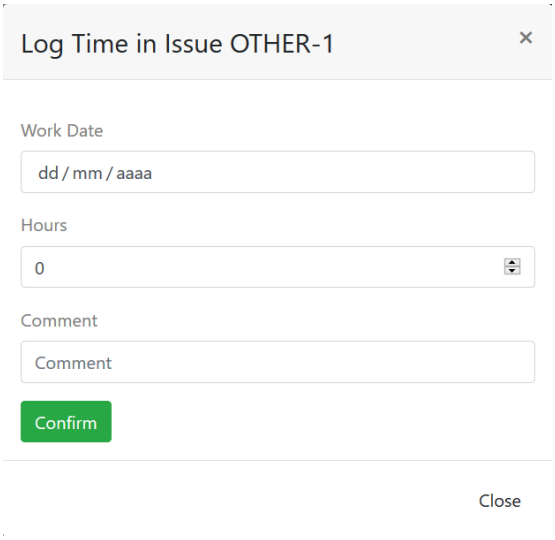


Figura 5.17: Registro de Tiempo Dedicado

5.19 Ver Tiempos Registrados

En esta página se mostrará una lista de tiempos registrados, mostrando para cada uno el identificador de la tarea, la fecha en la que se realizado, las horas trabajadas y el comentario que el usuario haya añadido al registrarlo.

A esta página podemos acceder desde el submenú de usuario de la barra superior o como una ventana modal desde la página de detalles de tarea (5.14). Si esto se abre como modal solo se mostrarán los tiempos registrados por el usuario en esa tarea. En caso de hacerlo desde el submenú, se mostrarán todos los tiempos registrados por el usuario, así como la opción para filtrar el resultado por identificador de tarea o identificador de proyecto.

En la figura 5.18 se muestra un ejemplo del contenido de esta página.

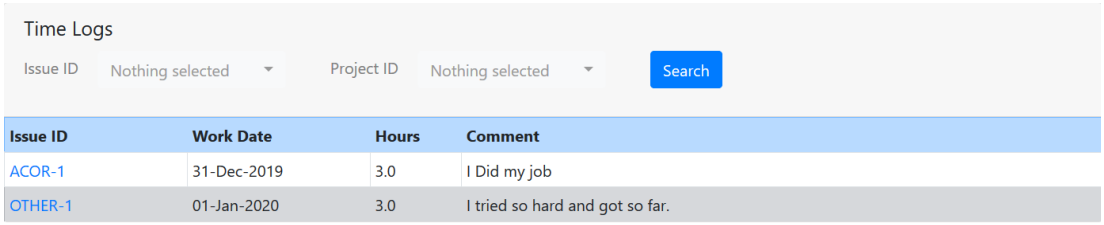


Figura 5.18: Tiempos Registrados por un Usuario

5.20 Gestión de Permisos

En esta página los administradores podrán modificar los permisos o privilegios que un usuario tiene en un proyecto concreto. Además también servirá para asignar un proyecto al usuario. Esta página es accesible desde la página de perfil de usuario (5.9).

En la parte izquierda se mostrará una sección similar a la página de asignar proyecto (5.11) con la diferencia de que en este caso solo aparece el apartado de seleccionar proyecto, y no se puede añadir mas usuarios.

A la derecha los administradores podrán manejar los privilegios que un usuario tiene en un proyecto, con un selector en donde seleccionar un proyecto que el usuario tenga asignado y se le quiera modificar algún permiso, seguido de un multiselector donde seleccionar los distintos permisos a asignar al usuario. En caso de no seleccionar ninguno, se le quitara el acceso al usuario al proyecto y dejará de tenerlo asignado.

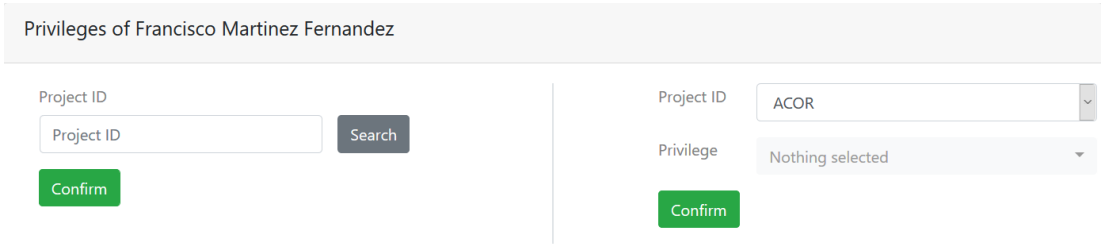


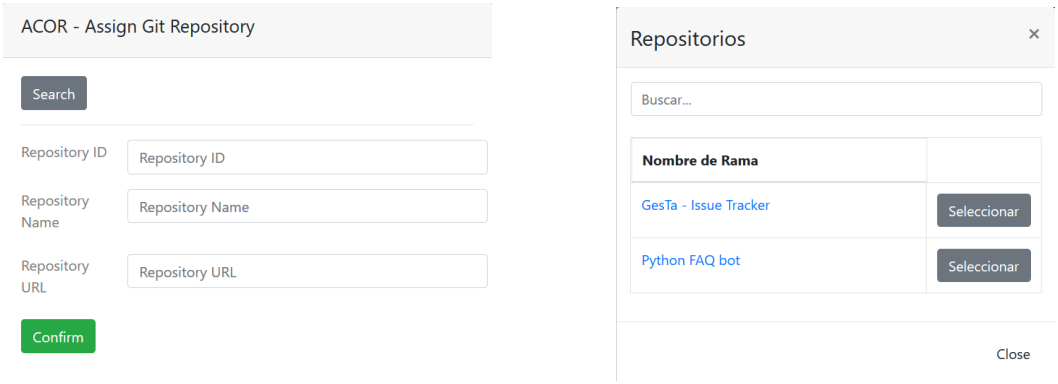
Figura 5.19: Gestión de Permisos

5.21 Asignar Repositorio

En esta página los administradores podrán asignarle un repositorio a un Proyecto.

Para ello la aplicación pedirá al usuario que indique el identificador del repositorio, el nombre del repositorio y por último la URL. Se puede ver un ejemplo de esto en la figura 5.20a.

En caso de que la aplicación tenga asociada una cuenta de GitLab, en la parte superior de la página habrá un botón que abrirá una ventana modal (Figura 5.20b) en la que se podrá seleccionar un repositorio a partir de una lista que contendrá aquellos pertenecientes a la cuenta asociada. También se permitirá filtrar esta lista mediante el cuadro de búsqueda de la parte superior.



(a) (b)

Figura 5.20: Asignar Repositorio

5.22 Detalles de Ciclo de Desarrollo

En esta página se podrá ver los detalles relacionados con un ciclo de desarrollo.

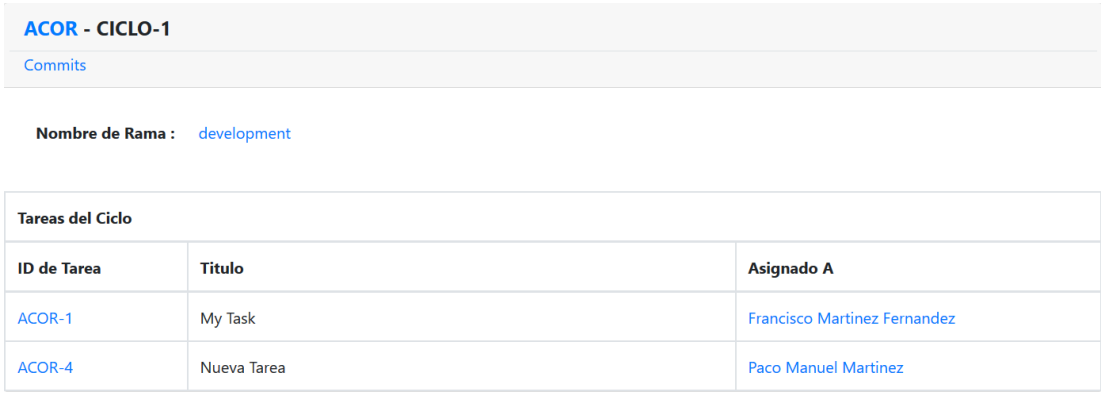


Figura 5.21: Detalles de un Ciclo de Desarrollo

En la parte superior, en caso de tener asociada una rama de un repositorio al ciclo de desarrollo, habrá un enlace para ver los commits hechos sobre esa rama (5.25), seguido de un enlace a la página de GitLab que se mostrará como el nombre de la rama.

Por último, contendrá una tabla con información de las tareas pertenecientes al ciclo, mos-

trando para cada una de ellas su identificador, siendo este un enlace a detalles de tarea (5.14), su titulo o nombre y el nombre completo del responsable de realizarla, siendo este un enlace a su perfil (5.9).

En la figura 5.21 se puede ver un ejemplo de esta página.

5.23 Búsqueda de Ciclos de Desarrollo

En esta página los usuarios podrán ver y buscar ciclos de desarrollo relacionados con un proyecto concreto. Para cada ciclo se mostrará su nombre como un enlace a la página de detalles del ciclo (5.22), y en la parte superior se podrá filtrar el resultado mediante un cuadro de texto en el que indicar el nombre del ciclo.

En la figura 5.22 se encuentra un ejemplo de esta página.

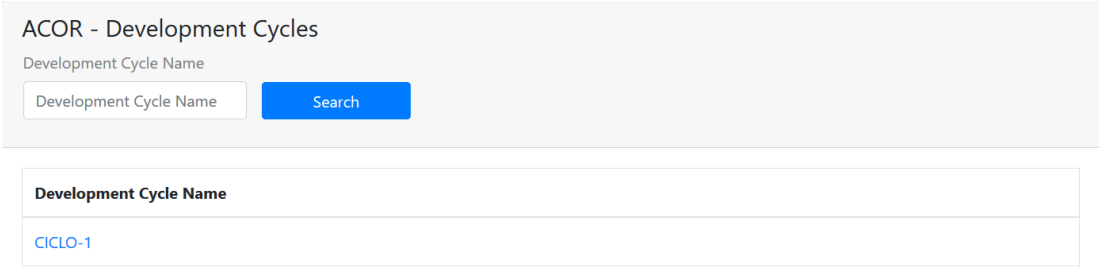


Figura 5.22: Búsqueda de Ciclos de Desarrollo

5.24 Búsqueda de Ramas Git

En esta página se puede ver una lista de las ramas Git para un proyecto con un repositorio de GitLab asignado. Para cada uno de los resultados se mostrará el nombre de la rama , que será también un enlace a la página de GitLab de dicha rama,y un cuadro de texto para filtrar los resultados.

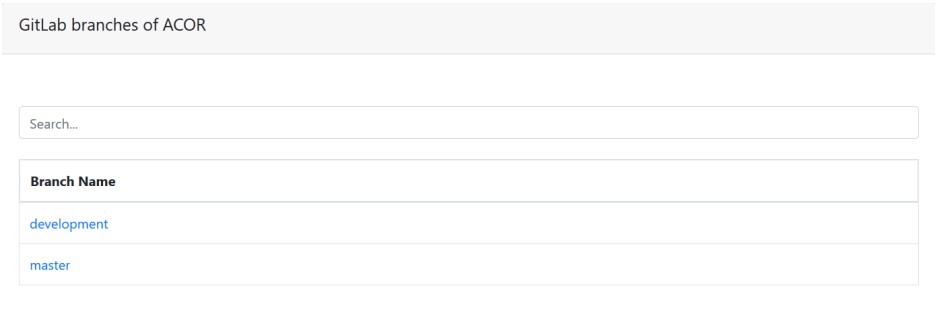


Figura 5.23: Ramas Git

5.25 Listado de Commits

Cuando la aplicación esta configurada para usar una cuenta de GitLab y cuando un proyecto tiene asignado un repositorio git, en la página de detalles de tarea (5.14) habrá un enlace que abrirá una ventana modal para ver los commits relacionados con dicha tarea, siendo estos los que en el mensaje contienen el identificador de la tarea. También hay otra versión en la cual se mostrarán los commits asociados con un ciclo de desarrollo, siendo estos los que se hacen sobre la rama que dicho ciclo tenga asociada.

Esta ventana contendrá una lista de commits, mostrándose de cada uno su identificador y su mensaje. El identificador será además un enlace a la página de GitLab con mas detalles sobre el commit.

En la parte superior además habrá un cuadro de texto con el cual se puede filtrar el resultado, tanto por identificador como por mensaje.

En la figura 5.24 se puede ver un ejemplo de esta página.

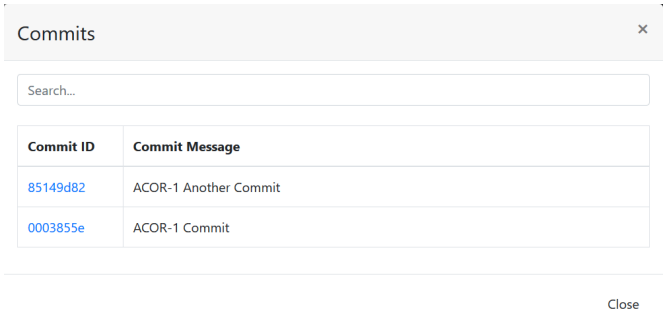


Figura 5.24: Listado de Commits

Capítulo 6

Conclusiones

En este capítulo se expondrá la situación final del trabajo y las lecciones aprendidas del desarrollo, junto con posibles cambios o desarrollos futuros.

6.1 Conclusiones del Proyecto

El objetivo de este proyecto era el análisis, diseño e implementación de una aplicación web para la gestión de proyectos utilizando la arquitectura MVC. Al finalizar se consigue una aplicación que nos permite gestionar proyectos, así como gestionar las tareas que se deben realizar en el proyecto y quien las debe realizar. Además también se ofrecen herramientas necesarias para que los lideres de proyectos puedan obtener mas información del desarrollo de las tareas y del tiempo dedicado a las mismas por los miembros del proyecto. Para la realización de este proyecto se decidió usar como lenguaje de programación Java, desarrollando principalmente con el framework Spring Boot, que ayuda a agilizar el proceso de configuración de distintos elementos para ser usados en la aplicación, como son la creación de los DAO con Spring Data o la creación de los servlets que atenderán a las peticiones HTTP con Spring MVC.

Se ha conseguido hacer una aplicación dinámica y responsive que se adecua a diferentes tamaños de pantalla como PC, tablets o móvil. En las páginas se ha intentado no sobrecargar con demasiada información, añadiendo ventanas modales para separar y así facilitar su comprensión.

Como se ha dicho con anterioridad, uno de los objetivos de realizar esta aplicación es obtener información de la gestión desde otro punto de vista. Con respecto a esto se ha conseguido, gracias al hecho de obtener requisitos, implementar y probar, entender de otra forma las necesidades para la gestión de proyectos.

Por último, cabe destacar el uso de diversas metodologías dadas durante la carrera, ya no solo PUDS, la metodología utilizada principalmente, si no también SCRUM y Kanban de las

cuales se han aprovecharon ciertos elementos.

6.2 Objetivos alcanzados

Los objetivos alcanzados en el momento en el que se finalizó el desarrollo son los siguientes:

- Se ha implementado una aplicación web con arquitectura MVC utilizando tecnologías Java.
- La aplicación permite crear y gestionar proyectos de forma sencilla.
- Se ha permite buscar proyectos, ciclos de desarrollo y tareas utilizando para ello una variedad de filtros.
- La aplicación realiza una gestión de usuarios correcta, limitando las acciones que un usuario puede realizar según su rol.
- Se incorpora la opción de conectar la aplicación con una cuenta de GitLab para poder asociar proyectos con repositorios de dicha cuenta así como ver y analizar el estado de los mismos sin abandonar la aplicación.
- Se ha internacionalizado la aplicación a Inglés, Castellano y Gallego.
- Se ha creado una interfaz adaptable para que sea sencilla y fácil de usar independientemente del dispositivo utilizado.

6.3 Líneas de trabajo futuras

Aunque la aplicación cumple con los elementos funcionales especificados en las fases iniciales, es posible incorporar todavía nuevas funcionalidades o mejoras. A continuación enumeramos algunas:

- Histórico de Cambios.
- Notificación de cambios en Proyecto y/o tareas por email.
- Implementación de subtareas.
- Permitir editar el estilo y formato de descripciones con el uso de lenguajes de marcado como MarkDown.
- Visualización en la web de elementos adjuntos, como archivos de texto o imágenes.

- Implementación de elementos de visualización del estado de proyecto, como tableros Kanban.
- Estados de tareas personalizados y configurables, a nivel de aplicación y proyecto.

Apéndices

Lista de acrónimos

AJAX Asynchronous JavaScript And XML (Javascript Asíncrono y XML).

API Application Programming Interface (Interfaz de Programación de Aplicaciones)

BD Base de Datos.

CD Continuous Delivery (Entrega Continua).

CI Continuous Integration (Integración Continua).

CRUD Create, Read, Update and Delete (Crear, Leer Actualizar y Eliminar).

DAO Data Access Object (Objeto de Acceso a Datos)

DTO Data Transfer Object (Objeto de Transferencia de Datos)

JVM Java Virtual Machine (Maquina Virtual de Java).

MVC Modelo Vista Controlador.

PUDS Proceso Unificado de Desarrollo Software.

URL Uniform Resource Locator (Localizador de Recursos Uniformes)

Glosario

Bytecode Código que generan los compiladores de ciertos lenguajes cómo Java o Erlang que es independiente de la maquina y se ejecuta a través de un intérprete.

Maquina Virtual de Java Java Virtual Machine (JVM) en inglés, es el interprete utilizado para la ejecución del bytecode generado al compilar código Java.

AJAX Asynchronous JavaScript And XML, Javascript Asíncrono y XML en Castellano, es una técnica usada en desarrollo web para la creación de páginas dinámicas. Se basa en el uso de Javascript para realizar llamadas asíncronas interactuando con el servidor en segundo plano, y luego, de ser necesario, se modifica la vista en función del resultado devuelto por el servicio. Muy usada en el desarrollo de aplicaciones de página única.

CSS Cascading Style Sheets, Hoja de Estilos en Cascada en Castellano, es un lenguaje para definir estilos para utilizar junto a lenguajes de marcado como HTML y que los navegadores pueden entender y cambiar como se visualizan los elementos de una página.

HTML HyperText Markup Language, lenguaje de marcas de hipertexto en castellano, es un lenguaje de marcado utilizado para definir la estructura de páginas WEB. Los navegadores web lo procesan y renderizan su contenido.

HTTP HyperText Transfer Protocol, Protocolo de Transferencia de Hipertexto en castellano, es un protocolo de transferencia de ficheros basado en cliente-servidor.

Integración Continua Continuous Integration (CI) en inglés, es una práctica utilizada en el desarrollo de software. Esta requiere que los desarrolladores actualicen frecuentemente un repositorio común con sus cambios, incluso varias veces al día. El contenido del repositorio se compilará en cada subida, permitiendo detectar errores de forma temprana.

Entrega Continua Continuous Delivery (CD) en inglés, es una práctica utilizada en el desarrollo de software. Se centra principalmente en hacer entregas habituales y automatizadas cada pocos cambios hechos en el código, permitiendo hacer de forma continua controles de calidad.

Métodos CRUD Métodos creados en una aplicación para realizar las acciones básicas contra base de datos, siendo estas crear, leer, actualizar y borrar elementos de una base de datos. Estas acciones son simples y sin filtros complejos.

Bibliografía

[1] A. Lesyuk, *Mastering Redmine*, 2013.

[2] M. M. Jon Loeliger, *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development.* O'Reilly Media, Inc, 2012.

[3] "GitLab." [En línea]. Disponible en: https://about.gitlab.com/

[4] Attlasian, "Sourcetree." [En línea]. Disponible en: https://www.sourcetreeapp.com/

[5] [En línea]. Disponible en: https://github.com/twbs/bootstrap

[6] "JQuery." [En línea]. Disponible en: https://jquery.com/

[7] "¿Qué es la tecnología java y para qué la necesito?" [En línea]. Disponible en: https://www.java.com/es/download/faq/whatis_java.xml

[8] "Spring Boot." [En línea]. Disponible en: https://spring.io/projects/spring-boot

[9] "Intellij IDEA." [En línea]. Disponible en: https://www.jetbrains.com/es-es/idea/

[10] yWorks, "Yed graph editor." [En línea]. Disponible en: https://www.yworks.com/products/yed

[11] "PlantUML." [En línea]. Disponible en: https://plantuml.com/es/

[12] "Taiga." [En línea]. Disponible en: https://taiga.io/

[13] I. Jacobson, G. Booch, and J. Rumbaugh, *El proceso unificado de desarrollo de software*, 1st ed. Addison Wesley, 2000.