UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación Web para la gestión interna en grupos Scout

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Cortés Reverón, Santiago

Tutor/a: Albert Albiol, Manuela

CURSO ACADÉMICO: 2022/2023

Resumen

La propuesta del TFG consiste en desarrollar una aplicación web orientada a monitores de grupos scout para que puedan de forma fácil gestionar procesos internos al grupo, automatizando algunos de estos procesos. Entre las funcionalidades de la aplicación, estará la generación de fichas de actividad, la gestión de autorizaciones para salidas y acampadas, o el seguimiento de tareas a realizar. Además, se definirán áreas personalizadas para cada monitor según los cargos que tenga en el grupo. Entre estas áreas estarán la gestión del inventario de material, la gestión de documentos internos del grupo o la gestión económica. El TFG se desarrollará utilizando el siguiente stack tecnológico: Javascript con React, HTML y TailwindCSS para la parte del Front-end y SpringBoot y Java para la parte del Back-end teniendo una base de datos NoSQL MongoDB para la persistencia.

Palabras clave: aplicación web, gestión de grupos scout, procesos

Abstract

The proposal of this TFG is to develop a web application aimed at scout group leaders, enabling them to easily manage internal group processes by automating some of these processes. The application will include various functionalities, such as activity form generation, authorization management for outings and camping trips, and task tracking. Additionally, personalized areas will be defined for each leader based on their roles within the group. These areas will encompass inventory management, internal document management, and financial management. This TFG will be developed using the following technological stack: JavaScript with React, HTML, and Tailwind for the front-end, and Spring Boot and Java for the back-end, with a NoSQL MongoDB database for data persistence.

Keywords : web application, scout group, management, processes

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

El presente Trabajo Fin de Grado (TFG) se centra en el desarrollo de una aplicación de gestión interna para grupos scout. Los grupos scout, reconocidos internacionalmente por su labor educativa y formativa en la juventud, requieren de herramientas efectivas para gestionar sus actividades, miembros, comunicaciones y recursos de manera eficiente.

El objetivo de este proyecto es investigar, diseñar, desarrollar y evaluar una solución innovadora para la gestión interna de grupos scout. Se busca mejorar la coordinación, comunicación y seguimiento de actividades dentro del grupo scout, ofreciendo una plataforma intuitiva y completa que facilite la organización y promueva una comunicación fluida entre los miembros.

A continuación, se expondrán los motivos que impulsaron la realización de este TFG, así como los objetivos que se persiguen y el impacto esperado que este proyecto pretende generar.

1.1. Motivación

Vivimos en un mundo cada vez más digitalizado en prácticamente todos los ámbitos de la sociedad. Esta digitalización ha ayudado a muchos sectores a modernizarse y agilizar sus procesos. Sin embargo, todavía existen ámbitos en los que la digitalización no ha tenido un impacto significativo y uno de ellos es el sector de los grupos scout. Los grupos scout son organizaciones educativas y de formación de jóvenes, donde la mayoría de las tareas que se desempeñan, ya sean administrativas o de gestión, se llevan a cabo de manera tradicional y manual.

Los grupos scout desempeñan una labor fundamental en la sociedad, brindando a los jóvenes de valores y respeto por la naturaleza, la solidaridad, la cultura, etc. Los grupos scout organizan a sus miembros (monitores) asignando roles, responsabilidades, y tareas. En ocasiones la coordinación de estos grupos se vuelve complicada y lenta.

Es por eso que surge la idea de este TFG de desarrollar una aplicación web para facilitar y agilizar la gestión interna de los grupos scout. Con este trabajo se pretende modernizar la forma de llevar a cabo los procesos internos de estas asociaciones, facilitando la comunicación entre sus miembros, la planificación de las actividades, la asignación de tareas y la gestión de recursos entre otras cosas. De esta manera se consigue que los grupos scout tengan una plataforma centralizada que permita optimizar el funcionamiento interno de dichos grupos.

Además, la digitalización y modernización en este ámbito también podría contribuir a que muchos jóvenes que pertenecen al mundo scout se interesen por continuar en estas organizaciones, y aquellos que aún no pertenezcan a ninguna pueden verse interesados por unirse a estas. Esto se debe a que los jóvenes de hoy en día están cada vez más

familiarizados con lo digital, por lo que ofrecerles una solución tecnológica resulta atractivo para que participen en los grupos scout.

En conclusión, el desarrollo de una aplicación web para los grupos scout surge de la necesidad de agilizar y modernizar la gestión de estas entidades para que puedan adaptarse y prosperar en el entorno digital en el que nos encontramos. Además, pretende eliminar la carga manual que conllevan las tareas de estos grupos, haciendo que los monitores puedan dedicar menos tiempo a las gestiones del grupo y más tiempo de calidad a la formación y crecimiento personal de los jóvenes.

1.2. Objetivos

El objetivo principal del TFG es desarrollar una aplicación web que permita gestionar de manera sencilla y ágil los procesos internos de grupos scouts. Esta aplicación estará disponible para los usuarios finales a través de la web con el nombre “WeKraal”.

Para lograr el objetivo principal se proponen los siguientes subobjetivos:

- Investigar las necesidades de los grupos scout para comprender a qué desafíos se enfrentan los grupos en su gestión interna.
- Implementar funcionalidades que faciliten a los grupos scout la gestión y programación interna en su día a día. Un aspecto importante será promover el trabajo en equipo y la participación activa que en muchos grupos scout se está perdiendo.
- Diseñar una interfaz de usuario intuitiva que proporcione a los usuarios una experiencia visual agradable y fácil de usar. Esto implica realizar un diseño claro, utilizar colores y elementos gráficos coherentes con la identidad del grupo scout y proporcionar una navegación fluida
- Garantizar la seguridad y privacidad de los datos: que la aplicación cumpla con los estándares de seguridad y protección de datos mediante protocolos de autenticación y autorización.

Además del objetivo principal de desarrollar la aplicación web, a nivel personal, se plantea como objetivo el aprendizaje de nuevas tecnologías: (1) utilización de SpringBoot, (2) familiarización con API REST, y (3) trabajar con nuevos tipos de bases de datos como lo es MongoDB

1.3. Impacto esperado

Esta aplicación supondrá una mejora de la eficiencia en el proceso de gestión interna de los grupos scout, así como una mayor facilidad para llevar a cabo dicha gestión. Al proporcionar herramientas y funcionalidades específicas para la programación de actividades y la gestión de procesos internos y tareas entre otras, se espera que los monitores puedan llevar a cabo estas tareas de manera más rápida, sencilla y precisa.

1.4. Estructura de la memoria

- Capítulo 1 – Introducción: en este capítulo se pone en contexto la problemática y se introduce la temática del proyecto, así como sus objetivos e impacto a generar que se desea
- Capítulo 2 – Estado del arte: Analiza las distintas opciones disponibles en el mercado que tienen un propósito similar o idéntico a "WeKraal" y evalúa las ventajas y desventajas de cada una de ellas.
- Capítulo 3 – Análisis del problema: Se lleva a cabo una planificación inicial para tener una comprensión clara de los actores involucrados, las acciones que llevarán a cabo para interactuar con la aplicación web y la forma en que las ejecutarán.
- Capítulo 4 – Diseño de la solución: Se describe detalladamente el proceso seguido durante el desarrollo de la aplicación, dividiendo cada una de sus partes en secciones independientes donde se explica de manera más específica las tecnologías utilizadas y cómo se implementaron en cada caso.
- Capítulo 5 – Desarrollo de la solución: desarrolla el proceso llevado a cabo durante el desarrollo para alcanzar el producto deseado.
- Capítulo 6 – Pruebas: Presenta los resultados de pruebas realizadas tanto a usuarios como técnicas, mostrando desde la perspectiva del cliente final la calidad y eficacia de la plataforma.
- Capítulo 7 – Conclusiones: En el último capítulo se realiza una evaluación objetiva del proyecto, abordando los diversos problemas y limitaciones encontrados durante el desarrollo, así como sugiriendo posibles cambios que podrían aplicarse en el futuro para mejorar la aplicación.

2. Estado del arte

El propósito de este capítulo es llevar a cabo un análisis del estado actual de las aplicaciones existentes que podrían utilizarse para la de gestión interna de grupos scout. En el apartado 2.1 se presentan aplicaciones de gestión que se usan en grandes organizaciones y en el apartado 2.2 se propone WeKraal como solución a las debilidades y carencias de estas aplicaciones.

2.1. Aplicaciones de gestión de organizaciones

Hoy en día son varias las herramientas que permiten gestionar y acelerar la gestión de una organización. El problema es que todas estas están orientadas a empresas, ya sean grandes o pequeñas, donde la gestión se realiza de una manera muy diferente. A continuación, se analizan algunas de estas herramientas.

Monday.

Monday [1][1] es una plataforma de gestión de proyectos y colaboración en equipo basada en la nube. Permite a los usuarios organizar y supervisar tareas, proyectos y flujos de trabajo de manera visual e intuitiva. Con su interfaz fácil de usar y personalizable, los usuarios pueden crear tableros Kanban, establecer plazos, asignar responsabilidades, compartir archivos y comunicarse con el equipo en tiempo real. Monday.com ofrece una amplia gama de funciones y herramientas, como seguimiento de tiempo, automatizaciones, integraciones con otras aplicaciones y visualización de datos en forma de gráficos y tablas.

[1] Página web oficial de Monday: https://monday.com/lang/es

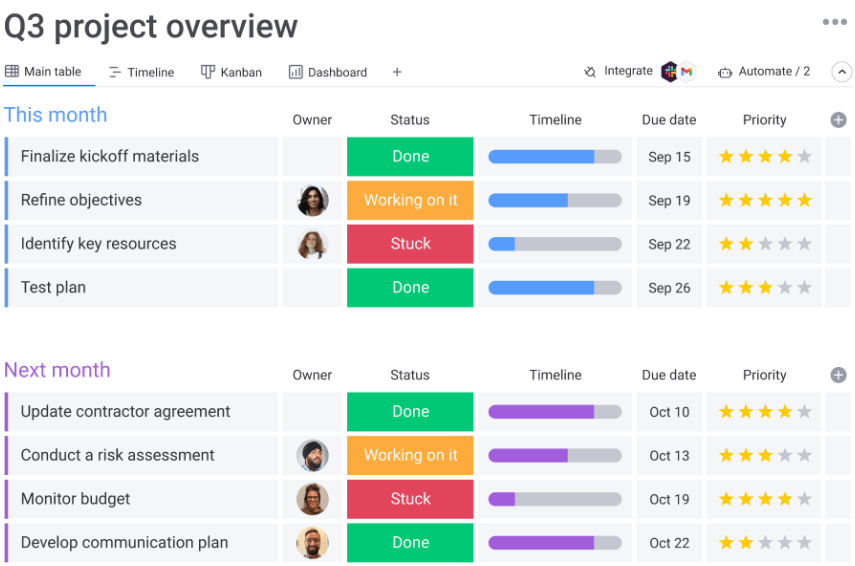


Ilustración 1. Panel principal de Monday

En la Ilustración 1. se puede observar que Monday ofrece en su pantalla principal los diferentes proyectos que hay creados en la aplicación y se están llevando a cabo en la empresa y muestra la planificación por etapas, en este caso meses, y el estado de sus tareas. Aunque sigue una estructura minimalista, quizás es demasiado simple para ser la pantalla principal de Monday cuando entras a la aplicación.

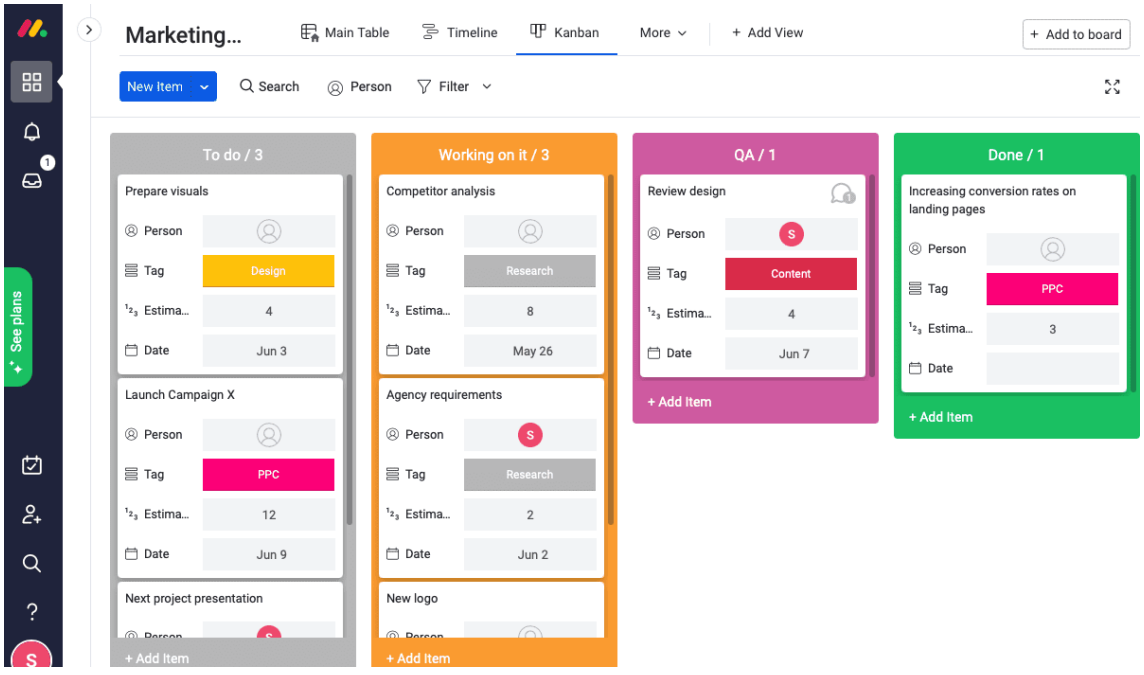


Ilustración 2. Kanban de Monday

En la Ilustración 2. de Monday se muestra el Kanban que hay implementado en la aplicación, que no deja de ser una manera distinta de ver la tabla que hemos visto en la anterior figura. La interfaz no destaca por su encanto y además se hace uso de una disposición de los elementos del Kanban un poco rara. En un proyecto es importante saber que tareas son prioritarias y en el Kanban de Monday no vemos que se haga ningún tipo de jerarquía entre las tareas.

Asana.

Asana [2] es un software de gestión de proyectos que funciona como un espacio de trabajo virtual para organizar y colaborar en tareas, ya sea en equipo o para proyectos individuales. Proporciona una plataforma centralizada donde los usuarios pueden crear y asignar tareas, establecer plazos, realizar seguimiento del progreso y comunicarse con otros miembros del equipo.

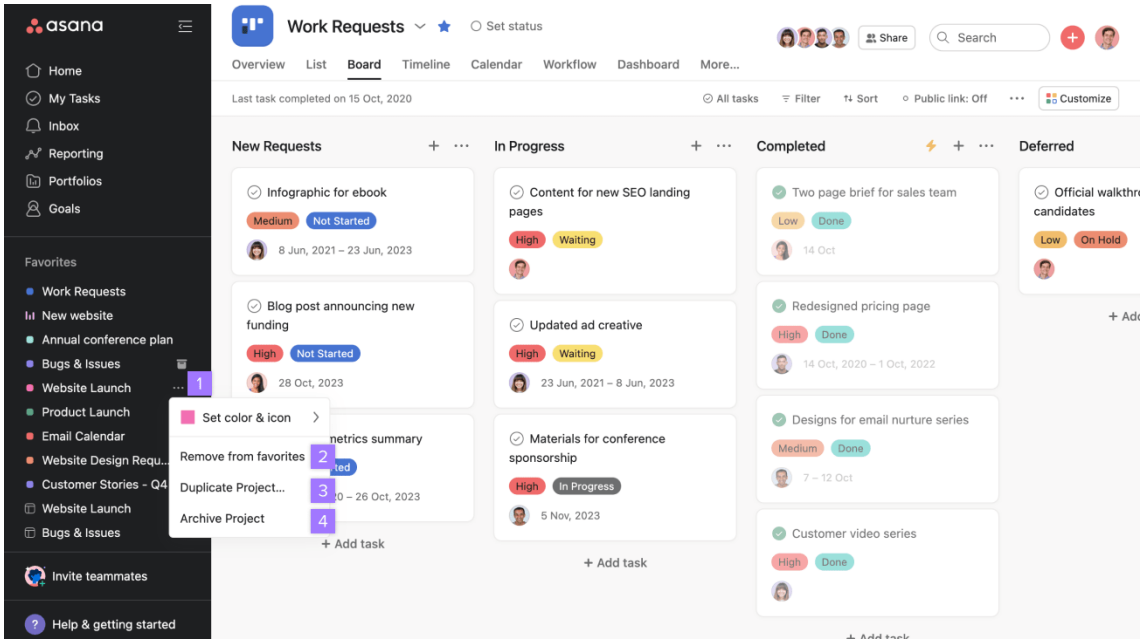


Ilustración 3. Kanban de Asana

En la Ilustración 3. podemos observar el tablero Kanban de Asana, el cual es similar al de Monday pero cambiando temas de estilo. Este es más completo que el de Monday, sin embargo, la pantalla parece estar sobrecargada con tantas etiquetas de distintos colores y las tareas no traen una descripción de acceso rápido, sino que tienes que meterte en la tarea para leer la información de esta.

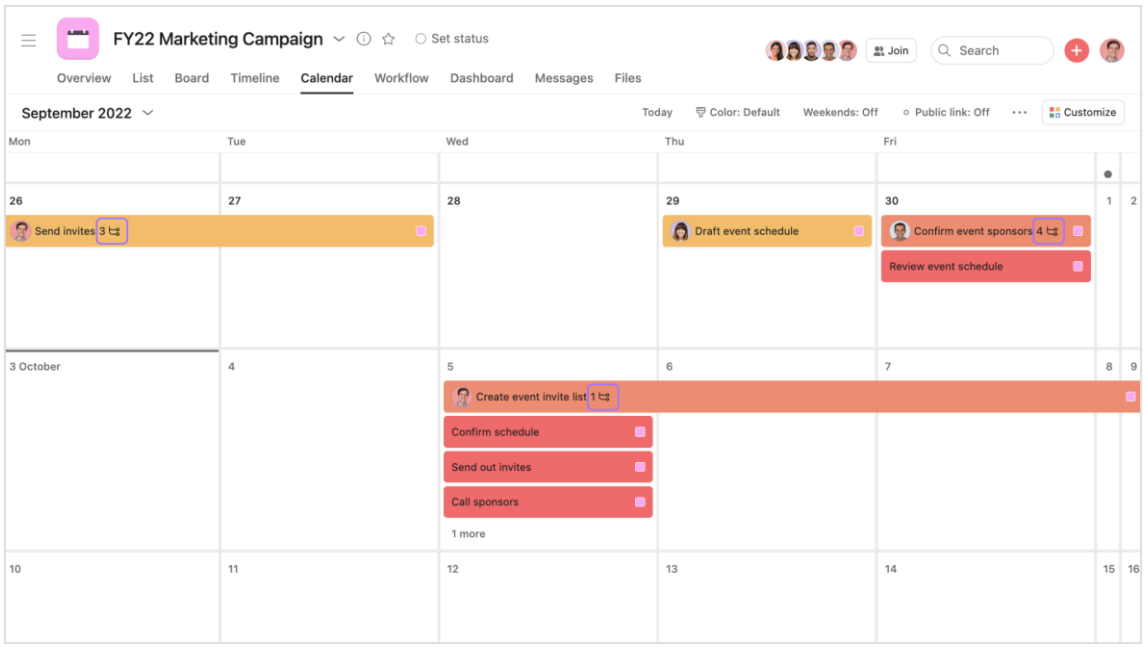


Ilustración 4. Calendario de Asana

En la Ilustración 4. vemos el calendario que integra la aplicación de Asana, una interfaz bonita pero el calendario depende de las demás funciones de Asana, es decir, es una herramienta dentro de la aplicación que es solo para visualizar eventos en el calendario. Carece de funciones como añadir eventos ya que se basa en los proyectos y tareas activas que existen en la aplicación.

Las dos aplicaciones presentadas ofrecen funcionalidades para la gestionar de una mejor forma los proyectos y los procesos que se llevan a cabo en una organización. El inconveniente de estas aplicaciones es que son muy generales, orientadas a empresas grandes y pequeñas o equipos de trabajo y no cumplen con las necesidades que necesitan los grupos scout.

2.2. Propuesta

Tras haber presentado y analizado las aplicaciones similares a “WeKraal”, existen varias razones por la cual se hace necesario el desarrollo de una aplicación para grupos scout, en este caso “WeKraal”.

“WeKraal” es una aplicación de gestión especialmente diseñada para satisfacer las necesidades específicas de los grupos scout. A diferencia de las herramientas como Asana y Monday, las cuales son muy generales, “WeKraal” se enfoca en proporcionar funcionalidades personalizadas y adaptadas a los procesos internos y requerimientos particulares de los grupos scout.

Esta aplicación ofrece características especializadas, como la creación de fichas de actividad para los eventos como acampadas o reuniones, un módulo de registro y

seguimiento de progresiones individuales de los miembros, la capacidad de compartir y acceder a documentos relevantes y el seguimiento de proyectos y tareas mediante un Kanban y un calendario. Estas funcionalidades están directamente adaptadas a las necesidades y flujos de trabajo específicos de los grupos scout, brindando una experiencia de gestión más eficiente.

Además, el gran punto fuerte de “WeKraal” es que fusiona varias de las funcionalidades que ofrece Monday y Asana como el calendario y el Kanban y además añade dos nuevas funcionalidades como subida de archivos y creación de actas como ofrecería Google Workspace.

En resumen, la creación de “WeKraal” como una aplicación de gestión destinada exclusivamente a los grupos scout se sustenta en su enfoque especializado, capacidad de adaptación y características personalizadas diseñadas para enfrentar los retos particulares que enfrentan estos grupos.

3. Análisis del problema

En este capítulo se realiza un análisis del problema que se quiere abordar con la aplicación a desarrollar. En la primera sección, se presenta la metodología que se va a utilizar para el desarrollo de la aplicación, y en la segunda sección, se describen los requisitos identificados.

3.1. Metodología aplicada

En un proyecto software de estas dimensiones se hace necesario aplicar prácticas y procesos para ayudar a los desarrolladores y a la gente involucrada en el proyecto a gestionar y desarrollar el producto de manera eficiente. La metodología por lo general nos proporciona una guía para el equipo de desarrollo para llevar a cabo una planificación y entrega con éxito del producto.

Para este proyecto en concreto se ha seguido una metodología ágil, basada en la adaptabilidad y entrega continua del producto. La metodología ágil que se ha escogido de todo el abanico existente ha sido la SCRUM [3]. SCRUM se centra en un desarrollo iterativo e incremental donde el producto se va entregando de manera continua. Al ser un proyecto individual en el cual no existe un cliente final todavía y no se trabaja en equipo, se ha adaptado un poco a estas necesidades.

Los principios fundamentales que se han cogido y adaptado de esta metodología han sido los siguientes:

- Backlog: El backlog es una lista de requisitos, características y/o funcionalidades las cuales tienen una prioridad dentro de esta. Estos elementos son los que se pretenden aplicar en el proyecto hasta su fase de finalización, aunque se tiene que mantener una revisión continua de este componente pues es susceptible a modificaciones.
- Sprints: Este componente de la metodología SCRUM es una iteración de tiempo en el ciclo de vida del proyecto durante el cual se lleva a cabo una mejora del producto. Estos suelen tener una duración máxima de 4 semanas y es en esta fase donde se seleccionan los requisitos del backlog para el Sprint.
  Aunque para el desarrollo de WeKraal no se ha usado una herramienta concreta para registrar estos Sprints, se ha realizado en libreta una planificación de requisitos que tienen que estar presentes tras finalizar cada semana. Se han realizado Sprints de 1 semana con el fin de implementar funcionalidades pequeñas de manera incremental a lo largo de cada semana, ya que este periodo de tiempo coincidía con la reunión de revisión con quien podríamos decir que ha sido el Product Owner de este proyecto, el tutor del TFG.

A pesar de que ha sido difícil seguir una metodología de trabajo al ser un proyecto individual, se ha intentado en todo momento que las prácticas ejecutadas para el desarrollo de la aplicación se asemejasen los más posible a la metodología ágil SCRUM.

Sumado a esta metodología seguida se ha pensado en llevar un control de versiones. El control de versiones nos ayuda a registrar y tener rastreados todos los cambios que se realizan en la aplicación cada vez que se implementa una funcionalidad. Esto nos facilita diferenciar las entregas del producto mediante un registro e historial de cambios.

Una ventaja de llevar un control de las versiones es que se puede retroceder a versiones anteriores en caso de que la versión actual del proyecto sea insostenible o nos otorgue errores que no se pueden solucionar o incluso en caso de que se haya planteado mal una fase del proyecto y en mitad del desarrollo se vea la necesidad de volver a una versión anterior.

Para llevar a cabo esta práctica se va a hacer uso de la herramienta Git para facilitar la gestión y el control del código de la aplicación. Se han creado 2 repositorios, uno para el front-end y otro para el back-end ya que son aplicaciones diferentes y se desarrollan en distintos IDE (Entornos de desarrollo). De esta manera se aíslan los cambios y podemos realizar cambios en el front-end sin necesidad de modificar o afectar al back-end.

3.2. Especificación de requisitos

En este apartado se va a presentar una descripción de los que buscamos que la aplicación web de “WeKraal” haga. Esta descripción establece los estándares para el diseño y creación de la aplicación. Para ello, en primer lugar, se presentan de forma informal, esto es, en lenguaje natural, los requisitos identificados. A continuación, se elaboran los casos de uso.

3.2.1. Requisitos funcionales

Tras realizar un análisis de las funcionalidades que ofrecen algunas aplicaciones de gestión, y realizar una toma de contacto con algunos miembros de grupos scout que participan en su gestión interna, se determinaron las principales necesidades que debía satisfacer la aplicación. Estas son las siguientes:

- Permitir la creación de la programación de las actividades que van a realizarse durante el año mediante un formulario y con la ayuda de un calendario, teniendo en cuenta salidas y acampadas, las cuales también tendrán una programación asignada.

- Gestionar proyectos y tareas con la ayuda de un tablero Kanban, cuyas columnas representan la fase en la que se encuentra la tarea. El creador de la tarea puede asignarle responsables, darle prioridad y establecer una descripción de lo que ha de llevarse a cabo.

- Cada usuario debe tener acceso a un área personalizada que se adaptará a sus cargos dentro del grupo. Dependiendo de dicho cargo tendrá acceso a unas

funcionalidades concretas relevantes para ejercer sus responsabilidades dentro del grupo.

- Posibilidad de subir documentos como fichas de actividad.

- Calendario integrado con funcionalidades de visualización y programación de actividades. Los usuarios podrán ver de forma clara y organizada las fechas y horarios de las actividades planificadas, así como agregar nuevas actividades directamente desde el calendario

- Panel principal con aspectos generales del grupo scout, donde los usuarios podrán ver información resumida sobre el estado actual del grupo, como el número de miembros, próximas actividades, tareas pendientes y eventos importantes.

- Área personal para cada usuario según sus cargos dentro del grupo scout. Cada usuario tendrá acceso a un espacio personalizado en la aplicación donde podrán ver y administrar información específica relacionada con sus responsabilidades dentro del grupo. Esto incluirá tareas asignadas, proyectos en los que están involucrados.

A partir de las necesidades que se identificaron, se han elaborado los requisitos funcionales de la aplicación. La aplicación de "WeKraal" se compone de los siguientes requisitos y casos de uso asociados a las necesidades previamente descritas:

1. **Cuenta de usuario**
   - CU1. Registro
   - CU2. Iniciar sesión
   - CU3. Cerrar sesión

2. **Gestión de Kanban**
   - CU4. Crear tarea
   - CU5. Controlar flujo de las tareas
   - CU6. Ver tareas
   - CU7. Eliminar tareas

3. **Calendario**
   - CU8. Crear evento
   - CU9. Eliminar evento

4. **Actas**
   - CU10. Redactar acta
   - CU11. Crear acta
   - CU12. Guardar acta
   - CU13. Eliminar acta

5. **Programaciones**
   - CU14. Crear ficha

- CU15. Asignar ficha

Estos requisitos funcionales se podrían representar mediante un diagrama de casos de uso como el de la Ilustración 5.:

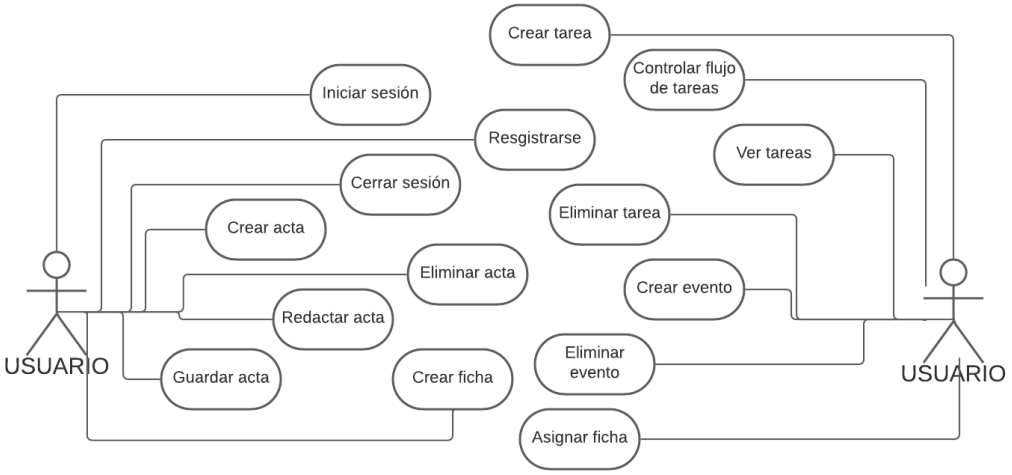


Ilustración 5. Diagrama de casos de uso

A continuación, se detalla cada caso de uso a través de una plantilla. Las plantillas utilizadas son plantillas estándar para casos de uso:

Nombre	CU1. Registro
Actores involucrados	Usuario y Administrador.
Objetivo	Registrarse en la aplicación.
Precondiciones	El administrador debe haber mandado la invitación al grupo para poder registrarse.
Escenario principal	1. El usuario entra en la aplicación y selecciona “Registrarse”. 2. Introduce sus credenciales, proporcionadas por la invitación del administrador 3. Se validan los datos
Postcondiciones	La aplicación redirige al usuario ya registrado al panel principal.

Nombre	CU2. Login

Actores involucrados	Usuario
Objetivo	Iniciar sesión en la aplicación.
Precondiciones	Poseer una cuenta de “WeKraal”
Escenario principal	1. El usuario entra en la aplicación y selecciona “Iniciar sesión”. 2. Introduce sus credenciales, nombre de usuario y contraseña 3. Se validan los datos
Postcondiciones	La aplicación redirige al usuario al panel principal.

Nombre	CU3. Cerrar sesión
Actores involucrados	Usuario
Objetivo	Cerrar la sesión iniciada en la aplicación
Precondiciones	Se debe poseer una cuenta registrada en el sistema y se debe haber iniciado sesión y estar en alguna de las ventanas de la aplicación.
Escenario principal	1. El usuario hace clic en el botón de cerrar sesión del header de la aplicación 2. Confirma si de verdad quiere cerrar su sesión en la aplicación 3. El usuario acepta cerrar su sesión.
Postcondiciones	La aplicación redirige al usuario a la ventana de login y registro

Nombre	CU4. Crear tarea
Actores involucrados	Usuario
Objetivo	Crear una tarea que añadir en el Kanban
Precondiciones	Estar iniciado sesión en la aplicación
Escenario principal	1. El usuario hace clic en el apartado “Tareas” de la barra lateral de navegación 2. Es redirigido a la ventana de tareas de la aplicación donde aparece el Kanban. 3. El usuario hace clic en el botón de “Nueva tarea” que hay en el tablero de tareas “ToDo”.

	4. Se muestra una ventana modal con un formulario para rellenar los datos de la nueva tarea. 5. El usuario introduce los datos de la tarea a crear. 6. Se validan los datos y se crea la tarea.
Postcondiciones	La aplicación cierra la ventana modal y muestra en el tablero la nueva tarea creada.

Nombre	CU5. Controlar flujo de tareas
Actores involucrados	Usuario
Objetivo	Organizar y clasificar las tareas según la prioridad que se considere, además de establecer el estado en que se encuentra la tarea.
Precondiciones	El número de tareas debe ser mayor que 0
Escenario principal	1. El usuario clica y mantiene el clic sobre una tarea que desee modificar de estado o prioridad. 2. El usuario arrastra la tarea sobre el mismo tablero o sobre otro tablero que desea modificar. 3. Una vez tiene la tarea posicionada donde va a ser su nuevo tablero o prioridad suelta el clic y la tarea se asigna a un nuevo tablero y se establece una nueva prioridad.
Postcondiciones	Se actualiza la prioridad de la tarea modificada y se actualizan los tableros según las tareas nuevas tareas introducidas o quitadas del tablero.

Nombre	CU6. Ver tareas
Actores involucrados	Usuario
Objetivo	Visualizar el progreso de las tareas y el estado en que se encuentran
Precondiciones	-
Escenario principal	1. El usuario hace clic en el apartado “Tareas” de la barra lateral de navegación 2. Es redirigido a la ventana de tareas de la aplicación donde aparece el Kanban. 3. El usuario puede ahora visualizar las tareas que hay y el estado en que se encuentran para poder tomar decisiones como priorizar algunas tareas o pasarlas de estado
Postcondiciones	-
Nombre	CU7. Crear evento

Actores involucrados	Usuario
Objetivo	Crear un evento para la lista de eventos de la ronda
Precondiciones	-
Escenario principal	1. El usuario hace clic en el apartado “Calendario” de la barra lateral de navegación de la aplicación. 2. El usuario es redirigido a la ventana de eventos de la aplicación. 3. El usuario hace clic en el botón de “Nuevo Evento”. 4. Se abre una ventana modal en la que aparece un formulario para rellenar con los datos del nuevo evento. 5. El usuario introduce los datos del evento y le da a “crear evento”. 6. Se validan los datos y se crea el nuevo evento.
Postcondiciones	La aplicación cierra la ventana modal y muestra en la lista de eventos el nuevo evento creado.

Nombre	CU8. Eliminar tarea
Actores involucrados	Administrador
Objetivo	Eliminar tareas, tanto aquellas que ya hayan finalizado como aquellas que no se vean necesarias o hayan surgido de una equivocación.
Precondiciones	Estar iniciado sesión como administrador
Escenario principal	1. El usuario hace clic en el apartado “Tareas” de la barra lateral de navegación 2. Es redirigido a la ventana de tareas de la aplicación donde aparece el Kanban. 3. El administrador clica en el botón de eliminar tarea que aparece en la esquina superior derecha del elemento que representa la tarea 4. Se abre una ventana modal en la que se indica si se quiere borrar la tarea. 5. El administrador clica el botón de confirmar de la ventana modal.
Postcondiciones	La aplicación cierra la ventana modal y muestra un mensaje de “Tarea eliminada correctamente”. Además, se actualiza el tablero.

Nombre	CU9. Eliminar acta
Actores involucrados	Administrador

Objetivo	Eliminar actas.
Precondiciones	Estar iniciado sesión como administrador
Escenario principal	1. El usuario hace clic en el apartado “Actas” de la barra lateral de navegación 2. Es redirigido a la ventana donde se muestran las actas que hay creadas 3. El administrador clica en el botón de eliminar acta que aparece en la esquina superior derecha del elemento que representa el acta 4. Se abre una ventana modal en la que se indica si se quiere borrar la tarea. 5. El administrador clica el botón de confirmar de la ventana modal.
Postcondiciones	La aplicación cierra la ventana modal y muestra un mensaje de “Acta eliminada correctamente”. Además, se actualiza la lista de actas.

Nombre	CU10. Eliminar evento
Actores involucrados	Administrador
Objetivo	Eliminar eventos. (Por ejemplo, aquellos que ya ha vencido su fecha)
Precondiciones	Estar iniciado sesión como administrador
Escenario principal	1. El usuario hace clic en el apartado “Calendario” de la barra lateral de navegación 2. Es redirigido a la ventana donde se muestran los eventos que hay en la ronda. 3. El administrador clica en el botón de eliminar evento que aparece en el elemento que representa el evento. 4. Se abre una ventana modal en la que se indica si se quiere borrar el evento. 5. El administrador clica el botón de confirmar de la ventana modal.
Postcondiciones	La aplicación cierra la ventana modal y muestra un mensaje de “Evento eliminado correctamente”. Además, se actualiza la lista de eventos.

Nombre	CU11. Escribir acta
Actores involucrados	Usuario
Objetivo	Redactar actas de consejo para el grupo
Precondiciones	Estar iniciado sesión en la aplicación Estar en el editor de texto de la ventana de actas

Escenario principal	1. El usuario redacta el acta usando las funcionalidades del editor de texto. 2. El usuario revisa y verifica el contenido del acta 3. Se aprueba el acta
Postcondiciones	-

Nombre	CU12. Crear acta
Actores involucrados	Usuario
Objetivo	Crear actas de consejo para el grupo
Precondiciones	Estar iniciado sesión en la aplicación
Escenario principal	1. El usuario hace clic en el apartado “Actas” de la barra lateral de navegación 2. Es redirigido a la ventana donde se muestran las actas que hay creadas 3. El usuario clica en el botón de crear acta. 4. Es redirigido a una ventana con un editor de texto para crear el acta 5. El usuario le da un nombre al acta y redacta el acta usando las funcionalidades del editor de texto.
Postcondiciones	Se crea el acta y es redirigido a la ventana de “Actas”

Nombre	CU13. Guardar acta
Actores involucrados	Usuario
Objetivo	Guardar actas de consejo para el grupo tras su edición
Precondiciones	Estar iniciado sesión en la aplicación
Escenario principal	1. El usuario hace clic en el apartado “Actas” de la barra lateral de navegación 2. Es redirigido a la ventana donde se muestran las actas que hay creadas 3. El usuario clica en el acta que quiere modificar para guardar. 4. Es redirigido a una ventana con un editor de texto para editar y guardar el acta 5. El usuario modifica el acta y posteriormente hace clic en el botón de guardar acta.

Postcondiciones	Se guarda el acta y es redirigido a la ventana de “Actas”

Nombre	CU14. Crear ficha
Actores involucrados	Usuario
Objetivo	Crear fichas de programación para las actividades y eventos
Precondiciones	Estar iniciado sesión en la aplicación
Escenario principal	1. El usuario hace clic en el apartado “Fichas” de la barra lateral de navegación 2. Es redirigido a la ventana donde se muestra un formulario para crear fichas de programación. 3. El usuario rellena los campos. 4. El usuario hace clic en el botón de crear acta de la parte superior izquierda de la ventana. 5. El sistema valida que los datos introducidos por el usuario sean correctos.
Postcondiciones	Se crea la ficha

Nombre	CU14. Asignar ficha
Actores involucrados	Usuario
Objetivo	Asignar fichas de programación a los eventos
Precondiciones	Estar iniciado sesión en la aplicación
Escenario principal	1. El usuario hace clic en el apartado “Eventos” de la barra lateral de navegación 2. Es redirigido a la ventana donde se muestra una lista de eventos como reuniones, acampadas y salidas. 3. El usuario hace clic en el evento al cual quiere asignarle una ficha de programación. 4. Se abre una ventana modal con una lista de las fichas disponibles para asignar 5. El usuario selecciona las fichas que quiere asignar al evento 6. El sistema verifica que la acción se ha hecho correctamente 7. Se muestra un mensaje por pantalla indicando que se han asignado con éxito las fichas de programación.
Postcondiciones	Se redirige al usuario a la ventana de eventos

3.2.2. Requisitos no funcionales

Los requisitos no funcionales se refieren a las características y atributos del sistema que no están directamente relacionados con las funciones específicas que este debe realizar. Estos requisitos determinan el comportamiento y rendimiento del sistema en términos de calidad, rendimiento, seguridad, usabilidad y otros aspectos importantes.

Seguridad: la web debe protegerse del acceso no autorizado.

- Integridad: la información no se debe alterar de manera no autorizada.
- Autenticación: la aplicación debe controlar el acceso de los usuarios según su rol y credenciales.
- Autorización: asignar los diferentes tipos de acceso según el rol del usuario.

Rendimiento: la aplicación debe soportar 2000 accesos sin que esta reduzca su rendimiento.

- Eficiencia: ha de realizar las funciones de manera rápida y con el menor consumo posible.
- Tiempo de respuesta: debe ser rápido.

Mantenibilidad: la web ha de ser fácilmente mantenible y soportar actualizaciones.

- Legibilidad del código: el código debe ser entendible por cualquier usuario, para que, en un futuro, este pueda ser mantenido sin ningún problema.
- Flexibilidad: debe ser flexible de cara a actualizaciones futuras y poder añadir funcionalidades nuevas que no comprometan la mantenibilidad del código.
- Modularidad: El código debe estar organizado por módulos, haciendo uso de buenas prácticas para facilitar su mantenimiento y adición de funcionalidades nuevas.

Fiabilidad: la aplicación debe cumplir con los requisitos del usuario.

- Tolerancia a fallos: debe poder funcionar sin errores ni interrupciones.
- Disponibilidad: debe estar disponible cuando sea necesario.

Usabilidad: la aplicación debe ser fácil de usar y comprender.

- Facilidad de aprendizaje: la curva de aprendizaje deber ser sencilla para que cualquier usuario pueda usarla, hasta los menos experimentados.
- Facilidad de memorizar: los usuarios deben poder recordar con facilidad como ejercer todas las funcionalidades de la aplicación sin que le suponga un esfuerzo a los usuarios.
- Retroalimentación: el usuario debe ser notificado sobre las acciones que se llevan a cabo en la aplicación, así como el estado en el que se encuentra la misma.

- Atractivo visual: la interfaz debe ser atractiva y estar bien estructurada para que los usuarios se sientan cómodos usándola y lo puedan hacer de manera intuitiva.

4. Diseño de la solución

El diseño es una fase indispensable en el desarrollo de una aplicación, ya sea una página web, una aplicación de escritorio o una aplicación móvil. En este capítulo se presenta una descripción del proceso de diseño seguido en el desarrollo de la solución.

4.1. Arquitectura

La arquitectura de una aplicación define cómo se conectan los componentes que la componen, de qué manera interactúan y cómo tratan la información. La arquitectura influye de manera significativa en los requisitos no funcionales, ya que de esta depende que la aplicación sea escalable, eficiente, mantenible y segura.

En el caso de una aplicación web existen varios patrones arquitectónicos estándar para su desarrollo. Entre estos patrones destacan el modelo MVC, la arquitectura de 3 capas, la arquitectura orientada a microservicios o la típica arquitectura cliente-servidor. Para elegir entre una de estas se han de considerar varios factores como los requisitos funcionales y no funcionales ya que cada modelo de arquitectura tiene sus ventajas y desventajas.

La arquitectura escogida para este proyecto ha sido la de cliente-servidor [4]. Esta arquitectura se basa en la separación de funcionalidades entre los dos componentes principales: el cliente y el servidor.

La comunicación entre estos componentes principales se realiza mediante protocolos de red como HTTP. El cliente envía las solicitudes al servidor y este le responde en base a la petición del cliente. Esta manera de comunicación permite que el cliente y el servidor se ejecuten en sitios diferentes, haciendo el sistema más modular y mantenible, facilitando la escalabilidad en un futuro. Este aspecto ha sido determinante en la elección de la arquitectura pues en un futuro se pretende complementar la aplicación web con una móvil para las familias. De esta forma, se aprovechan las características de esta arquitectura, pues pueden existir varios clientes, en este caso la aplicación web y la aplicación móvil, y un solo servidor que mantenga la lógica y el manejo de las peticiones.

En WeKraal se ha implementado la arquitectura siguiendo el patrón BFF [5], cuyas siglas significan “back-end for front-end”. Este diseño se basa en separar y especializar el back-end (servidor) para un front-end específico (cliente). De esta forma optimizamos tanto la experiencia del usuario como el rendimiento de cada back-end, pues cada front-end requerirá del uso de unos servicios y datos específicos. En el caso de WeKraal el diagrama que representa esta arquitectura se puede ver en la Ilustración 6.

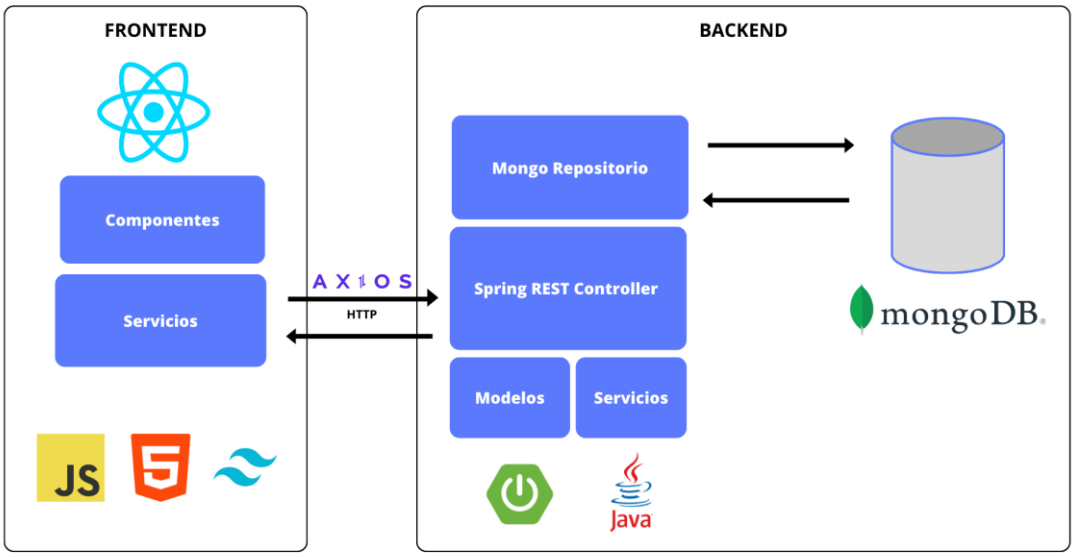


Ilustración 6. Arquitectura BFF

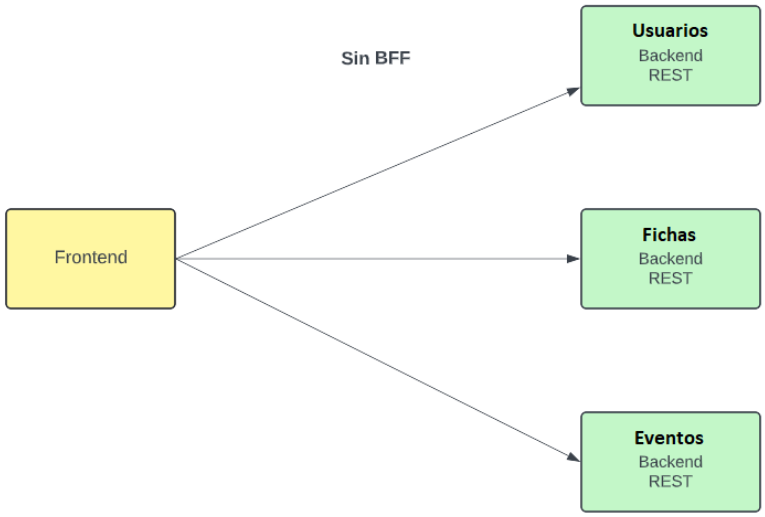


Ilustración 7. Arquitectura sin BFF

En la Ilustración 7. podemos observar que en el caso en que la arquitectura no estuviera formada por una capa intermedia, el front-end se comunicaría directamente con la capa de datos. Esto no es recomendable, pues rompe el principio de responsabilidad única [6] (que consiste en que una clase, método o módulo debe tener solamente una responsabilidad, es decir, solamente una razón para cambiar). En el caso del front-end su responsabilidad es mostrar la información al usuario en forma de vistas, por lo que no debería comunicarse directamente con el back-end.

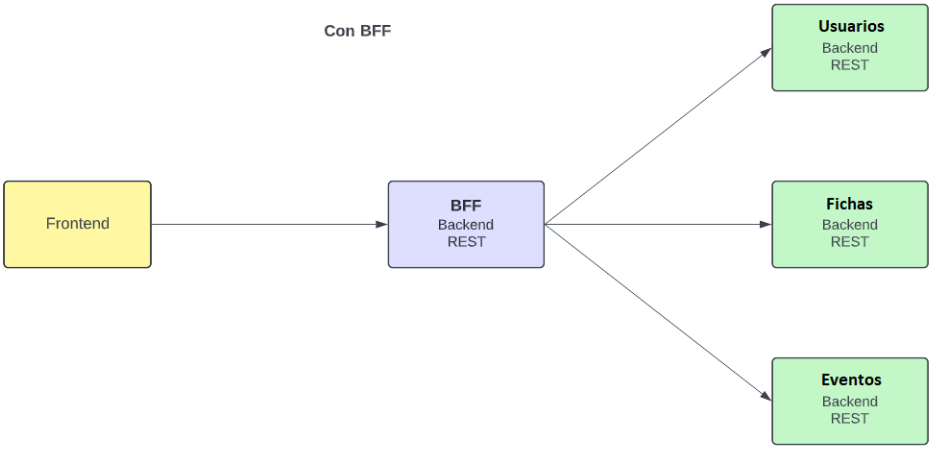


Ilustración 8. Arquitectura con BFF

Como se puede observar en la Ilustración 8., básicamente el BFF es una capa entre el front-end y el back-end, que dota al sistema de una mayor seguridad y desacoplamiento entre los componentes. Además, respeta el principio de responsabilidad única, pues no será el front-end quien se encargue de hacer las peticiones al back-end directamente ya que esa no es su función. De esta forma, hacemos que el front-end únicamente se encargue de mostrar los datos en un formato de interfaz para el usuario.

Las principales características de este patrón de diseño de arquitectura son las siguientes:

1. **Aislamiento y especialización**: En nuestro caso cada front-end que se diseña tendrá asociado su propio back-end el cual se encargará de manejar las peticiones realizadas por el front-end y retornar los datos necesarios haciendo uso de los servicios definidos. Es así como tenemos un mayor control sobre las funcionalidades específicas de cada front-end.
2. **Microservicios**: En este tipo de arquitectura los BFF podrían ser considerados como microservicios a los cuales acceden sus front-ends específicos. Esto se alinea con la arquitectura de microservicios ya que cada BFF puede ser mantenido y escalado de manera independiente a los demás.
3. **Experiencia del usuario**: Al tener los servicios adaptados a cada front-end se puede mejorar la experiencia del usuario, ofreciéndole funcionalidades específicas que se ajusten a cada plataforma.
4. **Escalabilidad**: Dado que poseemos un BFF para cada front-end se consigue una mejora de la escalabilidad pues cada uno de ellos puede ser desarrollado y desplegado sin influir sobre el resto de los componentes.

Podemos encontrar alguna característica más como la adaptación de datos y la composición de datos. La adaptación de datos se entiende como la adaptación que puede realizar el BFF sobre los datos que retorna al front-end. En el caso de que no hubiese una capa intermedia entre el front-end y el back-end, el primero se comunicaría directamente con el segundo y este le devolvería los datos en el formato establecido en el back-end por

lo que el front-end deberá interpretarlos y ajustarlo a sus necesidades. Sin embargo, con la capa intermedia como lo es el BFF este puede solicitar los datos al back-end según las peticiones del front-end y realizar las adaptaciones necesarias para retornarlo al front-end en un formato optimizado o incluso añadiendo información adicional. La segunda característica que se ha mencionado es la composición de servicios, esta consiste en que el BFF pueda necesitar comunicarse con algún servicio externo o varios internos para componer la respuesta para el front-end. El BFF puede actuar como un orquestador, realizando solicitudes entre los diferentes servicios internos y externos para componer la respuesta antes de ser enviada al front-end. Si la capa intermedia no existiera el front-end tendría que hacer varias peticiones al back-end y una vez obtenidas las respuestas, moldearlas según sus necesidades, con el BFF se elimina esta complejidad y es el mismo quien puede manejar estas solicitudes que implican el uso y la comunicación con servicios tanto internos como externos.

Una vez conocida la arquitectura que se ha propuesto para la aplicación de WeKraal se va a explicar cada elemento que conforma la arquitectura.

4.2. Componentes de la aplicación

Esta arquitectura, como ya se ha explicado, consta de 2 componentes principales, el front-end y el back-end. A continuación, se van a explicar cada una de estas partes del sistema de manera más detallada, incluyendo un diagrama de clases que muestra cómo se relacionan los datos en la aplicación.

4.2.1. Front-end

El front-end [7] es la parte visible de una web o aplicación que se relaciona con todo lo que ven las personas, como el diseño, los colores, las tipografías, las imágenes, los botones, las animaciones y los efectos. Su función es la de que el usuario interactúe con el sistema, y es por eso por lo que es tan importante ya que debe cumplir con los estándares de usabilidad y estética.

En la aplicación de WeKraal, para el front-end se ha optado por hacer uso de la biblioteca de JavaScript más utilizada y estandarizada, React. React proporciona herramientas para crear interfaces de usuario y sigue el paradigma de la programación declarativa, esto significa que te centras en cómo se ha de mostrar la interfaz, la cual va cambiando con la actualización de los estados de los componentes. Más adelante se explicará más detalladamente cómo funciona React en WeKraal. Gracias a React se pueden crear componentes los cuales a su vez pueden estar formados por componentes más pequeños. Esto permite la reutilización de código, dotando de una organización clara y modular la aplicación. Los componentes en React además mantienen un estado que nos da flexibilidad a la hora de mostrar la información. Una cosa muy importante que tiene React también es que solo actualiza los elementos del DOM afectados, lo que se traduce en un mayor rendimiento de la aplicación.

Es por eso por lo que el front-end se ha construido con React ya que es ideal para diseñar interfaces web con una gran modularidad y rendimiento.

4.2.2. Back-end

El back-end [8] de un sistema se podría definir como todo aquello que no ve el usuario. Este se encarga de procesar las solicitudes del usuario, que en nuestro caso vienen manejadas por el front-end. El back-end se encarga de gestionar los datos y proporciona mecanismos de acceso a los datos para almacenarlos o recuperarlos.

En este proyecto, como ya se ha mencionado antes se ha hecho uso de la arquitectura cliente-servidor implementado por un patrón de diseño llamado back-end for front-end. Este es la capa intermedia que se ha implementado entre la capa de presentación y la capa de acceso a datos. Esta se encarga de manejar las solicitudes que el usuario realiza sobre la interfaz gráfica de la aplicación retornándole una respuesta con los datos solicitados por el usuario. Además, en el caso de WeKraal también es el BFF el encargado de la autenticación y autorización de usuarios.

Para esta aplicación se ha optado por hacer uso de SpringBoot[2] para la implementación del BFF. SpringBoot [9] es un framework que facilita la creación de aplicaciones web y microservicios gracias a tres funcionalidades principales como la configuración automática, el enfoque obstinado de la configuración y la capacidad de creación de aplicaciones independientes. Estas características se explicarán más adelante.

En cuanto a la base de datos se trata de una base de datos NoSQL, concretamente MongoDB. Se ha optado por esta opción por ser más flexible y rápida. Además, se ha escogido para agilizar el proceso de desarrollo ya que el esquema de los datos ha ido cambiando con frecuencia. También ha influenciado en la elección de este tipo de base de datos el trabajo futuro, pues las bases de datos no relacionales tienen una gran escalabilidad horizontal a medida que aumente el número de usuarios y datos en la aplicación, que sumado a los algoritmos de indexación que poseen estas bases de datos pueden ser una mejor opción que las SQL.

Por último, para entender mejor como se relacionan los objetos en la base de datos se muestra a continuación un diagrama de clases que representa dicha estructura:

[2] Página web oficial de SpringBoot https://spring.io/projects/spring-boot

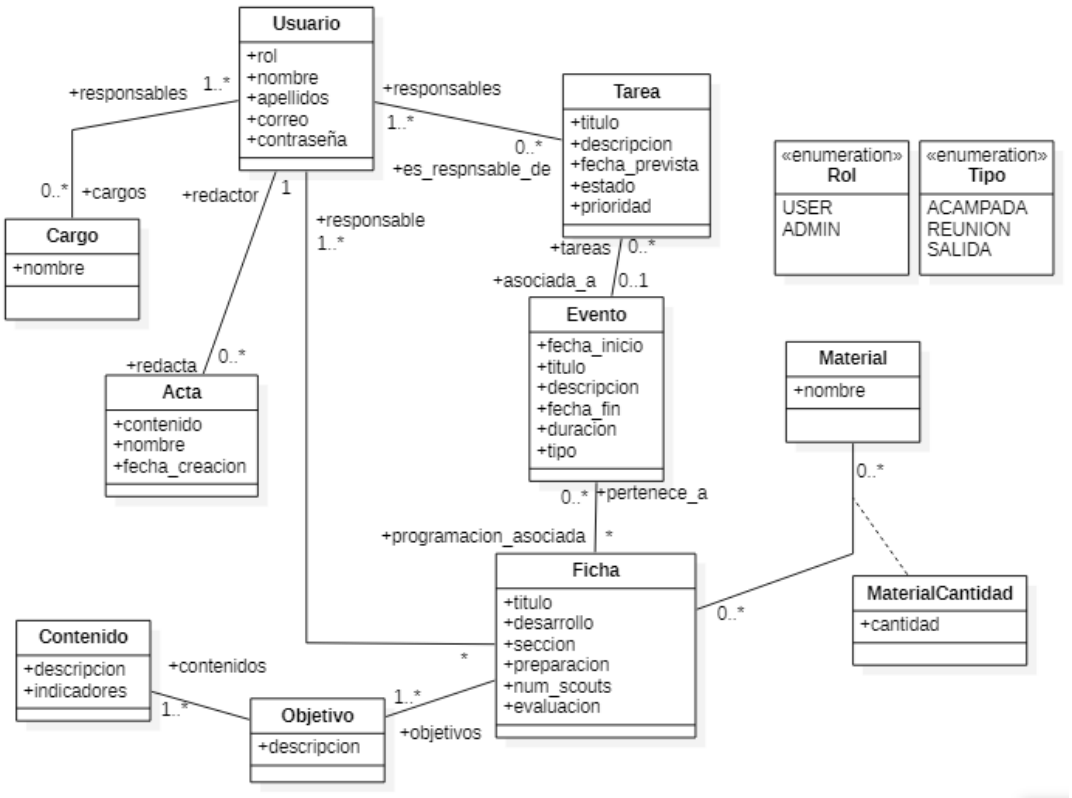


Ilustración 9. Diagrama de clases

4.3. Diseño de la IGU

En este apartado se van a mostrar las interfaces de usuario de la aplicación desarrollada.

Las capturas de pantalla proporcionan una visión general de las ventanas de la aplicación, de los menús y de los elementos interactivos que componen las funcionalidades de la aplicación.

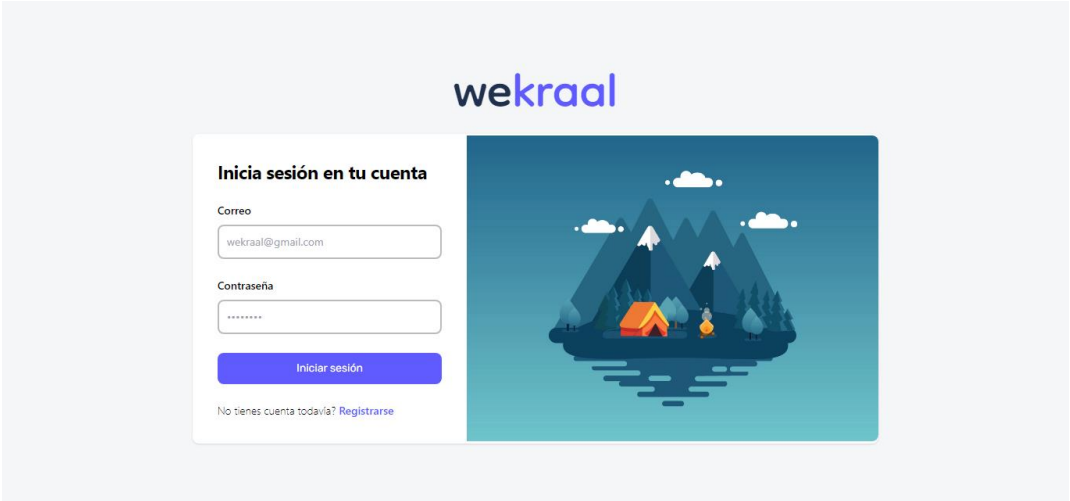


Ilustración 10. Inicio de sesión de la aplicación

En la Ilustración 10. se muestra la primera pantalla que observamos al entrar a la aplicación y es la que se corresponde con el inicio de sesión. La ventana de registro es semejante solo que cambia la parte izquierda del formulario, en el cual se piden datos adicionales como el nombre y los apellidos.

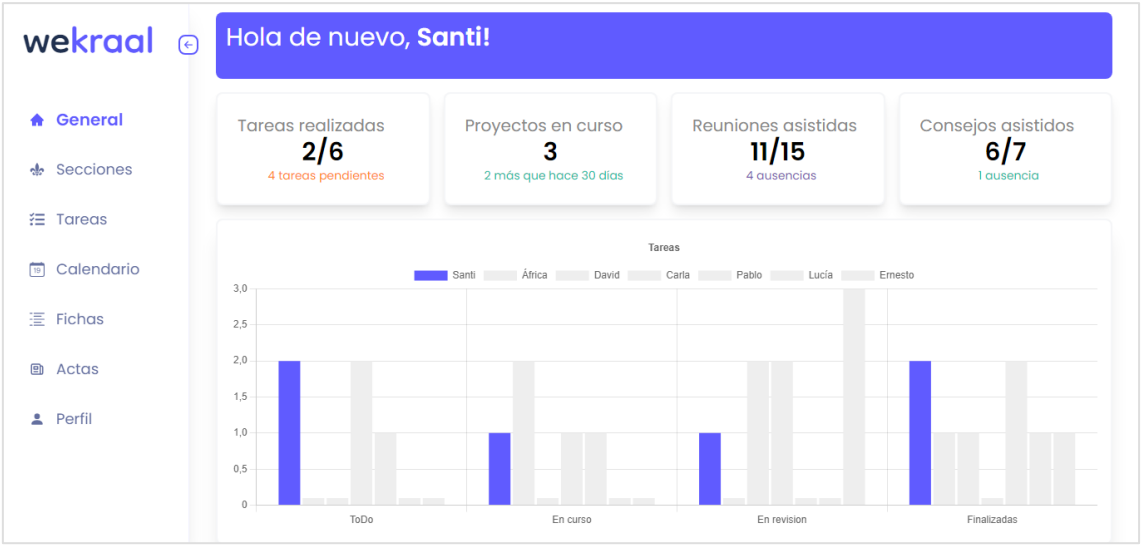


Ilustración 11. Pantalla principal de WeKraal

Esta es la pantalla principal de WeKraal (Ilustración 11.), en esta se pretende mostrar el estado de la gestión interna del grupo de manera visual e individual. Podemos observar que hay una serie de ítems que nos dan información sobre cuantas tareas hemos realizado, a cuantos consejos hemos asistido o a cuantas reuniones. Además, se ofrece un gráfico de barras para mostrar el estado de las tareas en las que estas asignado y como se encuentran en este momento las tareas.

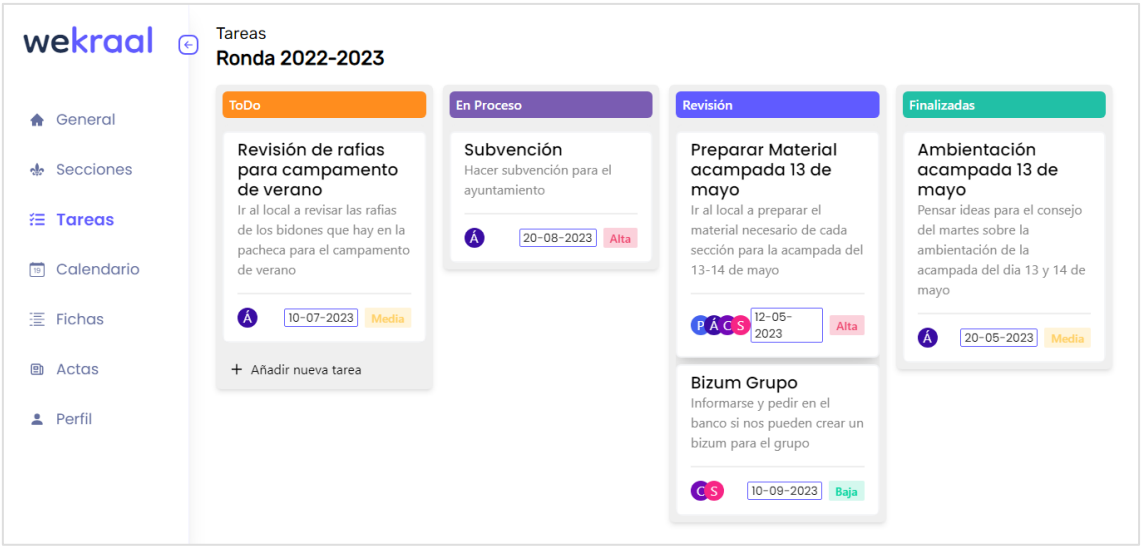


Ilustración 12. Pantalla de Kanban

La Ilustración 12. es la pantalla del Kanban donde podemos observar las tareas y su estado. Podemos ver que hay tareas pendientes, tareas en proceso, en revisión o finalizadas. También observamos que estas tareas están asignadas a varios usuarios de la aplicación y que tienen una fecha límite para estar completadas. Además, vemos que tienen una prioridad asignada para que los usuarios prioricen unas tareas frente a otras.

Podemos observar que en esa misma ventana podemos crear nuevas tareas. Este formulario para crear tareas aparece cuando se hace clic en el botón de nueva tarea y tiene el siguiente aspecto:

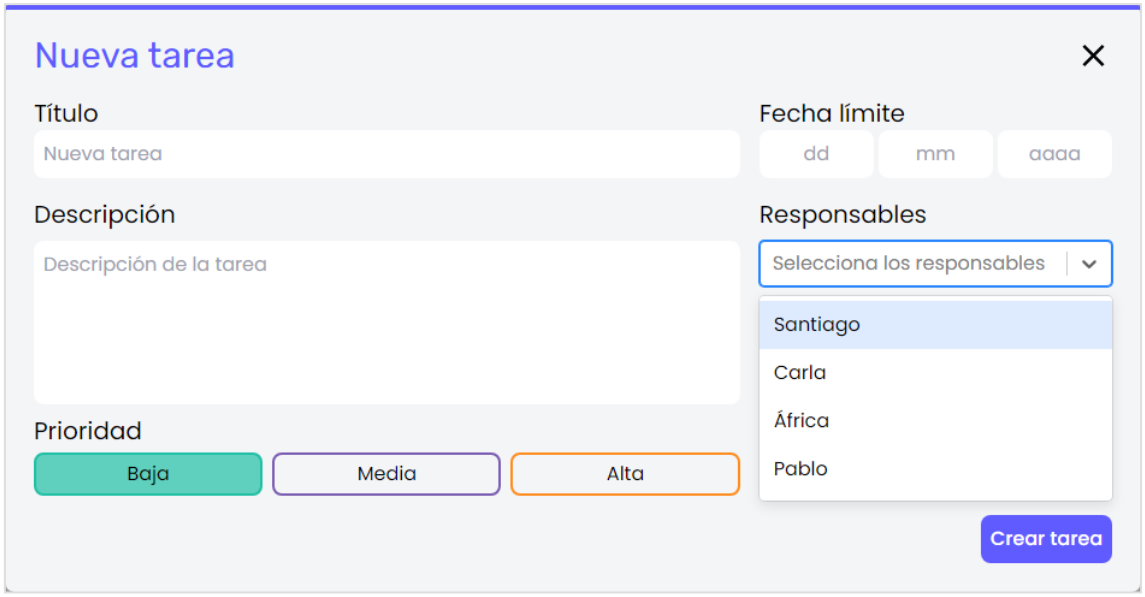


Ilustración 13. Formulario de creación de tareas

En este formulario de la Ilustración 13. observamos todos los campos necesarios a rellenar para crear tareas.

La Ilustración 14. representa la funcionalidad de un calendario, pero en formato de lista de eventos. De esta manera con una lista podemos acceder a todos los eventos próximos en el grupo scout como reuniones, acampadas o salidas de una manera más visual.

Ilustración 14. Pantalla de eventos de la aplicación

En la Ilustración 14. observamos que al hacer clic en un ítem de la lista de eventos se nos muestra a la derecha un componente con la información relevante del evento: titulo, descripción, material necesario...

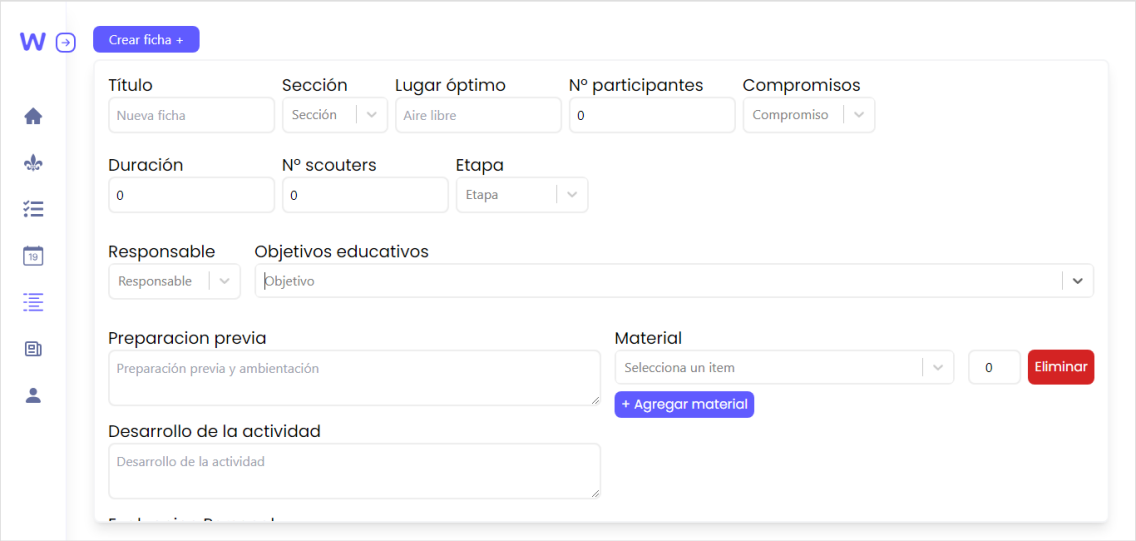


Ilustración 15. Pantalla de creación de fichas de programación

En la Ilustración 15. se muestra un formulario de creación de fichas de programación para las actividades. Estas fichas de programación son muy importantes dentro de un grupo scout pues estas se usan para programar acampadas, salidas, reuniones y campamentos. Son muy útiles porque reflejan como se ha de realizar una actividad en caso de no estar presente el responsable de esta, por lo que era necesario incluir una funcionalidad que permitiese crear fichas de programación para asignárselas a las actividades.

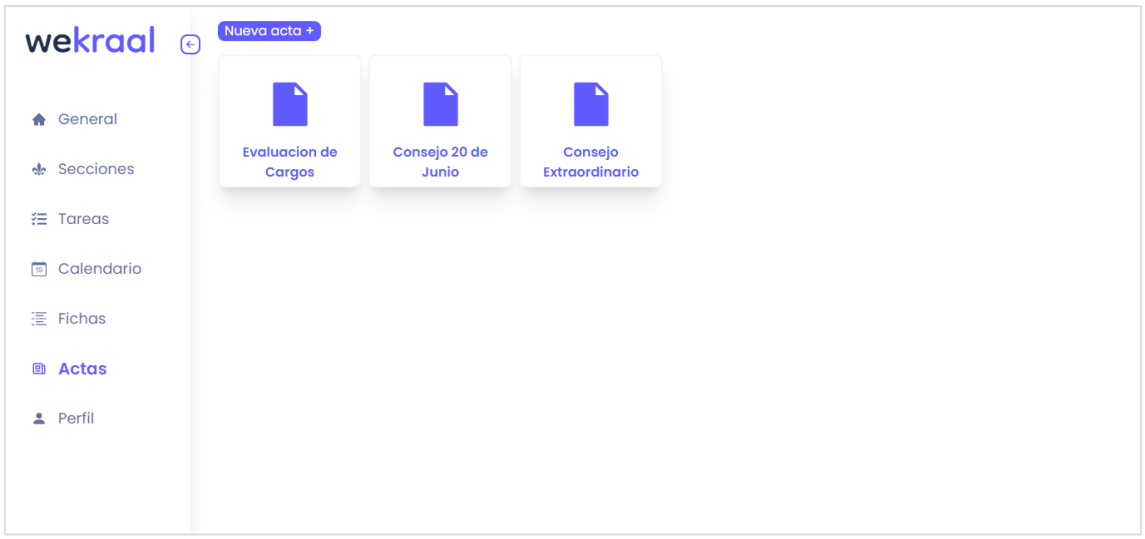


Ilustración 16. Pantalla de actas de la aplicación

En esta pantalla, que podemos observar en la Ilustración 16., vemos las actas que tenemos disponibles para editar o visualizar y además podemos crear nuevas. Esta funcionalidad es muy útil en grupos scout ya que necesitan tomar actas de los consejos para que todo lo que se hable de importancia quede por escrito. Cuando hacemos clic en uno de estos ítems que representan las actas accedemos al editor para hacer los cambios oportunos o visualizarla:

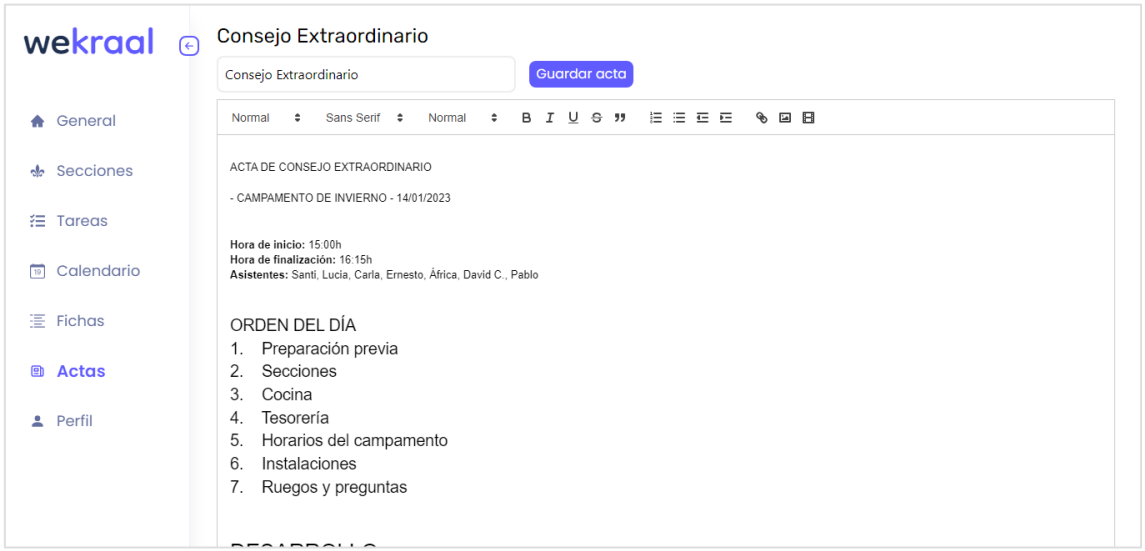


Ilustración 17. Pantalla de edición y guardado de Actas

En la Ilustración 17. podemos observar el editor de texto integrado en la pantalla de edición y guardado de tareas. Es aquí donde los usuarios editan y guardan las actas de los consejos y reuniones que se hacen para tratar temas importantes.

En resumen, las capturas de pantalla presentadas muestran el resultado de un esfuerzo dedicado en el diseño e implementación de interfaces intuitivas y atractivas para la

aplicación. Estas interfaces constituyen una parte fundamental de la experiencia de usuario, brindando una forma visualmente agradable y fácil de interactuar con la aplicación.

5. Implementación

La fase de implementación de una aplicación implica transformar el diseño y los requisitos previamente establecidos en un producto funcional. Durante esta etapa, se construye y codifica el software, se crean los diversos componentes, se implementan las funcionalidades necesarias y se lleva a cabo la integración de los diferentes módulos. A continuación, en este capítulo se va a explicar qué tecnologías se han usado para el desarrollo de la aplicación, qué patrones de diseño se han implementado y otros aspectos relevantes durante el desarrollo, como la introducción de seguridad y generación de documentación de la aplicación.

5.1. Tecnologías utilizadas

En esta sección, se explica el stack tecnológico propuesto para la implementación de la aplicación. El stack tecnológico se refiere a la combinación de herramientas y lenguajes de programación seleccionados para el desarrollo de un sistema. Para la selección de herramientas, se han considerado las fortalezas y debilidades de cada una, así como su interoperabilidad, ya que estas características afectarán el funcionamiento adecuado de la aplicación. Por lo tanto, es crucial que estas herramientas estén en consonancia con los requisitos definidos, ya que serán las responsables de proporcionar un rendimiento óptimo, agilidad en el desarrollo y flexibilidad para crear interfaces y mejorar la experiencia del usuario, entre otras funcionalidades.

5.1.1. Tecnologías para el Front-end

Para el front-end se han pensado herramientas punteras, de gran popularidad y sobre todo que permitan un desarrollo ágil de la aplicación. Al ser una aplicación web la gran mayoría de herramientas son robustas y están muy estandarizadas ya que no existen grandes alternativas para web que no incluyan el desarrollo con JavaScript, HTML y CSS. Es por eso por lo que se han escogido las herramientas más versátiles y utilizadas para desarrollar con estos lenguajes de programación:

- **React**
  Esta herramienta nos permite diseñar y crear componentes reutilizables que representan partes de la interfaz de usuario. Es una biblioteca de JavaScript que nos proporciona funcionalidades como un Virtual DOM para optimizar la actualización de la interfaz de usuario y renderización del lado del cliente y del servidor para brindar flexibilidad a la hora de renderizar y entregar los componentes al usuario.

React[3] se basa en HTML y JavaScript, aunque tiene buena compatibilidad con otros lenguajes como TypeScript, además da flexibilidad para usar JSX, una extensión propia de React para escribir conjuntamente HTML y JavaScript.

Esta biblioteca nos ofrece características como los *hooks*, estos son una pieza clave para el desarrollo ágil de la aplicación pues nos permiten controlar el estado de los componentes. Sin ayuda de los *hooks* sería necesario convertir el estado en componentes de la clase, es por eso que estos nos permiten hacer los componentes más legibles, mantenibles y fáciles de usar.
Los *hooks* más utilizados en React y los más presentes en WeKraal son los siguientes:

useState: Este *hook* nos permite agregarle estado a un componente. Nos permite establecer un valor al estado y operaciones de actualización de estado. Nuestro componente se comportará de manera diferente según el estado que almacene este *hook* en un determinado momento. Ejemplo:

```
1  const [selected, setSelected] = useState("General");
```

Ilustración 18. Uso de useState

En este ejemplo de la Ilustración 18. se muestra como asignarle un estado a un componente, en este caso se trata del componente *SideBar.js,* el cual necesita tener un estado para saber que opción de la barra lateral esta seleccionado. Necesita saberlo ya que en función de que elemento este seleccionado le añadirá un estilo u otro.

Ilustración 19. Funcionalidad useState

Como se puede observar en la Ilustración 19., gracias a que tenemos un estado en el componente para representar que elemento de la barra lateral esta

[3] Página web oficial de React https://es.react.dev/learn

seleccionado, podemos asignarle un estilo diferente a este. En este caso el estado asignado es “Tareas”.

useEffect: Este *hook* nos permite ejecutar funciones en el proceso de renderización de los componentes, también se usa para manejar la suscripción a eventos. Se usa para evitar cargar datos antes del renderizado de un componente, de esta forma conseguimos un control más preciso sobre el flujo de datos y la interacción con el ciclo de vida de los componentes.

Para la funcionalidad de la suscripción a eventos, este *hook* tiene un Array de dependencias que hace que cuando estas variables o componentes pertenecientes al Array cambien, se ejecute las funciones definidas en el useEffect.

```
1  useEffect(() => {
2      getInventarios().then((inventarios) => {
3        setInventarios(inventarios.data);
4      });
5      console.log(inventarios);
6    }, []);
```

Ilustración 20. Uso de useEffect

En la Ilustración 20. se muestra un ejemplo de useEffect, este ejecuta una función “getInventarios()” cuando el componente se renderiza. En este caso no existe ninguna dependencia en el Array que vemos al final del *hook,* pero si quisiéramos que se ejecutara el useEffect tras el cambio de una variable “material” solo habría que incluirla en el Array.

- **TailwindCSS**
  Esta herramienta es un framework de CSS que nos permite estilizar de manera rápida y fácil los componentes de React. A diferencia de otros como Bootstrap, que se centran en la estilización de componentes predefinidos, TailwindCSS[4] nos permite definir clases de utilidad que se aplican sobre la etiqueta HTML donde está siendo definida. De esta forma conseguimos poder estilizar cualquier marcado de HTML y nos da una gran flexibilidad a la hora de estilizar los componentes de la interfaz de usuario.
  Tailwind ofrece también características avanzadas que podemos definir en un archivo de configuración, lo que permite adaptar el framework a los requisitos de la aplicación. Además, ya que la estilización se define en un atributo *className*

[4] Página web oficial de TailwindCSS https://tailwindcss.com/

de la etiqueta HTML evitamos el exceso de estilos no utilizados como pasaría si usáramos CSS. A continuación, podemos ver un ejemplo de cómo se usa Tailwind y por qué se ha escogido frente a CSS:

```
<div className="bg-[#f4f5f7] p-6 relative rounded-lg font-texto m-2">
        <span className="absolute top-0 bg-scout-azul h-1 w-full rounded-t-lg inset-x-0" />
```

Ilustración 21. Uso de TailwindCSS

Podemos observar en la Ilustración 21. que el uso de este framework es intuitivo y sencillo de usar, pues usa un lenguaje claro que se asemeja a la sintaxis de CSS. Como vemos, hay elementos del className que parecen no ser propios de la sintaxis de CSS como “font-texto” y “bg-scout-azul” y es que Tailwind, como se ha mencionado anteriormente nos permite definir características y estilos propios en un archivo de configuración.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    screens: {
      sm: "576px",
      md: "960px",
      lg: "1440px",
    },
    fontFamily: {
      wekraal: ["Dongle", "sans-serif"],
      titulo: ["Rubik", "sans-serif"],
      texto: ["Poppins", "sans-serif"],
    },
    extend: {},
    colors: {
      scout: {
        background: "#f1f1ff",
        texto: "#20304f",
        azul: "#605BFF",
        naranja: "#ff8d1e",
        celeste: "#21c0a6",
        morado: {
          claro: "#d5cce6",
          oscuro: "#7a5cb2",
        },
      },
```

Ilustración 22. Archivo de configuración de TailwindCSS

En la Ilustración 22. Se muestra el archivo de configuración de Tailwind, en el podemos definir características como el estilo de las fuentes usadas para el texto de la aplicación, colores creados por nosotros y otras configuraciones específicas del proyecto. El archivo de configuración de Tailwind CSS se denomina **tailwind.config.js** y se encuentra en la raíz del proyecto.

Gracias a este archivo podemos personalizar muchos elementos y hacer la experiencia de estilización del front-end más personalizada, satisfactoria y sencilla

5.1.2. Tecnologías para el Back-end

Se ha utilizado SpringBoot como framework de desarrollo para el back-end y MongoDB como base de datos. Estas herramientas en conjunto permiten crear back-ends robustos y escalables con un entorno fácil de mantener y eficiente.

SpringBoot

Esta herramienta es un framework basado en Java Spring que nos proporciona una infraestructura lista para su uso y variedad de características que nos ayudan y simplifican el proceso de configuración y desarrollo de aplicaciones de Java.

Se ha escogido como herramienta principal para el desarrollo del BFF (Back-end for front-end) ya mencionado anteriormente, debido a que proporciona una gran cantidad de utilidades para el desarrollo de API REST y back-ends específicos.

Algunas de las características que nos ofrece esta herramienta son las siguientes:

- **Configuración automática**: SpringBoot simplifica la configuración de la aplicación al automatizar gran parte del proceso mediante anotaciones y dependencias preconfiguradas. Esto disminuye la necesidad de realizar configuraciones manuales y permite un rápido desarrollo.

```
@Document(collection = "usuarios")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Usuario implements UserDetails {
    @Id
    private ObjectId id;
    private String nombre;
    private String apellidos;
    private String correo;
    private List<String> cargos;
    private String color;

    private String password;

    private Rol rol;
```

Ilustración 23. Anotaciones de SpringBoot

Como podemos observar, en la Ilustración 23., la clase Usuario se reduce a estas líneas, pues gracias a las anotaciones tenemos toda la funcionalidad y características de una clase sin necesidad de tener que implementarlas manualmente.

@Document: Esta anotación nos permiten indicar mediante el parámetro collection en que colección de la base de datos se va a guardar un objeto de esta clase cuando sea creado.

@Data: Esta nos permite acceder a los *setters* y *getters* de todos los atributos definidos, es decir, nos crea los *setters* y *getters* por defecto y los podemos usar en cualquier momento sin necesidad de haberlos implementado manualmente, trabajo que suele ser costoso cuando tenemos gran cantidad de clases.

@NoArgsConstructor: Esta anotación nos crea por defecto el constructor vacío de la clase sin necesidad de que tengamos que escribirlo nosotros.

@AllArgsConstructor: Esta anotación nos crea por defecto el constructor de la clase con todos los atributos definidos, es como el anterior, pero en vez de crearlo vacío lo crea con todos los atributos.

@Builder: Esta otra es muy interesante ya que nos permite construir un objeto de una clase siguiendo el esquema del patrón de diseño Builder. Esta genera un constructor privado y una clase interna estática que tiene métodos para crear instancias de la clase original gracias al método *build()*. Este patrón se explicará más a fondo más adelante.

- **SpringBoot Starters:** Los starters de SpringBoot son dependencias preconfiguradas que nos simplifican el uso de características específicas de una aplicación de SpringBoot.

  Existen starters para la integración con bases de datos, la creación de API REST, la seguridad y muchas más cosas. Gracias a estos starters no tenemos que configurar nada de manera manual si queremos introducir una API REST en nuestra aplicación o si quisiéramos cambiar de base de datos sería tan sencillo como cambiar la dependencia que estamos usando para nuestra SQL y agregar una para MongoDB, obviamente con sus respectivos pequeños cambios en algunas clases.

  En el caso de WeKraal se han utilizado los siguientes starters, mostrados en la Ilustración 24., para la integración de MongoDB, API REST, seguridad, generación de documentación de la API y testing:

```
implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.boot:spring-boot-starter-security'
testImplementation 'org.springframework.boot:spring-boot-starter-test'
implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.0.3'
```

Ilustración 24. Starters de SpringBoot

MongoDB

Para la persistencia de los datos se ha escogido MongoDB como base de datos ya que encajaba mejor en el proyecto, donde se ha intentado llevar un proceso de desarrollo ágil de una aplicación. Además, MongoDB tiene muy buena integración con SpringBoot debido a las facilidades que nos otorga como los starters y las anotaciones, lo cual hace posible una población de la base de datos rápida, consistente y fácil para los desarrolladores.

5.2. Comunicación entre capas

A continuación, se va a explicar la comunicación entre las capas presentadas.

Comunicación front-end

Como se ha explicado en los apartados anteriores, el front-end es quien actúa de cliente en este sistema, y por tanto es el que se encarga de pedir al servidor los datos o recursos que necesita para mostrarle al usuario después de que este haya realizado una interacción con la interfaz.

Este cliente se comunica con el servidor haciendo uso de peticiones HTTP. El protocolo HTTP se utiliza para intercambiar información entre el cliente y el servidor y es gracias a este que se establece una comunicación estructurada que permite la interacción entre el usuario y los recursos disponibles.

Este protocolo permite al cliente realizar varios tipos de peticiones HTTP al servidor, que, según qué tipo de petición se haga, el servidor actuará de una forma u otra.

Para facilitar esta comunicación, se ha utilizado la librería Axios[5] [10] en el cliente. Axios es una biblioteca de JavaScript que permite realizar solicitudes HTTP de forma sencilla y eficiente. Proporciona métodos para enviar peticiones GET, POST, PUT, DELETE, entre otros, y maneja las respuestas del servidor de manera adecuada. Un ejemplo de cómo se usa esta biblioteca para hacer peticiones al servidor es el siguiente:

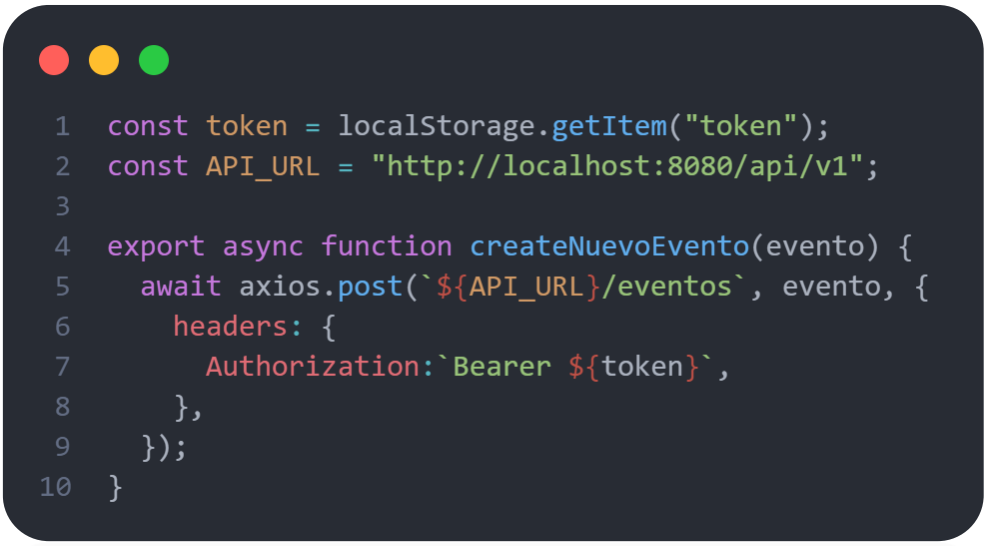
[5] Página web oficial de Axios: https://axios-http.com/es/docs/intro

```
const API_URL = "http://localhost:8080/api/v1";

export async function getTareaByTitulo(titulo) {
  return await axios.get(`${API_URL}/tarea/titulo=${titulo}`);
}

export async function createTarea(tarea){
    return await axios.post(`${API_URL}/tareas`, tarea);
}
```

Ilustración 25. Peticiones HTTP Axios

Como podemos observar en la Ilustración 25. con Axios ya tenemos métodos predefinidos para GET, POST PUT y DELETE que nos ayudan a realizar peticiones HTTP de manera sencilla y rápida. Encapsulando estas llamadas bajo una función podemos acceder a los datos con una simple ejecución de una función o crear elementos en la base de datos con un simple método al que únicamente es necesario pasarle como parámetro el elemento a crear.

Además, Axios nos permite también configurar todas las partes de una petición HTTP y enviarla junto a esta como puede ser los headers. En el caso en que necesitemos autorización para realizar una llamada a la API podemos añadir en la petición las cabeceras necesarias para ello. A continuación, se muestra un ejemplo en la Ilustración 26.



```
const token = localStorage.getItem("token");
const API_URL = "http://localhost:8080/api/v1";

export async function createNuevoEvento(evento) {
  await axios.post(`${API_URL}/eventos`, evento, {
    headers: {
      Authorization:`Bearer ${token}`,
    },
  });
}
```

Ilustración 26. Peticiones HTTP Axios con headers

Como se ha explicado anteriormente, las peticiones se realizan sobre un servidor, cuyo funcionamiento se maneja con SpringBoot, que es el que responde estas peticiones. En él están definidos los *mappings* para que este ejecute un código especifico según la petición que se realice y este retornará siempre una Response, por lo que cuando hacemos peticiones con Axios podemos acceder a la respuesta que nos ha dado el

servidor, es decir, la propia petición cuando se hace efectiva retorna un valor de parte del servidor.

Comunicación back-end

Por otro lado, tenemos el back-end, este es quien actúa de servidor. En este proyecto, el back-end se ha construido con SpringBoot y se ha utilizado una base de datos MongoDB. La comunicación de parte del back-end es más compleja que la del front-end ya que este solo necesitaba realizar peticiones HTTP, pero el back-end necesita de más componentes para ofrecer una comunicación rápida y efectiva.

Si seguimos el ciclo de vida de una petición HTTP, puesta de ejemplo en el apartado del front-end, nos habíamos quedado en que llega al servidor y este le contesta. Para entender cómo funciona el servidor se van a explicar los elementos que intervienen y de qué manera se comunican internamente entre sí para generar la respuesta que quiere el cliente. El back-end está compuesto por los siguientes elementos:

- **Modelos**
  Los modelos desempeñan un papel fundamental en el back-end al representar los objetos que se almacenarán en la base de datos y con los cuales se interactuará durante el proceso de comunicación. Estos modelos definen y representan los datos que el usuario solicita al servidor. De esta manera, los modelos actúan como una capa de abstracción que permite la manipulación y gestión eficiente de la información almacenada en la base de datos, facilitando así la interacción con los datos requeridos por el usuario.

- **Controladores**
  Los controladores, como bien indica la palabra, son los que se encargan de manejar las peticiones que realiza el cliente y coordinar la lógica de negocio correspondiente. Cuando el cliente realiza una petición al servidor, el controlador correspondiente recibe y procesa dicha petición. El controlador utiliza los modelos y servicios apropiados para llevar a cabo las operaciones necesarias, como la recuperación o manipulación de datos

- **Servicios**
  Los servicios son los que se encargan de implementar la lógica de negocio del back-end. Estos hacen de intermediarios entre los controladores y los repositorios. Estas funciones podría realizarlas el controlador cuando recibe una petición, pero no sería apropiado y se ha decidido aislar dicha funcionalidad para hacer el código más mantenible y modularizado. Además, es una forma de llevar un control de los errores y de esta manera aislarlos para identificarlos mejor, así sabemos cuándo un error está siendo producido por un servicio o por un controlador
  El objetivo principal de estos componentes del back-end es proporcionar una capa de abstracción para realizar operaciones más complejas y encapsular la lógica de negocio.

- **Repositorios**

Estos elementos son los que se utilizan para interactuar con los datos, estos proporcionan una interfaz para realizar las operaciones básicas ya comentadas anteriormente, CREATE, READ, UPDATE, DELETE, comúnmente llamadas con el nombre de operaciones CRUD que se asocian a cada una de las peticiones HTTP. Los repositorios, básicamente tienen como objetivo facilitar la interacción con la base de datos, eliminando le necesidad de escribir las consultas manualmente.

- **Configuraciones**
  Un elemento importante que nos facilita SpringBoot para el servidor son las clases de configuración. Estas clases definen configuraciones y filtros para la seguridad, la documentación, para la configuración de la aplicación en general y muchas más opciones. En WeKraal se han añadido las necesarias para proteger la aplicación, como la clase SecurityConfig, ApplicationConfig, o JwtAuthenticationFilter.

Una vez conocidos los elementos que intervienen en la generación de la respuesta por parte del back-end, se va a mostrar y explicar cómo se realiza la comunicación entre estos elementos para generar la respuesta.

Cuando llega una petición del front-end al back-end, esta tiene un controlador asociado que se encarga de manejar la solicitud. En SpringBoot las clases que actúan de controlador se tienen que indicar con anotaciones, además se debe otorgar a cada controlador un llamado RequestMapping, es gracias a este mapping que el controlador sabe que peticiones tiene que gestionar. Un ejemplo sería el siguiente:

```
@RestController
@RequestMapping("/api/v1/actas")
@CrossOrigin("http://localhost:3000")
public class ActaController {
```

Ilustración 27. Configuración clase Controlador

Como podemos observar en la Ilustración 27. se ha de indicar que es un controlador, mediante la anotación **@RestController,** y además se tiene que indicar cuales son las peticiones que va a tener que manejar, en este caso son todas aquellas que lleguen a través de "/API/v1/actas".

La anotación @CrossOrigin es para indicarle al controlador de donde tiene que permitir las peticiones, por lo que en este caso solo aceptara peticiones que provengan de http://localhost:3000.

Una vez llega la petición al controlador, este le asigna un método según el tipo de petición que sea y la ruta a la que está realizando la petición.

```
@GetMapping("/")
public ResponseEntity<List<Acta>> getAllActas() {
    return new ResponseEntity<>(actaService.getAllActas(), HttpStatus.OK);
}
```

Ilustración 28. GetMapping Controlador

Si la petición se hace sobre la raíz, es decir, sobre "API/v1/actas/" se le asigna este método de la imagen que devuelve una ResponseEntity cuyo cuerpo es la ejecución del método getAllActas() de la clase ActaService que se explica más adelante.

En el caso en que la petición se hiciese sobre "API/v1/actas/nombre={"Consejo 2 Mayo"}" se le asignaría este otro método que se encarga de en vez de retornar todas las actas, aquella que tenga asignado el nombre "Consejo 2 Mayo":

```
@GetMapping("/nombre={nombre}")
public ResponseEntity<Optional<Acta>> getActaByNombre(@PathVariable String nombre) {
    return new ResponseEntity<>(actaService.getActaByNombre(nombre), HttpStatus.OK);
}
```

Ilustración 29. GetMapping Controlador getByNombre()

Una vez visto el funcionamiento de los controladores se va a explicar cómo se genera el cuerpo de la respuesta que se pasa como valor de retorno en el controlador. Como se ha mencionado anteriormente, son los servicios del back-end los encargados de la lógica de negocio y por tanto son los que van a generar dicho cuerpo de la respuesta.

En el ejemplo anterior se ha visto que el cuerpo de la respuesta viene generado por una clase ActaService que tiene métodos como getAllActas() o getActaByNombre(). La clase ActaService es en realidad una interfaz en la que definimos estos métodos que se encargan de gestionar la lógica del servidor.

```
public interface ActaService {
    1 usage  1 implementation  sancore17
    List<Acta> getAllActas();
    1 usage  1 implementation  sancore17
    Optional<Acta> getActa(ObjectId id);
    1 usage  1 implementation  sancore17
    Acta createActa(Acta acta);
    1 usage  1 implementation  sancore17
    Acta updateActa(String nombre, Acta acta);
    1 usage  1 implementation  sancore17
    Optional<Acta> getActaByNombre(String nombre);
}
```

Ilustración 30. Interfaz ActaService

Como podemos observar en la Ilustración 30. se definen los métodos en esta clase interfaz, estos métodos son los que se van a usar luego en los controladores. Una interfaz no tiene sentido sin una implementación por tanto se ha creado una clase ActaServiceImpl para la implementación de esta interfaz, así como para todas las interfaces existentes que definen métodos para gestionar la lógica del servidor. A continuación, se muestra en la Ilustración 31. el ejemplo.

```
@Service
public class ActaServiceImpl implements ActaService {
```

Ilustración 31. Clase Implementación de Interfaz Servicio

Hay que tener en cuenta las anotaciones de SpringBoot, pues cuando definimos un Servicio tenemos que hacer uso de la anotación **@Service**, esta se usa para indicar que es una clase que se usa para la lógica de negocio y se conecta a repositorios.

```
@Override
public List<Acta> getAllActas() {
    return actaRepository.findAll();
}
1 usage  sancore17
@Override
public Optional<Acta> getActa(ObjectId id) {
    return actaRepository.findById(id);
}
1 usage  sancore17
@Override
public Acta createActa(Acta acta) {
    return actaRepository.save(acta);
}
1 usage  new *
@Override
public Optional<Acta> getActaByNombre(String nombre) {
    return actaRepository.findByNombre(nombre);
}
```

Ilustración 32. Implementación Servicio

En la Ilustración 32. podemos observar cómo sería la implementación de la interfaz ActaService, en la cual tenemos los métodos que son llamados por los controladores a través de la interfaz. En esta imagen solo aparecen los métodos relacionados con las peticiones GET y POST, pero también se incluyen los métodos para modificar y eliminar

las actas, los cuales contienen más lógica pues primero se ha de comprobar la existencia de estas en la base de datos y posteriormente actualizarlas o eliminarlas.

Podemos observar que el cuerpo de los métodos se reduce a una simple llamada a un método de las clases repositorio.

Por último, entran en juego los repositorios, los cuales a través de los servicios proporcionan los datos que está solicitando el cliente. Los métodos principales que tienen estos repositorios son los siguientes:

- **save()**: este método necesita un parámetro del tipo correcto y se encarga de almacenar el objeto en la base de datos. Típicamente este método se ejecuta cuando las peticiones del cliente son de tipo POST.
- **findAll()**: este otro sirve para retornar todos los elementos de la base de datos que sean del tipo de objetos que maneja el repositorio desde el cual se está ejecutando el método. Este método se suele ejecutar cuando el cliente solicita datos, pero no se pasa nada por parámetro, indicando que se quiere obtener todos los datos.
- **findById()**: este método necesita el identificador del objeto que se quiere recuperar. Este identificador es, típicamente, el índice que ocupa el objeto en la base de datos.
- **delete():** este método se encarga de borrar el objeto pasado por parámetro de la base de datos. Este se relaciona directamente con el tipo de petición DELETE que realiza el cliente.

Como se ha podido observar, los elementos del back-end son todos necesarios para el correcto funcionamiento de este. Una buena práctica es separar la funcionalidad del back-end en estos componentes mencionados para lograr abstracción y mayor modularidad. Con esto obtenemos una mejor trazabilidad de los errores y aislamos las funcionalidades en diferentes componentes, cumpliendo así el principio de responsabilidad única.

Para entender mejor el funcionamiento interno del back-end, a continuación, se muestra un diagrama de cómo se conectan y comunican los componentes entre sí.

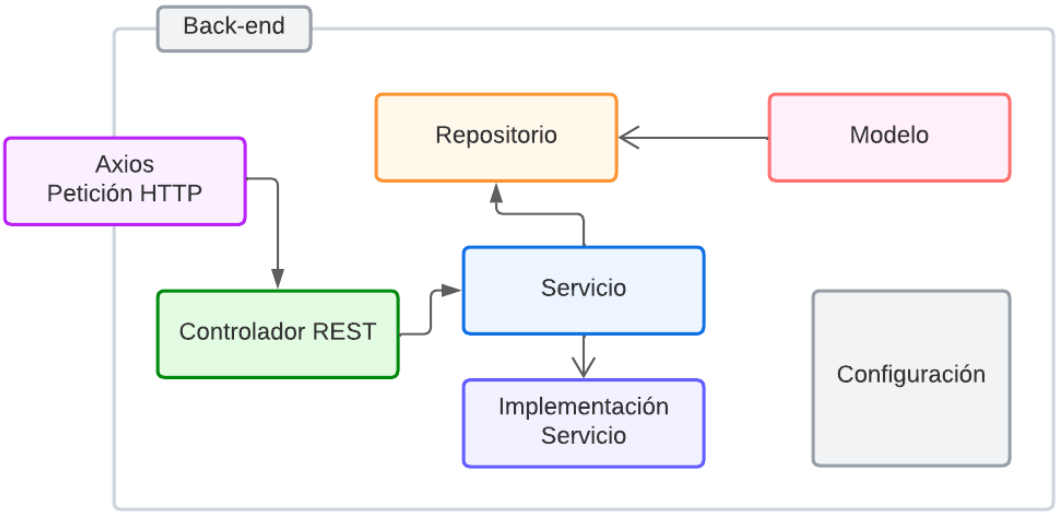


Ilustración 33. Diagrama de componentes del back-end

En la Ilustración 33. observamos el esquema del back-end, este es un esquema general que muestra de manera visual cómo se conectan los componentes del back-end. Resumiendo, las peticiones llegan a los controladores y este hace uso de una interfaz de servicio la cual tiene una implementación. En esta clase se ejecutan métodos que cambian o recuperan los datos. Todo este conjunto de funciones se ejecuta bajo un contexto que tiene una configuración asociada, definida por las clases de configuración del back-end. Una vez accedido a los datos, ya sea para actualizar, crear, borrar u obtener datos concretos, se retorna una respuesta al cliente con los datos correspondientes.

5.3. Seguridad

Un aspecto importante que se ha de tener en cuenta a la hora de desarrollar un sistema es la seguridad. Esta característica es crucial para garantizar la integridad, confidencialidad y disponibilidad de los datos y recursos del sistema. La seguridad abarca diversas áreas y requiere de la implementación de medidas adecuadas.

Lo primero que se ha pensar cuando se va a desarrollar un sistema con seguridad es realizar una evaluación de que riesgos y amenazas pueden afectar al sistema.

Una vez se han identificado los riesgos y vulnerabilidades se debe pensar en implementar los mecanismos de seguridad adecuados como firewalls, cifrado de datos o autenticación y autorización de usuarios. A esta implementación hay que sumarle que se ha de desarrollar siguiendo buenas prácticas de seguridad.

Gracias a la seguridad podemos mitigar el riesgo y asegurar una protección de los datos, es por eso por lo que en WeKraal se ha llevado a cabo la implementación de JWT (JSON Web Token) [11], un estándar para transmitir información segura a través de internet en

formato JSON. Con esta herramienta podemos, por ejemplo, identificar a los usuarios cuando inician sesión en la aplicación.

Con JWT[6] podemos autenticar y autorizar a los usuarios de la aplicación, de esta manera nos aseguramos de que únicamente aquellos con las credenciales correctas puedan interactuar con el sistema. El modo de funcionamiento de estas dos acciones en el framework JWT es el siguiente:

- **Autenticación:** con jwt, cuando un usuario se autentica correctamente, el back-end genera un token, un jwt, el cual contiene la información necesaria para identificar al usuario y permitirle realizar peticiones al servidor. Este token generado debe ser incluido en las peticiones posteriores, en el apartado de headers, como un Bearer token.
- **Autorización:** la autorización en el contexto de jwt, es el proceso por el cual se verifica si un usuario puede realizar acciones en el sistema en el caso que tenga los permisos necesarios para ello. Dado que pueden existir usuarios con varios roles como el de administrador o el de usuario común, jwt se encarga de determinar si un usuario tiene un rol u otro gracias a los llamados "claims" de jwt. Los claims contienen información adicional como el rol del usuario o una lista de permisos asociados al usuario donde se indica que cosas puede y que no puede hacer el usuario como realizar solicitudes POST, por ejemplo. Esta información es útil durante el proceso de autorización ya que esta determina si un usuario puede realizar determinadas acciones sobre el sistema.

Toda esta funcionalidad ofrecida por jwt se complementa con una clase de configuración propia de SpringBoot, esta tiene que heredar de WebSecurityConfiguration la cual nos proporciona el starter de spring-boot-starter-security. En esta clase se define un método de tipo SecurityFilterChain que nos permite definir las siguientes características:

- Definir reglas de acceso y restricciones para diferentes URLs.
- Configurar la autenticación y la autorización.
- Configurar la protección CSRF, que sirve para prevenir falsificación de peticiones.
- Habilitar o deshabilitar características de seguridad como los headers de las solicitudes o políticas de seguridad.

Un ejemplo de este método sería el que hay implementado en la aplicación de WeKraal:

[6] Página web oficial de JWT https://jwt.io/

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
    http
            .csrf() CsrfConfigurer<HttpSecurity>
            .disable() HttpSecurity
            .authorizeHttpRequests() AuthorizationManagerRequestMat...
            .requestMatchers( ...patterns: "/api/v1/auth/**") AuthorizedUrl
            .permitAll() AuthorizationManagerRequestMat...
            .anyRequest() AuthorizedUrl
            .authenticated() AuthorizationManagerRequestMat...
            .and() HttpSecurity
            .sessionManagement() SessionManagementConfigurer<HttpSecurity>
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and() HttpSecurity
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

Ilustración 34. Método SecurityFilterChain de la clase SecurityConfig

Como podemos ver en la Ilustración 34., ayudándonos de la funcionalidad del patrón Builder para construir objetos complejos, definimos las características de seguridad que queremos aportar a nuestra aplicación como el CSRF, indicar cuales son las rutas para autorizar las peticiones http, indicar que cualquier petición que se haga tiene que estar autenticada, así como añadir filtros de seguridad, manejo de sesiones y proveedores de autenticación.

En cuanto al aspecto de cifrado de los datos, durante la generación del token, este se genera con una clave nombrada como SECRET_KEY que es la que firma el token y le da validez cuando es verificado. Cabe destacar que la generación del token es el final del proceso de autenticación pues previamente se han de cumplir varias condiciones y pasar por unos cuantos procesos antes de generar este token. Para ello cuando el usuario se autentica, la contraseña introducida por este se ha de cifrar y comparar con la versión ya cifrada almacenada en la base de datos. Es aquí donde entra el algoritmo de cifrado *bcrypt*.

Bcrypt [12] es un algoritmo de hashing para almacenar contraseñas de forma segura, este añade al cifrado un valor aleatorio llamado “salt”, el cual se concatena con la contraseña antes de aplicarse el cifrado, lo que lo hace más difícil de descifrar. Además, este valor aleatorio que se añade a la cadena hace que contraseñas iguales no generen un mismo hash. Esto es un añadido de seguridad muy determinante ya que si se lograse acceder a la base datos sería muy difícil obtener las contraseñas originales.

Una vez *bcrypt* ha verificado la contraseña se procede a la generación del token jwt. Jwt en realidad es la llave que permite al cliente autenticarse y lo autoriza a realizar peticiones sobre el sistema, y es fruto de un proceso de verificación de credenciales implementado con algoritmos de seguridad como el comentado.

Para un mejor entendimiento del funcionamiento de la seguridad en WeKraal, a continuación, se muestran ejemplos de la implementación de jwt y su posterior uso en el front-end para permitir a los usuarios acceder y navegar por la aplicación.

Para generar un token necesitamos una respuesta valida del servicio de autenticación de la aplicación, este servicio contiene los métodos registro() y authenticate() que se ejecutan cuando un usuario hace registro o login en la aplicación. Estos métodos generan un token el cual se asignará a este usuario, en el caso del registro crea un token para el nuevo usuario y en el caso del login primero verifica que ese usuario exista en la aplicación y posteriormente le asigna un token.

```
public String generateToken(
        Map<String, Object> extraClaims,
        UserDetails userDetails
){
    return Jwts
            .builder()
            .setClaims(extraClaims)
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
            .signWith(getSigningKey(), SignatureAlgorithm.HS256)
            .compact();
}
```

Ilustración 35. Método generateToken

En la Ilustración 35. observamos que para la generación del token necesitamos crear las partes básicas de este, que son los extraClaims, el subject que en nuestro caso va a ser el username de UserDetails que en realidad es el correo, y necesitamos también la fecha de creación y la de expiración del token. Además, tenemos que indicarle con que clave se va a firmar el token, obtenida en la llamada al método getSigningKey(), y que algoritmo de firma se va a utilizar para cifrarlo, que en este caso es HS256.

Una vez generado correctamente el token, se entrega al cliente y este deberá incluirlo en la cabecera de *Authorization* de las solicitudes HTTP como se ha mostrado en ejemplos anteriores.

Cada vez que se haga una petición al servidor, este pasará la petición por un filtro que verificara si contiene la cabecera y si el token es válido, entonces dejará al cliente realizar la solicitud. Este filtro básicamente se encarga de comprobar que la solicitud se está realizando con un token y además comprueba que el usuario asociado al token sea válido. Lo siguiente que ocurre es que se comprueba la validez del token ya que este puede estar expirado o revocado y una vez validado el token, si es correcto, se crea un objeto el cual representa al usuario autenticado. Por último, una vez verificado que la llamada se está realizando correctamente, el filtro deja pasar a la solicitud y esta continua su procesamiento.

Para entender de manera visual el funcionamiento de jwt se muestra a continuación un esquema, en la Ilustración 36., que representa como se conectan todos los agentes involucrados en el sistema de seguridad de WeKraal:

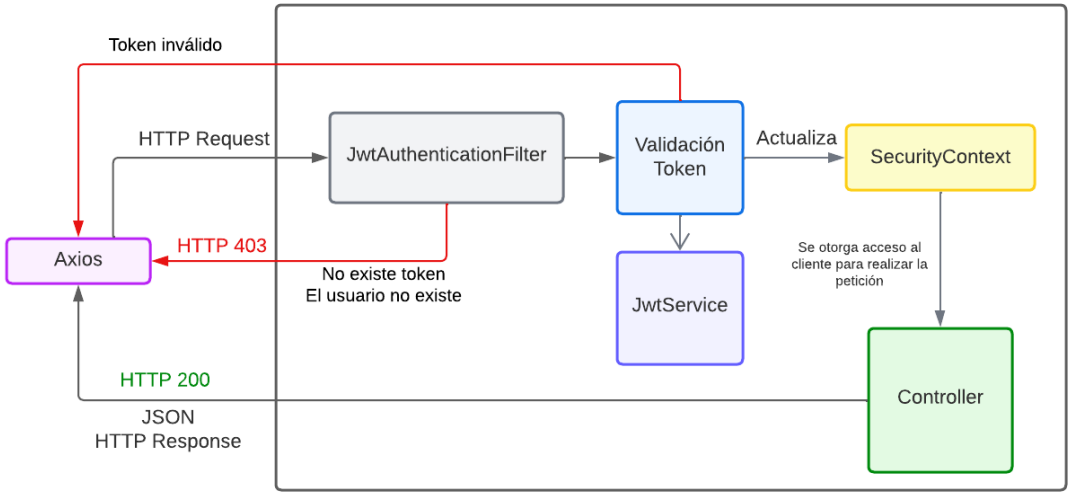


Ilustración 36. Esquema funcionamiento jwt

5.4. Documentación

Otro aspecto importante de la implementación ha sido la inclusión de herramientas que nos permitan documentar el proyecto. La documentación de un proyecto es una parte muy importante ya que de ella dependerá en gran medida que el sistema sea mantenible. Si el proyecto pasase a estar en manos de otro equipo de desarrollo, este no tendría el conocimiento necesario para saber que función realiza cada componente del sistema, por lo que es importante dedicar una fase del proyecto a documentarlo.

En WeKraal se ha utilizado la herramienta Swagger[7] [13] para generar la documentación de la API REST creada en el back-end. Swagger nos proporciona una documentación interactiva que define la estructura de nuestra API, con todos los endpoints, parámetros necesarios, tipos de datos, tipo de solicitudes y muchas más otras cosas.

En general Swagger nos ofrece las siguientes características.

- Documentación: en ella se describe todo acerca de la API, los endpoints, parámetros, tipo de respuestas…
- Interfaz de usuario: toda esta documentación nos la brinda a través de una interfaz gráfica llamada SwaggerUI donde se pueden probar las llamadas desde el propio navegador sin necesidad de usar herramientas como Postman[8] para probar la API.
- Validación y pruebas: gracias a la descripción de la API, Swagger puede realizar pruebas automáticas que nos ayudan a verificar el correcto funcionamiento de nuestra API y ayudar a corregir los errores en fases tempranas del desarrollo de esta.

[7] Página web oficial de Swagger: https://swagger.io/
[8] Página web oficial de Postman: https://www.postman.com/

Para que Swagger nos genere automáticamente la documentación de nuestra API tenemos que crear una clase de configuración de SpringBoot, esta debe tener las anotaciones **@Configuration** y **@EnableSwagger2.** Esta clase contiene un método api() de tipo Docket, objeto propio de Swagger, en el cual definimos la ruta donde se encuentran los controladores para que Swagger genere la documentación de la API.

Una vez creada la clase de configuración podemos acceder a la documentación a través de la url *localhost:8080/swagger-ui.hmtl.*

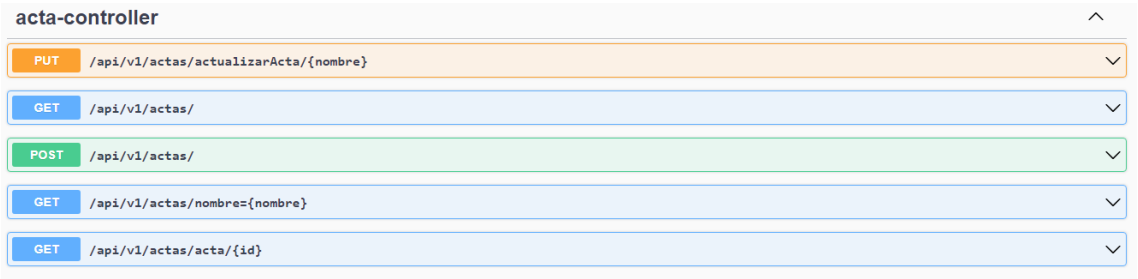


Ilustración 37. Interfaz SwaggerUI

Como podemos observar en la Ilustración 37. Swagger nos muestra una liste de todos los métodos asociados a un controlador, aquí podemos ver el acta-controller el cual tiene 3 métodos GET, 1 PUT y 1 POST. También nos muestra la ruta del endpoint donde se van a realizar las llamadas.

Si desplegamos por ejemplo el método POST obtenemos lo siguiente:

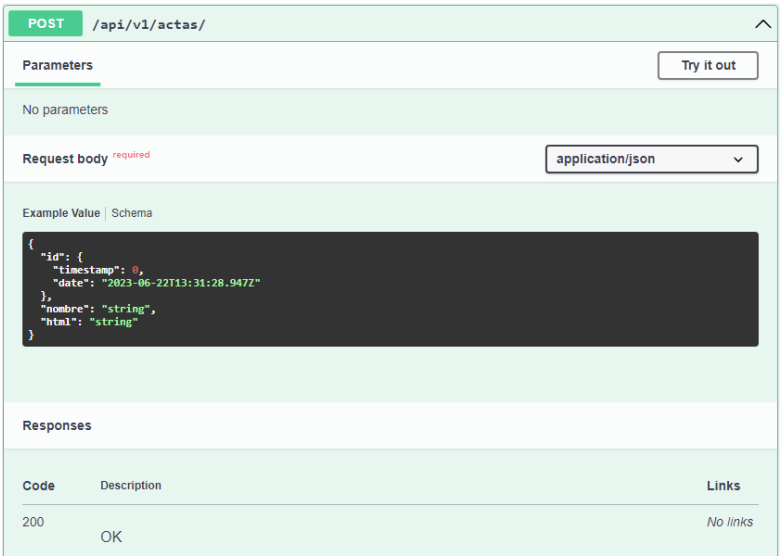


Ilustración 38. Desplegable SwaggerUI

En este desplegable de la Ilustración 38. se nos muestra la información necesaria para entender cómo funcionan las solicitudes POST a ese endpoint. Como podemos observar no son necesarios parámetros en la petición, pero sí que se indica como ha de ser el cuerpo de la Request, que en este caso debe tener un campo "id", otro campo "nombre" y un último campo "html". Más abajo se nos muestra cual va a ser el cuerpo de la Response que en este caso se va a devolver únicamente un 200, indicando que la solicitud se ha realizado con éxito. En el lateral superior derecho vemos un botón que dice *Try it*

out, el cual si pulsamos nos deja editar los valores de la solicitud para probar su funcionamiento.

Además, Swagger nos ofrece también en su interfaz los esquemas de los objetos que están presentes en la API.

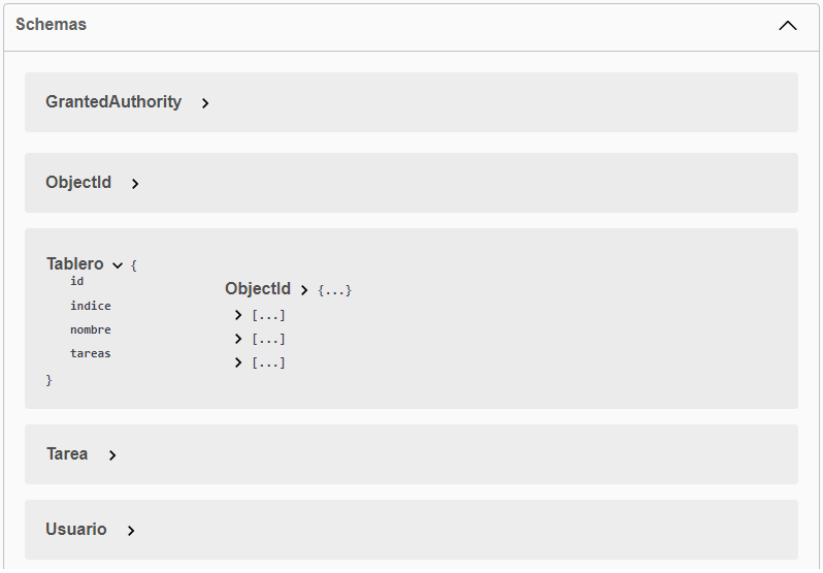


Ilustración 39. Esquemas de Objetos SwaggerUI

Además de conocer el funcionamiento de la API podemos también saber que datos manejan las solicitudes para un mejor entendimiento.

En resumen, Swagger nos proporciona una generación automática de nuestra API, así como también de los esquemas de los datos que esta maneja. Es una práctica que siempre se debería llevar a cabo, pues la documentación de un proyecto es un elemento crucial para su mantenibilidad. Swagger no solo nos permite generar la documentación, sino que también podemos probar las funcionalidades de nuestra API gracias a su interfaz de usuario.

Por último, cabe destacar que Swagger nos genera un JSON el cual es usado para construir la documentación y la interfaz gráfica pero que sirve también para importarlo en WSO2 API Manager y poder tener la API publicada con su respectiva configuración de límites de tasa, políticas de seguridad, etc. Este es un aspecto que valorar ya que permites que más desarrolladores puedan usar tu API, consiguiendo una promoción de tu servicio, generación de ingresos e innovación y escalabilidad pues gracias a tu API otros desarrolladores pueden crear nuevas aplicaciones innovadoras.

5.5. Patrones de diseño

Los patrones de diseño pueden ser una pieza fundamental durante la fase de implementación de un proyecto, ya que nos proporcionan soluciones efectivas a problemas que nos encontramos durante la fase de diseño e implementación. Los

patrones de diseño garantizan una mejora de la calidad del software, escalabilidad y flexibilidad y sobre todo mantenibilidad.

Para este proyecto se han aplicado dos patrones de diseño con ayuda de las facilidades que proporciona la herramienta SpringBoot. Como ya se ha comentado, SpringBoot ofrece un sistema de anotaciones que nos permite una configuración automática y ágil de nuestro proyecto y es gracias a estas que se ha podido aplicar los patrones Repositorio y Builder.

5.5.1. Patrón Repositorio

El patrón repositorio es un patrón de diseño que se encarga de gestionar el acceso a la base de datos. Comúnmente se utiliza para tratar con la capa de persistencia, por lo que su objetivo es proporcionar una capa de abstracción entre la lógica de negocio y el acceso a datos.

Este patrón proporciona una interfaz común para interactuar con los datos, cumpliendo el principio de responsabilidad única, la modularidad y ocultando detalles de implementación a la capa de negocio para que esta no tenga que preocuparse por cómo se interactúa con los datos.

En el caso de WeKraal, se ha implementado este patrón de diseño haciendo uso de las anotaciones y clases predefinidas de SpringBoot. Un ejemplo de cómo se ha aplicado el patrón repositorio en WeKraal es el siguiente:

```
public interface ActaRepository extends MongoRepository<Acta, ObjectId> {
    sancore17
    Optional<Acta> findByNombre(String nombre);
}
```

Ilustración 40. Patrón Repositorio

Como podemos observar en la Ilustración 40. para la implementación del patrón se ha creado una interfaz que ofrece métodos para el acceso a datos. La clase de la que extiende que es MongoRepository nos ofrece métodos predefinidos para realizar las operaciones básicas sobre los datos. Además, cuando se hace la herencia de esta clase tenemos que indicarle cual va a ser el tipo de datos con el que va a tratar, en este caso con datos de tipo Acta y ObjectId. De esta forma conseguimos que esta interfaz repositorio siempre trate con objetos de ese tipo por lo que podremos tener un repositorio para cada tipo de datos de nuestra aplicación sin que comprometa a los demás.

Otra funcionalidad que nos ofrece esta clase MongoRepository es que podemos definir métodos para acceder a los datos además de los que ya existen por defecto. La única restricción para esta funcionalidad es que el método tiene que acabar por el nombre de uno de los atributos que tenga el objeto. Por ejemplo, debido a los métodos predefinidos que me ofrece la clase MongoRepository no puedo obtener las Actas por su nombre, pero gracias a esta funcionalidad podemos definir un nuevo método siguiendo la sintaxis

típica de los métodos del repositorio (save, findBy, deleteBy...) con el que obtener las Actas por su nombre, en este caso el método findByNombre(). Además, podemos complicar los métodos para dotarlos de mayor utilidad usando las palabras clave que nos ofrece MongoRepository para la creación de métodos adicionales, como And, Or, In, Like, con los que podemos crear métodos como los mostrados en la Ilustración 41.:

```
Optional<Usuario> findByRolOrSeccionIn(Rol rol, List<Seccion> secciones);
no usages   new *
Optional<Usuario> findeByNombreOrApellidos(String nombre, String apellidos);
no usages   new *
Optional<Usuario> deleteByCorreo(String correo);
```

Ilustración 41. Métodos personalizados

En resumen, no se ha aplicado directamente el patrón repositorio, sino que se ha hecho uso de las herramientas disponibles para poder usar el patrón implementado por SpringBoot y sus paquetes. Gracias a este patrón repositorio que se ha implementado podemos facilitar y abstraer el acceso a los datos, dotándolo también de una gran flexibilidad gracias a los métodos personalizados que permite definir la clase MongoRepository. Y sobre todo debido a su implementación ganamos modularidad, reutilización y testabilidad del código.

5.5.2. Patrón Builder

Otro de los patrones que se han aplicado durante el desarrollo de la solución es el patrón Builder. Este patrón es un patrón de diseño creacional que nos permite construir objetos complejos parte por parte de tal forma que haga simple la construcción de estos objetos. Ya que estos objetos tienen muchas configuraciones posibles, se utiliza este patrón para evitar construir objetos con muchos parámetros en el constructor y enfocar la construcción de este en un proceso por pasos, consiguiendo el objeto con las partes deseadas.

En el caso de WeKraal el patrón Builder se ha aplicado con la ayuda de la anotación ofrecida por SpringBoot, **@Builder**. Esta anotación como ya se ha comentado anterior mente nos permite aplicar el patrón Builder en la clase donde se haya definido esta anotación. Por ejemplo, si se añade la anotación a la clase Usuario, podremos construir objetos de tipo Usuario siguiendo la estructura del patrón Builder como se muestra a continuación:

```
var user = Usuario.builder()
        .nombre(request.getNombre())
        .apellidos(request.getApellidos())
        .correo(request.getEmail())
        .password(passwordEncoder.encode(request.getPassword()))
        .rol(Rol.USER)
        .cargos(new ArrayList<>())
        .color(Usuario.generateRandomColor())
        .build();
```

Ilustración 42. Aplicación del patrón Builder

Como se puede observar en la Ilustración 42., añadiendo la anotación **@Builder** a la clase Usuario ya podemos crear usuarios siguiendo la estructura del patrón Builder y sin la necesidad de escribir constructores con tantos parámetros. Además, con esta estructura conseguimos un enfoque más visual de los elementos que intervienen en la construcción del objeto que en caso de errores nos permite identificar con más rapidez donde se encuentran estos.

Podemos concluir con que la aplicación de estos patrones nos ofrece soluciones a problemas comunes durante el desarrollo de software. Gracias a ellos podemos mejorar la calidad del código, su reutilización y el rendimiento de nuestro sistema.

6. Pruebas

La fase de pruebas en un proyecto software es una fase obligatoria, ya que gracias a las pruebas podemos evaluar la calidad, rendimiento y funcionalidad de nuestro sistema software. El objetivo principal de las pruebas es validar si la aplicación responde correctamente y cumple con las funcionalidades para las que ha sido desarrollada de manera eficiente, rápida y consistente. Esta fase no sirve únicamente para validar y verificar, sino que también nos ayuda a corregir errores y mejorar aspectos en las funcionalidades, nos ayudan también a evaluar cómo de robusto es el sistema desarrollado.

6.1. Pruebas Unitarias

Unas de las pruebas que se han realizado en el sistema de WeKraal son las pruebas unitarias. Con este tipo de pruebas conseguimos validar y verificar el correcto funcionamiento de cada componente de la aplicación. En este caso dado que tenemos una aplicación en SpringBoot que hace de API REST, tenemos que realizar pruebas unitarias para verificar que los componentes como los controladores, servicios y repositorios funcionan sin problema. Lo primero que se tiene que hacer para comenzar a hacer pruebas es configurar un entorno en el que poder testear la aplicación. Esto puede conllevar en ocasiones configuraciones adicionales como crear una base de datos en memoria para no interactuar directamente con la original o añadir las dependencias necesarias. En el caso de WeKraal, SpringBoot nos configura automáticamente el entorno. Para crear este entorno de testing, únicamente tenemos que hacer clic derecho sobre la clase la cual queremos probar y nos aparece una opción “Generate Test” que nos crea la clase de testing en el entorno de pruebas. De tal forma que, si queremos agregar más clases para probar, repetimos la acción y estas se crearan en el entorno de testing ya generado.

Dado que queremos verificar el funcionamiento de los componentes de nuestra aplicación sin acceder directamente a los datos tenemos que hacer Mocks. Un Mock es un objeto que simula el comportamiento de un objeto real durante el proceso de testing. De esta forma conseguimos obtener la funcionalidad de los componentes de manera simulada por lo que actuará de la misma forma que lo hace un componente real del sistema. Los Mocks nos ofrecen funcionalidades para definir como han de comportarse, de tal forma que podemos probar el correcto funcionamiento de nuestra aplicación sin necesidad de acceder a recursos externos o dependencias. Para estas pruebas se ha usado la librería de Mockito.

A continuación, se muestra cómo se han probado los principales componentes de la aplicación: Repositorios y Controladores

6.1.1. Repositorios

Para el caso de los repositorios tenemos que validar que los métodos definidos en los servicios están realizando bien las llamadas a los métodos de los repositorios y que por ende se está cumpliendo el correcto funcionamiento de los repositorios. Para ello necesitamos hacer pruebas unitarias de los métodos definidos en los servicios, los cuales tienen la implementación en las clases ServiceImpl. Un ejemplo de cómo se han probado los repositorios es el siguiente:

```
@Mock
private UsuarioRepository usuarioRepository;
4 usages
@InjectMocks
private UsuarioServiceImpl usuarioService;
12 usages
private Usuario usuario;
@BeforeEach
void setUp(){
    MockitoAnnotations.openMocks( testClass: this);
    usuario = new Usuario();
    usuario.setApellidos("Cortés Reverón");
    usuario.setNombre("Santi");
    usuario.setCargos(new ArrayList<>());
    usuario.setColor("#ffffff");
    usuario.setCorreo("sancore17@gmail.com");
    usuario.setId(new ObjectId());
    usuario.setPassword("newPassword*");
    usuario.setRol(Rol.USER);
}
```

Ilustración 43. setUp de testing para los Repositorios

En la Ilustración 43 podemos observar que para hacer pruebas de la funcionalidad de los repositorios necesitamos unos pasos previos. Lo primero que tenemos que hacer es mockear el objeto Repositorio que vamos a usar para testar su funcionalidad, en este caso UsuarioRepository. Como ya se ha dicho necesitamos hacer un Mock del objeto repositorio e inyectarlo en la clase que va a hacer uso de ese Mock, en este caso UsuarioServiceImpl. Una vez hecho esto tenemos disponible un método que ofrece JUnit para inicializar objetos y demás componentes antes de ejecutar las pruebas, este método es el setUp(), en él se ha creado un objeto Usuario con el cual se van a realizar las pruebas.

Algunos de los métodos de tipo GET que se han probado son los siguientes:

```java
@Test
void getAllUsers() {
    when(usuarioRepository.findAll()).thenReturn(Arrays.asList(usuario));
    assertNotNull(usuarioService.getAllUsers());
}

@Test
void getUser() {
    ObjectId id = new ObjectId();
    when(usuarioRepository.findById(id)).thenReturn(Optional.of(usuario));
    Optional<Usuario> resultado = usuarioService.getUser(id);
    assertTrue(resultado.isPresent());
    assertEquals(usuario, resultado.get());
}
```

Ilustración 44. Métodos GET Test

Como podemos ver en la Ilustración 44., los métodos de prueba se encargan de simular el funcionamiento del método original con métodos como when() y thenReturn() para indicarle al repositorio como ha de comportarse frente a llamadas concretas.

Para el caso de la prueba para el método POST seria semejante, al igual que para los demás métodos como PUT y DELETE. Un ejemplo para hacer las pruebas de los métodos POST sería el siguiente.

```java
@Test
void createUser(){
    Usuario usuario = new Usuario();
    usuario.setNombre("UsuarioPOST");
    when(usuarioRepository.save(usuario)).thenReturn(usuario);
    Usuario result = usuarioService.createUser(usuario);
    assertEquals(usuario, result);
}
```

Ilustración 45. Test de Método POST

Como se puede observar en la Ilustración 45., el funcionamiento se basa en simular lo que deben de hacer los Mocks. Los métodos PUT tienen un poco más de complejidad, pero llevan la misma dinámica. Aquí tenemos que simular tanto la búsqueda del objeto en la base de datos como el guardado del mismo pero modificado ya que estos dos métodos son los que se usan para modificar un objeto, primero se busca, luego se modifica y por último se guarda. Un ejemplo, mostrado en la Ilustración 46, sería el siguiente:

```
@Test
void updateUser(){
    ObjectId id = new ObjectId();
    Usuario usuarioExistente = new Usuario();
    usuarioExistente.setId(id);
    usuarioExistente.setNombre("UsuarioExistente");

    Usuario usuarioActualizado = new Usuario();
    usuarioActualizado.setId(id);
    usuarioActualizado.setNombre("UsuarioActualizado");

    when(usuarioRepository.findById(id)).thenReturn(Optional.of(usuarioExistente));
    when(usuarioRepository.save(usuarioActualizado)).thenReturn(usuarioActualizado);

    Usuario result = usuarioService.updateUser(id, usuarioActualizado);
    assertEquals(usuarioActualizado, result);
}
```

Ilustración 46. Testing método PUT

En este caso observamos cierto grado de complejidad, aunque sigue las mismas directrices que las pruebas anteriores, simular el funcionamiento.

Una vez ejecutado los test de la clase de testing que se ha creado para UsuarioServiceImpl obtenemos que los test se han pasado correctamente por lo que hemos verificado y validado que la funcionalidad se cumple correctamente y que el acceso a datos y sus modificaciones se realiza de manera exitosa.

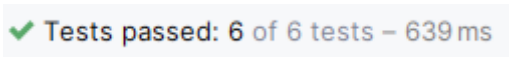


Ilustración 47. Tests pasados

Estas pruebas se han realizado sobre todas las clases ServiceImpl del proyecto para verificar que el acceso a los datos se hace de manera correcta.

6.1.2. Controladores

Una vez hemos realizado las pruebas sobre el acceso a datos tenemos también que verificar que los controladores están funcionando correctamente ya que estos son los que dan entrada a las peticiones HTTP y coordinan la lógica de negocio para estas llamadas.

Para hacer pruebas de estas clases, tenemos que simular las llamadas. Una vez simuladas las llamadas tenemos que comprobar que la respuesta generada por el controlador es la que tiene que ser. Para comprobar que la respuesta es válida tenemos que acceder a las propiedades de la respuesta y comprobar que los datos son los esperados. Estas comprobaciones las hacemos de la siguiente manera, mostradas en la Ilustración 48.:

```java
@Test
public void getUserByIdTest() throws Exception {
    Usuario usuario = new Usuario();
    usuario.setId(new ObjectId());
    usuario.setNombre("Santi");
    usuario.setCorreo("sancore17@gmail.com");
    when(usuarioService.getUser(new ObjectId())).thenReturn(Optional.of(usuario));
    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/api/v1/usuarios/{id}",  ...uriVariables: new ObjectId()))
            .andExpect(status().isOk())
            .andExpect((ResultMatcher) jsonPath( expression: "$.id", is(usuario.getId())))
            .andExpect((ResultMatcher) jsonPath( expression: "$.nombre", is(usuario.getNombre())))
            .andExpect((ResultMatcher) jsonPath( expression: "$.correo", is(usuario.getCorreo())));

}
```

Ilustración 48. Testing Controlador GET

En la imagen podemos observar cómo creamos un Usuario de prueba al que le ponemos los valores Nombre y Correo y le asignamos también un ObjectId. Después simulamos la llamada a la API sobre la ruta "/api/v1/usuarios/{id}" pasándole el id por parámetro para que encuentre a ese usuario. Por último, indicamos que valores esperamos con el método andExpect() y ejecutamos el test. De esta forma validamos el correcto funcionamiento del método GET getUserById() de nuestro UsuarioController. Dado que los métodos retornan siempre un ResponseEntity del tipo de objeto que trata, las pruebas para todos los métodos de cada controlador del sistema son semejantes.

6.2. Pruebas de rendimiento

Otras de las pruebas que se han realizado durante la fase de testing han sido las pruebas de rendimiento. Estas pruebas se basan en someter a la aplicación a un alto volumen de solicitudes sobre sus endpoints para ver si soporta la carga. En el caso de WeKraal se han hecho estas pruebas sobre sus endpoints usando la herramienta JMeter[9]. Con esta herramienta podemos crear grupos de hilos que realizan solicitudes HTTP sobre las rutas de la aplicación que le indiquemos. Por ejemplo, si queremos hacer pruebas de rendimiento sobre el endpoint: "localhost:8080/api/v1/usuarios", configuramos la herramienta para que el grupo de hilos apunte a ese endpoint y lo ejecutamos, pero antes debemos configurar un elemento más de JMeter y es que nuestra aplicación necesita que el cliente que realiza las solicitudes este autenticado por lo que tenemos que indicarlo de alguna manera. Es por eso por lo que tenemos que agregar al grupo de hilos (Thread Group) un componente llamado HTTP Header Manager. En este indicamos los headers necesarios para hacer solicitudes sobre la aplicación, en nuestro caso el Authorization Header, con el valor correspondiente. A continuación, se muestra el ejemplo en la Ilustración 49.:

[9] Pagina web oficial de JMeter: https://jmeter.apache.org/

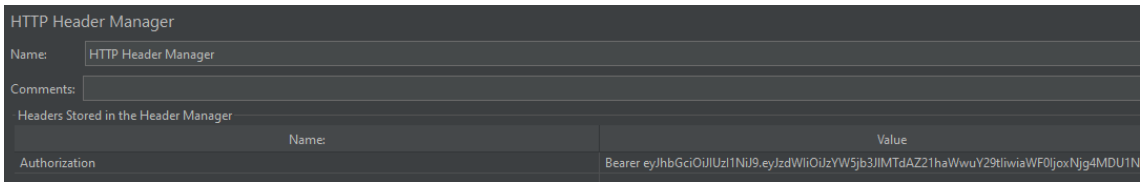


Ilustración 49. HTTP Header Manager JMeter

Después de haber configurado todo podemos ejecutar el grupo de hilos. Para que se muestre la información en formato de tabla tenemos que crear un elemento de JMeter para el grupo de hilos llamado "View results in table". Con este elemento ya configurado, si ejecutamos obtenemos la siguiente tabla con la información de la ejecución y de las solicitudes. En nuestro caso se ha indicado que se realicen un total de 2000 peticiones GET sobre el endpoint "localhost:8080/api/v1/usuarios" en un total de 10 segundos, y el resultado es el siguiente:

Sampl...	Start Time	Thread Name	Label	Sample Time...	Status	Connect T...	Bytes	Sent Bytes ↓	Latency
1	18:27:28.456	Thread Group 1-1	HTTP Request	247		3	2555	304	246
2	18:27:28.467	Thread Group 1-3	HTTP Request	236		1	2555	304	235
3	18:27:28.463	Thread Group 1-2	HTTP Request	242		1	2555	304	241
4	18:27:28.477	Thread Group 1-5	HTTP Request	249		1	2555	304	249
5	18:27:28.472	Thread Group 1-4	HTTP Request	255		1	2555	304	254
6	18:27:28.483	Thread Group 1-6	HTTP Request	245		1	2555	304	245
7	18:27:28.489	Thread Group 1-7	HTTP Request	241		1	2555	304	241
8	18:27:28.495	Thread Group 1-8	HTTP Request	244		1	2555	304	244
9	18:27:28.497	Thread Group 1-9	HTTP Request	250		1	2555	304	250
10	18:27:28.503	Thread Group 1-10	HTTP Request	244		1	2555	304	244
11	18:27:28.508	Thread Group 1-11	HTTP Request	272		1	2555	304	272
12	18:27:28.532	Thread Group 1-16	HTTP Request	248		1	2555	304	248
13	18:27:28.514	Thread Group 1-12	HTTP Request	268		1	2555	304	268
14	18:27:28.518	Thread Group 1-13	HTTP Request	264		1	2555	304	264
15	18:27:28.523	Thread Group 1-14	HTTP Request	263		1	2555	304	263
16	18:27:28.528	Thread Group 1-15	HTTP Request	258		1	2555	304	258

Ilustración 50. Tabla de resultados de rendimiento

Como se puede observar en la Ilustración 50. JMeter nos muestra una tabla donde aparece la información sobre la prueba de rendimiento sobre el endpoint indicado. En este caso las 2000 peticiones que se han realizado sobre el endpoint mencionado han resultado con un status OK, lo que indica que todas se han realizado correctamente y han obtenido una respuesta 200. Esto significa que la aplicación esta lista para soportar 2000 solicitudes en 10 segundos sobre ese endpoint sin comprometer el rendimiento de esta.

En resumen, la fase de pruebas es una fase con la misma importancia que la fase de diseño e implementación ya que esta nos permita aplicar prácticas para validar y verificar el correcto funcionamiento de nuestra aplicación, lo que garantiza una mejor experiencia de usuario y mejora del rendimiento de la aplicación. Además, es importante que la fase de pruebas no se haga al final del proyecto si no que se haga de manera iterativa. En este caso se ha llevado una metodología SCRUM con Sprints de una semana, es al final de cada Sprint que se ha hecho las pruebas para validar las nuevas funcionalidades implementadas durante este Sprint antes de hacer la entrega del MVP (Mínimo Producto Viable). Además, tener todas las funcionalidades validadas nos garantiza una mejora de la calidad del código, lo hace más legible y mantenible por lo que es una práctica que siempre debe llevarse a cabo si queremos un sistema fiable, robusto y de calidad.

7. Conclusiones

En este apartado se presentan las conclusiones de este Trabajo de Fin de Grado sobre una aplicación de gestión interna para grupos scout. Además, se ha identificado un camino futuro para mejorar la aplicación y adaptarla aún más al contexto scout.

7.1. Reflexiones finales

En cuanto al proceso de desarrollo, se ha llevado a cabo un enfoque metodológico sólido que ha permitido el diseño y la implementación efectiva de la aplicación de gestión interna para grupos scout. Se han llevado a cabo etapas cruciales como el análisis de requisitos, el diseño de la arquitectura, el desarrollo del código y las pruebas. Este enfoque ha garantizado que la aplicación cumpla con los objetivos establecidos y que esté alineada con las necesidades del grupo scout en el que se ha probado.

WeKraal podría convertirse en una herramienta fundamental en el ámbito del mundo scout. Ha simplificado y optimizado procesos administrativos y de comunicación, facilitando la gestión de tareas, eventos y recursos del grupo scout. Además, ha fomentado la colaboración y la participación activa de los miembros, promoviendo una mayor eficiencia y organización en las actividades del grupo.

7.2. Relación con asignaturas del Grado en Ingeniería Informática

Durante el desarrollo de esta aplicación, ha sido de gran ayuda los conocimientos adquiridos en el grado. Con asignaturas como, Proceso de Software, Diseño de Software, Calidad de Software e Ingeniería del Software he podido asentar las bases del proyecto, como la metodología seguida, la arquitectura del proyecto, la gestión de requerimientos y la implementación de buenas prácticas de codificación. Además, gracias a la asignatura Análisis y Validación de Software he aprendido a realizar pruebas para que la aplicación funcione correctamente.

7.3. Trabajo futuro

Como trabajo futuro, existen oportunidades para mejorar la aplicación y agregar funcionalidad adicional. Por ejemplo, se pueden implementar características como un sistema de seguimiento de méritos y logros scout, un módulo de planificación de actividades más avanzado y la integración con plataformas de mensajería instantánea para facilitar la comunicación entre los miembros.

Otro aspecto importante a considerar como trabajo futuro es dotar de una mayor personalidad scout a la aplicación. Esto implica añadir elementos visuales y estéticos que reflejen la esencia del movimiento scout, como iconografía scout, colores representativos de cada grupo scout que se registre en la aplicación y elementos de diseño inspirados en la naturaleza y el escultismo Esto ayudará a crear una experiencia de usuario más inmersiva y atractiva para los miembros de grupos scout.

Además, como parte del trabajo futuro, se plantea complementar WeKraal con otra aplicación móvil dedicada a los padres del grupo. Esta aplicación permitirá a los padres estar informados sobre las actividades, eventos y noticias del grupo scout, así como facilitar la comunicación con el equipo de comité y otros padres. Esto promoverá una mayor implicación de los padres en las actividades del grupo y fortalecerá la relación entre el grupo scout y las familias.

En resumen, el desarrollo de WeKraal ha demostrado ser un proceso sólido y ha evidenciado la importancia de una herramienta tecnológica de este estilo en el contexto scout. Las conclusiones del TFG señalan la necesidad de mejorar la funcionalidad existente, añadir características específicas del movimiento scout y complementar la aplicación con una aplicación móvil para los padres. Estas mejoras permitirán una gestión más eficiente, una mayor participación de los miembros y una experiencia scout más enriquecedora en los grupos scout que se unan a este proyecto de futuro.

Referencias

[1] Monday.com. (s.f.). ¿Qué es monday.com? Recuperado de https://support.monday.com/hc/es/articles/115005310945--Qu%C3%A9-es-monday-com-

[2] GanttPRO. (s.f.). Características de Asana: ventajas y desventajas. Recuperado de https://blog.ganttpro.com/es/caracteristicas-de-asana-ventajas-y-desventajas/#:~:text=Asana%20es%20un%20software%20de,cabo%20proyectos%20por%20su%20cuenta

[3] Atlassian. (s. f.). Scrum: qué es, cómo funciona y cómo empezar | Atlassian. Recuperado de https://www.atlassian.com/es/agile/scrum

[4] Nucba. (2022, 6 enero). ¿Qué es la arquitectura cliente-servidor? - NUCBA - Medium. Medium. Recuperado de https://nucba.medium.com/qu%C3%A9-es-la-arquitectura-cliente-servidor-eb9f402506cc

[5] Ribes, R. (2019, 7 de agosto). El patrón BFF (Back-end for Front-end). Recuperado de https://rlbisbe.net/2019/08/07/el-patron-bff-back-end-for-front-end/

[6] Pinzon, V. M. (2021). SOLID: Principio de Responsabilidad Única. DEV Community. Recuperado de https://dev.to/victorpinzon198/solid-principio-de-responsabilidad-unica-5ffo

[7] De Souza, I. (2021). Entiende las diferencias entre Front-End y Back-end en el ambiente de los sitios web. Rock Content - ES. Recuperado de https://rockcontent.com/es/blog/front-end-y-back-end/]

[8] ¿Qué es backend y para qué sirve en programación? (2022, 19 mayo). Recuperado de https://www.crehana.com/blog/transformacion-digital/que-es-el-backend-y-como-usarlo/

[9] ¿Qué es Java Spring Boot? (s/f). Ibm.com. Recuperado el 15 de junio de 2023, de https://www.ibm.com/mx-es/topics/java-spring-boot

[10] Empezando | Axios Docs. (s. f.). Recuperado de https://axios-http.com/es/docs/intro

[11] KeepCoding, R. (2023, 13 abril). ¿Qué es JWT (JSON Web Tokens)? | KeepCoding Bootcamps. KeepCoding Bootcamps. Recuperado de https://keepcoding.io/blog/que-es-jwt/

[12] Izertis. (s. f.). Encriptación de password en NodeJS y MongoDB: bcrypt. Izertis. Recuperado de https://www.izertis.com/es/-/blog/encriptacion-de-password-en-nodejs-y-mongodb-bcrypt

[13] Chakray. (2022, 10 diciembre). SWAGGER y SWAGGER UI: ¿Qué es y por qué es imprescindible en APIS? Chakray. Recuperado de https://www.chakray.com/es/swagger-y-swagger-ui-por-que-es-imprescindible-para-tus-apis/

Apéndice

A. Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.	X			
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.		X		
ODS 9. Industria, innovación e infraestructuras.	X			
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.			X	X
ODS 12. Producción y consumo responsables.				X
ODS 13. Acción por el clima.			X	
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

WeKraal tiene relación con varios de los ODS, con los que más seria con el 4 y el 9. Con el ODS de educación de calidad tiene gran relación ya que proporciona una plataforma de gestión para los miembros de los grupos scout que es permite agilizar los procesos internos, pudiendo pues dedicar más tiempo para promover una educación de calidad dentro del entorno scout. Con el objetivo número 9 ya que al utilizar tecnologías innovadoras para mejorar la gestión interna de los grupos scout se optimizan los procesos y promueven la eficiencia, WeKraal contribuye al desarrollo de infraestructuras sostenibles y promueve la innovación dentro del entorno scout.

Por otra parte, se ha indicado una relación de tipo “Media” con el ODS número 8 ya que WeKraal ofrece herramientas que pueden facilitar la organización y administración de los grupos scout, contribuyendo a un entorno de trabajo más eficiente y mejorando la

productividad. Sin embargo, su impacto en el crecimiento económico es nulo, ya que no está directamente relacionada con la generación de empleo o el fomento de oportunidades económicas.

Por último, se ha indicado que tiene una baja relación con los ODS 11 y 13 ya que en sí la aplicación no ofrece ninguna herramienta que tenga un impacto significativo sobre estos ODS, pero en el caso del número 11, WeKraal fomenta la colaboración y la comunicación entre los grupos scout y esto podría impulsar la creación de comunidades más fuertes y cohesionadas. Y con el número 13, aunque WeKraal no esté directamente relacionada con la mitigación del cambio climático, sí que promueve la planificación de actividades al aire libre, en contacto con la naturaleza, fomentando la conciencia ambiental en los grupos scout. Además, el hecho de tener las programaciones, fichas de actividad, inventarios de material, autorizaciones y demás documentos en una aplicación web podría considerarse una funcionalidad para la reducción del uso del papel, contribuyendo en menor medida a este ODS.