UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Implementación de un sistema de planificación de actividades y gestión de espacios de un centro educativo.

Autor

Daniel Moreno Jiménez

Directores

Óscar Ramón Adamuz Hinojosa

Juan José Ramos Muñoz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2024

Implementación de un sistema de planificación de actividades y gestión de espacios de un centro educativo.

Daniel Moreno Jiménez

Palabras clave: RESTLET, API, Automatización, Angular, Aplicación Web, Gestión de Actividades, Tecnología Educativa, Desarrollo de Software, MongoDB, Herramientas de Desarrollo

Resumen

En las instituciones académicas, como la E.T.S. de Ingenierías Informática y de Telecomunicación, existen, además de las clases habituales, diferentes tipos de actividades como charlas, talleres, etc. que enriquecen el conocimiento que los alumnos pueden adquirir en su etapa universitaria. Dichas actividades requieren de una planificación para agendarlas y hasta hoy en día, se les comunica al estudiantado de forma manual a través del correo de la Universidad de Granada.

Este proyecto aborda la necesidad existente en las instituciones educativas a la hora de optimizar la gestión de actividades académicas. Se ha detectado la existencia de un problema recurrente en las planificaciones y asignación de recursos. Esto se traduce en una utilización poco eficiente de las instalaciones y origina conflictos en la programación que afecta de forma directa a la experiencia educativa. Todo esto afecta tanto a profesores como a estudiantes, originando obstáculos en el proceso de enseñanza-aprendizaje.

Para resolver este desafío, se ha realizado una propuesta en la que se desarrolla una aplicación web que centralice y automatice la gestión de las actividades académicas. A través de esta solución tecnológica, se quiere dar la oportunidad a la comunidad educativa de una plataforma intuitiva y eficiente que les permita programar los diferentes eventos de una manera óptima, considerando la disponibilidad de recursos y minimizando los conflictos de programación. Para ello, se ha optado por un diseño e implementación de una aplicación web utilizando una base de datos acorde a las necesidades de la aplicación y que pueda albergar un gran conjunto de datos y puedan ser organizados de una manera correcta. Implementar un backend junto a una API (Application Programming Interface) que contengan funcionalidades para la gestión de las actividades y usuarios, además de poder realizarse distintas interfaces web o móviles que puedan usar esta API para utilizar las funciones desarrolladas en el backend. Al igual que un conjunto de reglas y protocolos que permite a diferentes aplicaciones software comunicarse e interactuar entre sí de manera efectiva y segura, ofreciendo un conjunto de funcionalidades a los usuarios que abarquen las necesidades que se requieren a nivel organizativo y que agilicen los procesos destinados a la gestión de las actividades, que junto a unas series de webs para que los usuarios disfruten de una interfaz amigable y una base de datos que contenga los recursos necesarios.

Implementation of an activity planning and space management system for an educational center.

Daniel Moreno Jiménez

Keywords: RESTLET, API, Automation, Angular, Web Application, Activity Management, Education Technology, Software Development, MongoDB, Development Tools

Abstract

In academic institutions, such as the E.T.S. de Ingenierías Informática y de Telecomunicación, there are, in addition to the usual classes, different types of activities such as lectures, workshops, etc. that enrich the knowledge that students can acquire in their university stage. These activities require planning in order to schedule them and until today, they are communicated to the students manually through the mail of the University of Granada.

This project addresses the need in educational institutions to optimize the management of academic activities. A recurrent problem has been detected in the planning and allocation of resources. This results in an inefficient use of facilities and causes conflicts in programming that directly affects the educational experience. All this affects both teachers and students, causing obstacles in the teaching-learning process.

To solve this challenge, a proposal has been made to develop a web application that centralizes and automates the management of academic activities. Through this technological solution, we want to provide the educational community with an intuitive and efficient platform that allows them to schedule the different events in an optimal way, considering the availability of resources and minimizing scheduling conflicts. For this, we have opted for a design and implementation of a web application using a database according to the needs of the application and that can accommodate a large set of data and can be organized in a correct way. Implement a backend together with an API (Application Programming Interface) containing functionalities for the management of activities and users, in addition to being able to create different web or mobile interfaces that can use this API to use the functions developed in the backend. As well as a set of rules and protocols that allow different software applications to communicate and interact with each other effectively and securely, offering a set of functionalities to users that cover the needs required at the organizational level and streamline processes for the management of activities, which together with a series of websites for users to enjoy a friendly interface and a database containing the necessary resources.

Yo, **Daniel Moreno Jiménez**, alumno de la titulación Grado en Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 26499389P, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Daniel Moreno Jiménez

Granada a 24 de Junio de 2024.

D. **Juan José Ramos Muñoz**, Profesor del Área de a de Ingeniería Telemática del Departamento Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Óscar Ramón Adamuz Hinojosa**, Profesor del Área de a de Ingeniería Telemática del Departamento Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Aulanda: Implementación de una Aplicación Web para gestionar las diferentes Actividades en la Escuela.***, ha sido realizado bajo su supervisión por **Daniel Moreno Jiménez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 24 de Junio de 2024 .

Los directores:

Juan José Ramos Muñoz **Óscar Ramón Adamuz Hinojosa**

Agradecimientos

Me gustaría expresar mi gratitud a aquellas personas que han sido un pilar fundamental en este camino. En primer lugar, agradecérselo a mi familia por su apoyo, animo y su creencia en cada paso que he tomado hasta llegar aquí. Su amor y su sacrificio han sido y serán mi mayor motivación y vitalidad.

A mis amigos, quienes han estado cerca mía durante todos estos años, les debo mi gratitud. Sus consejos, compañía y entendimiento han sido fundamentales en los momentos mas duros y en las victorias compartidas. Sin ellos, probablemente esto no seria posible.

A mi mismo, por mi tenacidad, esfuerzo y dedicación en cada una de las etapas de mi vida. Esta carrera ha sido un desafió constante, pero se ha superado todo fortaleciéndome como persona.

Finalmente, expresar mi sincero agradecimiento a aquellos profesores que han compartido su gran sabiduría, me han guiado y han sido una inspiración para mi en estos años.

A todos y cada uno de ustedes, mis mas profundo agradecimiento. Este logro nos pertenece a todos.

Índice general

1. Introducción. **1**
1.1. Introducción. . . . 1
1.2. Objetivos. . . . 2
1.3. Estructura de la Memoria. . . . 2

2. Planificación y estimación de costes. **5**
2.1. Planificación. . . . 5
2.1.1. Metodología. . . . 6
2.1.2. Cronograma. . . . 8
2.2. Estimación de costes. . . . 14
2.2.1. Software utilizado. . . . 15

3. Estado del Arte. **19**
3.1. Plataformas de gestión de recursos para relaciones entre organizaciones y clientes. . . . 19
3.1.1. Comparativas de plataformas relacionadas. . . . 19
3.2. Framework para desarrollo de aplicaciones web. . . . 25
3.2.1. React.js . . . 26
3.2.2. Vue.js . . . 27
3.2.3. Angular. . . . 28
3.2.4. Ember.js . . . 29
3.3. Framework de servicios web (RESTful). . . . 29
3.3.1. Express.js . . . 29
3.3.2. Django REST framework. . . . 30
3.3.3. Flask. . . . 31
3.3.4. RESTLET. . . . 32
3.4. Base de datos para aplicaciones web. . . . 33
3.4.1. Base de datos relacionales(SQL). . . . 33
3.4.2. Bases de datos no relaciones(NoSQL): . . . 34
3.4.3. MySQL. . . . 34
3.4.4. Microsoft SQL Server. . . . 35
3.4.5. MongoDB. . . . 36
3.4.6. Cassandra. . . . 36

4. Diseño de la aplicación. 39
4.1. Arquitectura general de la aplicación o servicio. . . . 39
4.2. Interfaz gráfica. . . . 41
4.2.1. Interfaz de la web de usuarios. . . . 42
4.2.2. Interfaz de la web de administración. . . . 43
4.3. Frontend. . . . 47
4.3.1. Web de usuarios. . . . 48
4.3.2. Web Administración. . . . 50
4.3.3. Comunicación entre clases. . . . 52
4.4. Backend. . . . 53
4.5. Servicio REST. . . . 55
4.6. BBDD. . . . 57
4.6.1. Diseño de bases de datos relacionales. . . . 57
4.6.2. Diseño de bases de datos no relacionales. . . . 58

5. Implementación de la aplicación. 59
5.1. Justificación del entorno de desarrollo. . . . 59
5.2. BBDD. . . . 60
5.3. Backend. . . . 63
5.3.1. ExampleRestlet. . . . 65
5.3.2. ServerWeb y LoginWeb. . . . 68
5.3.3. AuthenticationController. . . . 72
5.3.4. VerificadorUsuarios. . . . 74
5.3.5. SecretKeyManager. . . . 75
5.3.6. JwtVerifierAuth. . . . 76
5.3.7. JwtVerifier. . . . 77
5.3.8. CreateNewUser. . . . 78
5.3.9. SearchForParameters. . . . 79
5.3.10. ManageActivities. . . . 80
5.3.11. MongoDBManager. . . . 81
5.3.12. EmailResource. . . . 82
5.3.13. TwitterClient. . . . 84
5.4. Frontend. . . . 85
5.4.1. Preparación del Entorno. . . . 86
5.4.2. Página de actividades de usuarios. . . . 87
5.4.3. Página de actividades de Administración. . . . 89

6. Resultados de la implementación y prueba de funcionamiento. 95
6.1. Web de usuarios. . . . 95
6.2. Web de administración. . . . 98

7. Conclusiones y trabajo a futuro. 109

Índice de figuras

4.1. Arquitectura general del servicio. 41
4.2. Diseño de la interfaz web de usuarios. 42
4.3. Diseño de la interfaz web de administración. 43
4.4. Diseño de la parte de gestión de usuario en la interfaz web de administración. 45
4.5. Diseño de la parte de gestión de actividades en la interfaz web de administración. 46
4.6. Diagrama UML inicial de la interfaz web de usuarios. 48
4.7. Diagrama UML inicial de la interfaz web de administración. 50
4.8. Diagrama UML inicial del Backend. 53

5.1. Esquema de la aplicación web con los frameworks elegidos. 61
5.2. Resultado de la verificación de la instalación. 62
5.3. Diagrama UML del backend. 64
5.4. Diagrama UML del backend. 65
5.5. Diagrama UML del backend. 66
5.6. Verificación de la instalación de Node.js y npm. 86
5.7. Verificación de la instalación de AngularCLI. 87

6.1. Home de la web de usuarios. 96
6.2. Componente Search de la web de usuarios. 96
6.3. Comunicación entre la web de usuarios y la API. 97
6.4. Headers de la petición OPTIONS. 97
6.5. Headers de la petición GET. 98
6.6. Vista de la web de usuarios tras realizar una búsqueda de actividades. . . 99
6.7. Vista del calendario con actividades de ejemplo agendadas. 100
6.8. Peticiones en la comunicación para recoger todas las actividades. 100
6.9. Aspecto del login de la web de administración. 101
6.10. Mensaje de denegación al acceso a los recursos con credenciales invalidas o inexistentes. 101
6.11. Vista del componente home. 102
6.12. Petición con autenticación correcta. 102
6.13. Componente users de la pagina de Administración. 103
6.14. formulario para la creación de actividades. 104

6.15. Parte de la tabla de actividades. 105
6.16. Parte de la tabla de actividades. 106
6.17. Doble confirmación sobre el borrado de la aplicación. 107
6.18. Doble confirmación sobre el borrado de la aplicación. 107
6.19. Respuesta sobre la petición para modificar una actividad. 107

Capítulo 1

Introducción.

1.1. Introducción.

En el sistema de educación implantado en las distintas facultades y escuelas de las universidades de España, no solo existen en su día a día las clases sobre las diferentes asignaturas impartidas por el profesorado, sino que también se realizan un gran número con diversos tipos de actividades que tienen como objetivo nutrir al estudiante de conocimientos y habilidades para su desarrollo profesional, dar información a los mismos sobre las tecnologías del mercado actual, realizar retos para que el estudiante pruebe sus capacidades, informar sobre las nuevas tendencias tanto a nivel laboral como en el ámbito tecnológico, entre otras muchas.

Estas actividades las realizan en mayor parte los profesores que conforman el equipo docente de cada facultad o escuela de cada universidad, aunque a veces también se ponen en contacto empresas u otros entes del ámbito en el que se centra la institución para crear y organizar estas actividades. El proceso que existe hoy en día es algo primitivo, ya que para que se gestionen este tipo de actividades, por ejemplo, un profesor que quiera realizar una charla sobre el ámbito tecnológico y quiere que otro investigador sea el ponente de la misma, tiene que ver la disponibilidad de dicha persona y, a la vez, contactar con la persona encargada en su escuela para concertar una reunión y ver la disponibilidad que existe sobre los espacios del centro habilitados a tal fin, fechas donde no se pisen con otras actividades docentes,etc. Esto supone una carga adicional al profesorado, que dificulta su trabajo diario y entorpece sus labores como investigador y docente.

Como hemos comentado anteriormente en este capítulo, actualmente las actividades que se realizan en la Escuela se gestionan y planifican de una forma manual, ya que no tienen un sistema o aplicación que le ayude a dicha administración. Aun existiendo aplicaciones de gestión, hoy en día no existe ninguna que esté pensada para este propósito o no se ajustan realmente a los requerimientos específicos que exige este tipo de gestiones. En el ámbito actual de la educación, una gestión eficiente de recursos y actividades es

básica para un buen funcionamiento de las instituciones educativas. Debido al avance de las tecnologías de la información y la comunicación, surgen ciertas necesidades que abogan al desarrollo de herramientas que optimicen la planificación y gestión dentro de las mismas instituciones.

Por ello, en el presente proyecto, se propone la implementación de una aplicación que ayude a gestionar este tipo de actividades, a través de un sistema donde se vean claros dónde y cuándo se van a realizar este tipo de actividades y el profesorado pueda crear y modificar a su antojo estas actividades. Los alumnos tengan de una manera centralizada estas actividades y se les habiliten herramientas para saber cuándo se realizan y saber cuáles son las que más les interesan según sus metas profesionales, así como el desarrollo de métodos automatizados para la publicación de las mismas en diferentes plataformas para su divulgación.

1.2. Objetivos.

En este proyecto, el objetivo es el desarrollo de una aplicación web que permita la gestión de actividades académicas dentro de la E.T.S. de Ingenierías Informática y de Telecomunicación y que podría escalar a las demás Facultades de la Universidad de Granada. Para lograr dicho objetivo se proponen los siguientes sub-objetivos:

- **sub-objetivo 1**: Gestión de Actividades Académicas: La aplicación en sí, proporcionará herramientas para que el profesorado programe sus charlas, laboratorios y otros eventos. Mientras tanto, los estudiantes tendrán a su disposición una web donde podrán consultar dichos eventos y recibirán notificaciones sobre los cambios en las actividades.
- **sub-objetivo 2**: Desarrollo de una API (Interfaz de Programación de Aplicaciones): Se desarrollará una API que facilite una integración de la aplicación con otros sistemas o servicios externos. Esto ayudará a la interoperabilidad y el crecimiento futuro de la aplicación.
- **sub-objetivo 3:** Desarrollo de una Aplicación Web: Se pretende diseñar una aplicación web utilizando tecnologías actuales (HTML5,CSS3, JavaScript, Angular, etc.) para brindar una interfaz intuitiva y afable para el usuario.

1.3. Estructura de la Memoria.

En este apartado se muestra una visión general de la organización y disposición del documento, delineando el contenido que se aborda en cada uno de los capítulos:

- En el capítulo 1 **Introducción**, proporciona en contexto del trabajo de fin de grado y presenta la motivación para desarrollar una aplicación para gestionar efi-

cientemente las actividades de la escuela y los objetivos para desarrollarla. Este apartado sirve como punto inicial para la comprensión, propósito y relevancia del proyecto.

- En el capítulo 2 **Planificación y estimación de costes**, se detalla la metodología utilizada y se muestra un cronograma detallado del proyecto. Además, se ha realizado una estimación de costes que se vienen asociados con el desarrollo de la aplicación, donde se incluye el software utilizado y las horas dedicadas de trabajo.

- En el capítulo 3 **Estado del arte** se proporciona una revisión de otros sistemas similares, así como una descripción de las plataformas existentes con sus ventajas e inconvenientes. Adicionalmente, se analizan los frameworks que se han utilizado para la implementación de aplicaciones web, servicios web (RESTful) y las bases de datos para aplicaciones web.

- En el capítulo 4 **Diseño de la aplicación** se presenta la arquitectura general de la aplicación o servicio, donde se incluye la interfaz gráfica, el *fronted*, el *backend*, el servicio REST y la base de datos. En él se detallan qué decisiones de diseño se han tomado durante el proceso de implementación.

- En el capítulo 5 **Implementación de la aplicación** se definen los frameworks utilizados durante la fase de desarrollo de la aplicación, proporcionando información sobre las tecnologías que se han utilizado y qué recursos se han empleado.

- En el capítulo 6 **Resultados de la implementación y pruebas de funcionamiento**, se realizan pruebas de funcionamiento donde se evalúa la efectividad y el rendimiento de la aplicación, con el objetivo de validar el cumplimiento de los criterios establecidos.

- En el capítulo 7 **Conclusiones y trabajo a futuro**, se exponen las conclusiones obtenidas tras la implementación, se resumen a grandes rasgos las funciones principales de la librería, la solución del problema que resuelve y una serie de objetivos de trabajo futuro con los que se puede llegar a mejorar la eficiencia de la aplicación.

Capítulo 2

Planificación y estimación de costes.

En este capitulo se detalla la planificación prevista y la estimación de costes que están asociadas a la implementación de la aplicación. Este es un aspecto muy importante para conseguir el éxito de cualquier proyecto, ya que brinda una hoja de ruta estructurada y clara para realizar su ejecución adecuadamente. Adicionalmente, la estimación de costes es crucial para que se garantice los recursos necesarios y poder respetar el presupuesto asignado al proyecto.

2.1. Planificación.

Antes de iniciar cualquier proyecto, es necesario establecer una planificación segura que sirva de guía para todas y cada una de las actividades y permitan la realización eficiente y efectiva. En este apartado, se describe de forma detallada la planificación de este proyecto, desde la especificación de objetivos hasta la asignación de tareas y recursos.

La planificación incluye diversos aspectos, como el de definir las metas del proyecto, definición de responsabilidades, programación de las mismas y la estimación de recursos necesarios. Igualmente, se establecen los plazos de ejecución en cada una de las fases del proyecto, con el fin de garantizar el cumplimiento de los objetivos en el tiempo estimado.

Una adecuada planificación brinda la anticipación a posibles obstáculos y permite tomar medidas para disminuir su impacto. Adicionalmente, proporciona una base sólida en la toma de decisiones y una eficaz gestión de los recursos existentes.

En los siguientes apartados se detalla la metodología utilizada para la realización de la planificación del proyecto y la estimación de los costes asociados en la implementación de la aplicación.

2.1.1. Metodología.

Introducción.

La metodología de trabajo que hemos seguido para este trabajo de fin de grado es un elemento fundamental que orienta la ejecución del mismo. Se ha elegido en este caso la utilización de una metodología ágil, en específico, el marco de trabajo Scrum [1]. SCRUM es una metodología ágil que se enfoca en la entrega iterativa e incremental de productos, lo que facilita una adaptación de forma continua a medida que se avance en el proyecto. Scrum se identifica por su enfoque en equipos autoorganizados, iteraciones de un breve periodo de tiempo llamados *sprints* y una colaboración cercana entre todos y cada uno de los miembros existentes en el equipo. Dicha metodología está basada en una serie de roles, eventos y elementos que ayudarán a la estructura del trabajo y a que se garantice una entrega continua de valor.

Los roles principales en Scrum son:

- **Product Owner:** Principal integrante en la gestión del *backlog* del proyecto y la priorización de las funcionalidades a implementar. El *backlog* es una lista de todo el trabajo que se conoce y que se necesita realizar en el proyecto.
- **Scrum Master:** Persona encargada para ayudar en el proceso y eliminar los obstáculos que puedan surgir.
- **Equipo de Desarrollo:** Encargado de la realización del trabajo que se necesita para entregar las funcionalidades definidas en el *backlog*.

Los eventos clave en Scrum son:

- **Planificación del Sprint:** Se definen los objetivos y tareas del sprint.
- **Revisión del Sprint:** Evaluación del trabajo completado en el final de cada sprint.
- **Retrospectiva del Sprint:** Etapa de reflexión para mejorar el proceso en los futuros sprint.
- **Daily Scrum:** Reuniones que se realizan diariamente para ayudar a la coordinación del equipo.

Los elementos principales en Scrum son:

- **Backlog del Producto:** Listado que alberga todas y cada una de las funcionalidades que se desean desarrollar.
- **Backlog del Sprint:** Numero de funcionalidades que el equipo debe realizar en un sprint específico.

Justificación de la elección de la metodología.

Este tipo de metodología ha sido elegida ya que tiene una gran capacidad de adaptación al cambio y se centra en la entrega continua de valor al cliente. Ya que la implementación de software es un proceso que suele ser complejo y es cambiante durante su proceso, Scrum nos brinda un marco de trabajo flexible que facilita de una forma rápida responder a dichos cambios dentro de los requisitos del proyecto y cubrir las necesidades del cliente. Adicionalmente, el trabajo con interacciones de corto periodo de tiempo y entregas frecuentes nos brinda la obtención de una retroalimentación temprana del cliente para poder ajustar el enfoque del proyecto en consecuencia, lo que aumenta las posibilidades de éxito del proyecto.

Principios y practicas recomendadas.

La aplicación de Scrum de este trabajo de fin de grado se lleva a cabo siguiendo los principios y prácticas recomendadas por el marco de trabajo. Se comienza definiendo el backlog del proyecto en colaboración con el cliente, donde se declararán las funcionalidades y características claves que se desean implementar. Seguidamente, se planifican los sprints donde se escoge un conjunto de funcionalidades del backlog para que sean implementadas durante cada iteración. Durante dicho sprint, se celebran reuniones de forma regular para poder revisar el progreso y ajustar, en caso de ser necesario, el plan. Cuando se termine cada uno de los sprint se realiza una revisión para mostrar las funcionalidades desarrolladas y obtener una retroalimentación del cliente. Además, se realiza una retrospectiva donde se identifican las áreas de mejora y el ajuste del enfoque para el siguiente sprint.

Herramientas y recursos disponibles.

Para apoyar la aplicación de Scrum, se utilizan una variedad de herramientas y recursos, donde se incluyen software para al gestión de proyectos como Jira [2] o Trello [3] que se utilizan para gestionar el backlog y realizar un seguimiento del avance del mismo. Además, se utilizan herramientas como Google Meet para facilitar la comunicación en reuniones regulares de seguimiento, como el Daily Scrum, donde se mantienen sincronizados los tutores tanto como el alumno y se asegura que se esta avanzando en la dirección idónea para el desarrollo del trabajo de fin de grado.

Se ha usado la terminología equipo de desarrollo y cliente, la que es usada para definir los conceptos de Scrum, en este caso cada uno de estos entes se refieren al alumno que realiza el proyecto y a los profesores que le indican pautas y el contenido de la aplicación según sus necesidades a la hora de llegar a utilizarlo.

2.1.2. Cronograma.

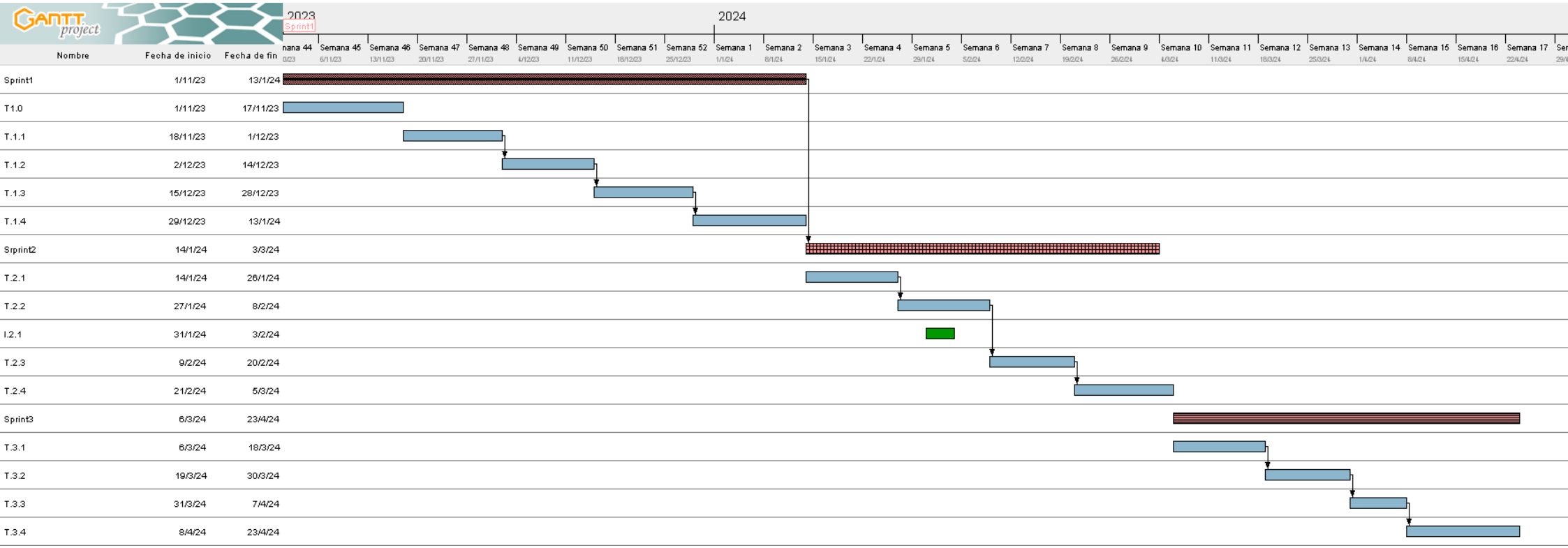
El cronograma es una herramienta fundamental para gestionar el tiempo y los recursos en un proyecto de una forma eficiente y además sirve de referencia para el cumplimiento de los plazos estipulados logrando los objetivos del mismo en el tiempo que se han previsto. A continuación se describe el cronograma propuesto en este trabajo de fin de grado:

- **Definición de Actividades:** Dentro de esta etapa, se han identificado todas y cada una de las actividades que son necesarias para llevar a cabo el proyecto. Esto se basa en los entregables que se han definido en el alcance del mismo. Dentro de las actividades e incluye tareas de análisis, diseño, implementación, pruebas y documentación.
- **Secuenciación de Actividades:** Al tener identificadas todas las actividades, se estipula el orden en que se deben realizar, donde e tiene en cuenta las relación entre ellas. Esto asegura una realización coherente y fluida del trabajo, reduciendo los posibles atrasos y cuellos de botella.
- **Estimación de la Duración de las Actividades:** Cada actividad tendrá una estimación según la cantidad de tiempo que requiera su elaboración. Los parámetros en los que se basa la estimación son la experiencia previa, el conocimiento del equipo o persona y cualquier otra información disponible para dicho propósito.
- **Creación del Cronograma:** Una vez recopilada la información de los puntos anteriores, implementaremos el cronograma. En este se visualiza el orden de actividades a lo largo del tiempo, al igual que los plazos de inicio y finalización de las actividades.
- **Asignación de Recursos:** El cronograma adicionalmente también incluye como se asignan los recursos necesarios para llevar a cabo cada actividad. Aquí se pueden incluir recursos humanos, financieros, materiales y tecnológicos. Dado el alcance de este proyecto este cronograma se ve reducido en este apartado dado que mucho de las conceptos nombrados están muy simplificados.
- **Seguimiento y Control:** Una vez que se ha creado el cronograma, se establece un seguimiento de forma regula para revisar el progreso del proyecto asegurando que se estén cumpliendo los tiempos establecidos. Además, se identifican y gestionan posibles desviaciones, tomando las acciones para mantener el proyecto en la dirección que debe.

A continuación se muestran en unas tablas 2.1 y 2.2 la duración de los sprints en los que se ha dividido este proyecto y cada una de sus tareas, junto con información relevante a la duración de las mismas, para que se pueda relacionar con el diagrama de Gantt que también se adjunta entre ellas en el documento y se pueda interpretar de una manera correcta:

Sprint/tareas	Fecha Inicio	Fecha Fin
Sprint 1	13/11/2023	13/01/2024
T.1.1 - Crear Servicio Restlet con las operaciones CRUD en base de datos MongoDB	18/11/2023	01/12/2023
T.1.2 - Creación de las diferentes base de datos y colecciones en MongoDB e integración con la API	02/12/2023	14/12/2023
T.1.3 - Creación de la Web de usuarios e integración con la API.	15/12/2023	29/12/2023
T.1.4 - Adición de Autenticación y autorización en las Bases de Datos y API	30/12/2023	13/01/2024
Sprint 2	13/01/2024	04/03/2024
T2.1 - Estructuración y diseño de la página web de administración.	13/01/2024	26/01/2024
T2.2 - Creación de los diferentes Componentes.	27/01/2024	08/02/2024
I2.1 - Creación de filtro Cors personalizado para evitar conflictos con el navegador.	31/01/2024	04/02/2024
T2.3 - Creación de filtro dedicado para la gestion de actividades.	09/02/2024	21/02/2024
T2.4 - Integración de las funcionalidades con la API.	22/02/2024	06/03/2024

Cuadro 2.1: Planificación del proyecto.


GANTT project
2023
2024
Sprint1
Nombre
Fecha de inicio
Fecha de fin
Semana 44
Semana 45
Semana 46
Semana 47
Semana 48
Semana 49
Semana 50
Semana 51
Semana 52
Semana 1
Semana 2
Semana 3
Semana 4
Semana 5
Semana 6
Semana 7
Semana 8
Semana 9
Semana 10
Semana 11
Semana 12
Semana 13
Semana 14
Semana 15
Semana 16
Semana 17
Sprint1 1/11/23 13/1/24
T1.0 1/11/23 17/11/23
T.1.1 18/11/23 1/12/23
T.1.2 2/12/23 14/12/23
T.1.3 15/12/23 28/12/23
T.1.4 29/12/23 13/1/24
Srprint2 14/1/24 3/3/24
T.2.1 14/1/24 26/1/24
T.2.2 27/1/24 8/2/24
I.2.1 31/1/24 3/2/24
T.2.3 9/2/24 20/2/24
T.2.4 21/2/24 5/3/24
Sprint3 6/3/24 23/4/24
T.3.1 6/3/24 18/3/24
T.3.2 19/3/24 30/3/24
T.3.3 31/3/24 7/4/24
T.3.4 8/4/24 23/4/24

Sprint 3	06/03/2024	24/04/2024
T3.1 - Creación de componentes y servicios en la página para el login.	06/03/2024	19/03/2024
T3.2 - Creación y adaptación en la API por autenticación mediante JWT.	20/03/2024	31/03/2024
T3.3 - Creación de la funcionalidad de exportar actividades a plantillas.	01/04/2024	08/04/2024
T3.4 - Automatización de para el envío de actividades a twitter o correo.	09/04/2024	24/04/2024

Cuadro 2.2: Planificación del proyecto.

Descripción de los Sprint.

- **Sprint 1. Investigación e implementación de las funciones principales de la aplicación web:** En este sprint los objetivo principal a desarrollar trata sobre la investigación e implementación de las funciones principales de la aplicaciones web.
- **Sprint 2. Implementación de interfaz de usuario principal:** En este sprint el objetivo principal se centro en la implementación de todos los componentes y la integración con las funcionalidades ya existentes.
- **Sprint 3. Implementación de funcionalidades adicionales:**En este sprint se marca coo objetivo desarrollar algunas de las posibles funciones mas importantes para gestionar las necesidades para que sea una aplicación funcional.

Descripción de las tareas.

- **T.1.0 - Estudio de la existencia y compatibilidad con el proyecto:** En esta primera etapa del proyecto, se centra en la búsqueda de las opciones que más se ajusten a los requerimientos técnicos que existen. Por ello se realiza una investigación que ayude a definir el tipo de framework, bases de datos..etc que se necesitan para la implementación del mismo.

- **T.1.1 - Crear Servicio Restlet con las operaciones CRUD en base de datos MongoDB:**
- **T.1.2 - Creación de las diferentes base de datos y colecciones en MongoDB e integración con la API** En esta tarea se definen y crean las bases de datos y colecciones que precisa la aplicación para el adecuado almacenamiento de los datos que albergara. Además se establece la conexión entre MongoDB y la API desarrollada en la tarea realizada con anterioridad, donde se asegura la comunicación entre ambos componentes.
- **T.1.3 - Creación de la Web de usuarios e integración con la API:** En esta tarea se realiza el desarrollo de la interfaz web que utilizaran los usuarios (alumnos), la cual brinda la interacción con la aplicación. En ella se diseña la página y componentes que se necesitan para la que los usuarios puedan visualizar las actividades segun sus necesidades. Además se integra la interfaz con la API desarrollada en tareas previas para asegurar el flujo de datos de forma eficiente entre el fronted y el backend.
- **T.1.4 - Adicción de Autenticación y autorización en las Bases de Datos y API:** En esta tarea se crea por la necesidad de añadir un sistema de autenticación y autorización que garantice seguridad en la aplicación. Se implementan mecanismo de autenticación básica en la API y se establecen permisos de acceso adecuado en las bases de datos en MongoDB, que aseguran que solo los usuarios autorizados puedan acceder y modificar la información en ellas.
- **T2.1 - Estructuración y diseño de la página web de administración:** En esta tarea se realiza el diseño y la estructuración de la página web de administración, la cual brinda a los administradores (profesorado) herramientas para la gestión y supervisión de la aplicación. En ella se define la arquitectura de la página y se diseñan los elementos visuales que garantizan una experiencia de usuario intuitiva y eficiente.
- **T2.2 - Creación de los diferentes Componentes:** En esta tarea se implementan los diferentes componentes necesarios para la página web de administración. Se crean los elementos necesarios que permiten a los administradores gestionar las actividades y recursos de la aplicación de manera adecuada.
- **I2.1 - Creación de filtro CORS personalizado para evitar conflictos con el navegador:** Esta incidencia se originó durante el desarrollo del proyecto relacionado con los conflictos CORS (Cross-Origin Resourcer Sharing) en el navegador. Por lo tanto se investigó e implementó un filtro CORS personalizado que evita los conflictos y asegura la comunicación correcta entre el *fronted* y el *backend*.
- **T2.3 - Creación de filtro dedicado para la gestión de actividades:** Esta tarea se creó por la necesidad de tener un filtro dedicado en la gestión de actividades académicas en la página web de la administración. este filtro brinda la búsqueda

y filtrado de una manera eficiente, mejorando de tal manera la experiencia de usuario.

- **T2.4 - Integración de las funcionalidades con la API:** En esta tarea se realiza la integración de las funcionalidades desarrolladas con la API, estableciendo una comunicación bidireccional entre la página web de administración y y el servicio Web.

- **T3.1 - Creación de componentes y servicios en la página para el login:** En esta tarea se desarrollan los componentes y servicios necesarios en la página web de administración para la autenticación y el inicio de sesión. Para ello, se crean formularios de inicio de sesión, así como la implementación de servicios para la gestión de sesiones de usuarios para una autenticación segura.

- **T3.2 - Creación y adaptación en la API por autenticación mediante JWT:** En esta tarea se realizan los cambios necesarios en la API para proveer a la API de autenticación mediante JSON Web Tokens (JWT). Se implementarán los mecanismos necesarios para la generación, validación y gestión de tokens JWT que garantizan un proceso seguro y fiable de autenticación.

- **T3.3 - Creación de la funcionalidad de exportar actividades a plantillas:** En esta tarea se implementa la funcionalidad de exportar las actividades que se quieran en un formato Mark Down (.md) que le da compatibilidad para cualquier tipo de editor y una fácil visualización. Para ellos se desarrollan los mecanismos necesarios en la página web de administración para generar y descargar dichas plantillas.

- **T3.4 - Automatización para el envío de actividades a twitter(X) o correo:** En esta tarea se desarrolla la funcionalidad de automatizar el envío de una serie de actividades seleccionadas a través de correo electrónico o a la red social de X (antiguamente llamada Twitter). Esto se realiza para mejorar la difusión y comunicación de las actividades dentro de la comunidad educativa de la Escuela Técnica Superior.

2.2. Estimación de costes.

La estimación de costes es una parte determinante en la planificación del proyecto, porque permite definir los recursos financieros necesarios para realizar el desarrollo del mismo de una manera eficaz. A continuación, se especifica la estimación de costes tanto de software como de personal que están asociados a este trabajo de fin de grado:

2.2.1. Software utilizado.

Durante la implementación del proyecto, se han utilizado varias herramientas de software que ejercen un papel fundamental en la gestión, desarrollo y despliegue de la aplicación. A continuación, se listan dichas herramientas utilizadas con una descripción de cada una de ellas:

- **Jira Software:** Jira [4] es una herramienta de gestión muy utilizada que permite el seguimiento de errores, incidencias y tareas en el desarrollo de un proyecto. Ha sido desarrollada por Atlassian y brinda una plataforma centralizada para realizar la planificación, colaboración y seguimiento del progreso del proyecto.
- **Java Platform Standard Edition 17 Development Kit:** JDK [5] se considera un entorno de desarrollo idóneo para crear aplicaciones y componentes que utilizan un lenguaje de programación Java. Aquí se incluyen ciertas herramientas con importancia para el desarrollo, prueba y monitorización de programas escritos en Java.
- **Visual Studio Code:** Esta herramienta [6] en un entorno de desarrollo integrado (IDE) que se utiliza para escribir, depurar y desplegar código. Contiene un gran abanico de extensiones y características que habilitan el desarrollo de aplicaciones en diversos lenguajes de programación.
- **GitHub:** Dicha plataforma [7] tiene la función de desarrollar de forma colaborativamente el proyecto, además de proporcionar un control de versiones en Git y funciones de gestión de de proyectos. Ha sido utilizada para el almacenamiento y gestión del código fuente.
- **Framework RESTLET:** Este framework [8] ha sido utilizado para la creación de la API de la aplicación. Esta herramienta proporciona un abanico de funcionalidades para la implementación de servicios web RESTful de una forma escalable y eficiente.
- **MongoDB:** Esta herramienta [9] se caracteriza por ser una base de datos no estructural orientada a documentos que se utiliza para almacenar y gestionar los datos de la aplicación. Brinda una escalabilidad y flexibilidad que lo hacen idónea para aplicaciones que necesitan un almacenamiento de datos ágil y dinámico.
- **Angular:** Se define como un framework [10] para desarrollar aplicaciones de una sola página (SPA) en el lado del cliente. Se utiliza HTML y TypeScript, permitiendo crear una interfaz de usuario que sea interactiva y dinámica. Dicha herramienta ha sido utilizada para implementar la interfaz de usuario y administración de la aplicación.
- **Google Meet:** Esta herramienta [11] ha sido utilizada para tener reuniones semanales con los profesores del proyecto. Utilizando una plataforma de comunicación

en línea y eficaz para poder discutir a lo largo del proceso de desarrollo los problemas y posibles soluciones y recibir retroalimentación del mismo.

Un aspecto en común y que hay que destacar de todas estas herramientas es que han sido utilizadas ya que son gratuitas. La justificación de todas y cada una de ellas se realiza en secciones posteriores donde se comparan con otras existentes en el mercado.

A continuación mostramos una tabla 2.4 con los costes asociados a cada una de las herramientas descritas anteriormente. Una vez se ha desarrollado los costes asociados al software utilizado, se calculan los costes asociados a los recursos humanos realizados en este proyecto 2.5. Este cálculo se ha basado en una estimación sobre las retribuciones que recomienda la Oficina de Transferencia de Resultados de Investigación (OTRI) de la Universidad de Granada [12].

La tabla 2.3 que contiene la suma total de costes, el total de este proyecto seria de unos 16.691,52 Euros. Lo que resulta un proyecto bastante económico, gracias a que la elección de los frameworks y plataformas que se han utilizado para implementar la aplicación, son de uso gratuito.

Tipo de costes	**Precio**
Recursos humanos	16.691,52 Euros
Software	0 Euros
Total	16.691,52 Euros

Cuadro 2.3: Estimación de costes de recursos humanos.

Herramienta	**Tipo de licencia**	**Coste**
Jira Software	Licencia gratuita basada en una licencia "Gratis para siempre"	0 Euros
Java Platform Standard Edition 17 Development Kit	GNU General Public License	0 Euros
Visual Studio Code	MIT License	0 Euros
GitHub	Licencia gratuita para repositorios públicos, cubierta por MIT License.	0 Euros
Framework RESTLET	Apache License 2.0	0 Euros
MongoDB	Sever Side Public License (SSPL)	0 Euros
Angular	MIT license	0 Euros
Google Meet	Es una herramienta que forma parte de Google workspace se rige por los Términos de servicio de Google	0 Euros

Cuadro 2.4: Estimación de costes de software.

Recursos humanos	**Precio**	**Duración**	**Total**
Desarrollador de software (Ingeniero de telecomunicaciones)	19,96 euros/hora	Invetigacion 78h + Desarrollo 642 h	14.371,2 Euros
Tutor 1 (Profesor contratado Doctor)	24,17 euros/hora	48 horas	1160,16 Euros
Tutor 2 (Profesor contratado Doctor)	24,17 euros/hora	48 horas	1160,16 Euros

Cuadro 2.5: Estimación de costes de recursos humanos.

Capítulo 3

Estado del Arte.

En este capitulo se realiza un análisis del estado actual de la tecnología relacionada con las aplicaciones de gestión. Además de examinar otros sistemas similares, se comparan frameworks de desarrollo de aplicaciones web, frameworks de servicios web (RESTful) y base de datos utilizadas en dicho contexto.

3.1. Plataformas de gestión de recursos para relaciones entre organizaciones y clientes.

3.1.1. Comparativas de plataformas relacionadas.

Hoy en día, la mayoría de aplicaciones para la gestión de actividades educativas se centran en la relación profesor-asignatura-alumno y no en las otras muchas actividades que se realizan fuera de este ámbito y además de que cuenten con procesos automatizados para la publicación de estas actividades en diferentes plataformas. A continuación, se presentan plataformas que hoy en día se utilizan en el mundo empresarial para la gestión y automatización de recursos entre empresas y clientes. Este tipo de aplicaciones son las más similares en el mercado a la que se desarrolla en este trabajo de fin de grado.

Por ello, se describen varios ejemplos de sistemas similares que hoy en día existen en el mercado relacionados con temas de gestión de recursos. Se describirán sus características, funcionalidades, ventajas e inconvenientes, donde se identificarán las principales soluciones disponibles y su aplicación en diferentes contextos.

Buffer.

Buffer [13] es una plataforma íntegra de gestión de redes sociales que brinda una amplia gama de herramientas que facilitan a usuarios y organizaciones gestionar sus perfiles sociales de manera eficiente. Dicha plataforma concede la oportunidad de programar y

publicar contenido en varias redes sociales como X, Facebook, Pinterest o LinkedIn, desde una única interfaz amigable al usuario. Además, ofrece varias herramientas avanzadas que tienen una finalidad de análisis que proporciona información detallada sobre el rendimiento de publicaciones, donde se incluyen estadísticas de participación, alcance y compromiso. Esto permite realizar un seguimiento del éxito de sus campañas y el ajuste de su estrategia de marketing en redes sociales según sea necesario.

A continuación se listan las **ventajas** de esta aplicación:

- Interfaz intuitiva y fácil de usar, idónea para principiantes y profesionales.
- Permite la programación de publicaciones en múltiples redes sociales desde una única plataforma, lo que reduce el tiempo y simplifica la gestión de contenido.
- Herramientas de análisis avanzadas que ofrecen información detallada sobre el rendimiento de las publicaciones. Esto permite a los usuarios decidir su estrategia de marketing en redes sociales.
- Programación flexible y personalizada de publicaciones, que brindó a los usuarios el ajuste de frecuencia y el momento de las publicaciones según sus necesidades.
- Se integra con diversas herramientas populares de marketing en redes sociales, como Canva y Pablo, para facilitar la creación de contenido visual de una forma atractiva.
- Tiene capacidad de compartir y colaborar en equipo en la gestión de perfiles sociales, mejorando la eficiencia y coordinación entre los miembros del equipo.
- Ofrece un soporte técnico y servicio al cliente útil, asistiendo de una manera rápida y dando solución a los problemas.

Después de nombrar las ventajas más relevantes, se presentan los **inconvenientes** que se han analizado:

- Existen limitaciones en cuanto al número de cuentas y programaciones dependiendo del plan de suscripción, lo que puede llegar a ser restrictivo para usuarios con necesidades más avanzadas.
- La integración implementada con algunas redes sociales puede ser limitada si se compara con otras plataformas de gestión de redes sociales.
- No permite utilizar las funciones avanzadas de análisis y reporte en su plan básico. Esta hecho puede limitar la capacidad de los usuarios en la realización de un seguimiento detallada del rendimiento en sus campañas.

- Respecto a la automatización, puede llegar a ser limitada si se compara con otras plataformas de gestión de redes sociales, lo que puede requerir una supervisión más activa por parte de los usuarios.
- Algunas de sus funciones avanzadas, como la personalización del contenido y la segmentación de audiencias, pueden estar disponibles solo en planes de subscripción más caros.
- La integración con herramientas de terceras puede llegar a ser limitada y requiere instalar complementos adicionales, pudiendo llegar a aumentar costos y complejidad.
- La curva de aprendizaje para exprimir al máximo todas las características y funcionalidades de Buffer puede ser complicada, sobre todo para usuarios nuevos en la plataforma.

HubSpot.

HubSpot [14] es una plataforma de gestión de marketing, ventas y servicios que le dan soporte a las empresas para atraer a visitantes, convertir leads y cerrar clientes. Las herramientas que abarca esta aplicación son la gestión de contenido, automatización de marketing, análisis de datos y CRM (Customer Relationship Management). Con esta plataforma, las empresas pueden crear y publicar contenido atractivo, automatizar sus procesos de marketing, analizar el rendimiento de sus campañas y gestionar sus relaciones con los clientes.

A continuación listamos las **ventajas** de esta aplicación:

- Suite de herramientas de marketing y ventas que abarcan todas y cada una de las etapas del embudo de ventas.
- Brinda una integración de CRM (Customer Relationship Management) que facilita la gestión eficiente de las relaciones con los clientes y realizar un seguimiento de las interacciones.
- Contiene una personalización avanzada que da la oportunidad a las empresas de adaptar sus estrategias de marketing.
- Una gran biblioteca con recursos para formación, que incluye blogs, libros electrónicos y seminarios web, facilita el aprendizaje de los usuarios y mejoran sus habilidades de marketing y ventas.
- Gran capacidad de automatización de marketing que facilita a las empresas ahorrar tiempo y recursos para automatizar tareas diarias y personalizar la experiencia del cliente.

- Ofrece un soporte técnico dedicado y recursos de aprendizaje, que ayuda a los usuarios a aprovechar al máximo todas las características y funciones de la plataforma.
- Se integra con otras herramientas populares de marketing y ventas, con Google Analytics, que amplían las capacidades de la plataforma y mejoran la productividad del equipo.

Después de nombrar las ventajas más relevantes, se presentan los **inconvenientes** que se han analizado:

- Los precios son mucho más elevados si se comparan con otras plataformas de marketing y ventas, lo que puede ser un impedimento para pequeñas empresas o usuarios independientes.
- Tiene una curva de aprendizaje elevada para aprovechar todas las funcionalidades, especialmente en usuarios nuevos.
- Contiene limitaciones en el número de contactos y usuarios dependiendo del plan de suscripción, lo que puede ser otro impedimento para organizaciones en crecimiento.
- Diversas funciones avanzadas requieren de un alto conocimiento técnico o experiencia en marketing y ventas para su implementación y aprovechamiento.
- Al igual que la plataforma anterior, su integración con herramientas de terceros es limitada y requiere de un proceso de instalación de complementos adicionales, lo que puede aumentar costos y dificultades.
- La plataforma en ocasiones experimenta periodos de inactividad o lentitud durante momentos de alto tráfico, esto puede afectar la productividad del equipo y la capacidad de realizar tareas de una manera eficiente.
- Esta plataforma puede ser demasiado compleja para las necesidades de algunas empresas, sobre todo en aquellas que tienen pequeños equipos o recursos limitados.

Asana.

Asana [15] es una plataforma de gestión de proyectos y tareas que facilita a equipos de empresas organizarse, gestionar y realizar el seguimiento de sus proyectos. Esto permite a los usuarios crear tareas, establecer fechas de vencimiento de las mismas, asignarlas a los diferentes integrantes del equipo y permitir la colaboración en los proyectos de una forma idónea. Además, los equipos pueden organizar sus proyectos en listas, tableros y calendarios para seguir el progreso de sus tareas en tiempo real.

A continuación listamos las **ventajas** de esta aplicación:

- Está dotada de una interfaz intuitiva y fácil de usar que facilita la colaboración entre los miembros del equipo y la gestión de proyectos.
- Tiene una flexibilidad para la gestión de proyectos de cualquier tamaño, abarcando desde pequeñas tareas hasta proyectos con gran complejidad con varios equipos y necesidades.
- Es capaz de integrarse con otras herramientas de productividad populares, como Google Drive, Slack y Microsoft Teams, mejorando de tal forma la eficiencia y colaboración de los equipos.
- Amplio abanico de vistas y herramientas que se utilizan para la organización del flujo de trabajo y adaptarse a sus necesidades específicas.
- Incorpora funciones avanzadas de seguimiento y asignación de tareas que facilitan a los equipos la realización del seguimiento de sus proyectos y asignación de recursos de manera idónea.
- Tiene la capacidad de generar informes y hacer análisis, proporcionando una valiosa información sobre el rendimiento de los equipos y el avance de los proyectos, permitiendo a los usuarios establecer áreas de mejora y la toma de decisiones informadas.
- También incluye un soporte técnico receptivo e instrumentos de aprendizaje, que facilitan a los usuarios a aprovechar todas las características y funcionalidades de la plataforma.

Después de nombrar las ventajas más relevantes, se presentan los **inconvenientes** que se han analizado:

- Existen limitaciones en el plan gratuito, como la cantidad de usuarios y el número de proyectos que se pueden crear, pudiendo llegar a ser restrictivo en equipos más grandes o proyectos más complejos.
- El aprendizaje de algunas funcionalidades avanzadas puede llegar a ser complicado para nuevos usuarios.
- Si se requiere integrar con otras herramientas de terceros puede requerir la instalación de complementos adicionales, lo que puede aumentar los costos y la complejidad.
- La personalización de las vistas y los informes podría ser limitada en comparación con otras herramientas para gestiona. Esto puede dificultar que se adapte a las necesidades del proyecto o equipo.

- La colaboración en tiempo real puede ser limitada en alguna situación si se compara con otras herramientas del mismo ámbito.Esto puede puede llegar a limitar su la productividad y eficiencia del equipo o proyecto.
- Algunas funcionalidades requieren una actualización de la suscripción de pago, por lo que aumentaría los costes a organizaciones que deseen tener todas las características de la plataforma.
- Asana experimenta dificultades en la gestión con un gran volumen de datos o proyectos con gran número de tareas. Este hecho puede ralentizar el rendimiento y la capacidad de respuesta de la plataforma en situaciones de carga pesada.

Zendesk.

Zendesk [16] es una plataforma de gestión que abarca las relaciones que las organizaciones tienen con sus clientes (CRM) y se centra en la atención al cliente y soporte técnico. Ofrece un gran abanico de herramientas y funcionalidades para ayudar a las empresas en la gestión efectiva sobre sus interacciones con los clientes, brindando la creación de tickets de soporte, automatización de procesos y la generación de informes para verificar el rendimiento. Esta plataforma se destaca por su facilidad de uso, escalabilidad y la capacidad de adaptación según las necesidades de las organizaciones.

A continuación listamos las **ventajas** de esta aplicación:

- Incorpora una interfaz intuitiva y fácil de manejar que permite la gestión de tickets de soporte y la colaboración entre los equipos.
- Gran variedad de canales de comunicación integrados, incluyendo correo electrónico, chat en vivo, teléfono y redes sociales, que brindan a las empresas la gestión de todas las interacciones con los clientes desde una única plataforma.
- Permite automatizar tareas repetitivas y crear flujos de trabajo personalizados, mejorando la eficiencia del equipo y la experiencia del cliente.
- Es capaz de integrarse con un gran abanico de herramientas y servicios de terceros, como CRM,ERP y sistemas de gestión de proyectos, que aumentan las capacidades de la plataforma y mejoran la colaboración interdepartamental.
- Alberga funciones avanzadas para la realización de informes y análisis que nutren a las empresas de información muy valiosa que trata sobre el rendimiento de los equipos de soporte y la satisfacción del cliente con estos, permitiendo realizar una toma de decisiones justificada.
- Cuenta con una gran escalabilidad, lo que permite a las organizaciones de todos los tamaños (Startup, grandes empresas, etc.) la utilización de la plataforma eficazmente y la posibilidad de adaptación según sean sus necesidades en el tiempo.

- Tiene un soporte técnico receptivo y recursos de aprendizaje, creados para ayudar a las empresas en la implementación de la plataforma de una manera eficiente y resolver cualquier impedimento que pueda surgir.

Después de nombrar las ventajas más relevantes, se presentan los **inconvenientes** que se han analizado:

- Requiere de costos adicionales para las funcionalidades avanzadas. Algunas características avanzadas solo estarán disponibles en planes de precios superiores, lo que puede resultar en gastos adicionales en las empresas que la necesiten.
- Tiene limitaciones en la personalización de la interfaz de usuario. Aunque tiene cierto grado, las opciones pueden llegar a ser limitadas si hablamos en términos de diseño y disposición de la interfaz. Esto puede ser un impedimento en las necesidades estéticas o de facilidad de uso en las organizaciones.
- Existen posibles problemas de privacidad y seguridad de los datos. Dado que se almacenan y gestionan datos sensibles del cliente, existe el riesgo potencial de que surjan violaciones de seguridad o brechas en la privacidad. Esto generaría repercusiones negativas en la confianza con el cliente y en la reputación de la empresa.
- Para las organizaciones que dependen de sistemas heredados o personalizados, podría ser un desafío la integración con la plataforma Zendesk. Lo que origina un desarrollo adicional o buscar soluciones de terceros para garantizar una compatibilidad.
- Como Zendesk tiene una dependencia de infraestructura en la nube para la prestación de servicios. Esto conlleva posibles interrupciones del servicio por problemas de conectividad, fallos del proveedor de servicios en la nube u otros imprevistos y esto afectaría a la disponibilidad del sistema y en la experiencia de usuario.
- Mientras se realizan actualizaciones de software o mantenimiento del sistema, se pueden experimentar interrupciones temporales en el servicio u originar problemas de rendimiento, lo que afecta la productividad y satisfacción del cliente.
- Pese a que ofrece soporte para varios idiomas, podrían existir limitaciones en términos de disponibilidad de idiomas específicos o en capacidad de traducción. Este hecho puede dificultar la atención al cliente en organizaciones internacionales.

3.2. Framework para desarrollo de aplicaciones web.

En esta sección se analizan diversos frameworks que se utilizan para desarrollar aplicaciones web. Se examinan las características, ventajas e inconvenientes respecto al uti-

lizado en este proyecto.

3.2.1. React.js

React.js [17] está desarrollado pro Facebook, se trata de un framework de JavaScript de código abierto con una gran popularidad en el mundo del desarrollo web en los últimos años. Se enfoca en la construcción de interfaces de usuario dinámicas e interactivas, evolucionando la forma en la que se desarrollan las aplicaciones web de hoy en día. Esto lo convierte en una herramienta muy potente para los desarrolladores con cualquier nivel de experiencia.

A continuación describiremos las principales características de este framework:

- Una de las características más importantes es el enfoque a la hora de crear componentes reutilizables. Las interfaces de usuario se convierten en pequeños componentes con independencia donde cada uno maneja una parte específica de la interfaz de usuario. Estos componentes son fácilmente reutilizados en cada una de las partes de la aplicación, facilitando la creación y manteniendo un código limpio y modular.

- React.js usa un paradigma de programación declarativa, esto significa que los desarrolladores tienen la oportunidad de describir cómo debe formarse la interfaz de usuario en un determinado momento. El framework es quien se encarga de la actualización automática de la interfaz de usuario cuando cambia el estado de la aplicación. Este hecho simplifica la gestión del estado de la aplicación y mejora enormemente la eficiencia del desarrollo de la misma.

- Utiliza un motor llamado **virtual DOM** de renderizado virtual que compara el estado actual de la interfaz con el anterior, actualizando solo los componentes que se han modificado. Por lo tanto, el número de operaciones que realiza el motor se minimiza y, por lo tanto, mejora en gran medida el rendimiento de la aplicación, sobre todo en aplicaciones dinámicas y complejas.

- Una consecuencia de utilizar el motor anteriormente mencionado y el enfoque en componentes reutilizables, se obtiene un gran rendimiento y eficiencia en las aplicaciones web. Esto se traduce en la experiencia de usuario, siendo más receptiva y fluida.

- Este framework tiene una gran comunidad de personas activas y además cuenta con una amplia biblioteca y herramientas complementarias que ayudan al desarrollo de aplicaciones web. La documentación es extensa y está bien organizada, lo que permite un aprendizaje rápido para dominar el framework.

3.2.2. Vue.js

Vue.js [18] es un framework de JavaScript de código abierto que nació en 2014 creado por Evan You. Siendo una opción muy popular y ampliamente adoptada en el marco de desarrollo de aplicaciones web modernas. Una de las características a destacar es su enfoque en la simplicidad, rendimiento, y flexibilidad , por lo que se convierte en una herramienta potente y versátil para cualquier desarrollador. El crecimiento fue impulsado por una comunidad activa de desarrolladores adquiriendo una documentación muy extensa con una curva de aprendizaje muy suave, por ello se ha ganado la confianza de muchas empresas líderes en la industria , lo que respalda su confiabilidad y robustez en la implementación de aplicaciones web a gran escala.

A continuación describiremos las principales características de este framework:

- Como sucede en React.js, este framework permite la división de la interfaz de usuario en componentes que se pueden reutilizar. Estos contienen las funciones y apariencia de partes concretas de la interfaz y por ello pueden ser fácilmente reutilizados en diferentes partes de la aplicación. Esto promueve la modularidad y reutilización de código, lo que simplifica la creación y mantenimiento de aplicaciones web.

- Utiliza un sistema de reactividad que facilita vincular datos a la interfaz de usuario fácil y eficientemente. Esto se traduce en que cualquier cambio respecto a los datos se refleja de una manera automática en la interfaz, sin tener la necesidad de realizar actualizaciones manuales. Esto se traduce en una gestión sencilla del estado de la aplicación y mejora la eficiencia en la implementación.

  Vue.js también ofrece una variedad de directivas y complementos integrados que ayudan en la creación de interfaces dinámicas e interactivas. Dichas directivas brindan la realización de tareas comunes, como manipular el DOM, gestionar eventos y enlazar datos de una manera sencilla y eficiente.

- Además de destacar por su simplicidad y facilidad de uso, también es bastante flexible y escalable. Es adecuado para una gran variedad de proyectos, desde pequeñas aplicaciones hasta aplicaciones a gran escala empresariales. Integrándose a su vez con otras tecnologías y bibliotecas, lo que añade versatilidad a sus características a destacar.

- Este framework se centra en la simplicidad y facilidad, con una sintaxis concisa y clara, permite a los desarrolladores la creación de interfaces de usuario complejas con un mínimo de esfuerzo y código. Por ello es una de las opciones preferidas para cualquier nivel de experiencia.

3.2.3. Angular.

Angular [10] es un framework de desarrollo de aplicaciones web de código abierto, que está mantenido por Google y que cuenta con una comunidad activa de desarrolladores. Surgió en 20210 con el nombre de AngularJS y fue evolucionando significativamente hasta convertirse en Angular, un framework moderno y potente para crear aplicaciones web dinámicas y escalables. Es conocida por su sólida arquitectura basada en componentes, que ayuda a la creación de aplicaciones web complejas a través de la composición de elementos reutilizables. Del mismo modo, su gama de características, como la internacionalización, optimización del rendimiento y seguridad integrada, se convierte en una opción muy llamativa para proyectos de cualquier tamaño. Cuenta con una curva de aprendizaje gradual y una documentación muy completa, por ello se convierte en una herramienta versátil y eficaz en el desarrollo de aplicaciones web modernas.

A continuación describiremos las principales características de este framework:

- Este framework sigue un enfoque de desarrollo basado en componentes, donde cada uno de los elementos que componen la interfaz se representa como un componente reutilizable. Esto ayuda la creación de interfaces modular y mantenibles, mejorando la legibilidad y escalabilidad del código.

- Ofrece una vinculación de datos bidireccional, lo que se traduce en que si se realiza un cambio en el modelo de la aplicación, este se refleja automáticamente en la vista y viceversa. Lo que conlleva que se simplifique el manejo de los datos en la aplicación y mejora la experiencia de usuario al ofrecer interfaces de usuario dinámicas y receptivas

- Angular provee una infraestructura solida que incluye características como inyección de dependencias, validación de formularios, enrutamiento y manejo de estado. Esto se traduce en que estas características aportan un desarrollo de aplicaciones completas y estructuradas, facilitando a los desarrolladores un mantenimiento de código limpio y organizado.

- Este framework ha sido escrito en TypeScript, un superconjunto de JavaScript que ofrece tipos estáticos opcionales y otras características avanzadas al lenguaje. Esto aporta a los desarrolladores detectar y corregir los errores en tiempo de compilación, que aumenta la calidad y robustez del código.

- Cuenta con un ecosistema amplio y activo que contiene una variedad de herramientas, bibliotecas y complementos desarrollados por la comunidad y que han sido verificados por Google. Esto permite la integración de Angular con otras tecnologías al igual que el aumento de sus capacidades según las necesidades del proyecto.

3.2.4. Ember.js

Ember.js [19] es un framework de desarrollo de aplicaciones web de código abierto enfocado en la productividad y escalabilidad. Surgió en 2011, ganando popularidad en la comunidad gracias a su enfoque en la convención sobre la configuración y la arquitectura de aplicaciones basada en convenciones. Proporcionando un abanico de herramientas y patrones predefinidos que brindan a los desarrolladores la creación de aplicaciones de forma ágil y eficiente con un mínimo de configuración.

A continuación se describen las principales características de este framework:

- Este framework promueve una estructura de aplicación coherente y predecible al seguir convenciones predefinidas para la organización del código y la gestión del flujo de datos.
- Además, ofrece la creación de aplicaciones de una sola página (SPA) al proporcionar un enrutador integrado que gestiona la navegación del usuario y la actualización de la interfaz en función de la URL.
- Utiliza un motor de plantillas Handlebars, permitiendo a los desarrolladores la creación de interfaces dinámicas y reactivas mediante la vinculación de datos y la generación de contenido HTML de forma declarativa.
- También incluye el **modulo Ember** que facilita la integración con servicios de backend al proporcionar una capa de abstracción para trabajar con modelos de datos y realizar operaciones CRUD de manera eficiente.
- Ember CLI es una línea de comandos que simplifica el proceso de desarrollo y brinda una estructura de proyecto predefinida, generadores de código y tareas automatizadas para realizar construcciones, pruebas y despliegues de la aplicación.

3.3. Framework de servicios web (RESTful).

Se exploran diferentes frameworks de servicios web (RESTful) más utilizados en el desarrollo de APIs. Se describen sus características y funcionalidades, así como su idoneidad para el desarrollo de la API requerida en el proyecto, teniendo en cuenta características como la facilidad de uso y escalabilidad.

3.3.1. Express.js

Express.js [20] se originó en 2010 como un proyecto paralelo de TJ Holowaychuk, un destacado desarrollador de Node.js. Se trata de un framework web minimalista y flexible para Node.js que se suele utilizar de manera común en la construcción de aplicaciones

web y servicios RESTful. Es famoso por su simplicidad y proporciona una capa delgada sobre Node.js, lo que habilita a los desarrolladores a construir servicios RESTful.

A continuación se describen las principales características de este framework:

- Permite definir rutas HTTP para manejar solicitudes de clientes y enviar correspondientes. Esto ayuda a la creación de una API RESTful con endpoints bien definidos.
- Utiliza un sistema de middleware que posibilita la ejecución de funciones en el ciclo de vida de una solicitud. Esto ayuda al desarrollo de funcionalidades como la autenticación, autorización, el registro de solicitudes y la manipulación de errores.
- Permite utilizar métodos y herramientas para acceder y manipular datos de solicitudes y respuestas HTTP. Esto ofrece la validación de datos de entrada, gestión de cookies y sesiones, y la generación de respuestas adecuadas.
- Pese a que su enfoque principal es el desarrollo de APIs RESTful, también admite la integración de motores de plantillas para la generación de contenido dinámico en aplicaciones web tradicionales.
- Cuenta con una gran escalabilidad para realizar aplicaciones, desde proyectos pequeños hasta sistemas empresariales complejos. Gracias a su arquitectura modular y a su amplia comunidad de desarrolladores, lo convierten en una opción interesante para muchos proyectos.
- Además, tiene una amplia variedad de middleware de terceros disponibles a través del ecosistema de Node.js. Lo que ofrece una fácil integración de funcionalidades adicionales, como la comprensión de datos, almacenamiento en caché y seguridad.

3.3.2. Django REST framework.

Django [21] REST framework es una poderosa y flexible herramienta para construir APIs web en el lenguaje de Python, se basa en el popular framework Django. Nació en 2011 como una solución para ayudar al desarrollo de servicios web RESTful en aplicaciones Django, brindando una estructura sólida y herramientas útiles para gestionar solicitudes HTTP, serialización de datos, autenticación, autorización y otras funcionalidades.

A continuación se describen las principales características de este framework:

- Tiene la capacidad de serialización de datos, ofreciendo la conversión de objetos Python en formatos de datos comunes como JSON o XML, viceversa.
- Permite la creación de vistas basadas en clases con funcionalidades comunes para la creación, lectura, actualización y eliminación de recursos, lo que facilita el desarrollo y reduce la duplicación de código.

- Cuanta con un sistema flexible para autenticar usuarios y autorizar accesos a recursos basados en roles y permisos, con soporte para esquemas de autenticación como token, OAuth, JWT, entre otros.

  Incluye herramientas integradas para la creación de documentos interactivos de APIs y navegación fácil de usar, lo que permite la comprensión y el consumo de la API por parte de los desarrolladores.

- Es muy extensible y proporciona la capacidad de personalizar casi todos los aspectos de la API, desde el comportamiento de las vistas hasta el estilo de la documentación, lo que lo hace adecuado para una amplia gama de proyectos y requisitos.

- Brinda la oportunidad de realización de pruebas unitarias y de integración para asegurar el funcionamiento correcto de la API, permitiendo utilizar herramientas integradas para realizar pruebas automatizadas y verificar el comportamiento de los endpoints.

- Además, tiene una comunidad activa de desarrolladores y una documentación completa, esto garantiza un soporte continuo y una rápida resolución de problemas.

3.3.3. Flask.

Flask [22] es un framework ligero y flexible para el lenguaje de Python que se utiliza ampliamente para desarrollar servicios RESTful y aplicaciones web simples y rápidas. Surgió en 2010 con la intención de ser una alternativa minimalista a los frameworks web más complejos, brindando una sintaxis sencilla y fácil de aprender, la cual permite a los desarrolladores crear APIs web rápidas y escalables.

A continuación se describen las principales características de este framework:

- Permite definir rutas y endpoints de manera clara y concisa, lo que agiliza la creación de APIs RESTful. Se pueden especificar las operaciones CRUD (Crear, leer, Actualizar, Eliminar) para cada una de los recursos de una forma intuitiva.

- Se integra con herramientas y bibliotecas externas de Python, aumentando su funcionalidad y habilitando a los desarrolladores aprovechar características adicionales, como la validación de datos, almacenamiento en bases de datos, entre otras.

- tiene un soporte integrado para la serialización y deserialización de datos en formatos comunes JSON, XML y YAML, lo que agiliza la comunicación entre el cliente y el servidor a través del intercambio de datos estructurados.

- Proporciona una API sencilla y consistente para la gestión de solicitudes HTTP, donde se incluye la gestión de parámetros de consulta, encabezados HTTP y métodos de solicitud. Esto permite a los desarrolladores crear APIs web robustas y seguras.

- No obliga a tener una estructura rígida del proyecto, por ello ofrece la oportunidad a los desarrolladores organizar su código de la manera que consideren más adecuada. Esto agiliza la adaptación a los requisitos específicos del proyecto y promueve la modularidad y reutilización del código.

- Adopta un ecosistema de complementos que ofrecen extender su funcionalidad de manera modular para que los desarrolladores puedan elegir entre una amplia gama de complementos para agregar características adicionales, como la autenticación basada en tokens, la paginación de resultados y la generación de documentación.

- Cuemta con una comunidad de desarrolladores activa y una amplia documentación que agiliza el aprendizaje y la resolución de errores. También existen numerosos recursos en línea que ayudan a los desarrolladores a aprovechar el potencial del framework.

3.3.4. RESTLET.

RESTLET [8] es un framework de desarrollo de servicios web RESTful que proporciona una arquitectura simple y poderosa para la creación de APIs web escalables y eficientes. Nació en 2005 como una de las primeras implementaciones de la arquitectura RES en Java y desde aquel entonces ha sido escogido por organizaciones y desarrolladores de todo el mundo.

A continuación se describen las principales características de este framework:

- Enfocado en los principios de la arquitectura REST, que incluyen la utilización de recursos, identificación uniforme de recursos (URI), operaciones HTTP estándar (GET, POST, PUT, DELETE) y el uso de representaciones de recursos como JSON o XML para la transferencia de datos.

- Proporciona una abstracción de servidor y cliente que facilita el desarrollo de servicios web RESTful en Java. Se pueden crear fácilmente tanto servidores como cliente RESTful con la misma API, agilizando la implementación de aplicaciones distribuidas y la interoperabilidad entre sistemas.

- Cuenta con soporte para anotaciones y enrutamiento, permitiendo definir de manera declarativa las rutas y endpoints de la API. Esto hace más sencilla la configuración y el mantenimiento de la API, promoviendo una arquitectura modular y sencilla de entender.

- Brinda una API intuitiva para el manejo de solicitudes y respuestas HTTP, donde se incluye la gestión de encabezados, parámetros de consulta, contenido de la solicitud y códigos de estado. Esto facilita la creación de servicios web robustos y seguros que cumplan con los estándares de la web.

- Se integra fácilmente con otras tecnologías Java, como JAX-RS, Servlets y JPA, lo que amplía su funcionalidad y permite que se aproveche al máximo el ecosistema de Java. Esta agiliza la creación de aplicaciones web completas y escalables en Java.
- Ofrece un mecanismo de complementos que permite extender su funcionalidad de manera modular. Se pueden seleccionar entre una variedad de complementos disponibles para agregar características adicionales con autenticación, autorización, compresión de datos y cacheado.
- Cuenta con una amplia documentación y una comunidad con soporte y comparten recursos útiles como ejemplo de códigos, tutoriales y foros de discusión. Esto ayuda en el aprendizaje.

3.4. Base de datos para aplicaciones web.

En esta sección se analizan diferentes tipos de base de datos que se utilizan para el desarrollo de aplicaciones web.

3.4.1. Base de datos relacionales(SQL).

Las bases de datos relacionales, también llamadas como bases de datos SQL (Structured query language) [23], son sistemas de gestión de bases de datos (DBMS) que están basadas en el modelo relacional. En dicho modelo, los datos están organizados en tablas relacionadas entre sí por claves primarias y claves foráneas. En cada tabla se representa una instancia de esa identidad. Las bases de datos relacionales se nutren del lenguaje de consulta estructurada (SQL) donde realizan operaciones como consultas, inserciones, actualización y eliminación de datos.

A continuación se muestran las características principales [24] de las bases de datos relacionales (SQL):

- Utilizan un modelo de datos tabular.
- Requieren un esquema fijo definido antes de insertar datos.
- Aseguran la integridad de los datos a través de restricciones y relaciones.
- Soportan transacciones ACID (Atomicidad, Continencia, Aislamiento y Durabilidad).
- Son efectivas para las consultas complejas que implican relaciones entre tablas.
- Son idóneas para aplicaciones que requieren consistencia y precisión en los datos.

3.4.2. Bases de datos no relaciones(NoSQL):

Estas bases de datos NoSQL (Not Only SQL) [25], tratan de sistemas de gestión de bases de datos diseñados para administrar grandes volúmenes de datos no estructurados. Por lo que se diferencian de las bases de datos relacionales, estas no constan de un esquema fijo y tienen la capacidad de almacenar datos en diversos formatos, como documentos, pares de clave-valor o gráficos. Las bases de datos NoSQL son altamente escalables y flexibles, lo que las hace idóneas para aplicaciones web que necesitan gestionar una gran cantidad de datos distribuidos y que deben de tener alta disponibilidad y rendimiento.

A continuación se muestran las características principales [26] de las bases de datos no relacionales (NoSQL):

- Utilizan un modelo de datos no tabular y no precisan de un esquema fijo.
- Son escalables y pueden administrar cargas de trabajo distribuidas de manera ágil.
- Son flexibles y se pueden adaptar a los cambios en la estructura de datos.
- Son idóneas para aplicaciones que necesitan alta disponibilidad y rendimiento.
- No son compatibles con las operaciones ACID en todos los casos, lo que puede llevar a una eventual consistencia en lugar de una consistencia inmediata.

Una ves Se han definido los dos tipos de bases de datos más utilizadas en aplicaciones web, se muestran varios ejemplo de ambas.

3.4.3. MySQL.

MySQL [27] es un sistema de gestión de bases de datos relacional de código abierto muy utilizado en el desarrollo de aplicaciones web y empresariales. Fue desarrollado por Oracle Corporation, MySQL brinda una combinación de rendimiento, confiabilidad, y facilidad de uso que lo convierte en una opción popular para el almacenamiento y gestión de datos en un amplio abanico de entornos.

A continuación se describen las principales características de esta base de datos:

- Es muy escalable, lo que se traduce en que puede manejar grandes volúmenes de datos y crecer con los requisitos de la aplicación.
- Cuenta con un rendimiento óptimo tanto para operaciones de lectura como de escritura, lo que asegura tiempos de respuesta bastante rápidos para las consultas de la base de datos.

- Es compatible con un gran tipo de plataformas, entre las que se incluyen Windows, Linux y macOS. Siendo versátil y fácil de integrar en diferentes entornos de desarrollo.
- Tiene varias funciones de seguridad, como la autenticación de usuarios, autorización y encriptación de datos, lo que asegura la protección de información confidencial.
- Es compatible con transacciones ACID (Atomicidad, Continencia, Aislamiento y Durabilidad), lo que asegura la integridad de los datos y la consistencia de entornos multiusuarios.
- Brinda funciones avanzadas para el almacenamiento y manipulación de datos geoespaciales, por ello lo convierte en una opción adecuada para aplicaciones que requieren el procesamiento de información basada en ubicación.

3.4.4. Microsoft SQL Server.

Microsoft SQL Server [28] se trata de un sistema de gestión de bases de datos relacional desarrollado por Microsoft. Siendo una solución integral que garantiza un entorno fiable y seguro para el almacenamiento y gestión de datos críticos en entornos empresariales y de aplicaciones web. Proporciona una amplia gama de características y herramientas que facilitan el desarrollo, implementación y administración de base de datos.

A continuación se describen las principales características de esta base de datos:

- Es muy escalable y puede gestionar un gran volumen de datos y cargas de trabajo. Proporciona opciones de escalabilidad vertical y horizontal para adaptarse a las necesidades cambiantes de las aplicaciones.
- Ofrece un rendimiento potente e idóneo para consultas y operaciones de bases de datos, debido a su motor de base de datos optimizado y a técnicas de optimización avanzadas.
- Se integra con otras tecnologías y herramientas de Microsoft, como el sistema operativo de Windows, el entorno de desarrollo Visual Studio y la plataforma de nube de Azure, lo que permite su implementación y gestión en entornos de Microsoft.
- Abarca un conjunto de herramientas de administración y monitorización que ofrecen a los administradores la gestión y optimización del rendimiento de las bases de datos de forma eficaz.
- Tiene capacidades de replicación y disponibilidad avanzadas, lo que asegura una continua disponibilidad de los datos y minimiza el tiempo de inactividad en caso de fallos.

3.4.5. MongoDB.

MongoDB [9] es una base de datos NoSQL de código abierto, desarrollada por MongoDB Inc., que se ha postulado como una opción atractiva para el almacenamiento de datos en aplicaciones web y móviles. Fue creada para manejar un gran volumen de datos no estructurados, basándose en un modelo de datos flexible y escalable que posibilita almacenar, recuperar y manipular datos de una manera eficiente.

A continuación se describen las principales características de esta base de datos:

- Emplea un modelo de datos flexible basado en documentos JSON (BSON), lo que posibilita el almacenamiento de datos estructurados, semiestructurados o no estructurados. Esto permite la representación de datos complejos y la adaptación a los cambios en el esquema de datos con el tiempo.
- Cuenta con un diseño de escalado horizontal cuando se agregan más nodos al clúster, lo que posibilita gestionar grandes volúmenes de datos y cargas de trabajo crecientes. Esto se consigue a través de la fragmentación de datos y la distribución de los mismos entre múltiples servidores.
- Brinda capacidades integradas de replicación y una tolerancia a fallos que asegura la disponibilidad de los datos incluso en caso de fallos de hardware o red. Emplea un modelo de réplica configurable para mantener copias redundantes de los datos en varios nodos.
- Permite índices secundarios y consultas complejas que admiten realizar operaciones de lectura y escritura de manera rápida y eficiente. Esto ayuda a la búsqueda y recuperación de datos en grandes colecciones.
- Abarca diferentes niveles de consistencia para adaptarse a las necesidades de rendimiento y disponibilidad de la aplicación. Los desarrolladores pueden elegir entre consistencia eventual y consistencia fuerte según los requisitos.
- Permite la realización de consultas ad hoc sobre los datos sin la necesidad de definir un esquema previamente. Esto ayuda a la exploración y análisis de datos de forma dinámica y flexible.

3.4.6. Cassandra.

Cassandra [29] es una base de datos NoSQL distribuida, escalable y de código abierto que fue desarrollada por Facebook y posteriormente donada a Apache Software Foundation. Se diseñó para manejar grandes volúmenes de datos distribuidos en múltiples servidores, esto la convierte en una de las opciones más llamativas para aplicaciones web.

A continuación se describen las principales características de esta base de datos:

- Está diseñada para crecer de manera lineal cuando se agreguen más nodos al clúster, lo que posibilita la gestión de grandes volúmenes de datos y cargas de trabajo crecientes.
- Utiliza un modelo de arquitectura descentralizada que erradica los puntos únicos de falla y asegura que los datos estén disponibles permanentemente, incluso si el hardware o la red falla.
- Ofrece un modelo de datos flexible que admite almacenar datos estructurados, semiestructurados o no estructurados, lo que la convierte en la base de datos ideal para una gran gama de casos de uso.
- Permite a los desarrolladores la elección entre diferentes niveles de consistencia, desde la coherencia fuerte hasta la eventual, adaptándose a los requisitos de rendimiento y disponibilidad de la aplicación.
- Está optimizada para realizar operaciones de lectura y escritura de manera rápida y eficiente, lo que la hace adecuada para aplicaciones con grandes necesidades de rendimiento.
- Cuenta con capacidades integradas de replicación y particionamiento que posibilita distribuir los datos de manera eficiente entre múltiples nodos en el clúster.
- Se basa en un diseño de tolerancia a fallos, lo que se traduce en que puede continuar operando incluso en situaciones de fallos de nodo individuales o particiones de red.

Capítulo 4

Diseño de la aplicación.

En este capítulo nos centramos en el diseño de la aplicación o servicio web. Empezando con una exploración de la arquitectura general de la aplicación, definiendo los componentes principales y su relación. Una vez realizada dicha exploración se expone el diseño de la **interfaz gráfica**, donde se explica la navegación y el aspecto visual de la página. Posteriormente, el capítulo **Frontend** se centra en el desarrollo del mismo, donde se definen las clases y componentes que forman la interfaz de usuario. Se explora la comunicación de las mismas entre sí para que proporcionen un funcionamiento correcto y fluido al usuario. Se continúa con un capítulo donde se aborda el desarrollo del **backend**, en el cual se diseñan un prototipo de las clases del mismo. En el capítulo de **Servicio REST** donde se definen las diferentes URLs y métodos HTTP con su propósito y cómo se deberían utilizar para interactuar con la aplicación. Finalmente, en el capítulo **BBDD** se propone el diseño de la base de datos que servirá para almacenar y nutrir de información a la aplicación web.

4.1. Arquitectura general de la aplicación o servicio.

La arquitectura general de la aplicación sigue un diseño que se basa en componentes claramente definidos. Entre estos componentes se encuentran una base de datos (BBDD), una API y dos interfaces web: una destinada para usuarios y otra para administradores. Este diseño modular ayuda a la separación de preocupaciones y aumenta la sostenibilidad del sistema. Si seguimos los principios de diseño de software bien establecidos [30] [31]. La utilización de un modelo de componentes preserva que cada parte de la aplicación evolucione de manera independiente, aumentando la escalabilidad y la robustez del sistema.

La arquitectura propuesta de este sistema se compone de los siguientes elementos:

1. **Base de Datos (BBDD)**: Este elemento tiene como función almacenar y gestionar los datos utilizados por el sistema. Puede llegar a ser una base de datos

relacional (SQL) o no relacional (NoSQL), que se decidirá en los capítulos siguientes.

2. **API (Interfaz de Programación de Aplicaciones):** Esta herramienta proporciona un conjunto de endpoints o puntos de acceso que brindan a las aplicaciones externas la comunicación con el sistema. Es el encargado de manejar las solicitudes de clientes, procesar datos y enviar respuestas adecuadas. En este sistema, la API actúa como un elemento intermedio entre la base de datos y las aplicaciones web.
3. **Aplicaciones Web:**
   - **Web de Usuarios:** Esta web tiene como objetivo proporcionar una interfaz amigable y fácil de usar para que los usuarios interactúen con los datos y realicen las acciones específicas.
   - **Web de administración:** Esta web tiene la labor de proporcionar a los administradores o usuarios autorizados el acceso a las funcionalidades administrativas avanzadas. ¿Cómo pueden llegar a ser la gestión de usuarios o de las actividades?

A continuación, se muestra en la figura 4.1 la arquitectura general de la aplicación web que se desea implementar:

La interacción de los datos en el esquema de la figura 4.1 entre los componentes, se puede visualizar su flujo de datos de la siguiente manera:

1. La base de datos es la encargada del almacenamiento y gestión de los datos que se utilizan en la aplicación.
2. La API es la encargada de realizar las consultas y actualizaciones en la base de datos a través de un protocolo estándar, esto asegura que todas las operaciones sean eficientes y seguras.
3. Las webs serán las encargadas de enviar las solicitudes HTTP a la API y esta las procesará devolviendo respuestas en algún formato estándar, como podría ser JSON, BSON, XML, etc.

Esta estructura modular asegura que cada componente pueda ser implementado, probado y mantenido de manera independiente, lo que brinda escalabilidad y flexibilidad de la arquitectura. A continuación se definen brevemente los beneficios:

- **Modularidad:** La separación de la responsabilidad de las distintas funciones hace que el mantenimiento y actualización de cada una de los componentes no afecten a los demás.
- **Escalabilidad:** La arquitectura facilita escalar cada componente independientemente según las necesidades que surjan en la misma.

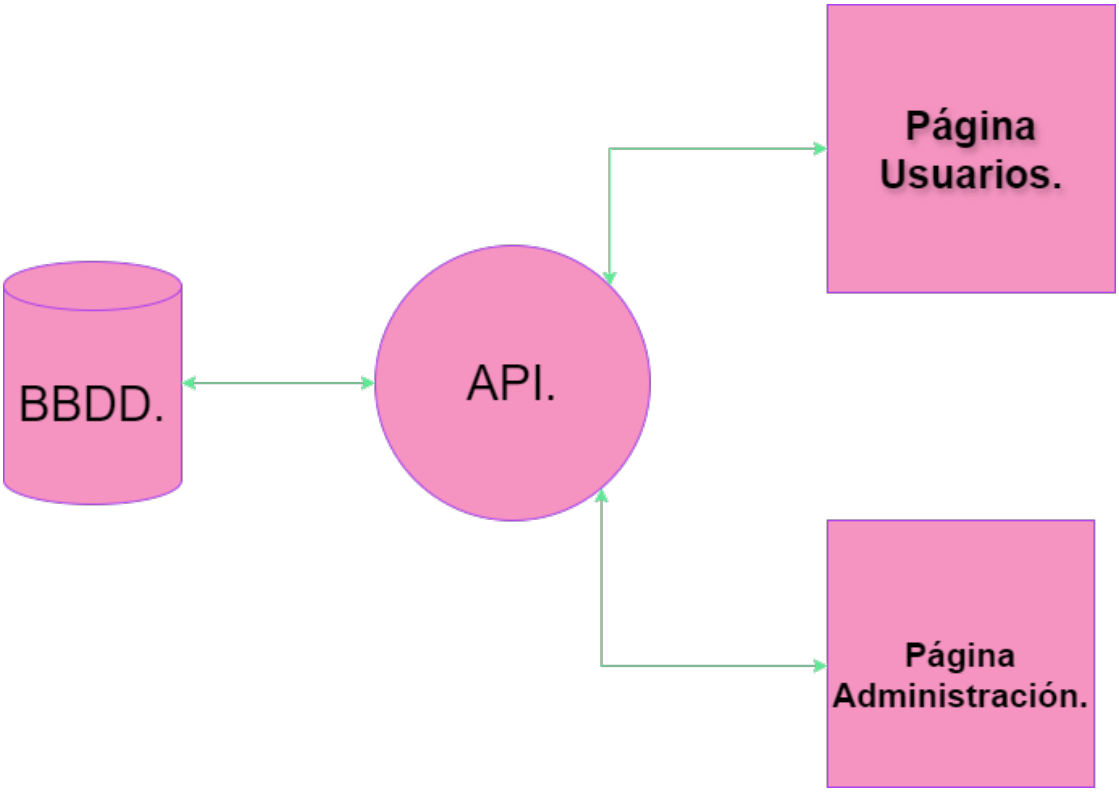


Figura 4.1: Arquitectura general del servicio.

- **Mantenibilidad:** La separación lógica del negocio, almacenamiento de datos y presentación en diferentes componentes simplifica la identificación y resolución de problemas.

4.2. Interfaz gráfica.

La interfaz gráfica de las páginas web se ha diseñado para brindar una experiencia de usuario fluida e intuitiva. Respecto a la página de usuarios, se ha creado un menú de navegación claro y accesible, lo que ayuda al acceso de las diferentes secciones de la web. Una interfaz gráfica es primordial para preservar una óptima usabilidad y accesibilidad [32], [33]. Realizando un diseño simple y coherente se asegura que los usuarios puedan interactuar con la aplicación de manera óptima. En esta sección, se explica cómo se ha creado el diseño previo de las diferentes interfaces, con las funcionalidades necesarias para la aplicación y su navegación entre las mismas.

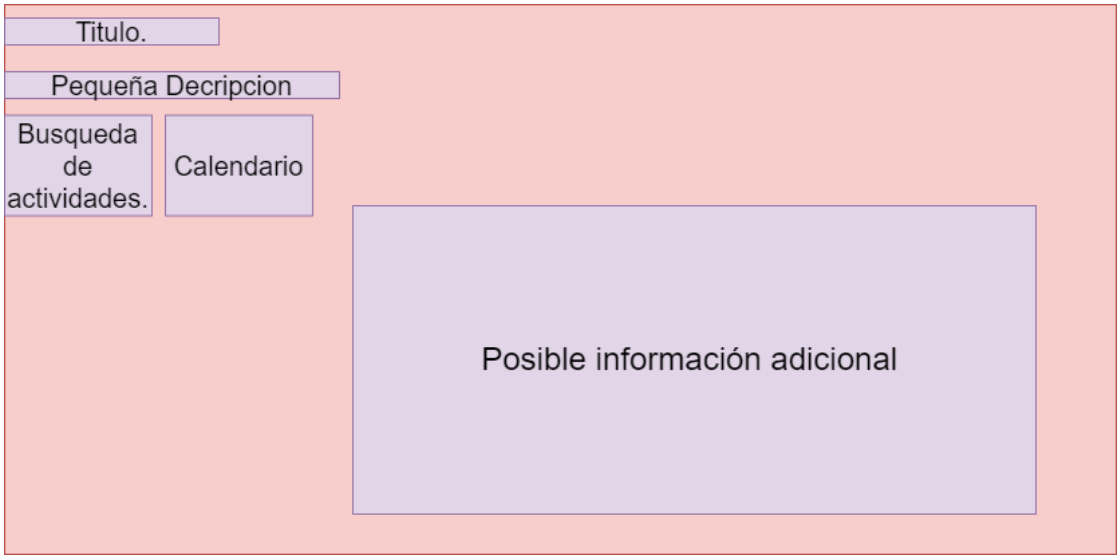


Figura 4.2: Diseño de la interfaz web de usuarios.

4.2.1. Interfaz de la web de usuarios.

La interfaz gráfica de la web de usuarios está diseñada para ser intuitiva y ágil en su navegación. Ofrece funciones clave y se pueden acceder a ellas desde la página principal (Home). A continuación, se van a describir la estructura y funcionalidades de dicha página a partir de la figura 4.2.

1. **Descripción General:**
    - *Título:* Se ubica en la parte superior de la página, el título proporciona una identificación clara de la sección o funcionalidad actual.
    - *Pequeña descripción:* Ubicado debajo del título, ofrece una breve descripción que ayuda a los usuarios a entender la función de la vista actual o proporciona información relevante.
    - *Búsqueda de actividades:* Un componente de la web que permitirá a los usuarios la búsqueda de actividades específicas. Esta función resulta esencial para la mejora de la experiencia de usuario, facilitando el acceso rápido a la información que necesitan.
    - *Calendario:* Un componente de calendario interactivo que permite a los usuarios ver las actividades programadas de una forma visual.
    - *Posible información adicional:* Un espacio dedicado para mostrar información adicional de la Escuela que sea relevante para los estudiantes (usuarios). Como ejemplo, podrían ser información relevante sobre actividades, noticias, actualizaciones o cualquier otra información que ayude al usuario.

2. **Navegación y funcionalidades:**

   - *Título y descripción:* El título debe ser visible para indicar la sección actual en la web. La pequeña descripción brinda contexto o instrucciones, ayudando a los usuarios a comprender la función de la página.
   - *Búsqueda de actividades:* Permite la búsqueda a través de diferentes criterios (nombre, fechas, categoría,ponente, etc.). Ayuda a mejorar la eficiencia a la hora de reducir el tiempo necesario para encontrar las actividades deseadas.
   - *Calendario:* Facilita una vista general de las actividades programadas, por ello los usuarios pueden navegar entre los diferentes meses para ver qué actividades existen.
   - *Posible información adicional:* En esta sección se podría actualizar dinámicamente para la ilustración de información relevante según la interacción del usuario.

4.2.2. Interfaz de la web de administración.

Esta interfaz se diseña con el objetivo de proporcionar a los administradores un acceso rápido y eficiente a las funcionalidades de gestión del sistema. El diseño del Home de esta web, que se presenta en la figura 4.3 se centra en la simplicidad y funcionalidad, lo que asegura que los administradores realicen sus tareas eficientemente.

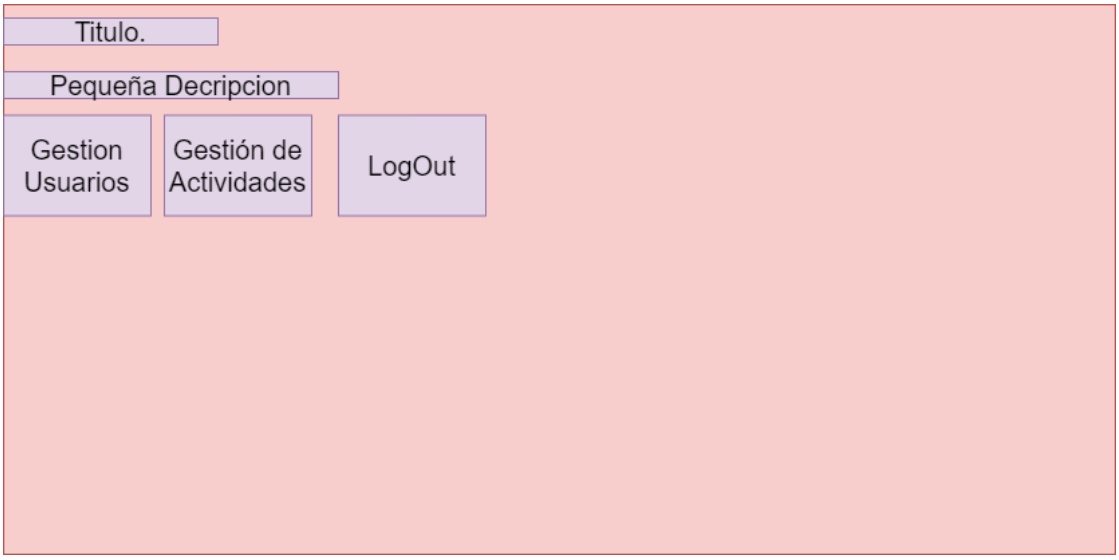


Figura 4.3: Diseño de la interfaz web de administración.

1. **Descripción General:**

- *Título:* Se ubica en la parte superior de la página, el título proporciona una identificación clara de la sección o funcionalidad actual.
- *Pequeña descripción:* Ubicado debajo del título, ofrece una breve descripción que ayuda a los usuarios a entender la función de la vista actual o proporciona información relevante.
- *Gestión de usuarios:* Un botón que redirige al componente donde están las funciones relacionadas con la gestión de los usuarios del sistema.
- *Gestión de actividades:* Un botón que redirige al componente donde se encuentran las funciones relacionadas con la gestión de actividades.
- *Logout:* Un botón que permite cerrar la sesión de los administradores de manera segura.

2. **Navegación y funcionalidades:**

- *Título y descripción:* El título debe ser visible para indicar la sección actual en la web. La pequeña descripción brinda contexto o instrucciones, ayudando a los usuarios a comprender la función de la página.
- *Gestión de usuarios:* Redirige a una página donde los administradores pueden crear y eliminar usuarios.
- *Gestión de actividades:* Redirige a una página donde los administradores pueden crear,editar y eliminar actividades. Brinda una vista detallada de las actividades programadas, con varias opciones de exportación.
- *Logout:* Permite cerrar la sesión de usuario de forma segura. Asegura que los usuarios autenticados tengan acceso a las funciones de administración.

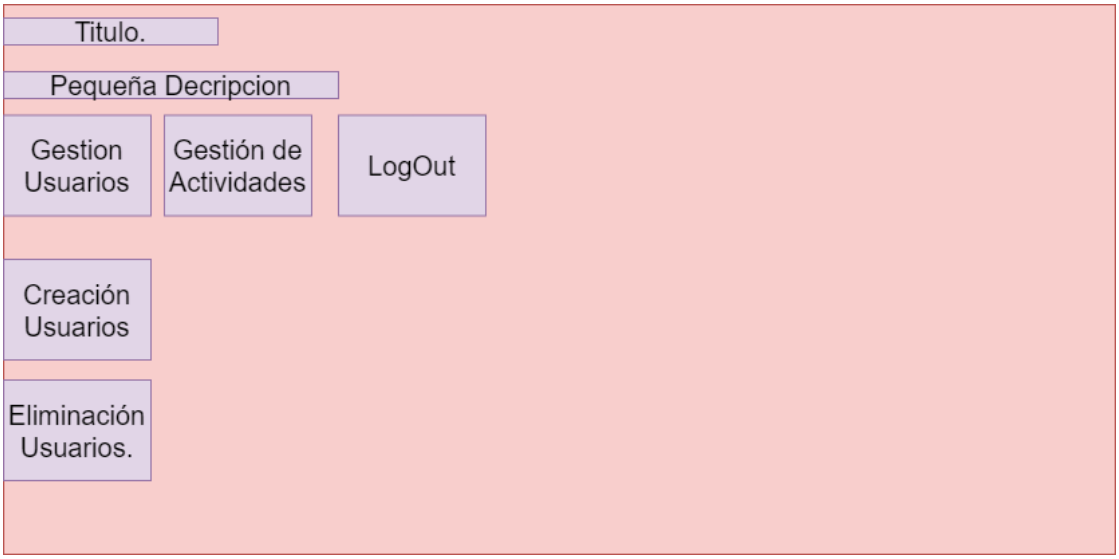


Figura 4.4: Diseño de la parte de gestión de usuario en la interfaz web de administración.

La página de administración de usuarios pretende tener un diseño como el que se muestra en la figura 4.4 que facilite la gestión de los usuarios del sistema por parte de los administradores. Manteniendo la estructura básica del Home que proporciona consistencia en la navegación y ayudando el acceso a las demás funciones principales de la web. Como ya se han descrito los botones y demás elementos del **Home** solo se describen los nuevos elementos que aparecen en esta vista.

A continuación se describen las interfaces de usuario sobre las funcionalidades de la página de administración. Para empezar, se explica el botón de **Gestión de Usuarios.**

1. **Descripción General:**
   - *Creación de Usuarios:* Brinda un formulario para agregar nuevos usuarios.
   - *Eliminación Usuarios:* Ofrece un formulario para la eliminación de usuarios existentes en el sistema.
2. **Navegación y funcionalidades:**
   - *Creación de Usuarios:* La función este botón es ofrecer un formulario para la creación de usuarios. Este formulario contiene un campo para insertar el nombre de usuario y otro para insertar la contraseña. Su propósito se basa en la adición de nuevos usuarios que permite una rápida expansión de la base de datos de usuarios según sea conveniente.
   - *Eliminación de Usuarios:* La función de este botón consiste en el despliegue de un formulario para eliminar usuarios. Contiene un campo para insertar el nombre del usuario que se desee eliminar. Permite la eliminación de usuarios

existentes en el sistema ágilmente, asegurando que solo los usuarios autorizados tengan acceso al sistema.

Por último, se describe el botón de **Gestión de actividades**. La página de administración de actividades pretende tener un diseño como el de la figura 4.5 para brindar a los administradores el control de las actividades del sistema. Dicha sección mantiene, al igual que la anterior, la estructura base del Home, asegurando una fácil navegación a las funciones esenciales de la web.

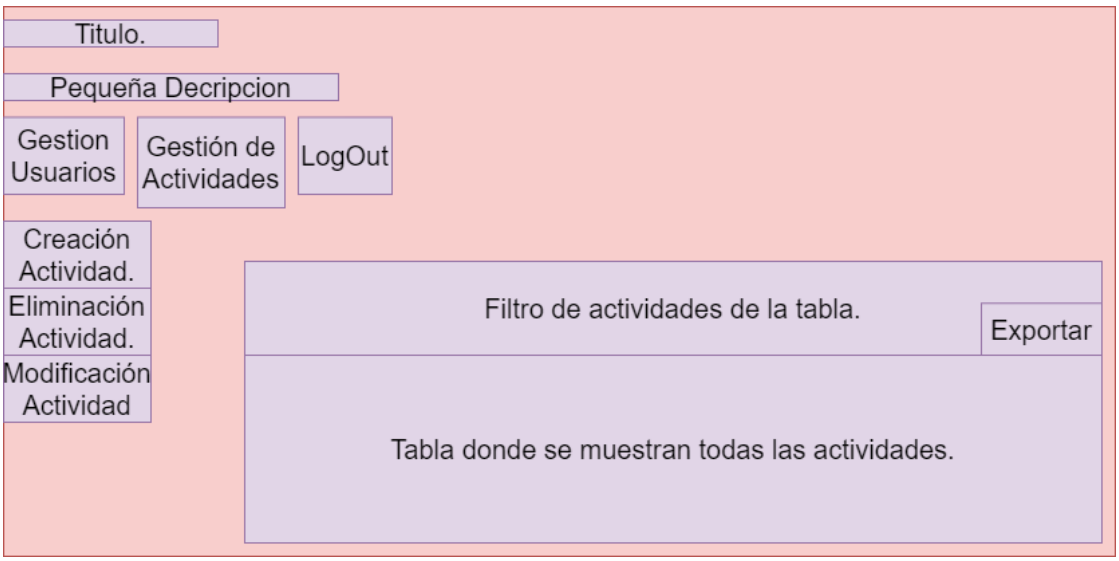


Figura 4.5: Diseño de la parte de gestión de actividades en la interfaz web de administración.

1. **Descripción General:**
   - *Creación de actividades:* Brinda un formulario para agregar nuevas actividades.
   - *Eliminación de actividades:* Ofrece un formulario para la eliminación de actividades existentes en el sistema.
   - *Modificación de actividades:* Ofrece un formulario para la modificación de actividades existentes en el sistema.
   - *tabla de actividades:* Muestra todas las actividades existentes en el sistema.
   - *Filtro de la tabla de actividades:* Filtro dedicado para que aparezcan en la tabla las actividades específicas.
   - *Exportar:* Permite descargar en un formato predefinido las actividades actuales de la tabla, al igual que realizar un envío por correo o publicación en X de las mismas, según se requiera.

2. **Navegación y funcionalidades:**

   - *Creación de actividades:* Al pinchar en este botón, debe aparecer un formulario para crear nuevas actividades. Este formulario tendrá tantos campos como se decidan en posteriores capítulos. El propósito es facilitar la adición de actividades nuevas de manera rápida y sencilla.
   - *Eliminación de actividades:* Al pinchar sobre este botón debe aparecer un formulario con un campo a rellenar con el nombre de la actividad para borrarla del sistema. Permite la eliminación de manera sencilla y ágil.
   - *Modificación de actividades:* Al pinchar en este botón, debe aparecer un formulario para modificar actividades existentes. Este formulario tendrá tantos campos como se decidan en posteriores capítulos. El propósito es facilitar la modificación de actividades nuevas de manera rápida y sencilla.
   - *tabla de actividades:* Provee de una lista con todas las actividades del sistema. Ofrece un filtro para buscar actividades específicas según varios criterios que se definirán en capítulos posteriores. Además, tiene la capacidad de exportar en diferentes métodos las actividades actuales de la tabla, lo que facilita la distribución y el almacenamiento de información sobre actividades.

El componente de administración de actividades pretende tener un diseño que proporcione a los administradores un control completo sobre la creación, eliminación y modificación de actividades. Manteniendo una base consistente y varias opciones de navegación claras, lo que habilita una gestión fácil sobre todas y cada una de las actividades en el sistema. Los formularios específicos para crear, modificar o eliminar actividades ayudan a los administradores actualicen ágilmente la información, mientras que la tabla de actividades y el filtro asociado proporcionan una visión clara y filtrada de las mismas, con opciones adicionales para compartir y exportar información.

4.3. Frontend.

El diseño del frontend de las páginas web se caracteriza por una etapa crucial en el desarrollo de cualquier aplicación web. En esta sección se aborda la estructuración preliminar de las clases y su comunicación entre ellas. Esto brinda una base sólida para la implementación del frontend. La descripción general de las clases y su interacción se orienta en ayudar a la creación de una interfaz coherente y funcional, sin especificar aún el framework que se va a escoger para su desarrollo [32], [33] [34]. La utilización de principios y patrones de diseño ayudan a que el código sea mantenible, reutilizable y escalable, lo que ofrece poder evolucionar de forma eficiente el frontend a medida que se desarrollan nuevas funcionalidades.

4.3.1. Web de usuarios.

En la planificación inicial del frontend de la web de usuarios, se identifican las siguientes clases principales, que servirán como los bloques fundamentales para la implementación de la web. A continuación vamos a realizar una descripción general de sus clases a partir de la figura 4.6.

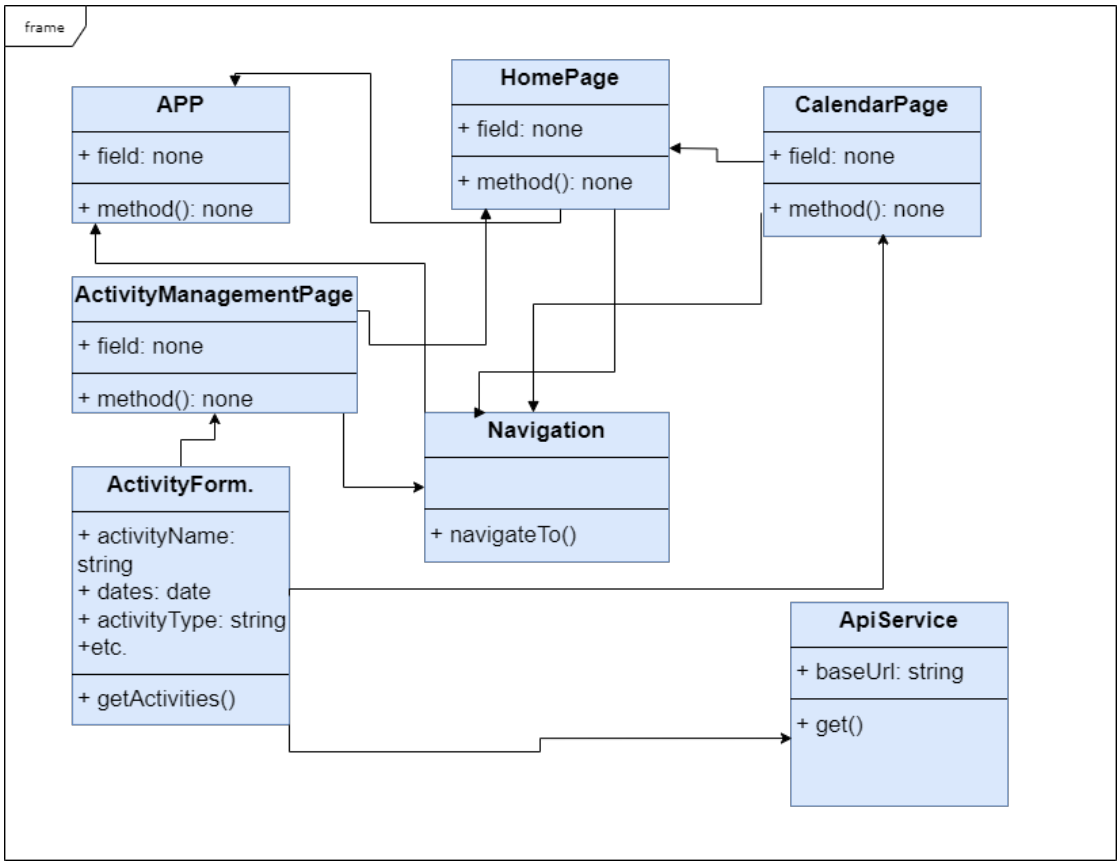


Figura 4.6: Diagrama UML inicial de la interfaz web de usuarios.

1. Clase APP:
   - Competencia: Actúa como el punto de entrada principal de la aplicación.
   - Funciones:
     - Inicializar la aplicación.
     - Gestionar el estado global de la aplicación.
     - Coordinar la navegación entre las diferentes vistas.
2. Clase HomePage:

- Competencia: Representa la página de Inicio.
- Funciones:
  - Mostrar el título y la descripción.
  - Proporcionar botones de navegación a otras páginas.
  - Gestiona el estado local relacionado con la navegación.

3. Clase ActivityForm:
   - Competencia: Manejar las actividades devueltas por el sistema.
   - Funciones:
     - Validar los datos devueltos por el sistema.
     - Servir información a las demás clases.
4. Clase Navigation:
   - Competencia: Manejar la navegación entre las distintas páginas de la aplicación.
   - Funciones:
     - Proporcionar una interfaz coherente para la navegación.
     - Mantener el estado de navegación.
5. Clase ApiService:
   - Competencia: Define la URL base para comunicarse con la API.
   - Funciones:
     - Manejar correctamente los métodos get(), post(), put(), delete() de las solicitudes HTTP.
6. Clase searchActivities:
   - Competencia: Recoger los requerimientos del usuario para la búsqueda de actividades.
   - Funciones:
     - Mostrar las actividades requeridas en el filtro por el usuario.
7. Clase CalendarPage:
   - Competencia:Tener un calendario donde aparezcan todas las actividades del sistema.
   - Funciones:
     - Mostrar las actividades de manera correcta en las fechas en las que se han creado.

4.3.2. Web Administración.

En la planificación inicial del frontend de la web de usuarios, se identifican las siguientes clases principales, que servirán como los bloques fundamentales para la implementación de la web. A continuación vamos a realizar una descripción general de sus clases a partir de la figura 4.7.

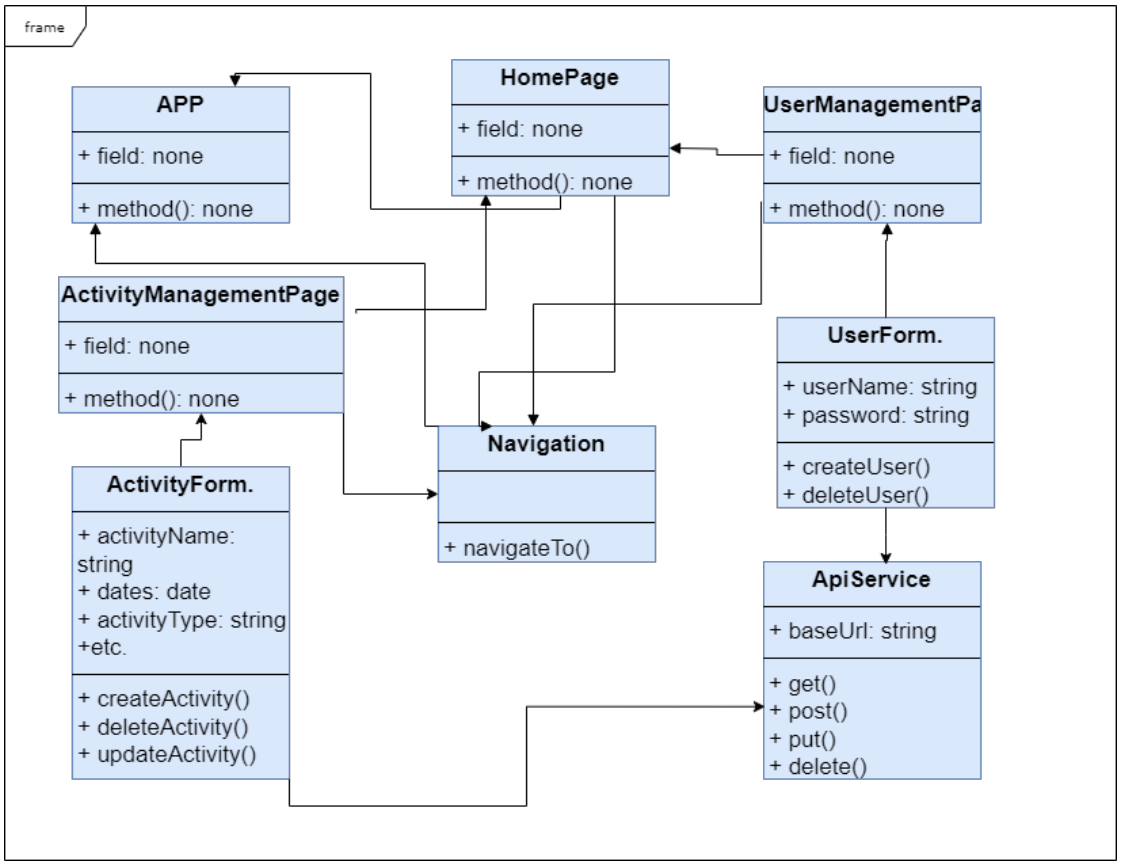


Figura 4.7: Diagrama UML inicial de la interfaz web de administración.

1. Clase APP:
   - Competencia: Actúa como el punto de entrada principal de la aplicación.
   - Funciones:
     - Inicializar la aplicación.
     - Gestionar el estado global de la aplicación.
     - Coordinar la navegación entre las diferentes vistas.
2. Clase HomePage:

- Competencia: Representa la página de Inicio.
- Funciones:
  - Mostrar el título y la descripción.
  - Proporcionar botones de navegación a otras páginas.
  - Gestiona el estado local relacionado con la navegación.

3. Clase ActivityForm:
   - Competencia: Manejar las actividades devueltas por el sistema.
   - Funciones:
     - Validar los datos devueltos por el sistema.
     - Servir información a las demás clases.
4. Clase Navigation:
   - Competencia: Manejar la navegación entre las distintas páginas de la aplicación.
   - Funciones:
     - Proporcionar una interfaz coherente para la navegación.
     - Mantener el estado de navegación.
5. Clase ApiService:
   - Competencia: Define la URL base para comunicarse con la API.
   - Funciones:
     - Manejar correctamente los métodos get(), post(), put(), delete() de las solicitudes HTTP.
6. Clase UserForm:
   - Competencia: Manejar los formularios de creación y eliminación de usuarios
   - Funciones:
     - Validar y enviar los datos del formulario.
     - Mostrar mensaje de éxito o error según la respuesta del sistema.
7. Clase UserManagmentPage:
   - Competencia: Gestiona la interfaz para la administración de usuarios.
   - Funciones:
     - Proporcionar formularios para la creación y eliminación de usuarios.
8. Clase ActivityManagmentPage:
   - Competencia: Gestiona la interfaz para la administración de actividades.

- Funciones:
  - Mostrar lista de actividades.
  - Proporcionar formularios para la creación, actualización y eliminación de actividades.
  - Filtrar y exportar la lista de actividades.
  - Actualizar la lista de actividades en respuesta a las acciones del usuario.

4.3.3. Comunicación entre clases.

La comunicación que se realiza entre las clases se realiza a través de eventos,estados compartidos y props. A continuación, se describe cómo se comunican entre sí, poniendo de ejemplo la web de administración, ya que la web de usuarios es igual pero con algunas clases cambiadas.

1. Comunicación entre *APP* y *HomePage*:
   - *APP* pasa el estado global y las funciones de navegación como props a *HomePage*
   - *HomePage* puede emitir eventos para solicitar cambios del estado global o navegación.
2. Comunicación entre *HomePage* y *UserManagementPage* / *ActivityManagementPage*:
   - *HomePage* pasa las funciones de navegación a *UserManagementPage* / *UActivityManagementPage*
   - Ambas páginas gestionan su propio estado local y comunican cambios relevantes a *HomePage*.
3. Comunicación entre *UserManagementPage* / *ActivityManagementPage* y *UserForm*/*ActivityForm*:
   - *UserManagementPage* pasa las funciones de creación y eliminación a *UserForm*.
   - *UserManagementPage* pasa las funciones de creación, actualización y eliminación a *ActivityForm*.
   - Los formularios validan los datos y envían resultados de vuelta a las páginas de gestión.

El diseño del frontend establece una estructura clara y modular, en la que cada clase tiene una responsabilidad específica y bien definida. La comunicación entre las clases está diseñada para ser eficiente y coherente. Este enfoque modular ayuda al desarrollo, mantenimiento y escalabilidad de las páginas, lo que proporciona una base estable para la implementación de cualquier framework que se decida utilizar.

4.4. Backend.

En esta sección vamos a realizar el diseño preliminar del backend que es fundamental para la correcta gestión y manipulación de los datos y la lógica de negocio de la aplicación. Para ello, se realiza un diseño sobre las clases y la comunicación entre ellas, ofreciendo una base estable para la implementación del backend de la API. No se nombra ni especifica ningún framework, ya que el objetivo es proporcionar una estructura conceptual que se pueda desarrollar en distintos entornos tecnológicos [35], [36], [37]. A continuación, se van a describir las clase que se muestran en la figura 4.8 sobre el diagrama UML que se ha realizado, el cual brinda una guía para la implementación del backend.

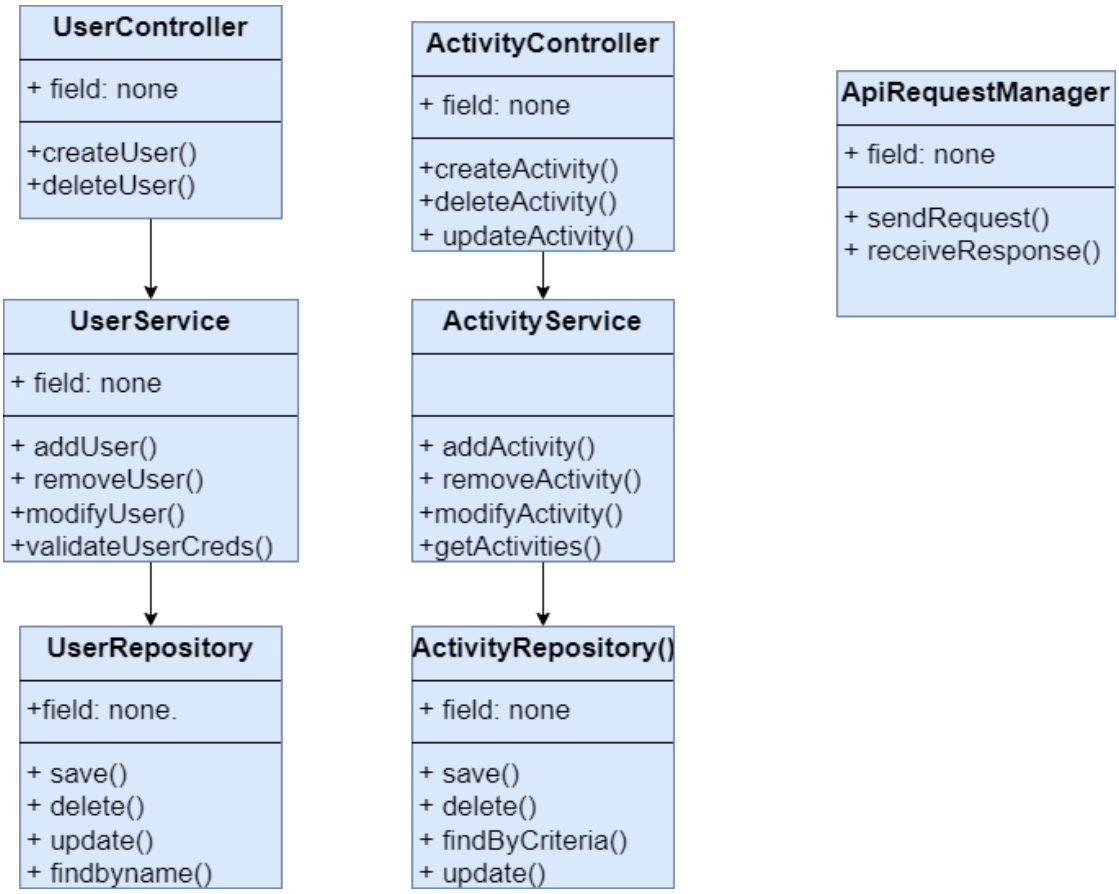


Figura 4.8: Diagrama UML inicial del Backend.

1. Controlador de Usuarios (UserController)
    - Descripción: Su finalidad reside en gestionar las solicitudes relacionadas con los usuarios.
    - Métodos principales:

- createUser()
- deleteUser()

2. Controlador de Actividades (ActivityController)
   - Descripción: Su finalidad reside en gestionar las solicitudes relacionadas con las actividades.
   - Métodos principales:
     - createActivity()
     - deleteActivity()
     - updateActivity()
3. Servicio de Usuarios (UserService)
   - Descripción: Su finalidad reside en contener la lógica de negocio que está relacionada con los usuarios.
   - Métodos principales:
     - addUser()
     - removeUser()
     - validateCredentials()
4. Servicio de Actividades (ActivityService)
   - Descripción: Su finalidad reside en contener la lógica de negocio que está relacionada con las actividades.
   - Métodos principales:
     - addActivity()
     - removeActivity()
     - modifyActivity()
     - getActivities()
5. Repositorio de Usuarios (UserRepository)
   - Descripción: Su finalidad reside en gestionar la persistencia de los datos de los usuarios.
   - Métodos principales:
     - save()
     - delete()
     - update()
     - findByName()
6. Repositorio de Actividades (ActivityRepository)

- Descripción: Su finalidad reside en gestionar la persistencia de los datos de las actividades.
- Métodos principales:
  - save()
  - delete()
  - update()
  - findByCriteria()

7. Repositorio de Actividades (ActivityRepository)
   - Descripción: Su finalidad reside en manejar las solicitudes HTTP a la API.
   - Métodos principales:
     - receiveResponse()
     - sendRequest()

Comunicación entre Clases

- Los controladores son los que se comunican con los servicios para ejecutar la lógica de negocio necesaria.
- Los servicios son los encargados de comunicarse con los repositorios para realizar las operaciones de persistencia en la base de datos.
- ApiRequestManager es la entidad que gestiona las solicitudes y respuestas entre el frontend y el backend.

4.5. Servicio REST.

En el diseño del servicio REST es muy importante definir cómo los diferentes componentes del sistema se comunicarán entre sí a través de la API. En este capítulo, se describen las URLs preliminares del proyecto, sus propósitos y los métodos HTTP al que están asociados (GET, POST, PUT, DELETE ,etc.). La API se ha diseñado siguiendo el modelo de madurez de Richardson para preservar una implementación RESTful completa [38] , [37], asegurando que la utilización de la API sea sencilla y pueda escalarse de manera óptima para manejar un gran número de peticiones. Este diseño preliminar proporciona una base sólida para el desarrollo de la API, independientemente del framework específico que se utilice.

1. Usuarios.
   - **Crear Usuario**
     - *URL:* /api/usuarios
     - *Método:* POST.

- *Descripción:* Crear un nuevo usuario en el sistema.
- *Datos del cuerpo:* {nombre: "string", contraseña: "string"}

- **Eliminar Usuario**
  - *URL:* /api/usuarios/{nombre}
  - *Método:* DELETE.
  - *Descripción:* Elimina un usuario existente del sistema.
  - *Parámetros de la URL:* nombre
- **Obtener Usuario**
  - *URL:* /api/usuario/{Id}
  - *Método:* GET.
  - *Descripción:* Obtiene un usuario que coincida con su Id.
  - *Parámetros de la URL:* ID

2. Actividades.

- **Crear Actividad**
  - *URL:* /api/actividades/
  - *Método:* POST.
  - *Descripción:* Crear una nueva actividad en el sistema.
  - *Datos del cuerpo:* { nombre: "string", fechas: "string", tipo: "string", otrosDatos: "json"}
- **Eliminar Actividad**
  - *URL:* /api/actividades/{nombre}
  - *Método* DELETE.
  - *Descripción* Elimina una actividad existente del sistema.
  - *Parámetros de la URL:* nombre
- **Obtener Actividades**
  - *URL:* /api/actividades
  - *Método:* GET
  - *Descripción:* Obtiene la lista de todas las actividades.
- **Modificar Actividades**
  - *URL:* /api/actividades/{nombre}
  - *Método:* PUT
  - *Descripción:* Modifica la información de una actividad existente.
  - *Datos del cuerpo:* {nuevosDatos: "json"}
- **Exportar Actividades**
  - *URL:* /api/actividades/exportar/
  - *Método:* POST.
  - *Descripción:* Exporta la lista de actividades a un archivo o las envía a una plataforma.
  - *Datos del cuerpo:* {formato: "string",destino: "string"}

4.6. BBDD.

El diseño de la base de datos se ha construido para cubrir las necesidades de la aplicación. Se han diseñado varias tablas para almacenar la información con la que se trabaja en esta aplicación, con relaciones entre ellas que aseguran la integridad de los datos. La elección entre bases de datos relacionales y NoSQL se basa en sus características específicas, donde se consideran factores como la escalabilidad y flexibilidad [39], [40]. Si se realiza un diseño de datos robusto, se preserva que los datos almacenados se recuperen de manera idónea, garantizando la consistencia y disponibilidad requeridas para un buen funcionamiento de la aplicación.

4.6.1. Diseño de bases de datos relacionales.

El diseño de la base de datos es una parte esencial para la implementación de cualquier aplicación o servicio. En este capítulo, se describen las estructuras de las tablas, relaciones y tipos de datos que se podrían llegar a utilizar. Este diseño preliminar brinda una guía clara para la implementación de la base de datos, independientemente del sistema de gestión de bases de datos (DBMS).

1. Tabla de Usuario.
   - Nombre de la tabla: usuarios.
   - columnas:
     - ID (INT,PRIMARY KEY, AUTO_INCREMENT) Identificador único del usuario.
     - nombre (VARCHAR(50)): nombre del usuario.
     - contraseña (VARCHAR(255): Contraseña encriptada del usuario.
   - índices:
     - nombre_idx (nombre): índice para búsquedas rápidas por nombre.
2. Tabla de Actividades.
   - Nombre de la tabla: actividades.
   - Columnas:
     - ID (INT,PRIMARY KEY, AUTO_INCREMENT) Identificador único de la actividad.
     - nombre (VARCHAR(100)): nombre de la actividad.
     - fecha_inicio (DATETIME): Fecha y hora de inicio.
     - fecha_fin (DATETIME): Fecha y hora de finalización.
     - tipo (VARCHAR(50)): tipo de actividad.
   - índices:

- nombre_idx (nombre): índice para búsquedas rápidas por nombre.

3. Relaciones entre tablas.

   - Usuarios y Actividades: Relación una a muchas entre usuarios y actividades: Un usuario puede crear múltiples actividades, pero una actividad solo puede ser creada por un usuario.

4.6.2. Diseño de bases de datos no relacionales.

El diseño de la base de datos es una parte primordial de la implementación de cualquier aplicación o servicio. en este subapartado, se muestra la estructura de las colecciones y documentos, sus relaciones y los tipos de datos que se podrían llegar a utilizar. Este diseño preliminar proporciona una guía clara para el desarrollo de la base de datos, independientemente del sistema de base de datos NoSQL que se elija.

1. Colección de Usuarios.

   - Nombre de la colección: usuarios.
   - Documento:
     - _id: Identificador único del usuario.
     - nombre: Nombre del usuario.
     - contraseña: Contraseña encriptada del usuario.

2. Colección de Actividades.

   - Nombre de la colección: actividades.
   - Columnas:
     - _id: Identificador único de la actividad.
     - nombre: Nombre de la actividad.
     - fecha_inicio: Fecha y hora de inicio.
     - fecha_fin: Fecha y hora de finalización.
     - tipo: Tipo de actividad.

3. Relaciones entre Colecciones.

   - Usuarios y Actividades: Relación una a muchos entre usuarios y actividades: Un usuario puede crear múltiples actividades, pero una actividad solo puede ser creada por un usuario.

Capítulo 5

Implementación de la aplicación.

La fase de implementación es vital en el ciclo de vida del desarrollo de software, donde las ideas y diseños previos se transforman en un sistema funcional y operativo. En este capítulo, se explica el proceso de realización del sistema propuesto, que se basa en el diseño conceptual del **capitulo 4**. Detallamos las decisiones técnicas, las tecnologías utilizadas y los pasos que se han llevado a cabo para la implementación correcta del frontend, backend y su integración. El objetivo se basa en proporcionar una guía estructurada que permita replicar el desarrollo del sistema, asegurando que se cumplan las necesidades funcionales y no funcionales que están definidas previamente. Esta fase está organizada en varias secciones, en las que se incluyen la configuración del entorno de desarrollo, el desarrollo del frontend y desarrollo del backend; incluyendo su integración. Cada sección detalla los componentes implementados y las interacciones necesarias para asegurar una comunicación fluida y eficiente entre las partes del sistema. A continuación, se detallan las etapas y componentes de la implementación, aportando ejemplos de código y configuraciones específicas utilizadas en la implementación del sistema.

5.1. Justificación del entorno de desarrollo.

Esta sección comienza justificando los frameworks que se han elegido descritos en el **capitulo 3. Estado del Arte.** para la implementación de la aplicación web.

1. **Angular:** Se ha elegido Angular para el framework de frontend ya que tiene una amplia adopción en la industria y cuenta con una robusta arquitectura. Según el informe de *Stack Overflow Developer Survey 2023*, este framework está entre las herramientas más utilizadas de frontend, con un 45 % de desarrolladores que lo utilizan diariamente [41]. Además, la documentación oficial que proporciona Angular incluye una guía exhaustiva y ejemplos prácticos que facilitan su aprendizaje y uso [42]. La arquitectura está basada en componentes que promueven modularidad y mantenimiento del código a largo plazo, lo que la hace idónea para aplicaciones

como la que se desea implementar [43].

2. **RESTLET:** En la implementación de la API REST, se ha elegido RESTLET ya que cuenta con una eficiencia y versatilidad durante el desarrollo de servicios web RESTful. Según el informe de *The State of API 2023*, este framework es ampliamente utilizado en la industria y está demostrado que ofrece un rendimiento superior en comparación con otros frameworks [44]. Está centrado en recursos y cuenta con una facilidad de implementación de los métodos HTTP, lo que lo hace un gran candidato para el desarrollo de APIs escalables y seguras [45]. Además, tiene una comunidad activa que brinda un sólido soporte técnico y una rápida resolución de problemas [46].

3. **MongoDB:** En la parte de la base de datos, se ha escogido MongoDB ya que cuenta con una capacidad para la gestión de datos no estructurados y su escalabilidad horizontal. Según el informe de **DB-Engines Rankingt 2023**, esta base de datos NoSQL es de las más populares y se utiliza ampliamente en aplicaciones modernas [47]. Su esquema flexible y su capacidad para el almacenamiento de datos en formato JSON ofrecen una mayor agilidad en su implementación y ayudan en la adaptación a cambios de requisitos del proyecto [48]. Por último, brinda unas capacidades de escalabilidad horizontal que son adecuadas para manejar grandes volúmenes de datos y un crecimiento idóneo sin problemas del sistema [49].

Una vez se ha realizado la elección de las herramientas que se van a utilizar para el desarrollo de la aplicación, como se muestran en la figura 5.1. Se procede a explicar la implementación de cada una de sus partes.

5.2. BBDD.

En este apartado se procede a cómo se ha realizado la implementación de la base de datos NoSQL MongoDB. Desde su instalación hasta la definición de la base de datos, creación de los componentes y definición de los documentos asociados a cada colección.

En primer lugar, explicamos el proceso de instalación. En el sitio web oficial [50] se debe descargar la versión para Windows, ya que es el sistema operativo utilizado en este proyecto. Se inicia la instalación por defecto y una vez que se termina, se verifica su instalación mediante el siguiente comando.

Listing 5.1: Comprobar versión despues de la instalacion.

```
1 mongod --version
```

El resultado del comando se muestra en la figura 5.2.

Una vez se ha realizado la instalación de la base de datos, se procede a la creación de la base de datos y las colecciones necesarias para nuestra aplicación. Se detallan a con-

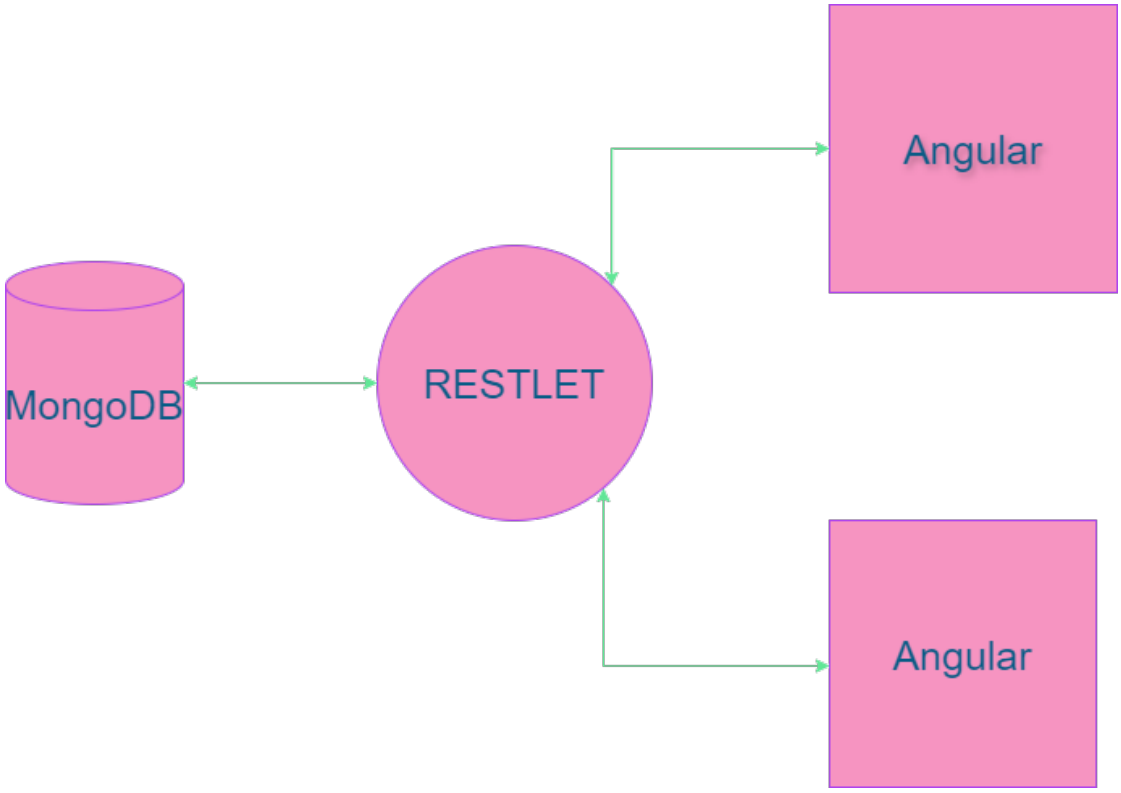


Figura 5.1: Esquema de la aplicación web con los frameworks elegidos.

tinuación los pasos para crear la base de datos llamada *base_prueba_v0* y las colecciones *UserApi* y *Actividades* con sus respectivos documentos.

El primer paso es abrir una terminal y acceder a la consola de mongooDB. Para crear la base de datos se usa el comando *use "nombreBaseDatos"*, si la base de datos no existe, MongoDB la crea y accede a ella. Una vez creada la base de datos y están dentro de ella en la consola a través del comando *db.createCollection("nombreColeccion")* se crean las colecciones correspondientes, en este caso UserApi y Actividades y mediante el comando *db."nombreColeccion". insertOne()* se crean los diferentes documentos con sus parámetros correspondientes. Todo este proceso se ejemplifica en la lista 5.2.

Listing 5.2: Creacion y verificacion de elementos en MongoDB.

```
use base_prueba_v0
db.createCollection("UsersApi")
db.UsersApi.insertOne({
  username: "exampleUser",
  hashedPassword: "exampleHashedPassword",
  salt: "exampleSalt"
})
db.createCollection("Actividades")
db.Actividades.insertOne({
```

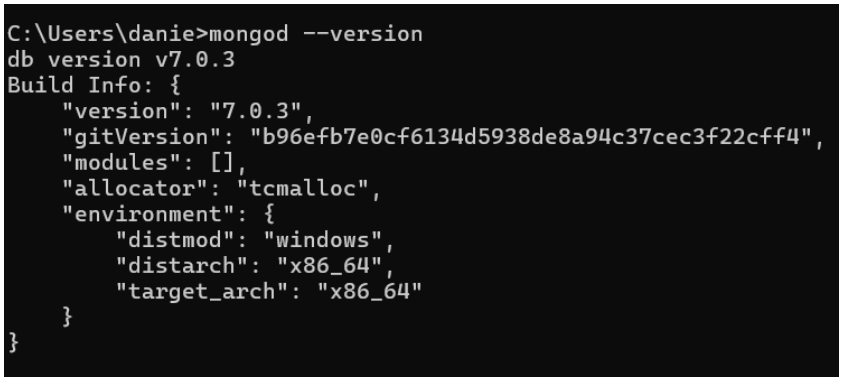


Figura 5.2: Resultado de la verificación de la instalación.

```
  titulo: "exampleTitle",
  tipo: "exampleType",
  fechaInicio: new Date("2024-06-01"),
  fechaFinal: new Date("2024-06-02"),
  ponente: ["exampleSpeaker1", "exampleSpeaker2"],
  organizador: ["exampleOrganizer"],
  lugar: "examplePlace",
  limiteAsistentes: "100",
  enlaceInformacion: "http://example.com/info",
  enlaceInscripcion: "http://example.com/register",
  descripcion: "exampleDescription",
  categoria: ["exampleCategory1", "exampleCategory2"]
})
show collections
db.UsersApi.find().pretty()
db.Actividades.find().pretty()
```

Por último, se verifican los elementos creados mediante los comandos *show collections* y *db."nombreColeccion".find().pretty()*. Una vez se han creado estos elementos indispensables para el funcionamiento de la aplicación, se llevan a cabo las siguientes acciones que se van a mostrar para la gestión de los usuarios y roles dentro del framework de MongoDB, esta asegura que los usuarios que utilizaran la API tengan los permisos adecuados para interactuar con la misma.

Una vez accedemos a la consola de mongo, como se explica en procedimientos anteriores, como lista 5.2, comenzamos creando un usuario con privilegios administrativos, para que pueda realizar operaciones de lectura y escritura en cualquier base de datos. Este proceso se muestra en la lista 5.3. Este comando crea el usuario privilegiado con su contraseña específica y los roles userAdminAnyDatabase y readWriteAnyDatabase.

Listing 5.3: Creacion usuario admin.

```
use admin
db.createUser({
  user: "ManagerDBs",
  pwd: "Zurbaran14",
  roles: [
```

```
    { role: "userAdminAnyDatabase", db: "admin" },
    { role: "readWriteAnyDatabase", db: "admin" }
  ]
})
```

A continuación se explica la creación de los usuarios específicos para la base de datos *¨base_prueba_v0.* en la lista 5.4 se exponen los comandos necesarios.

Listing 5.4: Creacion usuarios con roles en base de datos.

```
use base_prueba_v0
db.createUser({
  user: "ReadActivities",
  pwd: "someStrongPassword",
  roles: [
    { role: "read", db: "base_prueba_v0" }
  ]
})
db.createUser({
  user: "insertUser",
  pwd: "anotherStrongPassword",
  roles: [
    { role: "readWrite", db: "base_prueba_v0" }
  ]
})
```

En la lista 5.4, la única diferencia que existe respecto a la figura 5.3 es que los usuarios se crean en la base de datos donde se requiere, en vez de en *admin*. Por último, mediante el comando *db.getUser()* se verifica el éxito de esta última operación y deja los usuarios listos para que en la implementación de la API (**sección Backend**) puedan añadirse y que la comunicación entre ambos elementos del sistema sea segura.

5.3. Backend.

La implementación del backend es primordial en la arquitectura de cualquier aplicación web, es la responsable de la gestión lógica del negocio, manejar la interacción con las bases de datos, y brindar APIs para que los clientes (frontends) lleguen a comunicarse con el servidor de manera eficiente y segura. En este proyecto, se ha desarrollado un backend robusto utilizando varias tecnologías y prácticas de desarrollo para garantizar escalabilidad, seguridad y rendimiento.

Este backend se basa en Java, ya que utiliza el framework de Restlet para crear servicios RESTful, lo que ayuda a la interoperabilidad con diferentes clientes y servicios externos. Como ya se comentó en el apartado **4.1 Justificación del entorno de desarrollo.** la gestión de los datos se realiza a través de MongoDB, una base de datos NoSQL que aporta flexibilidad y escalabilidad, idónea para manejar grandes cantidades de datos estructurados y no estructurados.

A continuación, se especifican los principales componentes y clases del backend, des-

cribiendo sus responsabilidades y cómo interactúan entre sí para conformar una arquitectura coherente y eficiente. En este apartado se proporciona una visión sobre las decisiones de diseño y las tecnologías utilizadas, destacando la integración de autenticación mediante JWT, la gestión de usuarios y la interacción con servicios externos como X. El código de esta parte del proyecto se aloja en el siguiente repositorio de GitHub [51]. En la figuras 5.3, 5.4 y 5.5 se muestra el diagrama UML seccionado del backend completo, se realiza de esta manera ya que se separan claramente las diferentes partes que conforman el backend y sus funcionalidades asociadas.

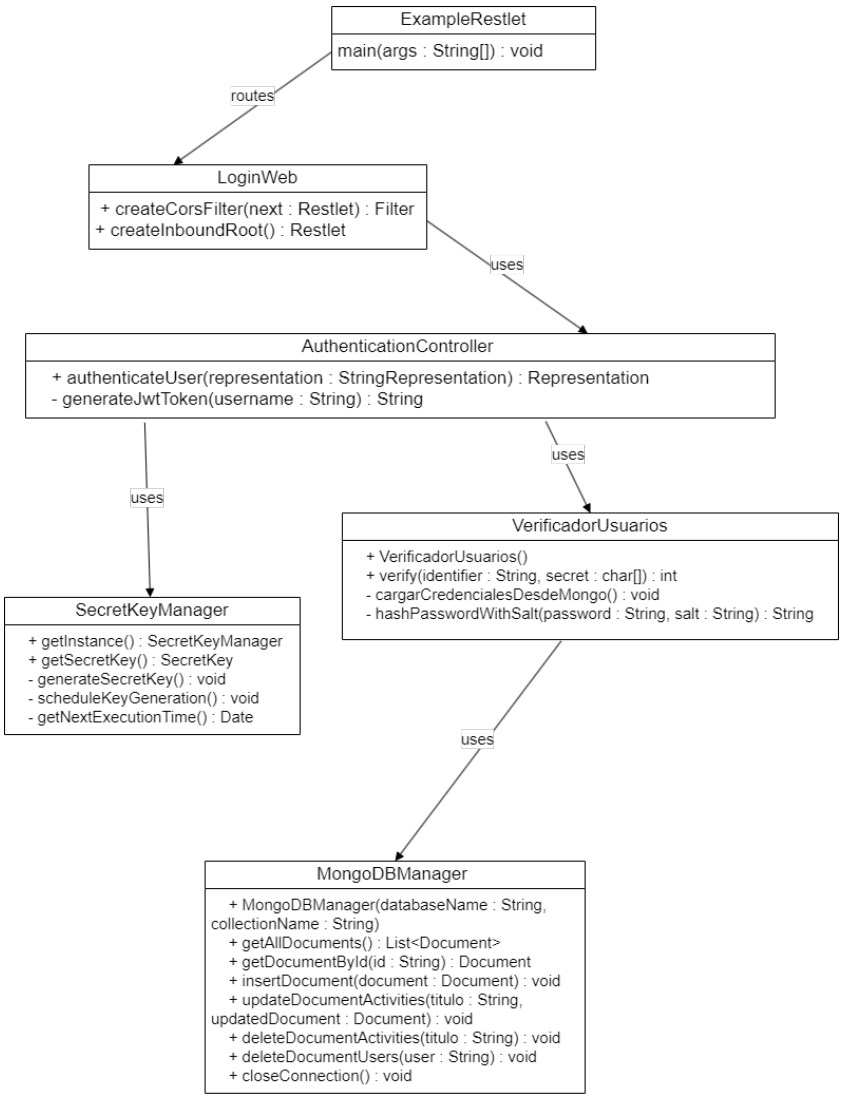


Figura 5.3: Diagrama UML del backend.

En el desarrollo del backend, las clases **ExampleRestlet**, **ServerWeb** y **LoginWeb** tienen un papel fundamental en la configuración y funcionamiento del servidor RESTful.

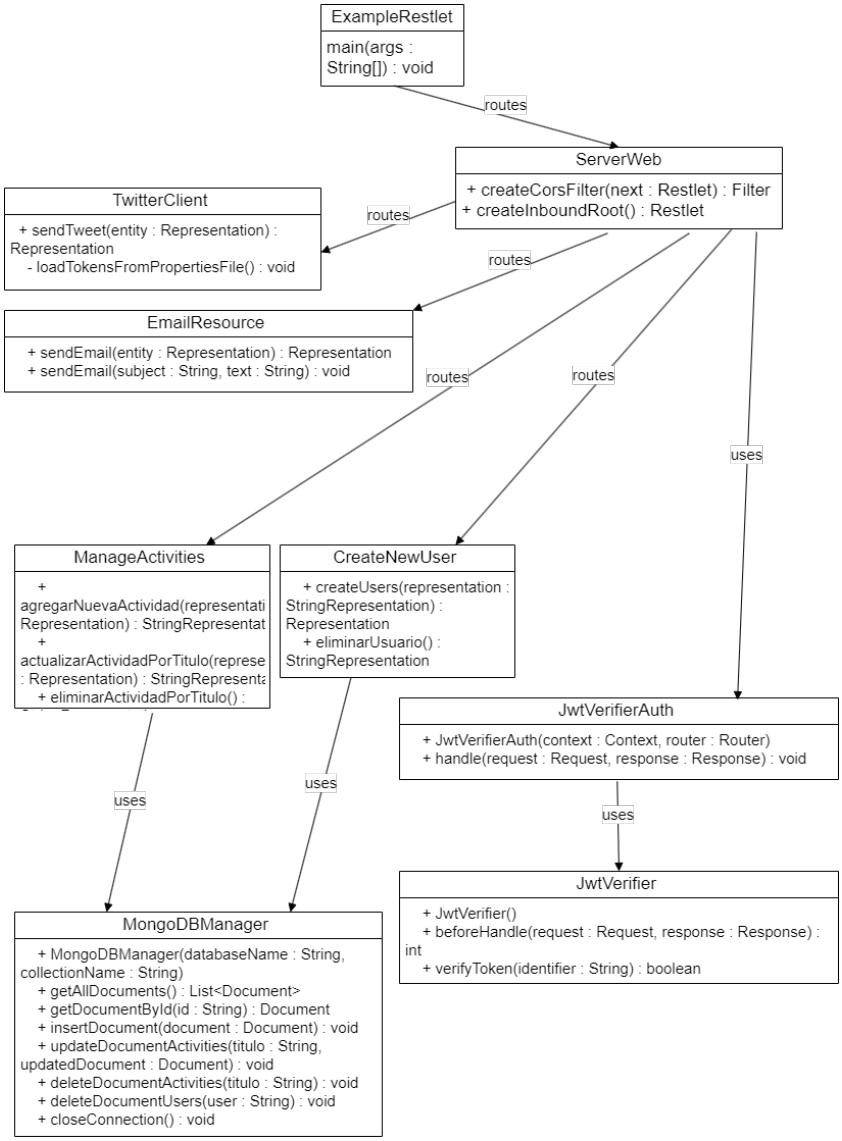


Figura 5.4: Diagrama UML del backend.

A continuación, se describen en detalle la función y propósito de cada una de ellas.

5.3.1. ExampleRestlet.

La clase llamada **ExampleRestlet** sirve como punto de entrada principal para la API. Tiene la responsabilidad de iniciar y configurar el servidor Restlet. Configura y pone en funcionamiento un componente Restlet con un servidor HTTPS, donde se especifican los parámetros necesarios para la configuración SSL. Además, se adjuntan dos

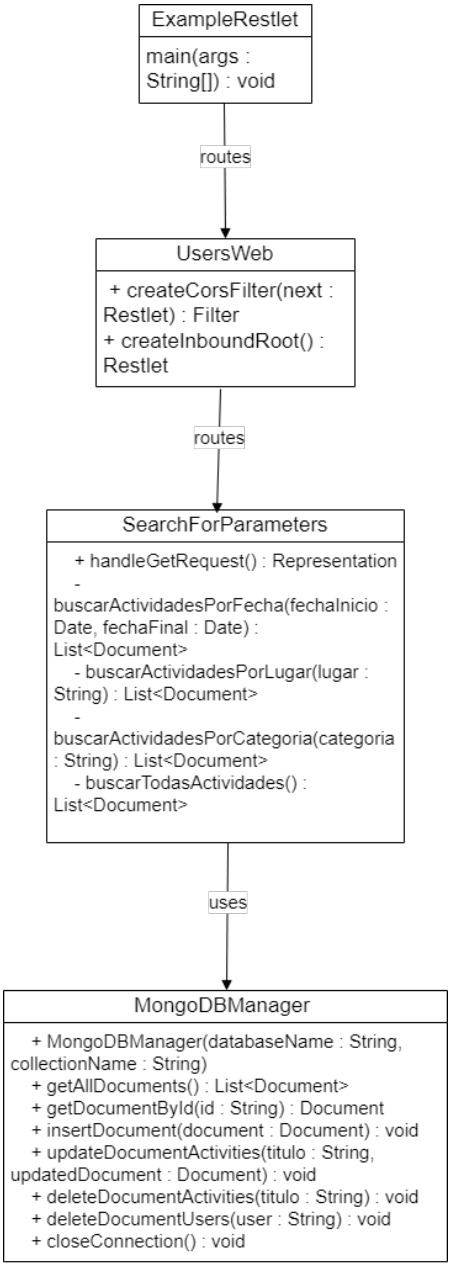


Figura 5.5: Diagrama UML del backend.

aplicaciones Restlet a rutas específicas y las inicia. Esto permite la gestión de peticiones HTTPS seguras dirigidas a las rutas **/ApiServerWeb**, **/ApiUserWeb** y **/LoginApi**.A continuación mostramos la clase para explicar su composición en la lista 5.5:

Listing 5.5: Clase pincipal Restlet.

```
1  import org.restlet.Component;
2  import org.restlet.data.Protocol;
3  import org.restlet.util.Series;
4  import org.restlet.Server;
5  import org.restlet.data.Parameter;
6
7  public class ExampleRestlet {
8
9          public static void main(String[] args) throws Exception {
10             // Create a new Component.
11             Component component = new Component();
12
13                 // Add a new HTTPS server listening on port 8183
14         Server server = component.getServers().add(Protocol.HTTPS, 8183);
15         Series <Parameter> parameters = server.getContext().getParameters();
16         parameters.add("sslContextFactory",
17         "org.restlet.engine.ssl.DefaultSslContextFactory");
18         parameters.add("keyStorePath", "certs/localhost.jks");
19         parameters.add("keyStorePassword", "NoloVasEncontrar");
20         parameters.add("keyPassword", "NoloVasEncontrar");
21         parameters.add("keyStoreType", "JKS");
22
23
24             // Attach the sample application.
25             component.getDefaultHost().attach("/ApiServerWeb", new ServerWeb());
26                 component.getDefaultHost().attach("/LoginApi", new LoginWeb());
27         component.getDefaultHost().attach("/ApiUserWeb", new UsersWeb());
28             // Start the component.
29             component.start();
30         }
31 }
```

Librerías usadas.

- org.restket.Component: Clase principal que crea y configura un componente Restlet. Un componente puede contener uno o más servidores y clientes.
- org.restlet.data.Protocol: Lista los diferentes protocolos de red soportados por Restlet (HTTP,HTTPS,etc.)
- org.restlet.util.Series: Representa una serie de parámetros o atributos.
- org.restlet.Server: Representa un servidor que gestiona peticiones y respuestas.
- org.restlet.data.Parameter: Representa un parámetro de configuración, usualmente usado en contextos de servidor o cliente.

En la *linea 11* de la lista 5.5 se crea un nuevo componente Restlet que actúa como contenedor para los servidores y aplicaciones. Desde la *línea 14 hasta la 21*, se añade un servidor HTTPS al componente escuchando en el puerto 8183. Además, se configuran los parámetros SSL, donde se incluye la ruta del almacén de claves *keyStorePath*, las contraseñas *keyStorePassword y keyPassword* y, por ultimo, el tipo de almacén de claves *keyStoreType*.

Listing 5.6: Creacion de certificado propio.

```
keytool -genkey -v -alias localhost -dname "CN=localhost,OU=IT,O=JPC,C=ES"
    -keypass NoloVasEncontrar -keystore localhost.jks -storepass NoloVasEncontrar
    -keyalg "RSA" -sigalg "MD5withRSA" -keysize 2048 -validity 3650
```

Para que las conexiones HTTPS sean seguras, se ha generado un certificado SSL mediante la herramienta *keytool*. A continuación se describe el proceso que se muestra en la lista 5.6.

- -genkey: Indica que se tiene que generar una clave.
- -v: Muestra información detallada mientras se genera la clave
- -alias: Indica un alias para la identificación de la entrada en el almacén de claves
- -dname: Aporta el nombre distintivo (Distinguished Name) para la clave.
- -keypass: Contraseña con la función de proteger la clave privada.
- -keystore: Nombre que recibe el archivo de almacén de claves.
- -storepass: Contraseña con la función de proteger el almacén de claves
- -keyalg: Algoritmo de la clave.
- -sigalg: Algoritmo de firma.
- -keysize: Tamaño de la clave.
- -validity: Periodo de validez del certificado.

En las *líneas 25 y 26* es donde se adjuntan las aplicaciones. **SeverWeb** para la ruta **ApiServerWeb** y **LoginWeb** para **/LoginApi**. Por ultimo, en la *línea 28* es donde se inicia el componente Restlet, arrancando los servidores configurados y las aplicaciones adjuntas.

5.3.2. ServerWeb y LoginWeb.

La clase ServerWeb configura la aplicación principal de Restlet, creando las rutas y filtros necesarios para su funcionamiento. Cada ruta se asigna a una clase que gestiona las solicitudes correspondientes:

- *https://localhost/ApiServerWeb/CreateNewUser:* Gestión de usuario.
- *https://localhost/ApiServerWeb/ManageActivities:* Gestión de actividades
- *https://localhost/ApiServerWeb/TwitterClient:* Envío automatizado de tweets.

- *https://localhost/ApiServerWeb/EmailResource:* Envío automatizado de Emails.

Listing 5.7: Clase ServerWeb.

```
import java.util.HashSet;
import org.restlet.Application;
import org.restlet.Restlet;
import org.restlet.routing.Router;
import org.restlet.util.Series;
import org.restlet.routing.Filter;
import org.restlet.engine.header.HeaderConstants;
import org.restlet.Request;
import org.restlet.Response;
import org.restlet.data.Header;
import org.restlet.data.Method;

public class ServerWeb extends Application {

    private Filter createCorsFilter(Restlet next) {
        Filter filter = new Filter(getContext(), next) {
    }

    @Override
    public synchronized Restlet createInboundRoot() {
        // Create a router Restlet that routes each call to a new instance of
            HelloWorldResource.
        Router router = new Router(getContext());

        // Define routes
        router.attach("/searchActivities", SearchForParameters.class);
        router.attach("/createUser", CreateNewUser.class);
        router.attach("/manageActivities", ManageActivities.class);
        router.attach("/PostTwitter", TwitterClient.class);
        router.attach("/PostEmail", EmailResource.class);
        // Intercepta todas las solicitudes
        // Crear un filtro para verificar el token JWT
        JwtVerifierAuth jwtVerifierAuth = new JwtVerifierAuth(getContext(), router
            );

        // Return the CORS filter
        return createCorsFilter(jwtVerifierAuth);
    }
}
```

Librerías usadas.

- org.restlet.Application: Base para las aplicaciones Restlet, encapsulan un conjunto de recursos y filtros.
- org.restlet.Restlet: Representa una unidad de procesamiento de solicitudes y respuestas.
- org.restlet.routing.Router: Permite el enrutamiento de solicitudes y respuestas a través del patrón de URI.
- org.restlet.util.Series: Representa una serie de parámetros o atributos.

- org.restlet.routing.Filter: Facilita el filtrado de solicitudes a recursos específicos.
- org.restlet.engine.header.HeaderConstants: Constantes utilizadas en os encabezados HTTP.
- org.restlet.Request: Representa una solicitud HTTP.
- org.restlet.Response: Representa una respuesta HTTP.
- org.restlet.data.Header: Representa una encabezado HTTP.
- org.restlet.data.Method: Contiene los métodos HTTP soportados.

Esta clase es extendida de *Application* y crea los enrutamientos y filtros que se requieren para la aplicación Restlet. Permite una configuración central para la gestión de rutas y aplicación de filtros como el filtro CORS y la verificación JWT (que se explican posteriormente en esta sección). El uso de *Application* se debe a que es una clase fundamental que representa una aplicación Web Restlet. Ofrece una estructura para gestionar y organizar recursos, enrutamientos, filtros y configuraciones.

La clase **LoginWeb** tiene la misma estructura que esta, la única diferencia es que su función se basa en la de dar permiso o no a los usuarios que intentan registrarse desde la página web de administración, por ello tiene configurado su enrutador a la ruta específica *login* que está asociada con el controlador *AuthenticationController*. Además de estas dos clases, existe la clase **UserWeb** la cual es utilizada por la web de usuarios y permite la búsqueda a través de ciertos criterios de actividades para que los alumnos puedan encontrar las actividades que más les interesan o rellenar el calendario con todas las actividades agendadas en cada mes y se muestren en él, a través de la ruta *serachActivities*, la cual no necesita autenticación y por ello no se incluye dentro de la clase **ServerWeb**. Estas funciones se explicarán más con detalle en esta y otras secciones del capítulo.

Filtro CORS.

A continuación se muestra el filtro CORS realizado:

Listing 5.8: Filtro CORS.

```
1  private Filter createCorsFilter(Restlet next) {
2         Filter filter = new Filter(getContext(), next) {
3             @SuppressWarnings("unchecked")
4             @Override
5             protected int beforeHandle(Request request, Response response) {
6                 // Initialize response headers
7                 Series<Header> responseHeaders = (Series<Header>) response
8                             .getAttributes().get(HeaderConstants.ATTRIBUTE_HEADERS
                                );
9                 if (responseHeaders == null) {
10                    responseHeaders = new Series<Header>(Header.class);
11                }
```

```
12                  // Request headers
13                  Series<Header> requestHeaders = (Series<Header>) request.
                        getAttributes().get(HeaderConstants.ATTRIBUTE_HEADERS);
14                  String requestOrigin = requestHeaders.getFirstValue("Origin",
                        false, "http://localhost:4200");
15                  String rh = requestHeaders.getFirstValue("Access-Control-Request-
                        Headers", false, "http://localhost:4200");
16                  // Set response headers
17                  HashSet<Method> methodHashSet = new HashSet<>();
18                  methodHashSet.add(Method.GET);
19                  methodHashSet.add(Method.POST);
20                  methodHashSet.add(Method.PUT);
21                  methodHashSet.add(Method.DELETE);
22
23                  response.setAccessControlAllowCredentials(true);
24                  response.setAccessControlAllowMethods(methodHashSet);
25                  response.setAccessControlAllowOrigin(requestOrigin);
26                  response.setAccessControlAllowOrigin(rh);
27
28                  java.util.Set<String> allowedHeaders = new HashSet<>();
29                  allowedHeaders.add("Content-Type");
30                  allowedHeaders.add("Authorization");
31                  allowedHeaders.add("X-Custom-Header");
32                  response.setAccessControlAllowHeaders(allowedHeaders);
33                  // Handle HTTP methods
34                  if (Method.OPTIONS.equals(request.getMethod())) {
35                      return Filter.STOP;
36                  }
37                  return super.beforeHandle(request, response);
38              }
39          };
40          return filter;
41      }
```

Surgió un problema al securizar todo el servicio debido a la configuración con las CORS. Al realizar una petición compleja (con credenciales,etc.), la API de Restlet no autorizaba realizar ningún tipo de solicitud a través de la web. Por ello, se realizó una investigación de cómo solucionar el problema, el cual tuvo solución a través de estos artículos [52], [53], [54], [55], [56] , [57]. Y se consiguió encontrar en internet una estructura de filtro válida [58] siendo la actual implementada una versión adaptada de la misma. Este filtro responde adecuadamente a solicitudes realizadas desde distintos orígenes, lo cual es necesario para aplicaciones web modernas que se despliegan en ambientes distribuidos.

El parámetro *Retlet next* que se encuentra en la *línea 1 de la lista 5.8* indica a qué siguiente componente Restlet se le debe pasar la solicitud una vez que ha sido procesada por el filtro. Se crea una clase anónima que extiende **Filter** y se sobrescribe el método **beforeHandle**. el propósito de este método es configurar los encabezados de respuesta para permitir CORS antes de la solicitud sea manejada por el siguiente componente. Para ello, lo primero que se hace es obtener los encabezados de respuesta **responseHeaders** y, si no existen, se crean. Se obtienen los encabezados de solicitud **requestHeaders** para extraer los valores más relevantes como *Origen y AccessControlRequestHeaders*. Se configuran los métodos HTTP permitidos y demás encabezados como *AccesControlA-*

llowCredentials, etc. Hay que destacar que si el método de la solicitud es **OPTIONS (preflight request en CORS**, se detiene el procesamiento adicional **Filter.STOP**.

5.3.3. AuthenticationController.

Esta clase se define como un recurso del servidor encargado de la gestión de autenticación de usuarios. esta clase extiende *Server Resource* y brinda un método para el procesamiento de solicitudes POST con credenciales de usuario.

Librerías usadas.

- org.restlet.representation.Representation: Sirve para representar los datos de la solicitud y respuesta.
- org.restlet.representation.StringRepresentation: Sirve para trabajar con representaciones de cadenas.
- org.restlet.resource.Post: Sirve para definir el método que maneja las solicitudes POST.
- org.restlet.resource.ServerResource: Sirve para crear recursos del servidor.
- org.restlet.security.Verifier: Interfaz que sirve para verificar las credenciales.
- io.jsonwebtoken.Jwts: Sirve para trabajar con JWT (JSON Web Tokens).
- javax.crypto.SecretKey: Sirve para manejar claves secretas usadas en la firma del JWT.
- org.bson.Document: Sirve para trabajar con documentos BSON, usados en MongoDB.
- org.restlet.data.Status: Sirve para manejar los códigos de estado HTTP.

Listing 5.9: Clase AuthenticationController.

```
1  import org.restlet.representation.Representation;
2  import org.restlet.representation.StringRepresentation;
3  import org.restlet.resource.Post;
4  import org.restlet.resource.ServerResource;
5  import org.restlet.security.Verifier;
6  import io.jsonwebtoken.Jwts;
7  import java.util.Date;
8  import javax.crypto.SecretKey;
9  import org.bson.Document;
10 import org.restlet.data.Status;
11
12 public class AuthenticationController extends ServerResource {
13
14     // Usar la clave generada
15     private static final SecretKey SECRET_KEY = SecretKeyManager.getInstance().
           getSecretKey();
16     //clave_secreta_para_firmar_el_token_jwt
17     @Post("json")
```

```
18      public Representation authenticateUser(StringRepresentation representation) {
19          // Obtener el cuerpo JSON de la solicitud
20          String jsonBody = representation.getText();
21          Document userData = Document.parse(jsonBody);
22          String username = userData.getString("username");
23          String password = userData.getString("password");
24          System.out.println("user:" + username);
25          System.out.println("pass:" + password);
26
27          // Verificar las credenciales utilizando la clase VerificadorUsuarios
28          VerificadorUsuarios verifier = new VerificadorUsuarios();
29          int authenticationResult = verifier.verify(username, password.toCharArray
                ());
30
31          // Manejar el resultado de la autenticacion
32          if (authenticationResult == Verifier.RESULT_VALID) {
33              // Generar y retornar un token JWT
34              String jwtToken = generateJwtToken(username);
35              return new StringRepresentation(jwtToken);
36          } else {
37              // Retornar un mensaje de error en caso de credenciales invalidas
38              getResponse().setStatus(Status.CLIENT_ERROR_UNAUTHORIZED);
39              return new StringRepresentation ("Credenciales invalidas");
40          }
41      }
42
43      private String generateJwtToken(String username) {
44          // Obtener la fecha actual y sumarle 1 hora
45          Date expirationDate = new Date(System.currentTimeMillis() + 3600 * 1000);
                // 1 hora en milisegundos
46          // Generar una clave secreta con la que se firmara el token
47
48          // Construir el token JWT con el nombre de usuario como sujeto
49          String jwtToken = Jwts.builder()
50                  .claim("sub", username)
51                  .expiration(expirationDate)
52                  .signWith(SECRET_KEY)
53                  .compact();
54
55          return jwtToken;
56      }
57  }
```

A continuación se describe la lógica de la clase:

1. Autenticación del usuario:
   - el método denominado **authenticateUser** recibe una representación JSON con las credenciales del usuario.
   - El JSON se parsea para recoger las credenciales.
2. Verificación de credenciales:
   - Mediante la clase **VerificadorUsuarios** (que se explicará a continuación) se realiza un proceso para verificar las credenciales.
   - Según el procesamiento de las credenciales, se devuelve un token JWT o se devuelve un mensaje de error.

3. Generación de JWT:
   - Genera un token JWT mediante **Jwts.builder()**, configurando el nombre de usuario como sujeto y una fecha de expiración que puede ajustarse a elección, en este caso de una hora.
   - El token es firmado por una clave secreta que se obtiene de **SecretKeyManager** (que se explicará a continuación).
4. Respuesta a la Solicitud:
   - Si es exitosa, se retorna token el JWT.
   - Si falla, se retorna un mensaje de error con el estado HTTP 401 (unauthorized).

Esta estructura afianza que las solicitudes de autenticación se manejan de manera segura y eficiente, brindando tokens JWT a usuarios autenticados para su uso en futuras solicitudes.

5.3.4. VerificadorUsuarios.

Esta clase es una extensión de **MapVerifier** la cual tiene la función de verificar las credenciales mediante datos almacenados en la colección *UsersApi* explicada en la sección BBDD. Gestiona la autenticación mediante hash de contraseñas con Sal.

Librerías usadas.

- org.restlet.Request: Representa una solicitud HTTP.
- org.restlet.security.MapVerifier: Interfaz realiza la verificación de usuarios mediante un mapa de credenciales.
- org.restlet.security.Verifier:Sirve para verificar las credenciales.
- org.restlet.data.Method: Contiene los métodos HTTP soportados.
- org.bson.Document: Sirve para trabajar con documentos BSON, usados en MongoDB.
- java.security.MessageDigest: Genera hash de contraseñas.
- java.security.NoSuchAlgorithmException: Maneja excepciones al crear hashes.
- java.util.Base64: Codifica y decodifica datos en Base64.
- java.util.HashMap: Almacena pares clave-valor.
- java.util.List: Gestiona la lista de documentos.

A continuación se describe la lógica de la clase:

1. Inicialización.
    - El constructor VerificadorUsuarios es quien se encarga de inicializar la clase y cargar las credenciales desde MongoDB.
2. Carga de credenciales.
    - El método **cargarCredencialesdeMongo** conecta a la base de datos de MongoDB, obtiene todos los documentos de la colección **UserApi**, y extrae las credenciales.
    - Estas credenciales se almacenan en **getLocalSecrets** y las sales en **saltMap**.
3. Verificación de contraseña.
    - El método **verify** se sobrescribe para comparar la contraseña introducida con el hash almacenado en la base de datos.
    - Si el método es **OPTIONS** se omite esta verificación.
    - Si la verificación es correcta, devuelve **RESULT_VALID**; si no, **RESULT_INVALID**.
4. Generación de Hash de contraseña con sal.
    - El método **hashPasswordWithSalt** utiliza **MessageDigest** para la generación de un hash SHA-256 de la contraseña entrante con la sal correspondiente.
    - Se compara con la contraseña almacenada.

Esta clase afianza una autenticación robusta a través del uso de hash de contraseñas con sal, lo que protege las credenciales de usuario almacenadas en la base de datos.

5.3.5. SecretKeyManager.

Esta clase es un singleton encargado de generar y gestionar una clave secreta **SecretKey** para la firma de tokens JWT. Esta clave se genera automáticamente cada dos días para que se mantenga una seguridad robusta.

Librerías usadas.

- javax.crypto.SecretKey: Representa la clave utilizada para las operaciones criptográficas.
- javax.crypto.spec.SecretSpec: Especifica la clave secreta que permite la construcción de una clave a partir de una matriz de bytes.
- java.security.NoSuchAlgorithmException: Maneja excepciones al crear hashes.
- java.security.SecureRandom: Genera números aleatorios seguros para fines criptográficos.

- java.util.Date: Gestiona fechas y horas.
- java.util.Timer: Sirve para la programación de tareas que se ejecutan periódicamente.
- java.util.TimerTask: Representa una tarea que pueda ser programada para ejecución periódica.

A continuación se describe la lógica de la clase:

1. Inicialización de Singleton:
   - El constructor privado **SecretKeyManager** es el encargado de generar una clave secreta inicial y programa la regeneración de la clave cada dos días a través de un **timer**.
2. Generación de clave secreta.
   - El método **generateSecretKey** utiliza **secureRandom** para crear una clave de 256 bits para el algoritmo HMAC SHA-256.
3. Programación de regeneración de Clave:
   - El método **scheludeKeyGeneration** programa la regeneración de la clave en un tiempo en el que se programe mediante un **Timer y TimerTask**.
4. Métodos públicos:
   - **getInstance:** Devuelve la instancia única del **SecretKeyManager**.
   - **getSecretKey:** Devuelve la clave secreta actual.

Esta clase asegura que la clave utilizada para firmar tokens JWT sea segura y se actualice para minimizar riesgos en la seguridad.

5.3.6. JwtVerifierAuth.

Esta clase extiende **Restlet** y es utilizada para la autenticación de solicitudes HTTP mediante tokens JWT.

Librerías usadas.

- org.restlet.Context: Ofrece el contexto de la aplicación Restlet.
- org.restlet.Request:Representa una solicitud HTTP.
- org.restlet.Response: Representa una respuesta HTTP.
- org.restlet.Restlet:Representa una unidad de procesamiento de solicitudes y respuestas.

- org.restlet.data.Status: Sirve para manejar los códigos de estado HTTP.
- org.restlet.routing.Router: Permite el enrutamiento de solicitudes y respuestas a través del patrón de URI.

A continuación se describe la lógica de la clase:

1. El constructor de **JwtVerifierAuth** coge un objeto **Context** y un enrutador **Router** como parámetros. Este constructor inicializa la clase base **Restlet** con el contexto proporcionado por la clase **ServerWeb** y almacena el enrutador en un campo privado.
2. El método **handle** es anulado para manejar las solicitudes entrantes. En este método:
   - Se recoge el token JWT del encabezado de autorización de la solicitud.
   - Se comprueba si el token existe, comienza con el prefijo **Bearer**.
   - Si existe, se elimina el prefijo **Bearer** y se verifica su autenticidad mediante el objeto **JwtVerifier** Se explica a continuación.
   - Si todo esta correcto, se pasa la solicitud al enrutador para procesarla.
   - Si no es correcto, sé estable el estado de la respuesta en **401 Unauthorized**, denegando el acceso.

Esta clase actúa como un filtro de autenticación, recoge las solicitudes entrantes, verifica la validez del token JWT y permite el acceso a los recursos.

5.3.7. JwtVerifier.

Esta clase es la encargada de realizar la verificación de validez de los tokens JWT antes de permitir el acceso a los recursos.

Librerías usadas.

- io.jsonwebtoken.Claims: Sirve para la representación de las reclamaciones (claims) de JWT.
- io.jsonwebtoken.ExpiredJwtException: Excepción que se lanza si un JWT ha expirado.
- io.jsonwebtoken.Jws y io.jsonwebtoken.Jwts: Sirven para verificar y parsear tokens JWT.
- io.jsonwebtoken.MalformedJwtException: Excepción que se lanza si un JWT tiene un formato incorrecto.

- javax.crypto.SecretKey: Representa la clave secreta necesaria para verificar la firma JWT.
- org.restlet.Request:Representa una solicitud HTTP.
- org.restlet.data.Status: Sirve para manejar los códigos de estado HTTP.
- org.restlet.routing.Filter: Facilita el filtrado de solicitudes a recursos específicos.

A continuación se describe la lógica de la clase:

1. el metodo **verifytoken** realiza la verificación real del token JWT mediante la clave secret. Si es correcto, devuelve **true**. Si ha expirado o existe algún problema con la firma, se manejan las excepciones que correspondan. Si ocurre cualquier otro error, se devuelve **false**. Estos valores le sirven a la clase anterior **JwtVerifierAuth** para dar permiso o no a la solicitud.

Esta clase es la encargada de comprobar la veracidad del token y devolverle el resultado a **JwtVerifierAuth** para que dé el acceso a los recursos o no.

5.3.8. CreateNewUser.

Esta es una clase que se presenta como un recurso Restlet que proporciona endpoints para la creación y eliminación de usuarios en un sistema, además asegura que las contraseñas almacenadas sean seguras mediante el uso de salteado y hash.

Librerías usadas.

- org.restlet.representation.Representation: Sirve para representar los datos de la solicitud y respuesta.
- org.restlet.resource.Delete: Sirve para definir el método que maneja las solicitudes DELETE.
- org.restlet.resource.Post: Sirve para definir el método que maneja las solicitudes POST.
- org.restlet.resource.ServerResource: Sirve para crear recursos del servidor.
- com.mongodb.MongoException: Sirve para manejar excepciones relacionadas con MongoDB.
- org.bson.Document: Sirve para trabajar con documentos BSON, usados en MongoDB.
- org.restlet.representation.StringRepresentation: Sirve para trabajar con representaciones de cadenas.
- java.security.MessageDigest: Genera hash de contraseñas.

- java.security.NoSuchAlgorithmException: Maneja excepciones al crear hashes.
- java.security.SecureRandom: Genera números aleatorios seguros para fines criptográficos.
- java.util.Base64: Codifica y decodifica datos en Base64.

A continuación se describe la lógica de la clase:

1. El método **createUsers** gestiona las solicitudes POST para la creación de usuarios:
   - Se obtiene el cuerpo JSON.
   - Se parsea el JSON para obtener los datos.
   - Se extrae el nombre de usuario y la contraseña.
   - Se crea un salteado aleatorio y se calcula el hash de la contraseña mediante el método **hashPassword**.
   - Se genera un documento MongoDB con el nombre de usuario, contraseña hasheada y el salteado, y se guarda en la colección de usuarios mediante **MongoDBManager**, que se explicará en esta sección.
   - Se devuelve un texto indicando el éxito o fracaso de la operación.
2. El método **eliminarUsuario** gestiona las solicitudes DELETE para la eliminación de usuarios:
   - Se obtiene el nombre de usuario de la petición.
   - Mediante el uso de MongoDBManager se elimina el documento correspondiente.
   - Se devuelve un texto indicando el éxito o fracaso de la operación.

5.3.9. SearchForParameters.

Esta clase es un recurso de Restlet que brinda endpoints para la búsqueda de actividades según diferentes criterios, como fechas,etc. Utiliza consultas en una base de datos MongoDB y devuelve los resultados en formato JSON.

Librerías usadas.

- java.text.ParseException, java.text.SimpleDateFormat, java.util.Date: Su función consiste en formatear y manejar fechas.
- java.util.ArrayList, java.util.List, java.util.Collectors: Su función es manejar colecciones y operaciones de transmisión.
- org.bson.Document: Sirve para trabajar con documentos BSON, usados en MongoDB

- org.restlet.resource.ServerResource: Sirve para crear recursos del servidor.
- org.restlet.representation.Representation: Sirve para representar los datos de la solicitud y respuesta.
- org.restlet.representation.StringRepresentation: Sirve para trabajar con representaciones de cadenas.

A continuación se describe la lógica de la clase:

1. El método **hanleGetRequest** gestiona las solicitudes GET:
   - Se obtiene el parámetro **METODO** de la petición.
   - Según el valor del parámetro, se ejecutan las siguientes acciones:
     - *forDate*: Busca actividades por rango de fechas.
     - *forPlace:* Busca actividades con un lugar específico.
     - *forCategory:* Busca actividades que tengan una categoría específica.
     - *All:* Devuelve todas las actividades sin filtrar.
2. en cada tipo de búsqueda, se realizan a través de métodos privados como **buscarActividadesPorFecha**, **buscarActividadesPorCategoria**, **buscarActividadesPorLugar** o **buscarTodasActividades**.
3. Los resultados se transforman en formato JSON y se devuelven como una representación de cadena.

5.3.10. ManageActivities.

Esta clase es un recurso Restlet que provee endpoints para agregar, actualizar y eliminar actividades en una base de datos MongoDB. Los datos se recogen en formato JSON y se utilizan en **MOngoDBManager** para interactuar con la colección de actividades en la base de datos, Devolviendo el resultado de las operaciones en formato de representación de texto.

Librerías usadas.

- org.restlet.resource.Post, org.restlet.resource.Delete, org.restlet.resource.Put: Sirven para definir los métodos que manejan las solicitudes POST, PUT y DELETE.
- com.mongodb.MongoException: Sirve para manejar excepciones relacionadas con MongoDB.
- org.bson.Document: Sirve para trabajar con documentos BSON, usados en MongoDB.
- org.restlet.resource.ServerResource: Sirve para crear recursos del servidor.

- org.restlet.representation.Representation: Sirve para representar los datos de la solicitud y respuesta.
- org.restlet.representation.StringRepresentation: Sirve para trabajar con representaciones de cadenas.

A continuación se describe la lógica de la clase:

1. Se crea una instancia de **MongoDBManager** para realizar la interacción con la base de datos MongoDB, en concreto en la base de datos **base_prueba_v0** y la colección **Actividades**.
2. método **agregarNuevaActividad:**
   - Recoge el cuerpo JSON de la petición.
   - Parsea el JSON y obtiene los datos de la actividad nueva.
   - Agrega el documento de la nueva actividad en la colección de MongoDB.
   - Devuelve una representación de cadena donde se indica el éxito o fracaso de la operación.
3. método **actualizarActividadPorTitulo:**
   - Recoge el cuerpo JSON de la petición.
   - Parsea el JSON y obtiene los datos actualizados de la actividad.
   - Recoge el título de la actividad para actualizar los datos.
   - Actualiza el documento de la actividad en la colección de MongoDB.
   - Devuelve una representación de cadena donde se indica el éxito o fracaso de la operación.
4. método **eliminarActividadPorTitulo:**
   - Recoge el título de la actividad en los parámetros de la petición.
   - Elimina el documento de la actividad en la colección MongoDB.
   - Devuelve una representación de cadena donde se indica el éxito o fracaso de la operación.

5.3.11. MongoDBManager.

Esta clase sirve para manejar operaciones comunes en MongoDB, como conectarse a una base de datos, insertar, actualizar, eliminar y obtener documentos de una colección específica. Esta implementación se ha realizado apoyándose en [59].

Librerías usadas.

- com.mongodb.MongoClientSettings, com.mongodb.MongoCredential, com.mongodb.MongoE com.mongodb.ServerAddress: Ofrecen funcionalidades para configurar y manejar la conexión a MongoDB.
- com.mongodb.client.MongoClients, com.mongodb.client.MongoClient, com.mongodb.client.M com.mongodb.clients.MongoCollection: Ofrecen funcionalidades para interactuar con la base de datos y colecciones de MongoDB.
- com.mongodb.client.model.Filters, com.mongodb.client.model.Projections: Se utilizan para la creación de filtros y proyecciones en las consultas.
- com.mongodb.client.result.DeleteResult: Se gestiona el resultado de las operaciones de eliminación.
- org.bson.Document, org.bson.conversion.Bson, org.bson.types.ObjectId: Sirve para trabajar con documentos BSON, usados en MongoDB y sus conversiones.

A continuación se describe la lógica de la clase:

1. Se configuran las credenciales de usuario (**username** y **password**) y la dirección del servidor (**ServerAddress**). Además, se configura el cliente MongoDB (**MongoClientSettings**).
2. Se realiza una instancia del cliente MongoDB (**MongoClient**) mediante las configuraciones establecidas. Además, se elige la base de datos (**database**) y la colección (**collection**).
3. Asegura que la conexión a MongoDB se cierre adecuadamente mediante **closeConnection**,
4. Operaciones CRUD:
   - Inserción: A través del método **insertDocument** para almacenar nuevos documentos en la colección.
   - Actualización: Mediante el método **updateDocumentActivities** se actualizan los documentos existentes a través del título de la actividad.
   - Eliminación: Mediante **deleteDocuments deleteDocumentUsers** para eliminar los documentos basados en el título o el nombre de usuario. Para ello se encuentra el criterio específico y luego se elimina el documento a través de su ID.
   - Obtención: Con **getAllDocuments** se obtienen todos los documentos de una colección y mediante **getDocumentById** el ID específico de un documento.

5.3.12. EmailResource.

Esta clase es otro recurso del servidor, el cual utiliza el framework de Restlet para la gestión de peticiones HTTP POST, las cuales contienen un mensaje en formato JSON

en el que se realiza una solicitud para enviar un correo electrónico. La implementación de esta clase está basada en [60].

Librerías usadas.

- org.restlet.resource.ServerResource: Sirve para crear recursos del servidor.
- org.restlet.representation.Representation: Sirve para representar los datos de la solicitud y respuesta.
- org.restlet.representation.StringRepresentation: Sirve para trabajar con representaciones de cadenas.
- javax.mail.*: Base para la gestión de correos electrónicos.
- java.mail.internet: Recurso específico para la manipulación y creación de mensajes de correo con el formato MIME.
- java.util.Properties: Configuración de cliente de correo. import org.json.JSONObject: Permite manipular objetos JSON.
- java.io.IOException: Gestiona excepciones de entrada/salida.
- java.io.FileInputStream; Clase para manejar archivos de entrada.

A continuación se describe la lógica de la clase:

1. Carga de Configuraciones:
   - Antes del envío del correo, esta clase carga las configuraciones necesarias desde un archivo de propiedades a través del método **loadTokenPorpertiesFile**.
2. Manejo de petición POST:
   - A través del método manejador de peticiones **sendEmail**, se recoge el cuerpo de la solicitud y se convierte en un objeto JSON. De este objeto se extrae el mensaje y se envía el correo.
3. Configuración y Envío de correo:
   - A través del método **sendEmail** se configuran las propiedades del cliente de correo, se crea una sesión de correo con autenticación a través del correo remitente y la contraseña. Se crea un correo con el asunto y el cuerpo, enviándolo mediante **Transport.send**.
4. Gestión de Errores:
   - Si se produce un error al enviar el correo o ocurre al cargar las configuraciones, se captura la excepción y se imprime el stack trace.

5.3.13. TwitterClient.

Esta clase es un recurso del servidor que gestiona peticiones HTTP POST. Su principal objetivo se centra en recibir una solicitud con un mensaje en formato JSON, leer las credenciales OAuth2 desde un archivo de propiedades y enviar un tweet a través de la API de Twitter. Esta implementación se ha basado en [61], [62].

Librerías usadas.

- org.restlet.resource.ServerResource: Sirve para crear recursos del servidor.
- org.restlet.representation.Representation: Sirve para representar los datos de la solicitud y respuesta.
- org.restlet.representation.StringRepresentation: Sirve para trabajar con representaciones de cadenas.
- com.github.scribejava.core.model.OAuth2AccessToken: Librería para el token de acceso OAuth2.
- java.util.Properties: Configuración de cliente de correo. import org.json.JSONObject: Permite manipular objetos JSON.
- java.io.IOException: Gestiona excepciones de entrada/salida.
- java.io.FileInputStream; Clase para manejar archivos de entrada.
- com.twitter.clientlib.ApiClientCallback: Se trata de una interfaz para gestionar callbacks de la API del cliente.
- com.twitter.clientlib.TwitterCredentialsOAuth2: Es necesaria para las credenciales de OAuth2 de tweets.
- com.twitter.clientlib.model.TweetCreateRequest: Es un modelo para la petición de creación de tweets.
- com.twitter.clientlib.model.TweetCreateResponse: Es un modelo para la respuesta de la creación de tweets.
- com.twitter.clientlib.api.TwitterApi: Clase primaria para la interacción con la API de X.
- com.twitter.clientlib.ApiException: Controla la excepción específica para la API de X.

A continuación se describe la lógica de la clase:

1. Carga de Configuraciones:
   - Antes del envío del tweet, esta clase carga las configuraciones necesarias desde un archivo de propiedades a través del método **loadTokenPorpertiesFile**.
2. Manejo de petición POST:

- A través del método **sendTweet**, se recoge el cuerpo de la solicitud y se convierte en un objeto JSON. De este objeto se extrae el mensaje y se envía el tweet.

3. Configuración de la API de Twitter:

   - A través del método **sendTweet**, se crea una instancia de *TwitterApi*, si no lo está y se configuran de las credenciales. Además, se añade un callback para gestionar la actualización de tokens.

4. Envío de Tweets:

   - Se realiza una solicitud de creación de tweets a través de **TweetCreateRequest**. Se envía el tweet utilizando **apiInstance.tweets() .create(tweetCreateRequest.execute**

5. Gestión de Errores:

   - Si se produce un error al enviar el correo u ocurre al cargar las configuraciones , se captura la excepción y se imprime el stack trace.

6. Clase MaintainToken.

   - La clase MaintainToken implementa **ApiClientCallBack** y se utiliza para gestionar la actualización de tokens de OAuth2. Cuando este se actualiza, se guarda en un archivo de propiedades.

5.4. Frontend.

El frontend de este proyecto tiene como propósito brindar unas interfaces gráficas web intuitivas y funcionales tanto para los alumnos como para los administradores de la escuela. Las páginas están desarrolladas en Angular,una plataforma de desarrollo web que posibilita la creación de aplicaciones SPA (Single Page Application) robustas y mantenibles. La justificación detallada del uso de este software se encuentra en el **capítulo 5, sección 5.1**.

Esta sección se divide en dos interfaces. La primera que se desarrolla es la interfaz de usuarios, la cual permite a los alumnos consultar actividades a través de un filtro de búsqueda, además de un calendario donde se pueden ver de forma gráfica las actividades agendadas a lo largo del año. La segunda página web está dedicada a los administradores de la aplicación, la cual brinda una interfaz intuitiva para la gestión de las actividades (creación, modificación, eliminación, exportación a plantillas, automatización de envío de correo y post en X) como la gestión de los mismos usuarios de la página (creación y eliminación).

A continuación se especifican los principales componentes, servicios, rutas e interfaces y cómo interactúan entre sí para conformar una estructura coherente y eficiente,

además de explicar cómo se ha realizado la integración con el backend. Por ello, se proporciona una visión de las decisiones tomadas en el diseño **capítulo 4.** El código de esta implantación se encuentra en [63], [64].

5.4.1. Preparación del Entorno.

Para la implementación del frontend en Angular, se necesita configurar un entorno de desarrollo apropiado. Por ello, a continuación se especifican los pasos a seguir para adecuar el entorno.

Instalación de Node.js y npm.

Node.js es un entorno de ejecución de JavaScript el cual permite la ejecución de scripts del servidor. Mientras que npm es un administrador de paquetes de Node.js que ayuda en la instalación y administración de módulos y dependencias de JavaScript.

1. Se descarga e instala Node.js desde [65]
2. Desde una terminal de comando, se verifican mediante los comandos que se muestran a continuación que Node.js y npm se han instalado correctamente.

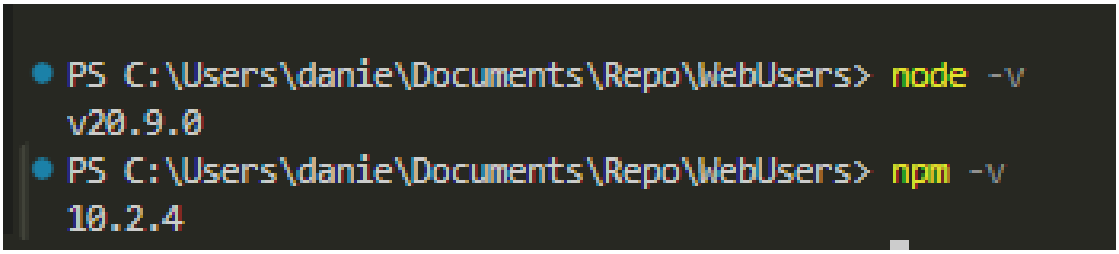


Figura 5.6: Verificación de la instalación de Node.js y npm.

Instalación de Angular CLI.

Angular CLI (Command Line Interface) es una herramienta de linea de comando que ayuda en la creación, generación y configuración de aplicaciones Angular.

1. Se debe ejecutar el siguiente comando para instalar Angular CLI en el sistema utilizado:

Listing 5.10: Comando de isntalacion Angular.

```
1        npm install -g @angular/cli
```

2. Desde una terminal de comando, se verifican mediante el comando que se muestra a continuación que Angular CLI se ha instalado correctamente.

Figura 5.7: Verificación de la instalación de AngularCLI.

5.4.2. Página de actividades de usuarios.

Componente search.

Este componente permite a los usuarios realizar búsquedas de actividades a través de diferentes filtros: lugar, fecha y categoría.

El componente search.component.ts es el que aloja la lógica del formulario para recoger los criterios de búsqueda del usuario. A su vez, se comunica con el servicio **apirestlet.service.ts** para utilizar los métodos que envían las peticiones y gestionan las respuestas con el backend. Las actividades se almacenan en la propiedad **actividades**.

La plantilla HTML y el archivo CSS asociados a este componente definen la estructura visual de esta parte de la web, donde los usuarios pueden introducir sus criterios y se muestran los resultados.

Componente activities.

El componente **activities** es quien se encarga de mostrar los detalles de las actividades. Mediante la directiva *@Input* se recibe la estructura del componente padre de cada

actividad, lo que permite ser reutilizable para mostrar diferentes actividades en base de los resultados de la búsqueda.

La plantilla HTML asociada es utilizada para definir cómo se presenta la información de cada una de las actividades en la web. Los componentes que utilizan este, son los encargados de mostrar esta información.

Componente calendar.

Este componente expone un calendario interactivo el cual muestra las actividades agendadas, a través de una navegación entre meses y cuatrimestres. A partir de la biblioteca *FullCalendar* [66] se comunica con el servicio *apirestlet* para obtener eventos y mostrarlos en el calendario. *calendarOptions* define la vista del calendario como un mes *dayGridMonth*, indica los plugins que se utilizan y define el gestor para el evento de clic en una fecha. A partir del método *buscarActividades* se comunica con el servicio *apirestlet* para obtener todas las actividades existentes en la aplicación y las parsea en eventos que *FullCalendar* interpreta de manera correcta. Esta implementación se ha basado en [67].

Servicio apirestlet.

Este servicio es indispensable para la aplicación Angular ya que gestiona las peticiones HTTP hacia el backend. Proporciona métodos para interactuar con las actividades, lo que permite buscar, filtrar y obtener actividades desde el servidor.

Detalles de la implementación:

- Método getAllActividades: hace una petición GET al endpoint del backend y devuelve todas las actividades.
- Método getActividadesPorCategoria: Este método, junto con *getActividades-PorFecha* y *getActividadesPorLugar* realizan peticiones GET HTTP a los endpoints del backend para obtener la lista de actividades.

Este servicio es el encargado de la comunicación entre el frontend y backend, habilitando a los componentes de Angular la función de obtener y manipular datos de las actividades de una forma eficiente y organizada.

Rutas.

Las rutas en Angular habilitan la navegación entre componentes de la aplicación. Esta configuración define cómo se mapean las URL a los componentes correspondientes.

El archivo app.routes.ts, aunque está vacío, es el punto central para la configuración de las rutas.

El archivo app-routing.module.ts. Es el módulo de enrutamiento de la aplicación. es donde están definidas las rutas y los componentes que se asocian a cada una.

- */search:* Esta ruta lleva al componente de búsqueda **SearchComponent**.
- */calendar:* Esta ruta lleva al componente de calendario **CalendarComponent**.

Para que funcione correctamente, el AppRountingModule tiene que estar importado en el módulo principal de la aplicación **AppModule**. Esto garantiza que las rutas estén disponibles cuando la aplicación se inicie.

Configuración del módulo principal.

La configuración del módulo principal de Angular se encuentra en el archivo **app.module.ts**. Este módulo organiza y centraliza los componentes, servicios y otros módulos que se existen en la página. Los elementos de este módulo son los siguientes:

Módulos:

- BrowseModule: Se necesita para ejecutar la aplicación en un navegador.
- FormsModule: Habilita utilizar formularios en Angular.
- HttpClientModule: Habilita los elementos necesarios para realizar solicitudes HTTP a servicios REST.
- AppRoutingModule: Habilita la configuración de las rutas de navegación en la aplicación entre los componentes.

5.4.3. Página de actividades de Administración.

Componente activities.

Al igual que en la página web de los usuarios. El componente **activities** es quien se encarga de mostrar los detalles de las actividades. mediante la directiva *@Input* se recibe la estructura del componente padre de cada actividad, lo que permite ser reutilizable para mostrar diferentes actividades en base de los resultados de la búsqueda.

La plantilla HTML asociada es utilizada para definir cómo se presenta la información de cada una de las actividades en la web. Los componentes que utilizan este, son los encargados de mostrar esta información.

Componente home.

Este componente actúa como un contenedor y punto central de navegación para las demás funcionalidades de la aplicación de administración, ofreciendo un medio para

acceder a las diferentes secciones y cerrar sesión de manera sencilla. Una vez que el administrador inicie sesión, puede acceder a la gestión de usuarios, actividades y cerrar sesión. Además, una vez se accede a uno de los dos recursos anteriores, este componente se mantiene en aquellos recursos para poder cambiar fácilmente entre los mismos, a la vez de mantener la funcionalidad de cerrar sesión.

Componente login.

Este componente es el encargado de la gestión del proceso de autenticación de los usuarios administradores. Es primordial para que se garantice que solo los usuarios autorizados puedan acceder a las funcionalidades administrativas de la aplicación. Mediante el servicio de autenticación, valida las credenciales y maneja la navegación tras un inicio de sesión exitoso.

Detalles de la implementación:

- Método login: Su función es enviar el formulario de inicio de sesión. A través de *AuthService* se realiza la autenticación con las credenciales obtenidas. Si esta es exitosa, se redirige a la página principal **/home**. Si existe un error, no deja acceder a los recursos de la página.

Componente users.

El componente **users** tiene la función de gestionar los usuarios en la interfaz de administración, brindando la oportunidad de crear y eliminar usuarios a través de interacciones con el backend.

Detalles de la implementación:

- Método CreateUser(): A partir de un formulario, recoge el nombre de usuario y su contraseña para crear una nueva entidad dentro de la aplicación, la cual tendrá acceso a la página web de Administración. A través del **ApiService** se envían los nuevos datos de usuario al backend.
- Método DeleteUser(): A partir de un formulario, recoge el nombre de usuario para eliminar una entidad existente dentro de la aplicación, la cual ya no tendrá acceso a la página web de Administración. A través del **ApiService** se envían los datos de usuario al backend.

Componente manageActivities.

El componente **manage Activities** habilita una interfaz que gestiona las actividades de la aplicación. Permitiendo crear,modificar,eliminar, filtrar y exportar actividades.

Habilita una interfaz amigable para los usuarios, lo que asegura una correcta interacción con el backend mediante *ApiService*.

Detalles de la implementación:

- En el método **nfOnInit**, se llama a *buscarTodasAcividades* al inicializar el componente para obtener todas las actividades existentes. El método buscarTodasAcividades utiliza *ApiService* para obtener dicha lista.
- La gestión principal de las actividades se realiza mediante los siguientes métodos:
  - **CreateActivities()**: Almacena los datos del formulario y llama a *postActivities* de *ApiService* para el envío de las nuevas actividades al servidor.
  - **ModifyActivities()**: Almacena los datos del formulario y llama a *putActivities* de *ApiService* para el envío de los nuevos datos de una actividad existente.
  - **DeleteActivities()**: Almacena el título de la actividad a eliminar y llama a *deleteActivities* de *ApiService*. Contiene un método llamado *confirmDelete* para pedir una confirmación previa a la eliminación de cualquier actividad.
- La exportación o automatización del envío de actividades se realiza mediante lo siguientes métodos:
  - Mediante el método **exportActivities** que llama al método *applyFilters* para recoger las actividades actuales en la tabla y exportar a dos tipos de plantillas estas actividades a través de los métodos *exportEmail* o *exportTwitter*.
  - Mediante el método **sendActivities** que llama al método *applyFilters* para recoger las actividades actuales en la tabla y redirige esta información al componente *template-preview* para la preparación del envío.
- La clase **FiltroActividadesPipe** es la encargada de aplicar los múltiples filtros disponibles de forma dinámica a la lista de actividades que se muestran en la tabla y se formatean en las diferentes plantillas para exportar o enviar, según requiera el administrador.

Componente template-preview.

Este es un componente con la función de visualizar y enviar plantillas en la aplicación de Angular. Gestiona la lógica de obtención de parámetros de consulta y se integra con *ApiService* para el envío de los mismos.

Detalles de la implementación:

- Método ngOnInit(): Se suscribe a los parámetros de consulta de la URL a través de *ActivateRoute*. Se comprueba si los parámetros *template* y *filename* no son nulos y, si es así, realiza la asignación de los valores a la variables locales del componente.
- Método sentTemplate(): A través de la variable fileName, elige el destino de envío (Correo o X).

Servicio apirestlet.

Este servicio es parecido al descrito en la web de usuarios, ya que gestiona las peticiones HTTP hacia el backend. Proporciona métodos para gestionar las actividades y usuarios desde el servidor además de realizar el envío de plantillas a la plataforma X o por correo electrónico.

Detalles de la implementación:

- Método getAllActividades(): A través de una petición GET, obtiene todas las actividades existentes.
- Método postActivities(): A través de una petición POST, solicita crear una nueva actividad.
- Método putActivities(): A través de una petición PUT, solicita modificar una actividad.
- Método deleteActivities(): A través de una petición DELETE, solicita eliminar una actividad.
- Método postUser(): A través de una petición POST solicita crear una nuevo usuario.
- Método deleteUser(): A través de una petición DELTE solicita eliminar un usuario.
- Método sendToTwitter(): A través de una petición POST solicita enviar una plantilla a la plataforma X.
- Método sendToEmail(): A través de una petición POST solicita enviar una plantilla por correo electrónico.

Servicio auth.

Este servicio es el encargado de la autenticación de los usuarios. Contiene métodos para el inicio y cierre de sesión, además de obtener el token de autenticación de usuario. Interactúa con el backend para la verificación e credenciales y el almacenamiento del token de autenticación en local del navegador.

Detalles de la implementación:

- Método login(): A través de una solicitud POST, envía las credenciales del usuario. Decodifica la respuesta (esperando un token) y lo almacena en el almacenamiento local.
- Método logout(): Elimina el token de autenticación local.
- Método getToken(): Recupera el token que se aloja en el almacenamiento local.

Guarda auth.

Este es un componente de Angular que habilita el desarrollo de la interfaz **CanActivate**, esto permite proteger las rutas. A través de la verificación, brinda al usuario el acceso a ciertas rutas. Si el usuario no está autenticado, redirige a la página de inicio de sesión.

Detalles de la implementación:

- Método canActivatwe(): Verifica si el usuario tiene un token a través del método *getToken* del servicio AuthService. Si este existe (esto quiere decir que el usuario está autenticado) devuelve un valor *true* que permite el acceso a la ruta. Si no, retorna un valor *false*, denegando el acceso a la ruta.

Rutas.

Las rutas en Angular habilitan la navegación entre componentes de la aplicación. Esta configuración define cómo se mapean las URL a los componentes correspondientes.

El archivo app.routes.ts, aunque está vacío, es el punto central para la configuración de las rutas.

El archivo app-routing.module.ts es el módulo de enrutamiento de la aplicación. es donde están definidas las rutas y los componentes que se asocian a cada una. Este componente utiliza el guarda **auth** para restringir el acceso a ciertas rutas si el usuario no está autenticado.

- */login:* Esta ruta lleva al componente de inicio de sesión **login**.
- */home:* Esta ruta lleva al componente donde se unifican los demás, denominado **home**. Está protegido por AuthGuard.
- */user:* Esta ruta lleva al componente donde se gestionan los usuarios, denominado **user**. Está protegido por AuthGuard.
- */manageActivities:* Esta ruta lleva al componente donde se gestionan las actividades, denominado **manageActivities**. Está protegido por AuthGuard.

- */template-preview:* Esta ruta lleva al componente donde se gestionan las plantillas, denominado **template-preview**. Está protegido por AuthGuard.

Para que funcione correctamente, el AppRountingModule tiene que estar importado en el módulo principal de la aplicación **AppModule**. Esto garantiza que las rutas estén disponibles cuando la aplicación se inicie.

Configuración del módulo principal.

La configuración del módulo principal de Angular se encuentra en el archivo **app.module.ts**. Este módulo organiza y centraliza los componentes, servicios y otros módulos que existen en la página. Los elementos de este módulo son los siguientes:

Módulos:

- BrowseModule: Se necesita para ejecutar la aplicación en un navegador.
- FormsModule: Habilita utilizar formularios en Angular.
- HttpClientModule: Habilita los elementos necesarios para realizar solicitudes HTTP a servicios REST.
- AppRoutingModule: Habilita la configuración de las rutas de navegación en la aplicación entre los componentes.
- MatPaginatorModule: . Es parte de Angular Material y habilita una interfaz de paginación para listas de datos.
- BrowserAnimationsModule: Proporciona soporte para animaciones en la aplicación. Es necesario para utilizar las animaciones que existen en Angular Material.

Capítulo 6

Resultados de la implementación y prueba de funcionamiento.

En este capítulo, se muestra el resultado del desarrollo que se ha realizado durante el proceso de implementación de la aplicación web y un conjunto de pruebas que demuestran el correcto funcionamiento de la misma.

Para ello, se van a mostrar diferentes capturas del aspecto de las webs, junto con capturas de la comunicación entre las mismas y la API, donde se demuestra que todas las funcionalidades desarrolladas funcionan correctamente. Cabe destacar que el proyecto aún se encuentra en fase de desarrollo, aunque estén completados todos los objetivos propuestos en este proyecto. Porque para pasar a las siguientes fases, falta la aceptación de los organismos responsables a los que se destina esta aplicación web. Si se aceptase, mediante unos cambios mínimos, el proyecto estaría preparado para pasar a las siguientes fases de una manera ágil y rápida. Por ello la dirección donde se aloja la API es en *https://localhost:8183* y las webs se alojan en *http://localhost:4200/* durante todo este capítulo.

6.1. Web de usuarios.

Esta sección se centra en la web de usuarios, donde se va a mostrar cómo ha quedado su aspecto y todas las funcionalidades que están asociadas a ella. En la figura 6.1 vemos el componente principal de la web, que contiene sus dos componentes principales: *Search* el cual se utiliza para la búsqueda de actividades a través de ciertos criterios, y el componente *Calendar* en el cual se muestran las actividades agendadas en los días correspondientes de cada mes del año actual.

A continuación, se describe en detalle el componente *Search*. En la figura 6.2 se muestra el aspecto inicial de este componente, el cual esta formado por varios formularios donde el cliente (alumno) podrá realizar unas búsqueda a través de 3 tipos de criterios:

Figura 6.1: Home de la web de usuarios.

fechas, lugar y categoría de las actividades.

Figura 6.2: Componente Search de la web de usuarios.

Ya que la comunicación en esta aplicación está cifrada mediante TLS (Transport Layer Security), el seguimiento de la comunicación entre los elementos de la aplicación se dificulta si se utiliza una herramienta como WireShark, por ello, a continuación en la figura 6.3 se muestra mediante la consola del navegador cómo la web ha obtenido mediante una búsqueda de actividades bajo el criterio de *categoría* las actividades correspondientes.

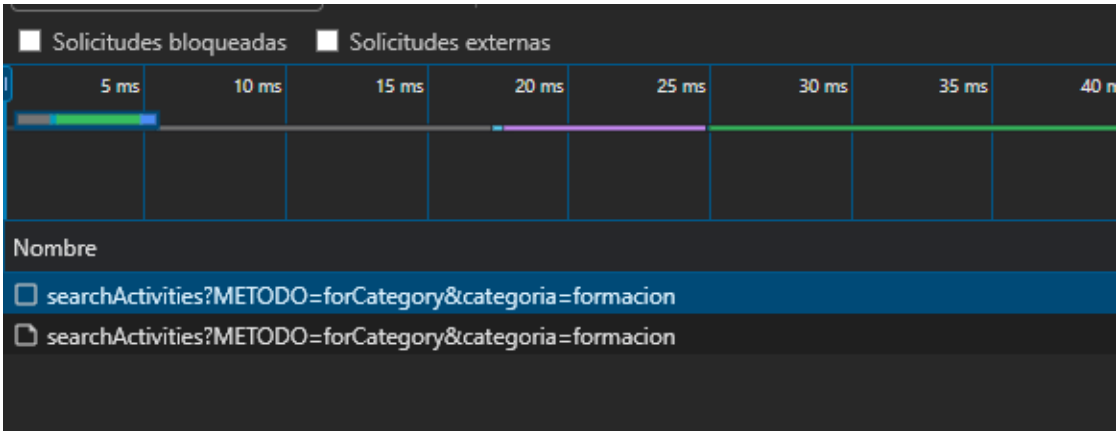


Figura 6.3: Comunicación entre la web de usuarios y la API.

En la secuencia de paquetes de la figura 6.3 se ven dos paquetes. El primero es una solicitud que se envía antes de cualquier petición HTTP, siendo parte del proceso de CORS (Cross-Origin Resource Sharing) en aplicaciones web. Esta solicitud de verificación se asegura si el servidor permite la operación deseada desde el dominio origen, mediante la comprobación de los permisos de métodos HTTP y encabezados. Esto se muestra en la figura 6.4 donde se observan cómo mediante los encabezados se le pregunta a la API qué tipo de solicitudes acepta, dominios aceptados para realizar peticiones,etc.

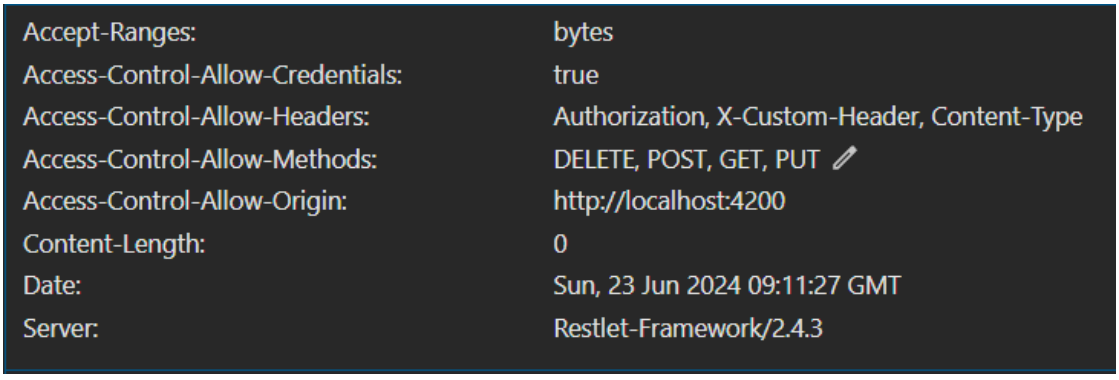


Figura 6.4: Headers de la petición OPTIONS.

En el segundo paquete, que se muestra en la figura 6.5 se ven los encabezados de la

petición GET donde la API devuelve la información necesaria para mostrarla en la web, que en este caso son las actividades que coinciden con el criterio de búsqueda utilizado.

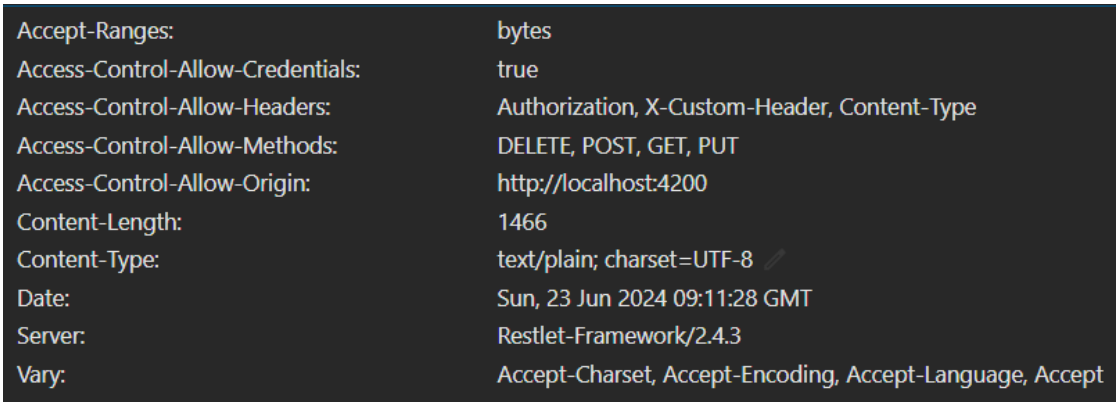


Figura 6.5: Headers de la petición GET.

Esta solicitud hace que se muestren en la web las actividades deseadas, tal y como aparece en la figura 6.6 donde se observan las actividades encontradas según el criterio de búsqueda utilizado en este ejemplo.

A continuación, se muestra la segunda funcionalidad de la web de usuarios, el calendario. La vista del calendario se muestra en la figura 6.7 donde se observa que las actividades creadas para realizar pruebas de este tipo están agendadas en sus días correspondientes mostrando el título de cada actividad y la hora a la que comienza.

Esta petición tiene la misma lógica que la anterior que se ha descrito, por ello lo único a destacar es que el proceso que recoge todas las actividades se inicia cada vez que se acceda a la web de usuarios o si se recarga la página. Este hecho se puede observar en la figura 6.8 donde se ven en el diagrama de secuencia, el proceso que inicia la web y justo después la petición a la API para obtener todas las actividades.

6.2. Web de administración.

Esta sección se centra en la web de administración, donde se va a mostrar cómo ha quedado su aspecto y todas las funcionalidades que están asociadas a ella. En la figura 6.9 vemos el componente que al que se accede hasta que no se realiza una autenticación exitosa y por lo tanto el componente inicial de la web, el componente *login*.

Si se intenta acceder de alguna u otra manera de que no sea con unas credenciales validas, se origina el mensaje que se muestra en la figura 6.10 donde indica que no tiene autorización para acceder a los recursos de la pagina.

Si la autenticación es exitosa, el componente *login* redirige al componente *home* el cual alberga los demás componentes que contienen las funcionalidades principales de la

Figura 6.6: Vista de la web de usuarios tras realizar una búsqueda de actividades.

aplicación. en la figura 6.11 se muestra el aspecto de dicho componente.

Respecto a las peticiones realizadas en la autenticación, se muestra en la figura 6.13 cómo se recibe la solicitud POST que contiene en el cuerpo de la misma las credenciales correctas, una respuesta con código 200 (autenticación satisfactoria) y en el cuerpo el token JWT que se utilizará para mantener la sesión activa y poder acceder a los demás recursos de la página.

A continuación se expone cómo se estructura el componente *users*, el más sencillo de la página web, el cual se compone de dos formularios: uno para la creación de usuarios y otro para la eliminación del mismo. Se ha de tener en cuenta que, aunque a simple vista solo son un par de formularios, la lógica que existe en la API y a la que se accede a través de este componente es esencial, que ya se ha descrito en la sección **5.3 Backend** del capítulo **5. Implementación de la aplicación**. Debido a la similitud que existe en la comunicación y realización de peticiones a la API, no se especifica, como en otras ocasiones en este capítulo, por qué se explica a continuación con el siguiente componente.

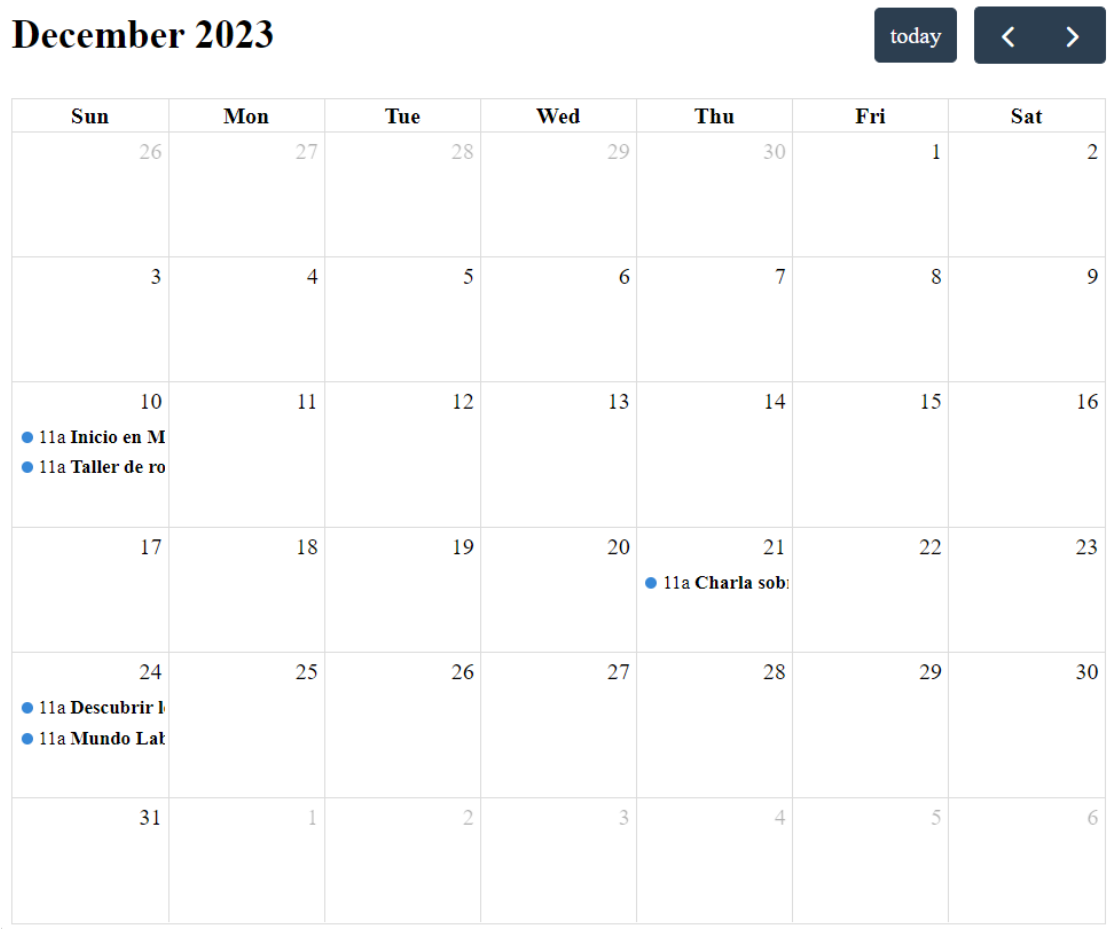


Figura 6.7: Vista del calendario con actividades de ejemplo agendadas.

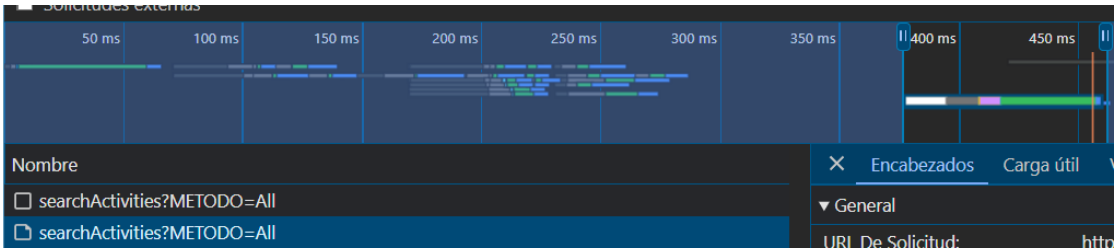


Figura 6.8: Peticiones en la comunicación para recoger todas las actividades.

Para finalizar, se muestra el componente *manageActivities*, que está formado de varias herramientas para una gestión eficiente y ágil sobre las actividades. Lo primero que se ve es un botón llamado *Crear Actividades* que si se pincha en él, se despliega un formulario que contiene todos los parámetros necesarios para crear una nueva actividad en la base

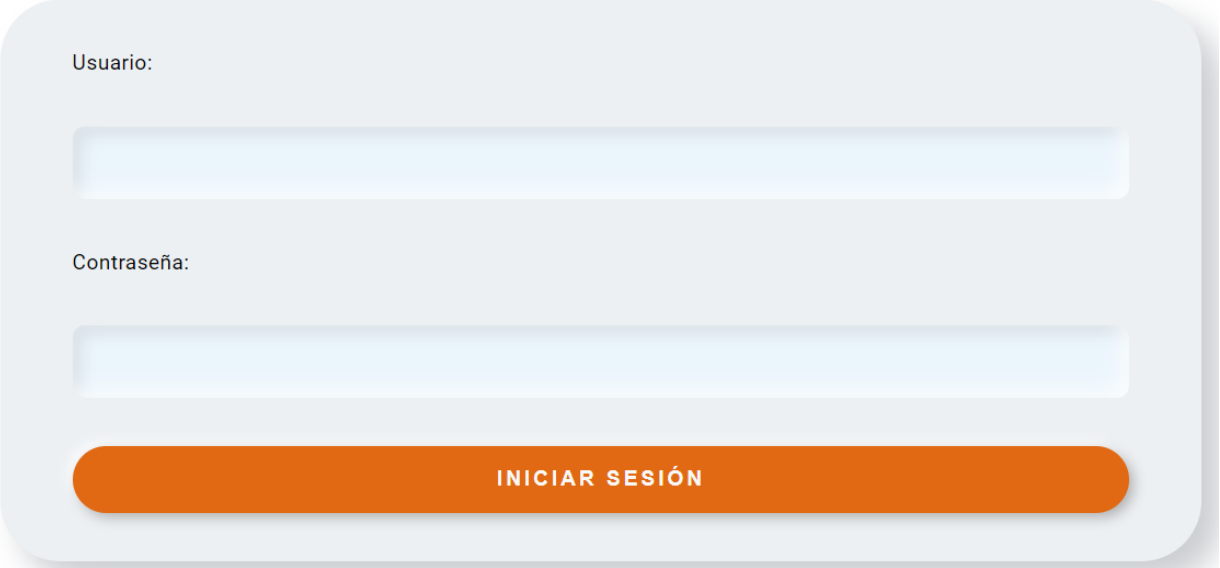


Figura 6.9: Aspecto del login de la web de administración.

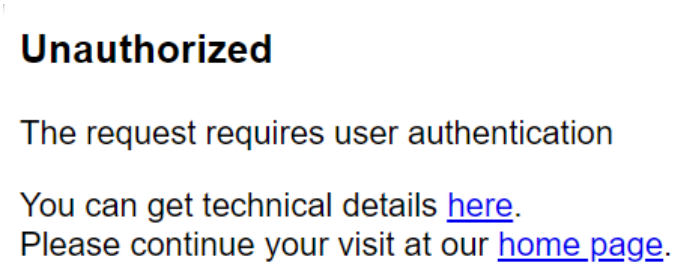


Figura 6.10: Mensaje de denegación al acceso a los recursos con credenciales invalidas o inexistentes.

de datos de MongoDB. Esto se muestra en la figura 6.14, no se ve la foto entera del formulario ya que es demasiado largo y no se puede capturar en una imagen.

Figura 6.11: Vista del componente home.

Figura 6.12: Petición con autenticación correcta.

Uno de los elementos principales de este componente es la tabla dinámica con filtros, con la opción de exportar a una plantilla las actividades que existan actuales en la tabla y la posibilidad de las mismas a través de correo o de X (twitter), que se muestra en la figura 6.15. Como en la figura 6.14 no se puede mostrar por completo la tabla, ya que sus dimensiones hacen que no se puedan ver de una forma legible todos sus detalles, por ello se explican por partes.

Figura 6.13: Componente users de la pagina de Administración.

Si se posa el ratón sobre algunas de las actividades, esta se sombrea y si se da clic, aparece un desplegable a partir del botón *Modificar Actividades* con los parámetros contenidos en dicha actividad para poder modificar fácilmente si se desea alguno de ellos. A su vez, abajo aparecen unos botones para confirmar la modificación o la eliminación de la actividad, según se requiera. Los botones que se describen se muestran en la figura 6.16.

Si se quiere eliminar una actividad, existe una doble confirmación para que no se borre de manera accidental y perjudique a la gestión de las actividades a través de la

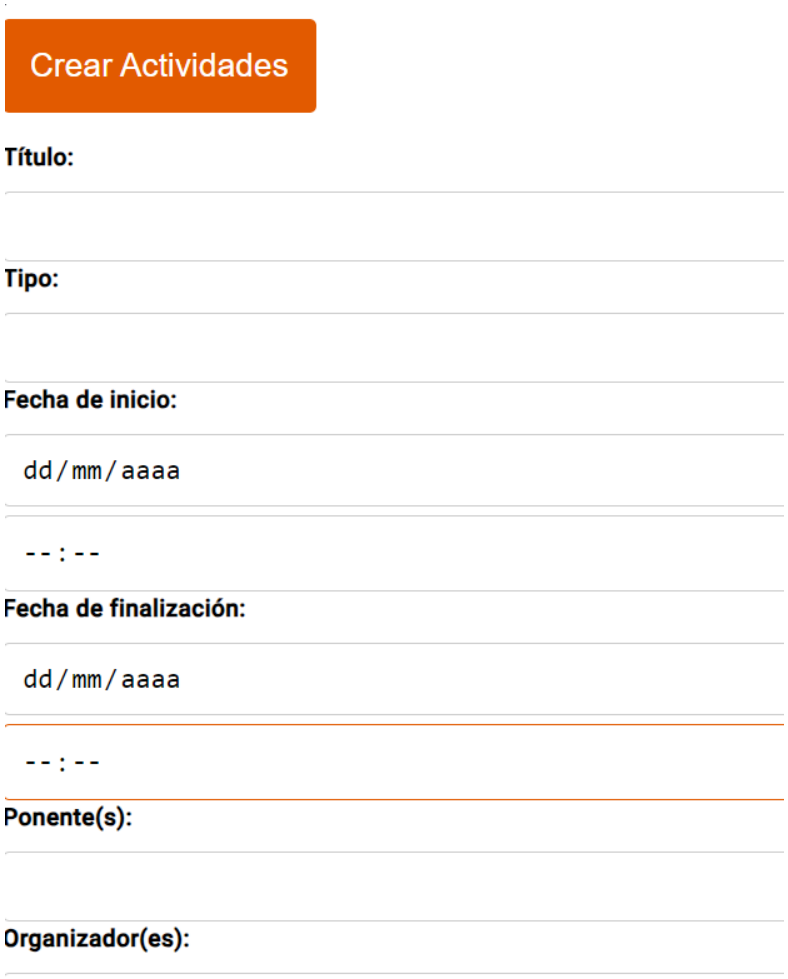


Figura 6.14: formulario para la creación de actividades.

aplicación. Esta confirmación se muestra en la figura 6.17.

Por último, se describen los botones de exportar y enviar. Como se muestra en la figura 6.18 vemos que aparecen ambos botones con un desplegable a su izquierda, que permite elegir si se quiere exportar una plantilla preparada para X o para correo.

Título	Tipo	Fecha de Inicio	Fecha de Finalización	Ponente(s)	Organizador(es)	Lugar
Charla sobre 6G	charla	2023-12-21T10:00:00Z	2023-12-21T11:00:00Z	Pedro Adamuz	UGR, dpto.Señales	0.1
Mundo Laboral después de la carrera	conferencia	2023-12-24T10:00:00Z	2023-12-24T11:00:00Z	Pablo Moreno	dpto.recursos humanos	0.2

Figura 6.15: Parte de la tabla de actividades.

Si se pincha en el botón de enviar, esto redirige a otro componente llamado *template-preview* el cual muestra una vista previa del formato del mensaje que se enviará a la plataforma elegida previamente. Esto hace que se pueda modificar el mensaje a enviar si se requiere por parte del administrador.

Las pruebas de funcionamiento en este componente se pueden simplificar en una única prueba, la cual justifica que las demás funciones están implementadas correctamente ya que la comunicación con la API es igual aunque el resultado en cada una de ellas es diferente. Esto se hace para evitar figuras repetitivas que no aportan información útil a la presente memoria. Por ello, se ha realizado la modificación en uno de los parámetros de la actividad para ver la interacción que existe. En esta ocasión, se manda una petición PUT a la API con la nueva información en los parámetros de la actividad a modificar, como los parámetros que no se cambiaron para que la API sobrescriba en la base de datos la nueva información sobre la antigua. Esto se muestra en la figura 6.19.

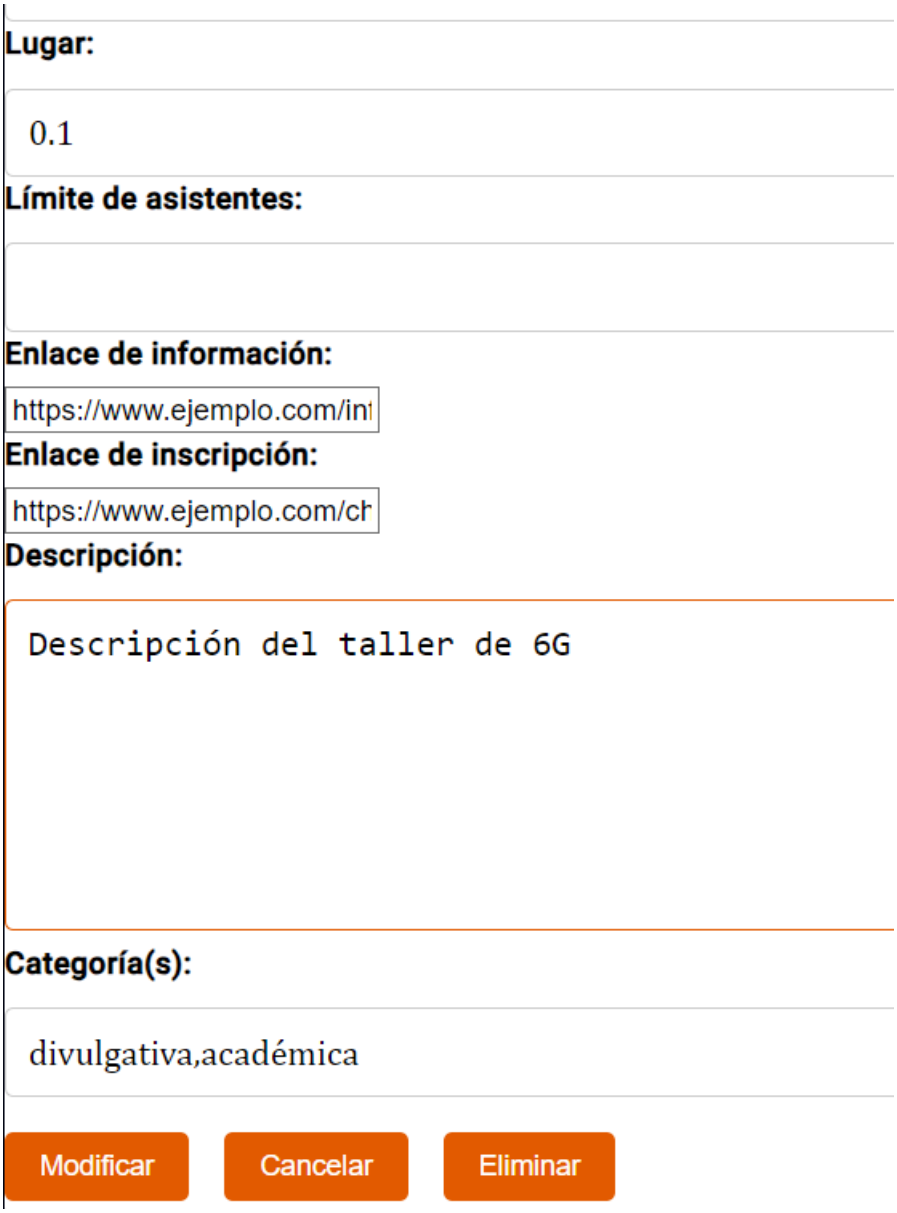


Figura 6.16: Parte de la tabla de actividades.

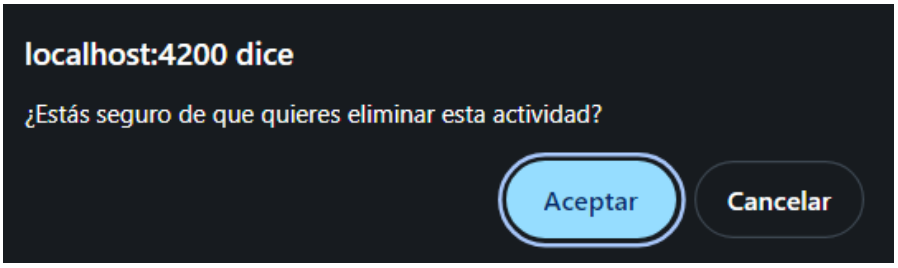


Figura 6.17: Doble confirmación sobre el borrado de la aplicación.

Figura 6.18: Doble confirmación sobre el borrado de la aplicación.

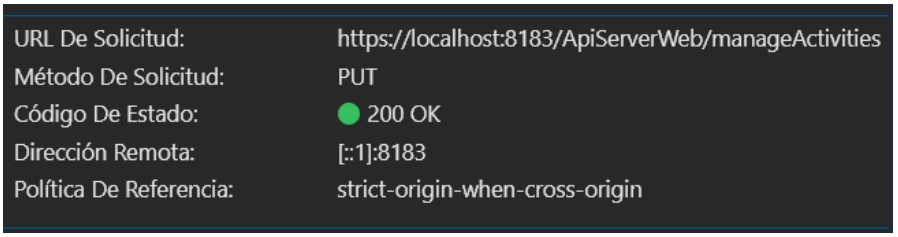


Figura 6.19: Respuesta sobre la petición para modificar una actividad.

Capítulo 7

Conclusiones y trabajo a futuro.

En este proyecto se ha abordado el problema que hoy existe respecto a la gestión de actividades como charlas, formación, talleres,etc. que hoy en día en la ETSIIT (Escuela Técnica Superior de Ingeniería Informática y Telecomunicación) en la Universidad de Granada. Este problema reside en que a la hora de agendar y organizar este tipo de actividades que suelen ser de gran interés para el alumnado, por lo que el equipo docente las tiene muy en cuenta a la hora de realizarlas, pero para realizar dicha acción se requiere de mucho tiempo y esfuerzo por su parte, ya que hoy en día se gestionan de forma manual, es decir, un profesor que requiera agendar una actividad de este tipo, debe ponerse en contacto con el responsable de estas tareas y a través de una reunión o hilo de correos, intentar agendar de una manera adecuada y todo esto sin que surjan otro tipo de problemas de disponibilidad que dificulta este proceso.

Por ello, se ha desarrollado una aplicación totalmente funcional para que este tipo de procedimientos sean ágiles y eficientes. Lo que resulta en un ahorro de tiempo y esfuerzo para el equipo docente de la escuela, a la vez que brinda una herramienta adicional a los alumnos para encontrar qué tipo de actividades existentes son las que más le interesan, según sus prioridades de crecimiento profesional.

Esto se ha realizado a través de un conjunto de frameworks y herramientas; por ejemplo, para albergar todos los datos se ha construido una base de datos no estructurados a través de MongoDB. Se ha implementado una API, a través del framework de RESTLET, la cual es capaz de permitir que terceras partes puedan desarrollar frontends u otros elementos que puedan desarrollarse para gestionar este tipo de aplicaciones. Como interfaz de usuarios se ha utilizado el framework de Angular, donde se ha creado un par de páginas web. Una de ellas destinada a los alumnos, que contiene un calendario y un filtro para que puedan encontrar de una manera ágil y eficiente las actividades que más les interesen, y una web de administración donde el equipo docente puede gestionar todas las actividades que organicen de una manera rápida y fácil, ya que esta web cuenta con numerosas herramientas tanto para la gestión, como para la difusión de las mismas mediante procesos automatizados. Como puede ser la creación y eliminación de usuarios;

la creación, eliminación y modificación de actividades que cuenta con una tabla dinámica donde se pueden encontrar todas las actividades y filtrarlas de una manera ágil y sencilla y la automatización de envío o descarga de actividades bajo plantilla predefinida para una rápida y cómoda difusión.

Finalmente, se proponen una serie de trabajos a futuro que se podrían realizar para seguir mejorando la aplicación implementada:

- Se podría diferenciar entre las plantillas de envío automatizado de actividades y las plantillas de exportación a un archivo de texto.
- En la web de usuarios, en la parte del calendario, se podría implementar una función que permita a los alumnos pinchar sobre las actividades que aparezcan en el mes en el que se encuentren y aparezca información adicional sobre la misma.
- Se podría implementar una función en la web de usuarios donde, a través del límite de asistentes a las actividades, los alumnos podrían inscribirse mediante algún dato personal como correo o nombre. Y, asimismo, indicarle cuántas plazas quedan disponibles para participar.
- Por último, se podría añadir al componente principal de la web de usuarios, un cuadro que muestre información de interés para los alumnos sobre la escuela, actividades o cualquier otro tipo.

Bibliografía

[1] Mark C Layton. *Scrum*. eng. Ed. por Inc Hoboken New Jersey : John Wiley Sons. Third edition. for dummies. Hoboken, New Jersey: John Wiley Sons, Inc., 2023. ISBN: 1-119-90468-4. URL: https://granatensis.ugr.es/permalink/34CBUA_UGR/1p2iirq/alma991014579346404990.

[2] John Funk. *Jira Work Management for business teams : accelerate digital transformation and modernize your organization with Jira Work Management*. eng. Birmingham, UK: Packt Publishing, 2022. ISBN: 1-80324-914-5.

[3] Brittany Joiner. *Supercharging productivity with Trello : harness Trello's powerful features to boost productivity and team collaboration*. eng. 1st ed. Birmingham, England: Packt Publishing, 2023. ISBN: 1-80181-828-2.

[4] Atlassian. *Jira Software*. 2024. URL: https://www.atlassian.com/software/jira.

[5] Oracle Corporation. *Java Platform Standard Edition 17 Development Kit*. 2024. URL: https://www.oracle.com/java/technologies/javase-jdk17-downloads.html.

[6] Microsoft Corporation. *Visual Studio Code*. 2024. URL: https://code.visualstudio.com/.

[7] Inc. GitHub. *GitHub*. 2024. URL: https://github.com/.

[8] Inc. Restlet. *Restlet Framework*. 2024. URL: https://restlet.com/.

[9] Inc. MongoDB. *MongoDB*. 2024. URL: https://www.mongodb.com/.

[10] Google LLC. *Angular - The modern web developer's platform*. 2024. URL: https://angular.io/.

[11] Google LLC. *Google Meet - Secure Video Meetings*. 2024. URL: https://meet.google.com/.

[12] Universidad de Granada. *Restibuc Artículo 23 BIS*. 2022. URL: https://investigacion.ugr.es/sites/vic/investigacion/public/documentos/personal/publicaciones/2022/restibucArt23BIS.pdf.

[13] Inc. Buffer. *Buffer - The social media management platform*. 2024. URL: https://buffer.com/.

[14] Inc. HubSpot. *HubSpot - Inbound Marketing, Sales, and Service Software*. 2024. URL: https://hubspot.com/.

[15] Inc. Asana. *Asana - Work Management Platform for Teams*. 2024. URL: https://asana.com/.

[16] Inc. Zendesk. *Zendesk - Customer Service Software & Sales CRM*. 2024. URL: https://zendesk.com/.

[17] Inc. Facebook. *React - A JavaScript library for building user interfaces*. 2024. URL: https://reactjs.org/.

[18] Evan You. *Vue.js - The Progressive JavaScript Framework*. 2024. URL: https://vuejs.org/.

[19] Ember Core Team. *Ember.js - A framework for ambitious web developers*. 2024. URL: https://emberjs.com/.

[20] Inc. StrongLoop. *Express - Fast, unopinionated, minimalist web framework for Node.js*. 2024. URL: https://expressjs.com/.

[21] Encode OSS Ltd. *Django REST framework - Powerful and flexible toolkit for building Web APIs*. 2024. URL: https://www.django-rest-framework.org/.

[22] Pallets Projects. *Flask - The Python micro framework for building web applications*. 2024. URL: https://flask.palletsprojects.com/.

[23] Ramez Elmasri y Shamkant B. Navathe. *Fundamentals of Database Systems*. 7th. London: Pearson, 2015. ISBN: 978-0133970777.

[24] S. Sudarshan Abraham Silberschatz Henry Korth. *Database System Concepts*. 7th. New York: McGraw-Hill Education, 2019. ISBN: 978-0078022159.

[25] Benjamin Bengfort, Rebecca Bilbro y Tony Ojeda. *Data Analytics with Hadoop: An Introduction for Data Scientists*. 2nd. Sebastopol, CA: O'Reilly Media, 2018. ISBN: 978-1491913703.

[26] Roger Hecht y Stefan Jablonski. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491903063.

[27] Oracle Corporation. *MySQL Documentation*. Accessed: 2024-05-28. 2024. URL: https://dev.mysql.com/doc/.

[28] Microsoft Corporation. *Microsoft SQL Server Documentation*. Accessed: 2024-05-28. 2024. URL: https://docs.microsoft.com/en-us/sql/.

[29] Apache Software Foundation. *Apache Cassandra Documentation*. Accessed: 2024-05-28. 2024. URL: https://cassandra.apache.org/doc/latest/.

[30] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN: 978-0201633610.

[31] David Geary y Cay Horstmann. *Core JavaServer Faces*. 2016. URL: https://www.pearson.com/store/p/core-javaserver-faces/P100000042104.

[32] Mozilla Developer Network. *Introduction to HTML*. 2024. URL: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML.

[33] Google Developers. *Web Fundamentals*. 2024. URL: https://developers.google.com/web/fundamentals.

[34] Addy Osmani. *Learning JavaScript Design Patterns*. 2024. URL: https://addyosmani.com/resources/essentialjsdesignpatterns/book/.

[35] Dino Esposito. *Programming ASP.NET Core*. Microsoft Press, 2018. ISBN: 978-1509308340.

[36] Martin Fowler. *Richardson Maturity Model*. 2024. URL: https://martinfowler.com/articles/richardsonMaturityModel.html.

[37] Mark Masse. *REST API Design Rulebook*. O'Reilly Media, 2011. ISBN: 978-1449310509.

[38] Apigee. *Web API Design: The Missing Link*. 2017. URL: https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf.

[39] Luc Perkins, Eric Redmond y Jim Wilson. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. 2018. URL: https://pragprog.com/titles/rwdata/seven-databases-in-seven-weeks/.

[40] Martin Fowler. *NoSQL Databases*. 2024. URL: https://martinfowler.com/articles/nosql-intro.html.

[41] Stack Overflow. *Stack Overflow Developer Survey 2023*. 2023. URL: https://insights.stackoverflow.com/survey/2023.

[42] Google LLC. *Angular Documentation*. 2024. URL: https://angular.io/docs.

[43] Brad Green e Igor Minar. *Angular: A Framework for Building Client-Side Applications*. Google, 2019.

[44] stateofapi. *The State of API 2023*. 2023. URL: https://www.stateofapi.com/.

[45] RESTLET. *RESTLET Documentation*. 2024. URL: https://restlet.com/open-source/documentation/.

[46] RESTLET. *RESTLET GitHub Repository*. 2024. URL: https://github.com/restlet/restlet-framework-java.

[47] db-engines. *DB-Engines Ranking 2023*. 2023. URL: https://db-engines.com/en/ranking.

[48] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media, 2018.

[49] Kristina Chodorow. *Scaling MongoDB*. MongoDB, Inc., 2017.

[50] Inc. MongoDB. *MongoDB Community Server*. MongoDB, Inc. 2024. URL: https://www.mongodb.com/try/download/community.

[51] Daniel Moreno Jimenez. *WebServiceAvtivityUgr*. 2024. URL: https://github.com/Daniel-MJ/WebServiceAvtivityUgr.

[52] Baeldung. "CORS Preflight Requests". En: (2024). URL: https://www.baeldung.com/cs/cors-preflight-requests.

[53] Stack Overflow. *CORS: Cannot use wildcard in Access-Control-Allow-Origin when credentials flag is true*. 2013. URL: https://stackoverflow.com/questions/19743396/cors-cannot-use-wildcard-in-access-control-allow-origin-when-credentials-flag-i.

[54] Stack Overflow. *Why does the preflight OPTIONS request of an authenticated CORS request work in Chrome but not Firefox?* 2013. URL: https://stackoverflow.com/questions/15734031/why-does-the-preflight-options-request-of-an-authenticated-cors-request-work-in.

[55] Stack Overflow. *CORS Unauthorized 401 Error When Calling Spring REST API*. 2019. URL: https://stackoverflow.com/questions/56973215/cors-unauthorized-401-error-when-calling-spring-rest-api.

[56] Stack Overflow. *How to use CORS in Restlet 2.3.1*. 2015. URL: https://stackoverflow.com/questions/28988671/how-to-use-cors-in-restlet-2-3-1.

[57] Chrome Developers. *Network Reference*. 2024. URL: https://developer.chrome.com/docs/devtools/network/reference?utm_source=devtools&hl=es-419#provisional-headers.

[58] Floodlight. *RestApiServer.java*. 2024. URL: https://github.com/floodlight/floodlight/blob/71fe8a7e72096eb0fd96c1d814a04e3b7b782830/src/main/java/net/floodlightcontroller/restserver/RestApiServer.java.

[59] Inc. MongoDB. *MongoDB CRUD Operations in Java*. 2024. URL: https://learn.mongodb.com/courses/mongodb-crud-operations-in-java.

[60] Oracle Corporation. *Envío de correos electrónicos con el servicio de Oracle Cloud Infrastructure Email Delivery usando JavaMail*. 2024. URL: https://docs.oracle.com/es-ww/iaas/Content/Email/Reference/javamail.htm.

[61] xdevplatform. *Twitter API Java SDK*. 2024. URL: https://github.com/xdevplatform/twitter-api-java-sdk.

[62] X (formerly Twitter). *Authentication with OAuth 2.0*. 2024. URL: https://developer.x.com/en/docs/authentication/oauth-2-0.

[63] Daniel Moreno Jimenez. *WebUsers*. 2024. URL: https://github.com/Daniel-MJ/WebUsers.

[64] Daniel Moreno Jimenez. *WebAdmin*. 2024. URL: %5Curl%7Bhttps://github.com/Daniel-MJ/WebAdmin%7D.

[65] Node.js. *Node.js Downloads - Package Manager*. 2024. URL: https://nodejs.org/en/download/package-manager.

[66] PrimeNG. *PrimeNG FullCalendar*. 2024. URL: https://www.primefaces.org/primeng-v14-lts/fullcalendar.

[67] *FullCalendar Angular Documentation*. FullCalendar. 2024. URL: https://fullcalendar.io/docs/angular.