UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación para la organización de eventos.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Lin, Yu

Tutor/a: Albert Albiol, Manuela

CURSO ACADÉMICO: 2023/2024

Resum

L'objectiu d'aquest projecte és desenvolupar una aplicació multiplataforma (mòbil, web i d'escriptori) per a l'organització i la gestió d'esdeveniments. L'aplicació s'adreça a grups de persones que tingan un interès comú sobre un tema (pot ser entreteniment, educatiu, activitats comercials, etc.). L'aplicació servirà per facilitar la planificació i la publicació d'esdeveniments amb el propòsit de compartir l'experiència i descobrir possibles usuaris amb la mateixa motivació. L'aplicació també proporcionarà un mitjà per mantenir les comunicacions tant entre els possibles organitzadors com entre els participants. Les tecnologies que es faran servir per al desenvolupament d'aquesta aplicació seran: Flutter, Java Spring Boot i Postgresql.

Paraula clau: Organització d'events; Flutter; Java Spring Boot; Postgresql; interès comú

Resumen

El objetivo de este proyecto es desarrollar una aplicación multiplataforma (móvil, web y de escritorio) para la organización y gestión de eventos. La aplicación está dirigida a grupos de personas que tengan un interés común sobre un tema (puede ser entretenimiento, educativo, actividades comerciales, etc.). La aplicación servirá para facilitar la planificación y publicación de eventos con el propósito de compartir la experiencia y descubrir posibles usuarios con la misma motivación. La aplicación proporcionará también un medio para mantener las comunicaciones tanto entre los posibles organizadores como entre los participantes. Las tecnologías que se utilizarán para el desarrollo de esta aplicación serán: Flutter, Java Spring Boot y Postgresql.

Palabra clave: Organización de eventos; Flutter; Java Spring Boot; multiplataforma; interés común.

Abstract

The aim of this project is to develop a cross-platform application (mobile, web and desktop) for the organisation and management of events. The application is aimed at groups of people who have a common interest in a topic (it can be entertainment, educational, commercial activities, etc.). The application will serve to facilitate the planning and publication of events with the purpose of sharing the experience and discovering potential users with the same motivation. The application will also provide a means to maintain communications between both potential organisers and participants. The technologies to be used for the development of this application will be: Flutter, Java Spring Boot and Postgresql.

Key words: Event organisation; Flutter; Java Spring Boot; Postgresql; Cross-platform; Common interest.

Índice general

Apéndice

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

En este capítulo de introducción se presenta una descripción general sobre el proyecto, explicando la motivación inicial, la solución propuesta para el problema, la metodología para su desarrollo y el resultado que se espera tener tras el despliegue del producto.

1.1 Motivación

En el mundo actual, la información es un elemento esencial en todos los aspectos de la vida. Es fundamental para la planificación, la resolución de los problemas y la búsqueda de nuevas oportunidades.

Sin embargo, no existe una fuente única de información. Provienen de diversos canales de manera repetitiva, con alta posibilidad de falta de proceso de verificación sobre el contenido, y con pocos filtros, de manera que, el usuario no recibe informaciones que resulten realmente útil o de interés.

En el caso concreto de los eventos, los usuarios obtienen la información relacionada con ellos a través de diversas plataformas, mientras que la inscripción se lleva a cabo en aplicaciones distintas. Esto genera inconvenientes tanto para los usuarios como los organizadores.

Los organizadores, por su parte, enfrentan la dificultad de tener que publicar la información en múltiples plataformas, lo que dispersa sus esfuerzos y complica la gestión. Además, para coordinar y organizar dichos eventos, a menudo deben recurrir a otras aplicaciones, como la de mensajería instantánea, lo que fragmenta aún más el proceso y reduce la eficiencia en la gestión del evento.

1.2 Objetivos

El objetivo que se busca alcanzar es el diseño y desarrollo de una plataforma centralizada que simplifique el proceso de creación de eventos y la inscripción a los mismos. La aplicación entre otras funcionalidades ofrecerá un chat integrado, facilitando la comunicación entre usuarios

dentro de la misma aplicación. También, cada usuario, podrá registrar sus preferencias, de forma que se recibirán notificaciones e informaciones personalizadas sobre eventos en las áreas o grupos de organizadores que sean de interés para el usuario.

Como objetivo personal, se pretende mostrar la capacidad para aplicar las habilidades de aprendizaje de nuevas tecnologías y el uso de conocimientos obtenidos en la escuela.

1.3 Metodologías

La finalidad de emplear una metodología para el desarrollo de software es mejorar la eficiencia y calidad de desarrollo, al aplicar un marco estructurado para gestionar los procesos y los recursos.

En términos generales, las metodologías se pueden clasificar en dos grandes categorías, las metodologías clásicas y las metodologías ágiles.

Las metodologías clásicas tienen un enfoque lineal y secuencial, en el que cada fase debe completarse para pasar al siguiente. Son menos flexibles y con mayores dificultades para adaptar a los cambios de requisitos, suelen ser utilizado por grupos experimentados en proyectos de gran tamaño, en los cuales se cuenta con una planificación clara y detallada en cada fase del proceso.

Las metodologías ágiles tienen un enfoque iterativo e incremental, permitiendo visualizar y comprobar los resultados de cada iteración o versión, proporcionando una mayor flexibilidad y capacidad de adaptación frente los cambios de requisitos.

La metodología que se aplica en el desarrollo de este proyecto es la metodología ágil Kanban, es un enfoque visual y flexible para gestionar el trabajo y los procesos. Utiliza un tablero dividido en columnas que representa las distintas fases del proceso, como hacer, por hacer y acabado, donde cada tarea se muestra en una tarjeta [1].

Este se adapta a las características del trabajo individual y de pequeño tamaño. Dado que el mismo desarrollador asume el rol de Product Owner se beneficia de una flexibilidad total, permitiendo la incorporación de cambios en los requisitos en cualquier momento sin incurrir en costos de comunicación.

La herramienta que se utiliza de tablero Kanban es Trello [2], la figura 1.1 muestra una captura de pantalla del tablero Trello.

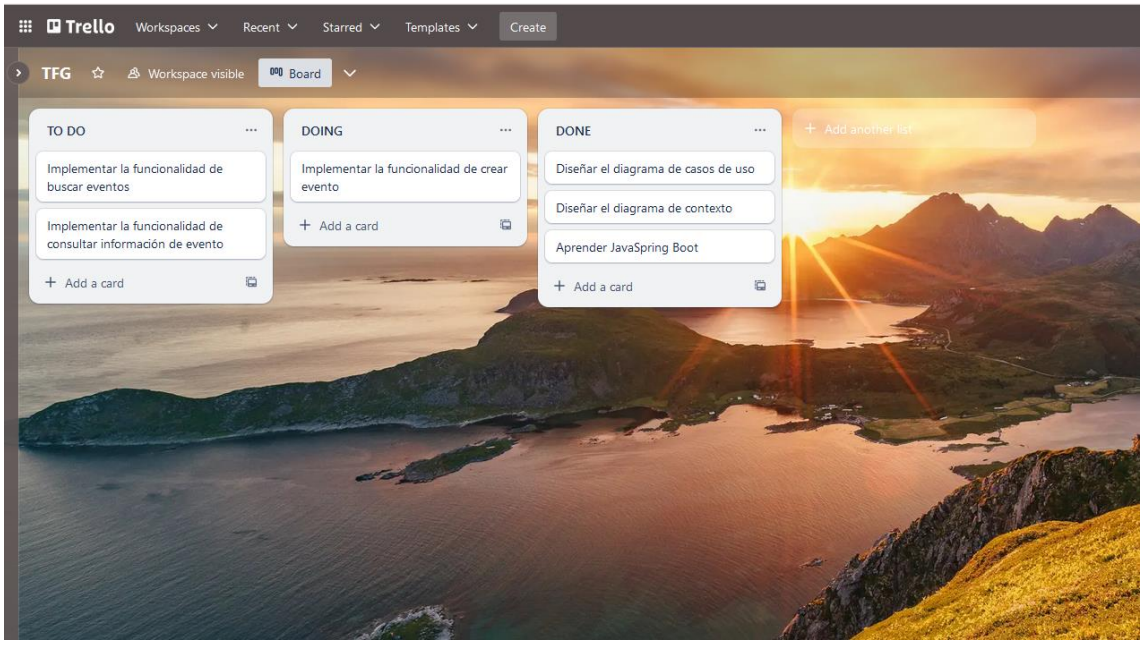

Figura 1.1: Tablero Trello

1.4 Estructura de memoria

En esta sección se detalla la estructura de la presente memoria, describiendo el contenido principal de cada capítulo:

■ **Estado del arte:** Se investiga las aplicaciones disponibles en el mercado actual que ofrecen funcionalidades similares a nuestra propuesta y se comenta las diferencias entre ellas.

■ **Análisis del problema:** Se analiza las funcionalidades esenciales para alcanzar los objetivos establecidos para este proyecto, empleando las técnicas para el análisis de requisitos.

■ **Diseño de la solución:** Explicar en detalle la arquitectura empleada para la implementación de la aplicación, así como las herramientas y las tecnologías utilizadas en el desarrollo.

■ **Desarrollo de la solución propuesta:** Presentar la implementación concreta de la solución propuesta, describiendo la estructura del base de datos y los componentes que construye el sistema.

■ **Implantación:** Describe el proceso para llevar al cabo la implantación del sistema, desde la preparación del entorno hasta la puesta en marcha en el entorno de producción.

■ **Pruebas:** Se exponen las pruebas diseñadas para evaluar la aplicación desarrollada y los resultados obtenidos en dichas pruebas.

■ **Conclusiones:** Se lleva al cabo una revisión de los trabajos realizados en relación con los objetivos establecidos en la fase inicial del proyecto, y se reflexiona sobre la correlación entre el proyecto y el estudio cursado en la escuela.

CAPÍTULO 2

Estado del arte

En este capítulo se investiga las aplicaciones que existen actualmente en el mercado que tengan funcionalidades similares a este proyecto, y se analizan sus ventajas y limitaciones en comparación con la solución propuesta en este trabajo.

2.1 Aplicaciones en el mercado

Eventbrite [3] es una plataforma ampliamente utilizada para la gestión de eventos y venta de entradas. Ofrece herramientas para creación de eventos, promoción, y análisis de asistentes. Se dispone de canales de comunicación tanto por correo electrónico como por mensajes de texto para contactar y mantener informados a los asistentes del evento.

La diferencia con nuestra propuesta es que se centra en la compra de billetes y de gestión de eventos y no proporciona las funcionalidades para que los usuarios puedan establecer nuevas conexiones y compartir las experiencias del evento.

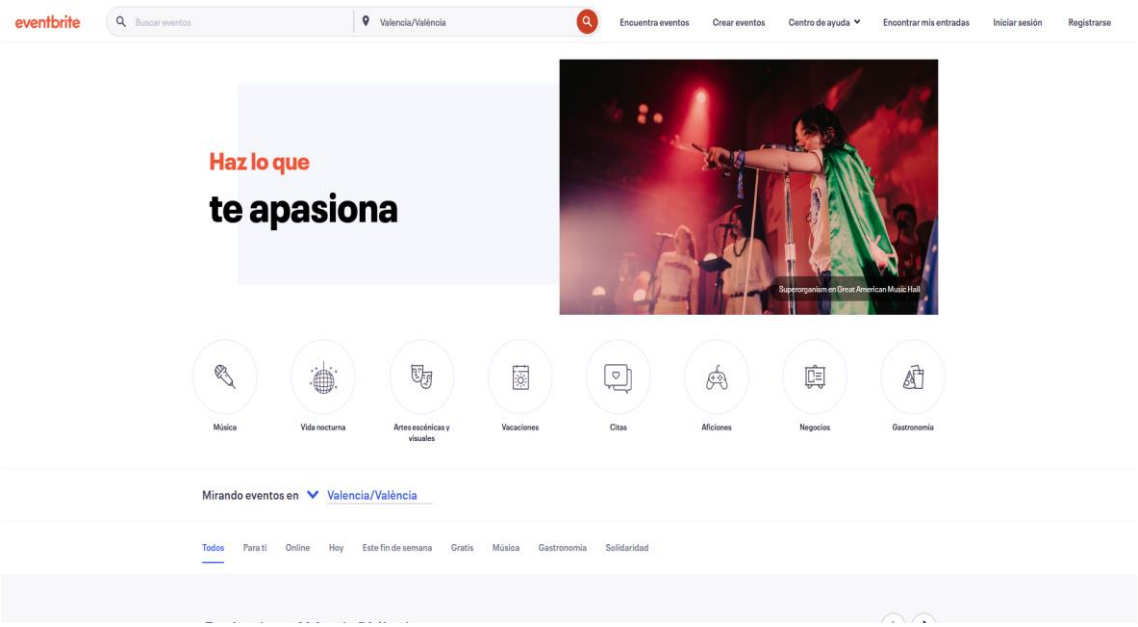

Figura 2.1: Página web Eventbrite

Cvent [4], aplicación para la gestión de eventos, que incluye registro de asistentes, creación de agendas, y análisis post-evento. Es utilizada tanto para eventos presenciales como virtuales. ofrece herramientas de networking que permiten a los asistentes conectarse entre sí, programar reuniones, o participar en grupos de discusión.

En esta plataforma se organiza eventos principalmente en el ámbito comercial y educacional, y no se ajusta al requisito de permitir a los usuarios organizar eventos en su área de interés, y a la búsqueda de personas con la misma motivación.

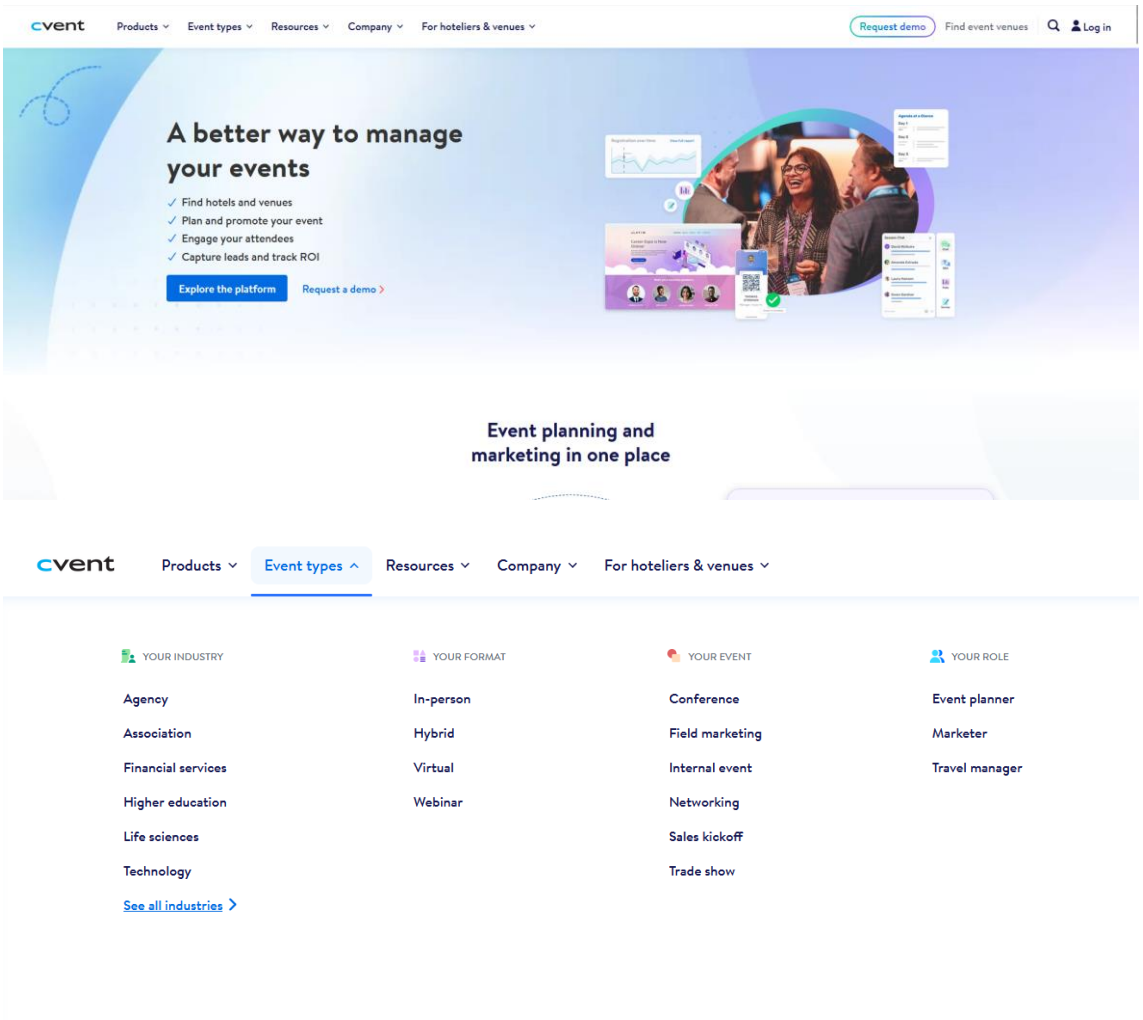

Figura 2.2: Página web Cvent

Whova [5] es una aplicación que facilita la gestión de conferencias y eventos. Ofrece funciones de registro, creación de agendas, y herramientas de interacción para los asistentes. Dispone de herramientas para la transmisión de sesiones en vivo y el acceso a contenido grabado. Proporciona análisis detallados sobre la asistencia y la participación. Los organizadores pueden generar informes personalizados para evaluar el resultado del evento.

Esta plataforma es similar a Cvent, con las mismas limitaciones, pero teniendo mayor enfoque en la experiencia de asistente y la interactividad.

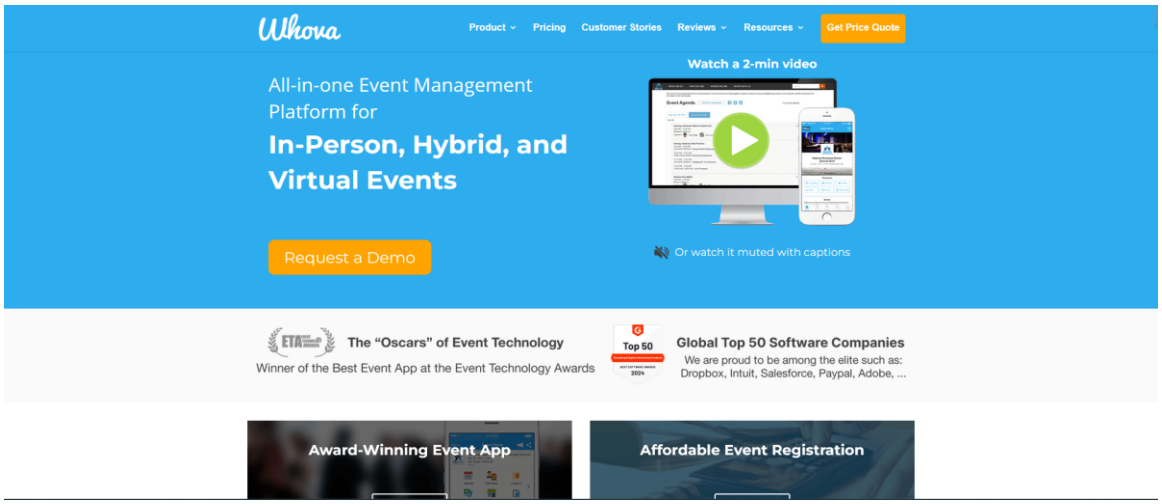

Figura 2.3: Página web Whova

2.2 Propuesta

Tras realizar la investigación sobre software en el mercado actual, se ha identificado que existe varias aplicaciones que ofrece funcionalidades similares a las propuestas en nuestro proyecto. Sin embargo, ninguna de estas alternativas podía satisfacer plenamente nuestra visión y objetivos específicos.

Nuestra solución se basa en el desarrollo de una plataforma integrada que permitirá a los usuarios organizar, gestionar o participar en eventos de manera eficiente. Además, en la aplicación se ofrece funciones que facilita a los usuarios la búsqueda de nuevas conexiones con personas que comparten intereses y motivaciones similares. Los usuarios no solamente pueden participar en los eventos, sino también comparten sus experiencias y refuerzan sus redes en un entorno colaborativo y dinámico.

Este enfoque pretende llenar un vacío existente en el mercado, proporcionando una herramienta que no solo simplifica la gestión de eventos, sino que también promueve la interacción y el intercambio de experiencias entre individuos con intereses comunes, creando una comunidad sólida y comprometida.

CAPÍTULO 3

Análisis del problema

En este capítulo se realiza un análisis de las funcionalidades esenciales necesarias para alcanzar los objetivos de este proyecto. Se especifica estas funcionalidades claves en los requisitos detallados utilizando las técnicas correspondientes, con una descripción clara y precisa de las necesidades que debe satisfacer la solución propuesta, garantizando la calidad y la consistencia del proceso de desarrollo.

3.1 Perfiles de stakeholders

En esta sección, se identifican los stakeholders involucrados en la aplicación. En la tabla 3.1 muestra los stakeholders indicando su nombre, rol, si son usuarios directos y sus intereses sobre la aplicación. Estas informaciones son esenciales para comprender sus necesidades y expectativas sobre el proyecto.

Nombre	Rol	Usuario Directo (sí/no)	Intereses
Organizador de eventos	Persona que usa la aplicación	Sí	Interfaz fácil de usar para crear, gestionar y promocionar los eventos. Un canal donde pueda publicar los resultados de los eventos con imágenes o audios.
Participante de eventos	Persona que usa la aplicación	Sí	Interfaz fácil de usar para búsqueda e inscripción de los eventos de interés. Función de chat con las personas que tienen la misma motivación.
Equipo de marketing	Equipo de marketing de la empresa organizadora de eventos	No (Sí en caso de que sea el departamento que publique el evento)	Las informaciones de los eventos se distribuyen de manera eficiente a los usuarios que interesen en el área de especialización de la empresa.
Equipo técnico	Equipo técnico de la aplicación	No	La aplicación sea seguro y fácil de mantener.

Tabla 3.1: Perfil de stakeholders

3.2 Límites del sistema

En la figura 3.1 se muestra el diagrama de contexto. En este diagrama se puede observar una representación gráfica de alto nivel con actores con la interfaz del sistema.

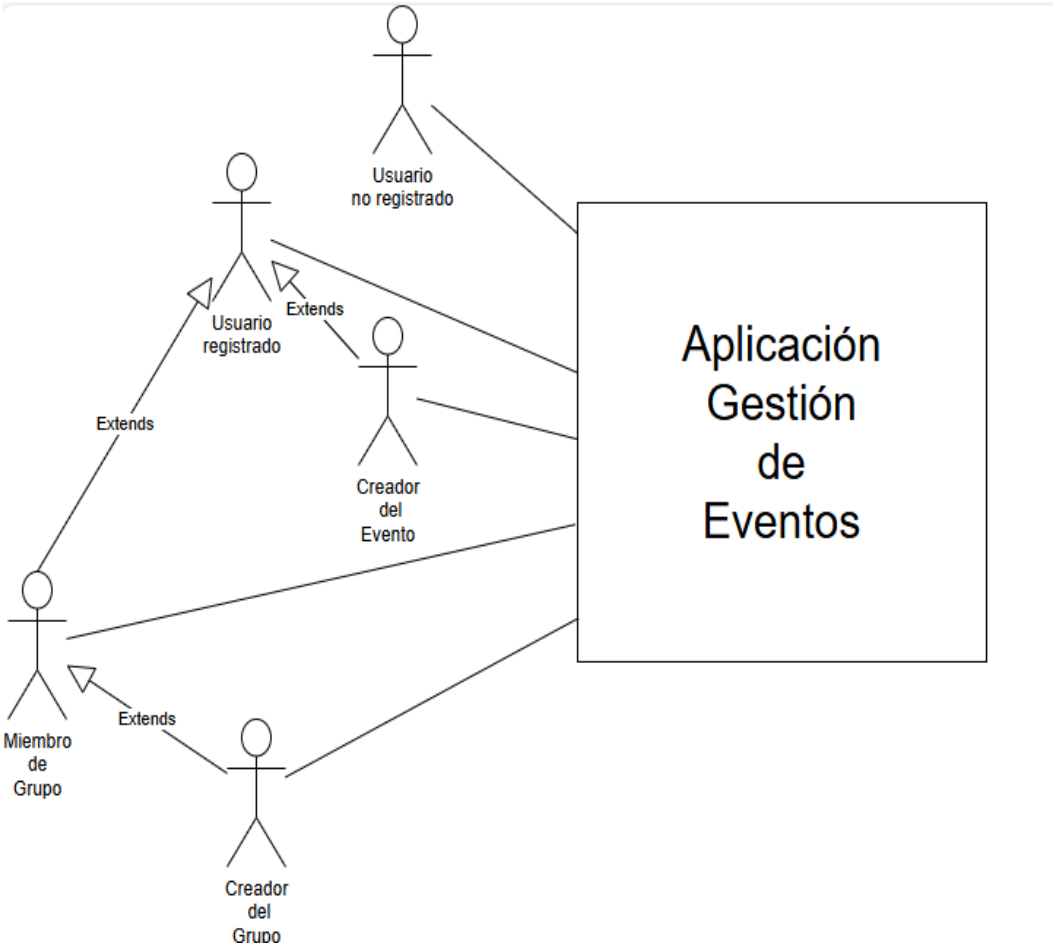

Figura 3.1: Diagrama de Contexto

3.3 Requisitos funcionales

En la figura 3.2 se presenta un Diagrama de Casos de Uso que ilustra los requisitos funcionales de la solución propuesta. Estos requisitos describen las acciones y funcionalidades que los usuarios puedan llevar a cabo a través de la aplicación.

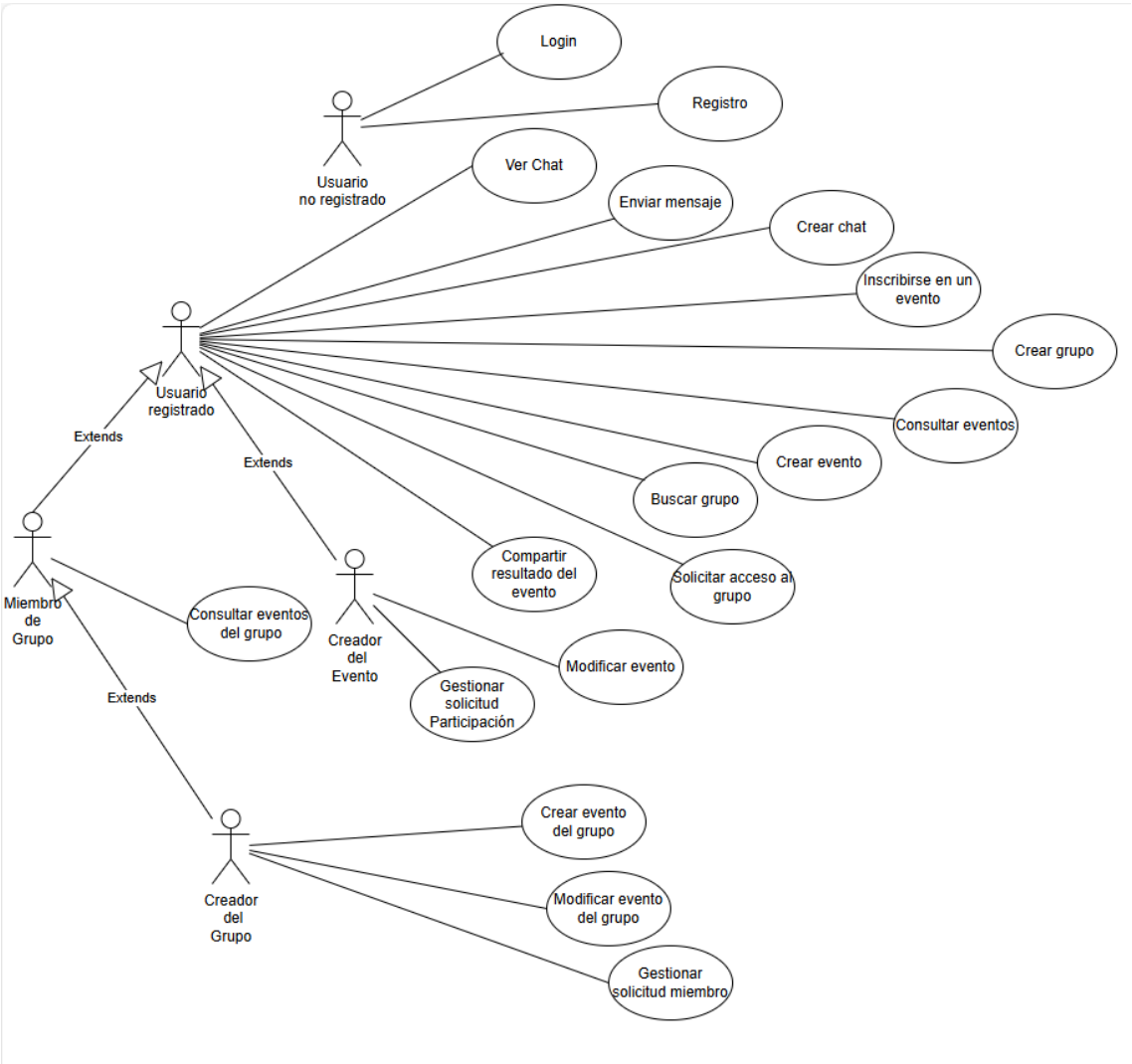

Figura 3.2: Diagrama de Casos de Uso

Actor **Usuario no registrado:**

- **Login:** El usuario se identifica en el sistema introduciendo las credenciales para realizar acciones como usuario registrado.
- **Registro:** El usuario se crea nuevas credenciales para autenticarse en el sistema.

Actor **Usuario Registrado:**

- **Crear chat:** El usuario selecciona un participante para crear un espacio de conversación.
- **Ver chat:** El usuario entra al espacio de conversación para leer los mensajes.
- **Enviar mensaje:** El usuario entra al espacio de conversación, edita el mensaje y se envía a otro/s participante/s.
- **Crear evento:** El usuario introduce las informaciones sobre el evento y se publica en el canal de los eventos. Se crea el chat de grupo para el evento.
- **Consultar eventos:** El usuario entra al canal de eventos y se realiza una búsqueda sobre los eventos existentes del momento.
- **Inscribirse en un evento:** El usuario selecciona un evento e introduce sus datos personales para inscribirse.
- **Crear grupo:** El usuario selecciona participantes y se crea un espacio de conversación para el chat del grupo.
- **Buscar grupo:** El usuario introduce nombre o tags y se busca el grupo.
- **Solicitar acceso al grupo:** El usuario selecciona un grupo y envía la solicitud de acceso al grupo a creador del grupo.
- **Compartir resultado del evento:** El usuario selecciona un evento participado y se envía imágenes o videos del evento.

Actor **Miembro de Grupo:**

- **Consultar eventos del grupo:** El usuario selecciona el grupo cuyo sea miembro y consulta la lista de eventos.

Actor **Creador del Evento:**

- **Modificar evento:** El usuario selecciona el evento y modifica las informaciones.
- **Gestionar solicitud participación:** El usuario abre la lista de solicitud de inscripción al evento y se revisa, acepta o rechazar las solicitudes.

Actor **Creador del Grupo:**

- **Crear evento del grupo:** El usuario selecciona el grupo cuyo sea creador y introduce las informaciones sobre el evento del grupo y se publica en el listado de los eventos del grupo.
- **Modificar evento del grupo:** El usuario selecciona el evento del listado del grupo cuyo sea creador y modifica las informaciones del evento.
- **Gestionar solicitud miembro:** El usuario abre el listado de la solicitud de acceso al grupo y se revisa, acepta o rechazar la solicitud.

3.4 Requisitos no funcionales

En esta sección se describe los requisitos no funcionales que debe cumplir el sistema.

Usabilidad

- Las funcionalidades de la aplicación deben ser intuitivas, permitiendo a los usuarios comprender y utilizar la aplicación sin necesidad de instrucciones extensas.
- El sistema debe permitir a los usuarios operarlo y controlarlo con facilitad.

Seguridad

- La aplicación debe evitar el acceso a los datos e informaciones de los usuarios no autenticados o no autorizados.
- La aplicación debe prevenir la modificación a los datos del programa de los usuarios no autenticados o no autorizados.
- El sistema debe garantizar la autenticidad de la identidad del usuario que accede a su cuenta, evitar una suplantación de identidad.

Mantenibilidad

- Las modificaciones de un componente del programa deben tener un mínimo impacto en otros.
- Los componentes de programa deben permitir usarlo en más de un sistema software o para construir otra aplicación

Rendimiento

- El tiempo de respuesta del sistema a las peticiones del usuario debe ser menor que 5 segundos.
- El sistema debe poder procesar peticiones de varios usuarios simultáneamente.

3.5 Fases del proyecto

En este apartado se realiza una descripción sobre el proceso de implementación de la solución propuesta, dividiendo en varias fases, desde el desarrollo, a la implantación y finalmente la validación.

Desarrollo:

1. **Preparación de backend y frontend:** en esta fase se configura y se instala las dependencias y las herramientas necesarias para el desarrollo, y se crea clases temporales de pruebas para comprobar el funcionamiento correcto entre varias tecnologías.
2. **Diseño de la interfaz:** en esta fase se lleva a cabo el diseño de los prototipos de la interfaz de usuario, con el objetivo de intentar representar la apariencia y el funcionamiento final de la aplicación. También se facilita una validación temprana con los usuarios finales sobre requisitos no funcionales de usabilidad.
3. **Implementación:** durante esta fase se desarrolla simultáneamente el frontend y el backend siguiendo los diagramas y el diseño de interfaz previamente definidos.

Pruebas:

1. **Pruebas de integración:** Se realizan pruebas para confirmar que los distintos módulos del sistema que participan en ejecución de una funcionalidad interactúan correctamente entre sí.
2. **Pruebas de aceptación:** Se realizan pruebas para verificar que el sistema cumple los requisitos y expectativas del usuario final.

Implantación:

1. **Preparación de entorno:** Se realiza las configuraciones necesarias para el soportar la aplicación en el entorno de producción y también la transferencia de los datos e informaciones usado en entorno de desarrollo para verificar el funcionamiento correcto.
2. **Despliegue:** Se instala la aplicación en el entorno de producción y se lanza a ejecución el sistema.

CAPÍTULO 4

Diseño de la solución

En este capítulo se detalla el diseño de la implementación de la propuesta, explicando la arquitectura que se utiliza para construir el sistema, los principios y patrones de diseño que se adoptan para obtener mejores prácticas en la organización y desarrollo del software, y las tecnologías seleccionadas para el desarrollo eficiente y alineado con los objetivos del proyecto.

4.1 Arquitectura del sistema

La arquitectura utilizada para la construcción del sistema es el patrón arquitectónico MVC (model-view-controller) [6], se organiza en tres componentes principales y cada uno con su responsabilidad especifica.

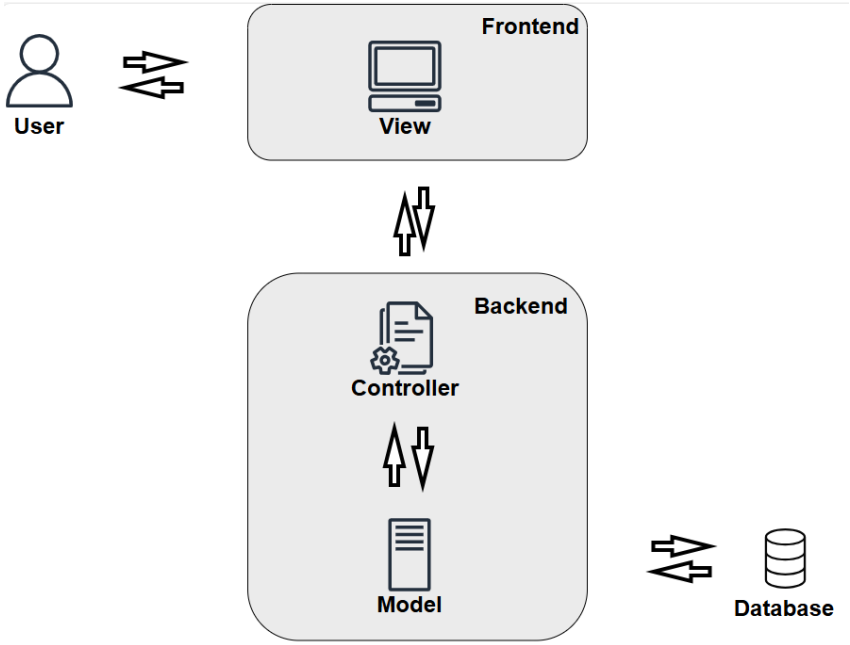

Figura 4.1: Arquitectura del Sistema

Modelo (Model):

El modelo se encarga de gestionar los datos y la lógica de negocio. Se define las estructuras de datos, gestiona la lógica de la aplicación y se comunica con la base de datos para almacenar y recuperar la información.

Controlador (Controller):

El controlador es el intermediario entre el modelo y la vista. Maneja la entrada del usuario, invoca acciones en el modelo y con el resultado que devuelve se actualiza la vista. Es responsable de manejar la lógica del flujo de la aplicación, gestionando la interacción entre el frontend y el backend.

Vista (View):

La vista se encarga de presentar los datos al usuario. Se define la interfaz de usuario usando los datos provenientes del modelo. Su objetivo es facilitar una interacción intuitiva y visual del usuario con la aplicación.

Flujo general del sistema:

1. **Operaciones del usuario:** El usuario se realiza acciones sobre la interfaz a través del frontend.
2. **Procesamiento del controlador:** Las peticiones del usuario se envían al backend, donde el controlador se procesa los datos de entrada y se invoca correspondientes módulos de modelo para realizar las operaciones necesarias.
3. **Operación del modelo:** Una vez que se han ejecutado las funciones del modelo, se devuelve el resultado al controlador.
4. **Actualización de la vista:** El controlador envía los datos procesados desde backend al frontend y la vista se refresca con los datos recibidos.

4.2 Patrones y principios de diseño

En esta sección se describe los patrones y los principios de diseño utilizados en el desarrollo del backend que se proporciona el framework de Java Spring Boot [7], que facilitan la creación de una aplicación robusta, escalable y fáciles de mantener.

IoC (Inversión de control) [8]:

Es un principio fundamental en el marco de trabajo de Spring. Se transfiere la responsabilidad de gestionar el flujo de la aplicación y de la creación de las dependencias al framework de desarrollo. En lugar de que los objetos creen sus dependencias, Spring la inyecta automáticamente usando la técnica de inyección de dependencias.

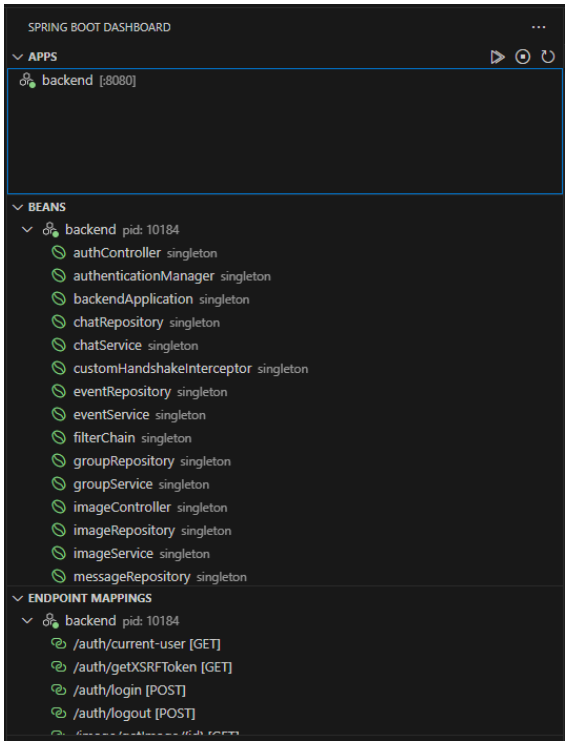

Figura 4.2: Java Spring Boot dashboard

Como se puede observar en la figura 4.2, el Spring Boot se encarga de crear y gestionar el ciclo de vida de los Beans, instancias de clases que están configuradas y administradas por el contenedor de Spring.

Inyección de dependencias [8]:

Es un componente fundamental en el diseño de aplicaciones modulares y mantenibles. Promueve el desacoplamiento entre las clases al eliminar la necesidad de que una clase conozca los detalles específicos de instanciación de sus dependencias. En este enfoque, una clase se declara explícitamente sus requerimientos, y estos son proporcionados por un contenedor gestionado por el framework de Spring.

```java
public class UserService extends InMemoryUserDetailsManager {

    @Autowired
    private UserRepository repository;
```

Figura 4.3: Ejemplo de Inyección de dependencias

Responsabilidad única [9]:

Es un principio que sugiere que una clase o un módulo debe encargarse de una sola tarea o función específica, obteniendo los beneficios del modularidad mejorado, mantenimiento simplificado y una mayor reusabilidad del código.

```java
@Repository
public interface UserRepository extends BaseRepository<User, String> {
    User findByUsername(String username);
```

```java
@RestController
@RequestMapping("/auth")
public class AuthController {
```

```java
@Service
public class ResultService {
```

Figura 4.4: Ejemplos de responsabilidad única

En Spring Boot se utiliza anotaciones como @Repository, @Service y @RestController para reforzar el concepto de la responsabilidad única, ayudan a definir las responsabilidades a nivel de componentes en la aplicación.

Patrón Singleton [10]:

Es un patrón de diseño creacional con el objetivo de asegurar que una clase tenga únicamente una instancia y ofrecer un punto de acceso global a dicha instancia. Se facilita el control centralizado, reduce los costes asociados con la creación de múltiples instancias y asegura una consistencia global en el acceso a recursos compartidos.

Patrón Builder [11]:

Es un patrón de diseño creacional que se facilita la creación de objetos complejos que requieren múltiples pasos para su construcción. En lugar de usar un constructor con numerosos parámetros o una serie de métodos de configuración, este patrón se maneja el proceso de construcción de manera secuencial. Esto permite crear diferentes variaciones del objeto final sin tener que modificar la lógica de construcción.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf((csrf) -> csrf
        .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
        .csrfTokenRequestHandler(new CsrfTokenRequestAttributeHandler()))
        .authorizeHttpRequests((authz) -> authz
        .requestMatchers(...patterns:"/auth/**").permitAll()
        .anyRequest().authenticated()
        )
        .sessionManagement((session) -> session
        .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED))
        .securityContext((securityContext) -> securityContext
        .securityContextRepository(new DelegatingSecurityContextRepository(
            new RequestAttributeSecurityContextRepository(),
            new HttpSessionSecurityContextRepository()
        )))
        .httpBasic(Customizer.withDefaults())
        .formLogin(Customizer.withDefaults());

    return http.build();
}
```

Figura 4.5: Ejemplo patrón Builder

En esta figura se proporciona parte de código para configurar la seguridad de la aplicación con Spring Security, en el que se utilizan métodos encadenados para establecer diferentes aspectos de la configuración.

4.3 Tecnología utilizada

Flutter [12]:

Flutter es un framework de desarrollo de software de código abierto desarrollado por Google para crear aplicaciones nativas de alta calidad para plataformas móviles, web y de escritorio a partir de un único código base. Se destaca por su capacidad para ofrecer un rendimiento superior y una experiencia de usuario coherente en múltiples plataformas.

Flutter utiliza un motor de renderizado propio que permite un control detallado de la apariencia y el comportamiento de una aplicación. Su función de "Hot Reload" fomenta el desarrollo ágil al permitir a los desarrolladores observar los cambios en tiempo real sin tener que reiniciar la aplicación.

El framework se basa en el lenguaje de programación Dart. Se ha elegido este framework porque ofrece una amplia gama de widgets personalizables que facilitan el diseño de interfaces de usuario adaptativas complejas.

Dart [13]:

Dart es un lenguaje de programación de código abierto desarrollado por Google para facilitar la creación de aplicaciones multiplataforma rápidas y eficientes. Se destaca por su enfoque en la productividad de los desarrolladores y el rendimiento de las aplicaciones. Es especialmente conocido por su uso en el framework Flutter, que permite el desarrollo de aplicaciones móviles, web y de escritorio utilizando una única base de código.

El lenguaje Dart combina las características de los lenguajes orientados a objetos y funcionales y está diseñado para ser fácil de aprender y utilizar. Su sintaxis es clara y expresiva, lo que permite escribir código de forma concisa y fácil de mantener. Con un sistema de tipos estático y una eficaz recogida de basura, Dart proporciona una ejecución rápida y eficaz para mejorar el rendimiento de las aplicaciones. Además, incluye herramientas integradas de prueba, depuración y compilación, que ayudan a desarrollar e implantar aplicaciones robustas y de alta calidad.

El propósito de elección de este lenguaje de programación es de desarrollar una aplicación que es compatible en la mayoría de las plataformas, facilitando su acceso a la aplicación en diferentes tipos de dispositivos.

Java Spring Boot [7]:

Spring Boot es un framework derivado del ecosistema Spring diseñado para simplificar el desarrollo de aplicaciones Java reduciendo significativamente la configuración manual. El framework se destaca por su capacidad para realizar unas configuraciones automáticas, adaptando de forma inteligente la aplicación en función de las dependencias presentes en el proyecto.

Una de las características más notables de Spring Boot es su capacidad para integrarse con servidores embebidos como Tomcat, permitiendo que las aplicaciones se ejecuten como archivos JAR independientes sin necesidad de un servidor de aplicaciones externo.

El motivo de elección de Spring Boot para el desarrollo de backend es debido a su capacidad para simplificar y acelerar el proceso de creación de aplicaciones robustas y escalables, Y mediante el uso de "starters", los paquetes preconfigurados, se puede integrar diversas características de Spring, como seguridad, bases de datos y servicios web, sin necesidad de complejas configuraciones, lo que acelera enormemente el desarrollo.

Maven [14]:

Maven es una herramienta de gestión de proyectos, principalmente para proyectos Java, que simplifica y automatiza los procesos de compilación, gestión de dependencias y configuración en el ciclo de vida de desarrollo de software. Se proporciona estructuras y configuraciones predeterminadas, reduciendo la necesidad de configuración manual y permitiendo a los desarrolladores centrarse más en la implementación de la lógica de negocio.

Se ha elegido esta herramienta por sus avanzadas características en gestión de dependencias, que facilita la inclusión y actualización de bibliotecas externas necesarias para el proyecto. Utiliza un archivo de configuración central llamado pom.xml (el modelo de objetos del proyecto), que define las dependencias y configuraciones del proyecto.

Tomcat [15]:

Tomcat es un servidor de aplicaciones de código abierto que implementa las especificaciones Java Servlet y JavaServer Pages (JSP). Se utiliza ampliamente para ejecutar aplicaciones web basadas en Java, proporcionando un entorno potente y fiable para aplicaciones empresariales y web.

Tomcat proporciona una plataforma ligera y eficiente para desplegar aplicaciones web, con un enfoque en el rendimiento y la escalabilidad. Su arquitectura modular permite integrar diversas funcionalidades mediante componentes adicionales, y su configuración flexible facilita la adaptación a diferentes necesidades y entornos.

Java [16]:

Java es un lenguaje de programación de propósito general, orientado a objetos y basado en clases. Promueve una amplia portabilidad del software que permite a los desarrolladores crear aplicaciones que se ejecutan en cualquier plataforma o sistema operativo con una máquina virtual Java (JVM) compatible.

El lenguaje Java se caracteriza por su sintaxis estructurada y su potente sistema de tipos, que facilita la detección temprana de errores y la construcción de aplicaciones robustas y seguras. Además, Java proporciona un gran número de bibliotecas y marcos de trabajo que cubren una amplia gama de aplicaciones, desde soluciones empresariales y aplicaciones web hasta desarrollo móvil y aplicaciones de servidor. Su modelo de ejecución basado en la JVM incluye mecanismos de gestión automática de la memoria mediante la recolección de basura, lo que contribuye a mejorar la estabilidad y eficacia de las aplicaciones.

Postgresql [17]:

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto conocido por su solidez y flexibilidad. Cumple estándares SQL y ofrece características avanzadas como la posibilidad de definir tipos de datos personalizados, crear funciones propias y la posibilidad de ampliar su funcionalidad con extensiones. Su compatibilidad con los formatos de datos JSON y JSONB ayuda a manejar eficazmente datos semiestructurados, mientras que su función de transacciones ACID garantiza la integridad y coherencia de los datos en todas las situaciones.

Además, PostgreSQL proporciona sofisticadas herramientas para optimizar el rendimiento, incluyendo varios tipos de índices y opciones de replicación para garantizar una alta disponibilidad. Su arquitectura permite tanto la escalabilidad vertical como horizontal para adaptarse a las distintas demandas de carga y volúmenes de datos.

Visual Studio Code [18]:

Visual Studio Code es un editor de código fuente desarrollado por Microsoft, conocido por su flexibilidad y sus avanzadas funciones. Proporciona una interfaz intuitiva y personalizable que permite a los desarrolladores adaptar su entorno de trabajo a sus necesidades específicas. Es compatible con una amplia gama de lenguajes de programación y ofrece un gran número de extensiones en el mercado, por lo que puede manejar fácilmente proyectos de todos los tamaños y complejidades.

Además, se integra funciones clave como el control de versiones a través de Git, herramientas avanzadas de depuración y un terminal integrado para agilizar los flujos de trabajo de desarrollo.

Github [19]:

GitHub es una plataforma de desarrollo colaborativo que facilita la gestión de proyectos de software utilizando el sistema de control de versiones distribuido Git. Es una herramienta esencial para la colaboración tanto en proyectos de código abierto como privados.

La plataforma ofrece una serie de características avanzadas, incluidos repositorios de código fuente, herramientas de seguimiento de problemas y pull requests para revisar y fusionar cambios de manera eficiente. Su intuitiva interfaz web y su capacidad para facilitar la colaboración y el intercambio de código entre desarrolladores simplifican los flujos de trabajo de desarrollo de software y mejoran la coordinación de proyectos complejos.

Android Studio [20]:

Android Studio es el entorno de desarrollo integrado para aplicaciones móviles en el sistema operativo Android, Se proporciona una serie de herramientas y características diseñadas para facilitar su creación, depuración y optimización.

Entre sus características destacadas se incluyen un editor de código inteligente con autocompletado y refactorización, un emulador de Android altamente configurable, y herramientas para el diseño de interfaces de usuario mediante un editor visual.

En este proyecto se ha utilizado la función de emulador para evaluar y verificar el comportamiento de la aplicación en el entorno de Android.

Postman [21]:

Postman es una herramienta para desarrollo de API (Application Programming Interfaces), que ayuda a crear, probar y documentar servicios web. Es muy utilizada por los desarrolladores para enviar peticiones HTTP a los servidores y analizar las respuestas para validar el rendimiento de las API y depurar eficazmente los errores.

Postman ofrece una interfaz intuitiva que permite configurar peticiones con diferentes métodos HTTP, cabeceras y parámetros, y mostrar las respuestas en diversos formatos, como JSON, XML y HTML, así como funciones avanzadas como la automatización de pruebas mediante scripts, la gestión de colecciones de peticiones y la generación de documentos interactivos.

Ha sido utilizado principalmente en la fase inicial de desarrollo de backend, para la comprobación del funcionamiento del Api implementado, puesto que no está creado en frontend los correspondientes interfaces para realizar estas solicitudes.

Trello [1]:

Trello es una herramienta de gestión de proyectos basada en la web. Utiliza un enfoque visual para organizar tareas y proyectos. Su estructura se basa en tableros, listas y tarjetas, que permiten a los usuarios desglosar y visualizar el trabajo de manera clara y eficiente.

Se ha utilizado para listar todas las tareas que debe realizar de este trabajo y es el elemento clave para aplicar la metodología ágil Kanban.

Draw io [22]:

Es una herramienta en línea gratuita para la creación de diagramas y gráficos. Su interfaz intuitiva permite a los usuarios diseñar una amplia variedad de diagramas, incluyendo diagramas de flujo, organigramas, diagramas de redes, y mapas mentales, entre otros.

Se ha utilizado para crear los diagramas de contexto, de casos de usos, de datos y el gráfico que representa la arquitectura del sistema.

Google forms [23]:

Es una herramienta en línea que permite crear formularios y encuestas personalizadas de manera sencialla y eficiente. Es parte de la suite de aplicaciones de Google.

Se ha utilizado para las encuestas en apartado de pruebas.

CAPÍTULO 5

Desarrollo de la solución propuesta

En este capítulo se describe detalladamente la implementación de la solución propuesta, explicando el flujo de la aplicación según la arquitectura diseñada la interacción entre frontend y backend. También se analizan las dificultades y problemas encontrados durante el proceso de aprendizaje de desarrollo con las soluciones y decisiones tomadas para resolver estos desafíos.

5.1 Estructura de Base de Datos

En esta sección se presenta la estructura de base de datos utilizado para el proyecto, como se puede observar en la figura 5.1.

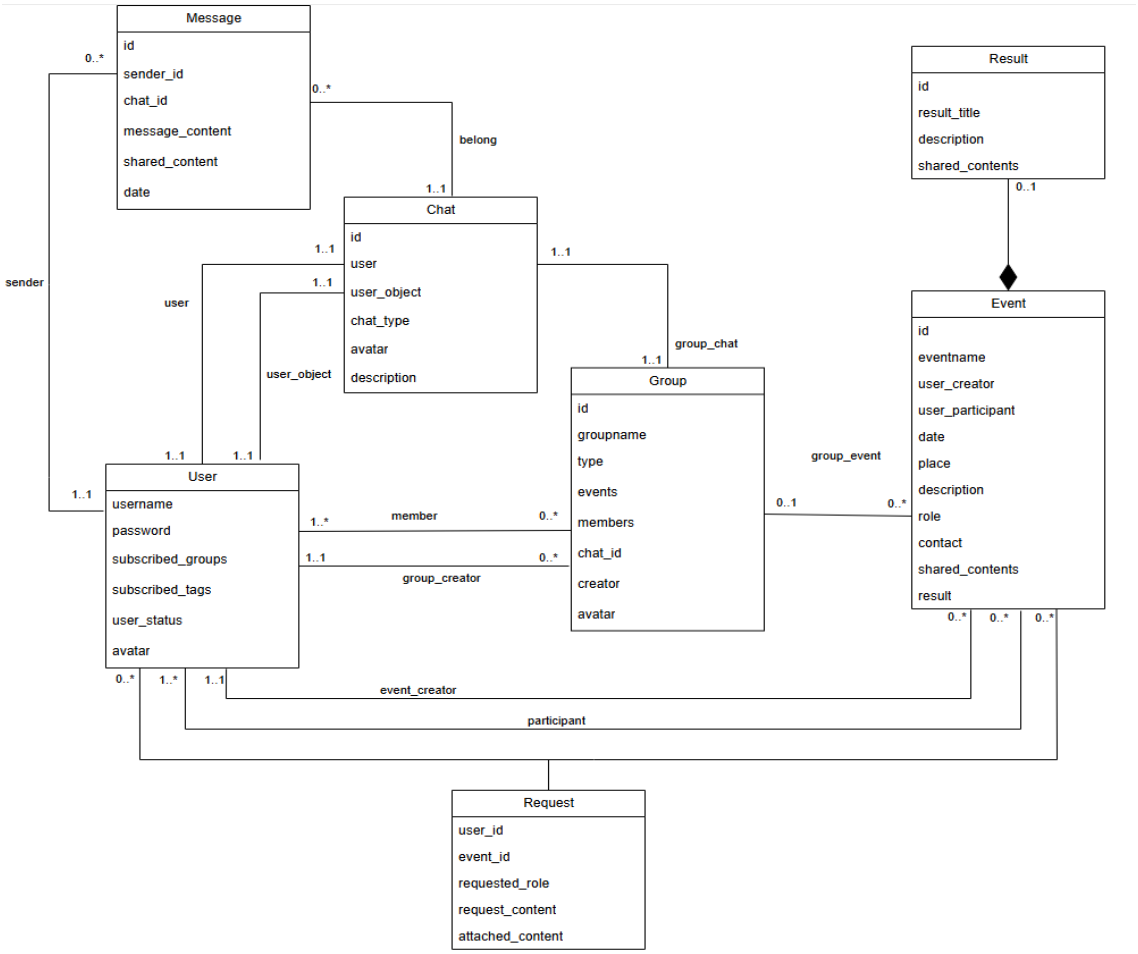

Figura 5.1: Diagrama de Datos

La estructura de datos de la aplicación está basada en un modelo relacional, donde las entidades y sus relaciones se organizan en tablas dentro de una base de datos SQL. Cada tabla representa una entidad clave del sistema, y las relaciones entre estas entidades están definidas mediante claves ajenas.

Cada entidad representa lo siguiente:

User:

El usuario registrado en el sistema, se almacena los datos como las credenciales de acceso.

Group:

El grupo con miembros que tenga el mismo interés, contiene la informaciones como el nombre de grupo, su creador y la lista de miembros que pertenece.

Event:

El evento gestionado por el sistema, se almacena informaciones como la descripción, el lugar de manifestación, fecha y la lista de participantes.

Chat:

El espacio de conversación entre dos o varios usuarios.

Message:

El mensaje intercambiado entre usuarios, se almacena informaciones como su contenido, fecha y el emisor.

Request:

La solicitud de participación a un evento, contiene informaciones de los datos necesarios pedido en formulario de solicitud.

Result:

La experiencia compartida por los usuarios del evento, puede contener imágenes u videos.

Relaciones entre las entidades:

- **event_creator (User-Event):** Un usuario puede ser creador de cero o muchos eventos y un evento solo puede ser creador por un usuario.
- **participant (User-Event):** Un usuario puede ser participante de cero o muchos eventos y un evento puede tener uno o muchos participantes.
- **member (User-Group):** Un usuario puede ser miembro de cero o muchos grupos, un grupo tiene uno o muchos miembros.
- **group_creator (User-Group):** Un usuario puede ser creador de cero o muchos grupos, un grupo puede ser creado por un usuario.
- **user y user_object (User-Chat):** Un usuario es remitente de conversación de un chat con otro usuario objetivo, un chat tiene un usuario remitente y un usuario objetivo.
- **sender (User-Message):** Un usuario puede ser emisor de cero o muchos mensajes y un mensaje tiene un emisor.
- **group_chat (Group-Chat):** Un grupo puede tener un chat y un chat pertenece a un grupo.
- **group_event (Group-Event):** Un grupo puede tener cero o muchos eventos, y un evento puede pertenecer de cero o un grupo.
- **belong (Message-Chat):** Un mensaje pertenece a un chat y un chat puede tener cero o muchos mensajes.
- **Event-Result:** Un resultado pertenece a un evento, no tiene sentido su propia existencia.

5.2 Desarrollo de Backend

Una vez presentada la estructura de base de datos, esta sección se centra en la implementación del backend. La primera parte para desarrollar es el modelo, y este se divide en dos capas, la capa de persistencia y la capa de la lógica de negocio.

5.2.1 Persistencia

En la capa de persistencia existen dos componentes, las clases que corresponden con las entidades en el base de datos y las clases de repositorios para manejar las operaciones CRUD (Create, Read, Update y Delete) sobre los datos.

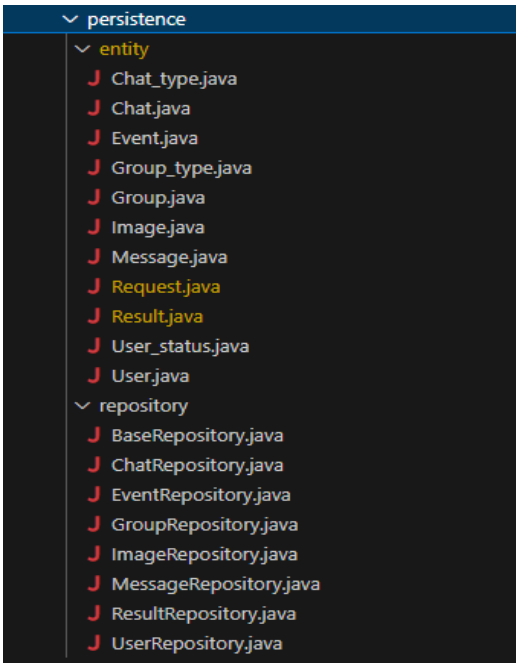

Figura 5.2: Clases de capa de persistencia

Entity

La clase de entidad se mapean directamente a una tabla en la base de datos y cada instancia de la entidad corresponde a una fila en esa tabla. Sus propiedades mapean directamente a las columnas de la tabla. Sabiendo esto podemos utilizar las funcionalidades de las clases de las anotaciones del Spring Boot.

```java
@Entity
@Table(name = "\"user\"")
public class User {

    @Id
    @Column(name = "username")
    private String username;

    @Column(name = "password")
    private String password;

    @ElementCollection
    @CollectionTable(name = "user_subscribed_groups", joinColumns = @JoinColumn(name = "username"))
    @Column(name = "subscribed_groups")
    private List<Integer> subscribed_groups;

    @ElementCollection
    @CollectionTable(name = "user_subscribed_tags", joinColumns = @JoinColumn(name = "username"))
    @Column(name = "subscribed_tags")
    private List<String> subscribed_tags;

    @Enumerated(EnumType.STRING)
    @Column(name = "user_status")
    private User_status user_status;

    @Column(name = "avatar")
    private byte[] avatar;
```

Figura 5.3: Clase entidad User

Se puede observar las anotaciones como:

@Entity, declara que esta clase es una entidad en base de datos;

@Table, se mapea con la tabla en la base de datos con el nombre introducido;

@Id, indica que el atributo es la clave primaria;

@Column, se mapea el atributo con la columna con el nombre introducido;

@ElementCollection se utiliza para declarar que el atributo es una colección de elementos y

@CollectionableTable se crea una nueva tabla con el nombre introducido en variable name y

@JoinColumn se elige un atributo de la tabla original para utilizarlo como clave primaria o parte de clave primaria junto con el dato de atributo que pertenece a la lista de elementos;

@Enumerated indica que el variable es de tipo enumerado.

Sobre los atributos de tipo lista de elementos, aunque se puede crearlo manualmente en base de datos Postgresql como tipos de datos primitivos, realmente se procesa y se guarda en una tabla aparte. Como podemos ver en la figura 5.4, el atributo subscribed_groups se guarda en otra tabla autogenerada.

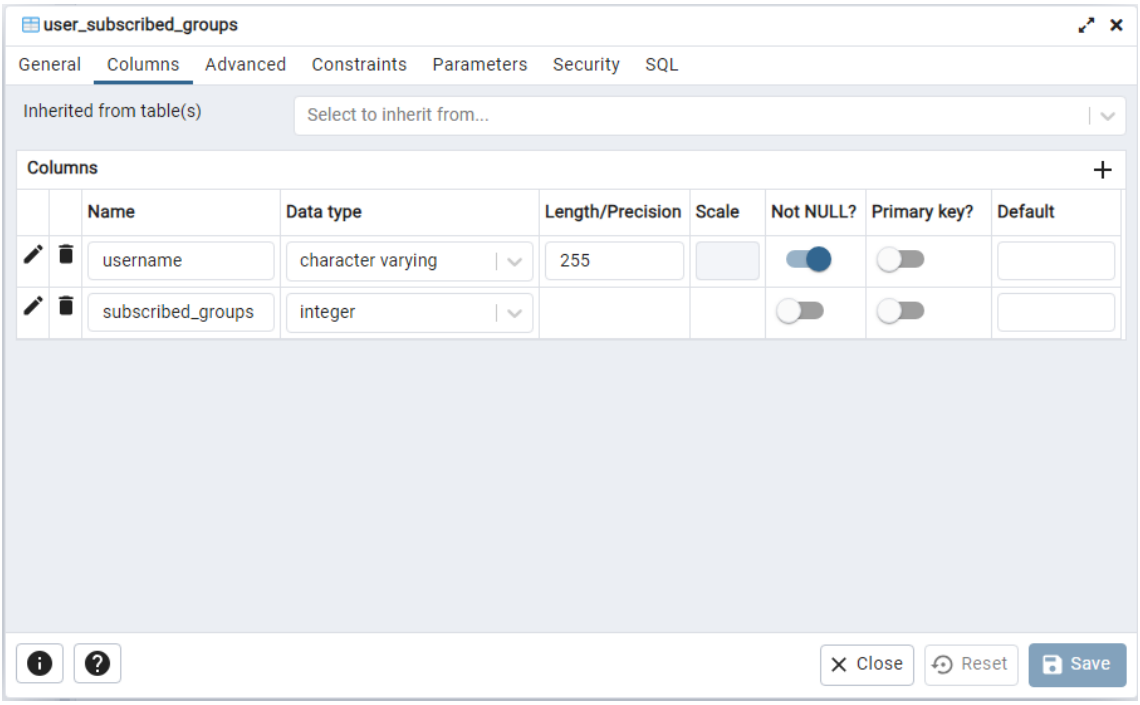

Figura 5.4: Tabla user_subscribed_groups

Con estas anotaciones, Spring Boot es capaz de generar automáticamente en base de datos, las tablas y las propiedades correspondientes.

La principal dificultad encontrado en esta parte es aprendizaje del uso de las anotaciones, especialmente para tratamiento de los variables de tipo colección, y también la falta de conocimientos en los detalles concretos de las herramientas, por ejemplo, es necesario poner el símbolo \ en el nombre de tabla como se puede observar en la figura 5.3, para nombre específicos como user, group, etc.

Repository

En la clase de repositorio se maneja las operaciones sobre los datos. En este proyecto se ha utilizado una interfaz proporcionada por Spring Data JPA que simplifica el acceso a datos en una base de datos relacional. Se ofrece una serie de métodos predeterminados para operaciones comunes como save(), findById(), findAll(), etc.

También se puede crear consultas personalizada añadiendo @Query y la sentencia de SQL. Se puede utilizar la interfaz directamente sin crear la clase de la implementación, porque Spring Data JPA se genera de manera automática una implementación de la interfaz en tiempo de ejecución.

```java
@Repository
public interface UserRepository extends JpaRepository<User, String> {
    User findByUsername(String username);

    @Query("select u from User u where u.user_status = 'ONLINE'")
    List<User> findOnlineUsers();
```

Figura 5.5: Clase UserRepository

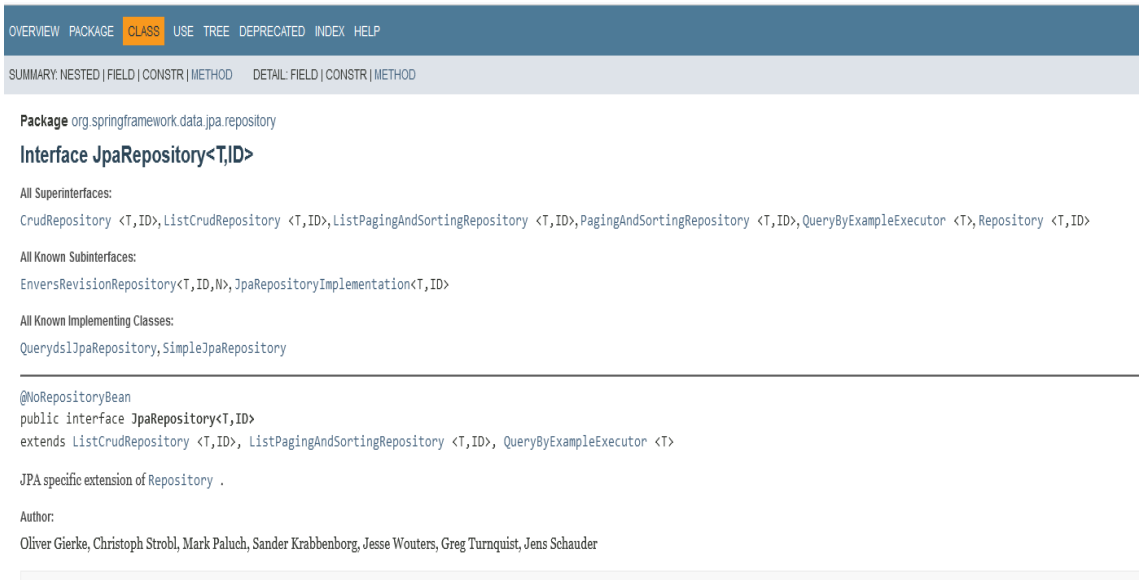

Figura 5.6: Interface JpaRepository [24]

5.2.2 Lógica de negocio

La capa de la lógica de negocio es una parte esencial del componente modelo que se encarga de encapsular y gestionar las reglas y procesos de negocio específicos de la aplicación. Su propósito principal es implementar la lógica que define cómo se deben realizar las operaciones sobre los datos. En Spring Boot la anotación para clases de servicios es @Service

```java
@Service
public class UserService extends InMemoryUserDetailsManager {

    @Autowired
    private UserRepository repository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException{
        User user = repository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException(msg:"User not found");
        }
        return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), new ArrayList<>());
    }

    public User getUsers(String username) {
        return repository.findByUsername(username);
    }
    public List<User> getAllUsers() {
        return repository.findAll();
    }

    public void insertUser(User user) {
        System.out.println("Inserting user: " + user.toString());
        repository.save(user);
    }
```

Figura 5.7: Clase UserService

La anotación @Autowired se utiliza para inyectar automáticamente una instancia de UserRepository, que es creado y manejado por Spring Boot siguiendo el patrón de diseño singleton. Al utilizar esta anotación en la clase de servicio, el framework se encarga de proporcionar la instancia del repositorio necesaria para interactuar con la base de datos.

Se puede observar la clase UserService se extiende de clase InMemoryUserDetailManager y se sobrescribe el método de loadUserByUserName, esto es necesario para la integración de Spring Security que se explica en apartado posterior.

5.2.3 Controlador

La clase controlador es fundamental para manejar la interacción entre la vista y el modelo. Cuando el usuario realiza una acción sobre la interfaz del frontend, este se envía solicitudes correspondientes al controlador situado en backend, se interpreta la petición, se interactúa con el modelo para que este procesa la lógica de negocio necesaria y se obtiene o se modifica los datos. Se devuelve los resultados al frontend y este se actualiza la interfaz para representar al usuario la respuesta de la solicitud.

En Spring Boot, los controladores se definen mediante la anotación @Controller. Los métodos de los controladores se asocian con rutas específicas de la aplicación a través de anotaciones como @RestMapping, @GetMapping o @PostMapping. Estos métodos reciben peticiones HTTP y responden con una vista que suele estar renderizada con un motor de plantillas como Thymeleaf [27] o JSP [28].

Se puede observar en la figura 5.8, en lugar de usar @Controller, la anotación utilizada es @RestController. La clase de RestController es una extensión especializada de los controladores estándar, diseñada específicamente para desarrollar APIs RESTful, a diferencia de un controlador típico, se devuelve directamente datos en formatos como JSON o XML, que son tipos datos más simples de procesar. Se ha utilizado @RequestMapping para indicar la ruta común para todos los métodos de la clase, y cada método añade su propio sufijo de ruta.

```java
@RestController
@RequestMapping("/auth")
public class AuthController {

    private final AuthenticationManager authenticationManager;

    public AuthController(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }

    @GetMapping("/getXSRFToken")
    public CsrfToken getXSRFToken(HttpServletRequest request) {
        return (CsrfToken) request.getAttribute(CsrfToken.class.getName());
    }

    @PostMapping("/login")
    public ResponseEntity<Void> login(@RequestBody User user, HttpServletRequest request) {
        System.out.println("Login request received: " + user.getUsername() + " - " + user.getPassword());
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword())
        );
        System.out.println("Authentication successful: " + authentication.getName());
        SecurityContextHolder.getContext().setAuthentication(authentication);
        System.out.println("Authentication context set: " + SecurityContextHolder.getContext());

        HttpSession session = request.getSession(create:true);
        session.setAttribute(HttpSessionSecurityContextRepository.SPRING_SECURITY_CONTEXT_KEY, SecurityContextHolder.getContext());
        return ResponseEntity.ok().build();
    }

    @PostMapping("/logout")
    public ResponseEntity<Void> logout(HttpServletRequest request) {
        HttpSession session = request.getSession(create:false);
```

Figura 5.8: Clase AuthController

5.2.4 Spring Security

Spring Security [25] es un framework de seguridad para aplicaciones basada en la plataforma Java, que proporciona una gran variedad de funcionalidades para proteger aplicaciones tanto a nivel de autenticación como de autorización. El principal objetivo de la integración en este proyecto es de asegurar que solo los usuarios autenticados y autorizados puedan acceder a los recursos de la aplicación. Para ello, Spring Security ofrece soporte para múltiples métodos de autenticación, como autenticación basada en tokens (como JWT[1]), OAuth2[2], etc. El método elegido es mediante el uso del nombre de usuario y de contraseña, siendo que otros métodos de autenticación son considerados para incluir en futuras versiones.

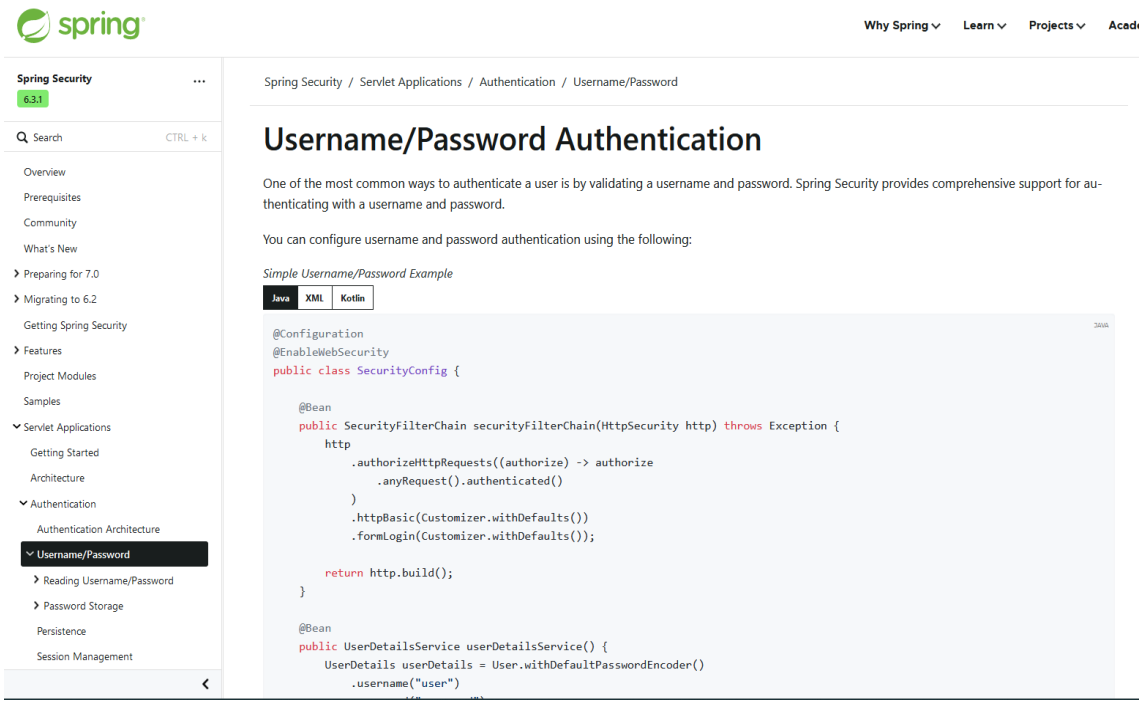

Figura 5.9: Spring Username/Password Authetication

Además, Spring Security integra mecanismos de protección contra amenazas comunes en aplicaciones web, como la falsificación de solicitudes en sitios cruzados (CSRF)[3] y la inyección de scripts (XSS)[4].

Para la implementación necesita crear una clase de configuración:

[1]JWT (JSON Web Token): un estándar abierto que se utiliza para transmitir información de manera segura entre dos partes, por ejemplo, un servidor y un cliente, en forma de un objeto JSON.

[2]OAuth2: un protocolo de autorización que permite a las aplicaciones obtener acceso limitado a recursos protegidos en nombre de un usuario sin compartir credenciales.

[3]CSRF (Cross-Site Request Forgery): un tipo de ataque en el que atacante engaña a un usuario autenticado realice acciones no deseadas en las aplicaciones web que esta autenticado.

[4]XSS (Cross-site Scripting): un tipo de ataque que busca vulnerabilidades en páginas web e inyecta scripts maliciosos.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private final UserService userService;

    public SecurityConfig(UserService userService) {
        this.userService = userService;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf((csrf) -> csrf
            .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
            .csrfTokenRequestHandler(new CsrfTokenRequestAttributeHandler()))
            .authorizeHttpRequests((authz) -> authz
            .requestMatchers(...patterns:"/auth/**").permitAll()
            .anyRequest().authenticated()
            )
            .sessionManagement((session) -> session
            .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED))
            .securityContext((securityContext) -> securityContext
            .securityContextRepository(new DelegatingSecurityContextRepository(
                new RequestAttributeSecurityContextRepository(),
                new HttpSessionSecurityContextRepository()
            )))
            .httpBasic(Customizer.withDefaults())
            .formLogin(Customizer.withDefaults());

        return http.build();
    }
```

Figura 5.10: Clase SecurityConfig

@Configuration declara que la clase anotada se actúan como fuente de definiciones de beans y permiten a los desarrolladores configurar objetos que son gestionados por el contenedor de Spring.

Por otro lado, @EnableWebSecurity habilita la configuración de seguridad, se activa la funcionalidad de Spring Security para gestionar la autenticación y autorización de solicitudes HTTP. Permite a los desarrolladores personalizar las políticas de seguridad utilizando el método de filterChain como se puede observar en la figura anterior.

Se explica con detalle las reglas más importantes aplicadas en este proyecto:

```
-   .csrf((csrf) -> csrf
-   .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
-   .csrfTokenRequestHandler(new CsrfTokenRequestAttributeHandler()))
-   .authorizeHttpRequests((authz) -> authz
-   .requestMatchers("/auth/**").permitAll()
-   .anyRequest().authenticated()
```

En estas líneas indica al framework para que se crea tokens para defender los ataques CSRF, guardando su información en la cookie, y se crea un repositorio para guardar esta cookie que contiene el token, este token se utiliza en posteriores solicitudes para garantizar la autenticidad de la identidad del usuario.

Por otra parte, se aplica la restricción de que autoriza el acceso de todos los usuarios a todos los métodos del controlador con la ruta que /auth. Se puede observar en la figura 5.8, son métodos para realizar las acciones de la autenticación, y la siguiente línea indica que una vez el usuario esta autenticado se le autoriza realizar peticiones a otros controladores para solicitar otras funcionalidades del sistema.

```
-   .securityContext((securityContext) -> securityContext
-   .securityContextRepository(new DelegatingSecurityContextRepository(
-                   new RequestAttributeSecurityContextRepository(),
-                   new HttpSessionSecurityContextRepository()
-               )))
```

En estas líneas se configura la manera de almacenamiento del contexto de seguridad creado para el usuario autenticado, creando una instancia de DelegatingSecurityContextRepository. Este repositorio se encarga de manejar el contexto de seguridad de las solicitudes HTTP, en el que incluye la información sobre la autenticación y autorización del usuario.

```
-    .sessionManagement((session) -> session
-    .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED))
```

En estas líneas declara que la política de creación de la sesión es cuando fuese necesario.

Por otra parte, si se desea personalizar algunos aspectos de la gestión de la seguridad proporcionada por Spring Security, se puede crear instancias necesarias para el framework, utilizando métodos anotados con @Bean. Esto permite cambiar las configuraciones por defecto y se sustituye por las que están definidas en estos métodos para crear los beans personalizados.

Como ejemplo, se puede observar en la figura 5.11, se ha modificado las clases como AuthenticationManager para que se utilice la clase propio de servicio y una clase cifrador de las credenciales específico.

```java
@Bean
public AuthenticationManager authenticationManager(UserService userService, PasswordEncoder passwordEncoder) {
    DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(userService);
    authenticationProvider.setPasswordEncoder(passwordEncoder);

    return new ProviderManager(authenticationProvider);
}

@Bean
public UserDetailsService userDetailsService() {

    return userService;
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Figura 5.11: Métodos para personalizar Spring Security

Sin embargo, el tipo de estas clases personalizados tiene que ser el tipo requerido por el framework, exponiendo un caso concreto, la clase de UserService, es necesario que extienda de la clase InMemoryUserDetailManager y sobrescribir el método loaduserByUsername.

```java
@Service
public class UserService extends InMemoryUserDetailsManager {

    @Autowired
    private UserRepository repository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException{
        User user = repository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException(msg:"User not found");
        }
        return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), new ArrayList<>());
    }
```

Figura 5.12: Detalles UserService para Spring Security

5.3 Desarrollo de Frontend

Esta sección corresponde a la Vista de la arquitectura MVC. Es el componente encargado de la presentación de datos al usuario. Se trata de la parte visual de la aplicación, que se proporciona una interfaz para que los usuarios puedan ver e interactuar. No contiene lógica de negocio y solo se actúa como una representación pasiva del modelo.

Uno de los requerimientos de este proyecto es que la aplicación está disponible para diferentes tipos de dispositivos, y por este motivo se ha optado el framework Flutter para el desarrollo del frontend, programando un código base, y el framework se genera diferentes versiones para diferentes plataformas, proporcionando una gran variedad de widget para la creación de una interfaz intuitiva y agradable.

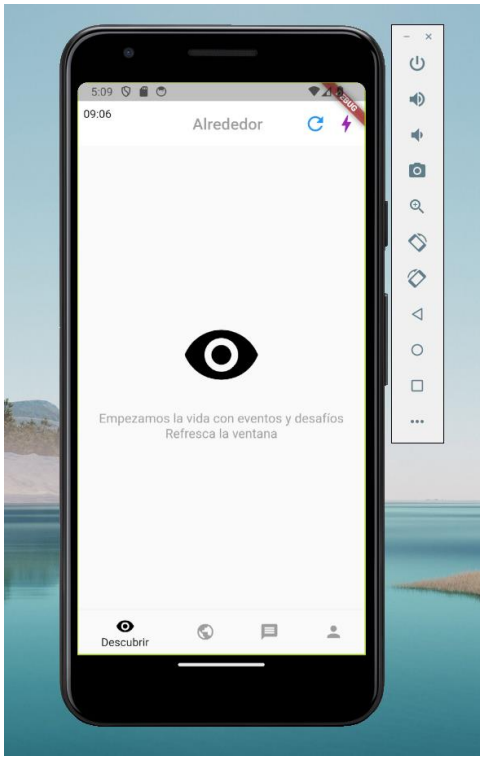

Figura 5.13: Interfaz de Usuario Android

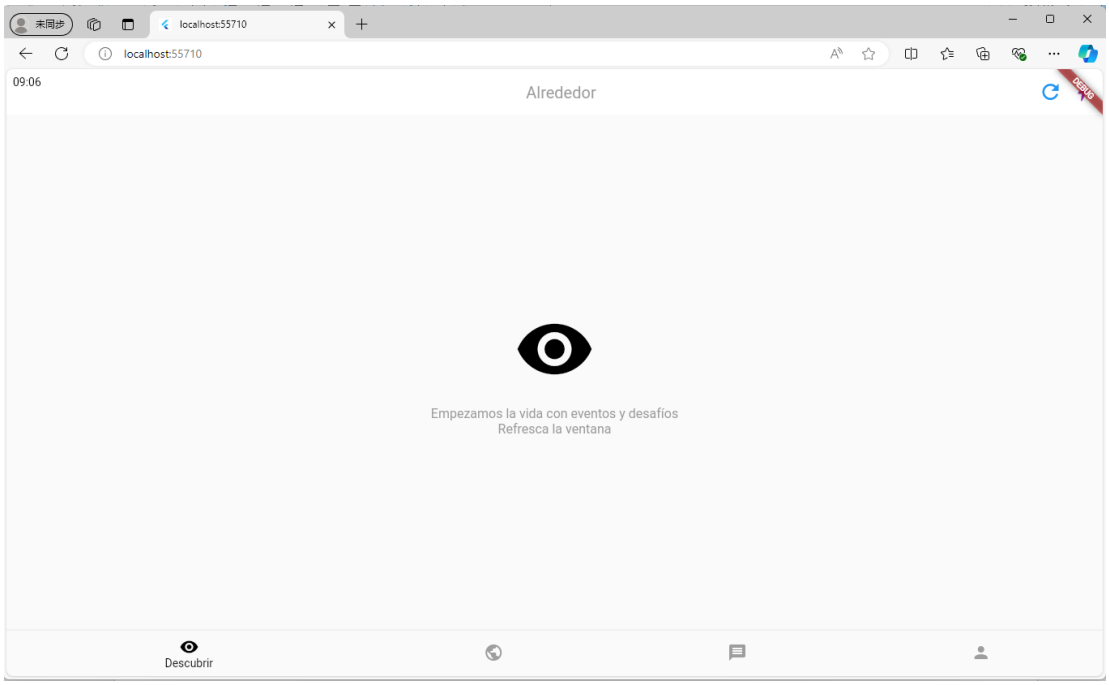

Figura 5.14: Interfaz de Usuario Web

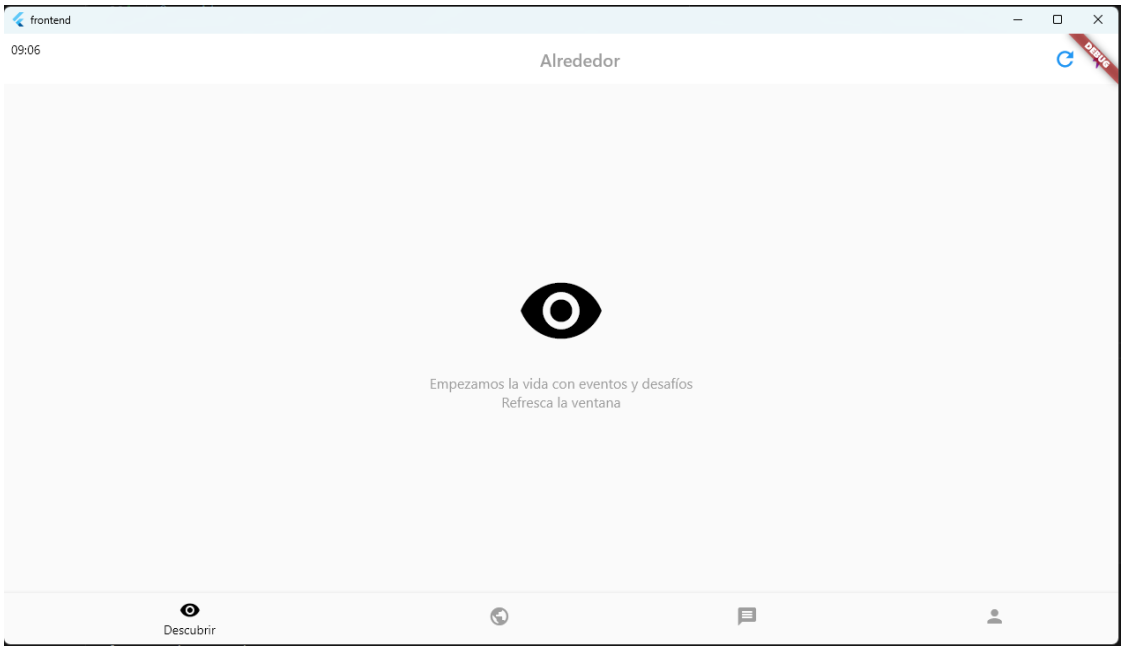

Figura 5.15: Interfaz de Usuario Windows

En esta parte, el código se centra principalmente en la solicitud de datos procesados en el backend, y utiliza los resultados obtenidos para actualizar la vista. Tal como se ilustra en la figura 5.16, se define un método asíncrono denominado fetchEvents, que realiza una petición al API RESTful correspondiente. Una vez que se reciben los datos, se construye un widget de tipo Listview para representar la información en la interfaz de usuario. Este enforque permite que los datos procesados en el backend se reflejen de manera dinámica en la vista de usuario.

```dart
class EventPage extends StatelessWidget {
  Future<List<Map<String, dynamic>>> fetchEvents() async {
    final response =
        await http.get(Uri.parse('http://localhost:8080/event/getEvents'));

    if (response.statusCode == 200) {
      return List<Map<String, dynamic>>.from(json.decode(response.body));
    } else {
      throw Exception('Failed to load events');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Events Info'),
      ), // AppBar
      body: Center(
        child: FutureBuilder<List<Map<String, dynamic>>>(
          future: fetchEvents(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return CircularProgressIndicator();
            } else if (snapshot.hasError) {
              return Text('Error: ${snapshot.error}');
            } else if (snapshot.hasData) {
              final events = snapshot.data!;
              return ListView.builder(
                itemCount: events.length,
                itemBuilder: (context, index) {
                  final event = events[index];
```

Figura 5.16: Clase EventPage

De esta manera, se obtienen las interfaces sobre las que el usuario pueda interactuar con el sistema y utilizar sus funcionalidades. A continuación, se muestran parte de las interfaces de la aplicación:

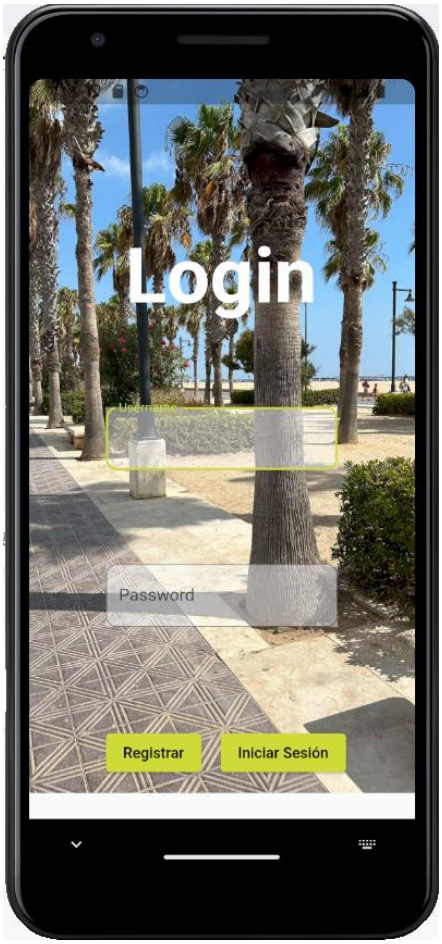

Figura 5.17: Interfaz Login

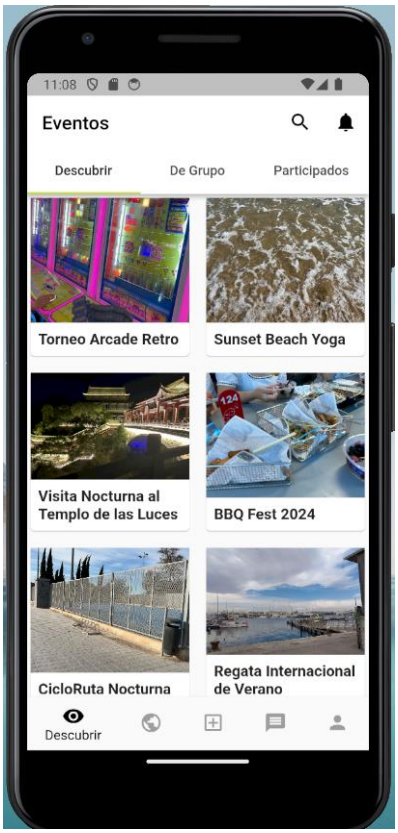

Figura 5.18: Interfaz Lista de eventos

Figura 5.19: Interfaz Detalle de eventos

Figura 5.20: Interfaz Inscripción del evento

Figura 5.21: Interfaz Canal compartir eventos

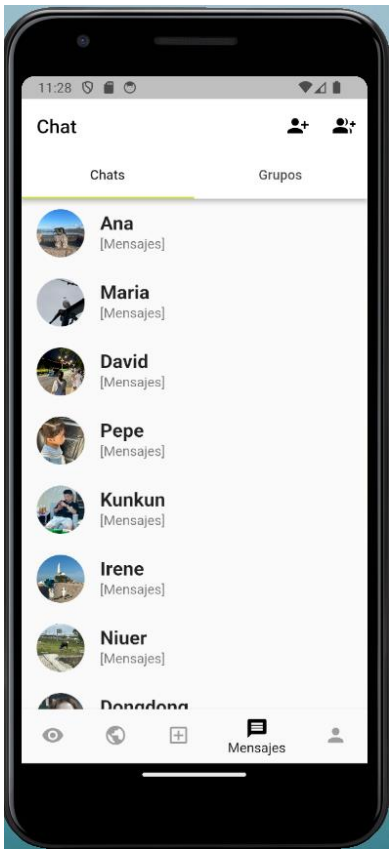

Figura 5.22: Interfaz Chat

Figura 5.23: Interfaz Chat con usuario

Figura 5.24: Interfaz Crear evento

Figura 5.25: Interfaz Notificaciones

CAPÍTULO 6

Implantación

En este capítulo se describe los pasos necesarios para la implantación del sistema, desde la preparación de entorno hasta la puesta en marcha final.

1. Preparación de entorno:

La primera tarea a realizar es la preparación de entorno. Para este proyecto, se recomienda alojar el servicio de backend en un servidor adecuado, que puede ser un servidor físico, una máquina virtual en la nube, o un contenedor de Docker. Es esencial configurar el servidor de aplicaciones (Tomcat en este caso), gestionar las dependencias de Java, e instalar y configurar el base de datos Postgresql, incluyendo la creación de los usuarios necesarios. De la misma manera, es crucial configurar el servidor web adecuado la versión web del frontend (puede utilizar el servidor Apache [26]).

2. Empaquetado y despliegue del backend de Spring Boot

El siguiente paso en el proceso de implantación consiste en empaquetar la aplicación de Spring Boot en un archivo ejecutable JAR (Java Archive). Este archivo contiene toda la lógica del backend, así como las dependencias y los recursos necesarios para la correcta ejecución de la aplicación. Una vez empaquetada la aplicación, el siguiente procedimiento consiste en desplegar el archivo JAR en el servidor configurado. En esta etapa, se procede a establecer las variables de entorno, las conexiones seguras a la base de datos, y se ajustan las reglas de firewall para permitir el acceso al backend desde frontend.

3. Compilación y despliegue del Frontend en Flutter

El frontend desarrollado en Flutter se compila en el formato apropiado según la plataforma de destino. Para las aplicaciones móviles, esto implica la generación de un archivo APK para Android o un archivo IPA para iOS. En el caso de aplicaciones web, Flutter produce un conjunto de archivos HTML, CSS, y Javascript que deben ser desplegados en un servidor web. Las versiones para móviles se pueden distribuir por tienda de aplicaciones, como Google Play [] y App Store. En el caso de versión web, los archivos generados se suben al servidor web.

4. Configuración de comunicación entre frontend y backend

Una parte importante del proceso de implantación es la configuración de la comunicación entre el frontend y el backend. Esta comunicación se realiza a través de APIs RESTful, en las cuales el backend expone puntos finales (endpoints, que son las rutas específicas de los métodos de los controladores que se ha explicado en apartados anteriores) que el frontend se utiliza para obtener o enviar los datos. Es fundamental garantizar los endpoints estén adecuadamente documentados, además es esencial la configuración del CORS (Cross-Origin Resource Sharing)[1] en el backend para permitir que el frontend acceda a los recursos desde diferentes dominios.

5. Pruebas

Con todo el sistema desplegado, se procede a realizar pruebas para garantizar que su funcionamiento es conforme a lo esperado. Estas pruebas incluyen la verificación de la conectividad entre el frontend y el backend, así como la confirmación de que todas las funciones de la aplicación se operan de manera correcta. Se realizarán las pruebas de aceptación con los usuarios finales para identificar y corregir cualquier defecto.

6. Mantenimiento

El mantenimiento regular de la aplicación es esencial para garantizar su durabilidad y pertinencia a lo largo de tiempo. Este proceso se incluye la implementación de actuaciones de seguridad, la mejora de las funcionalidades basadas en la experiencia y las opiniones de los usuarios y la realización de ajustes en la infraestructura para optimizar el rendimiento.

[1]CORS: Es un mecanismo de seguridad implementado en los navegadores web que permite a las aplicaciones web realizar solicitudes a recursos que se encuentran en un dominio diferente.

CAPÍTULO 7

Pruebas

En este capítulo se describen los procesos llevados a cabo para realizar las pruebas sobre el sistema. En particular, se detallan las pruebas de integración, que son utilizadas para verificar el correcto funcionamiento de la interoperabilidad entre diferentes módulos del sistema. Además, se explican las pruebas de aceptación, realizadas con usuarios finales para garantizar que el sistema cumple con los requisitos establecidos y satisface las expectativas.

7.1 Pruebas de Integración

La prueba de integración es un proceso esencial en el ciclo de desarrollo de software, cuyo propósito es verificar la correcta interacción entre los diversos módulos y componentes que conforman una aplicación. Se garantiza que estos módulos, al operar conjuntamente, se comporten de manera esperado y sin errores.

Este tipo de prueba es fundamental para identificar problemas relacionados con la comunicación, compatibilidad y dependencias entre los componentes, asegurando que el sistema funcione de manera consistente y fiable en un entorno de producción.

En el contexto de este proyecto, se puede clasificar las pruebas realizadas sobre el componente controlador como pruebas de integración, puesto que es el responsable de gestionar las interacciones entre la vista y el modelo. Al realizar estas pruebas, no solamente se evalúa la funcionalidad intrínseca de controlador, sino también su integración con otros componentes del sistema, tales como los servicios y los repositorios. De este modo, se garantiza que el controlador sea capaz de procesar solicitudes de manera adecuada, interactuar correctamente con las capas de lógica de negocio y acceder a los datos requeridos.

La herramienta utilizada para realizar estas pruebas es el Postman [21], una herramienta que permite diseñar, probar y documentar APIs. A continuación, se detalla el proceso para la creación de pruebas de integración utilizando esta herramienta, incluyendo como ejemplo los métodos del controlador para realizar la funcionalidad de autenticación en el sistema.

Para realizar la funcionalidad de autenticación se utiliza dos métodos, el método getXSRFToken, que solicita al backend un token CSRF y el método Login que pide este token por la configuración de seguridad y se genera una sesión para el usuario.

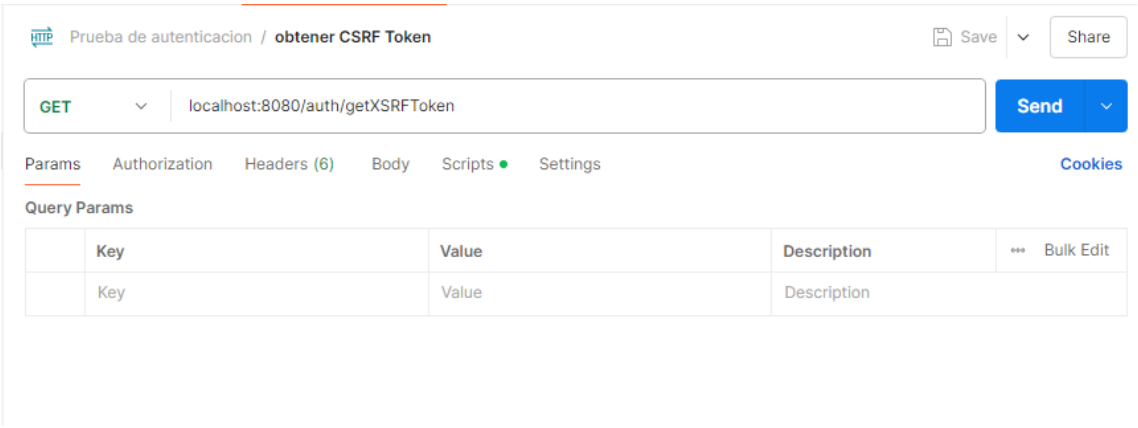

Figura 7.1: Prueba de API obtener CSRF Token

Figura 7.2: Script configuración variable csrfToken

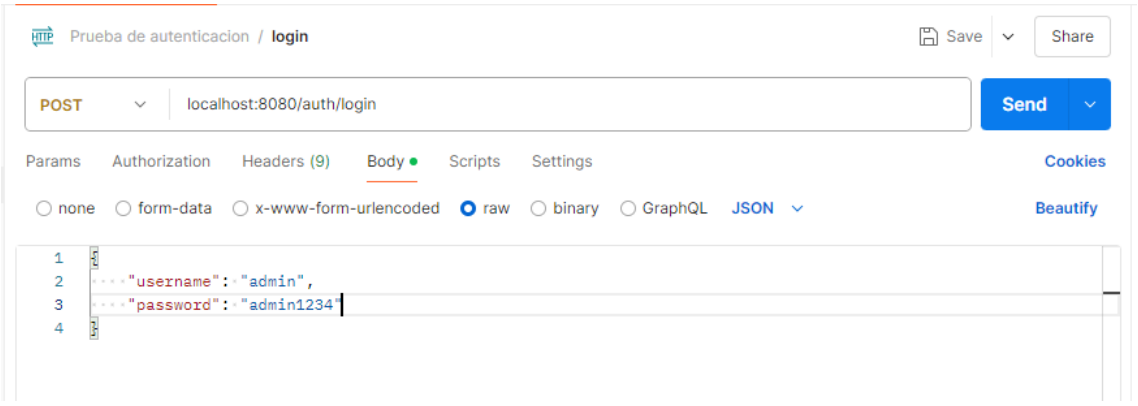

Figura 7.3: Cuerpo de prueba de API login

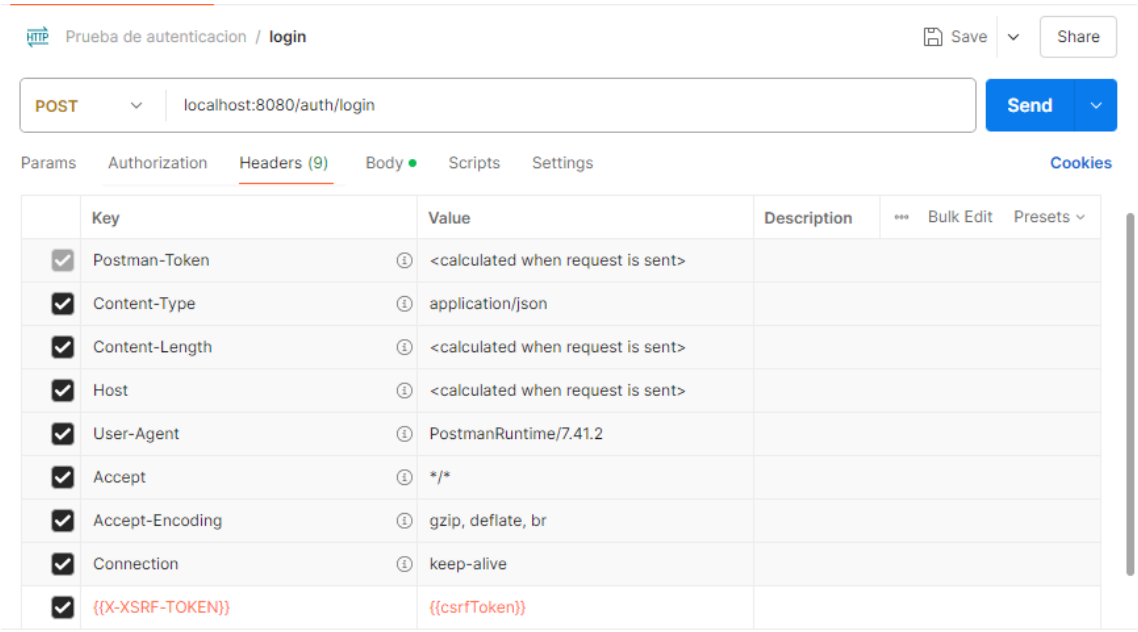

Figura 7.4: encabezado de prueba de API login

En las figuras anteriores se muestra el proceso para realizar una solicitud desde Postman y los datos necesarios que deben incluirse. Para automatizar este proceso de prueba, se define la variable csrfToken en la Figura 7.2, la cual contiene el valor recibido después de realizar la petición al método getXSRFToken. Este valor se asigna a la variable X-XSRF-TOKEN, requerida por el backend para la autenticación a través del método de login. La variable X-XSRF-TOKEN se añade en el encabezado de la solicitud del método Login.

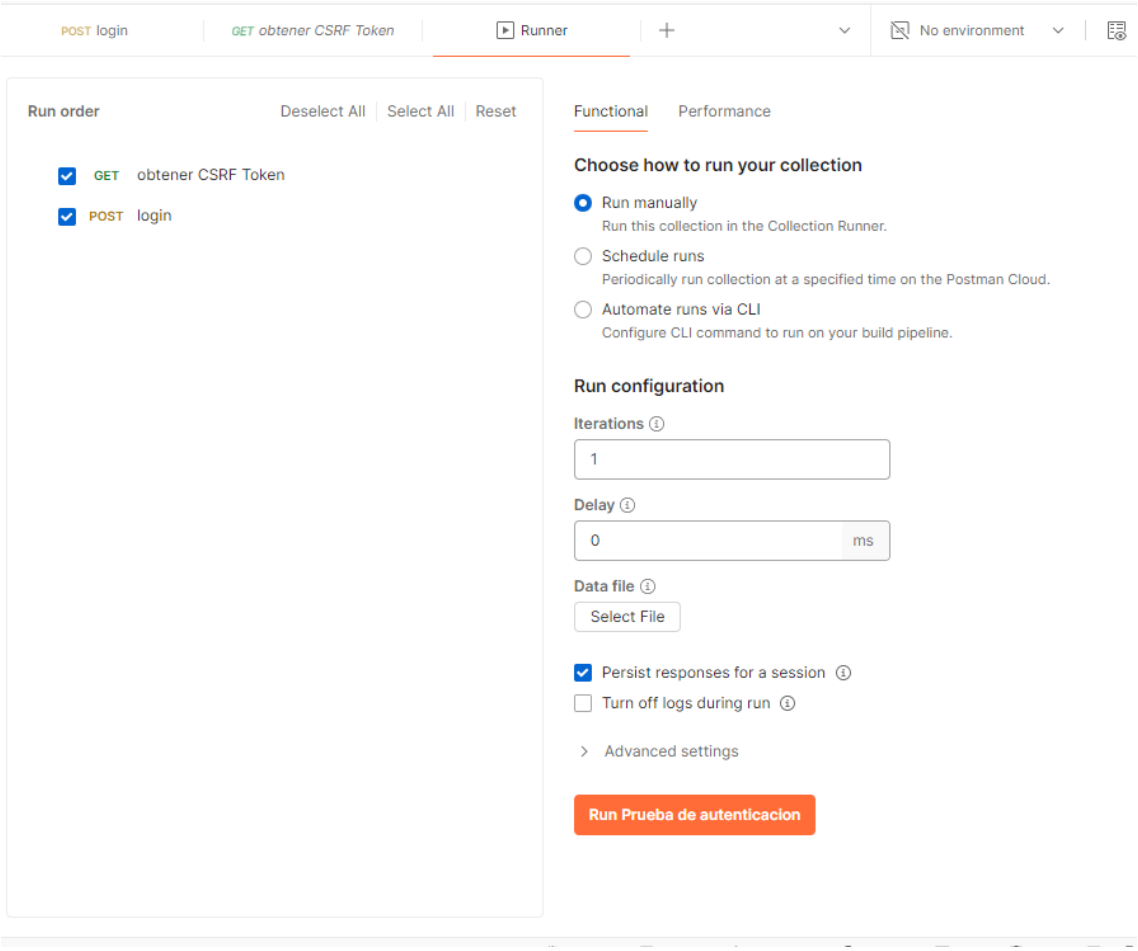

Figura 7.5: Runner Postman

Para ejecutar las pruebas se utiliza la función de runner, se incluye los dos métodos que se participan en la funcionalidad de la autenticación y se efectúa las pruebas de manera automática con las configuraciones hechas en el paso anterior, pulsando el botón de inicio y se obtiene los siguientes resultados.

Se puede observar que los métodos funcionan correctamente con las respuestas de 200 Ok del servidor, con el token de CSRF y la sesión generada para las posteriores solicitudes.

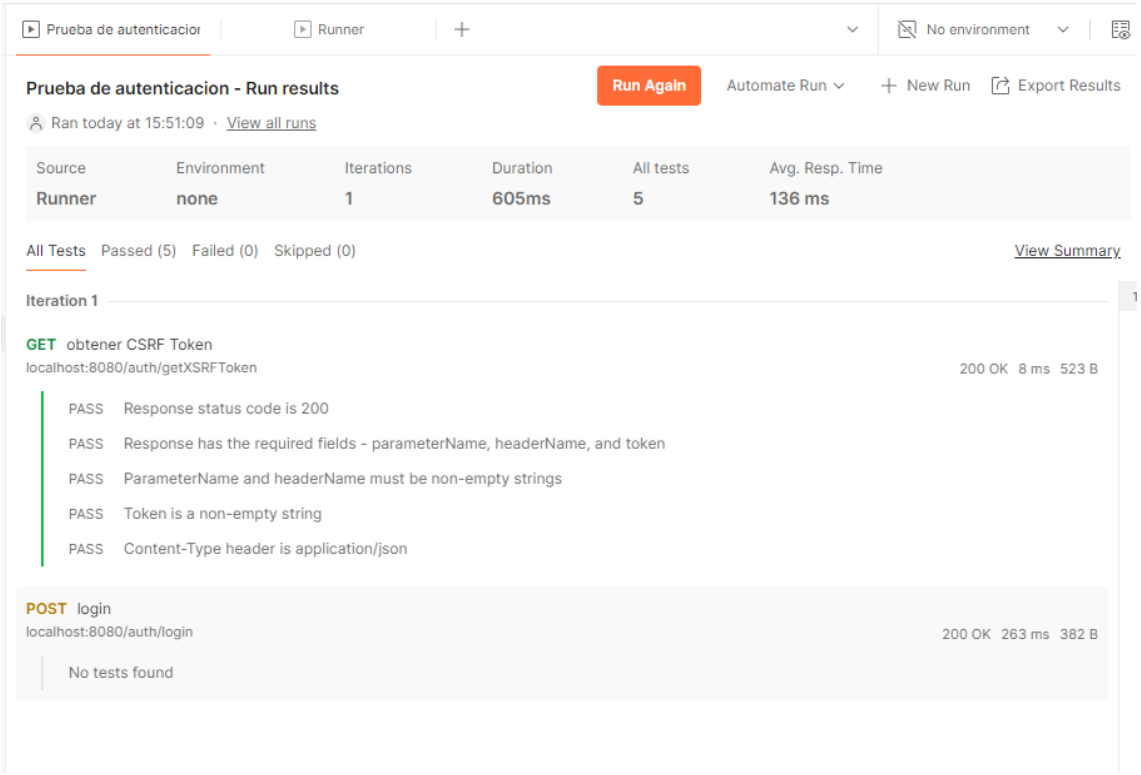

Figura 7.6: Resultado de pruebas de integración de autenticación

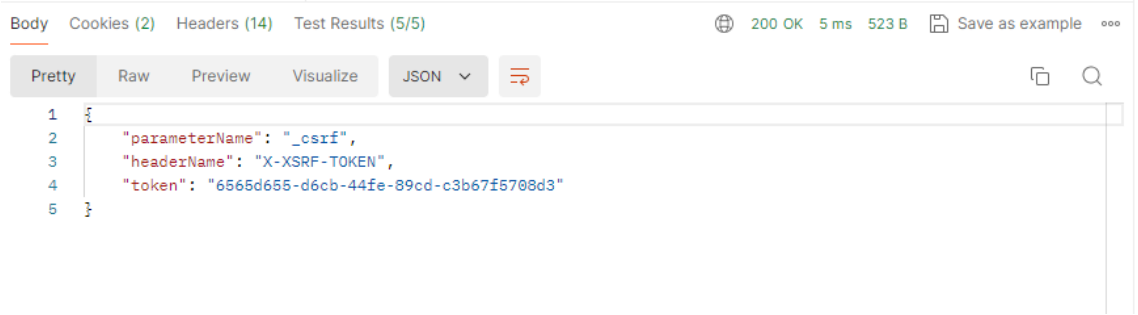

Figura 7.7: Respuesta de servidor de prueba de método getXSRFToken

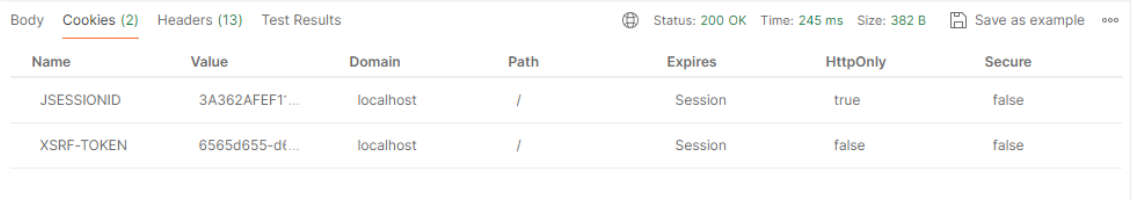

Figura 7.8: Respuesta de servidor de prueba método login

En resumen, se realizado los siguientes pasos para crear pruebas de integración usado Postman:

1. Crear una colección de pruebas, incluyendo todas las pruebas que participan en una funcionalidad.
2. Si existen dependencias de datos entre peticiones, se configuran los variables para colección y se crea los scripts para asignar los valores.
3. Utilizar el runner para ejecutar todas las pruebas automáticamente.
4. Se obtiene los resultados de ejecución de las pruebas con las respuestas del servidor.

7.2 Pruebas de aceptación

La prueba de aceptación verifica que el sistema cumple con los requisitos y las expectativas del usuario final. Estas pruebas se llevan a cabo con el propósito de validar las funcionalidades del sistema operen correctamente en un entorno real y el producto final es aceptado por parte de cliente. Para ello, se pone a marcha el sistema y se invita a varios usuarios para probar su funcionamiento. A través de las encuestas y retroalimentación, se identifican posibles problemas o defectos desde la perspectiva del usuario final.

Se ha obtenido las siguientes respuestas en las encuestas:

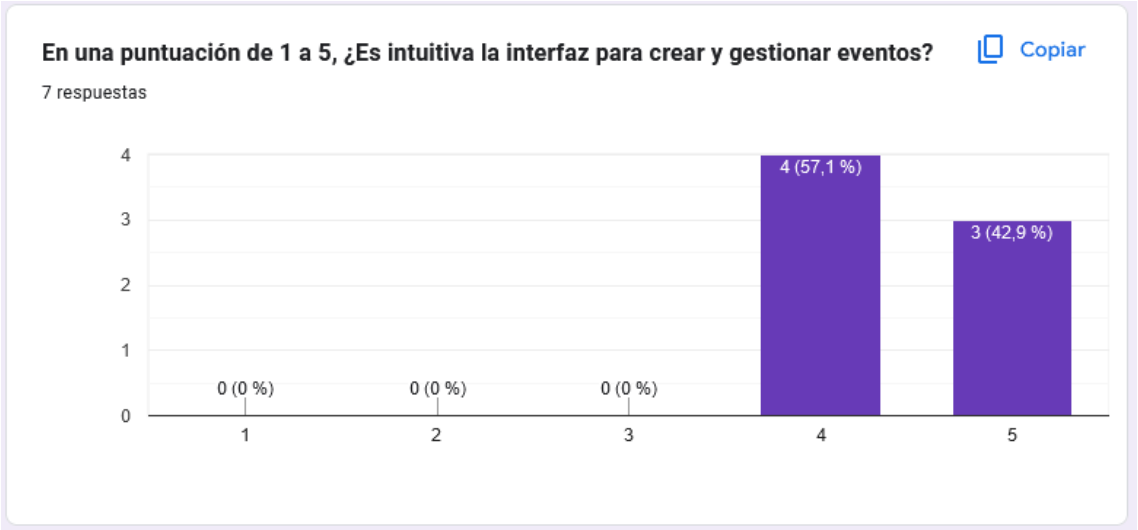

Figura 7.9: Pregunta 1 de encuesta

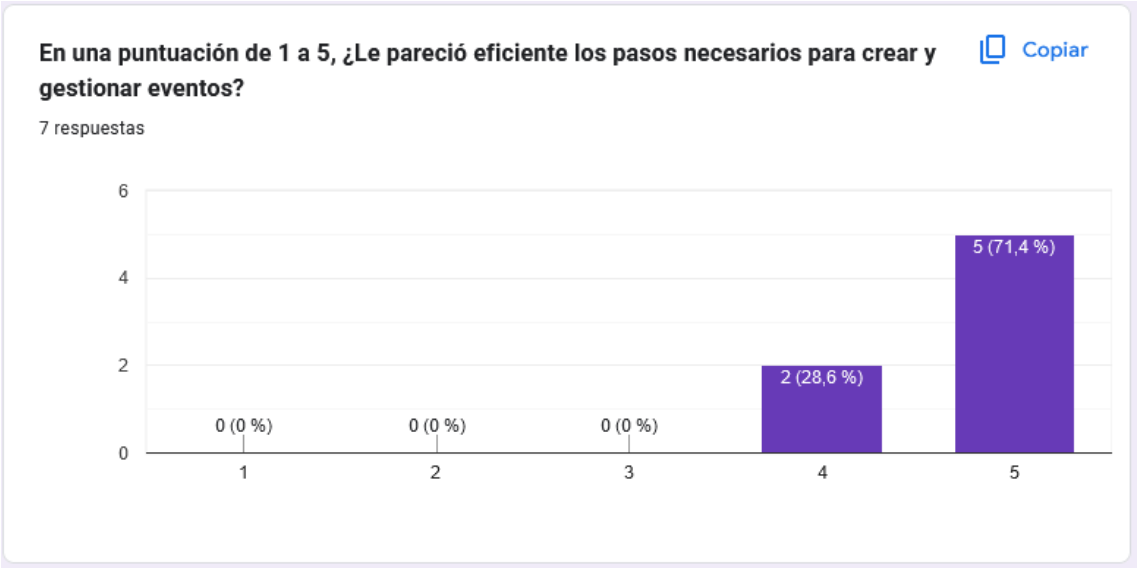

Figura 7.10: Pregunta 2 de encuesta

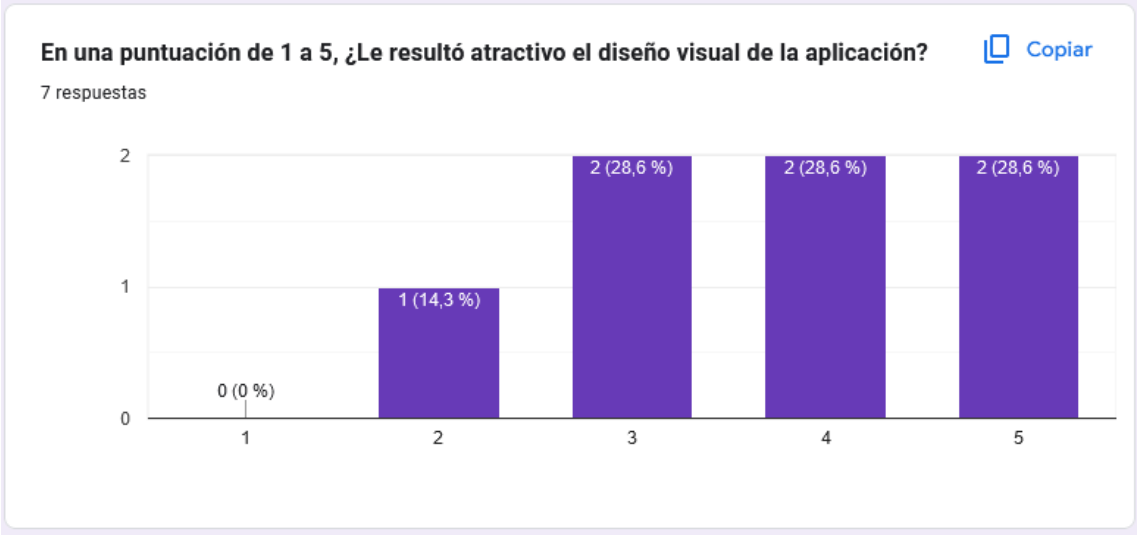

Figura 7.11: Pregunta 3 de encuesta

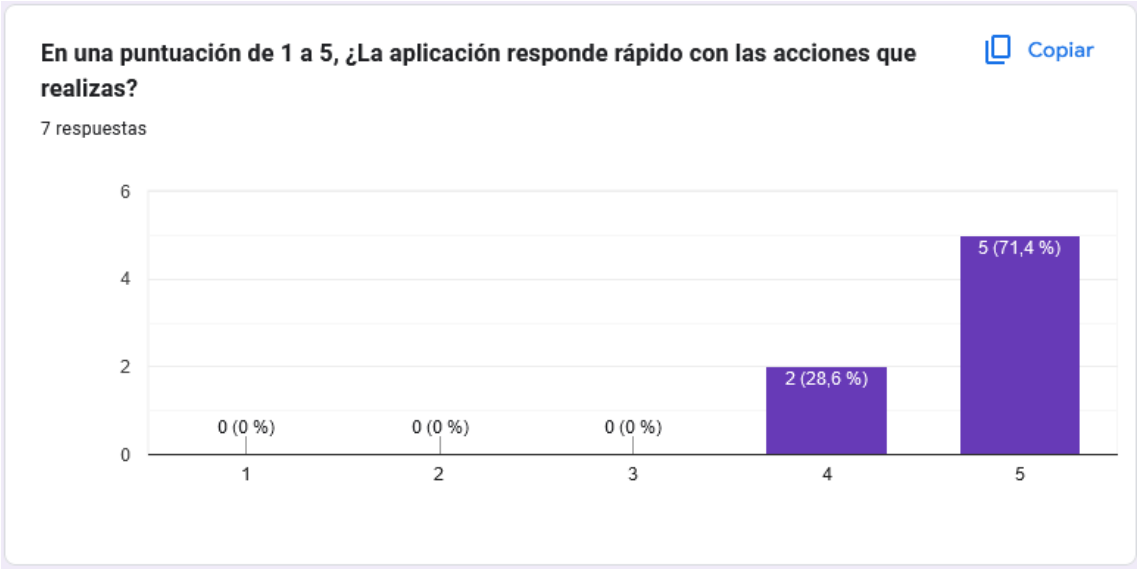

Figura 7.12: Pregunta 4 de encuesta

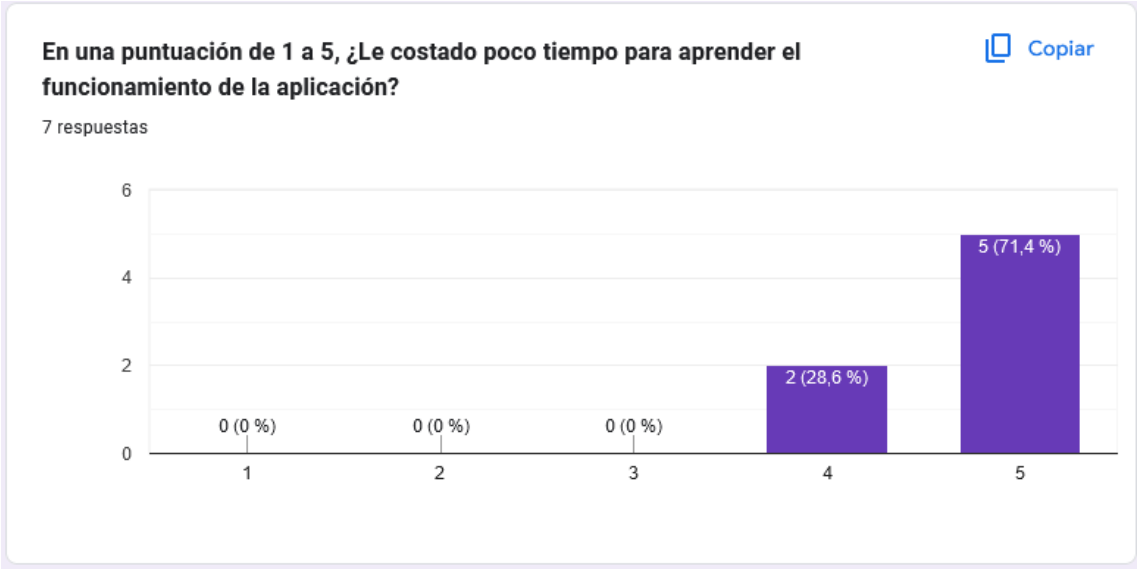

Figura 7.13: Pregunta 5 de encuesta

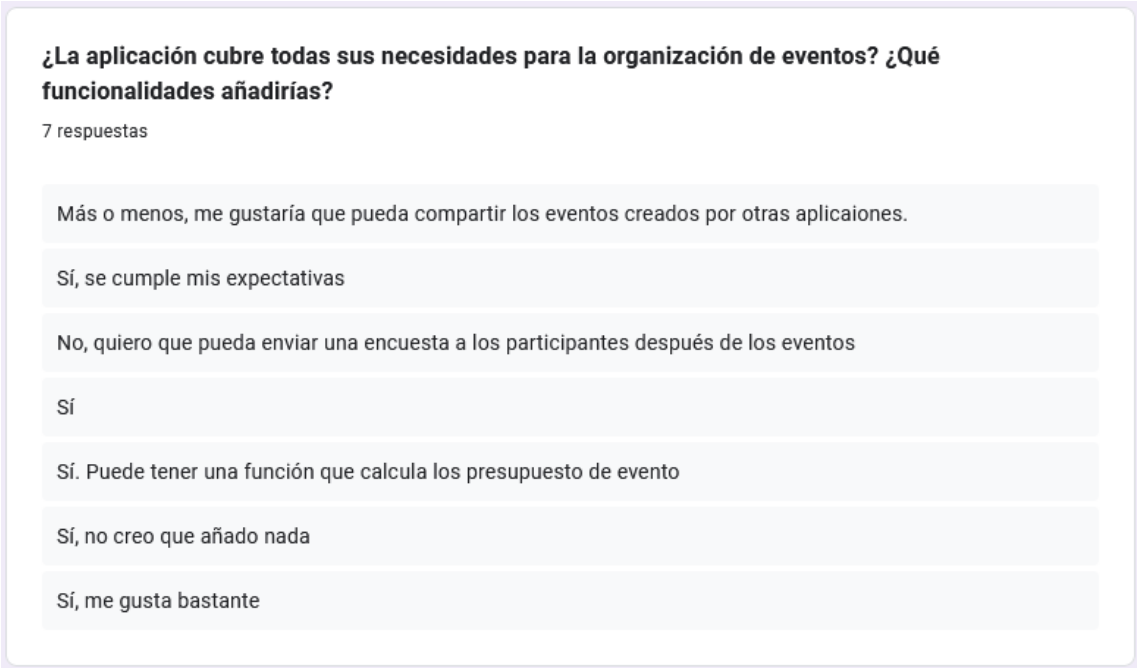

Figura 7.14: Pregunta 6 de encuesta

Según las opiniones de los usuarios, las interfaces para la organización y gestión de eventos son intuitivas y eficientes, destacando la facilidad de aprendizaje y rapidez en la respuesta de la aplicación. Sin embargo, algunos usuarios han expresado la insatisfacción con el diseño visual de la aplicación. Además, se sugieren incorporar funcionalidades adicionales, como la posibilidad de compartir informaciones de los eventos en las redes sociales, enviar encuestas a los participantes después de los eventos y una función para estimar el presupuesto necesario para organizar un evento.

CAPÍTULO 8

Conclusiones

En este capítulo se realiza una evaluación del logro de los objetivos planteados y se comenta sobre las relaciones del trabajo realizado con el estudio cursado en la escuela.

8.1 Objetivos cumplidos

En este proyecto, se ha llevado a cabo la implementación de una aplicación destinada a la creación y gestión de eventos, mejorando y agilizando los procesos necesarios para su organización. Se ha realizado la funcionalidad de chat para que permita la comunicación fluida y efectiva entre los usuarios, proporcionando un espacio para el intercambio de experiencias y facilitando el establecimiento de nuevas conexiones y relaciones profesionales.

Como objetivo personal, se ha hecho uso los conocimientos adquiridos en el estudio y se ha demostrado la capacidad de aprendizaje de las nuevas tecnologías. Esta experiencia no solo ha permitido alcanzar los objetivos propuestos, sino también ha contribuido a mis crecimientos profesionales, fortaleciendo las competencias como la análisis y resolución de problemas, la gestión de tiempo y capacidad para la toma de decisiones.

Aunque se han conseguido lograr los requisitos y la expectativa sobre este trabajo, existen algunos aspectos, como la parte de diseño visual, que aún requieren una mejora y perfeccionamiento. Estos flecos pendientes se resolverán en futuros desarrollos junto con los nuevo requisitos y necesidades que podrían tener los usuarios sobre este proyecto.

8.2 Relación con el estudio cursado

En el desarrollo de este proyecto, se ha hecho un uso extensivo de técnicas y metodologías que fueron adquiridas durante los cursos de estudio. La aplicación de estos conocimientos ha sido fundamental para la implementación de la solución propuesta.

Algunos de las asignaturas más relevantes pueden ser:

- **Análisis y especificación de requisitos**, ha proporcionado las herramientas necesarias para identificar y documentar de manera precisa las necesidades del usuario, asegurando que el desarrollo del software esté alineado con los objetivos del proyecto.

- **Proyecto de ingeniería de software,** ha enseñado las metodologías para llevar a cabo el desarrollo de una aplicación, mejorando las habilidades de colaboración en equipo y de la comunicación efectiva con los clientes.

- **Mantenimiento de software,** ha sido clave para comprender las necesidades relacionadas con los procesos de mantenimientos y ha impartido conocimientos sobre las herramientas para facilitar este proceso.

- **Calidad de software,** ha enseñado los atributos y las métricas esenciales para la evaluación de la calidad del software, permitiendo que la calidad sea medible.

- **Diseño de software,** se ha introducido a los patrones de diseño, que es crucial para la construcción de una aplicación robusta y escalable.

- **Base de datos,** ha proporcionado los conocimientos profundos sobre la estructura y el uso de las tecnologías de base de datos, facilitando el manejo y optimización de los datos en el desarrollo de la aplicación.

Para concluir, este TFG ha sido un desafío que ha permitido aplicar y consolidar los conocimientos adquiridos a lo largo de la carrera, y también desarrollar las habilidades críticas para la resolución de los problemas complejos en el desarrollo de software. Esta experiencia ha sido fundamental para mi crecimiento profesional, preparándome para enfrentar con confianza los retos profesionales futuros.

Bibliografía

[1] *Julia Martins*, *¿Qué es la metodología Kanban y cómo funciona?*,

https://asana.com/es/resources/what-is-kanban

[2] *Atlassian*, *«Trello»*, https://trello.com/es

[3] *eventbrite*, *«eventbrite»*, https://www.eventbrite.es/

[4] *cvent*, *«cvent»*, https://www.cvent.com/

[5] *Whova*, *«Whova»*, https://whova.com/

[6] *deit & IsaacAlvrt*, *MVC*, https://developer.mozilla.org/es/docs/Glossary/MVC

[7] *Spring, «Spring Boot»,* https://spring.io/projects/spring-boot

[8] *Spring, The IoC container,* https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html

[9] *Manu Pijierro, Principio SOLID: Principio de Responsabilidad única*, https://mpijierro.medium.com/principios-solid-principio-de-responsabilidad-%C3%BAnica-13eb4d5537c1

[10] *Refactoring Guru, Patrón Singleton*, https://refactoring.guru/es/design-patterns/singleton

[11] *Refactoring Guru, Patrón Builder*, https://refactoring.guru/es/design-patterns/builder

[12] *Google, «Flutter»*, https://flutter.dev/

[13] *Google, «Dart»*, https://dart.dev/

[14] *Apache software foundation*, *«Maven»*, https://maven.apache.org/

[15] *Apache software foundation*, *«Tomcat»*, https://tomcat.apache.org/

[16] *Oracle Corporation, «Java»*, https://www.java.com/es/

[17] *PostgreSQL, «PostgreSQL»,* https://www.postgresql.org/

[18] *Microsoft, «Visual Studio Code»,* https://code.visualstudio.com/

[19] *Github Inc, «Github»,* https://github.com/

[20] *Google, «Android Studio»,* https://developer.android.com/?hl=es-419

[21] *Postman*, *«Postman»,* https://www.postman.com/

[22] *Draw io, «Draw io»,* https://www.drawio.com/

[23] *Google, «Google Forms», https://www.google.es/intl/es/forms/about/*

[24] *Oliver Gierke, Christoph Strobl, Mark Paluch, Sander Krabbenborg, Jesse Wouters, Greg Turnquist & Jens Schauder, «Interface JpaRepository»,* https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html

[25] *Spring, «Spring Security»,* https://spring.io/projects/spring-security

[26] *Apache software foundation*, *«Apache Http Server»*, https://httpd.apache.org/docs/2.4/es/

[27] *Thymeleaf*, *«Thymeleaf»,* https://www.thymeleaf.org/

[28] *Oracle Corporation*, *«JSP»,* https://www.oracle.com/java/technologies/?er=221886

APÉNDICE A

Objetivos de Desarrollo Sostenible

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

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

Tabla A.1: Objetivos de Desarrollo Sostenible

Reflexión sobre la relación del TFG los ODS más relacionados:

- **ODS 3. Salud y bienestar.** La aplicación puede contribuir a este ODS si se utiliza para organizar y promocionar los eventos relacionados con la salud y el bienestar, como talleres de bienestar, seminarios sobre salud mental o jornadas deportivas. Además, las funcionalidades de chat y el intercambio de experiencias pueden fortalecer las comunidades de apoyo entre usuarios, promoviendo un estilo de vida saludable.

- **ODS 4. Educación de calidad.** La aplicación puede dar apoyo a este ODS al facilitar la creación y la organización de eventos educativos, tales como seminarios o cursos en línea. Al facilitar el acceso al conocimiento y ofrecer oportunidades de aprendizaje continuo, la aplicación impulsa el desarrollo personal y profesional de los usuarios, promoviendo una educación de calidad para todos.

- **ODS 8. Trabajo decente y crecimiento económico.** La aplicación puede tener un impacto positivo en el crecimiento económico y la promoción de trabajo decente, al proporcionar una plataforma que facilita el acceso a los eventos, se crean oportunidades para que las personas encuentren trabajo, mejoren sus habilidades en estos eventos y establezcan conexiones profesionales.

- **ODS 10. Reducción de las desigualdades.** La aplicación ayuda a reducir las desigualdades al facilitar el acceso equitativo a los eventos, sin importar los niveles socioeconómico, género u otros factores.

- **ODS 17. Alianzas para lograr los objetivos.** La aplicación puede dar soporte a este ODS, puesto que se facilita la colaboración y creación de alianzas a través de la organización de eventos. Se proporciona un espacio donde puedan reunirse, compartir conocimientos y colaborar en proyectos de desarrollo sostenible.