

# Documentación Programa de Gestión de Eventos

Gestión de Eventos (Mateo Picatoste y Pedro Monzón).

Trabajo Acceso a Datos - Segunda Evaluación

# 1 - Inicio del proyecto

## Objetivo del Sistema:

La aplicación tiene como objetivo principal proporcionar una plataforma interactiva y fácil de usar que permita a los usuarios crear, gestionar y registrarse en eventos. Busca simplificar el proceso de organización de eventos y facilitar la participación de los usuarios en actividades de su interés, promoviendo la interacción comunitaria y el intercambio de experiencias.

## Audiencia:

Este proyecto está dirigido a desarrolladores, diseñadores y cualquier persona interesada en comprender y contribuir al desarrollo del Sistema de Gestión de Eventos. También es relevante para los organizadores de eventos y usuarios que utilizarán la plataforma para crear y participar en eventos.

## Descripción General:

El Sistema de Gestión de Eventos permite a los usuarios crear eventos, gestionar inscripciones y controlar la participación. Los eventos pueden tener múltiples participantes, y cada usuario puede inscribirse en varios eventos.

## Principales Características:

- Creación de Eventos: Los organizadores pueden crear eventos, proporcionando detalles como fecha, ubicación, descripción, etc.
- Inscripciones: Los usuarios pueden inscribirse en eventos de su interés, gestionando su participación.
- Participantes: Cada evento puede tener varios participantes, y cada usuario puede participar en múltiples eventos.

## Modelo de Datos:

El sistema utiliza relaciones ManyToMany para vincular eventos con participantes y OneToMany para las inscripciones de usuarios en eventos.

## Lenguaje de Programación:

El proyecto está implementado en java, asegurando una base sólida y versátil para el desarrollo.

## 2 - Análisis y Planificación de Requerimientos

Identificación de Requisitos Funcionales y No Funcionales.

### 2.1 Requisitos Funcionales.

#### 2.1.1 Registro y Autenticación de Usuarios:

La aplicación permitirá a los usuarios registrarse proporcionando datos personales básicos y autenticarse para acceder a sus perfiles y funcionalidades específicas.

#### 2.1.2 Creación y Gestión de Eventos:

Los usuarios podrán crear eventos, especificando detalles como el título, descripción, fecha, hora y ubicación. Además, podrán editar o eliminar eventos que hayan creado.

#### 2.1.3 Inscripción en Eventos:

Los usuarios podrán inscribirse en eventos creados por otros usuarios, y ver una lista de los eventos en los que están inscritos.

#### 2.1.4 Visualización de Participantes:

Los organizadores de eventos podrán ver una lista de los usuarios inscritos en sus eventos.

#### 2.1.5 Perfil de Usuario:

Los usuarios tendrán acceso a una página de perfil donde pueden ver y editar su información personal, visualizar los eventos que han creado y a los que se han inscrito.

## 2.2 Requisitos No funcionales

### 2.2.1 Usabilidad:

La aplicación será intuitiva y fácil de navegar, con un diseño responsive que se adapte a diferentes dispositivos.

### 2.2.2 Rendimiento:

La aplicación deberá cargar y responder rápidamente a las interacciones del usuario, incluso durante picos de alta demanda.

### 2.2.3 Seguridad:

Los datos de los usuarios estarán protegidos mediante el uso de prácticas de seguridad actuales, incluyendo la encriptación de contraseñas y la protección contra vulnerabilidades comunes de la web.

### 2.2.4 Escalabilidad:

La arquitectura de la aplicación permitirá su escalabilidad para soportar un crecimiento en el número de usuarios y eventos.

## 2.3 Restricciones y Suposiciones

Se asume que los usuarios tienen acceso a internet y dispositivos compatibles para utilizar la aplicación.

La aplicación estará limitada por las capacidades del servidor y la base de datos MySQL en términos de manejo de carga y almacenamiento de datos.

El tiempo de desarrollo está condicionado por la disponibilidad de recursos y el alcance definido para la primera versión de la aplicación.

## 3 - Diseño de la Arquitectura

El diseño arquitectónico de nuestra aplicación de gestión de eventos está pensado para ser robusto, escalable y fácil de mantener. Utilizamos el patrón de diseño Modelo-Vista-Controlador (MVC) para separar las responsabilidades dentro de la aplicación, facilitando así la gestión del código y su evolución a lo largo del tiempo.

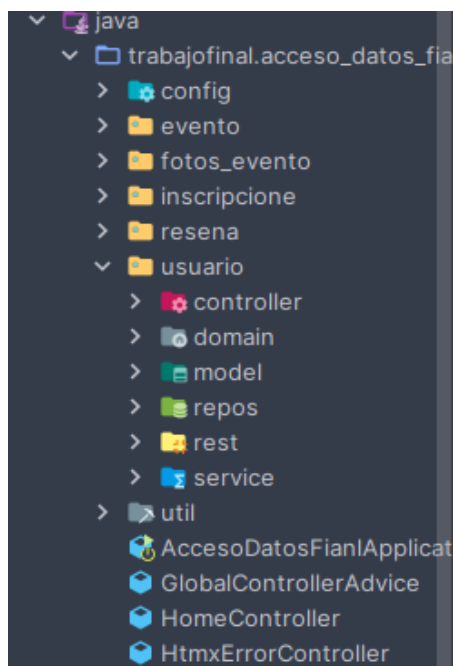
### 3.1 Arquitectura Modelo-Vista-Controlador (MVC)

La aplicación se estructura siguiendo el patrón MVC, lo que nos permite desacoplar la lógica de negocio de la interfaz de usuario y la interacción con la base de datos:

**Modelo:** Define la estructura de datos y la lógica de negocio. En nuestro caso, el modelo incluye entidades como Evento, Usuario, Inscripcion, y Resena, cada una representada en su propio subpaquete bajo domain y model.

**Vista:** Se encarga de la presentación de la información y la interacción con el usuario. Las vistas están implementadas utilizando plantillas HTML en el directorio resources/templates, organizadas por funcionalidad (evento, usuario, inscripción, etc.).

**Controlador:** Actúa como intermediario entre la vista y el modelo, gestionando el flujo de información entre ellos y actualizando la vista en respuesta a cambios en el modelo. Los controladores se encuentran en el subpaquete controller de cada entidad.



## 3.2 Estructura del Proyecto

La aplicación está organizada en múltiples paquetes, cada uno centrado en una entidad específica del dominio o una funcionalidad de la aplicación:

config: Contiene configuraciones globales de la aplicación, como la configuración de seguridad.

evento, fotos\_evento, inscripcion, resena, usuario: Cada uno de estos paquetes incluye subpaquetes para controller, domain, model, repos, rest, y service, siguiendo el patrón MVC y la separación de responsabilidades.

util: Proporciona utilidades y clases de ayuda comunes a toda la aplicación.

## 3.3 Tecnologías Utilizadas

Spring Boot: Marco de trabajo principal para el desarrollo de la aplicación, facilitando la configuración y el despliegue.

Thymeleaf: Motor de plantillas para la generación de vistas HTML dinámicas.

Spring Data JPA con Hibernate: Para la capa de persistencia, facilitando la interacción con la base de datos MySQL mediante ORM.

## 3.4 Escalabilidad y Mantenibilidad

Hemos diseñado la aplicación con la escalabilidad y la mantenibilidad en mente. La clara separación de responsabilidades, el uso de principios de diseño como SOLID y patrones de diseño establecidos, junto con la elección de tecnologías ampliamente adoptadas y soportadas, aseguran que la aplicación pueda crecer y adaptarse a nuevas necesidades con el tiempo.

## 3.5 Conclusión

Nuestra arquitectura está diseñada para soportar la complejidad inherente a la gestión de eventos, proporcionando una base sólida sobre la cual podemos construir y expandir. Con una estructura clara, un código bien organizado y la adopción de mejores prácticas y tecnologías líderes en la industria, nuestra aplicación está preparada para satisfacer las necesidades actuales y futuras de los usuarios.



## 4- Modelo de Datos Eficiente

El modelo de datos es la piedra angular de nuestro Sistema de Gestión de Eventos, diseñado para optimizar la eficiencia y mantener la integridad de los datos. Hemos desarrollado una estructura que refleja con precisión las complejas relaciones entre los usuarios, los eventos y sus interacciones.

### 4.1 Entidades Principales

Nuestro modelo consta de varias entidades clave, cada una con atributos específicos que capturan la esencia de los componentes del sistema:

#### 4.1.1 Eventos

EventoID (PK): Identificador único para cada evento.

OrganizadorID (FK): Referencia al usuario que organiza el evento.

Nombre: Nombre descriptivo del evento.

Descripción: Detalles sobre lo que involucra el evento.

FechaInicio/FechaFin: Duración del evento.

Ubicación: Localización física o virtual del evento.

EsExterior/EsGratis: Atributos booleanos que indican si el evento es al aire libre o gratuito.

Logo/Banner: Imágenes representativas del evento.

### 4.1.2 Usuarios

UsuarioID (PK): Identificador único de cada usuario.

Nombre: El nombre completo del usuario.

CorreoElectronico: Dirección de correo electrónico para contacto y acceso.

Contrasena: Contraseña segura y encriptada para protección de la cuenta.

FechaRegistro: Fecha en la que el usuario se unió a la plataforma.

FotoPerfil/Descripcion: Datos adicionales para personalizar el perfil del usuario.

## 4.2 Relaciones entre Entidades

Las relaciones definen cómo interactúan las entidades entre sí dentro de nuestra base de datos:

### 4.2.1 Inscripciones (OneToMany)

Una asociación entre usuarios y eventos donde un usuario puede inscribirse en varios eventos, y un evento puede tener múltiples inscripciones.

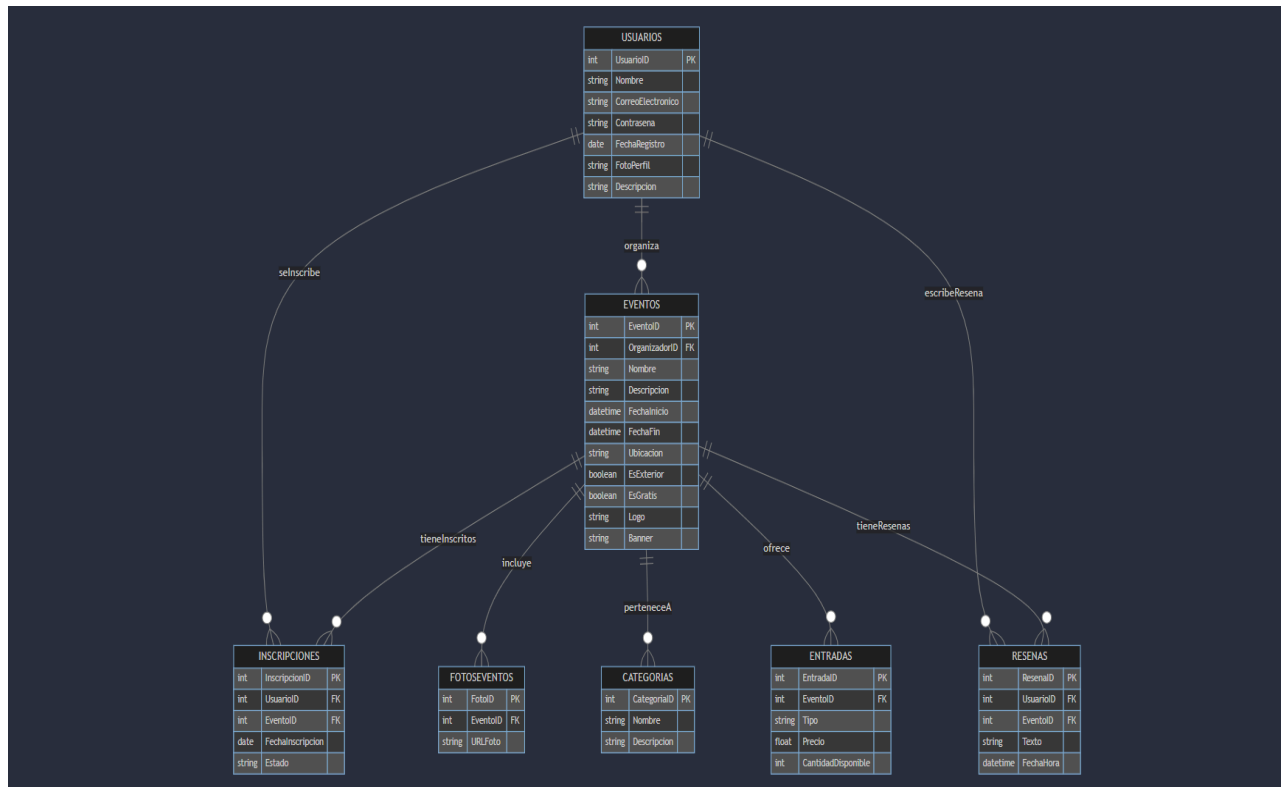
### 4.2.2 Reseñas (OneToMany)

Usuarios pueden escribir reseñas sobre eventos a los que asisten, proporcionando feedback y comentarios valiosos.

### 4.2.3 Categorías y Entradas

Los eventos se clasifican en categorías para facilitar la búsqueda y organización.

Las entradas se gestionan por evento, definiendo el tipo, precio y cantidad disponible.



## 4.3 Consideraciones de Eficiencia

Para garantizar un rendimiento óptimo, hemos implementado varias estrategias de optimización de la base de datos:

**Índices:** Mejoramos la velocidad de las consultas con índices en claves primarias y extranjeras.

**Normalización:** La estructura está normalizada para reducir la redundancia y asegurar la integridad de los datos.

Optimización de Consultas: Diseñamos consultas eficientes para acceder a la información relevante de manera rápida y confiable.

## 4.4 Mejoras Futuras

Estamos abiertos a la evolución del modelo de datos para incorporar nuevas características, como etiquetado avanzado de eventos, sistemas de recomendación basados en preferencias de usuarios, y cualquier otra mejora que enriquezca la experiencia del usuario y la gestión de eventos.

Nuestro modelo de datos establece una base firme para un sistema de gestión de eventos robusto y eficiente, asegurando una experiencia de usuario suave y una administración efectiva de la información de eventos.

## 4.5 Guía de Consultas

Esta guía ofrece ejemplos de consultas SELECT que pueden realizarse en la base de datos del Sistema de Gestión de Eventos. Estas consultas permiten extraer diferentes conjuntos de datos para la presentación, análisis o procesamiento adicional.

### 4.5.1 Consulta Eventos

Para obtener una lista de todos los eventos con su información detallada:

```
sql Copy code  
  
SELECT * FROM Eventos;
```

Para obtener los eventos organizados por un usuario específico:

```
sql Copy code  
  
SELECT * FROM Eventos WHERE OrganizadorID = 1; -- Reemplazar con el ID del organizador
```

### 4.5.2 Consulta Usuarios

Para listar todos los usuarios registrados en la plataforma:

```
sql Copy code  
  
SELECT UsuarioID, Nombre, CorreoElectronico, FechaRegistro FROM Usuarios;
```

Para encontrar un usuario por su correo electrónico:

```
sql Copy code  
  
SELECT * FROM Usuarios WHERE CorreoElectronico = 'usuario@example.com';
```

### 4.5.3 Consulta de Inscripciones

Para visualizar todas las inscripciones de un evento específico:

```
sql Copy code  
  
SELECT * FROM Inscripciones WHERE EventoID = 1; -- Reemplazar con el ID del evento es
```

Para consultar los eventos a los que un usuario está inscrito:

```
sql Copy code  
  
SELECT Eventos.* FROM Eventos  
INNER JOIN Inscripciones ON Eventos.EventoID = Inscripciones.EventoID  
WHERE Inscripciones.UsuarioID = 1; -- Reemplazar con el ID del usuario
```

#### 4.5.4 Consultas de Reseñas

Para obtener todas las reseñas escritas por un usuario:

```
sql Copy code  
  
SELECT * FROM Resenas WHERE UsuarioID = 1; -- Reemplazar con el ID del usuario
```

Para obtener las reseñas de un evento específico:

```
sql Copy code  
  
SELECT * FROM Resenas WHERE EventoID = 1; -- Reemplazar con el ID del evento
```

#### 4.5.5 Consultas Avanzadas

Para listar eventos con la cantidad de inscripciones:

```
sql Copy code  
  
SELECT Eventos.Nombre, Eventos.Descripcion, COUNT(Inscripciones.EventoID) as NumeroIr  
FROM Eventos  
LEFT JOIN Inscripciones ON Eventos.EventoID = Inscripciones.EventoID  
GROUP BY Eventos.EventoID;
```

Para listar usuarios y la cantidad de eventos a los que se han inscrito:

```
sql Copy code  
  
SELECT Usuarios.Nombre, Usuarios.CorreoElectronico, COUNT(Inscripciones.UsuarioID) as  
FROM Usuarios  
LEFT JOIN Inscripciones ON Usuarios.UsuarioID = Inscripciones.UsuarioID  
GROUP BY Usuarios.UsuarioID;
```

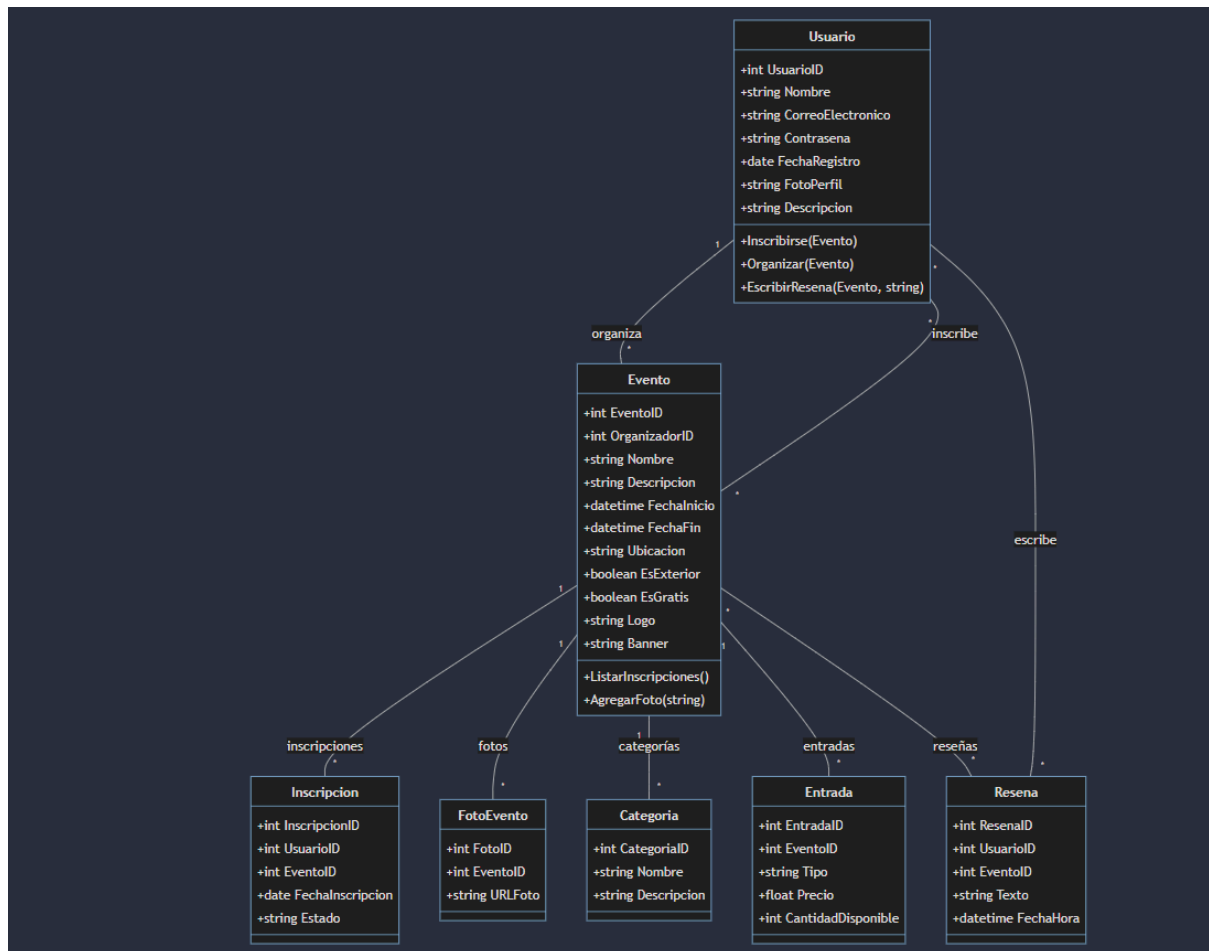
## 4.6 Consideraciones

Asegúrate de tener los permisos adecuados para ejecutar estas consultas. Reemplaza los valores de ejemplo (como ID de usuario o ID de evento) con los valores reales de tu base de datos.

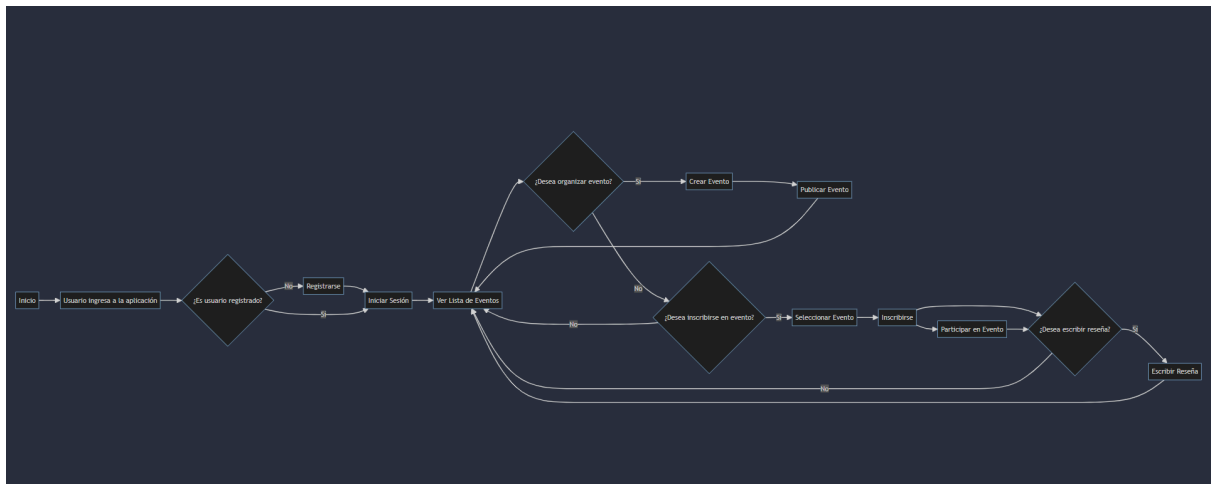
Para consultas más complejas, considera la posibilidad de crear vistas o procedimientos almacenados que encapsulan la lógica de la consulta.

## 4.7 Diagramas

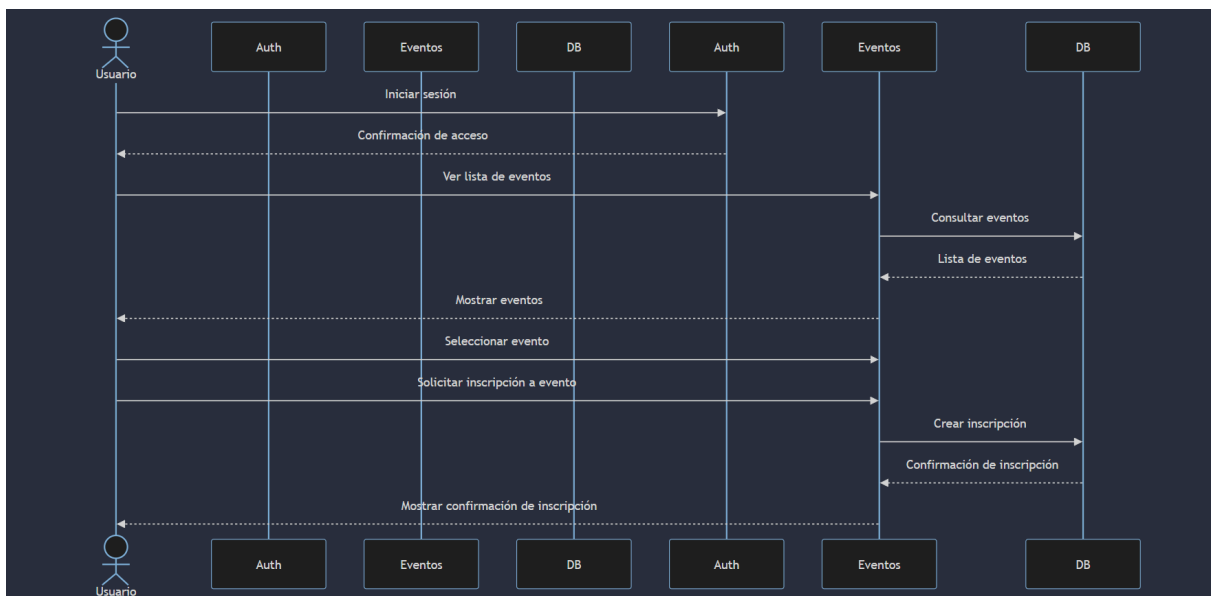
### 4.7.1 Diagrama UML



## 4.7.2 Diagrama de Flujo

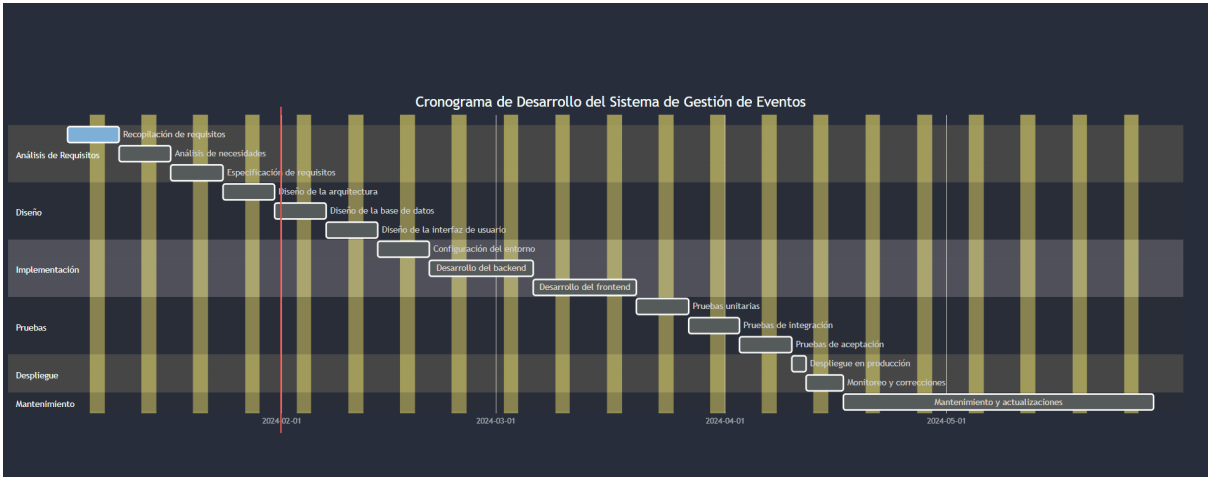


## 4.7.3 Diagrama Secuencial





4.7.4 Diagrama de Gantt



## 5 - Diseño UI/UX

El diseño de la interfaz de usuario (UI) y la experiencia del usuario (UX) son aspectos fundamentales para el éxito y la aceptación de nuestro Sistema de Gestión de Eventos. Nos hemos centrado en crear una interfaz intuitiva y atractiva que facilite la navegación y mejore la experiencia del usuario.

<https://www.figma.com/file/8vOIkRtfcraK9RFG21WQjK/Untitled?type=design&node-id=0%3A1&mode=design&t=AgcATrz1qQgSrNZu-1>

### 5.1 Principios de Diseño

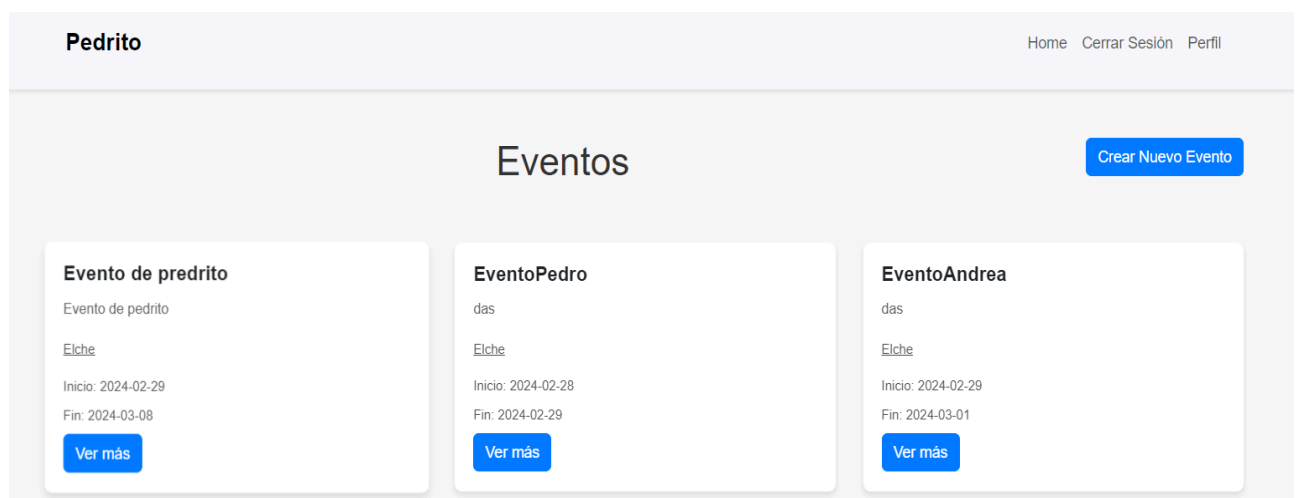
Hemos seguido los siguientes principios de diseño para garantizar una UI/UX efectiva:

**Simplicidad:** La interfaz se mantiene simple y libre de desorden, facilitando la comprensión y el uso intuitivo.

**Consistencia:** Se ha aplicado un diseño coherente en toda la aplicación para proporcionar una experiencia uniforme.

**Retroalimentación Visual:** Proporcionamos retroalimentación visual clara para las acciones del usuario, como confirmaciones de envío y cambios de estado.

**Flujo de Usuario Lógico:** El diseño sigue un flujo de usuario lógico, guiando a los usuarios de manera natural a través de las funciones clave.



## 5.2 Diseño Responsivo

La aplicación se ha diseñado para ser accesible en diferentes dispositivos y tamaños de pantalla, garantizando una experiencia coherente y funcionalidad completa tanto en computadoras de escritorio como en dispositivos móviles.

## 5.3 Elementos de Interfaz Clave

### 5.3.1 Creación de Eventos

- Un flujo de trabajo claro y paso a paso para la creación de eventos.
- Formularios intuitivos con campos relevantes y validación en tiempo real.

### Add Evento

Back to list

Nombre\*

Descripcion\*

Fecha Inicio\*

Fecha Fin\*

Ubicacion\*

☐ Es Exterior\*

☐ Es Gratis\*

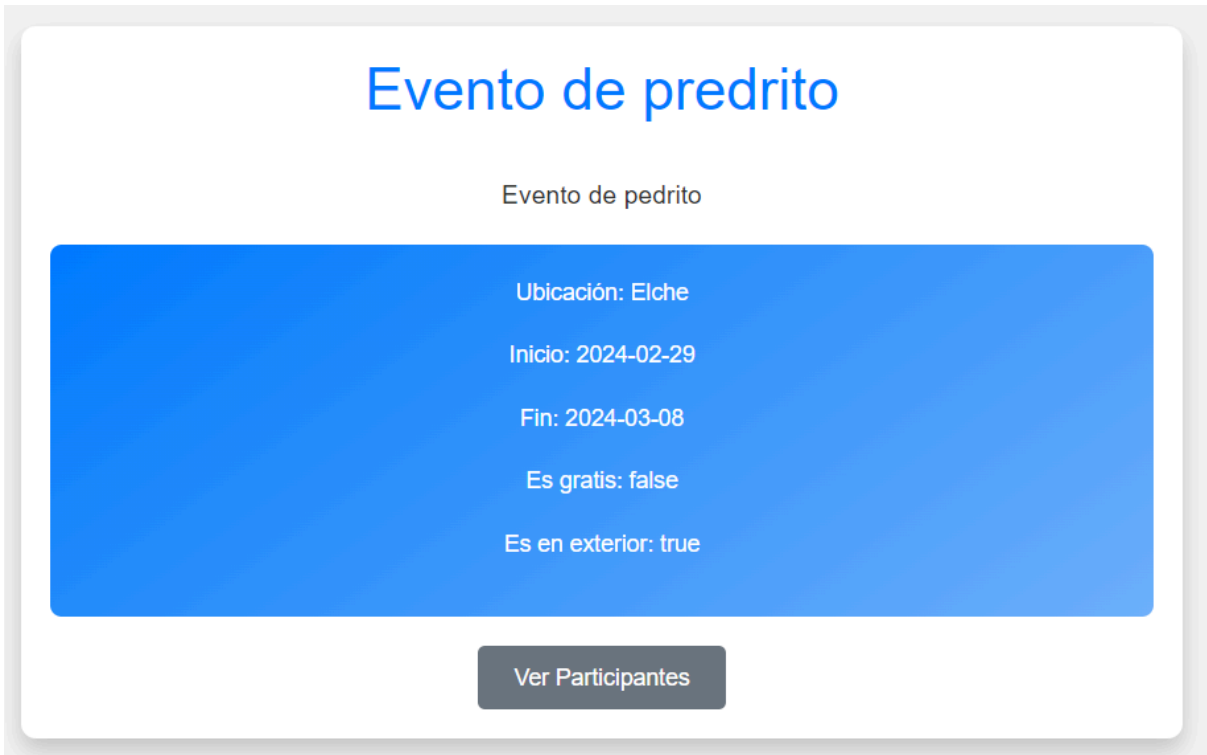
Logo

Banner

Add Evento

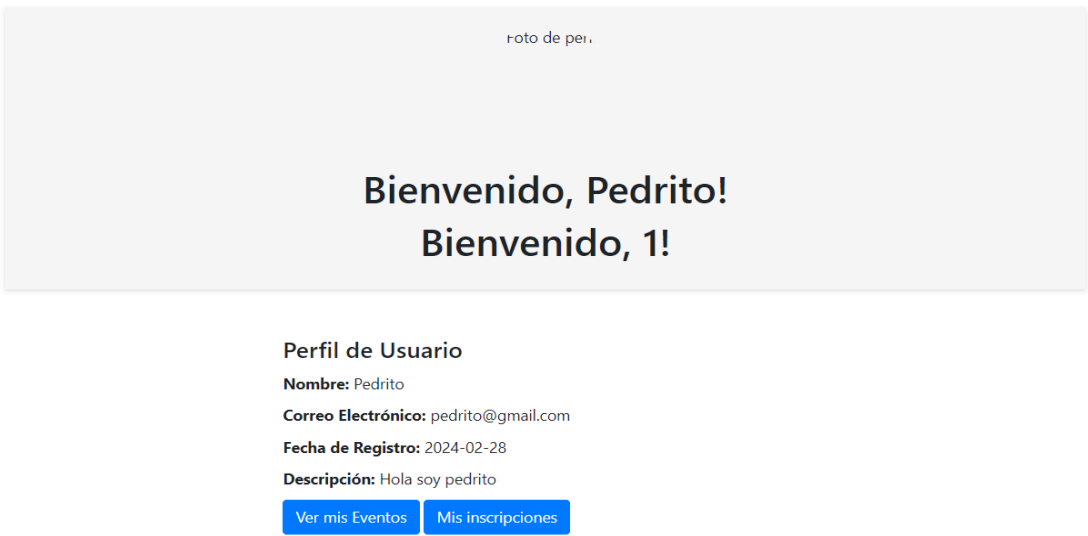
### 5.3.2 Lista los Datos del Evento

Presentación clara y eficiente de los datos del evento.



### 5.3.3 Perfil del Usuario

- Pantalla de perfil intuitiva con información personal y eventos relacionados.
- Configuración de cuenta y preferencias de notificación.



### 5.3.4 Login y Registro

La aplicación cuenta con una pantalla de login y registro, moderna e intuitiva. Sencilla y fácil de entender.

### Login

Correo Electrónico

Contraseña

Iniciar Sesión

[Registrarse](#)

### Registro

Nombre\*

Correo Electronico\*

Contrasena\*

Fecha de Registro

Foto de Perfil

Ninguno archivo selec.

Descripcion

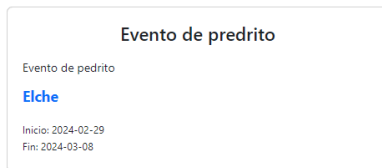
Registrarse

### 5.3.5 Listar tus eventos e inscripciones activas.

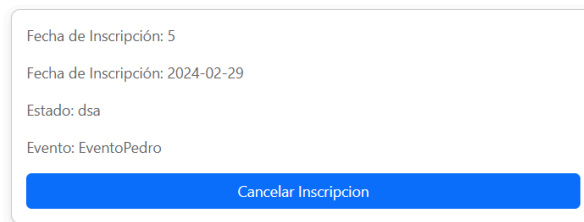
La aplicación cuenta con una parte dentro del perfil para ver tus eventos activos. Con un botón muy intuitivo y práctico.

Cuenta también con otro botón con el que puedes ver las inscripciones activas que tienes.

#### Mis Eventos



#### Mis Inscripciones



## 5.4 Estilo y Paleta de Colores

Hemos seleccionado una paleta de colores amigable y coherente que refleje la naturaleza dinámica y positiva de los eventos. Los elementos de la interfaz, como botones y llamadas a la acción, se destacan visualmente para una fácil identificación.

### 5.4.1 Colores

Azul: #007bff

Azul más oscuro: #0056b3

Color de los titulos: #333

## 5.5 Pruebas de Usabilidad

Se llevarán a cabo pruebas de usabilidad periódicas con usuarios reales para evaluar la eficacia del diseño y recopilar comentarios para posibles mejoras.

## 5.6 Accesibilidad

Hemos incorporado características que mejoran la accesibilidad, como descripciones altas en imágenes, texto legible y navegación intuitiva, para asegurar que la aplicación sea utilizada de manera efectiva por usuarios con diversas necesidades.

## 5.7 Iteración Continua

El diseño de la interfaz será un proceso iterativo. A medida que la aplicación evoluciona, se realizarán ajustes y mejoras en respuesta a la retroalimentación de los usuarios y a las tendencias del diseño.

## 6 - Fase de Testing y Aseguramiento de Calidad

### 6.1 Sección de Pruebas Unitarias

#### 6.1.1 Introducción al Testing Unitario

En este proyecto, utilizamos pruebas unitarias para asegurar la calidad y el correcto funcionamiento de los componentes individuales de nuestra aplicación. Las pruebas unitarias son esenciales para identificar errores en las etapas tempranas del desarrollo, facilitar la refactorización del código y garantizar que las nuevas características no afecten a las funcionalidades existentes.

#### 6.1.2 Herramientas Utilizadas

Para la implementación de nuestras pruebas unitarias, empleamos las siguientes herramientas:

JUnit 5: Framework de pruebas para Java que permite la creación de pruebas repetibles con anotaciones y aserciones fáciles de usar.

Mockito: Framework de simulación que nos permite crear y configurar objetos mock (simulados) para aislar las dependencias de las clases que estamos probando.

#### 6.1.3 Estructura de las Pruebas

Nuestras pruebas unitarias están organizadas de acuerdo con la estructura de paquetes del proyecto. Para cada clase en el paquete `src/main/java`, existe una clase de prueba correspondiente en el paquete `src/test/java`.

Por ejemplo, las pruebas para `UsuarioService` se encuentran en:

Clase a probar:

[src/main/java/trabajofinal/acceso\\_datos\\_fianl/usuario/service/UsuarioService.java](#)

Clase de prueba:

[src/test/java/trabajofinal/acceso\\_datos\\_fianl/usuario/service/UsuarioServiceTest.java](#)




### 6.1.4 Ejecución de las Pruebas

Para ejecutar las pruebas unitarias, puedes utilizar tu IDE favorito que soporte JUnit, como IntelliJ IDEA, Eclipse, o Visual Studio Code, o ejecutarlas desde la línea de comandos con Maven o Gradle, dependiendo de la herramienta de construcción que estés utilizando en tu proyecto.

#### Ejemplo con Maven:


bash

 Copy code

```
mvn test
```

#### Ejemplo con Gradle:

bash

 Copy code

```
./gradlew test
```

## 6.2 Añadir Nuevas Pruebas

Para mantener la calidad del código a medida que el proyecto evoluciona, es crucial añadir pruebas unitarias para las nuevas funcionalidades o componentes que se desarrollen. Al añadir nuevas pruebas, considera las siguientes buenas prácticas:

**Nombre Descriptivo:** Elige nombres descriptivos para tus métodos de prueba que reflejen claramente lo que están probando.

**Independencia:** Asegúrate de que cada prueba sea independiente y no dependa del resultado de otras pruebas.

**Cobertura:** Intenta cubrir tanto los casos de uso típicos como los casos extremos o inusuales.

**Uso de Mocks:** Utiliza objetos mock para simular las dependencias de la clase que estás probando, lo que te permite probar cada unidad de forma aislada.

## 6.3 Ejemplo de una Prueba Unitaria

Aquí tienes un ejemplo de cómo se ve una prueba unitaria típica en nuestro proyecto, utilizando JUnit y Mockito:

```
java Copy code

@Test
void createUsuario_Success() {
    // Preparar datos de entrada y mocks necesarios
    UsuarioDTO usuarioDTO = new UsuarioDTO("nombreUsuario", "correo@example.com");
    when(usuarioRepository.save(any(Usuario.class))).thenReturn(new Usuario());

    // Ejecutar el método que se está probando
    Integer usuarioId = usuarioService.create(usuarioDTO);

    // Verificar los resultados y las interacciones con los mocks
    assertNotNull(usuarioId);
    verify(usuarioRepository, times(1)).save(any(Usuario.class));
}
```

## 6.4 Conclusión

Las pruebas unitarias son un componente esencial de nuestro flujo de trabajo de desarrollo de software. Siguiendo las prácticas y estructuras descritas en esta sección, puedes contribuir eficazmente a mantener y mejorar la calidad y estabilidad de nuestra aplicación.

## 7 - Implementación de Logging y Monitorización

La implementación de logging en el Sistema de Organizador de Eventos es fundamental para asegurar la trazabilidad de las acciones, diagnosticar problemas y optimizar la experiencia del usuario. A continuación, se detalla la configuración y el enfoque adoptado para la integración del sistema de logging en nuestra aplicación.

### 7.1 Configuración del Sistema de Logging

#### 7.1.1 Biblioteca de Logging

Para la implementación del logging, se ha utilizado SLF4J (Simple Logging Facade for Java) en combinación con Logback, aprovechando su integración nativa en el framework Spring Boot. Esta combinación ofrece una solución de logging potente y flexible, permitiendo una configuración detallada a través de archivos externos y soportando múltiples niveles de log.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

#### 7.1.2 Niveles de Logging

Los niveles de logging se han definido de la siguiente manera para facilitar la clasificación y el análisis de los eventos registrados:

**DEBUG:** Información detallada que facilita el diagnóstico de problemas durante el desarrollo y en entornos de prueba.

**INFO:** Eventos significativos que reflejan el flujo normal de la aplicación, como el inicio de operaciones o la realización de tareas clave.

**WARN:** Situaciones que podrían indicar un problema potencial o cambios inesperados en el comportamiento esperado de la aplicación.

**ERROR:** Errores que afectan a la operación de la aplicación pero que no impiden su funcionamiento general.

**FATAL:** Errores críticos que pueden comprometer la integridad o la disponibilidad del sistema, requiriendo atención inmediata.

## 7.2 Implementación en la Aplicación

La implementación del sistema de logging se ha realizado siguiendo las mejores prácticas y se ha integrado en puntos estratégicos de la aplicación para capturar eventos relevantes.

### 7.2.1 Puntos de Integración

Controladores: Se han añadido registros en los controladores HomeController, EventoController y UsuarioController para capturar eventos como accesos a endpoints, resultados de operaciones y manejo de errores.

Ejemplos de registros implementados:

Entrada a endpoints: `logger.info("Accediendo a la página de inicio de sesión");`

```
2024-03-01 11:14:14 - Accediendo a la página de inicio de sesión
```

Resultado de operaciones: `logger.info("Registro de usuario exitoso");`

```
2024-03-01 11:15:20 - Usuario añadido exitosamente
```

Errores: `logger.error("Errores en el formulario de registro");`

Servicios y Repositorios: Aunque el enfoque principal ha sido en los controladores, se recomienda la extensión del logging a los servicios y repositorios para una trazabilidad más detallada de la lógica de negocio y el acceso a datos.

### 7.2.2 Consideraciones de Seguridad

Se ha tenido especial cuidado en no registrar información sensible como contraseñas o datos personales, para cumplir con las normativas de protección de datos y seguridad de la información.

## 7.3 Monitorización

Aunque la monitorización no se ha implementado en esta fase, se recomienda la integración con herramientas como Splunk, ELK Stack o Prometheus para el análisis en tiempo real de los logs y la monitorización del rendimiento de la aplicación.

## 7.4 Mejoras Futuras

Para futuras iteraciones, se contempla la integración de funcionalidades avanzadas como:

Trazas distribuidas: Para seguir las peticiones a través de microservicios o componentes distribuidos.

Alertas automatizadas: Configurar alertas basadas en patrones específicos de logs o niveles de error para una respuesta rápida ante incidentes.

Análisis de logs: Integración con herramientas de análisis de logs para obtener insights operacionales y de negocio a partir de los datos registrados.

## 7.5 Ejemplos Visuales

```
7
Pedriiii10
@PostMapping("/register")
public String registerUser(@ModelAttribute("usuario") @Valid UsuarioDTO usuarioDTO,
                           BindingResult bindingResult, RedirectAttributes redirectAttributes) {
    if (bindingResult.hasErrors()) {
        logger.error("Errores en el formulario de registro");
        return "home/register";
    }
    logger.info("Registro de usuario exitoso");
}
```

```
@PostMapping("/add")
public String add(@ModelAttribute("usuario") @Valid final UsuarioDTO usuarioDTO,
                 final BindingResult bindingResult, final RedirectAttributes redirectAttributes) {
    if (bindingResult.hasErrors()) {
        // Aquí se deberían incluir los errores en el RedirectAttributes para que estén disponibles después de la redirección
        redirectAttributes.addFlashAttribute(attributeName: BindingResult.MODEL_KEY_PREFIX + "usuario", bindingResult);
        redirectAttributes.addFlashAttribute(attributeName: "usuario", usuarioDTO);
        logger.error("Errores en el formulario de añadir usuario");
        return "redirect:/register"; // Redirige de nuevo a la página de registro
    }
    userService.create(usuarioDTO);
    redirectAttributes.addFlashAttribute(WebUtils.MSG_SUCCESS, WebUtils.getMessage(code: "usuario.create.success"));
    logger.info("Usuario añadido exitosamente");
    return "redirect:/";
}
```

## 8. Manejo de Errores

El adecuado manejo de errores es esencial para mantener la estabilidad y la usabilidad de nuestra aplicación de gestión de eventos. Nuestra estrategia de manejo de errores se enfoca en proporcionar retroalimentación clara al usuario, registrar los errores para análisis posteriores y, cuando sea posible, ofrecer una vía para la recuperación o la continuación segura de la operación.

### 8.1 Principios Generales

**Prevención de Errores:** Donde sea posible, prevenimos errores mediante validaciones de entrada y lógica de negocio robusta.

**Captura y Manejo:** Los errores inesperados se capturan y se manejan de manera que el usuario reciba una notificación comprensible y no se revele información sensible.

**Registro y Monitorización:** Todos los errores se registran con suficiente detalle para permitir un diagnóstico y corrección efectivos.

### 8.2 Tipos de Errores

Nuestra aplicación puede encontrar principalmente dos tipos de errores:

**Errores del Cliente (4xx):** Estos errores ocurren cuando las solicitudes del cliente son erróneas o incompletas. Ejemplos comunes incluyen el error 404 Not Found cuando se solicita un recurso que no existe y el error 400 Bad Request cuando la solicitud no puede ser procesada.

**Errores del Servidor (5xx):** Estos errores son el resultado de condiciones inesperadas en el lado del servidor, como un error 500 Internal Server Error para excepciones no capturadas en el código.

### 8.3 Manejo de Errores en la Aplicación

#### 8.3.1 Errores del Cliente

Para el manejo de errores del cliente, se implementan las siguientes estrategias:

**Validación de formularios** tanto en el lado del cliente como en el servidor para prevenir entradas inválidas.

**Páginas de error personalizadas** que informan al usuario sobre el problema y ofrecen enlaces para volver o realizar otras acciones.

### 8.3.2 Errores del Servidor

El manejo de errores del servidor sigue estos pasos:

Un manejador global de excepciones captura cualquier error no capturado en el flujo normal de la aplicación.

Los usuarios reciben una página de error genérica que no revela detalles del stack de la excepción para evitar la exposición de información sensible.

Los detalles del error se registran en un archivo de log para revisión por parte del desarrollador o el administrador del sistema.

## 8.4 Registro de Errores

Utilizamos el framework de logging integrado en Spring Boot para registrar errores. La configuración por defecto registra los errores en la consola y en archivos de log rotativos, asegurando que tengamos un registro histórico sin consumir espacio de almacenamiento excesivo.

## 8.5 Personalización y Extensión

Los desarrolladores pueden personalizar el manejo de errores para añadir lógica específica de la aplicación o para integrarse con sistemas de monitorización externos. Esto se puede realizar extendiendo la clase `ResponseEntityExceptionHandler` y anotando la nueva clase con `@ControllerAdvice` para manejar excepciones de manera global.

## 8.6 Pruebas y Validación

El manejo de errores debe ser probado exhaustivamente para asegurar que los casos de error se manejan como se espera. Esto incluye pruebas unitarias y de integración que simulan condiciones de error y verifican las respuestas de la aplicación.

## 9. Accesibilidad y Localización

La accesibilidad y localización son aspectos cruciales de nuestra aplicación de gestión de eventos, que aseguran que todos los usuarios puedan encontrar fácilmente el lugar donde se llevarán a cabo los eventos y que la información sea comprensible para una audiencia global.

### 9.1 Integración con Google Maps

Nuestra aplicación incluye una integración con Google Maps, proporcionando a los usuarios una manera intuitiva y fácil de visualizar las ubicaciones de los eventos:

**Enlaces Directos:** Cada evento listado en la plataforma tiene un enlace directo a Google Maps, permitiendo a los usuarios ver la ubicación exacta con solo un clic.

**Mapas Interactivos:** Donde es técnicamente posible, se incorporan mapas interactivos en la página de detalles del evento, ofreciendo una vista previa inmediata de la ubicación.

**Instrucciones de Navegación:** Los usuarios pueden obtener instrucciones detalladas sobre cómo llegar al evento utilizando las funcionalidades de Google Maps.

### 9.2 Diseño y Usabilidad

Hemos diseñado la funcionalidad de mapas pensando en la usabilidad:

**Accesibilidad Móvil:** Los mapas son totalmente responsivos, asegurando que los usuarios puedan acceder a ellos desde cualquier dispositivo, incluyendo smartphones y tabletas.

**Carga Perezosa:** Para mejorar el rendimiento y la velocidad de la página, utilizamos la carga perezosa (lazy loading) de los mapas, lo que significa que solo se cargan cuando el usuario los necesita.

**Alto Contraste y Ampliación:** Los mapas se presentan con opciones de alto contraste y se pueden ampliar para ayudar a los usuarios con visibilidad reducida.



## 9.3 Localización y Soporte Multilenguaje

Entendiendo la importancia de la localización, nuestra aplicación está preparada para soportar múltiples idiomas:

**Etiquetas Multilenguaje:** Los textos y etiquetas en los mapas pueden ser localizados en diferentes idiomas según las preferencias del usuario.

**Formato de Direcciones Localizado:** Las direcciones se muestran en el formato local del usuario, facilitando la comprensión y la navegación.

## 9.4 Consideraciones de Privacidad

La privacidad de los usuarios es una prioridad:

**Información Anónima:** La aplicación no comparte información personal de los usuarios con Google Maps u otros servicios de terceros sin el consentimiento explícito del usuario.

**Cumplimiento de GDPR:** Nos adherimos a las regulaciones de privacidad como el GDPR, asegurando que los datos de localización se manejen adecuadamente.

## 9.5 Instrucciones para Desarrolladores

Para los desarrolladores que trabajan con la funcionalidad de mapas, proporcionamos:

**Documentación de la API:** Instrucciones detalladas sobre cómo utilizar la API de Google Maps dentro de la aplicación.

**Claves de API:** Guía sobre cómo gestionar y proteger las claves de API necesarias para la integración con Google Maps.

## 9.6 Conclusión

La inclusión de Google Maps y la atención a la accesibilidad y localización aseguran que los usuarios puedan interactuar con nuestra aplicación de manera efectiva y sin barreras. Continuaremos mejorando estas características para ofrecer la mejor experiencia posible a nuestra audiencia global.

## 10. Cumplimiento de Aspectos Legales

Nuestra aplicación se compromete a cumplir con todas las leyes y regulaciones aplicables a la privacidad de datos, derechos de autor y accesibilidad. Aquí detallamos cómo abordamos estos aspectos críticos:

### 10.1 Privacidad de Datos y Protección

GDPR y otras regulaciones: Cumplimos con el Reglamento General de Protección de Datos (GDPR) de la UE y otras leyes de protección de datos aplicables.

Política de Privacidad: Proporcionamos una política de privacidad transparente que describe cómo se recopilan, utilizan y protegen los datos personales de los usuarios.  
Consentimiento del Usuario: Obtenemos el consentimiento explícito de los usuarios antes de recopilar datos personales, en línea con las leyes de privacidad.

### 10.2 Derechos de Autor y Contenido de Terceros

Licencias y Permisos: Aseguramos que todo el contenido con derechos de autor, incluyendo imágenes y textos, se utilice con el debido permiso y se acredite adecuadamente.

Contenido Generado por Usuarios: Implementamos políticas para manejar el contenido generado por usuarios, asegurando que se respeten los derechos de autor y la propiedad intelectual.

### 10.3 Accesibilidad Web

WCAG: Nuestro diseño y funcionalidades están en conformidad con las Pautas de Accesibilidad al Contenido en la Web (WCAG) para garantizar que la aplicación sea accesible para todos los usuarios.

Revisión y Mejoras Constantes: Realizamos auditorías periódicas de accesibilidad para identificar y corregir cualquier problema, garantizando la conformidad continua con las regulaciones de accesibilidad.

# 11 - Gestión de Cambios

La gestión de cambios es un aspecto crucial en el desarrollo y mantenimiento del Sistema Organizador de Eventos. Para garantizar la integridad, la estabilidad y la calidad del software, se ha adoptado una metodología estructurada que utiliza GitHub como plataforma central para el control de versiones y la gestión de cambios. A continuación, se detalla el proceso seguido para la gestión de cambios en el proyecto.

## 11.1 Metodología de Trabajo

### 11.1.1 Uso de GitHub

GitHub se ha utilizado como sistema de control de versiones y colaboración, permitiendo al equipo trabajar de manera coordinada y eficiente. Las características clave de GitHub, como las ramas, las pull requests y las revisiones de código, forman la columna vertebral de nuestra gestión de cambios.

Repositorio Utilizado: <https://github.com/Pedriiii10/TrabajoFinalAccesoDatos>

### 11.1.2 Trabajo por Ramas

Para cada nueva característica, corrección de errores o mejoras, se crea una rama dedicada a partir de la rama principal (main o master). El nombre de la rama sigue una convención que refleja su propósito, por ejemplo, feature/nueva-funcionalidad, bugfix/corrección-error, etc.

## 11.2 Proceso de Cambios

### 11.2.1 Creación de Issues

Cada cambio comienza con la creación de un "Issue" en GitHub, donde se detalla el requerimiento, tarea o problema a resolver. Esto permite una planificación y seguimiento efectivo de las tareas pendientes.

### 11.2.2 Desarrollo y Commit

El desarrollo se realiza en la rama específica creada para el issue en cuestión. Los commits deben ser atómicos y su mensaje debe reflejar claramente el cambio realizado, siguiendo buenas prácticas y convenciones establecidas por el equipo.

### 11.2.3 Pull Requests

Una vez completado el desarrollo, se crea una Pull Request (PR) para fusionar los cambios de la rama de desarrollo a la rama principal. La PR debe incluir una descripción detallada de los cambios, enlazar al issue correspondiente y, si es necesario, añadir notas para el revisor o pruebas realizadas.

### 11.2.4 Revisión de Código

La revisión de código es un paso crítico antes de la fusión. Otros miembros del equipo revisan el código en la PR, proporcionando comentarios, sugerencias y aprobando los cambios cuando se consideran listos para ser integrados.

### 11.2.5 Fusión y Cierre de Issues

Una vez aprobada la PR, los cambios se fusionan a la rama principal. Tras la fusión, el issue relacionado se cierra, marcando la finalización del ciclo de cambio para esa tarea específica.

## 11.3 Normativas y Mejores Prácticas

**Pruebas:** Antes de solicitar la fusión de cambios, es esencial realizar pruebas exhaustivas para asegurar que no se introduzcan errores o problemas de regresión.

**Documentación:** Cualquier cambio significativo en la funcionalidad debe acompañarse de la correspondiente actualización en la documentación del proyecto.

**Comunicación:** Mantener una comunicación clara y efectiva dentro del equipo es fundamental para coordinar las tareas y gestionar las dependencias entre cambios.

## 11.4 Herramientas y Recursos Adicionales

**GitHub Actions:** Para automatizar pruebas, integración y despliegue continuo, se recomienda utilizar GitHub Actions o herramientas similares.

**Herramientas de Integración Continua (CI):** Herramientas como Jenkins, CircleCI, o Travis CI pueden integrarse con GitHub para mejorar la calidad del código y automatizar flujos de trabajo.

## 12 - Documentación

La documentación del proyecto de Gestión de Eventos ha sido estructurada cuidadosamente para proporcionar una comprensión clara y completa de la aplicación, tanto a nivel de usuario como de desarrollador. El objetivo es ofrecer una guía exhaustiva que facilite la familiarización con el sistema, su arquitectura, características y código fuente. La documentación se divide en las siguientes secciones principales:

### 12.1. Inicio del Proyecto

Esta sección introductoria establece el alcance del sistema, describiendo el propósito, la audiencia objetivo y las características principales. También proporciona una visión general del modelo de datos y las tecnologías de programación empleadas.

### 12.2. Análisis y Planificación de Requerimientos

Detalla los requisitos funcionales y no funcionales, identificando las capacidades del sistema, estándares de rendimiento y seguridad, así como las restricciones y suposiciones que fundamentan el diseño y desarrollo.

### 12.3. Diseño de la Arquitectura

Presenta la arquitectura Modelo-Vista-Controlador (MVC) del sistema, la estructura de los paquetes del proyecto y las tecnologías utilizadas. Se explican los principios de escalabilidad y mantenibilidad, concluyendo con un resumen del enfoque de diseño.

### 12.4. Modelo de Datos Eficiente

Explica la estructura de la base de datos, las entidades y sus relaciones. Incluye un análisis de la eficiencia del diseño, mejoras futuras y una guía de consultas para interactuar con la base de datos.

## 12.5. Diseño UI/UX

Describe los principios de diseño de la interfaz de usuario, la respuesta del diseño y los elementos de la interfaz, así como las consideraciones de estilo y pruebas de usabilidad.

## 12.6. Fase de Testing y Aseguramiento de la Calidad

Cubre las pruebas unitarias, incluyendo herramientas utilizadas, estructura de pruebas, ejecución y la adición de nuevas pruebas, con ejemplos específicos.

## 12.7. Implementación de Logging

Detalla la configuración del sistema de registro, los niveles de logging, la implementación específica en la aplicación y estrategias de monitorización. Se discuten las potenciales mejoras futuras.

## 12.8. Manejo de Errores

Presenta los principios generales para el manejo de errores, los tipos de errores capturados y cómo se gestionan a lo largo de la aplicación. Incluye también las convenciones de registro y las prácticas recomendadas para pruebas.

## 12.9. Documentación del Código con Javadoc

El código fuente de la aplicación está documentado utilizando Javadoc, que proporciona comentarios descriptivos y etiquetas para clases, interfaces, métodos y variables. Esto incluye:

Descripción General: Una visión global de cada clase y su propósito dentro de la aplicación.

Uso de Etiquetas: Utilización de etiquetas Javadoc como `@param`, `@return`, `@throws`, y `@see` para describir parámetros, valores de retorno, excepciones lanzadas y referencias cruzadas.

Generación de Documentación: Se utiliza la herramienta javadoc para generar automáticamente una documentación HTML navegable a partir del código fuente, facilitando así la comprensión de la estructura y las funciones del sistema.

**Ejemplos de Código:** Donde sea pertinente, se incluyen ejemplos de cómo utilizar las clases y métodos para clarificar su uso.

**Versionado y Autoría:** Cada clase documentada con Javadoc contiene metadatos sobre su versión y autor, asegurando el seguimiento de cambios y contribuciones.

La documentación generada por Javadoc es una parte integral de nuestra base de conocimiento y sirve como referencia rápida para los desarrolladores, permitiendo una integración y colaboración efectiva en el proyecto.

## 12.10 Integración y Mantenimiento de la Documentación

Para garantizar la actualidad y relevancia de la documentación técnica, se establecen procedimientos para su continua actualización y revisión:

**Actualizaciones Regulares:** La documentación se revisa y actualiza en paralelo con el ciclo de vida del desarrollo para reflejar los cambios y las nuevas características del sistema.

**Revisión por Pares:** Las actualizaciones de la documentación están sujetas a revisión por pares, asegurando la precisión y la claridad del contenido.

**Accesibilidad:** La documentación está disponible para todo el equipo de desarrollo y partes interesadas a través de un sistema de control de versiones, asegurando el acceso a la versión más reciente.

## 12.11 Accesibilidad y Localización

La accesibilidad y localización son aspectos cruciales de nuestra aplicación de gestión de eventos, que aseguran que todos los usuarios puedan encontrar fácilmente el lugar donde se llevarán a cabo los eventos y que la información sea comprensible para una audiencia global.

## 12.12 Cumplimiento de Aspectos Legales

Nuestra aplicación se compromete a cumplir con todas las leyes y regulaciones aplicables a la privacidad de datos, derechos de autor y accesibilidad.

## 12.13 Gestión de cambios

La gestión de cambios es un aspecto crucial en el desarrollo y mantenimiento del Sistema Organizador de Eventos. Para garantizar la integridad, la estabilidad y la calidad del software, se ha adoptado una metodología estructurada que utiliza GitHub como plataforma central para el control de versiones y la gestión de cambios. A continuación, se detalla el proceso seguido para la gestión de cambios en el proyecto.

## 12.13 Conclusión

La documentación proporcionada es un recurso exhaustivo diseñado para facilitar una comprensión profunda de la aplicación de gestión de eventos. Se anima a los desarrolladores y colaboradores a utilizar esta documentación como una guía práctica para el desarrollo y la colaboración en el proyecto, y a contribuir a su mejora continua.



