Memoria Fallap

Índex

1-Marco del proyecto

1.1-Descripción del proyecto

Resumen de Fallap:

Fallap es una aplicación web responsive diseñada específicamente para el campo de las Fallas. Nuestro objetivo principal es ofrecer una solución integral que cubra las necesidades de los casales falleros, que actualmente carecen de herramientas digitales adecuadas para su gestión.

Mediante Fallap, los falleros y falleras tendrán acceso a una plataforma donde podrán agrupar todas las necesidades relacionadas con su casal. Desde comentarios y recordatorios hasta el pago de eventos organizados, Fallap simplifica los trámites y evita la necesidad de contar siempre con una persona encargada de gestionar los pagos y recordatorios de los miembros.

Además, en el muro de Fallap los usuarios encontrarán información relevante sobre eventos, encuestas y comentarios relacionados con su casal, lo que facilita la comunicación y la interacción entre los miembros.

Una de las ventajas destacadas de Fallap es su enfoque en la gestión y administración del casal por parte del administrador. El sistema se desarrolla utilizando tecnologías modernas y escalables, lo que significa que no es necesario depender de sistemas o módulos externos costosos. Solo se requiere una base de datos y un servidor para alojar los archivos de la aplicación web, lo cual se ofrecerá como servicio directamente al cliente una vez se decida implantarlo.

En resumen, Fallap ofrece una solución digital integral y accesible para los casales falleros, mejorando la gestión, la comunicación y la administración del casal de manera rápida y eficiente.

Fallap abstract:

Fallap is a responsive web application specifically designed for the field of Fallas. Our main objective is to provide a comprehensive solution that meets the needs of casales falleros, which currently lack suitable digital tools for their management.

Through Fallap, falleros and falleras will have access to a platform where they can gather all the requirements related to their casal. From comments and reminders to event payments, Fallap streamlines procedures and eliminates the need for someone to constantly manage member payments and reminders.

Furthermore, on the Fallap wall, users will find relevant information about events, surveys, and comments related to their casal, enhancing communication and interaction among members.

One notable advantage of Fallap is its focus on casal management and administration by the administrator. The system is developed using modern and scalable technologies, eliminating the dependence on expensive external systems or modules. Only a database and server are required to host the web application files, which will be offered as a service directly to the client once the decision to implement it is made.

In summary, Fallap offers a comprehensive and accessible digital solution for casales falleros, improving management, communication, and administration of the casal in a quick and efficient manner.

1.2- Objetivos:

1. Proporcionar al colectivo fallero una herramienta ágil y sencilla para mantener al día todo lo relacionado con su casal u organización.

2. Agrupar las acciones más comunes realizadas de manera convencional y ponerlas al alcance de un clic.
3. Permitir a los administradores crear, gestionar y controlar el estado del casal mediante una navegación sencilla en la aplicación.
4. Mantener a los usuarios informados sobre las últimas noticias, eventos y encuestas relacionadas con su casal.
5. Agilizar y hacer más fiable la toma de decisiones mediante la participación de los usuarios en encuestas.
6. Facilitar el pago de eventos desde dispositivos móviles, evitando la necesidad de hacerlo en persona y detenerse en doble fila.

1.3- Tipo de proyecto:

En cuanto al ámbito, se trata de un proyecto de carácter interno, enfocado en la mejora y optimización de las operaciones dentro de la organización.

Cuando menciono "ámbito interno" en el contexto del proyecto, me refiero a que está dirigido a mejorar y optimizar las operaciones y procesos internos dentro de la organización misma. Esto implica que el proyecto tiene como objetivo principal beneficiar y brindar soluciones a los miembros y las actividades internas de la organización en cuestión, en lugar de tener un enfoque externo dirigido a clientes o usuarios externos.

En el caso de Fallap, al ser una herramienta diseñada para casales falleros, se consideraría un proyecto de ámbito interno, ya que su objetivo es proporcionar una solución digital a los falleros y falleras para mejorar la gestión y administración de su casal, facilitando las comunicaciones internas, el pago de eventos y la participación en encuestas relacionadas con el casal.

1.4- Orientaciones para el desarrollo y recurso:

Para llevar a cabo el proyecto he tenido que pedir ayuda en varias ocasiones. Las fuentes que me han servido en esta labor han sido:

- Para el diseño con Figma de la interfaz he solicitado orientación a un amigo Diseñador gráfico que me ha sabido aconsejar en temas estéticos y me ha explicado como utilizar correctamente la herramienta. También me he servido de la documentación.
- Para el desarrollo del backend he recurrido a mi tutor Ricardo Sanchez y a la documentación oficial de Symfony.
- En cuanto al frontend, al utilizar la api de composición de Vue 3 he tenido que recurrir a muchos foros y comunidades de desarrolladores debido a que es una tecnología bastante moderna y la documentación oficial, en mi opinión, tiene ciertas carencias en explicaciones y ejemplificación.
- Para la arquitectura de la base de datos también pedí ayuda a Ricardo Sánchez, sobre todo con la elaboración de los diagramas y las relaciones.
- Para el uso de Vuetify he tenido que leer bien la documentación que está bien explicada y me resultó bastante entendible.

A continuación enumerare la lista de documentación empleada para el desarrollo:

Tecnologías backend

Tecnología	Sitio Web	Versión
JavaScript	JS-Docs	ES2022
Vue	VUE-Docs	3
Vue-API	API-Composition-Docs	Composition
Vuex	Vuex-Docs	4.x
Vue-Router	Router-Docs	4.x
Vuetify	Vuetify-Docs	3
Bootstrap	BootStrap-Docs	5.3
Figma	Figma-Docs	9.0
Vite	Vite-Docs	3
Npm	NPM-Docs	

Tecnologías frontend

Tecnología	Sitio Web	Versión
Php	PHP-Docs	8
Symfony	Symfony-Docs	6.2
Composer	Composer-Docs	2.5
Nelmio	Nelmio-Docs	4.0
Lexikt-Jwt	Lexikt-jwt-Docs	2.x
Rest-bundle	FosRest-Docs	3.x
Doctrine	Doctrine-Docs	2.15.2
MariaDb	MariaDB-Docs	10.4.28
PhpMyAdmin	PhpMyAdmin-Docs	5.2.1
OpenSSL	OpenSSL-Docs	1.1.1t

2-Analisis del estado actual

2.1. Descripción del sistema actual:

El sistema actual, conocido como Fallap, es un sitio web que permite gestionar las fallas de una ciudad y organizarse para llevar a cabo diferentes actividades y eventos. El sistema cuenta con diferentes roles de usuario, como SuperAdmin, Admin y Usuario Falla.

El SuperAdmin tiene acceso a un tablón de noticias, desde donde puede publicar noticias relevantes para todos los usuarios de Fallapp. Además, puede crear fallas, usuarios y administradores, importar usuarios mediante un archivo CSV y gestionar los premios entregados anualmente.

Los Admins de cada falla pueden crear nuevos usuarios, crear eventos con opciones de pago, gestionar pagos, añadir comentarios al feed de la falla y crear votaciones o encuestas. También pueden contestar las encuestas y realizar pagos.

Los Usuarios Falla pueden visualizar todo, suscribirse a eventos, realizar pagos, contestar encuestas y descargar archivos.

2.2. Viabilidad del sistema actual:

Tras evaluar el sistema Fallap en función de los objetivos planteados, se ha determinado que el sistema actual es viable para satisfacer las necesidades y expectativas de gestionar las fallas de una ciudad y organizar actividades y eventos. Sin embargo, se identifican oportunidades de mejora en la experiencia de los usuarios y en la funcionalidad del sistema, lo que justifica la realización de mejoras y el desarrollo de nuevas funcionalidades.

2.3. Requerimientos del nuevo sistema:

A partir del análisis del sistema actual, se establecen los siguientes requerimientos para el nuevo sistema Fallapp:

1. Tablón de noticias para el SuperAdmin: El nuevo sistema debe contar con un tablón de noticias en el panel de administración, que permita al SuperAdmin publicar noticias relevantes para todos los usuarios de la plataforma. Cada noticia debe incluir un título, descripción o texto, imagen de portada.
2. Creación de nuevo usuario para el Admin: Se debe proporcionar la funcionalidad para que los Admins puedan crear nuevos usuarios en la plataforma, ingresando su nombre completo, correo electrónico y teléfono de contacto.
3. Importación de usuarios mediante CSV para el SuperAdmin: Se debe facilitar al SuperAdmin la importación masiva de usuarios mediante un archivo CSV, que contenga los campos necesarios como nombre completo, correo electrónico y teléfono de contacto.
4. Creación de eventos para el Admin: El nuevo sistema debe permitir a los Admins crear eventos en la plataforma, especificando título, descripción, archivo adjunto (opcional), fecha, imagen de portada y la opción de establecer un pago para asistir al evento.
5. Gestión de pagos por parte del Admin: Se debe permitir a los Admins agregar pagos a los eventos que lo requieran. Deben poder especificar título, descripción breve, fecha límite de pago, lista de usuarios que han pagado y lista de usuarios pendientes de pago. Los Admins también deben tener la opción de pagar en nombre de otros usuarios o usuarios externos a la falla.
6. Añadir comentarios para el Admin: Los Admins deben poder agregar comentarios en las publicaciones de la plataforma, incluyendo texto e imágenes.
7. Creación de votaciones/encuestas para el Admin: Los Admins deben tener la posibilidad de crear votaciones o encuestas en la plataforma, especificando el tema o pregunta, las opciones de respuesta, la fecha límite para votar y un contador de votos.
8. Edición de perfil y suscripción a eventos para usuarios de falla: Se debe permitir a los usuarios de cada falla editar su perfil en Fallapp, incluyendo su nombre completo, falla, descripción breve, nombre de usuario, foto de perfil, teléfono y contraseña. Además, los usuarios deben poder suscribirse a eventos de la plataforma, confirmando su asistencia mediante un botón.
9. Realización de pagos para usuarios de falla: Los usuarios de cada falla deben tener la opción de realizar pagos en la plataforma, a través de una pasarela de pago que se discutirá y definirá en detalle.

3-Diseño de la solución

3.1. Análisi de las posibles soluciones

Solución 1: Desarrollo a medida

- Descripción: Desarrollar una aplicación web personalizada desde cero para satisfacer los requerimientos específicos de Fallap.
- Características:
  - Creación de todos lo componentes desde cero
  - Implementar las funcionalidades a cada componente y hacerlos reutilizables
  - Utilizar twig para realizar la interfaz de usuario

- Ventajas:
  - Solución altamente personalizada que se ajusta a los requerimientos específicos de Fallapp.
  - Posibilidad de adaptar y mejorar continuamente la aplicación a medida que surjan nuevas necesidades.
- Desventajas:
  - Mayor tiempo y costo de desarrollo en comparación con soluciones preexistentes.
  - Requiere recursos técnicos especializados para el desarrollo y mantenimiento.
  - Menor rendimiento.

Solución 2: Desarrollo a medida con el uso de herramientas y tecnologías existentes

- Descripción: Desarrollar una aplicación web personalizada utilizando herramientas y tecnologías existentes, como Figma y Vue.js, para agilizar el proceso de creación, pero manteniendo la flexibilidad necesaria para ajustar y personalizar el sistema según los requerimientos específicos de Fallap.
- Características:
  - Utilización de Figma para diseñar la interfaz de usuario de forma visual y colaborativa.
  - Desarrollo utilizando Vue.js, junto con Vuetify, para acelerar el desarrollo y aprovechar su flexibilidad.
  - Alta seguridad para proteger los datos personales y contraseñas.
- Ventajas:
  - Uso de herramientas y tecnologías existentes para acelerar el proceso de creación.
  - Mayor flexibilidad y personalización en comparación con una solución preexistente.
  - Posibilidad de aprovechar la comunidad y recursos disponibles de Vue.js para el desarrollo.
- Desventajas:
  - Requiere conocimientos técnicos en el uso de Figma y Vue.js para el diseño y desarrollo.
  - Posible necesidad de retocar y ajustar el resultado del conversor de Figma a Vue.js para adaptarlo a las necesidades específicas de Fallap.

3.2- Descripción de la solución elegida:

La solución consiste en desarrollar una aplicación web personalizada utilizando Figma y Vue.js junto a librerias de componentes y estilos. Se aplicarán altos estándares de seguridad y se garantizará la compatibilidad con diferentes navegadores y dispositivos.

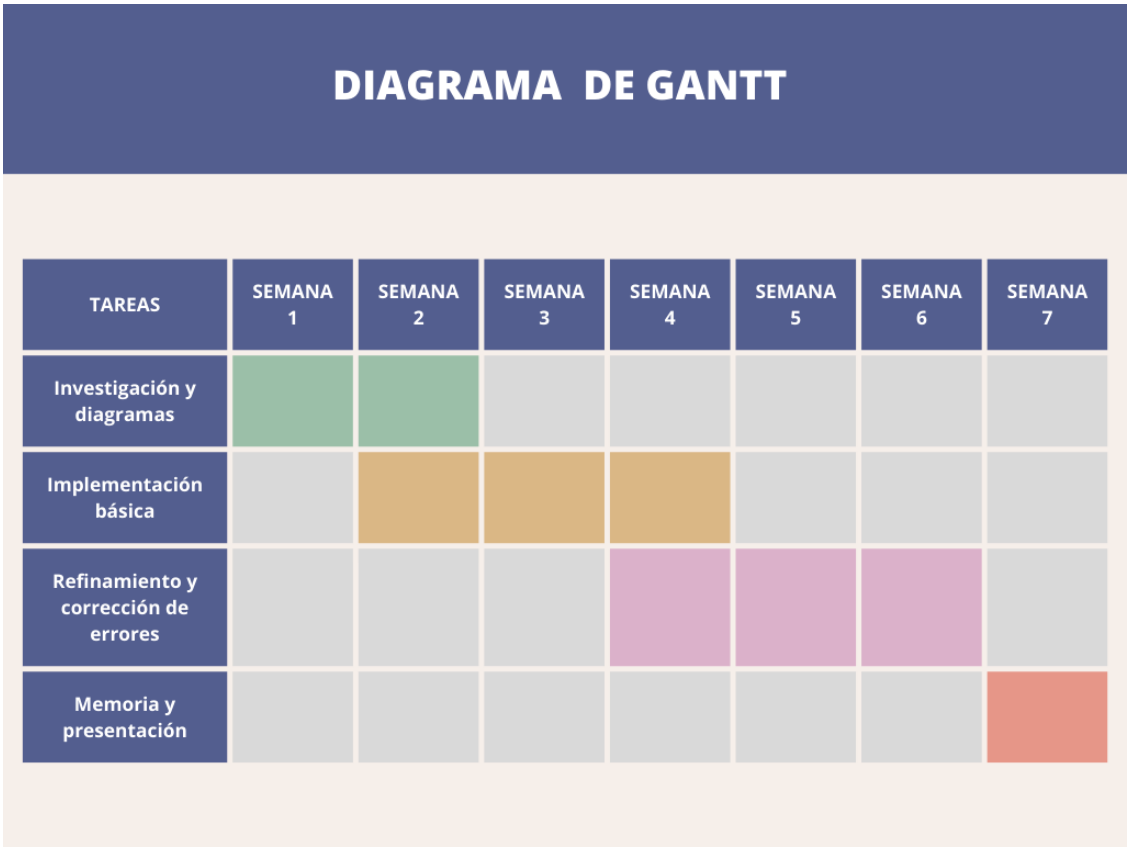
Tareas y subtareas:

1. Análisis y diseño:
   1.1. Reunión con el equipo para definir los requisitos y alcance del proyecto (2 días).
   1.2. Diseño de la arquitectura de la aplicación (1 día).
   1.3. Diseño de la interfaz de usuario utilizando Figma (3 días).
2. Desarrollo:
   2.1. Configuración del entorno de desarrollo (1 día).
   2.2. Implementación del tablón de noticias para el SuperAdmin (5 días).
   2.3. Desarrollo de la funcionalidad de creación de usuarios y eventos para los Admins (7 días).
   2.4. Integración de la importación de usuarios mediante un archivo CSV para el SuperAdmin (3 días).
   2.5. Implementación de la pasarela de pago para los usuarios (7 días).
   2.6. Desarrollo de la funcionalidad de encuestas y votaciones para los Admins (5 días).
   2.7. Implementación de la seguridad y la compatibilidad con diferentes navegadores y dispositivos (4 días).
3. Pruebas y ajustes:
   3.1. Realización de pruebas de funcionamiento y rendimiento (3 días).
   3.2. Ajustes y correcciones necesarias (2 días).
4. Implementación y puesta en marcha:
   4.1. Preparación del entorno de producción (1 día).
   4.2. Despliegue de la aplicación en el entorno de producción (1 día).

Estimación de duración:

Teniendo en cuenta las tareas y subtareas mencionadas, se estima que el desarrollo completo de la solución tomará aproximadamente 50 días hábiles, distribuidos en un período de 2 meses.

A continuación se muestra un posible diagrama de Gantt que ilustra la planificación de las tareas:



4-Desarrollo de la solución

4.1- Recursos disponibles

- Hardware: Ordenador de sobremesa con chip intel i5 y 16gb de memoria RAM
- Software: PhpStorm, WebStorm, Postman, Figma, Notion, Chrome.

4.2- Desarrollo del sistema

Para llevar a cabo el desarrollo, se siguieron los siguientes pasos de manera organizada y sistemática:

1. Análisis de requerimientos:
    - Se realizó un exhaustivo análisis de los requisitos del proyecto.
    - Se entendieron las necesidades y funcionalidades que debían implementarse.
    - Esto permitió tener una visión clara de lo que se esperaba lograr.
2. Creación de diagramas:
    - Se crearon diagramas de casos de uso, diagramas de clases y diagramas entidad-relación.
    - Estos diagramas ayudaron a visualizar y comprender la estructura y las interacciones del sistema antes de comenzar la implementación.
3. Configuración del proyecto:
    - Se utilizó la herramienta Composer para crear un nuevo proyecto de Symfony.

- Se establecieron las dependencias necesarias.
- Se realizó el scaffolding inicial para tener una estructura básica sobre la cual trabajar.

4. Limpieza y configuración:
    - Se realizó una limpieza del proyecto, eliminando componentes innecesarios como Twig.
    - Se instalaron bundles y librerías necesarios para el desarrollo, como FOSRest y Nelmio.
5. Creación de entidades:
    - Con la configuración básica en su lugar, se comenzó a crear las entidades del sistema utilizando el CLI de Symfony.
    - Se generaron las entidades, especificando sus atributos y tipos de datos, y definiendo las relaciones entre ellas.
6. Controladores y rutas:
    - Una vez creadas las entidades, se generó un controlador para cada una de ellas.
    - Los controladores contenían las rutas necesarias para interactuar con las entidades, como obtener, crear, editar y borrar registros.
    - Se realizaron ajustes adicionales, como agregar una ruta para obtener registros por correo electrónico en lugar de por identificador.
7. Repositorios y operaciones con la base de datos:
    - En los repositorios de las entidades, se aprovecharon al máximo los métodos proporcionados por Doctrine para interactuar con la base de datos.
    - Esto permitió realizar operaciones de búsqueda y recuperación de datos utilizando métodos predefinidos, evitando la necesidad de escribir consultas SQL personalizadas en la mayoría de los casos.
8. Pruebas y correcciones:
    - Una vez completada la implementación de las rutas y los controladores, se realizaron pruebas exhaustivas del sistema.
    - Se utilizaron herramientas como Postman y YARC para enviar solicitudes HTTP y verificar que todas las funcionalidades se comportaran correctamente.
    - Durante esta etapa, se identificaron posibles fallos y se realizaron las correcciones necesarias para garantizar el correcto funcionamiento de todo el sistema.

   Además, se trabajó en paralelo en el diseño de la interfaz utilizando Figma:

9. Diseño de la interfaz con Figma:
    - Se definió la fuente de letra y los tamaños de texto que se utilizarían en la interfaz.
    - Se seleccionaron los colores y los iconos necesarios para crear una apariencia visual coherente.
    - Se siguió la metodología de "Mobile First" y se diseñaron inicialmente los componentes y las vistas para dispositivos móviles.

- Este enfoque permitió abordar la adaptación de la interfaz a pantallas de escritorio de manera más eficiente.
- Se realizaron cambios y modificaciones continuas, adaptándose a medida que se refinaban los detalles.

Una vez que el diseño en Figma alcanzó un punto considerado "listo", se procedió a la creación del proyecto de Vue utilizando Vite:

10. Creación del proyecto de Vue con Vite:
    - Se utilizó Vite, una herramienta que facilita la creación de la estructura inicial de un proyecto de Vue.
    - Se enfocó inicialmente en crear los componentes de manera estética, centrándose en la apariencia visual y dejando preparado el terreno para cuando los datos de la API estuvieran disponibles.
    - Gracias al conocimiento previo de los datos que se utilizarían y cómo se integrarían, fue relativamente sencillo adaptar los componentes a medida que los datos de la API se iban incorporando.

4.3- Base de datos

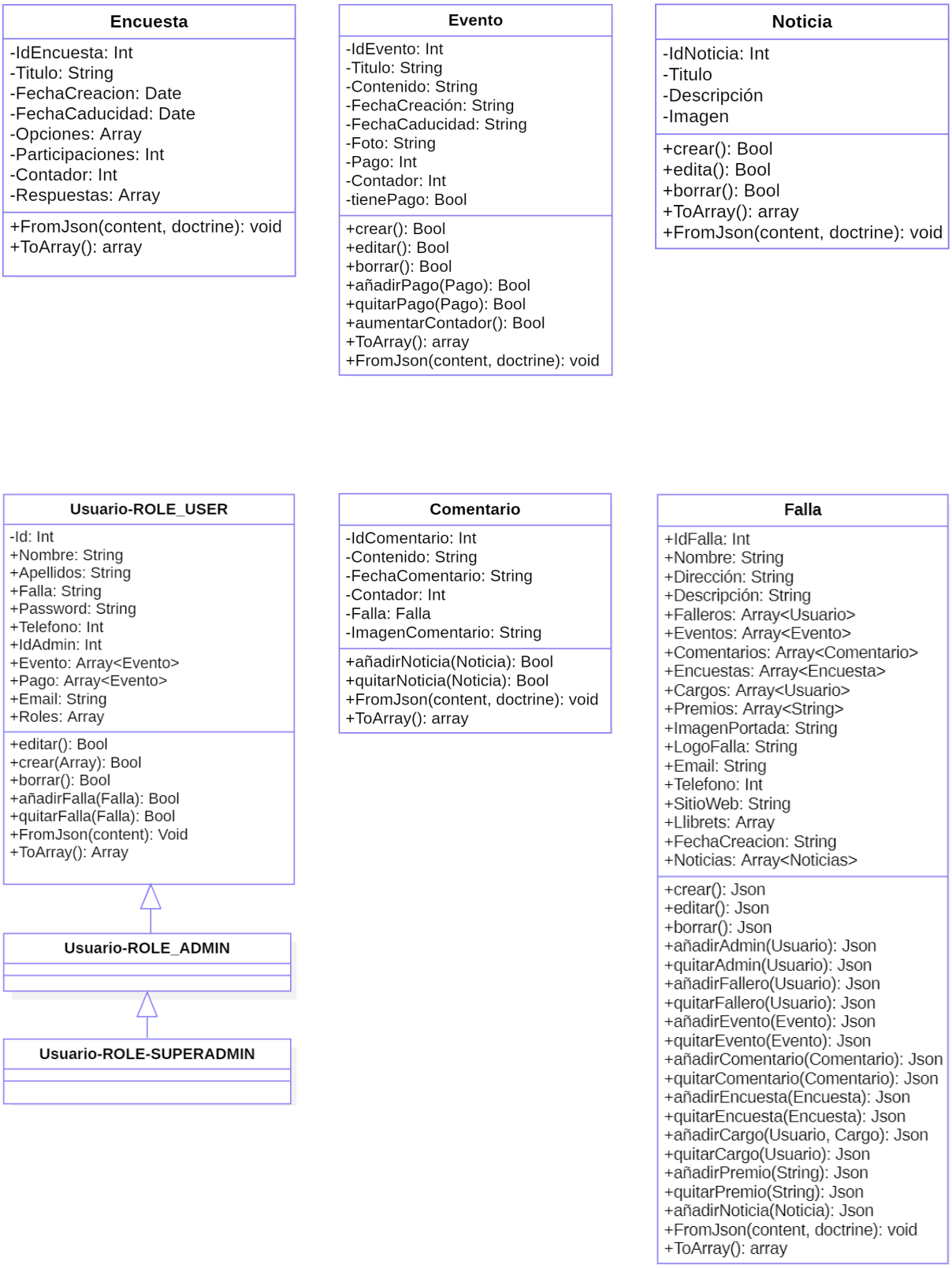
La base de datos escogida para el desarrollo es Mariadb junto con PhpMyAdmin como interfaz grafica para interactuar con ella. Ambas pertenecen al servidor montado para el desarrollo, un xampp, por la facilidad de uso y los conocimientos previos.

Para la conexión desde symfony a la base de datos, deberemos crear un fichero .env.local y colocar la linea de conexión con el host, el nombre de la base de datos, usuario y contraseña.

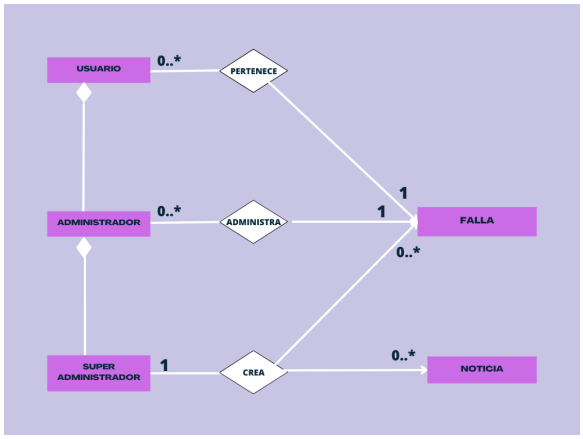
Utilizaré Doctrine, un ORM muy bien integrado con Symfony el cual mapea las clases y trabaja con objetos directamente. El proceso es bastante sencillo, una vez inicializado el proyecto de symfony, desde la consola se lanza el comando symfony console make:entity NombreEntidad, estoy lanzará un dialogo por consola en el cual nos irá preguntando el nombre de las propiedades, el tipo, si puede ser null y en caso de que el tipo sea relación te preguntará que tipo de relación y con que otra entidad quieres relacionarlo. Siguiendo estos pasos he creado todas las entidades que se observan. Una vez creadas, basta con lanzar symfony console make:migrate, lo que te genera un fichero en migrations con las acciones necesarias SQL, para ejecutar este fichero debemos escribir: symfony console doctrine:migrations:migrate, esto crea automáticamente todas las tablas en la base de datos.

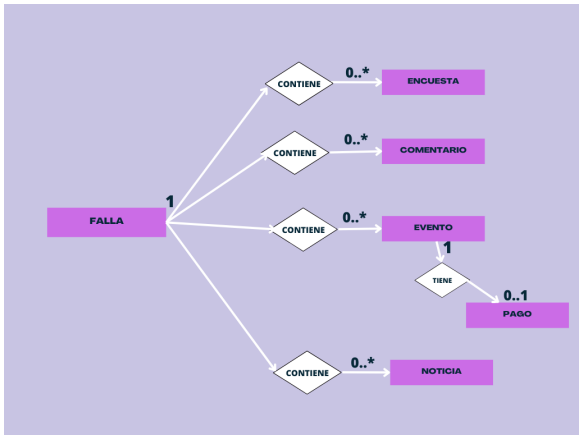
Al crear las entidades, se crea un repositorio para cada una de ellas en el cual vienen implementados métodos para obtener información de la tabla de manera sencilla y también se pueden implementar nuevos métodos para operar con la base de datos.

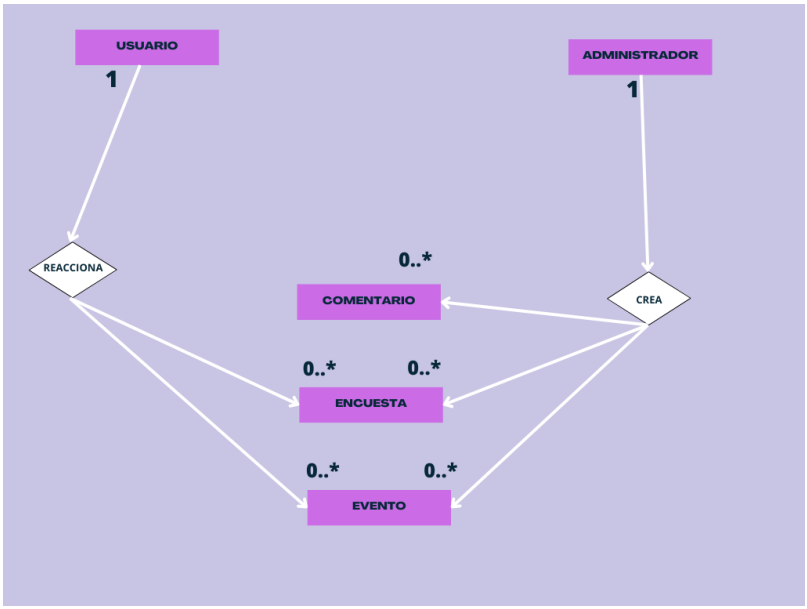
A continuación veremos las clases que he creado para la implementación de la base de datos.



Vemos ahora las relaciones establecidas entre las clases.







Restricciones:

- 0 o muchos Usuarios pueden pertenecer a una Falla
- 0 o muchos Administradores pueden administrar una Falla
- 1 Super administrador puede crear 0 o muchas Fallas
- 1 Super administrador puede crear 0 o muchas Noticias
- 1 Falla puede contener 0 o muchos eventos
- 1 Falla puede contener 0 o muchos comentarios
- 1 Falla puede contener 0 o muchas encuestas
- 1 Falla puede contener 0 o muchas noticias
- 1 Evento puede contener 0 o 1 pago
- 1 Usuario puede reaccionar a 0 o muchas encuestas

- 1 Usuario puede reaccionar a 0 o muchos eventos con y sin pago
- 1 Administrado puede crear 0 o muchos comentarios
- 1 Administrado puede crear 0 o muchos eventos con y sin pago
- 1 Administrado puede crear 0 o muchas encuestas
- 1 Super administrador puede crear 0 o muchas noticias

Una vez hecho esto, creo un controlador para cada una de las entidades en el cual se encontraran todas las rutas necesarias para recibir y enviar datos. Utilizaré la librería de friendsOfSymfony REST, con ella puedo utilizar la anotación de symfony y establecer en la cabecera de la función la ruta y el método HTTP permitido.

4.4- Rutas Backend

Controladores:

-Comentario:

- General: api/comentario
  - Obtener todos los comentarios: GET ⇒ /
    - Retorna un array de objetos con todos los comentarios creados.
  - Crear un nuevo comentario: POST ⇒ /new
    - Recibe un FormData con los datos del comentario
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Actualizar un comentario: POST ⇒ /{id<\d+>}
    - Recibe un FormData con la estructura del comentario y el id del comentario como parámetro
    - Retorna un Json con true o false dependiendo del resultado del update.
  - Borrar un comentario: DELETE ⇒ /{id<\d+>}
    - Recibe el id del comentario a borrar por parámetro
    - Retorna un Json con true o false dependiendo del resultado del delete.

-Encuesta:

- General: api/encuesta
  - Obtener todas las encuestas: GET ⇒ /
    - Retorna un array de objetos con todas las encuestas creadas.
  - Crear una nueva encuesta: POST ⇒ /new
    - Recibe un Json con la estructura de la encuesta
    - Retorna un Json con true o false y el mensaje correspondiente.

- Actualizar una encuesta: POST ⇒ /{id<\d+>}
  - Recibe un Json con la estructura de la encuesta y el id de la encuesta como parámetro
- Reaccionar a una encuesta: PUT ⇒ /react
  - Recibe un Json con la respuesta y el id de la encuesta y lo actualiza
  - Retorna un Json con true o false y el mensaje correspondiente.
- Borrar una encuesta: DELETE ⇒ /{id<\d+>}
  - Recibe el id de la encuesta a borrar por parámetro
  - Retorna un Json con true o false dependiendo del resultado del delete.

-Evento:

- General: api/evento
  - Obtener todos los eventos: GET ⇒ /
    - Retorna un array de objetos con todos los eventos creados.
  - Crear un nuevo eventos: POST ⇒ /new
    - Recibe un FormData con la estructura del evento
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Actualizar un eventos: POST ⇒ /{id<\d+>}
    - Recibe un FormData con la estructura del evento y el id del evento como parámetro
    - Retorna un Json con true o false dependiendo del resultado del update.
  - Añadir participante: PUT⇒ /setParticipante/{id<\d+>}
    - Recibe un Json con el id del usuario a setear y el id del evento por parámetro
    - Retorna un Json con true o false dependiendo del resultado del update.
  - Borrar un evento: DELETE ⇒ /{id<\d+>}
    - Recibe el id del evento a borrar por parámetro
    - Retorna un Json con true o false dependiendo del resultado del delete.

-Falla:

- General: api/falla
  - Obtener todas las fallas: GET ⇒ /
    - Retorna un array de objetos con todas las fallas creadas.
  - Crear una nueva fallas: POST ⇒ /new
    - Recibe un FormData con la estructura de la falla
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Actualizar una falla: POST ⇒ /{id<\d+>}
    - Recibe un FormData con la estructura de la falla y el id como parámetro

        - Retorna un Json con true o false dependiendo del resultado del update.
    - Obtener fallas para selector: GET ⇒ /select
        - Retorna un array de objetos con el id y el nombre todas las fallas creadas.
    - Obtener falleros para selector: GET ⇒ /falleros/{id<\d+>}
        - Recibe el id de la falla por parámetro
        - Retorna un array de objetos con el id, el nombre y el rol de todos los usuarios de la falla recibida por parámetro.
    - Añadir premios a una falla: POST ⇒ /premio
        - Recibe un Json con los premios y el id de la falla
        - Retorna un Json con true o false dependiendo del resultado del update.
    - Añadir llibret a una falla: POST ⇒ /llibret
        - Recibe un FormData con el llibret y el id de la falla
        - Retorna un Json con true o false dependiendo del resultado del update.
    - Eliminar premio: POST ⇒ /delete-premio
        - Recibe un Json con el premio y el id de la falla
        - Retorna un Json con true o false dependiendo del resultado del delete.
    - Borrar una falla: DELETE ⇒ /{id<\d+>}
        - Recibe el id de la falla a borrar por parámetro
        - Retorna un Json con true o false dependiendo del resultado del delete.

-Noticia:

- General: api/noticia
    - Obtener todas las noticias: GET ⇒ /
        - Retorna un array de objetos con todas las noticias creadas.
    - Crear una nueva noticia: POST ⇒ /new
        - Recibe un FormData con la estructura de la noticia
        - Retorna un Json con true o false y el mensaje correspondiente.
    - Actualizar una noticia: PUT ⇒ /{id<\d+>}
        - Recibe un FormData con la estructura de la noticia y el id como parámetro
        - Retorna un Json con true o false dependiendo del resultado del update.
    - Borrar una noticia: DELETE ⇒ /{id<\d+>}
        - Recibe el id de la noticia a borrar por parámetro
        - Retorna un Json con true o false dependiendo del resultado del delete.

-User:

- Login: /api/login_check
  - La ruta para el login esta definida en el firewall del security.yaml asignando la entidad Usuario al app_user_provider y especificando la propiedad que lo identifica en este caso, el email.
    - Recibe un Json con las credenciales de acceso
    - Si es correcto retorna un token jwt firmado con los datos
    - Si no es correcto retorna error.
- General: api/user
  - Obtener todos los usuarios: GET ⇒ /
    - Retorna un array de objetos con todas los usuarios creados.
  - Obtener un usuario por el identificador: GET ⇒ /id
    - Recibe el email del usuario por parámetro
    - Retorna los datos del usuario o false en formato Json
  - Crear un nuevo usuario: POST ⇒ /new
    - Recibe un Json con la estructura del usuario
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Crear un usuarios desde CSV: POST ⇒ /importCsv
    - Recibe un FormData con los datos necesarios
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Actualizar un usuario: PUT ⇒ /{id<\d+>}
    - Recibe un Json con la estructura del usuario y el id por parámetro.
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Cambiar el rol de un usuario: POST⇒ /setRoles
    - Recibe un Json con el id del usuario y el rol y se encarga de cambiarlo.
    - Retorna un Json con true o false y el mensaje correspondiente.
  - Borrar un usuario: DELETE ⇒ /{id<\d+>}
    - Recibe el id del usuario a borrar por parámetro
    - Retorna un Json con true o false dependiendo del resultado del delete.

Para controlar el CORS y permitir la peticiones al servidor he utilizado un paquete de composer llamado Nelmio, este crea un fichero en la capeta config/packages el cual permite configurar que peticiones, en el fichero .env se crea una variable de entorno que dice la ruta permitida. Este se encargará de controlar todas las peticiones que reciba la api.

4.5- Guard Backend:

Jerarquía de roles:

- Todos los usuarios con el rol ROLE_ADMIN también tendrán el rol ROLE_USER.
- Todos los usuarios con el rol ROLE_SUPER_ADMIN también tendrán los roles ROLE_ADMIN y ROLE_USER. El rol ROLE_ALLOWED_TO_SWITCH define que solo el ROLE_SUPER_ADMIN tiene permiso para cambiar los roles.

Roles:

- ROLE_ADMIN ⇒ ROLE_USER
- ROLE_SUPER_ADMIN ⇒ [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]

Control de acceso por roles:

- Para controlar quién puede realizar peticiones a las distintas rutas de la API, he establecido una jerarquía de rutas dentro de la sección de control de acceso en el archivo security.yaml.
- La primera ruta que coincide afectará a los usuarios, por lo que he establecido el login como la primera ruta para que todos puedan acceder.
- Las rutas se ordenan de más restrictiva a menos restrictiva.

PATH	ROLES	METHODS
^/api/login	PUBLIC_ACCESS	POST
^/api/user/importCsv	ROLE_SUPER_ADMIN	POST
^/api/falla/setPremios	ROLE_SUPER_ADMIN	POST
^/api/user/setRole	ROLE_SUPER_ADMIN	POST
^/api/(noticia | falla)	ROLE_USER	GET
^/api/user/{id}	ROLE_USER	GET
^/api/(noticia | falla | user)	ROLE_SUPER_ADMIN	GET, PUT, POST, DELETE
^/api/user/new	ROLE_SUPER_ADMIN	POST
^/api/(comentario | evento | encuesta)	ROLE_USER	GET, PUT
^/api/(comentario | evento | encuesta)	ROLE_ADMIN	GET, PUT, POST, DELETE
^/api/evento/setParticipante	ROLE_USER	PUT
^/api/pago/setPagador	ROLE_USER	PUT
^/api/encuesta/setRespuesta	ROLE_USER	PUT
^/api	IS_AUTHENTICATED_FULLY	GET, PUT, POST, DELETE

4.6- Rutas Frontend:

PATH	NAME	COMPONENT	PARAMS	AUTH
/	Redirect ⇒ /login			False

PATH	NAME	COMPONENT	PARAMS	AUTH
/login	Login	Login		False
/falla	Home	Falla		True
/sa-actions	Acciones super-admin	SuperAdminActions		True
/a-actions	Acciones admin	AdminActions		True
/new-user	Nuevo usuario	AddUser		True
/new-user-csv	Usuarios desde csv	UsersFromCSV		True
/admin-manage	Manejar administradores	AdminManage		True
/new-premi	Nuevo premio	PremiosManage		True
/new-falla	Nueva falla	AddFalla	:titulo, :edit	True
/create	Crear	Crear		True
/new-llibret	Nuevo llibret	AddLlibrets		True
/wall-falla	Muro falla	WallFalla		True
/wall-notice	Muro noticias	WallNotice		True
/new-event	Nuevo evento	AddEvent	:titulo?, :contenido?, :idEvento?, :pago?, :tienePago?, :fecha?	True
/new-coment	Nuevo comentario	AddComent	:contenido?, :idFalla?, :idComent?	True
/new-enquest	Nueva encuesta	AddEncuesta	:contenido?, :opciones?, :idEncuesta?, :idFalla?, :fecha?	True
/new-notice	Nueva noticia	AddNotice	:titulo?, :contenido?, :idNoticia?, :idFalla?	True

4.7- Guard Frontend:

En el main.js de mi proyecto de vue he implementado un guard que cotrola el acceso a las rutas definidas en el router.

En primer lugar verifica si la ruta a la que se dirige requiere autorización (to.meta.requiresAuth). Luego, obtiene el token de acceso desde el sessionStorage, 1. Decodifica el token utilizando jwtDecode y extrae el rol del usuario desde el token (jwtDecode(token).roles[0]), y comprueba si ha expirado utilizando la función checkTokenExpiration.

Si existe un token válido y no ha expirado:

- Si la ruta de destino es "/sa-actions" y el rol del usuario no es "ROLE_SUPER_ADMIN", redirige a la página de acceso denegado ("/").
- Si la ruta de destino es "/sa-actions" o "/a-actions" y el rol del usuario es "ROLE_USER", redirige a la página de acceso denegado ("/").
- En cualquier otro caso, permite la navegación a la ruta de destino.

Si no hay un token válido o ha expirado, redirige a la página principal ("/") y posiblemente muestra un mensaje de error (comentado en el código).

4.7.1- Main.js:

En el main.js de mi aplicación tengo incorporadas las siguientes librerías de manera global para poder utilizarlas en todo el proyecto sin necesidad de importarlos cada vez.

1. Vuetify
2. Router: En una carpeta aparte, tengo un fichero con los imports de los componentes y otro con las rutas montadas para exportar al main.js, esto evita que se sature este fichero.
3. Store: En un fichero aparte tengo montado todo el store el cual importo a main.js con la misma finalidad que las rutas.

Antes de añadirlos como use, se han configurado al gusto siguiendo las documentaciones, Vuetify por ejemplo te permite elegir el tema a utilizar o la libreria de iconos. En router de deja añadir el modo web history, entre otros ejemplos.

En este fichero tengo también las funciones que comprueba el tiempo de vida del token que son utilizadas en el guard.

createApp(App).use(vuetify).use(router).use(store).mount('#app')

4.7.2- Store de Vuex:

El store de vuex es un espacio de almacenamiento compartido para toda la aplicación lo que facilita la comunicación entre ellos y el intercambio de datos. En mi caso lo utilizo para almacenar los datos del usuario logeado y poder tener a acceso a ellos en diferentes puntos del ciclo de vida. Un inconveniente que tiene el store, es que al recargar la pagina lo reinicia, pero por suerte, apoyándote en el sessionStorage se puede hacer que los datos persistan. Con las siguientes lineas consigo este cometido, comprueba si hay datos, si hay, los sustituye por lo nuevos y cada vez que se produzca una mutación en el store, se guarda el nuevo estado en el sessionStorage con la clave 'myAppStore', convirtiendo el estado en formato JSON antes de guardarlo. Esto se logra utilizando el método subscribe en el almacenamiento store.

```
const savedState = sessionStorage.getItem('myAppStore');
if (savedState) {
    store.replaceState(JSON.parse(savedState));
}

// Guardar el estado en el almacenamiento local cuando se produzca un cambio
store.subscribe((mutation, state) => {
    sessionStorage.setItem('myAppStore', JSON.stringify(state));
});
```

Para el manejo de las peticiones HTTP he decidido utilizar Axios, que es una librería de JavaScript que funciona de manera simple y eficaz y tiene buena integración trabajando con asincronía.

El store esta compuesto por la siguiente estructura:

- State: en el se declaran todas las variables que deseas utilizar.
- Mutations: se utilizan para interactuar con el state, esto se debe a que no se puede asignar un valor directo a una variable declarada en el state por lo que se hacen por medio de las mutations.
- Actions: aquí se declaran los métodos que deseas utilizar desde diferentes puntos de tu aplicación, en mi caso, todas las peticiones Axios se encuentran aquí.
- Getters: son similares a los computed de Vue, sirven para interactuar con información o transformarla de manera dinámica,

Para utilizar el store desde los componentes se debe importar useStore y iniciarlo en una constante, luego se llama: store.dispatch para las actions, store.commit para las mutations.

4.8 -IMPLEMENTACIÓN DE SISTEMAS:

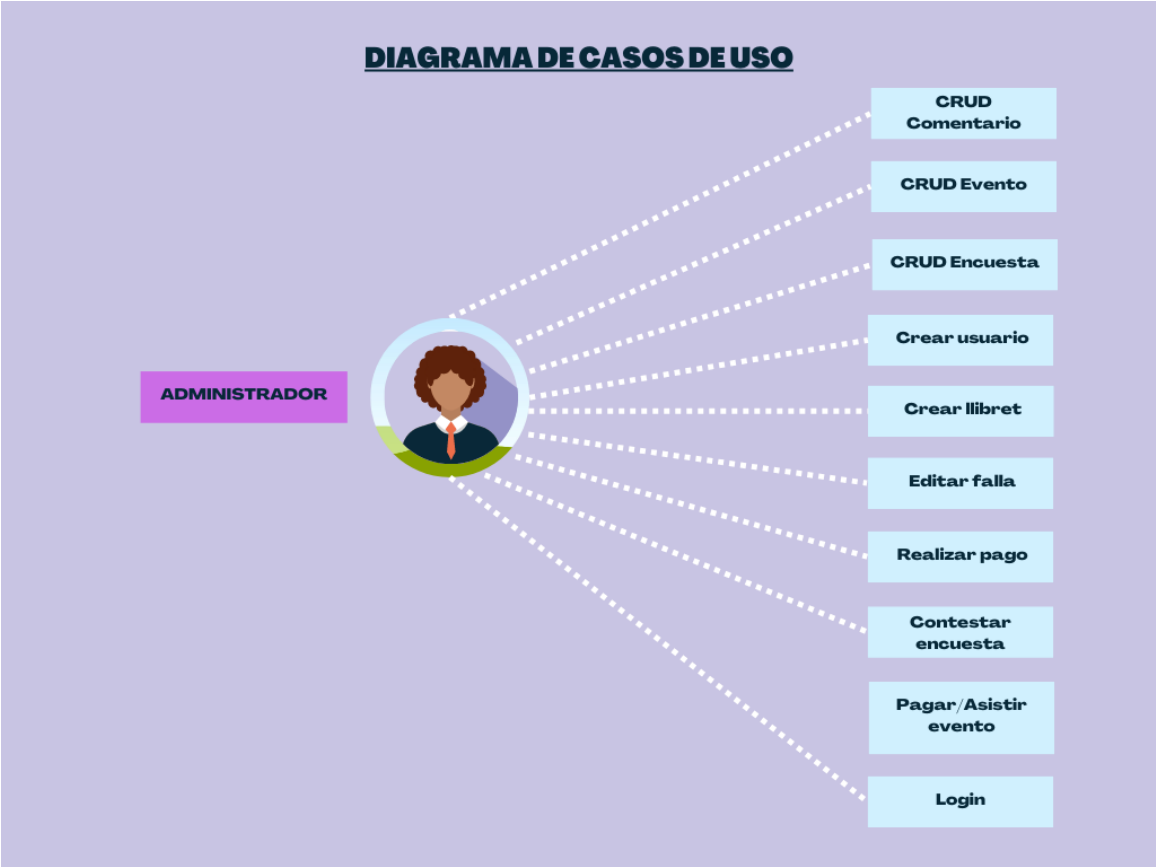
Antes definir cada uno de los sistemas quiero explicar cual es la infraestructura y como se comporta. El backend, diseñado como API REST, se encarga de comunicarse con la bbdd y hacer las operaciones. El frontend hace peticiones al backend, el cual recupera la información y la devuelve. Una vez aclarado esto, empezamos:

1. Login ⇒ La interfaz es un pequeño formulario con dos inputs, uno para el email y otro para la contraseña. Una vez que se introducen las credenciales, estas viajan a la ruta del backend /login_check que se hace lo descrito anteriomente. Si es satisfactorio, te redirige a la pagina principal y almacena el token Jwt y todos los datos en el sessionStorage, esto reduce la carga de peticiones al servidor. La información almacenada no es sensible, son los datos que se mostraran al usuario más adelante. Si no, te salta un error de credenciales erroneas.
2. Acciones Administrador ⇒ Aquí encontraremos un panel con las diferentes acciones que puede realizar el administrador.
    a. Crear: Al pulsar sobre el botón de Crear, se cargará otro submenú con tres opciones:
        i. Evento: Al pulsar, se cargara el formulario para crear un evento, el cual contiene como campos obligatorios: Titulo del evento, Fecha del evento y como opcionales: foto del evento, descripción y un checkbox que al activar te muestra el input para establecer la cantidad a pagar . Una vez sea valido se enviará al servidor para almacenarlo, si va bien mostrara en mensaje de éxito y si sale mal de error
        ii. Comentario: Se carga el formulario de crear comentario. El único campo requerido es la descripción del comentario y tiene un campo para ponerle foto al comentario de manera opcional. Una vez sea valido se enviará al servidor para almacenarlo, si va bien mostrara en mensaje de éxito y si sale mal de error
        iii. Encuesta: Se carga el formulario para crear encuestas, el cual dispone de: Pregunta, Respuestas: se cargan tres inputs default para introducir posibles respuestas a la encuesta. Estos inputs se pueden eliminar si lo deseas, pero para que sea valido tiene que tener por lo menos dos respuestas y por ultimo un input para establecer la fecha de finalización de la encuesta.

b. Actualizar “Llibrets”: Muestra el formulario para almacenar un nuevo “llibret” en formato Pdf, los campos son: Titulo, Archivo pdf y año, y son todos requeridos. En la parte superior de la vista se podran ver los “llibrets” que hay actualmente subidos y si se pulsa en la X que incorporan, se eliminaran, previa confirmación

c. Vincular pagos: Esta vista esta pendiente de ser creada debido a que la implementación de el sistema de pagos se me ha obstaculizado por el tiempo y por el momento queda pendiente. La idea es que muestre un formulario en el cual se pueda ligar con la cuenta de stripe del usuario para que los pagos se reciban ahí.

d. Ver encuetas: Esta vista se trata de un muro en el cual se pueden ver las mismas encuestas que en el muro de la falla, pero con la diferencia de que no se pueden reaccionar, son simplemente informativas dado que aqui se muestra cada respuestas que porcentaje de votos tiene en el momento.



3. Acciones Super Admin ⇒ Veremos un panel similar al de administrador pero con diferentes acciones:

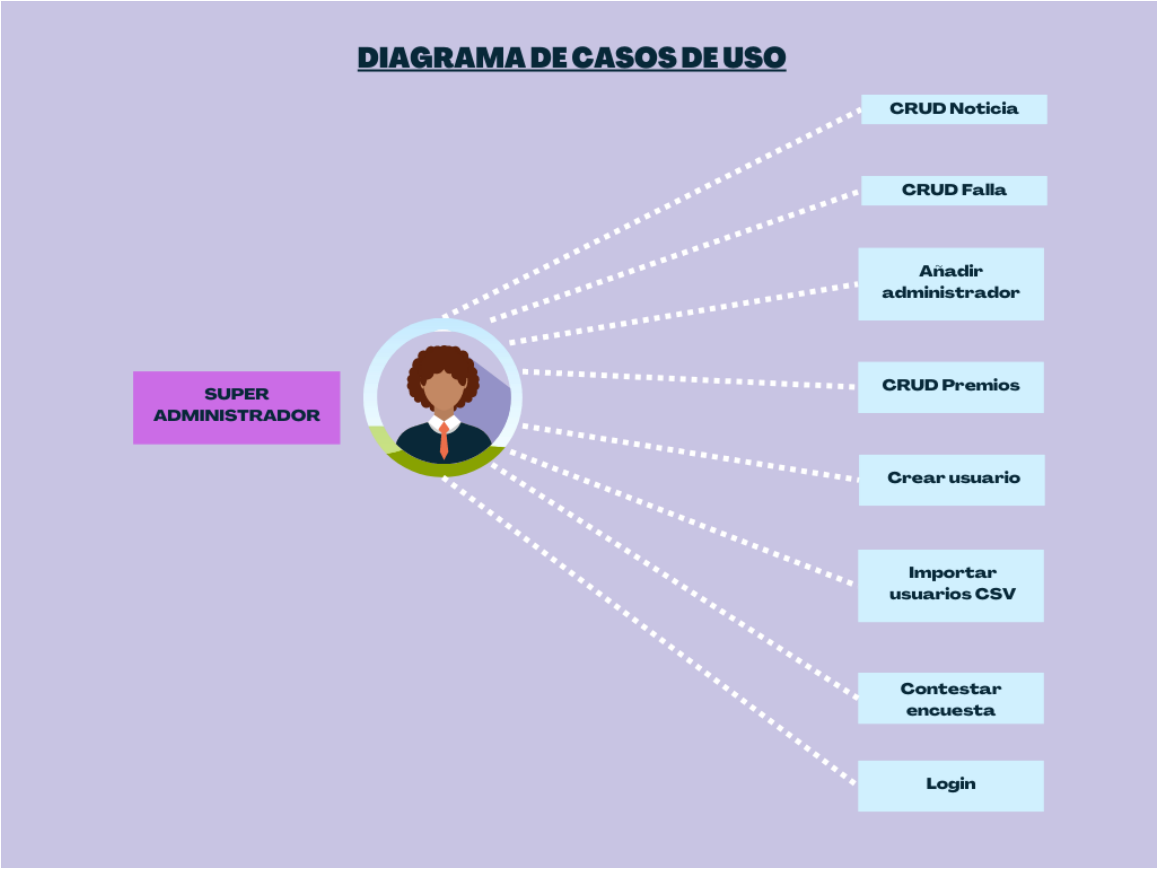
   a. Añadir administrador: Mostrara la vista de AdminManage en la cual se encuentra un selector con todas la fallas disponibles, al seleccionar una de las fallas aparecerá un nuevo selector con todos los usuarios de la falla seleccionada. Al seleccionar un usuario de la lista, el checkbox aparecerá marcado si es administrador y desmarcado si no lo es, al pulsar confirmar se le asignara lo indique el checbox.

   b. Actualizar premios: Se cargará la vista de PremiosManage en la que veremos de primeras tres selectores, el primero elegira la falla a la que se le va a asignar el premio, el segundo

selecciona el tipo de premio, si el premio es la mejor falla, se mostraran bajo dos selectores más, uno para la sección y otro para decir si es infantil o mayor, y por ultimo el selector de la posición, el cual muestra: primera posición, segunda, tercera y sin premio. En la parte superior se mostraran los premios de la falla seleccionada y se borraran con el mismo comportamiento que “llibrets”

c. Crear falla: Mostrará el formulario de la vista AddFalla en el cual, como he dicho antes, solo es requerido el Nombre de la falla. Antes de enviarla de preguntara si esta seguro su se acepta se procederá a hacer el insert. Si el insert fallara se mostrará un mensaje de error.

d. Crear noticia: Las noticias son similares a los comentarios pero tiene añadido un input para el titulo. Las nosticias las creará el superadministrador y se mostraran en un muro dedicado exclusivamente a las noticias. Al ser un array dentro de la entidad falla, he creado una falla con id 99999 llamada Noticias, en ella solamente almacenaré las noticias y las recuperaré para mostrarlas a todos los usuarios.



4. Acciones compartidas admin y superadmin:

a. Añadir usuario: Se cargará el formulario de crear un nuevo usuario, si es super-admin, muestra un selector con todas las fallas disponibles y se selecciona a la que se le quiere insertar. Si es administrador, el selector no se mostrará y se asignara el idFalla almacenado en el store, forzando que solo pueda crear usuarios para su falla. Los campos requeridos son: Nombre, email y la fecha de nacimiento. La contraseña se crea en el servidor mediante un algoritmo creado bits random y se envia al usuario al email proporcionado y se almacena encriptada en la bbdd utilizando el hasher que proporciona el user interface de symfony. El

objetivo de esto es que no se pueden registrar usuarios libremente, solo si pertenecen a una falla.

b. Añadir usuarios desde CSV: En esta vista se repite el comportamiento anterior, si es superadministrador muestra un selector de fallas y si es administrador lo inserta en la falla almacenada en el store. Bajo del selector hay un input tipo file el cual solo acepta documentos csv. El csv ha de tener esta estructura: en las cabeceras las propiedades a insertar, y luego en linia y siguiendo el orden los datos de los usuarios a insertar. Tiene los mismo requerimientos que añadir usuario.

c. Editar falla: Se mostrará la vista de crear falla pero se sustituirá el titulo y el botón por Editar falla y en los inputs aparecerá la información de la falla actual, para poder editarla y para evitar que se mande un campo vacío y se elimine información. El único campo obligatorio para crear la falla es el Nombre, el resto se pueden completar luego al editarla

5. Acciones usuarios: He querido limitar las acciones de los usuarios con ROLE_USER por que mi intención era que fuera lo menos parecido a una red social, mi idea es que los usuarios solo tenga permiso de ‘lectura’ para poder informarse y ver todo lo que esta pasando y pendiente de pasar y sus acciones de limiten a:

   a. Contestar encuestas: Cuando el usuario visite el muro de la falla, encontrara una pestaña en la que se mostraran las encuestas actuales. Puede votar y el voto se reflejará en el momento. A futuro me gustaría limitar a un voto por persona, deshabilitando los checboxs una vez se haya votado y sustituyéndolos por unos chips con la información de como va la encuesta.

   b. Asistir a evento sin pago: Los eventos que no tengan pago podrán mostrarán un botón de asistir, al pulsar se sumara uno al contador y asi se puede ver en un solo vistazo cuantas personas están interesadas en este evento. A futuro me gustaría que el administrador pueda ver una lista con los nombres de los asistentes.

   c. Asistir y pagar evento: Los eventos con pago mostrarán, asistir y pagar, al pulsar, se desplegara una plataforma de pago para hacer efectivo el pago. Una vez completado se sumará a la lista de asistente y la de pagadores y el chip de no pagado rojo, se mostrará como pagado en verde. Actualmente por cuestiones de tiempo no he podido implementar la pasarela de pago, he investigado y la opción mas viable sería utilizar stripe, que es sencilla y gratuita y permite al administrador vincular su propia cuenta de stripe para recibir allí los pagos.

   d. Ver y descargar ‘llibrets’: Cuando visite el perfil de la falla, mostrara una pestaña con los llibrets disponibles. Si pulsa encima de uno, este se abrirá en una pestaña nueva del navegador dando la opción de descargar al usuario. A futuro me gustaría que preguntara si quiere recibirlo al email y mandarlo.

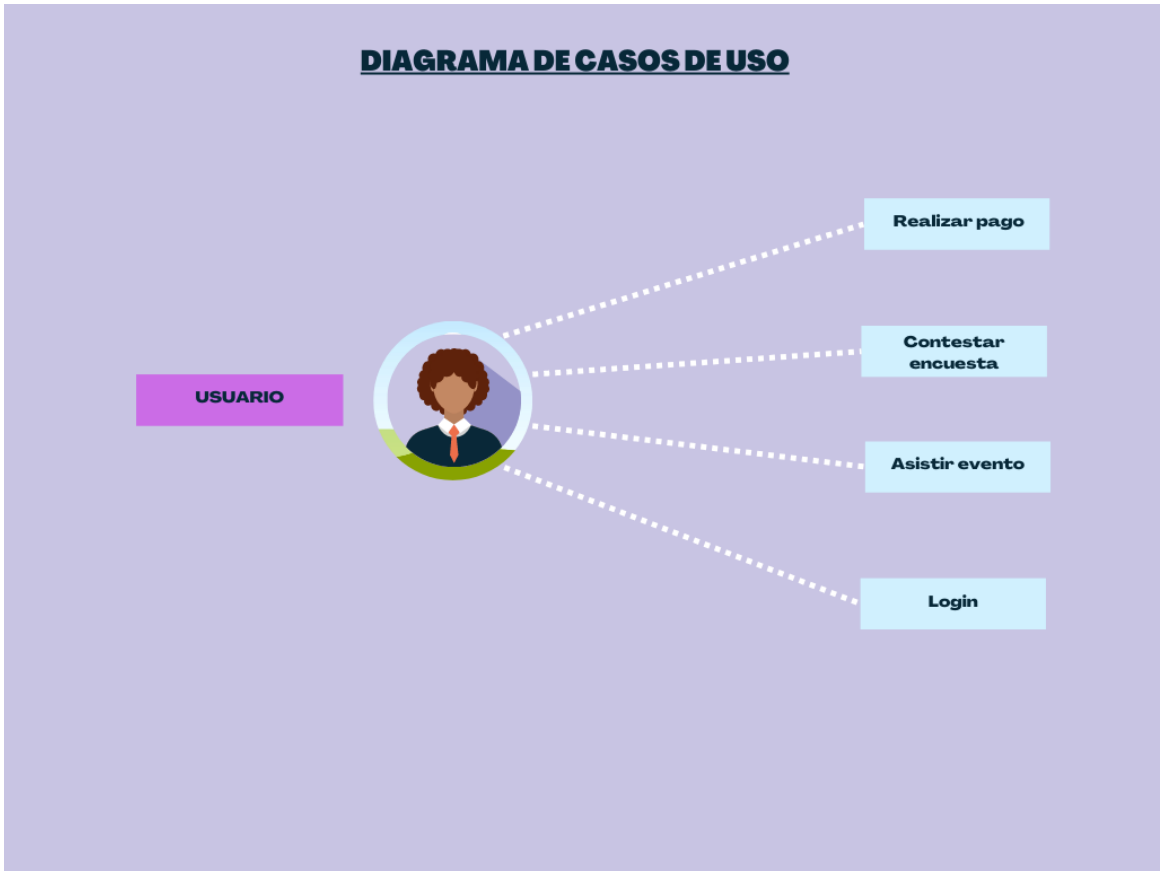


DIAGRAMA DE CASOS DE USO
USUARIO
Realizar pago
Contestar encuesta
Asistir evento
Login