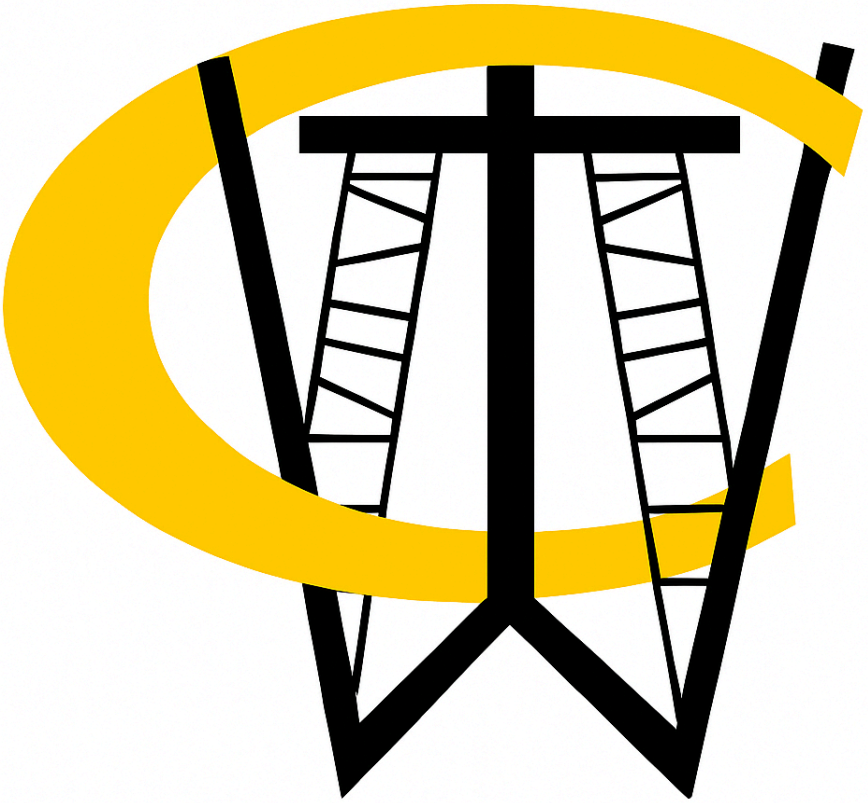
# MANUAL DE ADMINISTRADOR
# “PASSION KOFRADE”



JAVIER GUERRERO MONTERO - IES PORTADA ALTA

# ÍNDICE

# INTRODUCCIÓN

El mundo cofrade ha sido tradicionalmente un pilar cultural y religioso de gran relevancia, con una comunidad fiel y activa. En los últimos años, su presencia y visibilidad han crecido notablemente, impulsada en parte por la difusión en medios digitales y redes sociales. Si bien este aumento de notoriedad ha traído consigo numerosos beneficios para los implicados en este ámbito, también ha puesto de manifiesto ciertas carencias, especialmente en lo relativo a la localización y acceso a la información sobre eventos cofrades.

Actualmente, para conocer detalles sobre procesiones, conciertos, besamanos, cultos u otras actividades organizadas por las cofradías, es necesario realizar búsquedas exhaustivas a través de diversas redes sociales y canales informativos, sin garantía de encontrar la información deseada de forma clara y centralizada.

Con el objetivo de solucionar esta necesidad, nace **PASSION KOFRADE**, un proyecto cuyo propósito es unificar en una única plataforma todos los eventos cofrades cercanos, facilitando su consulta de manera rápida, accesible y actualizada. Esta herramienta está pensada para servir de referencia a toda la comunidad cofrade, permitiendo que los usuarios accedan a la información de forma sencilla desde cualquier lugar y en cualquier momento.

# OBJETIVOS DE LA APLICACIÓN

Tal y como se ha expuesto previamente, el objetivo principal de esta aplicación es proporcionar a los usuarios un acceso centralizado, cómodo y eficiente a la información relacionada con las distintas actividades y eventos organizados por las cofradías. Esta necesidad surge ante la dispersión de datos en múltiples canales de comunicación, lo cual dificulta, en muchas ocasiones, el seguimiento de la agenda cofrade por parte de los fieles, devotos y apasionados de este ámbito.

**PASSION KOFRADE** pretende solventar esta problemática actuando como un punto de encuentro digital, donde se pueda consultar de manera clara y ordenada la programación de procesiones, cultos, conciertos, besamanos, actos extraordinarios y demás eventos vinculados al mundo cofrade (además de poder añadirlos a favoritos para una mejor consulta). No obstante, el proyecto no se limita únicamente a la recopilación de actividades, sino que también incorpora funcionalidades complementarias que enriquecen la experiencia del usuario.

Entre estas funcionalidades destaca la posibilidad de consultar y gestionar artículos y publicaciones al estilo de un periódico digital, permitiendo así mantener informada a la comunidad sobre noticias relevantes, entrevistas, reportajes o comunicados oficiales. Asimismo, la aplicación integrará un módulo destinado a ofrecer información detallada sobre cada una de las cofradías de la ciudad de Málaga, incluyendo su historia, titularidades, sede canónica, días de salida y otros datos de interés, con el fin de fomentar el conocimiento y la divulgación del patrimonio cofrade malagueño.

En conjunto, **PASSION KOFRADE** se presenta como una solución tecnológica integral orientada a facilitar la difusión, consulta y preservación de la actividad cofrade, adaptándose a los nuevos hábitos de consumo de información y al contexto digital actual.

# PLANIFICACIÓN DEL PROYECTO

El desarrollo del proyecto se ha estructurado en varias fases bien definidas, siguiendo una metodología que combina análisis, diseño, implementación, pruebas y despliegue, con el objetivo de garantizar un producto funcional, escalable y alineado con las necesidades reales de los usuarios.

En primer lugar, se llevó a cabo un **análisis detallado de los requisitos funcionales y no funcionales**, centrado especialmente en las necesidades del usuario final. Entre las funcionalidades identificadas como prioritarias se encuentran la posibilidad de crear, editar y eliminar eventos asociados a cada cofradía, así como un sistema de registro y autenticación de usuarios que garantice una gestión segura y personalizada de la información.

Posteriormente, se diseñó la **arquitectura del sistema**, abordando tanto la estructura de la base de datos como los flujos de trabajo internos y la interfaz de usuario. En este punto, se priorizó la usabilidad, la claridad visual y la coherencia en la navegación, para garantizar una experiencia accesible y sencilla para todo tipo de usuarios.

Durante la fase de implementación, se utilizó **Angular** para el desarrollo del cliente (frontend), y **Laravel** como framework principal del servidor (backend), empleando **MySQL** como sistema gestor de bases de datos. Además, se integraron diferentes APIs externas para ampliar las funcionalidades del sistema, entre las que destacan **SendGrid** para el envío de correos electrónicos y **WeatherAPI** para obtener información meteorológica, una funcionalidad especialmente útil para la comunidad cofrade, dado que las condiciones climáticas influyen directamente en la planificación y celebración de muchos de sus eventos.

Una vez desarrollada la aplicación, se procedió a la **fase de pruebas funcionales**, utilizando datos reales para validar el comportamiento del sistema, corregir posibles errores y optimizar el rendimiento general.

Estas pruebas permitieron verificar la fiabilidad de las funcionalidades implementadas y garantizar una experiencia de uso satisfactoria.

Finalmente, el proyecto fue **desplegado en un entorno de producción**, concretamente en un servidor privado virtual (VPS) proporcionado por OVH, utilizando dos contenedores Docker para la separación del frontend y el backend. Esta arquitectura facilita la escalabilidad y el mantenimiento de la aplicación. Además, el proyecto contempla una fase posterior de **mantenimiento y soporte técnico**, donde se prevé la posibilidad de realizar futuras ampliaciones y actualizaciones según la evolución de las necesidades del sistema y de sus usuarios.

A continuación se hará una estimación de las horas que dura y el coste del proyecto. El proyecto se cobrará a **25€/h.**

| PARTE DEL PROYECTO | TIEMPO ESTIMADO (h) | COSTE (€) |
|---|---|---|
| Planteamiento | 10 | 375 |
| Desarrollo Base de Datos | 4 | 38 |
| Desarrollo del Backend + Conexión con la Base de Datos | 100 | 3750 |
| Desarrollo del FrontEnd + Uso de APIs | 80 | 3000 |
| Despliegue | 10 | 375 |
| Pruebas finales + arreglos | 46 | 1725 |
| **TOTAL:** | **250** | **9263** |

Para calcular el coste del proyecto, he multiplicado cada tiempo de duración de cada tarea por **1,5** y, después, por los **25€/h**.

# ANÁLISIS Y DISEÑO DEL SISTEMA

El desarrollo del sistema ha requerido una planificación detallada de la estructura de datos y de las relaciones entre los distintos elementos que forman parte de la lógica de la aplicación. El back-end se sustenta sobre una base de datos relacional implementada en **MySQL**, compuesta por cinco tablas principales (sin contar las tablas internas generadas automáticamente por el sistema de autenticación de Laravel, como personal_access_tokens o password_resets). Cabe destacar que todas ellas se instalan gracias a migraciones y seeders.

A continuación, se describen las tablas fundamentales del sistema:

- **ARTICULOS**: almacena los artículos publicados por los usuarios registrados. Contiene los siguientes campos:
  id, titular, id_autor, cuerpo.
  El campo id_autor establece una relación con la tabla USERS.
- **COFRADÍAS**: recoge la información principal de cada cofradía. Sus campos son:
  id, nombre, titular1, titular2, titular3, direccion, parroquia.
- **EVENTOS**: contiene los datos de los eventos relacionados con las cofradías. Incluye:
  id, nombre, cofradia, fecha.
  El campo cofradia establece una relación con el campo id de la tabla COFRADÍAS.
- **USERS**: tabla que gestiona los datos de los usuarios registrados en la aplicación. Sus campos son:
  id, nombre, email, password, codigo.
  **FAVORITOS**: recoge los eventos que los usuarios han marcado como favoritos. Incluye los campos:
  id, id_evento, id_usuario.
  El campo id_usuario se relaciona con el id de USERS, mientras que id_evento se relaciona con el id de EVENTOS.

```
mysql> select * from eventos;
+----+-----------------------------------------+----------+---------------------+---------------------+---------------------+
| id | nombre                                  | cofradia | fecha               | created_at          | updated_at          |
+----+-----------------------------------------+----------+---------------------+---------------------+---------------------+
| 11 | Triduo de Humildad y Paciencia          |        4 | 2025-04-15 19:00:00 | NULL                | NULL                |
| 15 | Procesión de Salutaciónasdad            |        3 | 2025-05-01 21:00:00 | NULL                | 2025-05-13 01:41:10 |
| 16 | Rosario de la Aurora de Humildad y Paciencia |        4 | 2025-05-03 07:00:00 | NULL                | NULL                |
| 17 | Salida Procesional de La Humildad       |        5 | 2025-05-05 20:00:00 | NULL                | NULL                |
| 20 | Restauración de la Virgenasd            |        1 | 2025-06-15 12:00:00 | NULL                | 2025-05-20 16:45:21 |
| 23 | javier                                  |        1 | 2025-05-03 18:20:00 | 2025-05-20 16:19:26 | 2025-05-20 16:19:26 |
+----+-----------------------------------------+----------+---------------------+---------------------+---------------------+
6 rows in set (0.00 sec)
```

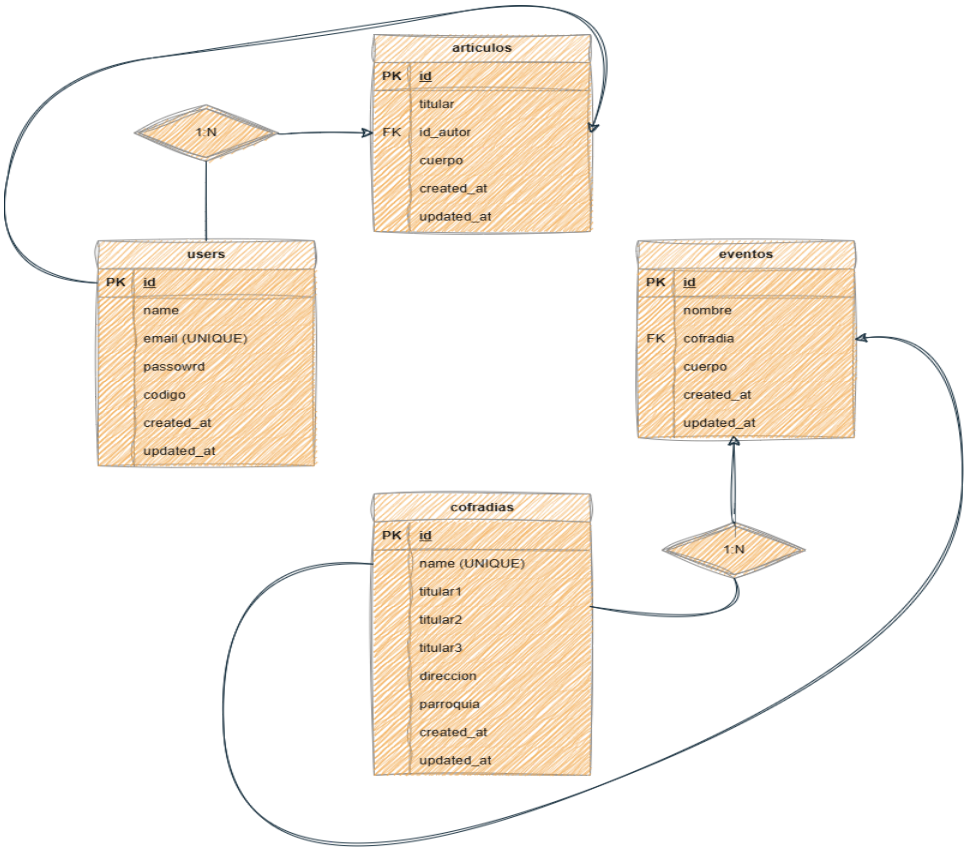
Vista de la tabla “eventos” a modo de ejemplo

**Relaciones entre tablas**

Las relaciones principales definidas en el sistema son las siguientes:

- ARTICULOS.id_autor → USERS.id: un usuario puede ser autor de múltiples artículos.
- FAVORITOS.id_usuario → USERS.id: un usuario puede tener múltiples eventos marcados como favoritos.
- FAVORITOS.id_evento → EVENTOS.id: un evento puede ser marcado como favorito por varios usuarios.
- EVENTOS.cofradia → COFRADÍAS.id: cada evento está vinculado a una cofradía específica.

Esta estructura relacional está diseñada para garantizar la integridad de los datos, facilitar las consultas y asegurar una escalabilidad eficiente del sistema conforme aumente el volumen de usuarios, artículos y eventos.



# ESPECIFICACIONES DEL SISTEMA

## Tecnologías y Software empleado

El desarrollo de este proyecto ha implicado la integración de un conjunto de tecnologías modernas, seleccionadas estratégicamente por su fiabilidad, escalabilidad y facilidad de uso tanto en entornos de desarrollo como de producción.

En el **back-end**, se ha utilizado **Laravel**, un framework PHP basado en el patrón de arquitectura **MVC (Modelo–Vista–Controlador)**. Laravel destaca por su estructura ordenada, su robusto sistema de rutas, su integración nativa con bases de datos relacionales y su amplio ecosistema de paquetes. Todo ello facilita una programación limpia, mantenible y orientada a buenas prácticas.

Para el **front-end**, se ha optado por **Angular**, un framework basado en TypeScript que permite el desarrollo de aplicaciones web dinámicas y altamente estructuradas. Angular ofrece una clara separación entre componentes, servicios y módulos, lo que favorece una organización eficiente del código y una experiencia visual atractiva tanto para desarrolladores como para usuarios finales. En cuanto a la maquetación y el diseño visual, se ha empleado **Bootstrap**, una librería de estilos ampliamente utilizada que facilita la creación de interfaces responsivas, con componentes predefinidos y una estructura coherente de diseño.

En relación a la **base de datos**, se ha utilizado **MySQL**, un sistema de gestión de bases de datos relacional basado en SQL, reconocido por su rendimiento, seguridad y compatibilidad con aplicaciones de alto tráfico.

Además de estas tecnologías principales, se han incorporado herramientas y servicios complementarios que enriquecen las funcionalidades de la aplicación:

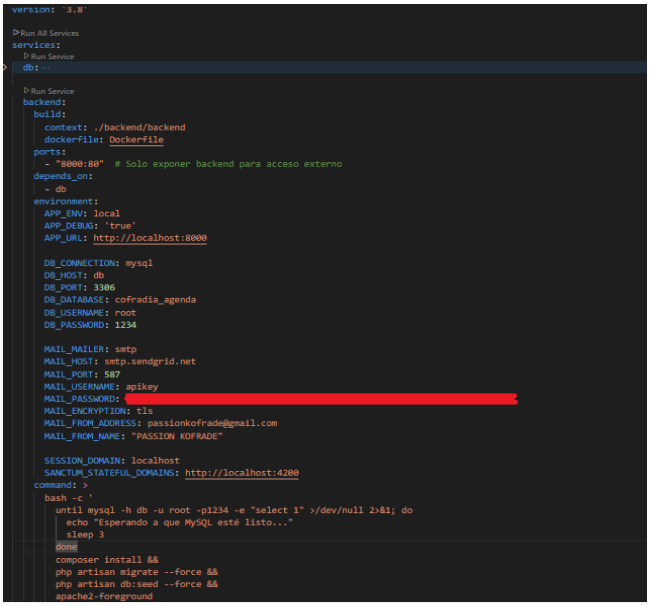
- **SendGrid** y **Mailer**: servicios externos utilizados para la gestión y envío de correos electrónicos automatizados, fundamentales en procesos como el registro, la recuperación de contraseña o las notificaciones.

- **cookie-service** (Angular): permite gestionar y recuperar información del usuario almacenada en cookies, mejorando la experiencia personalizada.

- **Karma** (Angular): herramienta de testing utilizada para realizar pruebas unitarias del front-end y validar el correcto funcionamiento de los componentes.

- **PHPUnit**: herramienta de testing utilizada en el back-end para garantizar la estabilidad y fiabilidad del código desarrollado en Laravel.

Para el **despliegue**, se ha utilizado **Docker** junto con **docker-compose** (véase en la siguiente imagen), permitiendo contenerizar la aplicación tanto en su parte cliente como servidor. Gracias a esta estrategia, el proyecto puede ser desplegado de forma sencilla y reproducible en cualquier servidor compatible, garantizando portabilidad, consistencia y aislamiento de los entornos de ejecución. Por último, añadir que se ha usado **nginx** para desplegar el Frontend en el contenedor.




Parte del backend del docker-compose.

```yaml
Run Service
frontend:
  build:
    context: ./frontend/frontend
    dockerfile: Dockerfile
  ports:
    - "4200:80"  # Aquí se sirve con Nginx o similar en Dockerfile
  depends_on:
    - backend
  networks:
    - app_network

volumes:
  dbdata:

networks:
  app_network:
    driver: bridge
```

Parte del frontend del docker-compose.

Dockerfile del backend.

```dockerfile
FROM php:8.3-apache    The image contains 4 high vulnerabilities

# Instalar extensiones necesarias + Node.js y npm
RUN apt-get update && apt-get install -y \
    git unzip libzip-dev zip libpng-dev libonig-dev libxml2-dev curl nodejs npm default-mysql-cl
    && docker-php-ext-install pdo pdo_mysql zip

# Activar mod_rewrite para Laravel
RUN a2enmod rewrite

# Establecer directorio de trabajo
WORKDIR /var/www/html

# Copiar el contenido del proyecto
COPY . /var/www/html

# Establecer permisos para Laravel
RUN chown -R www-data:www-data /var/www/html/storage /var/www/html/bootstrap/cache

# Configuración de Apache personalizada
COPY ./000-default.conf /etc/apache2/sites-available/000-default.conf

# Instalar Composer (usamos imagen oficial como fuente)
COPY --from=composer:2 /usr/bin/composer /usr/bin/composer

# Instalar dependencias y librerías necesarias de Laravel
RUN composer install \
 && composer require sendgrid/sendgrid \
 && composer require symfony/mailer \
 && composer require --dev phpunit/phpunit \
 && vendor/bin/phpunit --version
```

```dockerfile
# Etapa de build
FROM node:20-alpine AS build    The image contains 1 high vulnerab
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build -- --configuration production

# Etapa de producción con Nginx
FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/dist/frontend/browser /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile del frontend.

# Instalación y configuración de la aplicación

Las instrucciones para instalar **PASSION KOFRADE** son las siguientes:

1) Tener instalado y abierto Docker.
2) Contar con un acceso a una terminal con permisos para ejecutar comandos Docker (para cargar el archivo .yaml)
3) Configurar las variables de entorno (archivo .env) para adaptarse a cada servidor.
4) Asegurarse que en la carpeta frontend las urls están correctamente configuradas (en algunos casos, habrá que cambiarlas, todo depende de dónde se despliegue el backend).
5) Ejecutar docker-compose up --build -d, para cargar las imágenes, los contenedores y desplegar el proyecto en el servidor.
6) Acceder a la URL proporcionada por los contenedores.

**Notas adicionales**

- El servidor web **NGINX** se utiliza en el contenedor del frontend para servir la aplicación Angular de forma eficiente.
- El backend está basado en Apache, configurado para servir Laravel.
- La base de datos MySQL se ejecuta en un contenedor separado para garantizar aislamiento y facilidad de mantenimiento.

# APARTADOS DESARROLLADOS

A continuación, se analizará con profundidad cada componente de la web, así como las funcionalidades y operaciones que se pueden realizar en cada uno. Todos los componentes tienen diseño responsive y se centran en las vistas de escritorio y móvil.

## Barra de navegación

Este componente es el encargado de cambiar entre componentes de Angular mediante la herramienta del routing. Contiene el botón cuya función hace referencia al "logout" del backend, permitiendo que el usuario pueda cerrar su sesión revocando el token actual en el que está. Este botón recoge los datos del usuario y, si está logueado, aparece como disponible, haciendo el acto contrario si no lo está.


Foto de la barra de navegación en formato escritorio.


Foto de la barra de navegación en formato móvil (sin menú desplegado).


Foto de la barra de navegación en formato móvil (menú desplegado).


Botón cuando el usuario está logueado (hover) Botón cuando el usuario NO está logueado

Cabe puntuar que el puntero cambia entre click y un ”prohibido”

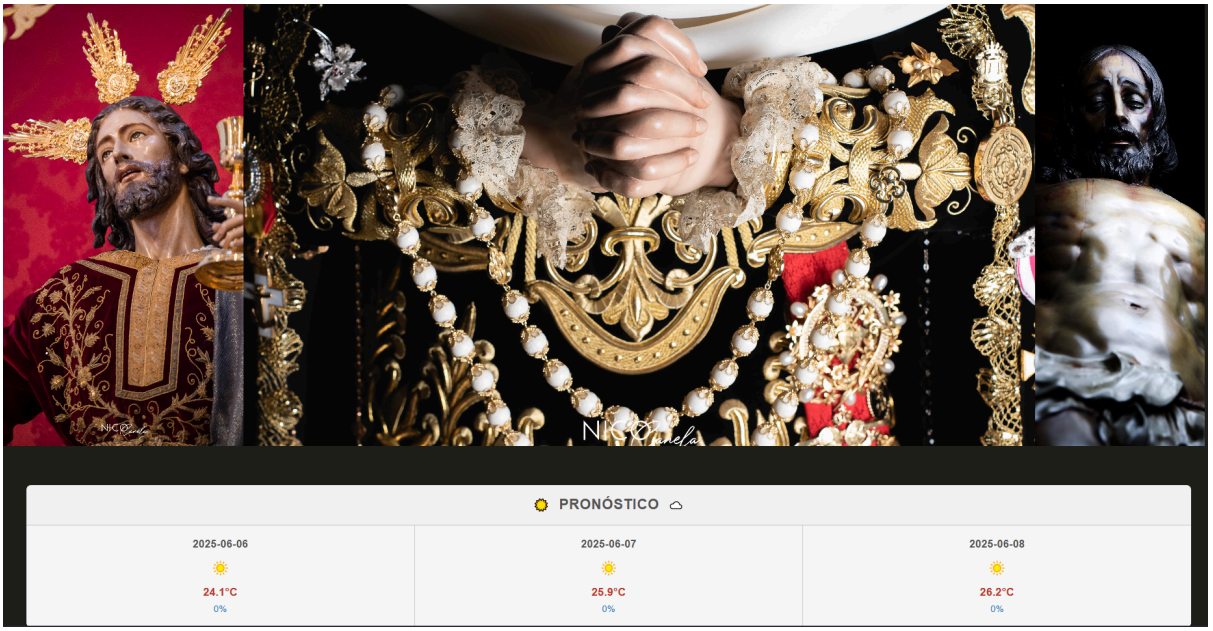
```php
public function logout(Request $request)
{
    if ($request->user()) {
        $request->user()->currentAccessToken()->delete(); // Revoca el token actual

        Log::info('Usuario deslogeado: ', [
            'status' => 200,
            'usuario' => $request->user()->name,
            'email' => $request->user()->email,
        ]);
        return response()->json(['message' => 'Sesión cerrada correctamente'], 200);
    } else {
        return response()->json(['error' => 'Usuario no autenticado'], 401);
    }
}
```

Lógica del botón de cerrar sesión.

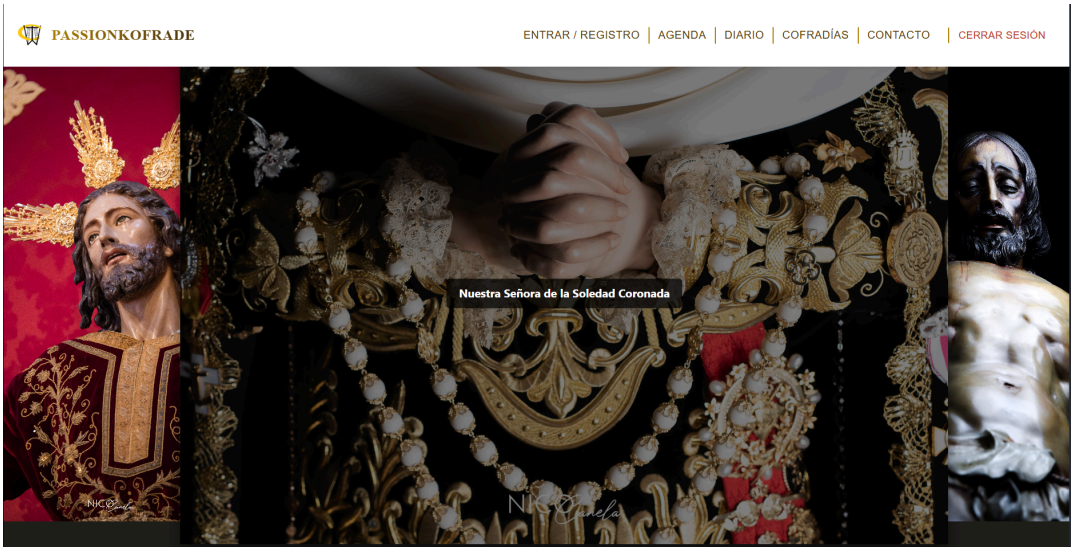
# Inicio

Componente que da la bienvenida a la web. Las tres imágenes se agrandan cuando se les pasa el ratón por encima, haciendo que a su vez aparezca el nombre del titular.

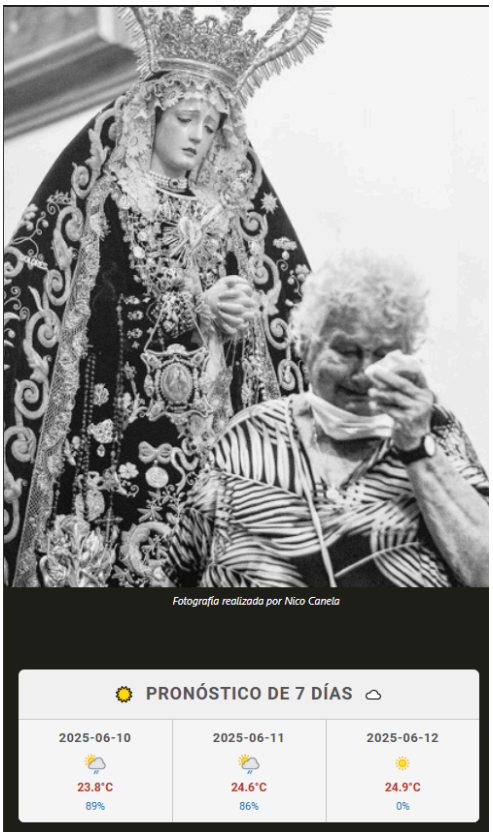
Los datos del pronóstico son obtenidos gracias a WeatherAPI.




Vista  del inicio sin la barra de navegación.




Vista del inicio cuando se pasa el ratón por encima de una imagen.

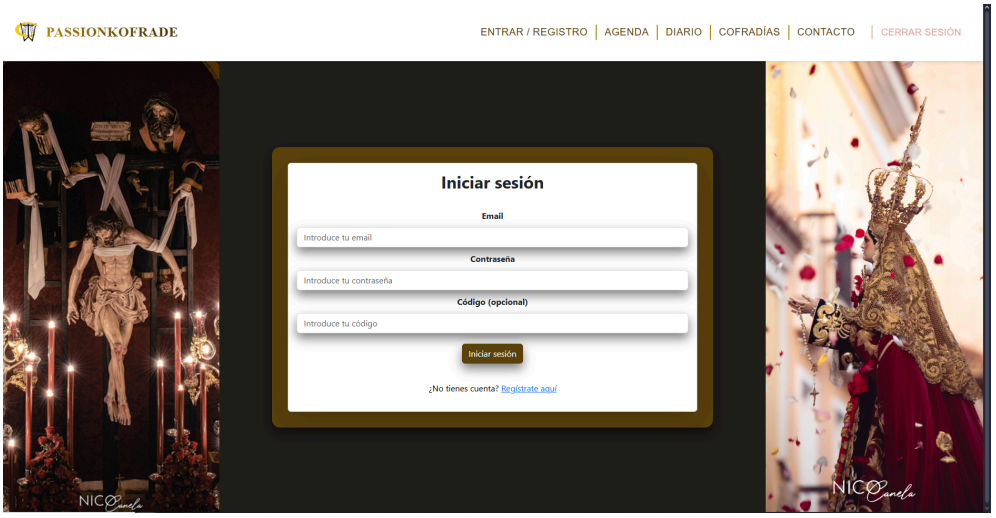



Visa del inicio para dispositivos móviles.

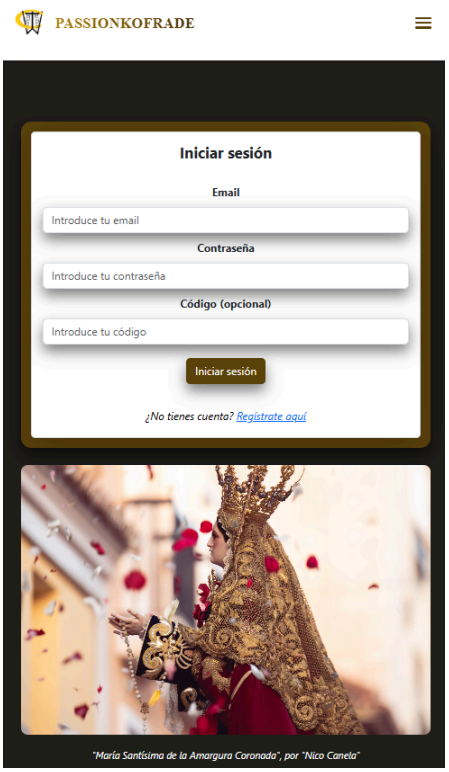
## Inicio de sesión

Componente compuesto de dos imágenes con la misma funcionalidad que las del inicio y de un formulario que llama al método “login” del backend. La única diferencia de los formularios de inicio de sesión corrientes de otras páginas es el “código”. Este código permite acceder como rol “cofradía”, lo que te da ciertos permisos en la web, como por ejemplo crear un evento en el componente “AGENDA”, o editar y eliminar los que te pertenezcan; o crear, editar y eliminar un artículo en el “DIARIO”. Dicho código será administrado por el propio creador de la página web. Si no se pone el código, se loguea con el rol “USUARIO”, que permitirá añadir eventos de la agenda en favoritos.

Esto será explicado con más detalle a continuación.




Vista del componente “INICIAR SESIÓN”.




Versión para móviles.

```php
public function login(Request $request)
{

    $user = Usuario::where('email', $request->email)->first();

    if (!$user || !Hash::check($request->password, $user->password)) {
        return response()->json(['error' => 'Credenciales inválidas'], 401);
    }

    if (!empty($request->codigo) && $request->codigo !== $user->codigo) {
        return response()->json(['error' => 'Código incorrecto'], 403);
    }
    if (!empty($request->codigo) && $request->codigo === $user->codigo) {
        $rol = 'cofradia'; // Si el código ingresado coincide con el del usuario, es una cofradía
    } else {
        $rol = 'usuario'; // Si no se ha ingresiado código o es incorrecto, es un usuario
    }
    $token = $user->createToken('authToken')->plainTextToken;
```

Primera parte de la lógica del método “login” del backend.

Como se puede ver en la imagen, se obtiene el usuario cuyo email coincida con el email enviado en el formulario y, después de una serie de validaciones y obtener el rol, se crea un token con la información del usuario, que será usado más adelante para otras funciones.

```php
    Log::info('Usuario logeado: ', [
        'status' => 200,
        'usuario' => $user->name,
        'email' => $user->email,
        'rol' => $rol,
    ]);


    return response()->json([
        'data' => [
            'accessToken' => $token,
            'toke_type' => 'Bearer',
            'user' => [
                'id' => $user->id,
                'name' => $user->name,
                'email' => $user->email,
                'codigo' => $user->codigo,
                'role' => $rol,
                'created_at' => $user->created_at,
                'updated_at' => $user->updated_at
            ]
        ]
    ]);
}
```

Segunda parte de la lógica del método “login” del backend.

Como se puede observar, se envía un Log de información con los datos del usuario (para el control de errores y de logs) y, a continuación, la información es transmitida al frontend para confirmar que el usuario ha

sido logueado con éxito y, por supuesto, para usar la información en futuras operaciones.

```
this.authService.login(formData).subscribe(
  (response) => {
    console.log('Login exitoso:', response);

    // Guardamos el token y los datos del usuario
    localStorage.setItem('token', response.data.accessToken);
    localStorage.setItem('user', JSON.stringify(response.data.user));


    alert(`Logeado como ${response.data.user.name}`);
     this.router.navigate(['/']).then(() => {
      location.reload(); // Esto evita bugs en Docker
    });
  },
  (error) => {
    console.error('Error al iniciar sesión:', error);
    this.error = 'Credenciales incorrectas';
  }
);
```

Código del frontend donde se usa esta información.

En la imagen adjuntada obtenemos la respuesta y la almacenamos en un token para poder ser usada en la agenda o en el diario.

Por último, se envía al usuario a la página inicio.

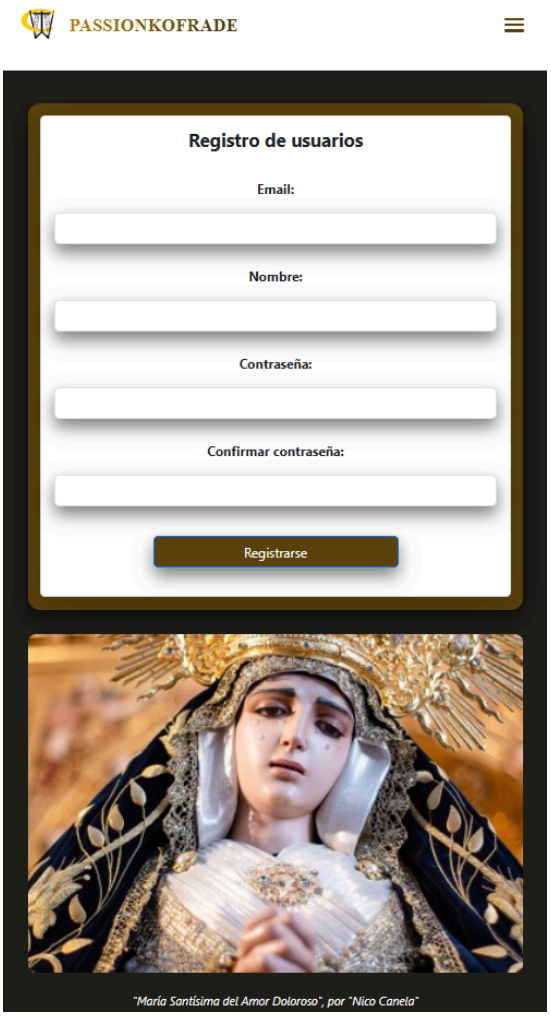
# Registro de usuarios

Se ha implementado un componente similar al de “Inicio de Sesión”, cuya función es permitir la creación de nuevos usuarios en el sistema. A través de un formulario, el usuario introduce sus datos, que son almacenados en la base de datos. La contraseña es encriptada antes de guardarse, garantizando así un mayor nivel de seguridad.

Una vez registrados, estos usuarios pueden iniciar sesión y acceder a las funcionalidades correspondientes según su rol en la plataforma, ya sea como usuario general o como representante de una cofradía.

Además, se ha añadido una función que envía un correo electrónico al administrador cada vez que se crea un nuevo usuario. Este correo incluye el nombre, el email y la contraseña del usuario (en fases de prueba), permitiendo llevar un control de los registros.


Vista del componente “REGISTRO”.




Vista para móviles.

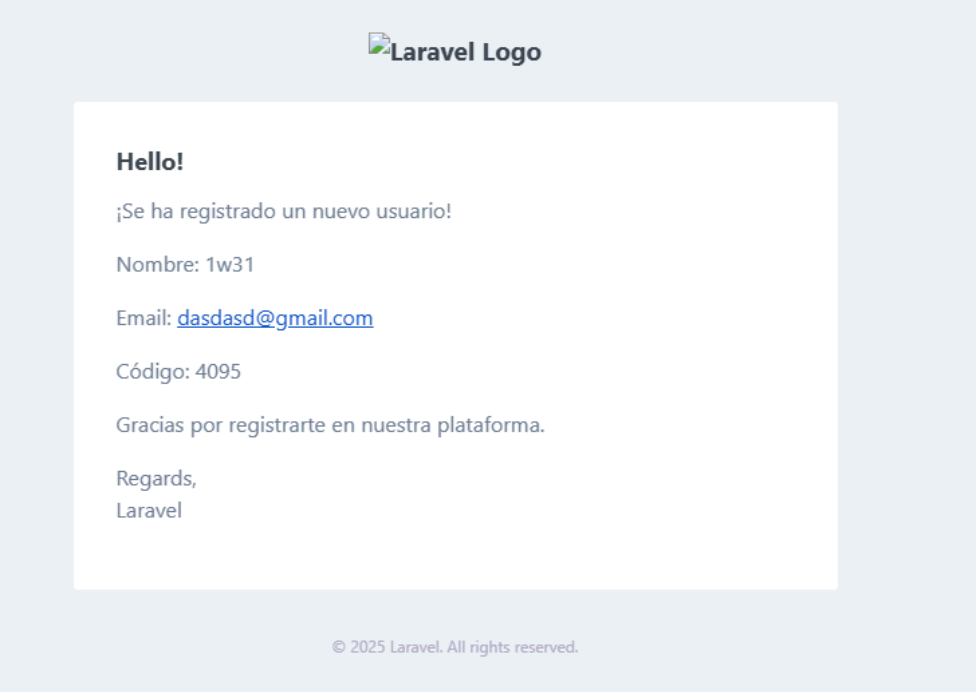
```
this.authService.register(formData).subscribe(
  (response) => {
    console.log('Registro exitoso:', response);
    alert(`Registrado como ${this.name}`);

    this.router.navigate(['/login']);
  },
  (error) => {

    alert(
      'Ocurrió un error al registrarse: ' + error.error.message ||
        'Error desconocido'
    );

  }
);
}
}
```

Código del componente en el frontend.

```php
public function register(Request $request)
{
    $request->validate([
        'email' => 'required|email|unique:users,email',
        'password' => 'required|min:6'
    ]);

    // Crear un nuevo usuario
    $usuario = new Usuario();
    $usuario->email = $request->email;
    $usuario->name = $request->name;
    $usuario->password = Hash::make($request->password);

    // Generar un código aleatorio único
    do {
        $codigoAleatorio = rand(1111, 9999);
    } while (Usuario::where('codigo', $codigoAleatorio)->exists());

    $usuario->codigo = $codigoAleatorio;
    $usuario->save();

    Log::info('Usuario registrado: ', [
        'status' => 200,
        'usuario' => $usuario->name,
        'email' => $usuario->email,
        'rol' => $usuario->rol,
    ]);

    // Enviar la notificación a tu correo personal --- error aqui
    try {
        Notification::route('mail', 'javierguerreromontero1@gmail.com')
            ->notify(new RegisterNotification($usuario));
    } catch (\Exception $e) {
        Log::error('Error al enviar correo de registro: ' . $e->getMessage());
    }
}
```

Código del método register en el backend.

Este método resulta especialmente interesante, ya que no solo se encarga de crear un nuevo usuario en la base de datos, aplicando las validaciones necesarias para garantizar la integridad de los datos, sino que además genera un código aleatorio único y envía un mensaje automático al correo del administrador de la web, facilitando así el seguimiento y control de los nuevos registros de manera segura y eficiente:




Vista del email enviado por laravel gracias al método de la Notification.

Este código es el que permite a los usuarios loguearse como "Cofradía". Será proporcionado por el administrador de la web una vez se haya comprobado que es una Cofradía real y no un usuario con ganas de desmantelar PASSION KOFRADE.

```php
    // Configura el mensaje de correo
    public function toMail($notifiable)
    {
        return (new MailMessage)
                    ->subject('Nuevo registro de usuario')
                    ->line('¡Se ha registrado un nuevo usuario!')
                    ->line('Nombre: ' . $this->usuario->name)
                    ->line('Email: ' . $this->usuario->email)
                    ->line('Código: ' . $this->usuario->codigo)
                    ->line('Gracias por registrarte en nuestra plataforma.');
    }
}
```

Código de la clase Notification, la encargada de enviar los emails.

# Agenda

Este módulo constituye el eje central de la plataforma, ya que a partir de él surgió la idea inicial del proyecto. Su función es recopilar todos los eventos almacenados en la base de datos y mostrarlos al público, proporcionando información relevante sobre los próximos cultos, procesiones y conciertos organizados por las distintas cofradías de Málaga.

La consulta de estos eventos está disponible tanto para usuarios registrados como para visitantes no registrados, facilitando así el acceso abierto a la agenda cofrade malagueña. Es importante destacar que la creación, edición y eliminación de estos eventos estará exclusivamente en manos de las propias hermandades, funcionalidad que se detallará más adelante.

Este componente se ha desarrollado en tres versiones distintas. A continuación, se presenta la versión destinada a los usuarios no registrados:


Componente “Agenda” sin iniciar sesión.

Los usuarios no registrados podrán consultar información básica de cada evento, incluyendo la fecha y la cofradía organizadora. Junto a cada uno de ellos se presenta un botón que invita al visitante a iniciar sesión en la plataforma. Al acceder con una cuenta, se habilita la posibilidad de añadir eventos a la tabla de **Favoritos**, permitiendo así guardar y gestionar aquellos que sean de su interés. Cabe destacar que el contenedor principal donde se guardan los eventos es totalmente scrolleable, así como el de los favoritos.

```php
    */
    public function index()
    {
        $eventos = Evento::orderBy('fecha', 'asc')->get();
        $cofradias = Cofradia::all();
        $favoritos = Favorito::all();
        $usuario = Auth::user();

        Log::info('Listado de eventos consultado', [
            'status' => 200,
            'count' => $eventos->count(),
        ]);

        return response()->json(['status' => 200, 'eventos' => $eventos, 'favoritos' => $favoritos, 'cofradias' => $cofradias, 'usuario' => $usuario]);
    }
```

Código para obtener los eventos en el backend.

En este código obtenemos los eventos ordenados por fecha, las cofradías y los favoritos.

```typescript
//problema --------------
calculaCofradiaNombre(): void {
  if (this.cofradias && this.evento) {
    console.log('Cofradías:', this.cofradias);
    console.log('Evento:', this.evento);
    const cofradia = this.cofradias.find(
      (c) => c.id === this.evento.cofradia
    );
    this.cofradiaNombre = cofradia ? cofradia.nombre : 'Desconocida';
  }
} //problema --------------
```

Desde el lado del **frontend**, cabe destacar un método específico de interés. Dado que en la tabla de eventos solo se almacena el **ID de la cofradía**, es necesario realizar un proceso adicional para mostrar el **nombre completo de la cofradía** en lugar de su identificador numérico.

Para ello, se implementa una lógica que, a partir del ID, consulta o accede a los datos de la cofradía correspondiente, permitiendo así que en pantalla se muestre información más legible y útil para el usuario. Este enfoque mejora notablemente la experiencia de navegación y comprensión de los eventos por parte del público.

```
ngOnInit(): void {
  this.calculaCofradiaNombre(); // Calcula el nombre de la cofradía
  const usuario = this.authService.getUsuarioData(); // Obtiene el usuario desde localStorage
  if (usuario) {
    this.role = usuario.role;
    this.nombreUsuario = usuario.name;
  } else {
    // Si no hay usuario, asigna valores por defecto para que los botones no se muestren
    this.role = ''; // o cualquier valor que tu lógica interprete como "sin usuario"
    this.nombreUsuario = '';
  }
}

ngOnChanges(changes: SimpleChanges): void {
  if (changes['cofradias'] || changes['evento'] || changes['usuario']) {
    this.calculaCofradiaNombre();
  }
}
```

Código principal del frontend.

También es relevante destacar el **método principal** del componente, encargado de la **carga inicial de los datos**. En este proceso se recupera la información necesaria para el correcto funcionamiento del módulo, incluyendo la obtención del **nombre de usuario** y la **identificación del rol asignado** (visitante, usuario registrado o representante de una cofradía). Esta distinción resulta fundamental para determinar qué funcionalidades y vistas estarán disponibles según el tipo de usuario, asegurando así una experiencia personalizada y acorde a los permisos establecidos.

```
@if (role == 'cofradia' && nombreUsuario == cofradiaNombre) {
<div class="acciones">
  <button class="btn btn-danger btn-sm" (click)="eliminarEvento(evento.id)">
    <i class="bi bi-trash-fill"></i> Eliminar
  </button>
  <button class="btn btn-primary btn-sm" (click)="editarEvento(evento.id)">
    <i class="bi bi-pencil-fill"></i> Editar
  </button>
</div>
} @else if (role == 'usuario') {
<div class="favorito-container">
  <button (click)="anadirFavorito(evento.id)" class="btn btn-favorito">
    <i class="bi bi-star-fill"></i> Añadir a Favoritos
  </button>
</div>
}
</div>
```

Como se puede observar, el **rol del usuario** cumple una función clave en la interfaz, ya que determina dinámicamente la **visualización de ciertos botones y funcionalidades**. Dependiendo del rol asignado, el sistema mostrará u ocultará acciones específicas, adaptando la experiencia de usuario a sus permisos. Estos elementos serán detallados más adelante en este documento. En el caso de ser una cofradía, se compara el nombre de usuario con el nombre de la cofradía de la base de datos (pues deben ser iguales, ya que profesionalmente, suponemos que una cofradía se registra con su nombre correcto).


Componente de la agenda al iniciar sesión como “usuario”.

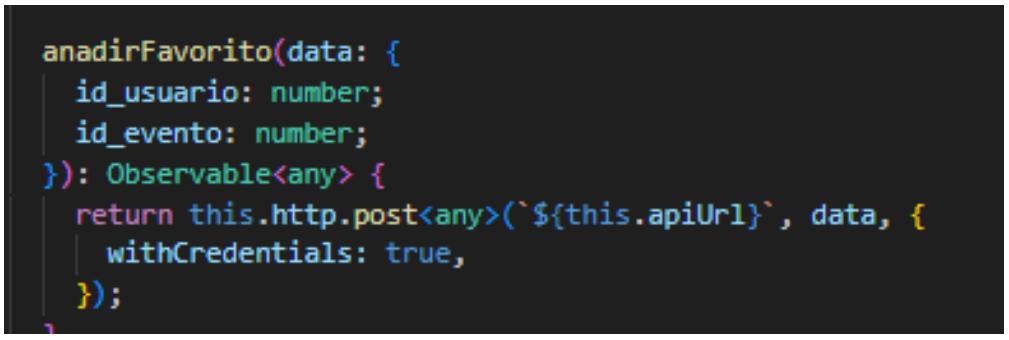

Versión para móviles.

Esta versión del componente se activa cuando el usuario inicia sesión de forma convencional, es decir, sin introducir ningún código adicional que lo identifique como representante de una cofradía. En este caso, el sistema reconoce al usuario con un **rol estándar**, lo que le permite **añadir eventos a su lista de favoritos**. Esta acción se registra en la base de datos mediante la asociación de la **ID del usuario** con la **ID del evento** correspondiente. Asimismo, cada evento marcado como favorito incluye un botón **“Eliminar”**, que permite al usuario **retirarlo fácilmente de su lista personal**.

```php
public function store(Request $request)
{
    $request->validate([
        'id_usuario' => 'required|integer|exists:users,id', // Verifica que el usuario exista
        'id_evento' => 'required|integer|exists:eventos,id', // Verifica que el evento exista
    ]);

// Comprobamos si ya existe ese favorito
$favoritoExistente = Favorito::where('id_usuario', $request->id_usuario)
                             ->where('id_evento', $request->id_evento)
                             ->first();

if ($favoritoExistente) {
    return response()->json(['status' => 409, 'message' => 'Este evento ya está en tus favoritos'], 409);
}

    Favorito::create($request->all());

    Log::info('Favorito añadido', [
        'status' => 200,
        'id_usuario' => $request->id_usuario,
        'id_evento' => $request->id_evento,
    ]);

        return response()->json(['status'=>200, 'message' => 'Favorito añadido correctamente'], 200);
    }
```

Código del backend para añadir un favorito.

En esta imagen se puede observar el **método utilizado para añadir eventos a la lista de favoritos**. La lógica implementada permite registrar la relación entre el usuario y el evento seleccionado de forma eficiente. Por otro lado, el **método para eliminar eventos** sigue una estructura habitual: simplemente se recibe la **ID del evento como parámetro**, y se procede a **eliminar el registro correspondiente de la tabla de favoritos**.

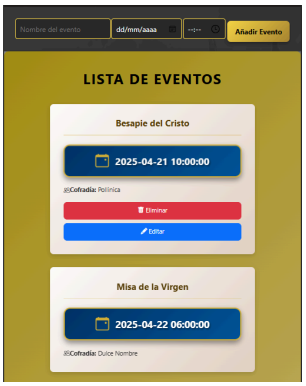


```typescript
anadirFavorito(data: {
  id_usuario: number;
  id_evento: number;
}): Observable<any> {
  return this.http.post<any>(`${this.apiUrl}`, data, {
    withCredentials: true,
  });
}
```

Método del servicio de Favoritos en el frontend, que añade un favorito a la tabla.


Vista del componente “Agenda” cuando el usuario inicia sesión como “Cofradía”.

Cuando el usuario inicia sesión utilizando el **código proporcionado por el administrador**, se le asigna automáticamente el **rol de “Cofradía”**. Este rol le otorga permisos específicos dentro de la aplicación, como la **creación, edición y eliminación de sus propios eventos**, sin afectar los eventos pertenecientes a otras cofradías, lo cual garantiza la integridad y seguridad de los datos. Esta funcionalidad se implementa mediante una serie de **condicionales lógicas**, que se detallarán en el siguiente apartado. Como se ha puntuado antes, si el nombre de usuario coincide con la cofradía, aparecerán los botones de editar y eliminar eventos, haciendo posible una gestión más segura de los datos.




Vista para móviles.

```php
public function update(Request $request, $id)
{
    //encontramos el evento por su id
    $evento = Evento::find($id);

    if (!$evento) {
        return response()->json(['message' => 'Evento no encontrado'], 404);
    }

    // Validar los datos recibidos
    $request->validate([
        'nombre' => 'required|string|max:255',
        'cofradia' => 'required|integer|exists:cofradias,id',
        'fecha' => 'required|date',
    ]);

    // Actualizar los datos del evento
    $evento->update([
        'nombre' => $request->nombre,
        'cofradia' => $request->cofradia,
        'fecha' => $request->fecha,
    ]);

    Log::info('Evento actualizado', [
        'status' => 200,
        'evento_id' => $evento->id,
        'nombre' => $evento->nombre,
        'cofradia' => $evento->cofradia,
        'fecha' => $evento->fecha,
    ]);

    return response()->json(['message' => 'Evento actualizado correctamente', 'evento' => $evento], 200);
}
```

Método del backend para editar un evento.

Este evento es editado en un nuevo componente, llamado editar-evento, compuesto de un pequeño formulario con el nombre del evento y la fecha, guardándose el id de la cofradía desde la agenda. El método de eliminar es como cualquier otro, por lo cual no es necesaria su muestra.

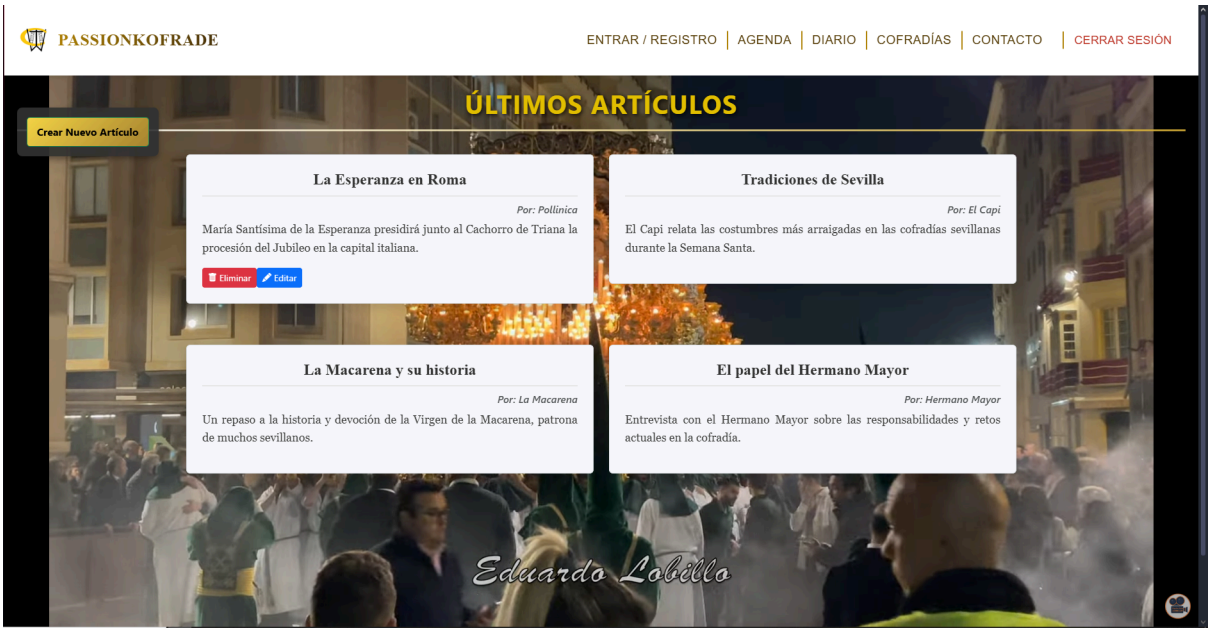
```php
public function store(Request $request)
{

    // Validar los datos enviados desde el formulario
    $request->validate([
        'nombre' => 'required|string|max:255',
        'cofradia' => 'required|integer|exists:cofradias,id',
        'fecha' => 'required|date',
        'hora' => 'required|date_format:H:i', // Asegura que la hora e  sté en formato HH:MM
    ]);

    $fechaCompleta = $request->fecha . ' ' . $request->hora;

    // Crear el evento en la base de datos
    $evento = Evento::create([
        'nombre' => $request->nombre,
        'cofradia' => $request->cofradia,
        'fecha' => $fechaCompleta, // Guardamos la fecha con la hora
    ]);

    Log::info('Evento creado', [
        'status' => 201,
        'evento_id' => $evento->id,
        'nombre' => $evento->nombre,
        'cofradia' => $evento->cofradia,
        'fecha' => $evento->fecha,
    ]);

    // Retornar respuesta JSON con código 201
    return response()->json([
        'message' => 'Evento creado con éxito',
        'evento'  => $evento
    ], 201);
}
```

Método del backend para crear un evento.

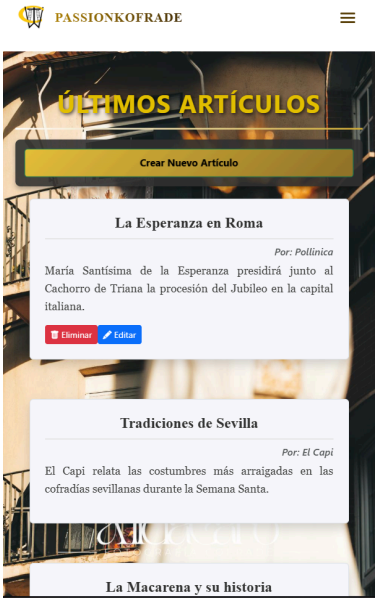

Vista del componente para editar un evento.

# Diario

Pensando en futuras ampliaciones, este componente ha sido diseñado con el objetivo de permitir que los propios usuarios puedan **compartir artículos y opiniones**, funcionando de forma similar a una red social. Para **publicar, editar o eliminar artículos**, es necesario haber iniciado sesión, independientemente de si se accede como **Usuario** o como **Cofradía**, ya que la lógica de permisos aplicada es similar a la del componente de agenda, aunque con una **estructura condicional más sencilla**.




Vista del componente “Diario” al haber iniciado sesión.



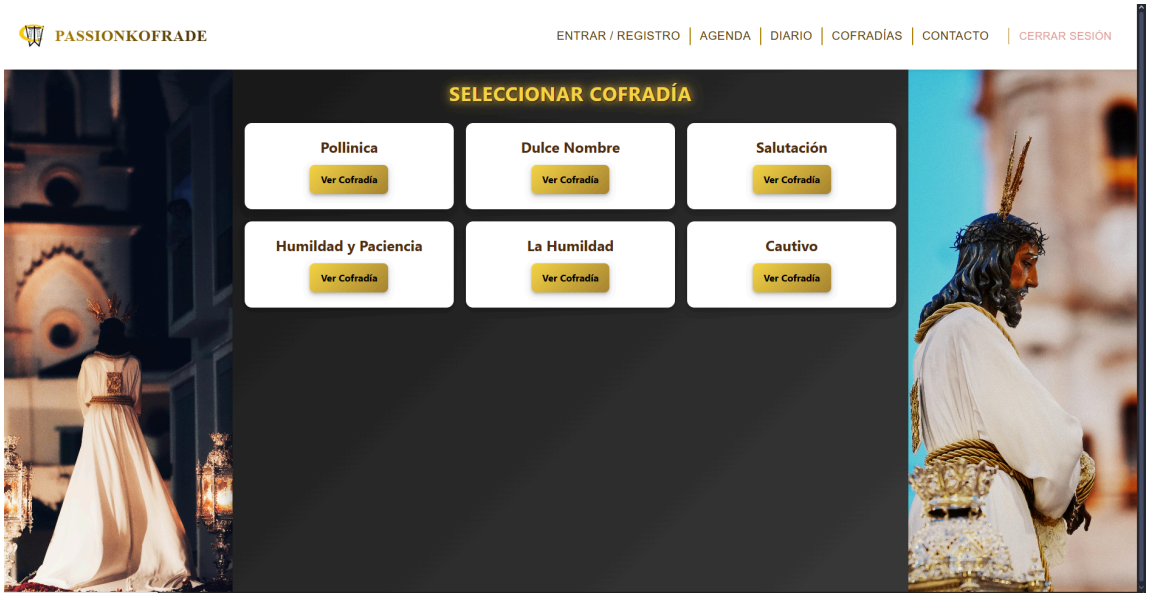

Versión para móvil.

Una característica destacable de este componente es la incorporación de un **vídeo de YouTube como fondo**, en este caso mostrando una escena de la Esperanza en la calle Molina Lario, implementado mediante el uso de un **elemento `<iframe>` en HTML/CSS**. Al cambiar a la versión móvil, el fondo se reemplaza por una imagen estática del Cautivo, optimizando así la experiencia visual en dispositivos con menor capacidad de procesamiento.

Los artículos presentan un comportamiento completamente **scrollable**, permitiendo la navegación sin afectar la reproducción o visibilidad del vídeo de fondo (el botón para crear un artículo es estático y no aparece si no se ha iniciado sesión). Su funcionamiento es análogo al de los eventos de la agenda, con la diferencia de que, en lugar de validar la cofradía, se verifica la **ID del usuario**. Por esta razón, no es necesaria una explicación detallada adicional ni la inclusión de fragmentos de código.

# Selección de cofradías

Este componente es el “puente” para la parte del frontend que muestra los datos de cada cofradía de la base de datos. Sin más misterio, se obtiene el nombre de cada elemento de la tabla cofradías.




Componente de selección de cofradías.


Versión para móvil.

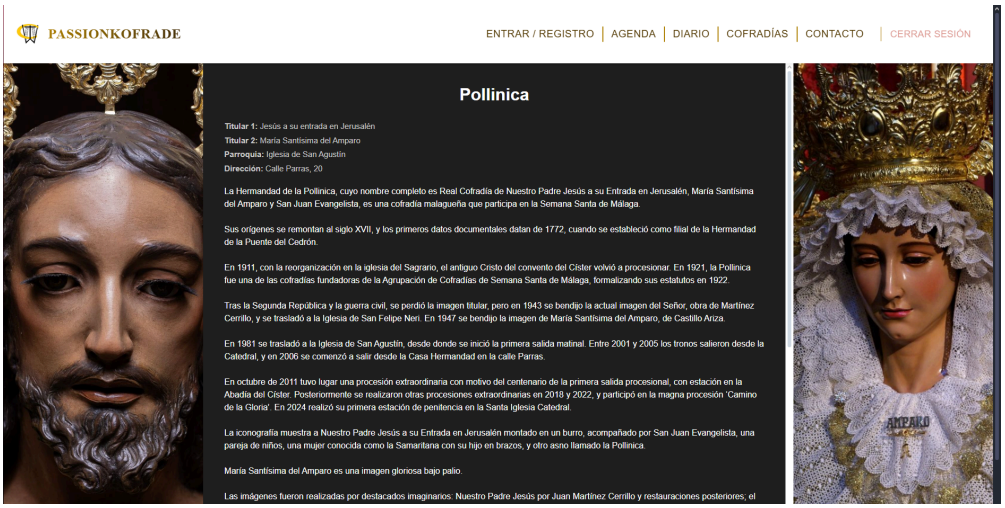
```
@for (cofradia of cofradias; track cofradia.id) {
  <div class="card-cofradia">
    <div class="card-body">
      <h3 class="nombre-cofradia">{{ cofradia.nombre }}</h3>
      <a class="btn-ver-cofradia" [routerLink]="'/cofradia/' + formatCofradiaName(cofradia.nombre)">
        Ver Cofradía
      </a>
    </div>
  </div>
}
```

Código del frontend que envía al usuario a una URL con el nombre de la cofradía seleccionada.

El código para obtener el nombre de la cofradía es similar a los códigos mostrados anteriormente, y el “formatCofradiaName”, tiene como función eliminar las mayúsculas y los espacios para una mejor navegación.
Al presionar el botón para ver la cofradía, el usuario será enviado al próximo componente.

# Información de la cofradía

Este componente ha sido desarrollado con el **único propósito de mostrar información relevante de cada cofradía**. Su función es exclusivamente informativa, por lo que **no incorpora métodos de mantenimiento ni funcionalidades interactivas**. Todo el contenido textual que se presenta es **procesado previamente desde el backend** y enviado al frontend para su visualización.



Vista del componente que muestra la información de una cofradía específica.

Versión móvil.

Este componente se encarga de mostrar la **información detallada de cada cofradía**, obtenida directamente desde la base de datos, incluyendo sus **titulares, parroquia y dirección**. Además, accede a un archivo de texto almacenado en una carpeta del backend denominada /storage, donde cada archivo lleva como nombre el de la cofradía correspondiente. Este contenido es **procesado en el servidor** y enviado al frontend, donde se muestra de forma clara al usuario (véase en la siguiente imagen).

```
  getRutaCristo(): string {
  if (!this.cofradia?.cofradia?.nombre) return '';
  return 'public/cofradiasDatos/' + this.cofradia.cofradia.nombre.toUpperCase().replace(/ /g, '-') + '/cristo.jpg';
}
  getRutaVirgen(): string {
    if (!this.cofradia?.cofradia?.nombre) return '';
    return 'public/cofradiasDatos/' + this.cofradia.cofradia.nombre.toUpperCase().replace(/ /g, '-') + '/virgen.jpg';
  }
}
```

Método para obtener las fotos.

Durante el desarrollo se identificó un problema inicial relacionado con los **espacios en las URLs**, ya que los navegadores no los interpretan correctamente. Para solventarlo, se optó por reemplazar los espacios por guiones medios (-) al construir las rutas hacia las carpetas o recursos. Por ejemplo, en el caso de **“Humildad y Paciencia”**, la URL se transforma en humildad-y-paciencia. En cambio, para cofradías cuyos nombres no contienen espacios, como **“Cautivo”**, no es necesario aplicar ninguna modificación.

```php
public function mostrar($nombre)
{
    // Obtener datos de la base de datos
    $cofradia = Cofradia::where('nombre', $nombre)->first();
    if (!$cofradia) {
        return response()->json(['message' => 'Cofradía no encontrada'], 404);
    }

    // Ajustar nombre carpeta con guiones
    $nombreCarpeta = strtoupper(str_replace(' ', '-', $nombre));
    $carpeta = public_path("storage/cofradiasDatos/" . $nombreCarpeta);

    $texto = file_exists("$carpeta/info.txt") ? file_get_contents("$carpeta/info.txt") : 'Información no disponible.';

    $imagenes = array_values(array_filter(scandir($carpeta), function ($archivo) {
        return preg_match('/\\.(jpg|jpeg|png|gif)$/i', $archivo);
    }));

    Log::info('Cofradía consultada', [
        'status'=>200,
        'nombre' => $cofradia->nombre,
    ]);

    return response()->json([
        'cofradia' => $cofradia,
        'texto' => $texto,
        'imagenes' => $imagenes
    ]);
}
```

Método del backend para devolver los datos y procesar el texto.

El tratamiento del texto en el backend sigue una lógica similar a la gestión de las imágenes en el frontend: es necesario **procesar la URL correctamente** para acceder al recurso correspondiente. El contenido se encuentra en un archivo .txt ubicado en la carpeta cuyo nombre coincide con el de la cofradía. Actualmente, estos textos provienen de **fuentes como Wikipedia** y se utilizan de modo provisional. Está previsto que, en el futuro, este contenido sea **sustituido por información oficial proporcionada directamente por las hermandades malagueñas**, garantizando así mayor precisión y autenticidad.

# CONCLUSIONES

Este proyecto nació con un propósito inicialmente didáctico, ya que desde el comienzo del curso tenía definida la temática de mi Trabajo Fin de Grado. Sin embargo, a medida que se fue desarrollando, adquirió un enfoque más práctico al evidenciarse como una posible solución a una de las principales carencias del ámbito cofrade: la falta de un sistema centralizado y accesible para la difusión de eventos y actos organizados por las distintas hermandades.

Gracias a esta aplicación, que incluye tanto un diario de eventos como un componente informativo sobre cada cofradía, los usuarios pueden consultar fácilmente la fecha y hora de los actos que desean seguir. Además, el proyecto ya cuenta con el respaldo de varias cofradías, lo cual abre la puerta a futuras mejoras, como la incorporación del lugar del evento, la eliminación automática de eventos pasados o la inclusión de un sistema de filtrado por hermandad.

Este trabajo me ha permitido adentrarme en nuevas áreas del desarrollo web, como la integración de APIs externas (WeatherAPI, SendGrid) y la creación y gestión de bases de datos, enriqueciendo así mi formación técnica. La experiencia ha sido especialmente motivadora al estar relacionada con un ámbito que me apasiona profundamente.

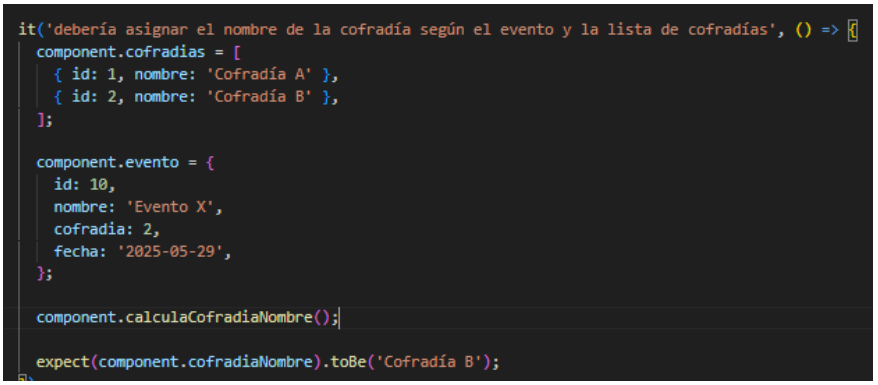
# APÉNDICES

En este apartado se presentan algunas de las **pruebas realizadas sobre los métodos más relevantes** utilizados en esta aplicación web.

En lo que respecta al **registro de usuarios y al inicio de sesión**, la aplicación incorpora un sistema de alertas visuales que **informa al usuario sobre el éxito o el fallo de la operación**, mejorando así la experiencia de uso. A continuación, se muestran capturas donde se puede apreciar el funcionamiento de estas notificaciones:
(foto) (foto)

Asimismo, desde el punto de vista técnico, se llevaron a cabo **pruebas internas tanto en el backend como en el frontend**, con el objetivo de **verificar la correcta ejecución de los métodos sin necesidad de desplegar la aplicación completa**. Estas pruebas permitieron identificar y corregir errores lógicos, validar rutas, asegurar la integridad de los datos y comprobar las respuestas de la API en distintos escenarios.

Se comenzará explicando algunos tests del frontend, los cuales hemos podido comprobar gracias a **Karma**.



```
it('debería asignar el nombre de la cofradía según el evento y la lista de cofradías', () => {
  component.cofradias = [
    { id: 1, nombre: 'Cofradía A' },
    { id: 2, nombre: 'Cofradía B' },
  ];

  component.evento = {
    id: 10,
    nombre: 'Evento X',
    cofradia: 2,
    fecha: '2025-05-29',
  };

  component.calculaCofradiaNombre();

  expect(component.cofradiaNombre).toBe('Cofradía B');
```

Test del frontend que comprueba que funciona bien el método que obtiene el nombre de una cofradía a través de su id.

```
EventoComponent
  • debería asignar "Desconocida" si no encuentra la cofradía
  • debería asignar el nombre de la cofradía según el evento y la lista de cofradías
  • should create
```

Vista de la comprobación del código spec (tests).

Aquí se puede verificar que Karma devuelve una respuesta de éxito al procesar las pruebas necesarias para comprobar que los métodos de los tests desempeñan con éxito sus funciones.

```
calculaCofradiaNombre(): void {
  if (this.cofradias && this.evento) {
    console.log('Cofradías:', this.cofradias);
    console.log('Evento:', this.evento);
    const cofradia = this.cofradias.find(
      (c) => c.id === this.evento.cofradia
    );
    this.cofradiaNombre = cofradia ? cofradia.nombre : 'Desconocida';
  }
}
```

Código del método testeado.

En la anterior imagen se muestra una prueba realizada para verificar que el sistema recupera correctamente el nombre de la cofradía a partir del ID asociado a un evento. Como se puede observar, el proceso se ejecuta sin errores y devuelve el nombre esperado, lo que confirma el correcto funcionamiento de esta funcionalidad.

```
20
21    it('DEBE DEVOLVER LOS DATOS DEL USUARIO', () => {
22      const mockUser = { id: 1, name: 'Juan' };
23      localStorage.setItem('user', JSON.stringify(mockUser));
24      expect(service.getUsuarioData()).toEqual(mockUser);
25    });
26
```

Test del frontend que controla la obtención de los datos del usuario.

En la anterior imagen se muestra una prueba realizada para verificar que el sistema recupera correctamente los datos de un usuario, que ha sido enviado por el backend.

```
  x should create

AuthService
  • DEBE DEVOLVER LOS DATOS DEL USUARIO
  • debería ser creado
```

Vista de la comprobación del código spec de ese método.

Como se puede observar, Karma muestra que el método pasa satisfactoriamente todas las pruebas necesarias para comprobar que funciona correctamente.

```
it('DEBERIA OBTENER TODOS LOS ARTICULOS', () => {
  const dummyArticulos = [
    { id: 1, titulo: 'Articulo 1' },
    { id: 2, titulo: 'Articulo 2' }
  ];

    service.getArticulos().subscribe(articulos => {
    expect(articulos.length).toBe(2);
    expect(articulos).toEqual(dummyArticulos);
  });

  const req = httpMock.expectOne(service['apiUrl']);
  expect(req.request.method).toBe('GET');
  req.flush(dummyArticulos);
});


  it('DEBERÍA CREAR UN ARTÍCULO', () => {
  const nuevoArticulo = {
    titular: 'Nuevo artículo',
    cuerpo: 'Contenido del artículo',
    id_autor: 1
  };

  const respuestaEsperada = { ...nuevoArticulo, id: 123 };

  service.crearArticulo(nuevoArticulo).subscribe(respuesta => {
    expect(respuesta).toEqual(respuestaEsperada);
  });

  const req = httpMock.expectOne(service['apiUrl']);
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(nuevoArticulo);
  req.flush(respuestaEsperada);
});

it('DEBERIA BORRAR UN ARTICULO', () => {
  const articuloId = 1;
  service.eliminarArticulo(articuloId).subscribe(respuesta => {
    expect(respuesta).toBeNull();
  });
  const req = httpMock.expectOne(`${service['apiUrl']}/${articuloId}`);
  expect(req.request.method).toBe('DELETE');
  req.flush(null);
});
});
```

Vista del código spec del servicio de artículos.

```
DiarioService
  • DEBERIA BORRAR UN ARTICULO
  • DEBERÍA CREAR UN ARTÍCULO
  • DEBERIA OBTENER TODOS LOS ARTICULOS
  • should be created
```

Vista de la comprobación de los métodos del servicio de artículos.

En cuanto al backend, se ha utilizado PHPUnit para realizar pruebas automatizadas. Se han creado clases específicas de test que verifican el correcto funcionamiento de los métodos más relevantes de la aplicación, comprobando tanto las respuestas esperadas como la correcta persistencia de datos en la base de datos.

```php
class ArticuloTest extends TestCase
{
    public function test_se_puede_crear_un_articulo()
    {
        $usuario = \App\Models\User::factory()->create();

        $payload = [
            'titular' => 'Artículo de prueba',
            'cuerpo' => 'Contenido del artículo de prueba',
            'id_autor' => $usuario->id,
        ];

        $response = $this->postJson('/api/articulos', $payload);

        $response
            ->assertStatus(201)
            ->assertJson([
                'message' => 'Artículo creado con éxito',
                'articulo' => [
                    'titular' => 'Artículo de prueba',
                    'cuerpo' => 'Contenido del artículo de prueba',
                    'id_autor' => $usuario->id,
                ]
            ]);

        $this->assertDatabaseHas('articulos', [
            'titular' => 'Artículo de prueba',
            'id_autor' => $usuario->id
        ]);
    }
}
```

Código de prueba de la creación de artículos.

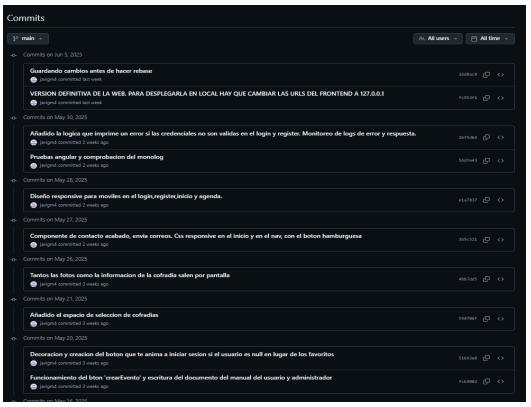
```
PASS  Tests\Feature\ArticuloTest
✓ se puede crear un articulo
```

Comprobación del éxito de la prueba.

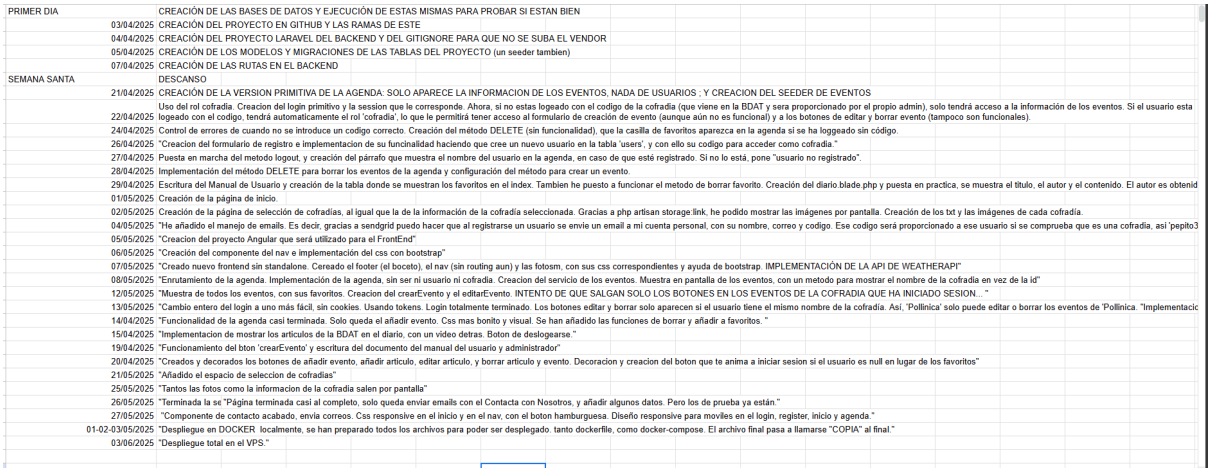
# GITHUB Y CONTROL DE VERSIONES¡

Para el control de versiones y el almacenamiento en la nube del proyecto, se ha utilizado **GitHub** como sistema de gestión de versiones.

Esta herramienta ha permitido subir el proyecto de forma progresiva, realizando *commits* desde la terminal integrada de Visual Studio. Cada una de estas versiones ha sido documentada mediante mensajes descriptivos, facilitando así un seguimiento detallado de la evolución del desarrollo.


Además, se ha llevado a cabo un seguimiento personal del trabajo en hojas de cálculo realizado durante cada jornada del periodo de desarrollo (tres meses), lo que ha permitido organizar y planificar de forma eficiente las distintas fases del proyecto.



| | | |
|---|---|---|
| PRIMER DIA | | CREACIÓN DE LAS BASES DE DATOS Y EJECUCIÓN DE ESTAS MISMAS PARA PROBAR SI ESTAN BIEN |
| | 03/04/2025 | CREACIÓN DEL PROYECTO EN GITHUB Y LAS RAMAS DE ESTE |
| | 04/04/2025 | CREACIÓN DEL PROYECTO LARAVEL DEL BACKEND Y DEL GITIGNORE PARA QUE NO SE SUBA EL VENDOR |
| | 05/04/2025 | CREACIÓN DE LOS MODELOS Y MIGRACIONES DE LAS TABLAS DEL PROYECTO (un seeder tambien) |
| | 07/04/2025 | CREACIÓN DE LAS RUTAS EN EL BACKEND |
| SEMANA SANTA | | DESCANSO |
| | 21/04/2025 | CREACIÓN DE LA VERSION PRIMITIVA DE LA AGENDA: SOLO APARECE LA INFORMACION DE LOS EVENTOS, NADA DE USUARIOS ; Y CREACION DEL SEEDER DE EVENTOS |
| | 22/04/2025 | Uso del rol cofradia. Creacion del login primitivo y la session que le corresponde. Ahora, si no estas logeado con el codigo de la cofradia (que viene en la BDAT y sera proporcionado por el propio admin), solo tendrá acceso a la información de los eventos. Si el usuario esta logeado con el codigo, tendrá automaticamente el rol 'cofradia', lo que le permitirá tener acceso al formulario de creación de evento (aunque aún no es funcional) y a los botones de editar y borrar evento (tampoco son funcionales). |
| | 24/04/2025 | Control de errores de cuando no se introduce un codigo correcto. Creación del método DELETE (sin funcionalidad), que la casilla de favoritos aparezca en la agenda si se ha loggeado sin código. |
| | 26/04/2025 | "Creacion del formulario de registro e implementacion de su funcinalidad haciendo que cree un nuevo usuario en la tabla 'users', y con ello su codigo para acceder como cofradia." |
| | 27/04/2025 | Puesta en marcha del metodo logout, y creación del párrafo que muestra el nombre del usuario en la agenda, en caso de que esté registrado. Si no lo está, pone "usuario no registrado". |
| | 28/04/2025 | Implementación del método DELETE para borrar los eventos de la agenda y configuración del método para crear un evento. |
| | 29/04/2025 | Escritura del Manual de Usuario y creación de la tabla donde se muestran los favoritos en el index. Tambien he puesto a funcionar el metodo de borrar favorito. Creación del diario.blade.php y puesta en practica, se muestra el titulo, el autor y el contenido. El autor es obtenid |
| | 01/05/2025 | Creación de la página de inicio. |
| | 02/05/2025 | Creación de la página de selección de cofradías, al igual que la de la información de la cofradía seleccionada. Gracias a php artisan storage:link, he podido mostrar las imágenes por pantalla. Creación de los txt y las imágenes de cada cofradía. |
| | 04/05/2025 | "He añadido el manejo de emails. Es decir, gracias a sendgrid puedo hacer que al registrarse un usuario se envíe un email a mi cuenta personal, con su nombre, correo y codigo. Ese codigo será proporcionado a ese usuario si se comprueba que es una cofradia, así 'pepito3 |
| | 05/05/2025 | "Creacion del proyecto Angular que será utilizado para el FrontEnd" |
| | 06/05/2025 | "Creación del componente del nav e implementación del css con bootstrap" |
| | 07/05/2025 | "Creado nuevo frontend sin standalone. Cereado el footer (el boceto), el nav (sin routing aun) y las fotosm, con sus css correspondientes y ayuda de bootstrap. IMPLEMENTACIÓN DE LA API DE WEATHERAPI" |
| | 08/05/2025 | "Enrutamiento de la agenda. Implementación de la agenda, sin ser ni usuario ni cofradia. Creacion del servicio de los eventos. Muestra en pantalla de los eventos, con un metodo para mostrar el nombre de la cofradia en vez de la Id" |
| | 12/05/2025 | "Muestra de todos los eventos, con sus favoritos. Creacion del crearEvento y el editarEvento. INTENTO DE QUE SALGAN SOLO LOS BOTONES EN LOS EVENTOS DE LA COFRADIA QUE HA INICIADO SESION..." |
| | 13/05/2025 | "Cambio entero del login a uno más fácil, sin cookies. Usando tokens. Login totalmente terminado. Los botones editar y borrar solo aparecen si el usuario tiene el mismo nombre de la cofradia. Así, 'Pollinica' solo puede editar o borrar los eventos de 'Pollinica. "Implementacic |
| | 14/04/2025 | "Funcionalidad de la agenda casi terminada. Solo queda el añadir evento. Css mas bonito y visual. Se han añadido las funciones de borrar y añadir a favoritos. " |
| | 15/04/2025 | "Implementacion de mostrar los articulos de la BDAT en el diario, con un video detras. Boton de deslogearse." |
| | 19/04/2025 | "Funcionamiento del bton 'crearEvento' y escritura del documento del manual del usuario y administrador" |
| | 20/04/2025 | "Creados y decorados los botones de añadir evento, añadir articulo, editar articulo, y borrar articulo y evento. Decoracion y creacion del boton que te anima a iniciar sesion si el usuario es null en lugar de los favoritos" |
| | 21/05/2025 | "Añadido el espacio de seleccion de cofradias" |
| | 25/05/2025 | "Tantos las fotos como la informacion de la cofradia salen por pantalla" |
| | 26/05/2025 | "Terminada la se "Página terminada casi al completo, solo queda enviar emails con el Contacta con Nosotros, y añadir algunos datos. Pero los de prueba ya están." |
| | 27/05/2025 | "Componente de contacto acabado, envia correos. Css responsive en el inicio y en el nav, con el boton hamburguesa. Diseño responsive para moviles en el login, register, inicio y agenda." |
| | 01-02-03/05/2025 | "Despliegue en DOCKER localmente, se han preparado todos los archivos para poder ser desplegado. tanto dockerfile, como docker-compose. El archivo final pasa a llamarse "COPIA" al final." |
| | 03/06/2025 | "Despliegue total en el VPS." |

# BIBLIOGRAFÍA

**Canela Ballesteros, Nicolás Javier** y **Caro Padilla, Alicia**. (s.f.). *Imágenes originales de Sagrados Titulares*. Fotografías propias utilizadas en el sistema. Usuario en redes sociales: @NicoCanela, @AliciaCaro

**Wikipedia**. (s.f.). *Hermandades y cofradías de Málaga*. Fuente provisional para los textos de los componentes informativos. Recuperado de: https://es.wikipedia.org/

**WeatherAPI**. (s.f.). *WeatherAPI Documentation*. Proveedor de datos meteorológicos mediante su API. Recuperado de: https://www.weatherapi.com/docs/

**SendGrid**. (s.f.). *SendGrid Email API Documentation*. Servicio utilizado para la gestión del envío de correos electrónicos en procesos de registro e inicio de sesión. Recuperado de: https://docs.sendgrid.com/

**OpenAI – ChatGPT**. (2024). *Asistente de redacción, análisis y generación de código*. Utilizado como apoyo durante el desarrollo del sistema y documentación técnica. Disponible en: https://chat.openai.com/

**GitHub Copilot**. (2024). *Asistente de codificación basado en inteligencia artificial*. Herramienta complementaria empleada en el desarrollo de funcionalidades y componentes. Disponible en: https://github.com/features/copilot

**Imágenes sin acreditar**. (s.f.). Algunas imágenes asociadas a las cofradías individuales han sido obtenidas a través de Internet. Al no disponer de información sobre su autoría, no ha sido posible acreditarlas adecuadamente.

**Material didáctico del ciclo formativo de Grado Superior en Desarrollo de Aplicaciones Web (2º curso)**. (s.f.). Apuntes, recursos y documentación técnica facilitados por el profesorado del ciclo DAW, que han servido de base para el diseño, desarrollo e implementación del sistema.