

**EEP  
iGroup**

**APLICACIÓN WEB VIVAJOIN**

**CFGS  
2024**

# **ESCUELA DE ESTUDIOS PROFESIONALES**

**CICLO FORMATIVO DE GRADO SUPERIOR EN  
DESARROLLO DE APLICACIONES MULTIPLATAFORMA**



## **TRABAJO DE FIN DE GRADO**

**Aplicación web vivaJoin**

**CARLOS DOLZ MARTÍN**

**JUNIO de 2024**



**EEP iGroup Arturo Soria**  
**TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES**  
**MULTIPLATAFORMA**

**APLICACIÓN WEB VIVAJOIN**

**ALUMNO:** Carlos Dolz Martín

**TUTOR ACADÉMICO:** Mariona Nadal Farré

**FECHA DE PRESENTACIÓN:** junio de 2024

**BREVE DESCRIPCIÓN:**

El proyecto *vivaJoin* consiste en el desarrollo de una aplicación *web* para la gestión de eventos sociales. Es una plataforma que permite a los usuarios crear sus propios eventos y suscribirse a eventos publicados de diferentes temáticas.

La aplicación ofrece una agradable experiencia al usuario, con una interfaz sencilla e intuitiva, contando con diferentes herramientas para poder encontrar eventos de su gusto o crear los suyos propios.

El proyecto se centra en aspectos clave como el almacenamiento de datos y archivos, las peticiones *HTTP* y las conexiones entre un servidor y un cliente, tocando todos los puntos de la arquitectura estandarizada de una *web* moderna.

**FIRMA DEL TUTOR**

**FIRMA DEL ALUMNO**

## Resumen

El proyecto es un desarrollo de una aplicación de eventos sociales llamada *vivaJoin*. El propósito de este proyecto es entender el desarrollo *web* en profundidad, desde la parte del cliente, la interacción con el servidor, los futuros problemas y las vulneraciones de seguridad y de integridad que se pueden presentar.

Está desarrollado en lo que se conoce como un *MEAN stack*, utiliza *MongoDB* como base de datos, *Express.js* como servidor, *Angular* como cliente y *Node.js* como motor. Ha resultado ser una combinación excelente para el desarrollo de aplicaciones *web*, con una rápida instalación y montaje. Es una muy buena opción a nivel profesional, y para la comprensión del flujo de datos en una aplicación *web*.

Ha resultado ser una experiencia con un gran aprendizaje y el inicio de un desarrollo que admite una gran cantidad de futuras implementaciones en su funcionalidad.

Palabras clave: *MEAN, JavaScript, Angular, MongoDB, Express.js, Node.js, web*.

## Abstract

My project is the development of a social events application called *vivaJoin*. The purpose of this project is to gain a deep understanding of web development, covering client-side, server interaction, potential future issues, and security and integrity vulnerabilities that may arise.

It's built using what's known as a *MEAN* stack, utilizing *MongoDB* as the database, *Express.js* as the server, *Angular* as the client, and *Node.js* as the engine. This combination has proven to be excellent for web application development, with quick installation and setup. It's a highly recommended choice for professional use and for understanding data flow in a web application.

This has been a valuable learning experience and the beginning of a development journey that allows for a wide range of future feature implementations.

Keywords: *MEAN, JavaScript, Angular, MongoDB, Express.js, Node.js, web.*

# Índice

<b>1. Introducción .....</b>	<b>4</b>
<b>2. Marco teórico .....</b>	<b>5</b>
<b>3. Metodología .....</b>	<b>8</b>
<b>4. Desarrollo e implementación .....</b>	<b>13</b>
<b>5. Resultado .....</b>	<b>17</b>
<b>6. Discusión .....</b>	<b>22</b>
<b>7. Conclusiones .....</b>	<b>23</b>

# 1. Introducción

En la era digital actual, existe una fuerte demanda de desarrollos de aplicaciones para satisfacer las demandas de unos usuarios cada vez más familiarizados con la navegación por *Internet*. Una de estas demandas es la participación en eventos sociales y de ocio, pues el usuario puede disfrutar de un plan para una fecha en concreto sin tener que llamar, solo con una rápida búsqueda puede apuntarse o reservar en unos minutos. La existencia de plataformas *web* que cumplan esta funcionalidad no solo afecta a usuarios, ofreciéndoles comodidad y ahorrándoles tiempo, sino que también promociona a empresas de todos los tamaños haciendo que su producto llegue a más gente.

Este proyecto responde a dicha demanda, y trata de explorar las características y funcionalidades que debe tener una página *web* de eventos sociales para poder brindar al usuario una experiencia agradable, pero centrándose en el desarrollo *web* y el aprendizaje de la arquitectura de una aplicación, en concreto de lo que se conoce como *MEAN stack*, un conjunto de tecnologías que, enlazadas entre sí, nos proporcionan un almacenamiento de datos, un servicio y una interfaz para lograr este propósito.

*Angular* es una tecnología en auge hoy en día, demandada por muchas empresas en todo el mundo, con constante servicio y actualizaciones, destinada íntegramente al desarrollo *web* y que se ajusta perfectamente a las necesidades de este proyecto. Es por eso que, por razones de aprendizaje y de cara a un futuro laboral como desarrollador, se ha considerado la elección ideal y la tecnología principal para este desarrollo en el que se pretende crear una *web* perfectamente funcional.

## 2. Marco teórico

Como ya se ha introducido antes, se ha seguido la estructura de un proyecto *MEAN stack*. Estas siglas corresponden a *MongoDB* (base de datos), *Express.js* (servidor), *Angular* (cliente) y *Node.js* (motor de ejecución). El lenguaje de programación principal ha sido *JavaScript*, y su derivado *TypeScript*, que es el lenguaje que utiliza *Angular*. De hecho, en su página web oficial, dice textualmente que [1] "*TypeScript* es *JavaScript* con sintaxis para tipado" (Página Oficial de Typescript, Microsoft, 2024).

*Angular*, según su documentación oficial, [2] es una plataforma de desarrollo construida con *TypeScript*, que incluye un *framework* basado en componentes, una extensa librería para añadir gran cantidad de funcionalidades y las herramientas de desarrollo pertinentes para optimizar nuestro código (Documentación Oficial de Angular, Google, 2024). Efectivamente, *Angular* utiliza componentes con una estructura de *HTML*, *CSS* y *TypeScript*, que son declarados en módulos, los cuales permiten la inyección de estos componentes en otros componentes, teniendo cada uno su propia estructura, lógica y estilos.

Ofrece también un sistema de enrutamiento, con validaciones para proteger las *URI* en base a permisos. Cada ruta carga un componente, bien como una vista completa de la página, o bien dentro de un *layout* o estructura, en el que se mantiene la vista y se cambia el contenido con la etiqueta *router-outlet* de su *RouterModule*.

Para los formularios, ofrece la tecnología *ReactiveForms*, con validaciones para los campos, tanto estáticas como asíncronas, una herramienta muy útil para la gestión de los datos que ingresan los usuarios en nuestras páginas web.

Aquí llegamos a las peticiones al servidor una vez introducidos estos datos. *Angular* nos ofrece los observables, que escuchan estas peticiones *HTTP*, y a los que nos podemos subscribir para gestionar estas respuestas, y los *pipes*, para regular la cantidad de peticiones que hacemos al servidor de diversas maneras.

Así mismo, para proteger también las URL de nuestro servidor, *Angular* nos ofrece los *interceptors*, que añaden permisos a la cabecera de nuestra petición, haciéndolas inaccesibles sin dicha información.

*Express.js*, según su página web oficial, [3] es un *framework* para web de *Node.js*, de montaje rápido y con pocas restricciones o directrices sobre cómo debe de implementarse, dejando esa libertad al usuario (Página Oficial de Express.js, OpenJS Foundation, 2024).

Nos ofrece una sencilla definición de rutas para el manejo de solicitudes *HTTP* con los métodos *GET*, *POST*, *PUT* y *DELETE*. También nos ofrece los *middlewares*, funciones de rápida implementación que interceptan estas peticiones y manejan los objetos de entrada y salida, permitiendo tratarlos de manera muy dinámica y sencilla. Los *middlewares* permiten realizar tareas como la validación de datos, la autenticación de los usuarios, la gestión de una sesión y la carga de archivos entre muchas otras.

*Node.js* se podría definir, basándonos en la documentación de su página web oficial, [4] como un entorno de ejecución, gratuito y de código libre, para aplicaciones en *JavaScript* (Página Oficial de Node.js, OpenJS Foundation, 2024). Es el que nos ofrece todas las librerías y herramientas para el desarrollo de un servidor en *JavaScript*.

Entre estas librerías, en el proyecto encontramos algunas con un peso muy importante en el funcionamiento de la parte del servidor:



- *Bcrypt.js*, que nos ayuda a encriptar y desencriptar las contraseñas de usuarios que se almacenarán en nuestra base de datos a la hora del registro.
- *Mongoose*, que se encarga de realizar la conexión entre nuestro servicio y nuestra base de datos, y también se encarga de definir los esquemas de datos que interactuarán con dicha base de datos, así como añadirles validaciones, para asegurar su consistencia.
- *Jsonwebtoken*, que genera *tokens* utilizados tanto en la cabecera de mis peticiones *HTTP* como en mi protección de rutas, permitiendo la creación de una sesión una vez el usuario ha accedido a la aplicación. Esta librería se complementa con *Js-cookie*, que almacena este *token* en mi parte *front* en las *cookies* de mi navegador, y con sus métodos *get*, *set* y *remove* permite el mantenimiento y el finalizado esta sesión.
- *Multer*, que funciona como una función *middleware* que permite la carga de archivos introducidos en el navegador, como imágenes en el caso de este proyecto, a un directorio local destinado a este almacenamiento.

La última tecnología para completar la dinámica del proyecto es *MongoDB*, una base de datos no relacional, que guarda objetos *JSON* en documentos dentro de colecciones, asignándoles un objeto identificador único para evitar duplicados y para poder relacionarse con documentos de otras colecciones.

Todas estas tecnologías forman lo que se conoce como una arquitectura *REST*. La primera característica que nos indica el uso de esta arquitectura son los métodos *HTTP GET*, *POST*, *PUT* y *DELETE* para realizar las conexiones entre el servidor y la aplicación cliente. La segunda, la representación de los recursos, que, en el caso de este proyecto, es en formato *JSON*. [5] Cada petición desde el cliente tiene la información necesaria para generar una

respuesta que cumpla con la solicitud, lo que fomenta la distribución de la carga de información, pues cada solicitud es independiente (Página web de Astera, Astera Software, 2024).

Como entorno de desarrollo se ha escogido *Visual Studio Code*, ampliamente utilizado en este tipo de desarrollos, que nos brinda infinidad de extensiones para facilitar la implementación y el testing del código. Entre estas extensiones se encuentran *Api.rest* para realizar peticiones *HTTP* y *Git-Hub Copilot*, una inteligencia artificial destinada al desarrollo de código que ofrece soluciones en tiempo real en base a las necesidades de nuestro proyecto.

### 3. Metodología

El enfoque principal ha sido el desarrollo de un proyecto de *Angular*, siguiendo una metodología de desarrollo conocida como *Frontend-First*. En base a eso, se comenzó con una investigación sobre el proceso de realizar la parte frontal de la aplicación, la comprensión de los componentes y las rutas, centrándose en cómo sería la experiencia para el usuario en la navegación, que datos necesitaría registrar, y que datos debería visualizar, para luego conectar esta parte con un servicio más simple, manejar estos datos y almacenarlos. Tras esto, se comenzó a utilizar el *MEAN stack* como ruta a seguir para realizar el proyecto, así que se prosiguió con el desarrollo en el *back* con *Express.js* y se crearon los primeros registros en la base de datos. En base a esto, la metodología de trabajo ha sido personalizada, siguiendo este orden de trabajo:

- Primero, un análisis de la funcionalidad que se considera conveniente para la página *web*. En esta fase, se deduce el tipo de petición *HTTP* que se quiere realizar y los datos que manejará según esa funcionalidad.

- Como segundo paso, se crea la petición *HTTP*, se define el esquema de datos a utilizar, y se realiza la petición a modo de testeo con una extensión de *Visual Studio Code* conocida como *Api.rest*, en el que se crea un archivo *.rest* con el contenido de las peticiones.
- Como tercer paso, en la parte de los servicios de *Angular*, se crea el observable que se va a retornar para realizar la suscripción en el componente, con el mismo método *HTTP* de la petición en el *back* y la *URL* en el servidor de dicha petición.
- Como cuarto paso, se crea el componente, se adapta el esquema de datos en *Angular* para que encaje con el que se definió en la parte del servicio, y se registran o se visualizan los datos en base a si es una petición *POST* o *GET*.

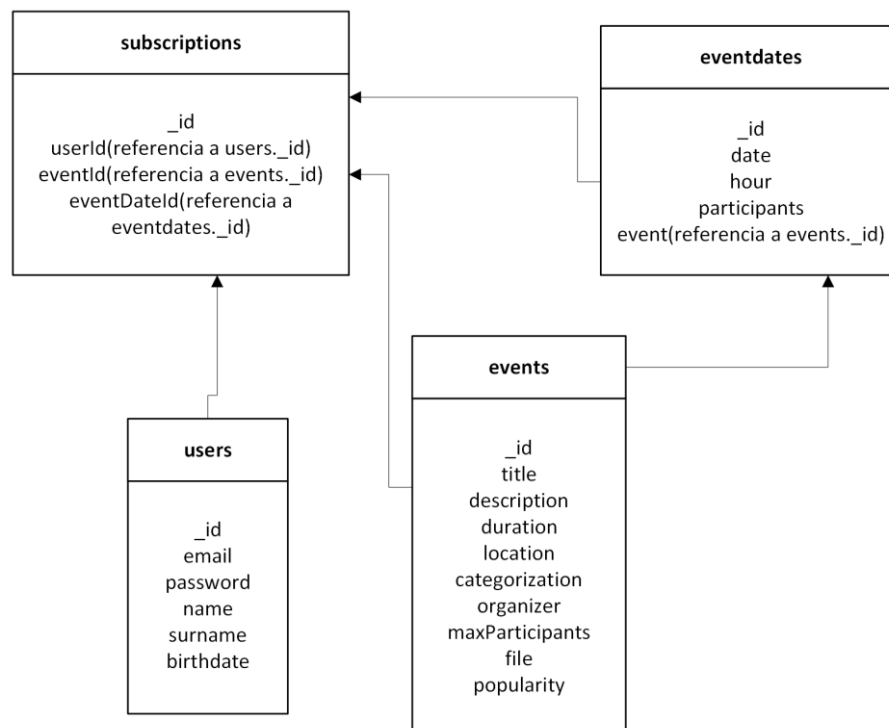
La estructura de la base de datos no se definió hasta el final, pues al ser una base de datos no relacional, es más cómodo crear colecciones con datos que se quieren almacenar que pensar la estructuración al revés, y se puede ir definiendo su arquitectura según las funcionalidades que se van implementando.

Se comenzó con el registro de usuarios y el inicio de sesión, para garantizar que hubiera un acceso y una sesión en la aplicación desde el primer momento. Luego, se continuó con el registro de eventos, para garantizar que se pudiera visualizar algo de contenido en el proyecto. A partir de este punto, el siguiente registro fue la asistencia a los eventos, y finalmente, las peticiones *GET* para visualizar todos estos registros de las colecciones en la base de datos.

En cuanto a la arquitectura de la base de datos, esta es la forma en la que se relacionan los documentos entre las colecciones explicado en la siguiente Figura 1:

Figura 1

### Esquema de la Base de Datos



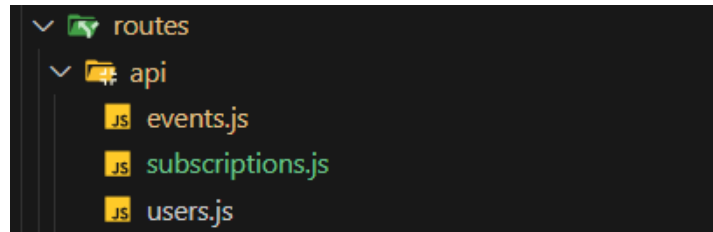
Esta Figura 1 se explica de la siguiente forma:

- Las colecciones *users* y *events* registran documentos únicos e independientes.
- Un evento se puede celebrar en diferentes fechas, de ahí la colección *eventdates*, que obtiene como referencia del evento la *\_id* de *events*.
- La suscripción al evento se hace por parte de un usuario, un evento y una fecha concreta, de ahí que la colección *subscriptions* reciba como parámetros las *\_id* de un documento concreto en las restantes colecciones, es decir, relaciona un evento, un usuario y una fecha concreta.

En cuanto a la arquitectura del servidor, la carpeta *routes* maneja todas las *URL* de la parte de la aplicación en *Express.js*, realizando las consultas a la base de datos y alojando las direcciones a las que irán destinadas las peticiones *HTTP*, como se muestra en la Figura 2.

Figura 2.

*Carpeta con las Rutas en Express.js*



En la Figura 3, se muestra un ejemplo de una consulta a la base de datos con el método que recibe una petición *POST*, y envía una respuesta en formato *JSON* al cliente:

Figura 3.

*Consulta de Registro de Usuario en la Base de Datos*

```
// Apunta a la ruta de registro
router.post('/sign-up', async (req, res, next) => {
  try {
    // Encriptamos la contraseña
    req.body.password = bcrypt.hashSync(req.body.password, 12);

    // Introducimos los datos en la BBDD con .create
    await User.create(req.body);

    // enviamos respuesta de éxito
    res.json({ message: 'Registro exitoso' });
  } catch (error) {
    next(error);
  }
});
```

En el código mostrado en la Figura 3 se puede ver una petición *POST* que registra en la base de datos un usuario, encriptando su contraseña para mayor seguridad.

Esta petición tiene un servicio correspondiente en *Angular*, alojado en la carpeta *services* apuntando a esa dirección y retornando un observable disponible para la suscripción.

A continuación, en la Figura 4, se muestra la estructuración de los servicios en la carpeta antes mencionada, y en la Figura 5, se muestra un ejemplo del código de un servicio de *Angular*:

Figura 4.

*Carpeta que aloja los Servicios en Angular*

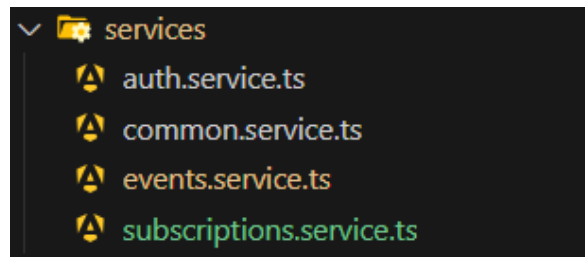


Figura 5.

*Ejemplo de código de un Servicio en Angular*

```
register(formValue: any): Observable<any> {  
  return this.httpClient.post<any>(`${this.baseUrl}/users/sign-up`, formValue);  
}
```

Toda esta interacción se podría entender con este diagrama de implementación que se muestra en la Figura 6, que nos ayuda a entender la arquitectura de la aplicación al completo:

Figura 6.

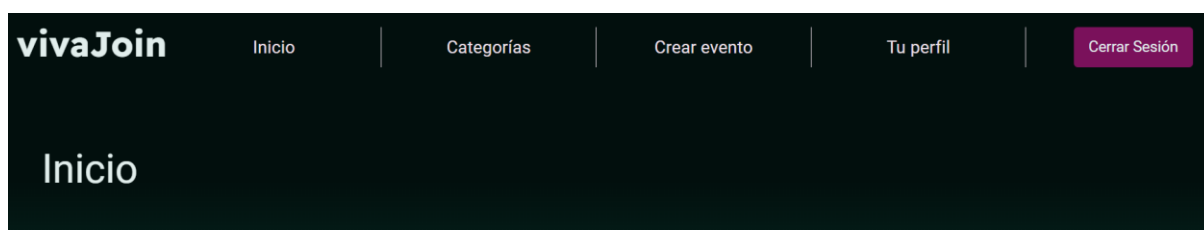
*Diagrama de implementación*



En cuanto al diseño de la aplicación, existe un *navbar* permanentemente en el *layout* o estructura de la *web*, que permite la navegación intuitiva y la búsqueda de registros o consulta de información de todo lo almacenado de diversas maneras. A continuación se muestra el *navbar* en la Figura 7.

Figura 7.

*Navbar de la aplicación*



De esta manera, la aplicación se hace muy reconocible, manteniendo siempre la misma estructura y ofreciendo al usuario solo las herramientas necesarias para ver la información relevante, sin contenido extra que le impida llegar a su objetivo, que es la suscripción a un evento.

## 4. Desarrollo e implementación

La implementación de la aplicación comenzó con el desarrollo de componentes en *Angular*. Se creó la carpeta *shared* con la intención de desarrollar pequeños componentes que fuesen reutilizables en toda la aplicación, con su correspondiente módulo para poder inyectarlos. Simultáneamente, para poder monitorizar su desarrollo y su aspecto, se creó la carpeta *pages* también con su correspondiente módulo. En esta carpeta están todos los componentes que se cargan como vista de página completa.

El archivo *app-routing.module* es el archivo de enrutamiento por excelencia de *Angular*. Es donde se señalaron las rutas correspondientes que cargaban las páginas antes descritas, que alojan en su interior estos componentes *shared*. Como estos componentes estaban destinados a hacer peticiones *HTTP* al *back*, se creó también la carpeta *services*. En *services* se apuntan a las *URL* del *back*, se retorna un observable y la propia *page* o el componente *shared* se suscribe a él para recibir los resultados de la base de datos, sirviéndose de los métodos *POST* o *GET* de *Express.js* alojados en la carpeta *routes* que ya se ha mostrado en el apartado de metodología en la Figura 2.

Esto conllevó la implementación de la protección de las rutas. Como existe una sesión, el proceso que se implementó fue el siguiente:

- Un formulario de registro con un formulario consecuente de inicio de sesión. En la respuesta que llega del *back* se recibe un *token* al inicio de sesión, creado con la biblioteca *Jsonwebtoken*.
- Este *token* se almacena en las *cookies* del navegador con la biblioteca *Cookies.js* en la parte del proyecto de *Angular*, y sirve para crear una guarda en *Angular* que comprueba la validez del *token* y que tiene la siguiente lógica: si existe *token* hay un inicio de sesión y por lo tanto las rutas del registro y el inicio de sesión quedan inaccesibles hasta que no se cierre la sesión con un botón que se implementó. En consecuencia se creó otra guarda por la cual si no existe *token* ninguna de las *URL* está disponible salvo las de inicio de sesión o registro.
- Al haber un *token* se creó un interceptor en *Angular* que añadía dicho *token* a las cabeceras de las peticiones *HTTP*. De esta manera se protegieron las rutas del servidor



en *Express.js*. Se desarrolló también un *middleware* en el servidor para verificar la validez de ese *token* que venía en la cabecera.

- Guardas e interceptores se alojaron en la carpeta *core*.

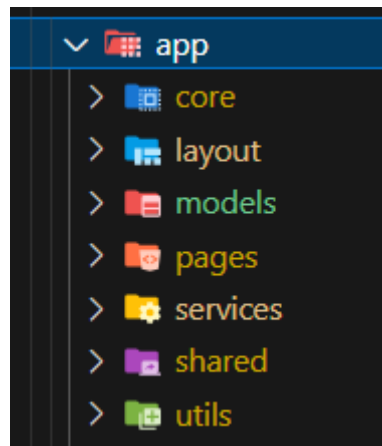
El siguiente paso fue crear el componente *layout* con su correspondiente módulo alojado en la carpeta con el mismo nombre. Este componente tiene la barra de navegación y el pie de página de manera permanente, y la etiqueta *router-outlet* que sirve para cargar cualquier página hija de la ruta padre. Una vez se accede a la sesión la base de la *URL* es *'/home'* y se carga el componente *layout*. Las consecuentes páginas se cargan dentro de este componente manteniendo la estructura.

Como hay registro de datos se utilizó la tecnología *ReactiveForms* de *Angular*, que sirve para construir un formulario con controles, asignarlos al *HTML* y poder obtener los datos en la lógica de *TypeScript*. Esta tecnología tiene validaciones por defecto para impedir que los usuarios introduzcan datos corruptos en la base de datos. Como las validaciones por defecto tienen sus limitaciones, se creó la carpeta *utils* que aloja los validadores personalizados para los formularios.

Como última mención en la parte de la implementación en *Angular* están los *models*, estructuras de modelos de datos de objeto *JSON*, que son los que recogen los datos del formulario y los envían en una petición *POST*, o reciben los datos en modelo *JSON* desde *mi back* y se aseguran de que se adapten a la misma estructura tanto en el cliente como en el servidor.

En la Figura 8 se muestra la estructuración completa del proyecto de *Angular*:

Figura 8.

*Estructuración del proyecto de Angular*

En la parte de *Express.js* ya se ha comentado como se implementaron los métodos para gestionar peticiones *HTTP* pero no se ha explicado el proceso a fondo. Se utilizó la biblioteca *Mongoose* para crear esquemas de datos que apuntaban a las colecciones en la base de datos. Estos esquemas son necesarios y recurrentes en todos estos métodos porque son los que se encargan, con sus métodos *create* y *find*, de hacer las consultas.

Cabe mencionar que en uno de los formularios del proyecto se incluyen archivos, en concreto imágenes. Para esta implementación se utilizó la biblioteca *Multer*, que, mediante un *middleware*, recoge los archivos que se introducen en el *front* y los envía a un directorio de almacenamiento local alojado en la raíz del proyecto de *Express.js*.

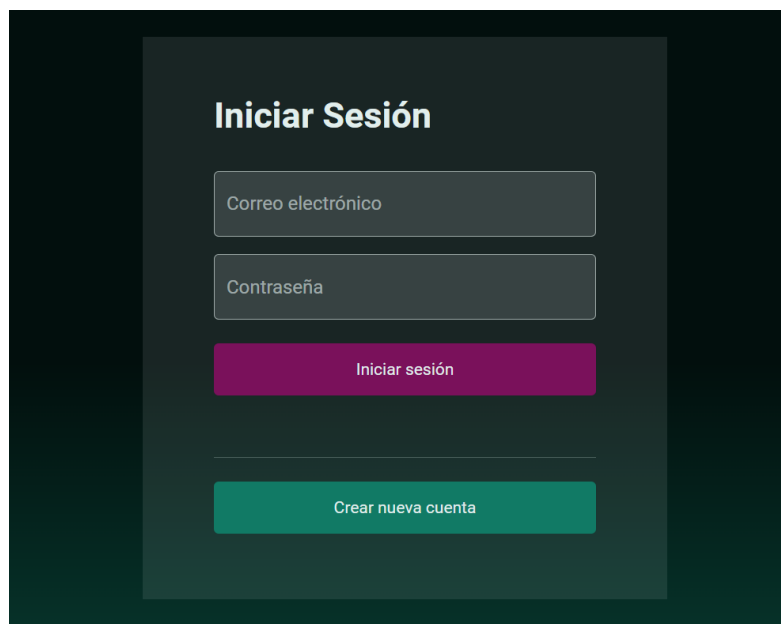
Por último, para controlar que no existan fechas o suscripciones obsoletas en la base de datos, se creó un *script* que se ejecuta cuando se inicia la aplicación que realiza una consulta de borrado.

## 5. Resultado

Como resultado del análisis y la posterior implementación, se procede a mostrar la aplicación ya en funcionamiento. Iniciamos la aplicación por el *login*, que tiene este aspecto mostrado en la Figura 9:

Figura 9

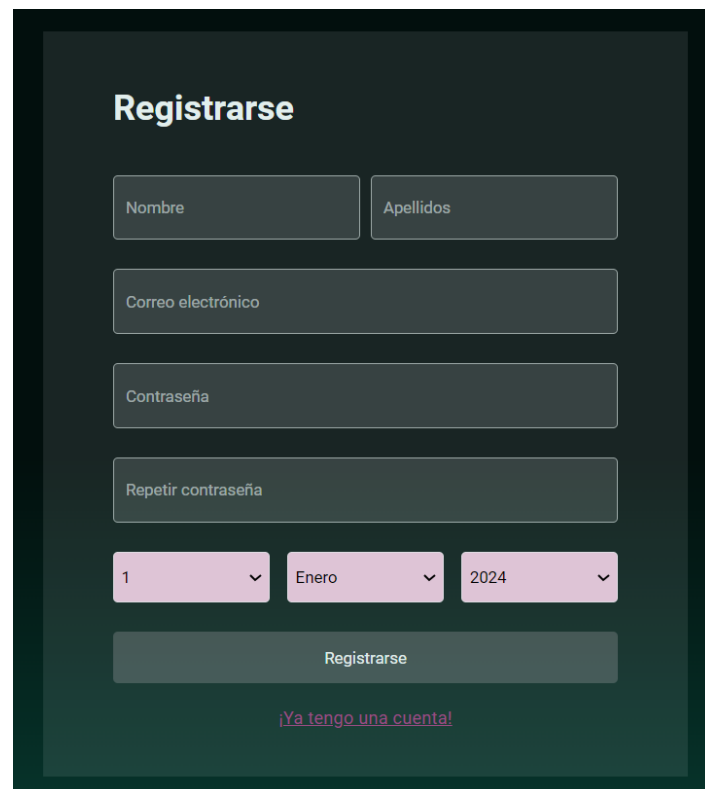
*Formulario de Login*

The image shows a login form titled "Iniciar Sesión" (Login) centered on a dark background. The form is contained within a light gray rectangular box. It features two input fields: "Correo electrónico" (Email) and "Contraseña" (Password). Below these fields is a magenta button labeled "Iniciar sesión" (Login). At the bottom of the form is a teal button labeled "Crear nueva cuenta" (Create new account).

En este formulario se puede acceder a la aplicación si las credenciales son válidas o bien acceder al formulario de registro.

Este formulario de registro tiene validaciones para evitar datos corruptos, como nombres que solo sean un espacio, contraseña segura o mayoría de edad. También cuenta con un enlace para redireccionar al usuario de nuevo al formulario de inicio de sesión. Tiene el siguiente diseño que se muestra a continuación en la Figura 10:

Figura 10:

*Formulario de Registro*

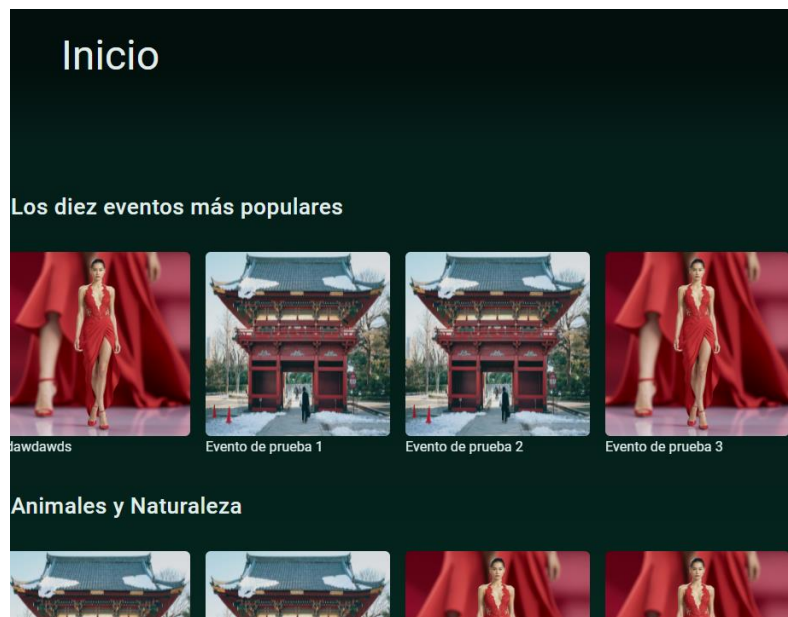
The image shows a registration form titled "Registrarse" on a dark background. The form includes the following elements:

- Two input fields for "Nombre" (Name) and "Apellidos" (Surname).
- A single input field for "Correo electrónico" (Email).
- A single input field for "Contraseña" (Password).
- A single input field for "Repetir contraseña" (Repeat password).
- Three date pickers: the first shows "1", the second shows "Enero", and the third shows "2024".
- A "Registrarse" button.
- A link at the bottom that says "¡Ya tengo una cuenta!" (I already have an account!).

Una vez dentro de la aplicación podemos ver el inicio. Tiene unos *slider* que muestran un límite de diez eventos. El primer *slider* los diez eventos más populares, teniendo en cuenta que la popularidad de los eventos aumenta con cada suscripción. Los siguientes *sliders* muestran los diez eventos más recientes de cada categoría. En la Figura 11 se procede a mostrar el inicio de la aplicación.

Figura 11.

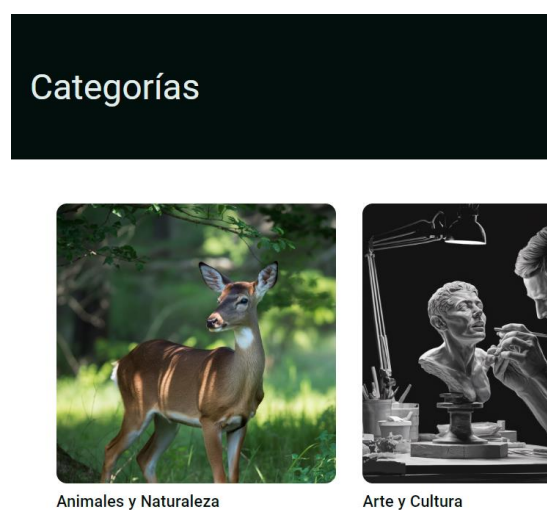
*Inicio o Home de la aplicación*



Siguiendo la estructura del *navbar* tenemos el apartado de categorías. Dentro de cada categoría se hace una consulta para sacar todos los registros que haya en la base de datos buscando por dicho parámetro. Se procede a mostrar el aspecto del apartado de categorías en la Figura 12.

Figura 12.

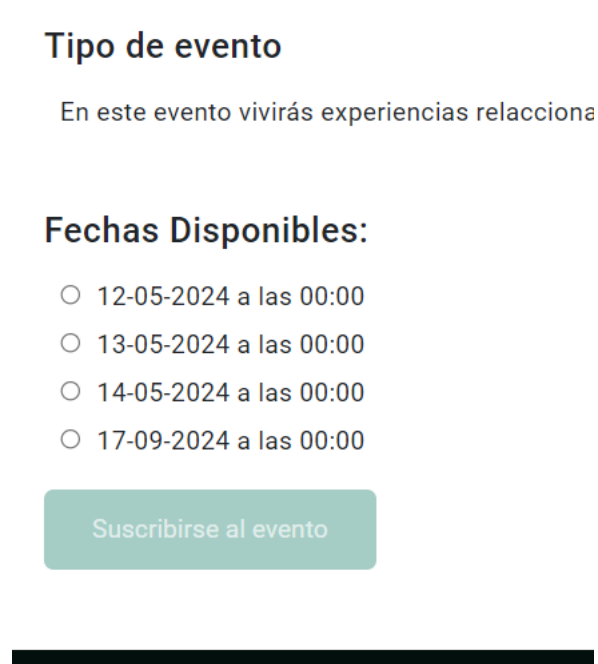
#### *Apartado de categorías*



En cualquiera de los apartados anteriores en los que se visualizan eventos, al hacer *click* en la imagen accedemos al detalle de dicho evento, y aquí podemos encontrar su información y que fechas hay disponibles para suscribirse, como se procede a mostrar en la Figura 13:

Figura 13.

*Formulario para Suscripción a Eventos en base a su fecha*



**Tipo de evento**

En este evento vivirás experiencias relaciona

**Fechas Disponibles:**

- ☐ 12-05-2024 a las 00:00
- ☐ 13-05-2024 a las 00:00
- ☐ 14-05-2024 a las 00:00
- ☐ 17-09-2024 a las 00:00

Suscribirse al evento

---

El siguiente apartado de nuestra aplicación es la creación de eventos. Cuenta con validaciones de texto que ya se han visto en el formulario de registro de usuarios. También cuenta con una validación extra para introducir imágenes comprobando si la extensión del archivo es válida.

Cabe destacar que el usuario puede registrar varias fechas para un evento (hasta un máximo de diez), y con un solo botón, todos los registros se almacenarían en la base de datos. En la Figura 14 se procede a mostrar el formulario de creación de fechas.

Figura 14.

*Formulario de Creación de Fechas para Eventos*

¿Cuántas veces se va a repetir el evento? (Máximo 10 registros):

Registra aquí las fechas:

de 2024

Registrar

Por último, el usuario puede consultar todos los registros en el apartado de su perfil, junto con parte de los datos que registró en un principio. Aquí puede también cancelar la suscripción de un evento si lo desea. Se procede a mostrar dicho apartado en la Figura 15:

Figura 15

*Perfil de Usuario*

## Datos del usuario:

Nombre completo: **Carlos Dolz Martín**Fecha de nacimiento: **12-09-1994**Correo electrónico: **carlosdolzm@gmail.com**

## Mis suscripciones:

**Cena en la Azotea "Noches de Estrella"**

Asistencia el día 26-07-2024 a las 21:00 horas

Cancelar asistencia

Se puede apreciar que la aplicación tiene una interfaz amigable para el usuario, los apartados son claros y concisos, y su uso no requiere de formación especial o conocimientos avanzados de informática. Además, al estar protegidas todas las *URL*, hasta las que no están especificadas, la ejecución de la aplicación no se interrumpe en ningún momento manteniendo un paso fluido entre cambio de páginas.

## 6. Discusión

Hay varios puntos a mejorar en la aplicación, que se han destinado a futura implementación, si bien su funcionalidad es completa en cuanto a registro y visualización de datos.

El primero es una gestión de permisos de usuario. Se introduciría así el concepto de administrador, separando la función de crear eventos y gestionar las cuentas de las funciones del usuario, que se limitaría a suscribirse a los eventos.

No se ha implementado tampoco un cambio de contraseña, lo cual obliga al usuario a tener que registrarse con un nuevo correo en caso de olvido de esta. En esta línea, tampoco se ha implementado la posibilidad de editar en general la información del usuario.

Son realmente muy amplias las posibilidades en este tipo de desarrollos, teniendo en cuenta que en el proyecto no se han contado con *APIs* externas que añaden mucha funcionalidad a la aplicación, aunque no se ha considerado como uno de los objetivos principales del proyecto. Personalmente lo he considerado más como un reto de aprendizaje, y en ese sentido, el proyecto ha superado mis expectativas.

*Angular* se caracteriza por tener una curva de aprendizaje muy complicada, y en la aplicación de mis conocimientos en este proyecto, he llegado a ver las ventajas de su



arquitectura de separación de componentes, permitiendo luego una inyección fácil a la hora de construir las vistas y las plantillas.

## 7. Conclusiones

Considero que el proyecto ha cumplido mis expectativas a nivel de aprendizaje. Al desarrollar la funcionalidad de la aplicación tanto en la parte del cliente como en la parte del servidor, he podido llegar a entender los retos que supone el desarrollo de una aplicación *web*, tanto en su fase de análisis, que debe ser minuciosa y exhaustiva, como en la parte de implementación.

En todos los desarrollos se plantean nuevos retos que no se han previsto, y aunque mucho del conocimiento aplicado en este proyecto es compatible con otros proyectos, hay muchas maneras de llegar al mismo camino, por lo que considero que es un buen inicio en el desarrollo *web*, y a la vez la apertura de un horizonte de posibilidades aún por explorar.

## Referencias.

- [1] Página Oficial de TypeScript, Microsoft. (2024). <https://www.typescriptlang.org/>
- [2] Documentación Oficial de Angular, Google. (2024). ¿Qué es Angular? <https://angular.io/guide/what-is-angular>
- [3] Página Oficial de Express.js, OpenJS Foundation (2024). <https://expressjs.com/>
- [4] Página Oficial de Node.js, OpenJS Foundation (2024). <https://nodejs.org/en>
- [5] Página web de Astera, empresa de gestión de datos, Astera Software (2024). Beneficios de las API REST. <https://www.astera.com/es/type/blog/rest-api-definition/>
- Mario Girón, Garage de Ideas | Tech, Crea tu primera app con Mean Stack paso a paso (16 de mayo de 2023), <https://www.youtube.com/watch?v=EB9-bxI8whI&t=457s>
- Página de npmjs para librerías Node.js, (2024), <https://www.npmjs.com/package/multer>, <https://www.npmjs.com/package/jsonwebtoken>, <https://www.npmjs.com/package/bcrypt>
- Página de documentación de Swiper.js para versión de Angular 8.4.6, (2024), <https://v8.swiperjs.com/angular>
- Página oficial de MongoDB, (2024), <https://www.mongodb.com/>