Resumen

Mi 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.

Esta 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

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 navegación 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 centrándonos 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, utilizadas en conjunto, 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, me parece la elección ideal para este desarrollo en el que se pretende crear una web perfectamente funcional.

2. Marco teórico

Como ya he introducido antes, he 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 url 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.

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, así como 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, 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 mi 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 mi navegador, como imágenes en el caso de mi proyecto, a un directorio local destinado a este almacenamiento.

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

Como entorno de desarrollo he 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

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

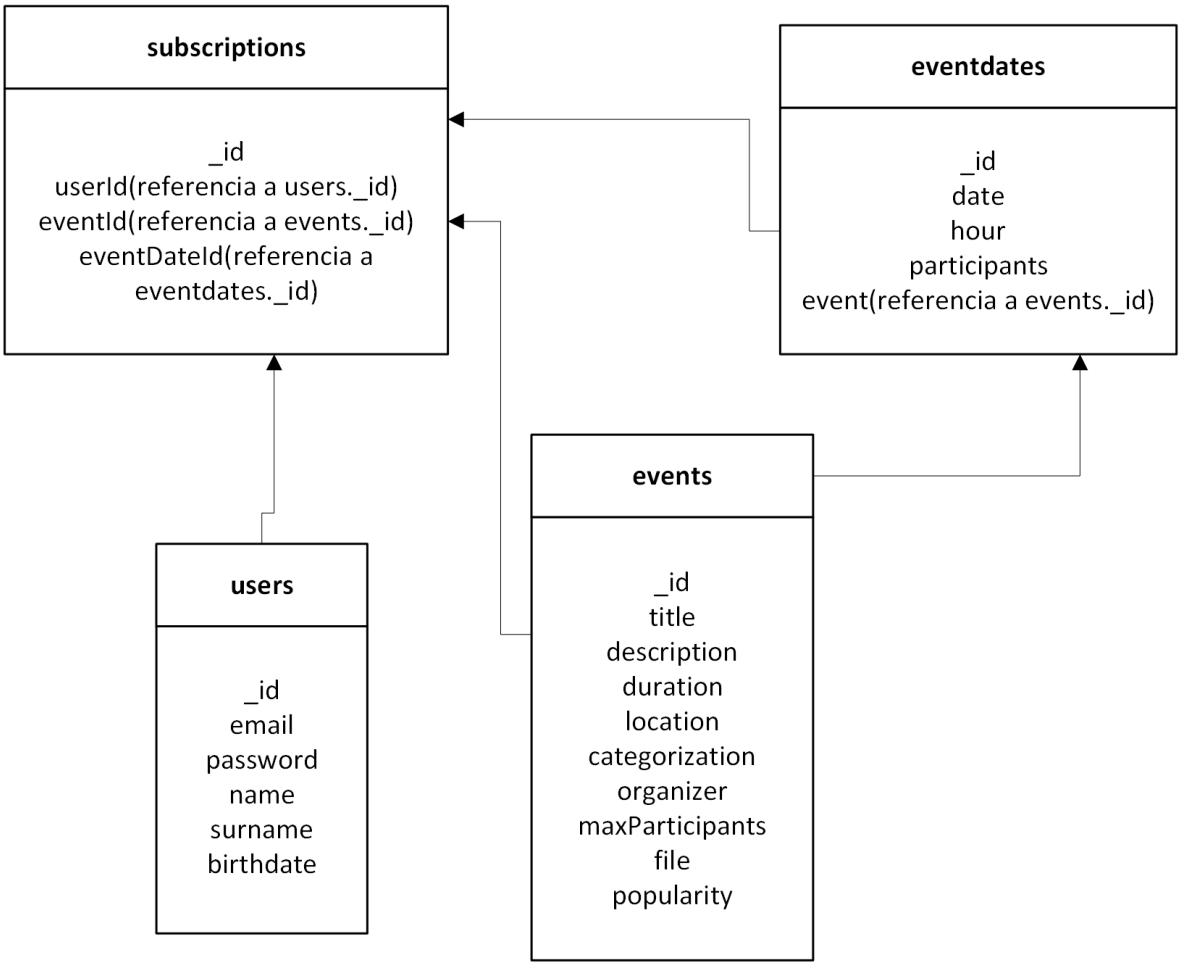
- Primero, un análisis de la funcionalidad que creo conveniente que tenga mi página web. En esta fase, deduzco el tipo de petición HTTP que quiero realizar y que datos va a manejar en base a dicha funcionalidad.
- Como segundo paso, creo la petición HTTP, defino el esquema de datos a utilizar, y realizo la petición a modo de testeo con una extensión de VisualCode conocida como api.rest, en el que creamos un archivo .rest con el contenido de las peticiones.
- Como tercer paso, en la parte de los servicios de Angular, creo el observable que voy a retornar para subscribirme en mi componente, con el mismo método HTTP de mi petición en el back y la url de dicha petición.
- Como cuarto paso, creo el componente, adapto mi esquema de datos en Angular para que encaje con el que se definió en la parte del servicio, y registro o visualizo 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 quiero almacenar que

pensar la estructuración al revés, y puedo ir definiendo su arquitectura según las funcionalidades que voy implementando.

Comencé con el registro de usuarios y el inicio de sesión, para garantizar que había un acceso y una sesión en la aplicación desde el primer momento. Luego, comencé con el registro de eventos, para garantizar que se podía visualizar algo de contenido en el proyecto. A partir de este punto, mi siguiente registro fue la asistencia a los eventos, y finalmente, las peticiones get para visualizar todos estos registros de mis colecciones.

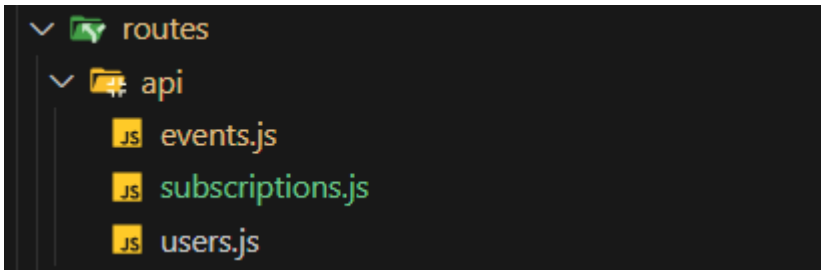
En cuanto a la arquitectura de mi base de datos, esta es la forma en la que se relacionan los documentos entre las colecciones explicado en un diagrama:



Este diagrama 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 en las restantes colecciones.

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.

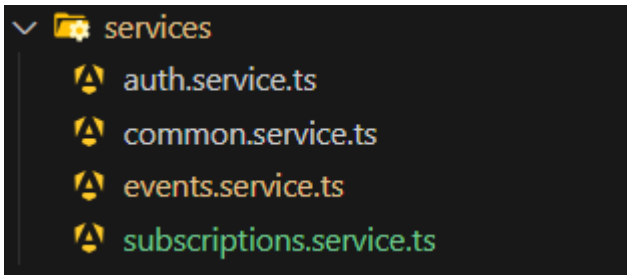


se puede comprobar aquí la dinámica de una consulta básica en la base de datos con su respuesta hacia el cliente:

```
// 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 imagen anterior podemos 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, alojados en la carpeta services apuntando a esa dirección y retornando un observable disponible para la suscripción, como muestro a continuación:

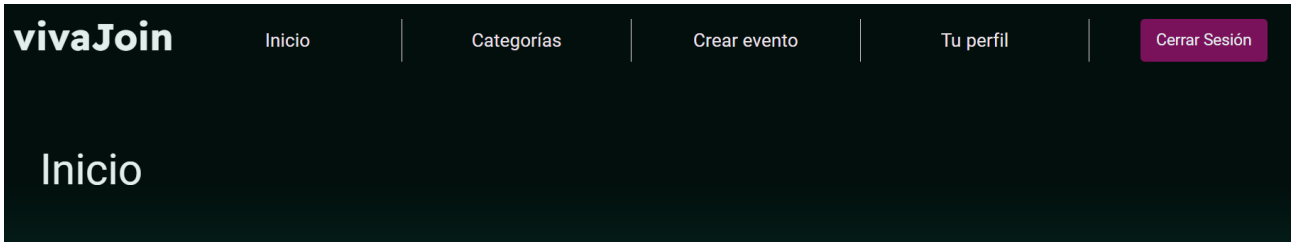


Y aquí muestro el código correspondiente a ese servicio, en base a la petición post mostrada anteriormente:

```
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 voy a mostrar a continuación, que nos ayuda a entender la arquitectura de la aplicación al completo:

En cuanto al diseño de mi 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.



De esta manera, la aplicación se hace muy reconocible, manteniendo siempre la misma estructura y ofreciendo al usuario solo las herramientas necesarias para per 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. Simultaneamente y 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 señalé 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 o bien 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 he mostrado en el apartado de metodología.

Esto me llevó a comenzar a proteger todas las rutas. Como existe una sesión, el proceso que implementé 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 mi 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, y, en consecuencia, se creó otra guarda, por la cual, si no existe token, ninguna de las url están disponibles salvo las de inicio de sesión o registro.
- Al haber un token, se creó un interceptor en Angular, que añadía este token a las cabeceras de las peticiones HTTP, y de esta manera, se protegieron mis 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'.

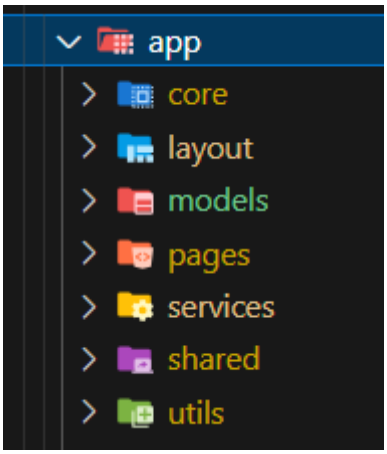
Mi siguiente paso fue crear el componente, con su correspondiente módulo, llamado 'layout' alojado en la carpeta con el mismo nombre. Este componente tiene la barra de

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', se carga el componente layout, y las consecuentes páginas se cargan dentro de este componente manteniendo la estructura.

Como hay registro de datos, me apoyé en la tecnología ReactiveFroms 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 mi base de datos. Como las validaciones por defecto tienen sus limitaciones, se creó la carpeta 'utils' que aloja los validadores personalizados para mis 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 o recogen los datos de mi 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.

Esta sería la estructura de implementación del proyecto:



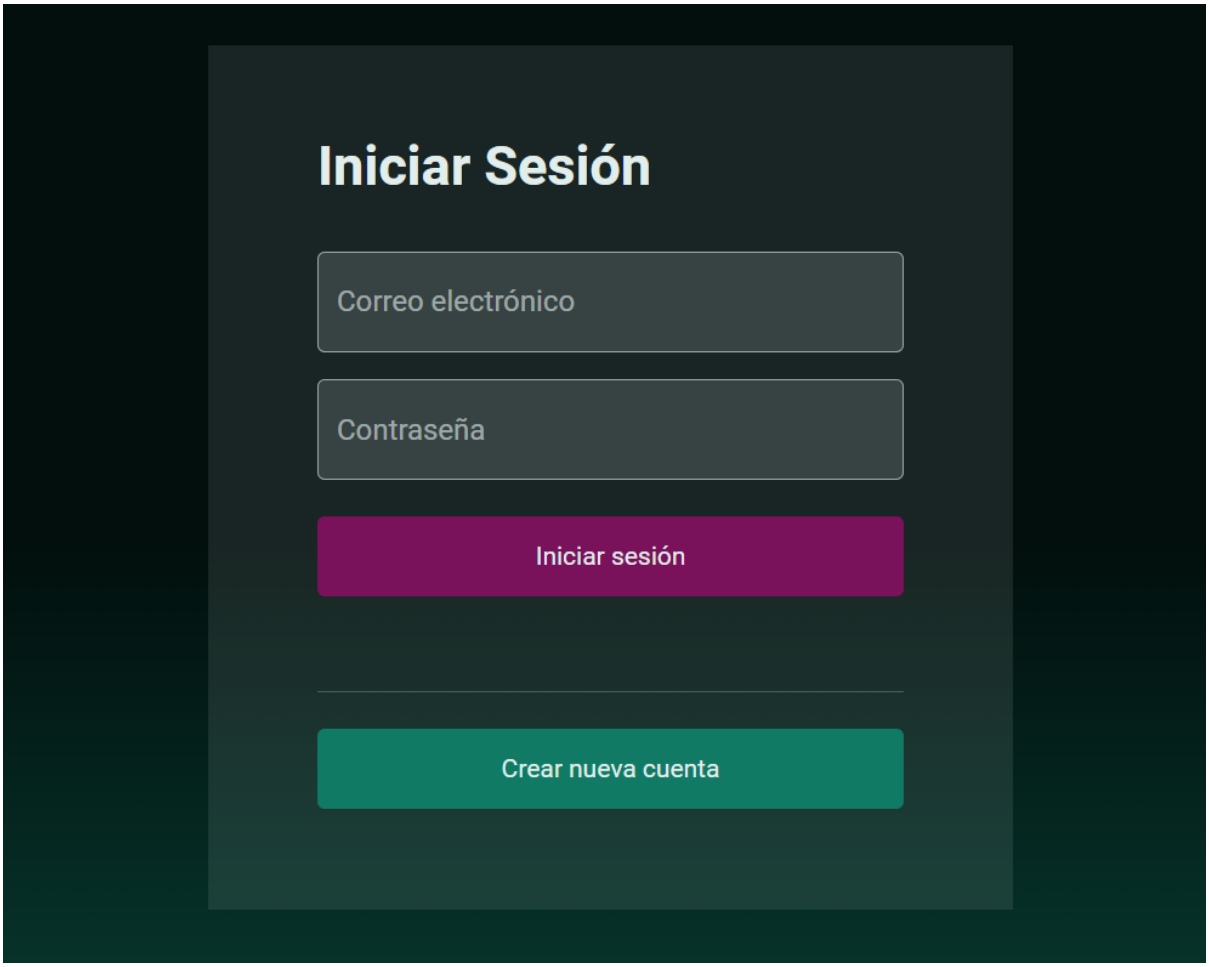
En la parte de Express ya he dado unas pinceladas de como implementé los métodos para gestionar peticiones HTTP, pero no he explicado el proceso a fondo. Utilicé la

biblioteca Mongoose para crear esquemas de datos que apuntaban a mis colecciones en la base de datos, y son necesarios y recurrentes en todos estos métodos porque son los que se encargan, con sus métodos 'create' y 'find', de hacer las consultas.

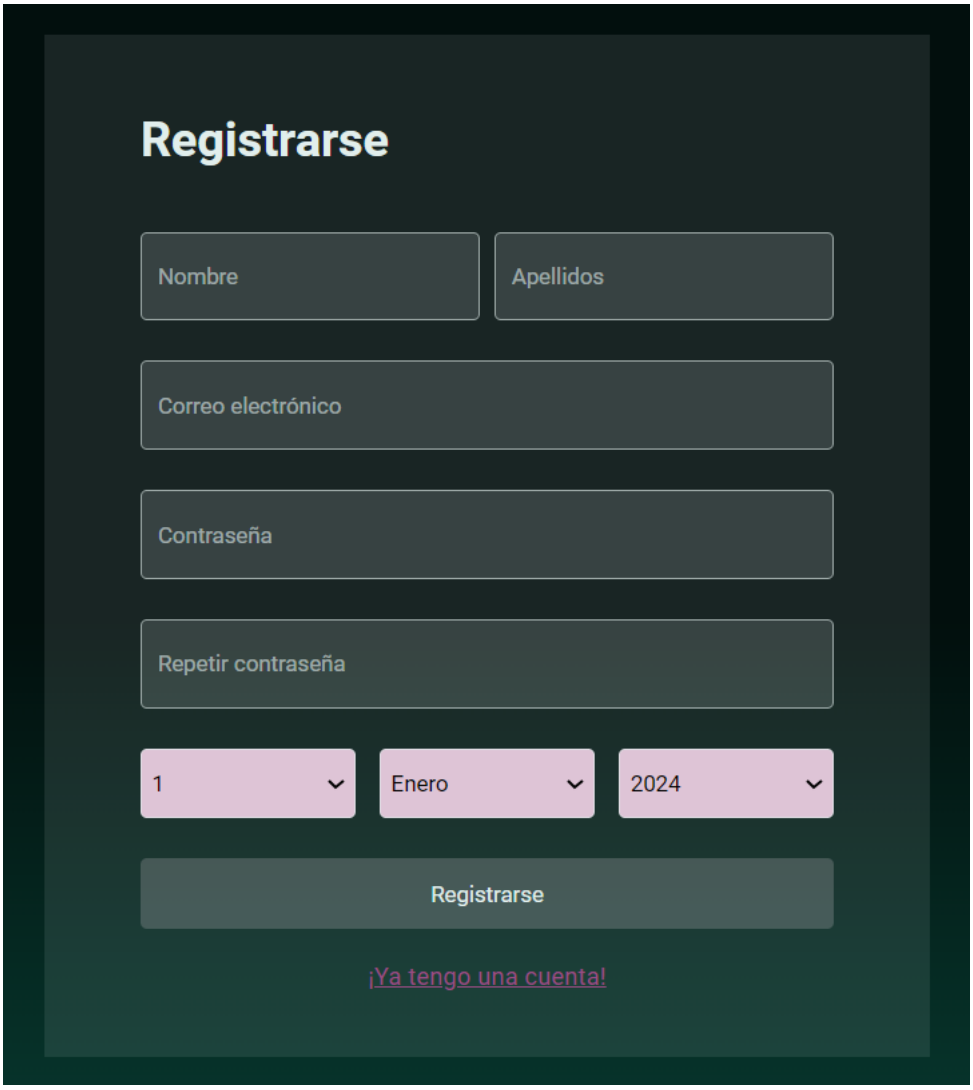
Cabe mencionar que, en mi aplicación, en uno de los formularios, se incluyen archivos, en concreto imágenes, y esta implementación la hice utilizando la biblioteca Multer, que mediante un middleware, recoge los archivos que se introducen en mi front y los envía a un directorio de almacenamiento local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

5. Resultado

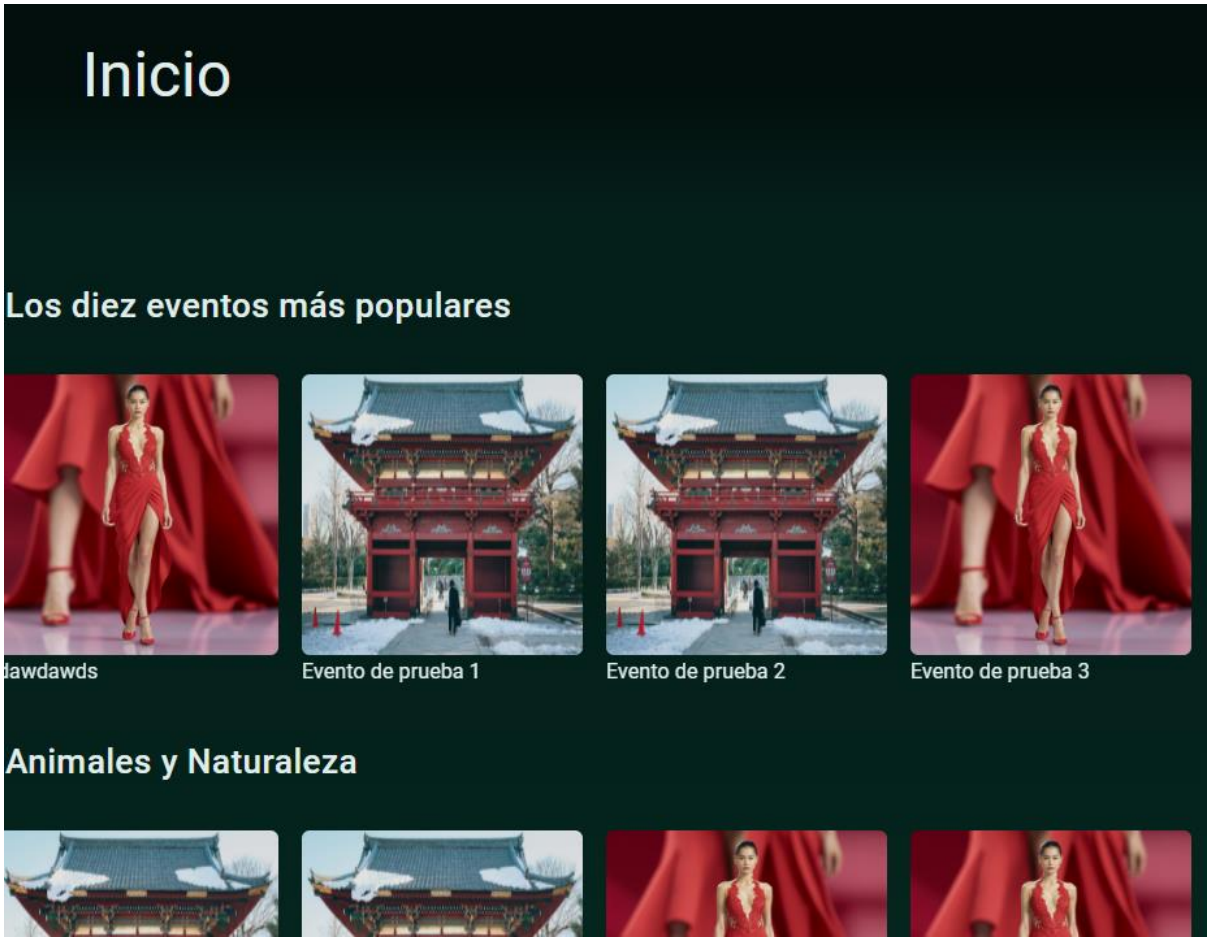
Como resultado de mi análisis y mi posterior implementación, procedo a mostrar el resultado en la aplicación ya levantada. Iniciamos la aplicación por el login, que tiene este aspecto:



Puedes acceder o redireccionar al formulario de registro si no tienes una cuenta. Este formulario si tiene validaciones para evitar datos corruptos, como nombres que solo sean un espacio, contraseña segura y mayoría de edad. También cuenta con un enlace para redireccionar al usuario a el formulario de inicio de sesión. Tiene el siguiente diseño:



Una vez dentro de la aplicación, podemos ver el inicio. Tiene unos slider que muestran diez eventos, los más populares, basándose en la popularidad del evento que crece con cada suscripción, o bien diez eventos por categoría.



Siguiendo la estructura del 'navbar', tendríamos las categorías. Dentro de cada categoría se haría una consulta para sacar todos los registros que haya en la base de datos buscando por dicho parámetro.

Categorías

Animales y Naturaleza

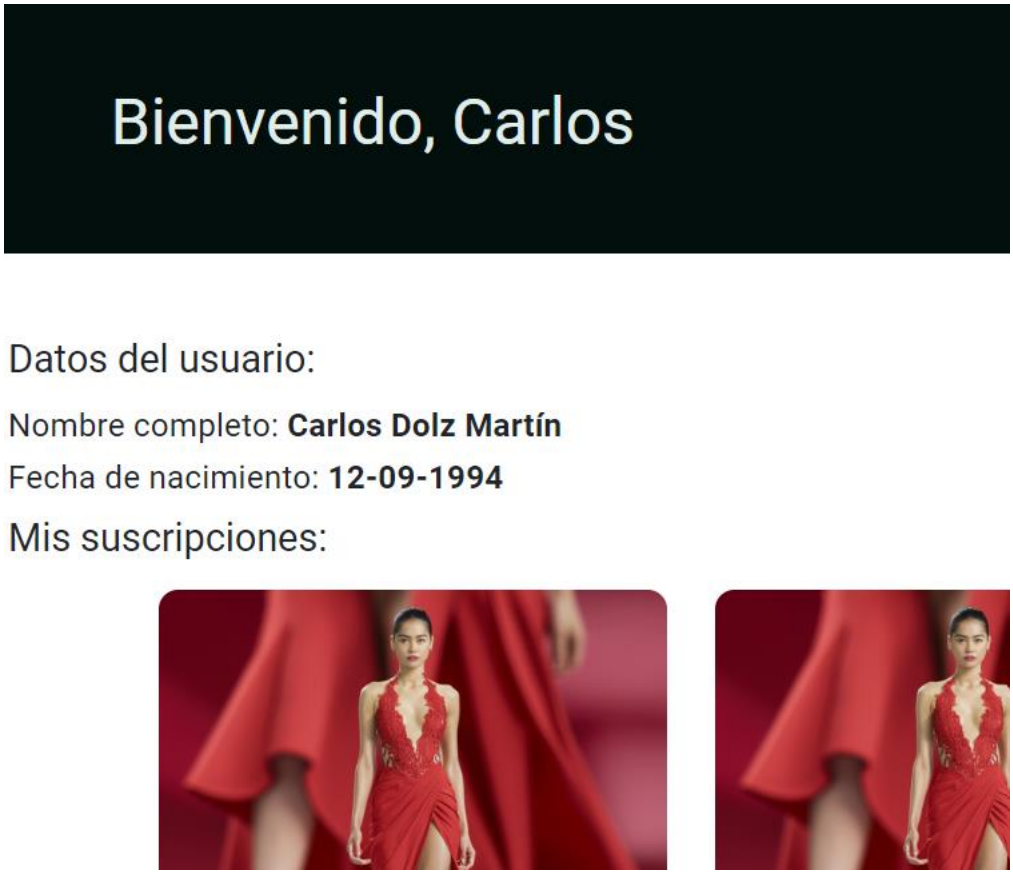
Arte y Cultura

De alguna manera, accederíamos a un evento, pues todos son enlaces que nos llevan al detalle, y aquí podríamos encontrar la información del evento y que fechas hay disponibles para suscribirse:

Como ya he comentado durante la implementación, se ofrece al usuario la opción de crear un evento, debe cumplimentar unos datos en el formulario para poder registrarlo. Este formulario también cuenta con validaciones.

Cabe destacar que el usuario puede registrar varias fechas para un evento hasta un máximo de 10, y con un solo botón, todos los registros se almacenarían en la base de datos.

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. Este es el aspecto de la página de perfil:



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, y 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 he destinado a futura implementación, si bien la funcionalidad de la aplicación 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 la 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 contraseña, ni la baja a eventos, quedando estos registros de manera permanente en la base de datos, cosa que a futuro debería ser implementada para optimizar el espacio.

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 considero que fuese el objetivo 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/

Documentación Oficial de Angular, Google. (2024). *¿Qué es Angular?* https://angular.io/guide/what-is-angular

[2] Página Oficial de Express.js, OpenjS Foundation (2024). https://expressjs.com/

[3] Página Oficial de Node.js, OpenjS Foundation (2024). https://nodejs.org/en

Mario Girón, Garage de Ideas | Tech, Crea tu primera app con Mean Stack paso a paso(16 de mato 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

Página de documentación de Swiper.js para versión de Angular 8.4.6, (2024), https://v8.swiperjs.com/angular