1. Introducción

1.1. Presentación y objetivos

Os presento Viva-join, una aplicación web de eventos sociales. Me pareció una gran idea desarrollar una aplicación así, en la que el usuario pudiese registrarse, registrar eventos y poder asistir a ellos, pues es un tipo de aplicación que yo utilizo y encuentro realmente útil.

La idea principal era elegir una temática que me permitiese tener flexibilidad a la hora de implementar las tecnologías, que tuviese una lógica compleja en cuanto a la cantidad de funcionalidades que se pueden añadir, y a la vez fuese intuitiva para el usuario y fácil a la hora de interactuar con ella.

He utilizado las tecnologías de Angular y Express.js, en lo que se conoce como un MEAN stack (MongoDB, Express.js, Angular y Node.js) por su rápida implementación y su facilidad a la hora de desarrollar la aplicación, centrándome en la estructura de un C.R.U.D que hemos conocido durante el curso (Create, Read, Update, Delete) transformado al flujo completo de una aplicación web, es decir, el manejo de las peticiones HTTP con los métodos POST, GET, PUT y DELETE.

1.2.Justificación.

Desde que inicié mis estudios en la programación, el desarrollo web siempre ha sido la parte que más atractivo me ha generado como salida profesional. Este proyecto intenta abarcar los conocimientos de funcionamiento usual de una aplicación web como las que vemos actualmente, el flujo de datos, su almacenamiento, su parte visual e interactiva...

Considero que este proyecto, aun siendo menor que una aplicación web con un servidor potente, refleja con exactitud cómo se comporta cualquier web que utilicemos normalmente.

Mi motivación era entender el proceso de vida de un dato, desde que se escribe en una página web, hasta llegar a la base de datos alojada en el servidor, entender el funcionamiento interno de una web y la cantidad de procesos que se hacen mientras navegamos por ella. También quería comprender procesos habituales como el registro de un usuario, el encriptamiento de contraseñas, el mantenimiento de una sesión mediante tokens... y muchas más cosas que he desarrollado y que me encargaré de explicar más adelante.

2. Marco teórico.

2.1. Descripción de las tecnologías utilizadas.

Como ya he introducido antes, he seguido la estructura de un proyecto MEAN stack. El lenguaje de programación principal ha sido JavaScript, pues TypeScript, que es el lenguaje que utiliza Angular, es un derivado de este. De hecho, en su página web oficial, dice textualmente que "TypeScript es JavaScript con sintaxis para tipado" (Página Oficial de Typescript, Microsoft, 2024).

No ha sido realmente una decisión motivada por nada en particular, simplemente me gustaba Angular como framework a utilizar en mi parte front de la web, y sentía que el resto de elementos se adaptaban mejor que otros a mis necesidades.

Angular, según su documentación oficial, 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, 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.

La primera es 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.

Otra muy importante es 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.

La siguiente librería clave en mi proyecto es 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.

Por último, nos queda nombrar la librería 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 para consultar y relaciones entre documentos de otras colecciones.

3. Metodología

3.1. Enfoque metodológico.

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 y almacenarlos.

Fue entonces cuando descubrí 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.

Cabe destacar que la estructura de la base de datos no se definió hasta el final, pues al ser una base de datos no relacional, es mucho más cómodo ir creando las colecciones en base a los datos que quiero almacenar que pensar la estructuración al revés, y puedo ir definiendo su arquitectura en base a 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.

Referencias.

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

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

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