Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

TripTrackingTT: aplicación web para seguimiento de rutas de viajes

TripTrackingTT: Web application for travel route tracking

Yago Pérez Molanes

La Laguna, 12 de julio de 2023

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"*TripTrackingTT: aplicación web para seguimiento de rutas de viajes*"

ha sido realizada bajo su dirección por D. **Yago Pérez Molanes**, con N.I.F. 78.819.642-E.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 12 de julio de 2023

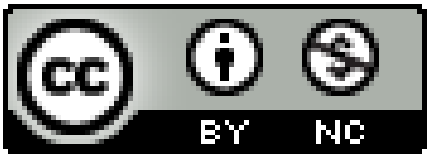
Agradecimientos

Me gustaría agradecer en primer lugar a mi madre, por ser la persona que siempre ha estado ahí para ayudarme, para escucharme cuando tenía problemas con las asignaturas y las prácticas, y por ser la persona que me apoyó para salir fuera de mi isla de residencia e iniciar esta etapa universitaria.

También, a las personas que he conocido en la residencia y la universidad, gracias a ellos mi estancia aquí ha sido mucho más amena y acogedora, y me han permitido crecer como persona, conocer a muchos amigos y hacer migas con ellos.

Por último, a mi tutor académico que me dio la oportunidad de empezar este proyecto, y a los profesores que han impartido clase durante mi periodo académico.

Licencia



Resumen

Es muy común el uso de sistemas que permiten realizar un seguimiento de la posición del usuario, y emplear la información de geolocalización para poder mostrar los datos obtenidos en un mapa interactivo.

Inicialmente, el objetivo del presente proyecto era desarrollar una aplicación web progresiva, o webapp que realizara un seguimiento de las rutas aéreas que tomase el usuario, pudiendo visualizar la ruta en un mapa interactivo en tiempo real, de forma offline, sin necesidad de depender de ninguna clase de conectividad.

Sin embargo, en el desarrollo del proyecto, con la API web soportada para calcular los datos recibidos por el GPS aún no es posible calcular la posición del usuario sin conexión, ya que internamente depende de las tecnologías Wi-Fi, Bluetooth y redes móviles para calcular dichos datos.

Por lo tanto, el enfoque del trabajo consiste en desarrollar una aplicación web progresiva diseñada para realizar un seguimiento o tracking, de rutas genéricas, ofreciendo así la posibilidad de que en el futuro cuando se desarrolle la tecnología que permita que en una aplicación web se pueda calcular la posición del usuario de forma offline, sea posible implementarse en la app.

El desarrollo de la aplicación se basará en stack MEAN, la tecnología por parte del cliente que se empleará como framework de desarrollo web será Angular, mientras que del lado del servidor se usará Express y NodeJS para construir la aplicación web servidora, además del empleo de MongoDB como base de datos no relacional.

Palabras clave: Geolocalización, Webapp, Offline, API, GPS, Wi-Fi, Bluetooth, Tracking, MEAN, Framework, Angular, Servidor, Express, NodeJS, MongoDB

Abstract

It is very common to use systems that allow tracking of the user's position and to use geolocation information to display the data obtained on an interactive map.

Initially, this project aimed to develop a progressive web application or web app, that would track the air routes taken by the user, being able to visualise the route on an interactive map in real-time, offline, without the need to rely on any kind of connectivity.

However, in the project's development, with the supported web API to calculate the data received by GPS it is not yet possible to calculate the user's position offline, as it internally relies on Wi-Fi, Bluetooth and mobile network technologies to calculate such data.

Therefore, the focus of the work consists of developing a progressive web application designed to track generic routes, thus offering the possibility that in the future, when the technology is developed that allows a web application to calculate the user's position offline, it will be possible to implement it in the app.

The development of the application will be based on the MEAN stack, the technology on the client side that will be used as a web development framework will be Angular, while on the server side, Express and NodeJS will be used to build the server web application, in addition to the use of MongoDB as a non-relational database.

Keywords: *Tracking, Geolocation, Webapp, Offline, API, GPS, Wi-Fi, Bluetooth, MEAN, Framework, Angular, Server, Framework, Express, NodeJS, MongoDB*

Índice general

Índice de Figuras

Índice de Tablas

Índice de listados

Capítulo 1

Introducción

En este apartado se introducen los motivos por los cuales se inició el presente proyecto, así como el estado actual de la temática sobre la que se va a trabajar, y la metodología empleada y objetivos que persigue el trabajo.

1.1. Motivación y propósito del trabajo

Desde que empecé a estudiar en Tenerife, los viajes en avión y trayectos en guagua y tranvía son mucho más frecuentes, es por ello que me empecé a preguntar si sería posible registrar las rutas que realizo entre mi isla de residencia y la isla en la que me encuentro desarrollando mis estudios, así como las rutas que realizo en la propia isla, y poder compararlas entre sí y analizarlas.

Investigando, resulta que existen varios sistemas que son capaces de realizar un seguimiento del tráfico aéreo en tiempo real, además es posible encontrarse con aplicaciones de senderismo o rutas de viajes que permiten calcular los datos de la posición del usuario.

Las aplicaciones de seguimiento de rutas aéreas ofrecen sus servicios en la web, como una aplicación web, sin embargo, muchas de ellas se pueden instalar de forma nativa, al igual que las apps de senderismo y rutas de viajes.

Al probar las aplicaciones relacionadas con los viajes en avión, me di cuenta de que no podía usar ninguna sin que dependiera de la conectividad de red, es por ello que me planteé la posibilidad de desarrollar una aplicación que cumpliera con la funcionalidad de emplearse sin conexión.

La problemática surge en la tecnología web que se usa para calcular la geolocalización, ya que depende de las conexiones del dispositivo, tales como el Wi-Fi, Bluetooth y redes móviles. Es posible geolocalizar solo con el GPS, pero en el estándar W3C [1], de donde proviene la API que se está empleando, todavía no está implementado. Dicha implementación se está retrasando por cuestiones de privacidad [2].

Por lo tanto, se descarta que el sistema que se pretende desarrollar sea únicamente para rutas aéreas, así, el alcance de la aplicación se limita a la conectividad del dispositivo.

En definitiva, el propósito de este proyecto consiste en desarrollar una aplicación web progresiva, que realice un seguimiento de las rutas de viajes que el usuario incluya, de forma que en tiempo real se visualice en un mapa interactivo en la propia aplicación, y poder consultarse posteriormente.

En el marco del desarrollo de la aplicación, ésta se orientará hacia dispositivos móviles, es por ello que se adoptará un enfoque *#mobilefirst* en cuanto al diseño de la aplicación.

1.2. Antecedentes y estado actual del tema

Desde sus inicios el ser humano ha viajado a lo largo del mundo, a través de diferentes medios de transporte, evolucionando desde los viajes a pie, hasta los trayectos en vehículos como el automóvil o el avión.

Sin embargo, hoy en día resulta muy sencillo y cotidiano comprar un billete de avión, barco, tren, entre otros, y poder viajar a casi cualquier lugar del mundo, todo ello gracias a los avances tecnológicos que se han producido en estos últimos años.

Actualmente existen varias aplicaciones de seguimiento de rutas de viajes. Varios ejemplos pueden ser Flightradar24 [3], MarineTraffic [4] o Strava [5].

Flightradar24 es una aplicación de seguimiento de rutas aéreas que permite realizar un seguimiento en tiempo real de la trayectoria de los vuelos, ver las estadísticas actuales de demoras, obtener una lista completa de todos los vuelos en el aire en una ubicación determinada. Incluso es posible verificar hasta 365 días del historial de vuelo y conocer detalles del avión.

MarineTraffic es una plataforma que proporciona información sobre la ubicación de los barcos que se encuentran navegando o en puerto. Entre sus principales funcionalidades destacan el acceso a información sobre barcos, seguimiento en tiempo real de la posición del navío, detalles sobre su velocidad, rumbo y destino, así como el poder visualizar las rutas y un histórico de los trayectos del barco.

Strava es una aplicación web y móvil que está orientada al senderismo y al registro de actividades físicas y al aire libre, con ella es posible visualizar el estado de la ruta tomada por el usuario en tiempo real en un mapa, y poder registrar dichas rutas en un dispositivo móvil, o en dispositivos compatibles como relojes inteligentes. También se graban datos como la velocidad, el ritmo o la altitud durante el trayecto, además del componente social de visualizar las rutas de otros usuarios y poder compartir las propias.

Uno de los elementos en los que se basan la mayoría de estas aplicaciones, es en el uso de mapas, ya sean interactivos, dinámicos o estáticos, pero que permiten, en esencia, visualizar de manera clara la trayectoria de un viaje determinado. Es en este punto donde nos planteamos qué tecnologías permiten implementar o construir mapas de la manera más sencilla y eficaz posible.

Leaflet [6] es una librería de JavaScript que sirve para crear y manipular mapas en una aplicación web. Lo importante a destacar de esta librería es que se integra especialmente bien con aplicaciones *#mobilefirst* [7], justo lo que buscamos en la aplicación. Con Leaflet es posible crear mapas que soporten la ampliación o reducción del mapa, así como una vista panorámica del mismo.

De cara al desarrollo de la aplicación, además de realizar un seguimiento de la posición del usuario, sería interesante obtener y mostrar datos, como la velocidad actual, la altitud media, o la orientación del dispositivo. Es posible calcular estos datos gracias a la API de geolocalización web del estándar W3C [1] (*World Wide Web Consoritum*), proporcionada por los navegadores web modernos para obtener la ubicación geográfica del usuario a través del navegador.

Entre la disyuntiva habitual que existe entre utilizar aplicaciones nativas o aplicaciones web, encontramos una problemática en el acceso a funciones que en principio solo son accesibles por las primeras, pero que en las actuales *webapps* [8] han ido evolucionando. Entre estas características hemos de destacar las siguientes:

- Acceso GPS.
- Carga de archivos desde un móvil o tablet.
- Uso del giroscopio o acelerómetro.
- Empleo de navegación deslizante.
- Funcionamiento sin conexión.

Gracias a estos avances, es posible instalar una aplicación web progresiva, y hacer que esté disponible en la pantalla de inicio y en el menú de aplicaciones como si de una aplicación nativa se tratase, esto se consigue gracias al empleo de los *service workers*.

Los *service workers* [9] son *scripts* en segundo plano que se ejecutan de forma independiente del hilo principal de la aplicación, y permiten realizar varias de las tareas mencionadas anteriormente.

Una de las líneas de trabajo consiste en analizar estas funcionalidades y valorar si es factible utilizarlas en el desarrollo del proyecto.

1.3. Objetivos

El objetivo de este proyecto consiste en desarrollar una aplicación web progresiva que sea capaz de hacer un seguimiento de las rutas de viajes que el usuario desee introducir, para ello se aprovecharán las nuevas tecnologías y herramientas que permitan construir una aplicación web de forma escalable, segura, y con un diseño responsivo, que facilite la apariencia de la aplicación en distintos dispositivos.

Entre las funcionalidades principales que posee la aplicación, se encuentra el poder seguir la ruta que ha introducido el usuario en tiempo real, así como registrar la misma para poder acceder a ella posteriormente. También se podrá consultar las rutas que otros usuarios hayan introducido, es decir, que se podrán compartir para la vista de otros usuarios.

1.4. Metodología

Las principales actividades a realizar para el desarrollo del proyecto son las siguientes:

- Planificación.
- Prototipado de la aplicación.
- Setup de la infraestructura del proyecto.
- Diseño de la API.
- Desarrollo del Backend.
- Setup del Frontend.
- Desarrollo del Frontend.
- Integración continua y calidad de código.
- Despliegue continuo.

A pesar de que inicialmente para cada tarea en el desarrollo del trabajo se planificó con una fecha y duración establecidas, lo cierto es que no se cumplió con la planificación establecida puesto que se comenzó por la infraestructura del proyecto y, posteriormente, en toda la configuración necesaria, para después pasar a implementar de forma paralela el *Frontend* y *Backend* de la aplicación.

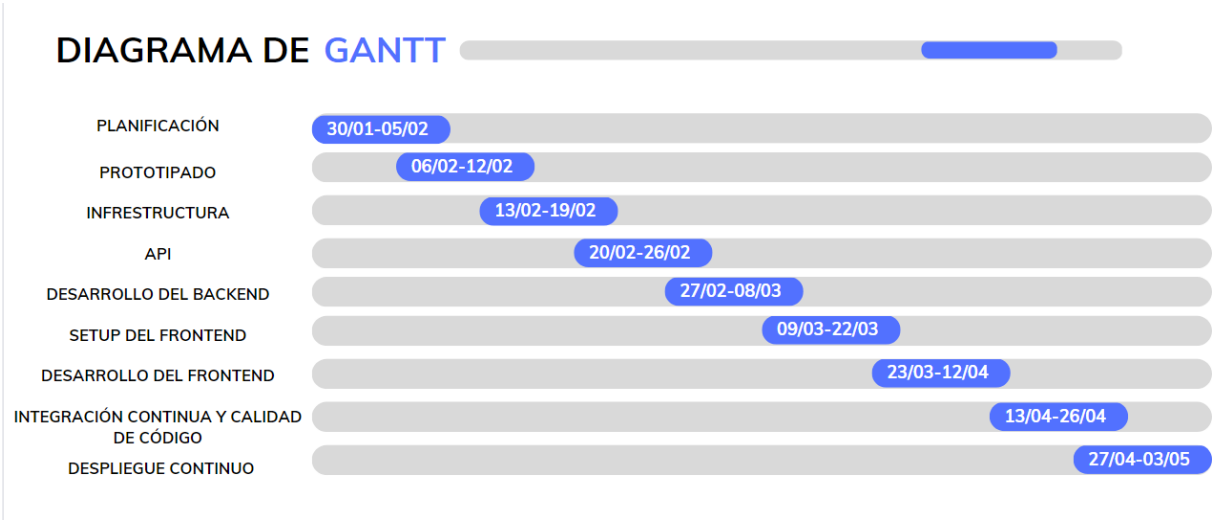


Figura 1.1: Planificación de las tareas a realizar en el proyecto

Capítulo 2

Tecnologías y herramientas

En este apartado se describen las tecnologías y herramientas que se han empleado para el desarrollo del proyecto.

2.1. Entorno de desarrollo

En este subapartado se detallan aquellas herramientas que han facilitado el desarrollo del proyecto que, aún no habiendo sido partícipes de forma activa en el desarrollo del código fuente del programa, sí que han resultado útiles para el desarrollo del proyecto.

Las tecnologías que se incluyen guardan relación con el editor de código, la integración continua y calidad de código, el despliegue de la aplicación y el prototipado de la misma.

2.1.1. Visual Studio Code

Visual Studio Code [10] es un editor de código desarrollado por la empresa Microsoft, disponible para Windows, Linux, macOS y navegadores web. En este caso, y para el desarrollo del proyecto, se ha elegido como sistema operativo Windows, ya que las herramientas relacionadas con el *frontend* y *backend* se integran correctamente con este sistema.

Se ha elegido Visual Studio como entorno de desarrollo, principalmente por la facilidad de uso, y las extensiones que ofrece, así como de las herramientas que vienen incluidas, como la consola de comandos integrada.

2.1.2. Github

Github [11] es una plataforma que permite alojar proyectos y realizar un seguimiento de los mismos gracias al sistema de control de versiones Git. Además

de guardar todos los datos del proyecto, Github ofrece herramientas relacionadas con la integración continua de la aplicación.

Entre estas herramientas destaca Github Actions [12], un servicio que permite automatizar las tareas y los flujos de trabajo. Resulta de mucha utilidad ya que facilita el trabajo y ahorra tiempo.

2.1.3. Heroku

Heroku [13] es una plataforma que se usa para desplegar aplicaciones de servidor, así como poder ejecutarlas en la nube, permitiendo al usuario centrarse únicamente en el desarrollo de la aplicación, mientras Heroku se encarga de la infraestructura que hay detrás del proyecto.

2.1.4. Vercel

Vercel [14] es un servicio de alojamiento y despliegue de aplicaciones web, que permite realizar una configuración rápida en la puesta a punto de la aplicación. Una de sus principales características es su enfoque en la implementación de aplicaciones web estáticas, y se integra especialmente bien con Github.

2.1.5. Sonarcloud

Sonarcloud [15] es una plataforma en la nube que sirve como herramienta de análisis y calidad de código, ya que proporciona datos relacionados con la fiabilidad, el mantenimiento y la seguridad del código, como errores en el código fuente, vulnerabilidades, o incluso duplicaciones de código, y pautas para un mejor desarrollo del proyecto.

También se pueden visualizar estadísticas en el cubrimiento de código, y se integra con Github Actions.

2.1.6. ESLint

ESLint [16] es un *linter* para JavaScript y TypeScript que analiza el código fuente en busca de errores de sintaxis, y permite adoptar un estilo de código para ajustarse a alguno de los estándares empleados, consiguiendo uniformidad en el código. Se basa en el uso de reglas y se puede integrar con los editores de código más empleados.

2.2. Backend

En este apartado se explican las herramientas relacionadas con el *backend*, incluyendo el entorno de ejecución del servidor, el *framework* usado, el conjunto

de pruebas, el estándar de gestión de la sesión y la base de datos usada.

2.2.1. NodeJS

NodeJS [17] es un entorno de tiempo de ejecución para el marco de aplicaciones de servidor, basado en el lenguaje de programación JavaScript, asíncrono, con una arquitectura orientada a eventos y no bloqueante. Es ideal para manejar aplicaciones de alto rendimiento y escalables.

Una de sus principales características es que cuenta con un gestor de módulos y paquetes, llamado *npm* [18], lo que permite a los desarrolladores acceder a una amplia selección de funcionalidades que facilita el desarrollo de las aplicaciones web.

2.2.2. Express

Express [19] es un framework de desarrollo web de NodeJS que se emplea para construir aplicaciones web y APIS de una forma sencilla. Entre sus principales características destacan el manejo de rutas, las respuestas *http*, la gestión de *middelware* o el manejo de peticiones.

Se ha elegido este *framework* debido principalmente a su flexibilidad y su simplicidad a la hora de crear aplicaciones web.

2.2.3. Insomnia

Insomnia [20] es una aplicación que permite hacer peticiones y consumir una API, funcionando como una aplicación cliente, para comprobar las respuestas que recibe del servidor y poder así ejecutar pruebas.

2.2.4. Mocha y Chai

Mocha [21] es un framework de pruebas diseñado para escribir y ejecutar pruebas en un navegador y una aplicación Express. Permite varios modos de pruebas, como *BDD* [22] (Behavior-Driven Development) o *TDD* [23] (Test-Driven Development).

Chai [24] es una biblioteca de aserciones de JavScript que se emplea con Mocha, y en la que se pueden escribir distintos tipos de aserciones, como *should*, *expect* y *assert*, lo que permite flexibilidad a la hora de escribir los test unitarios.

2.2.5. JWT

JWT [25] (*JSON Web Token*) es un estándar utilizado para el intercambio de información en forma de objetos JSON, con el fin de proporcionar un mecanismo

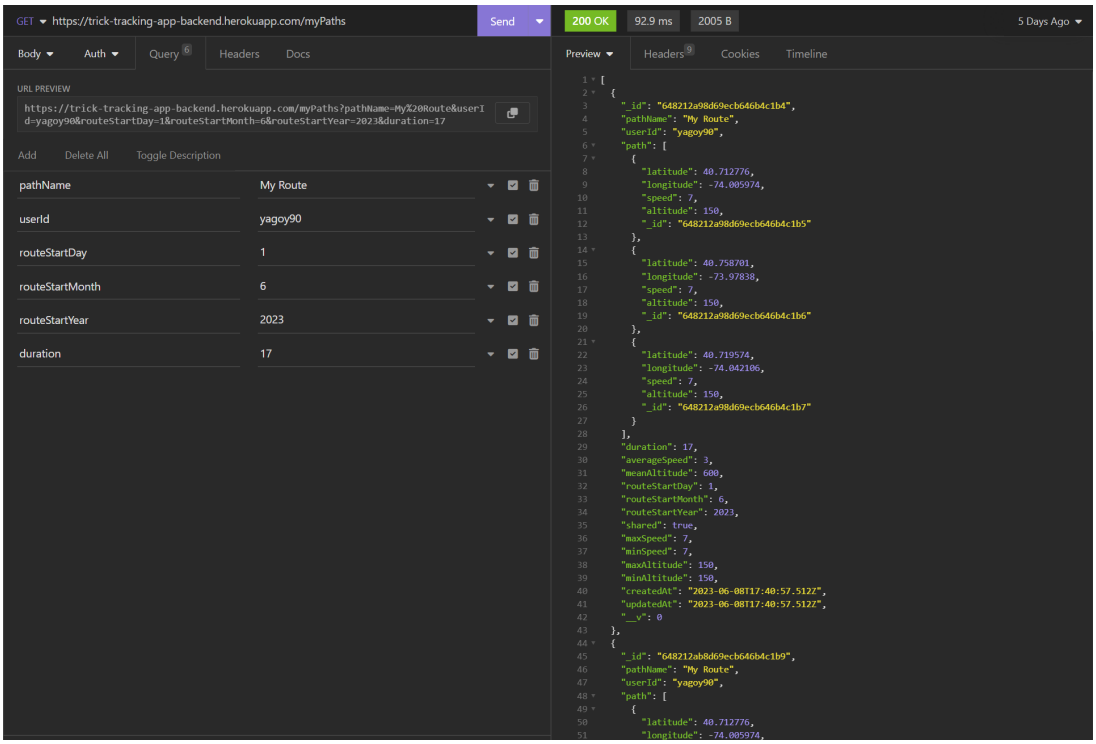


Figura 2.1: Ejemplo de petición HTTP en una colección de Insomnia

de autorización y autenticación de usuarios en aplicaciones web y APIS.

2.2.6. MongoDB

MongoDB [26] es una base de datos no relacional, ya que en lugar de guardar los datos en tablas, como se haría en una base de datos relacional, MongoDB guarda dichos datos en documentos en un formato similar a los objetos JSON, lo interesante a destacar es que no se exige una estructura rígida para guardar las entradas de una colección.

MongoDB Atlas [27] es una plataforma en la nube que ofrece una base de datos como servicio, dando la posibilidad a los desarrolladores de crear una instancia de MongoDB en la nube, y poder administrar y acceder a ella fácilmente.

2.3. Frontend

En este apartado se explican aquellas tecnologías y herramientas relacionadas con la interfaz de usuario, en concreto, los *frameworks* empleados, el marco de pruebas empleado, la conexión con el *backend*, las herramientas relacionadas con las aplicaciones web progresivas y la integración de mapas.

2.3.1. Angular

Angular [28] es un *framework* de desarrollo web creado por Google, diseñado para crear páginas web estáticas (SPA) [29] y aplicaciones móviles híbridas. Se basa en el lenguaje de programación TypeScript, que agrega características, desde el tipado de datos, al lenguaje de programación JavaScript.

Angular utiliza una arquitectura basada en componentes, lo que hace que la aplicación se divida en una serie de componentes reutilizables que se comunican entre sí por medio de eventos. Cada componente se basa en una plantilla HTML, una hoja de estilos CSS y una clase en TypeScript.

2.3.2. Angular Routing

Angular Routing [30] es un módulo y una de las funcionalidades más importantes en Angular, puesto que es el que permite navegar entre las distintas páginas de la aplicación.

Se basa en el enrutamiento por parte del cliente, lo cual quiere decir que el cambio entre páginas ocurre dentro del navegador. Este enrutamiento se basa en una configuración centralizada en un archivo llamado *app-routing.module.ts*.

2.3.3. Angular CLI

Angular CLI [31] es la interfaz de línea de comandos para ejecutar comandos de Angular, como crear un componente o servicio de Angular, o ejecutar las pruebas.

2.3.4. Angular HTTP Client Module

El módulo HTTP [32] de Angular es una de las partes esenciales del *framework*, ya que permite la comunicación del cliente con el servidor, por medio de peticiones *HTTP*, el manejo de respuestas y la manipulación de datos JSON.

En esencia este módulo pretende mejorar la comunicación con servicios web.

2.3.5. Jasmine y Karma

Jasmine [33] y Karma [34] son dos de las herramientas más populares a la hora de codificar y ejecutar pruebas en una aplicación Angular.

Jasmine es un *framework* de pruebas unitarias para el lenguaje JavaScript, en el contexto de Angular, se usa fundamentalmente para probar el comportamiento de los componentes de la aplicación, construyendo pruebas unitarias.

Karma es un entorno de ejecución de pruebas para JavaScript, aportando lo

necesario para poner en marcha las pruebas escritas en Jasmine. Además, permite la ejecución continua de pruebas mientras se desarrolla el código.

2.3.6. Bootstrap

Bootstrap [35] es un *framework* de desarrollo web que facilita la creación de interfaces de usuario y la aplicación de estilos, puesto que ofrece un conjunto de plantillas predefinidas que facilitan el trabajo.

2.3.7. Service Workers

Los *service workers* son pequeños scripts de JavaScript que se ejecutan en el navegador web, estos son independientes de la aplicación cliente que se esté ejecutando, lo que hace posible agregar ciertas características relacionadas con las aplicaciones web progresivas.

Entre las funcionalidades más importantes a destacar, cabe mencionar que gracias a la ejecución de un *service worker*, la aplicación cliente de Angular puede instalarse en un dispositivo móvil, además otro elemento relevante es el almacenamiento en caché de los datos.

2.3.8. Openstreetmap

OpenStreetMap [36] es una plataforma de mapas libre proveedora de mapas, que sirve para obtener datos geográficos para su libre distribución, siendo su propósito el de ofrecer una fuente de datos fiable para su uso en servicios y aplicaciones de mapas.

2.3.9. Leaflet

Leaflet es una librería de código abierto de JavScript destinada al manejo de mapas. Proporciona métodos y funciones para mostrar mapas y manipularlos, tales como realizar *zoom* sobre el mapa y navegar sobre el mismo, agregar marcadores, polígonos o marcar rutas sobre el mapa, entre otros.

Capítulo 3

Desarrollo de la aplicación

En este apartado se detallan todos aquellos aspectos relacionados con el desarrollo de la aplicación, tales como la configuración inicial del proyecto y el entorno de desarrollo, el modelo de datos utilizado, la construcción del *frontend* y del *backend*, y las herramientas empleadas para el despliegue continuo.

3.1. Configuración inicial del proyecto

Antes de empezar a codificar la aplicación, es necesario realizar ciertas configuraciones con el fin de facilitar el desarrollo de la aplicación.

En primer lugar, se crearon dos repositorios, uno que aloja la aplicación cliente [37], y otro que aloja el servidor [38]. En cada repositorio se llevan a cabo las configuraciones relacionadas con el despliegue de la aplicación y la integración de la misma.

Posteriormente, se originó un proyecto en TypeScript en donde se creó un servidor básico en Express, mientras que en el otro repositorio se generó una aplicación Angular sencilla con el intérprete de comandos de Angular.

Más adelante, se pasó a crear los flujos de trabajo, o *workflows*, de las Github Actions, para formalizar el proceso de integración continua para ambas implementaciones, tanto el *frontend*, como el *backend*.

```
1 name: Node.js CI
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
```

```yaml
jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [14.x, 16.x, 18.x]
        # See supported Node.js release schedule at https://
           nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v3
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm install
    - run: npm run test
```

Listado 3.1: Acción de integración continua del servidor

Con ambos *workflows*, se permite que cada vez que se envíen cambios a la rama principal se ejecuten las pruebas en diversas versiones de node, para comprobar si son un éxito en distintos entornos de ejecución, además, se genera una tabla con el porcentaje de cobertura de código. Cabe destacar que tenemos dos flujos de trabajo relacionados con la ejecución de pruebas, uno para el *frontend* y otro para el *backend*.

Posteriormente se configuró ESLint como *linter*(el preprocesador de formato establecido) de ambas implementaciones, y se habilitó SonarCloud en el repositorio en el que se aloja el *frontend*, y se comenzó a configurar el despliegue continuo una vez que se tenía un esqueleto de código básico para el cliente y el servidor.

Heroku se utilizó para desplegar el servidor, mientras que para la aplicación cliente se usó Vercel. Heroku ofrece la posibilidad de desplegar la aplicación de forma automática, en su propia interfaz, sin embargo, con Vercel si que es necesario configurar un *workflow* para desplegar la aplicación.

```yaml
# Simple workflow for deploying static content to GitHub Pages
name: Deploy to vercel

on:
  # Runs on pushes targeting the default branch
```

```yaml
  push:
    branches: ["main"]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# Sets permissions of the GITHUB_TOKEN to allow deployment to GitHub Pages
permissions:
  contents: read
  pages: write
  id-token: write

# Allow only one concurrent deployment, skipping runs queued between the run in-progress and latest queued.
# However, do NOT cancel in-progress runs as we want to allow these production deployments to complete.
concurrency:
  group: "pages"
  cancel-in-progress: false

jobs:
  # Single deploy job since we're just deploying
  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Setup Pages
        uses: actions/configure-pages@v3
      - name: Upload artifact
        uses: actions/upload-pages-artifact@v1
        with:
          # Upload entire repository
          path: '.'
      - name: Deploy to GitHub Pages
        id: deployment
        uses: actions/deploy-pages@v2
```

Listado 3.2: Acción de despliegue continuo de Vercel

Además de todo ello, se realizó el proceso de creación de un *clúster* en MongoDB Atlas para guardar los datos generados por la aplicación.

3.2. API y modelo de datos

Antes de empezar a codificar la lógica de la aplicación, también es importante formalizar los datos con los que se va trabajar.

Teniendo en cuenta las funcionalidades requeridas por la aplicación en anteriores apartados, se consideró que lo adecuado sería disponer de dos modelos de datos, uno para los usuarios que empleen la aplicación, y otro para las rutas que registraran dichos usuarios a la hora de usar la aplicación.

Estos datos serían accesibles desde los *endpoints* de la API diseñados para manejar las solicitudes que reciba el servidor desde la aplicación cliente.

A continuación se muestra le estructura de los endpoints comentados previamente:

	Tipo	Ruta	Descripción
	GET	/	Ruta inicial
User	POST	/signUp	Registro de usuario
User	POST	/signIn	Inicio de sesión
Path	POST	/paths	Guardar una ruta
Path	GET	/myPaths	Obtiene las rutas privadas de un usuario
Path	GET	/social	Obtiene las rutas públicas.
Path	DELETE	/paths/:id	Elimina una ruta privada.
Path	PUT	/paths/id	Actualiza los datos de una ruta privada

Figura 3.1: Endpoints de la API diseñada

Hacemos especial mención al modelo de datos diseñado para guardar las rutas que los usuarios registren, ya que presenta numerosas propiedades que merecen la pena comentar:

```
1 /* eslint-disable linebreak-style */
2 const mongoose = require('mongoose');
```

```
3
4 const pathSchema = new mongoose.Schema({
5   pathName: {
6     type: String,
7     required: true,
8   },
9   userId: {
10    type: String,
11    required: true,
12  },
13  path: {
14    type: [
15      {
16        latitude: {type: Number, required: true},
17        longitude: {type: Number, required: true},
18        speed: {type: Number, required: true},
19        altitude: {type: Number, required: true},
20      },
21    ],
22    required: true,
23  },
24  duration: {
25    type: Number,
26    required: true,
27    validate: {
28      validator: Number.isInteger,
29      message: '{VALUE} no es un número entero',
30    },
31  },
32  averageSpeed: {type: Number, required: true, default: 0},
33  meanAltitude: {type: Number, required: true},
34  routeStartDay: {
35    type: Number,
36    required: true,
37    validate: {
38      validator: Number.isInteger,
39      message: '{VALUE} no es un número entero',
40    },
41  },
42  routeStartMonth: {
43    type: Number,
44    required: true,
45    validate: {
46      validator: Number.isInteger,
47      message: '{VALUE} no es un número entero',
48    },
49  },
```

```
50   routeStartYear: {
51     type: Number,
52     required: true,
53     validate: {
54       validator: Number.isInteger,
55       message: '{VALUE} no es un número entero',
56     },
57   },
58   shared: {type: Boolean, required: true, default: false},
59   maxSpeed: {type: Number, required: true},
60   minSpeed: {type: Number, required: true},
61   maxAltitude: {type: Number, required: true},
62   minAltitude: {type: Number, required: true},
63 }, {
64   timestamps: true,
65 });
66
67 module.exports = mongoose.model('Path', pathSchema);
```

Listado 3.3: Modelo de datos para las rutas

En el modelo de datos de las rutas, cabe destacar que el *id* del usuario corresponde al nombre de usuario que ha iniciado sesión, y es único.

Por otro lado, existen algunas propiedades, como la velocidad media, o la altura máxima, que dependen de la propiedad *path*, quizás la más importante, puesto que consiste en un *array* o vector de rutas en las que se guardan las coordenadas, así como la velocidad y la altitud cada vez que la aplicación calcula los datos de geolocalización.

3.3. Servidor

A continuación se explican los pasos que se han seguido para desarrollar la aplicación del servidor, el diseño de la API, la gestión de la sesión del usuario, y el marco de pruebas empleado.

3.3.1. Arquitectura de la aplicación

En primer lugar merece destacar un archivo de especial interés que, aunque no guarda relación directa con el patrón de diseño de la aplicación, sirve para establecer la conexión con la base de datos.

A continuación, se muestra la función que establece la conexión con la base de datos remota creada con MongoDB Atlas, dicha función retorna una promesa:

```
mongoose.connect('mongodb+srv://',
    {useNewUrlParser: true,
      useUnifiedTopology: true,
    })
    .then((db) => console.log('Database is connected'))
    .catch((err) => console.log(err));
```

Listado 3.4: Función que conecta la base de datos de MongoDB

La arquitectura de la aplicación se basa en el modelo de vista-controlador [39].

Cuando se comentaba en el apartado anterior los modelos usados para el manejo de datos en el servidor, estos hacen referencia a los modelos, es decir, a la estructura de los datos, en el patrón MVC.

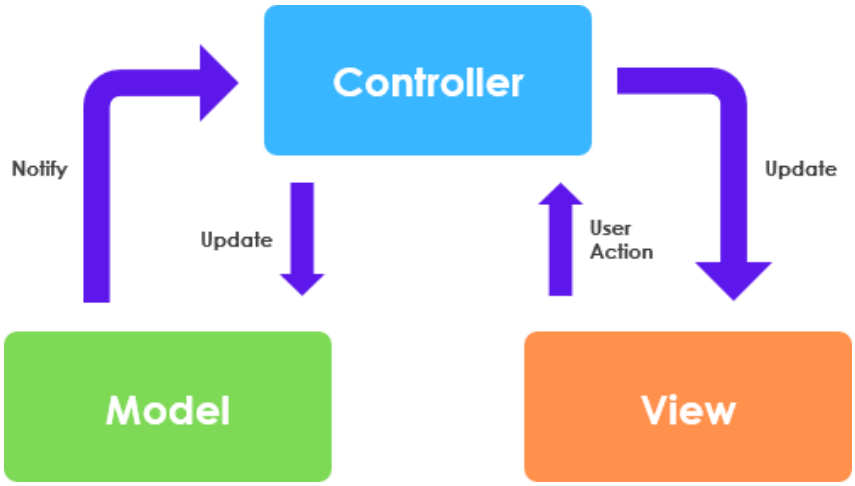


Figura 3.2: Representación del modelo Vista-Controlador

Mientras que, por otro lado, en el archivo *server.js*, el archivo principal de la aplicación, representa al controlador de la arquitectura MVC, el cual maneja las solicitudes y respuestas *HTTP* procesando los datos e interactuando con los modelos según sea necesario.

En las siguientes líneas se muestran las funciones encargadas del manejo de peticiones *HTTP* relacionadas con el inicio de sesión de usuario y el guardado de rutas:

```
app.post('/signIn', async (req, res) => {
  const {name, password} = req.body;
  const user = await User.findOne({name});
  if (!user) {
    return res.status(401).send('El usuario que ha introducido no existe');
  };
  if (user.password !== password) {
    return res.status(401).send('Contraseña incorrecta');
```

```
  }

  const token = jwt.sign({id_: user.id}, 'secretkey');
  res.status(200).json({token: token});
});
```

Listado 3.5: Controlador para el inicio de sesión

```
app.post('/paths', async (req, res) => {
  const {pathName, userId, path, duration, averageSpeed...} = req.body;
  const newPath = new Path({pathName, userId, path, duration, averageSpeed...});

  try {
    await newPath.save();
    res.status(200).json(newPath);
  } catch (err) {
    if (newPath.pathName == '') {
      return res.status(401).send('Debe aportar un nombre a la ruta');
    }

    if (newPath.userId == '') {
      return res.status(401).send('Debe iniciar sesión antes de...');
    }
    return res.status(500).json({error: err.message});
  }
});
```

Listado 3.6: Controlador para el guardado de rutas

En el controlador de inicio de sesión del usuario, se recogen los datos recibidos del cuerpo de la petición, para posteriormente comprobar si el nombre de usuario existe en la base de datos.

En el caso de que sea así, se comprueba la contraseña, y si es correcta, se envía un *token*, que se explicará en el siguiente apartado, con un código de estado 200 indicando que la operación ha sido un éxito.

En el controlador que guarda las rutas, que también se trata de un *post*, hacia otra ruta *(/paths)*, también se recogen los datos del cuerpo de la petición, que contienen a la ruta que el usuario desea guardar, la cual se guarda en la base de datos de MongoDB.

3.3.2. JSON Web Token

Como se comentaba previamente, para el control de la sesión de usuario se utiliza lo que se conoce como JSON Web Token, que se basa en el envío de un *token* por parte del servidor cuando un usuario se registra o inicia sesión.

El objetivo es que cuando el usuario tenga que acceder a un recurso en el cual se necesite autenticación, éste tenga que enviar el *token* a través de las cabeceras de la petición hacia el servidor.

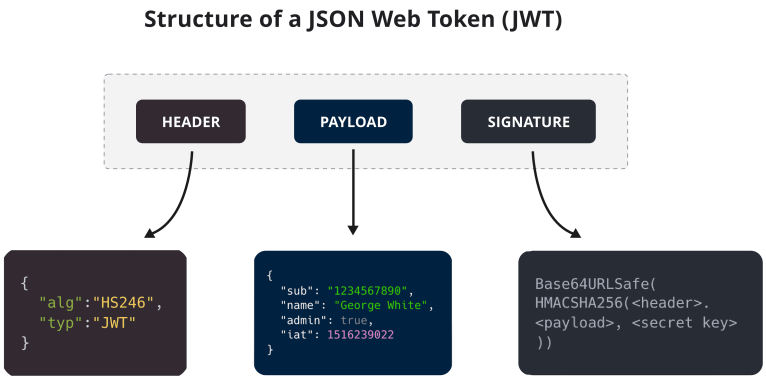


Figura 3.3: Estructura de los JSON Web Token

Como se aprecia en la imagen anterior, un *token* JWT se caracteriza por estar estructurado en tres partes: el encabezado, que contiene información sobre el tipo de *token* y el algoritmo de firma usados; una carga útil o *payload*, que contiene los datos codificados que se transmiten de un lado a otro; y la firma, que se utiliza para verificar la integridad del *token*, para asegurar que se no se ha modificado.

En la aplicación Angular que se ha desarrollado se guarda este *token* en el almacenamiento local del navegador, cuando dicho *token* se recibe del servidor.

3.3.3. Testing con Mocha y Chai

Como se comentaba en apartados anteriores, se ha elegido Mocha como *framework* de desarrollo de pruebas, ya que es uno de los más empleados en entornos de desarrollo de *backend*, mientras que Chai se usa como librería para escribir las aserciones de los test unitarios.

Principalmente los test que se han desarrollado han sido test unitarios, estos se han codificado en un fichero llamado *test.js*.

A continuación se muestran algunos test:

```
1 it('POST /signIn debería autenticar al usuario y retornar un token', (done) => {
2 chai
3     .request(app)
4     .post('/signIn')
5     .send({name: 'testuser', password: 'password'})
6     .end((err, res) => {
7       expect(res).to.have.status(200);
8       expect(res.body).to.have.property('token');
```

```
9         done();
10      });
11  });
```

Listado 3.7: Extracto de prueba unitaria para el inicio de sesión

```
1  it('GET /myPaths debería retornar los caminos que coinciden...', (done) => {
2      chai
3          .request(app)
4          .get('/myPaths')
5          .query({userId: 'id_de_usuario'})
6          .end((err, res) => {
7            expect(res).to.have.status(200);
8            expect(res.body).to.be.an('array');
9            done();
10         });
11   });
```

Listado 3.8: Extracto de prueba unitaria para la obtención de rutas de un usuario

Por otro lado, también se realizan pruebas de la API, con la aplicación Insomnia, con el objetivo de probar el correcto funcionamiento de los *endpoints* del servidor y verificar que las respuestas de la API son adecuadas.

3.4. Aplicación Cliente

En este apartado se describen los pasos que se han seguido para el desarrollo de la aplicación cliente, desde la estructura de la aplicación, hasta la integración incluyendo los mapas y los *service workers*.

La aplicación cliente se ha desarrollado con el *framework* de desarrollo web Angular, siendo este el eje principal en el desarrollo del *frontend*.

3.4.1. Estructura de la aplicación

La estructura de la aplicación sigue el diseño clásico de una aplicación Angular, con los siguientes elementos:

- Directorio raíz: en el directorio raíz de la aplicación se encuentran principalmente archivos de configuración, como *package.json*, para la gestión de las dependencias del proyecto, *angular.json*, relacionada con la configuración de Angular, o *tsconfig.json*, para las opciones del compilador de TypeScript.
- Directorio src/app: aquí se encuentra el código fuente de la aplicación, dividida en varios elementos:

- Components: el directorio que contiene los componentes de la aplicación. Cada componente tiene su archivo de código fuente TypeScript, su hoja de estilos CSS, su archivo HTML, y su archivo de pruebas .spec.ts.
- Services: el directorio que contiene los servicios de la aplicación. Por cada servicio se encuentra un archivo de código fuente de TypeScript, y un archivo de pruebas .spec.ts.
- app-routing-module.ts: el archivo que define las rutas y la navegación entre los distintos componentes.
- app.module.ts: el módulo principal de la aplicación. Sirve para importar y configurar los componentes, servicios y otros módulos de la aplicación.

- Directorio src/assets: aquí se almacenan todos aquellos recursos estáticos de la aplicación ,como imágenes o archivos de configuración.
- Otros archivos: también se encuentran otros archivos dentro del directorio *src* como el *index.html* y la hoja de estilos globales *styles.css*.
- Directorio node-modules: en este directorio se encuentran las dependencias instaladas a través de *npm*.

3.4.2. Service Workers

Los *service workers* son una tecnología fundamental a la hora de permitir que una aplicación funcione como una aplicación nativa o como una aplicación web progresiva *(PWA)*. Se tratan de *scripts* en segundo plano que se ejecutan de forma independiente al hilo principal de JavaScript en el navegador.

Los *Service Workers* son cruciales para convertir una aplicación Angular en una aplicación web progresiva, ya que gracias a ellos, es posible instalar la aplicación agregándose a la pantalla de inicio, permitiendo un acceso rápido y sin conexión.

De esta forma la aplicación se puede cargar de forma sucesiva tras abrirse varias veces.

A continuación, se explica el proceso de configuración de los *Service Workers* en una aplicación Angular:

- ng add @angular/pwa: comando que añade las funcionalidades de los *service workers* a la aplicación Angular, añadiendo los archivos *ngsw-config.json* y *src/manifest.webmanifest*, y modificando los archivos *angular.json*, *package.json*, *src/app/app.module.ts* y *src/index.html*.

- ngsw-config.json: archivo de configuración del *Service Worker* que permite configurar la estrategia de caché, los archivos y otras configuraciones.

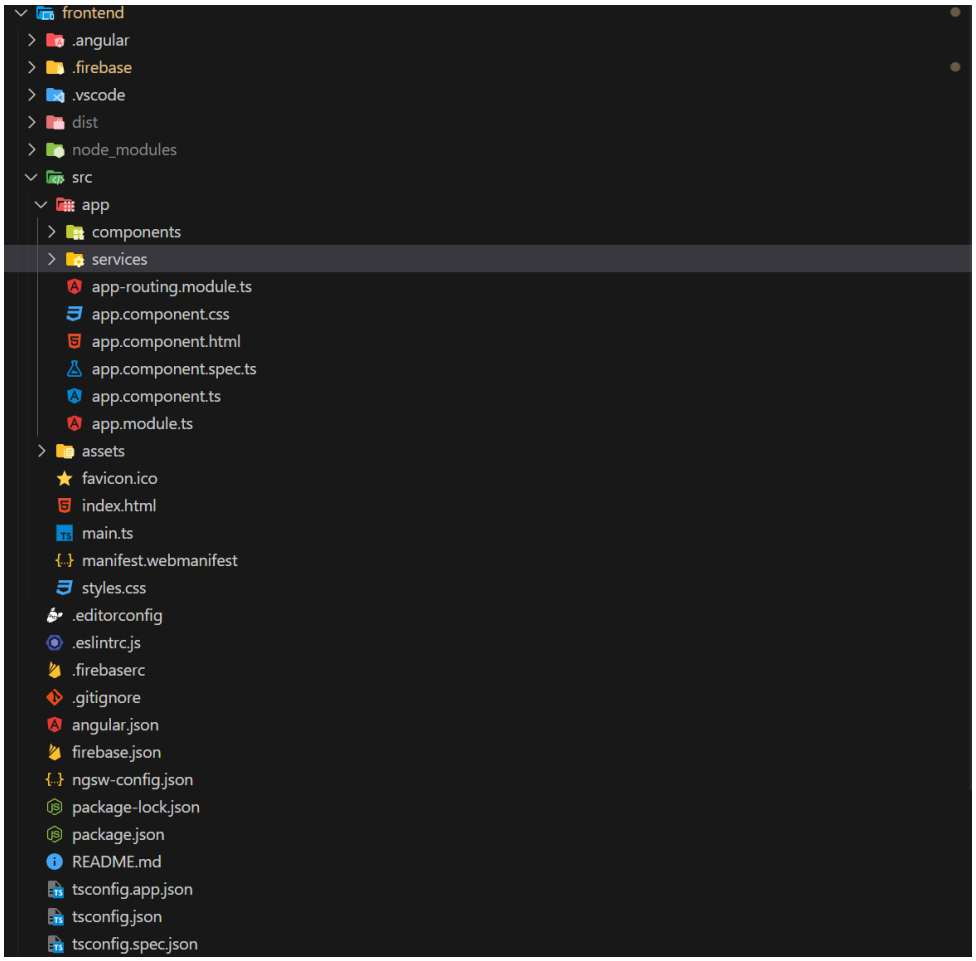


Figura 3.4: Estructura de la aplicación Angular

- src/manifest.webmanifest: archivo de configuración de la aplicación o archivo de manifiesto de la aplicación web que permite configurar el nombre, colores, iconos y otras configuraciones.
- angular.json: agrega la configuración del *Service Worker*.
- package.json: incorpora la biblioteca *@angular/service-worker*.
- src/app/app.module.ts: agrega la configuración del *Service Worker*.
- src/index.html: introduce la configuración del archivo de manifiesto y el color del tema.

3.4.3. Integración de mapas

En cuanto a la integración de mapas, se ha usado la librería Leaflet de JavaScript, disponible también en TypeScript, la cual hace posible ampliar o reducir la vista sobre el mapa, añadir marcadores, o crear polígonos y figuras sobre el mapa.

OpenStreetMap se ha elegido como fuente de datos para la integración con Leaflet, en concreto, el mapa predeterminado, *"Mapnik"*, que ofrece una repre-

sentación detallada de características geográficas como carreteras o edificios, así como datos relacionados con el relieve del lugar.

3.5. Despliegue

El despliegue de una aplicación es un proceso importante en el desarrollo de un proyecto de estas características, ya que permite preparar y llevar a producción todo el código desarrollado. En nuestro caso, se ha optado por desplegar el *backend* y el *frontend* en dos plataformas diferentes.

3.5.1. Despliegue del backend

Para el servidor se ha optado por usar Heroku como servicio de despliegue de aplicaciones web.

Se ha creado un proyecto y se ha conectado a Github para el despliegue continuo, sin necesidad de hacer uso de las Github Actions, ya que Heroku proporciona la funcionalidad de conectar aplicaciones a Github de forma sencilla en su propia interfaz de usuario.

Cada vez que se empujen cambios a la rama principal, Heroku detecta automáticamente dichos cambios y realiza las acciones necesarias para actualizar la aplicación desplegada, evitando la necesidad de hacer despliegues manuales.

La URL disponible para acceder al backend es la siguiente:

- `https://trick-tracking-app-backend.herokuapp.com/`

3.5.2. Despliegue del frontend

En cuanto al *frontend*, Angular proporciona herramientas para desplegar la aplicación en la máquina local con el comando *ng serve* de Angular CLI. Dicho comando crea un servidor temporal en la que el usuario puede acceder a la aplicación desplegada en local. Normalmente esta suele ser la dirección que usa Angular por defecto:

- http://localhost:4200/

Si no está disponible, Angular proporciona una dirección de forma automática.

Para el despliege en producción se necesitan tres acciones a realizar. Primero, realizar un *build* de la aplicación, o lo que es lo mismo, compilar la aplicación, ya que Angular usa TypeScript como lenguaje de programación, y no se puede ejecutar directamente en el navegador.

Esto se puede realizar con el comando *ng build* de Angular CLI, lo que hace es generar los archivos estáticos que se usarán en un entorno de producción, debido a que, a diferencia del *backend*, el *frontend* es una aplicación estática, no se encuentra constantemente ejecutándose.

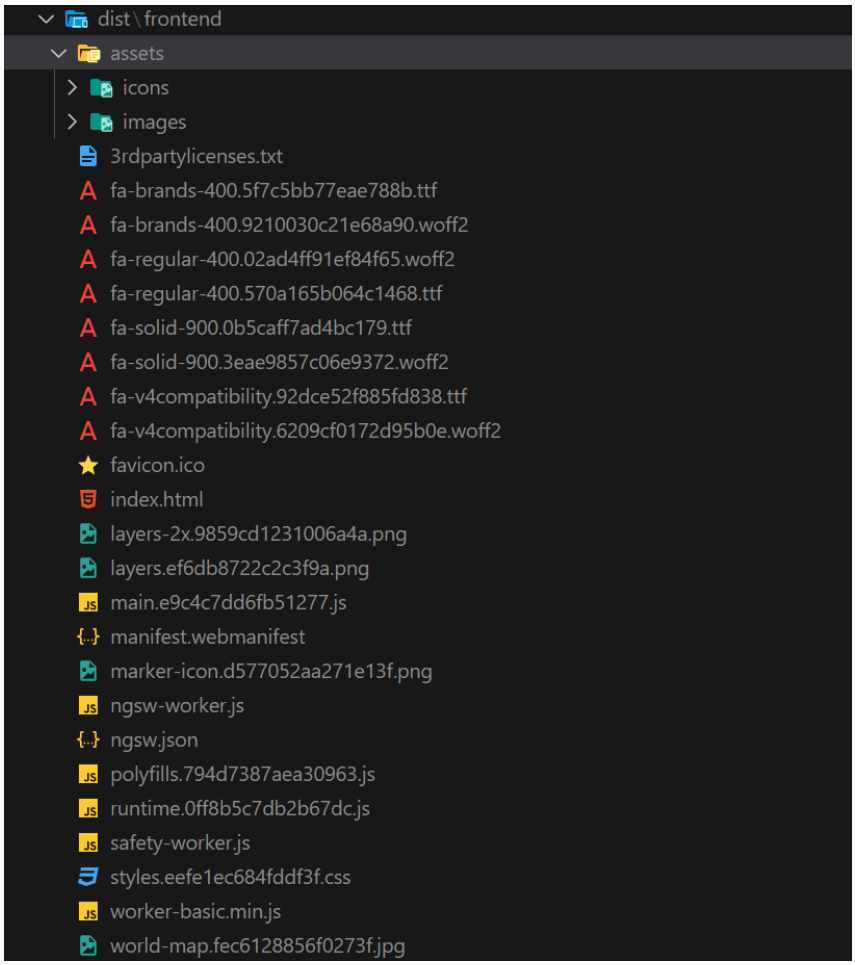


Figura 3.5: Resultado de la compilación de la aplicación Angular

El resultado que se obtiene es una serie de archivos, destacando los más importantes de ellos se encuentran la estructura de la página web HTML, los archivos de estilo CSS, y los archivos de código fuente de JavaScript, listos para ser entregados a un servidor web estático.

En esta línea, y como siguiente acción, se ha optado por usar Vercel como herramienta de despliegue de aplicaciones estáticas, proporcionando una forma sencilla de implementar rápidamente proyectos *frontend*.

Para iniciar el proceso, es necesario instalar la interfaz de comandos de vercel, como dependencia de desarrollo del proyecto, o como dependencia global. Una vez que se ha compilado la aplicacación, desde la raíz del proyecto de Angular se ejecuta el comando *vercel*, dando lugar al flujo de despliegue interactivo, configurándose el despliegue según las preguntas proporcionadas por Vercel.

Una vez finalizada la configuración, Vercel comenzará a implementar la aplica-

ción Angular y generará una URL pública en donde estará disponible la aplicación cliente para cualquiera que quiera usarla.

Para automatizar el proceso de despliegue, y como último paso, se ha creado un *workflow* de Github para hacer posible que cuando se hagan cambios en el repositorio, automáticamente se envíen dichos cambios a Vercel, y actualice la aplicación desplegada.

Las URL disponibles para acceder a la aplicación Angular son las siguientes (En el proyecto se han generado dos dominios personalizados):

- `https://tfg-2023-yago-perez-molanes-flying-track.vercel.app/`
- `https://tfg-2023-yago-perez-molanes-flying-track-ull-ing-inf.vercel.app/`

Capítulo 4

Descripción de la aplicación

En este apartado se explica el funcionamiento de la aplicación desde las pantallas que componen la interfaz de usuario, describiendo cada funcionalidad desarrollada en la aplicación.

Las aplicaciones web son capaces de ejecutarse en varios dispositivos, incluyendo móviles, tablets u ordenadores, es por ello que, en cuanto al diseño de la aplicación, se ha optado por seguir un diseño responsivo, en el que la apariencia de la aplicación se adapte a los distintos dispositivos que se use.

Además de este enfoque, y cómo Trip Tracking se va a usar mayoritariamente en dispositivos móviles, también se ha tenido en cuenta la técnica *#mobilefirst*, que consiste en desarrollar la aplicación para dispositivos móviles y luego adaptarlo a tablets y sistemas de escritorio.

En los siguientes subapartados se explican las vistas y componentes que forman la interfaz de usuario. Cabe destacar que en las sucesivas pantallas que se muestran a continuación se ha creado una cabecera con el nombre de la aplicación y un menú en la barra de navegación, adaptado para cuando el usuario use la aplicación en un móvil se convierta en un botón, y cuando haga click sobre él, se expanda mostrando las opciones de navegación disponibles.

Se observa la barra de navegación en la parte superior de la pantalla, y como al hacer click sobre el botón se muestran los siguientes elementos de navegación:

- Tracking: realiza un seguimiento de la ruta que inicia el usuario sobre el mapa.
- Mis rutas: visualiza las rutas que un usuario ha guardado en concreto.
- Social: visualiza las rutas que los usuarios han compartido.
- Iniciar sesión: inicio de sesión de un usuario.
- Registrarse: registra un usuario para acceder a las funcionalidades de la aplicación.

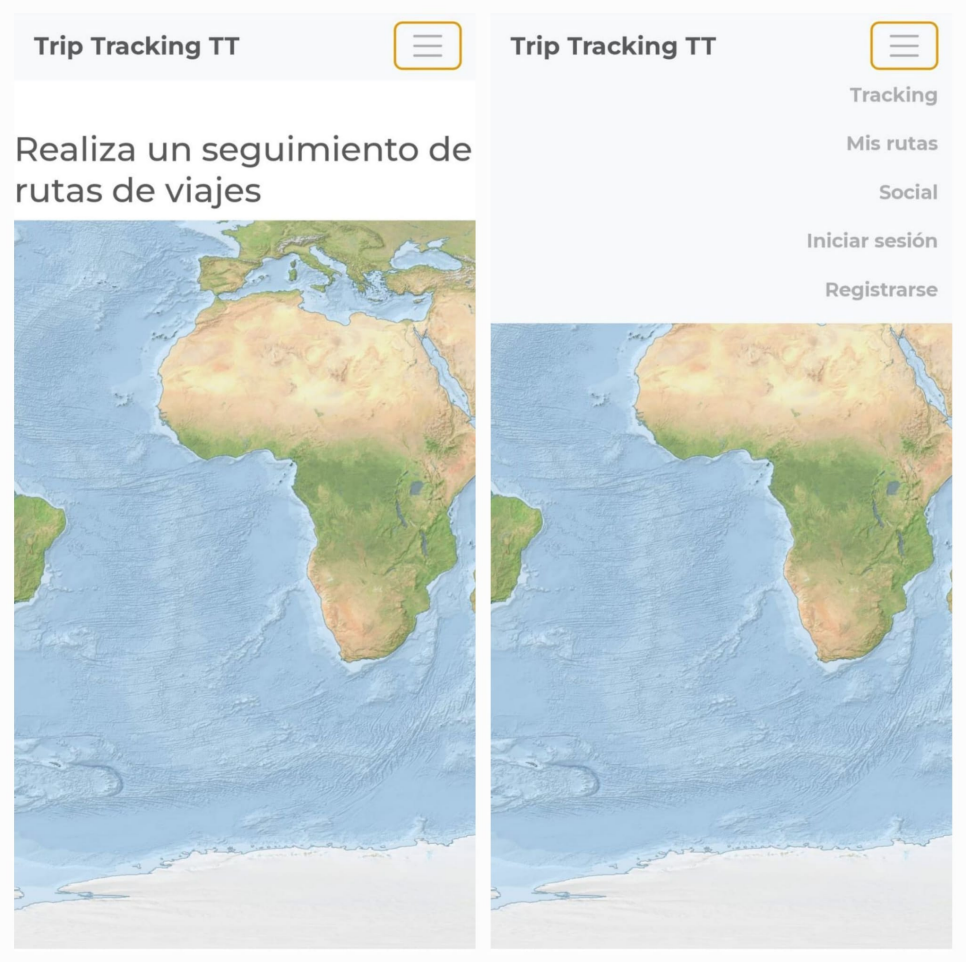


Figura 4.1: Vista de la pantalla inicial de Trip Tracking TT

4.1. Registrarse e Iniciar sesión

El registro y el inicio de sesión de usuario son esenciales, ya que si el usuario no tiene una sesión iniciada, no va a poder registrar o guardar la ruta de la que está haciendo el seguimiento. Se pueden acceder a estos componentes a través de la barra de navegación en la parte superior de la pantalla.

Cuando se accede a cualquiera de ellos, se puede encontrar un formulario con la misma apariencia en las dos vistas, salvo por el contenido que se puede enviar. En el registro de usuario, se deben introducir los campos de correo electrónico, nombre de usuario y contraseña, mientras que en el inicio de sesión se accede únicamente con el nombre de usuario y su contraseña.

4.2. Tracking

Este es el eje principal de la aplicación. En esta pantalla se muestra un mapa interactivo en la parte superior mediante el cual se va mostrando la posición del usuario, mientras que debajo del mismo se encuentran tres botones: uno que sirve

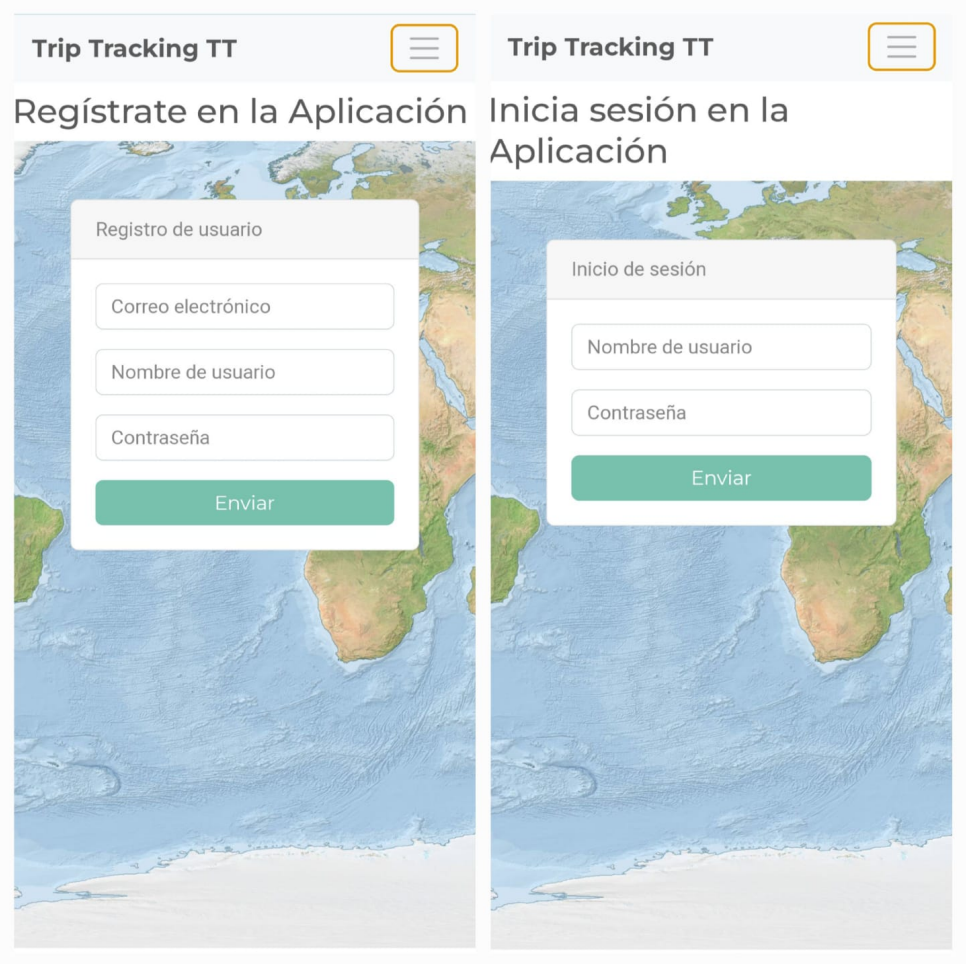


Figura 4.2: Vista de las pantallas de Registro e Inicio de Sesión de Trip Tracking TT

para iniciar la ruta, otro que sirve para finalizar la ruta, y por último otro que la cancela.

En la parte inferior de los botones, se puede observar una caja en la que se muestra información sobre la geolocalización actual del usuario, como las coordenadas, la velocidad o altura actuales.

El usuario puede decidir si compartir la ruta, o mantenerla privada a través de un botón, y, a su vez, debe proporcionarle un nombre a la ruta, si no, no va a poder guardarse.

4.3. Mis rutas y Social

Estas vistas sirven para visualizar las rutas que los usuarios van registrando de dos maneras posibles:

En la vista de *Mis Rutas* se puede apreciar las rutas privadas del usuario que ha iniciado sesión. En dicha vista se presenta un mapa en la parte superior, y un buscador en la parte inferior, que permite encontrar una ruta en base a distintos

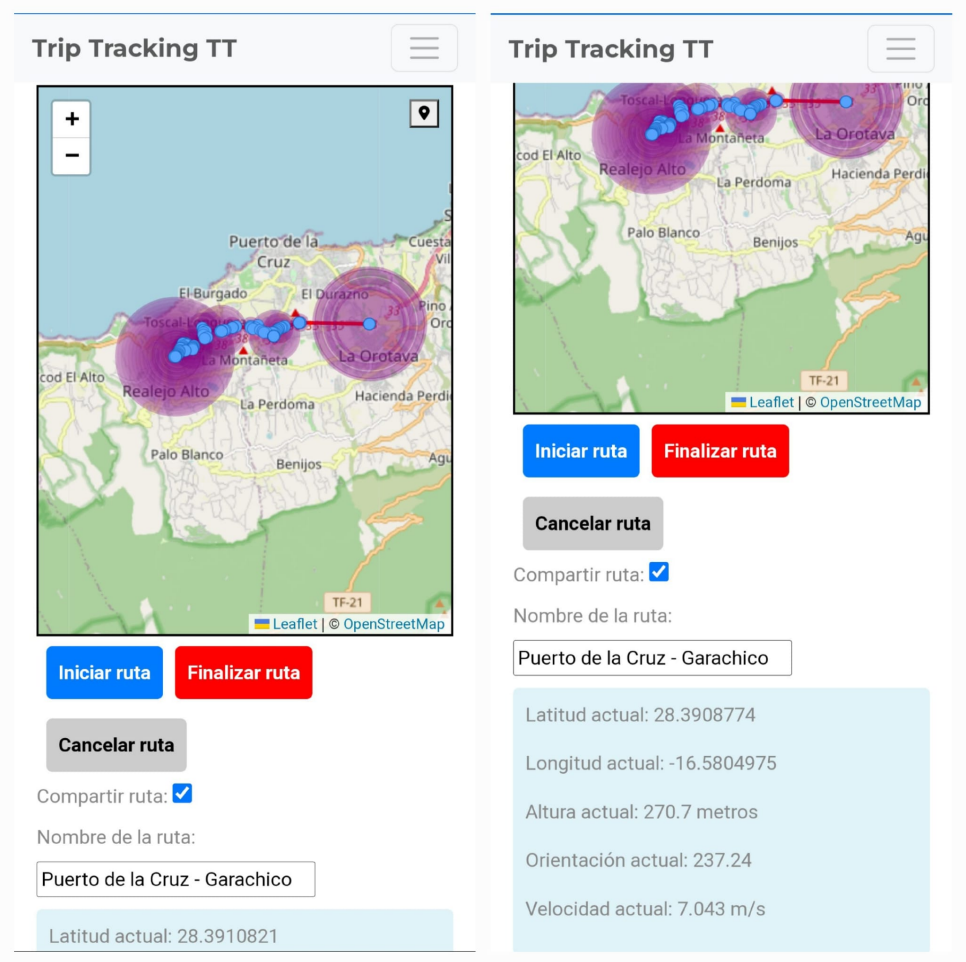


Figura 4.3: Vistas de la pantallas de Tracking de Trip Tracking TT

criterios, como el nombre de la ruta, la duración, o la fecha.

Cuando se inicia la búsqueda, en la parte inferior se pueden observar enlaces con la información de las rutas correspondientes. Si se hace click sobre cualquiera de ellos, la aplicación pinta la ruta sobre el mapa. También es posible editar cierta información de la ruta, como el nombre, o la posibilidad o no de compartir la ruta. Además es posible eliminarla.

Otro detalle es que se pueden visualizar varias rutas a la vez sobre el mismo mapa, y se dispone de un botón para que sobre el mapa no se muestre ninguna ruta.

Sobre la vista de *Social* las rutas aparecen en forma de tarjetas, y si se hace click en alguna de ellas la vista cambia para mostrar un mapa con la información de la ruta en la parte inferior, y un mapa encima de los datos, con la ruta dibujada, indicando los puntos inicial y final.

Por último, mencionar que los datos de las rutas, tanto en la vista *Mis Rutas*, como en *Social*, hacen referencia a la fecha en la que se guardó la ruta, su duración, la velocidad y altitud medias y las máximas y mínimas de cada una de ellas.

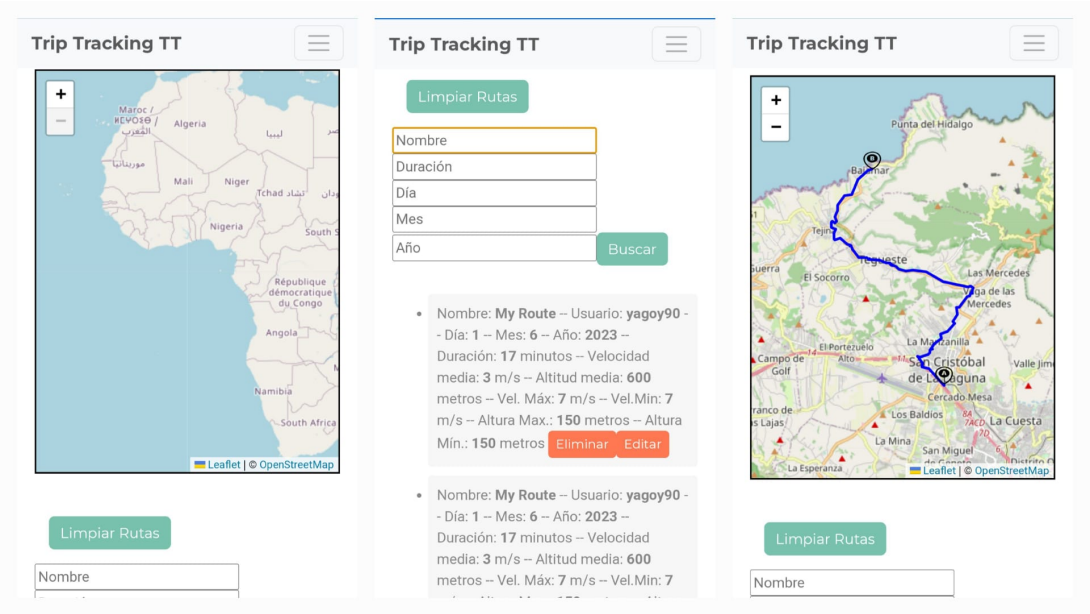


Figura 4.4: Vistas de las pantallas de Mis Rutas de Trip Tracking TT

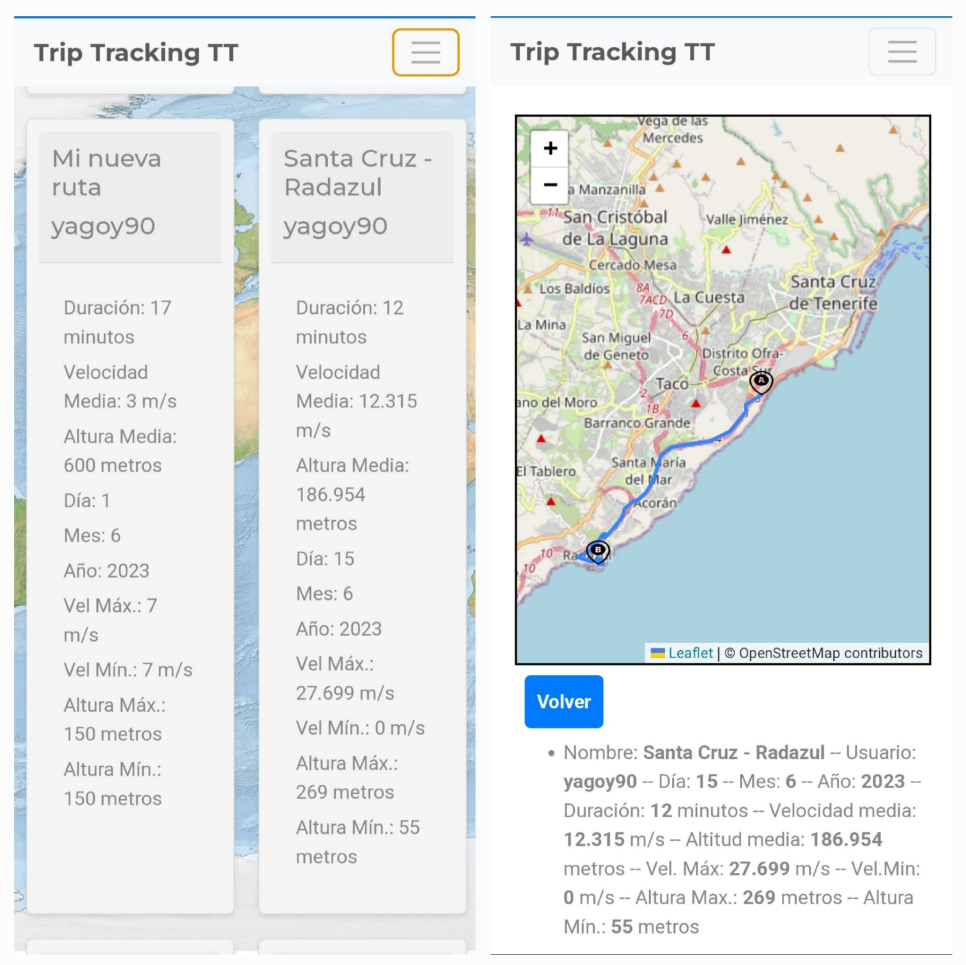


Figura 4.5: Vistas de las pantallas de Social de Trip Tracking TT

Capítulo 5

Conclusiones y líneas futuras

En este capítulo se describen las conclusiones que se han obtenido tras la finalización del proyecto, así como los caminos a seguir para mejorar el trabajo realizado.

5.1. Conclusiones

El desarrollo del presente proyecto ha servido para crear una aplicación que es capaz de seguir una ruta que el usuario inicia, así como registrar rutas de forma genérica, con interacción de mapas y datos de geolocalización en tiempo real, además de compartir las rutas que el usuario ha realizado.

La característica principal de la aplicación web es que funciona como una aplicación web progresiva y posee funcionalidades propias de una aplicación nativa, como la posibilidad de instalarse y obtener actualizaciones automáticas, y guardar datos en caché.

En cuanto a las tecnologías empleadas, cabe destacar que la mayoría de ellas ya se habían utilizado con anterioridad, salvo las relacionadas con el despliegue de la aplicación, Vercel y Heroku, pero que gracias a su facilidad de uso, permiten realizar un despliegue de la aplicación de forma rápida.

Pasando al desarrollo del *backend*, mencionar que el *framework* de desarrollo web ya se había trabajado con anterioridad en otros proyectos, y, por lo tanto, la creación del modelado de datos y el desarrollo de la API no conllevaron especiales dificultades. El problema residió más en el *frontend*.

El desarrollo del *frontend*, la parte más costosa, se centra en el diseño responsivo y en el diseño para móviles, en concreto en la adaptación de la apariencia de la aplicación a distintos dispositivos. También requirió mayor atención la parte de *testing*, debido al uso de la API de geolocalización, que dio problemas en varias etapas del desarrollo.

El resultado final es una aplicación web progresiva que depende de la conecti-

vidad del dispositivo y que cumple con los requisitos iniciales comentados al inicio del presente trabajo.

5.2. Líneas futuras

Tras la finalización del proyecto, sería conveniente incorporar ciertos aspectos con el fin de mejorar la aplicación.

En primer lugar, está la ausencia de conectividad, recordemos que en primera instancia la aplicación iba a realizar un seguimiento de rutas aéreas, sin embargo, y puesto que no se consiguió calcular los datos de geolocalización de forma offline, se tuvo que cambiar el paradigma del proyecto, para pasar a desarrollar una aplicación de rutas genéricas, añadiendo la posibilidad de que en un futuro la API de geolocalización pueda funcionar en ausencia de red.

En segundo lugar, destacar que la aplicación sólo es capaz de registrar tracking en primer plano, es decir, que cuando se cierra la aplicación, o se bloquea el dispositivo, la aplicación ya no calcula los datos de geolocalización, y sería interesante investigar de qué manera se puede cumplir con dicha funcionalidad.

Por último, se podrían mejorar algunos aspectos en la apariencia de la aplicación, como el esquema de colores empleado, o el tamaño de los elementos.

Capítulo 6

Summary and Conclusions

In this chapter, the conclusions drawn from the completion of the project are described, as well as the possible paths to be taken to improve the work that has been done.

6.1. Conclusions

The development of this project has resulted in the creation of an application capable of following a route initiated by the user, as well as recording routes in a generic way, with interactive maps and real-time geolocation data. It also provides the possibility of sharing the routes that the user has taken.

The main feature of the web application is that it functions as a progressive web app, with functionalities similar to those of a native application, such as the ability to be installed, receive automatic updates and cache data.

Regarding the technologies used, it is worth noting that most of them had already been employed, except for those used for deploying the application, Vercel and Heroku. However, thanks to their ease of use, they allow for a quick application deployment.

Moving on to the backend development, it should be mentioned that the web development framework had already been worked on in previous projects. Therefore, creating data modeling and API development posed no particular difficulties. The challenge was mainly in the frontend.

In the frontend development, the most challenging part was focused on responsive design and mobile design, specifically adapting the application's appearance to different devices. Additionally, more attention was required for testing due to the geolocation API, which caused issues during various stages of development.

The final result is a progressive web application that relies on device connectivity and fulfils the initial requirements discussed at the beginning of this work.

6.2. Future work

There are several other aspects that would be interesting to incorporate into the application to improve it.

Firstly, there is the absence of connectivity. Let's remember that initially, the application was supposed to track aerial routes. However, since it was impossible to calculate geolocation data offline, the project paradigm had to be changed to develop a generic route application. In the future, it would be beneficial to add the possibility for the geolocation API to function without a network connection.

Secondly, it is worth noting that the application can only track locations in the foreground. In other words, when the application is closed or the device is locked, the application no longer calculates geolocation data. It would be interesting to investigate how to fulfill this functionality.

Lastly, some aspects of the application's appearance could be improved, such as the color scheme used or the size of elements.

Capítulo 7

Presupuesto

En este capítulo se describe el presupuesto necesario para la realización del proyecto y su despliegue durante cinco meses. En el mismo, se incluyen el coste de los recursos humanos, así como el despliegue de la aplicación en los servicios a las que se ha suscrito.

Los recursos humanos para el desarrollo del proyecto solamente incluyen un desarrollador *full-stack*, mientras que en los servicios en la nube, solamente se debe mencionar los necesarios para el despliegue de la aplicación, puesto que no se necesita ninguna característica o plan adicional de Github, y los servicios de almacenamiento en la nube de MongoDB incluyen un *clúster* gratuito.

Sin embargo, para el despliegue del servidor se ha usado el plan *basic dynos*, que incluye los servicios de despliegue para pequeñas aplicaciones de servidor que no necesitan escalado. Para el *frontend*, se ha usado el plan gratuito de Vercel.

En la tabla que figura en la siguiente tabla se resumen los costes del desarrollo del proyecto para un mes, y para el periodo total del trabajo realizado en la asignatura.

Tipos	Descripción	Coste	Coste Total
Recursos humanos	1 desarrollador full-stack	1300€	6500€
Heroku	Plan Basic Dynos	6,44€	32,2€
Vercel	Plan gratuito de Vercel	0€	0€
MongoDB Atlas	Clúster 0 en AWS	0€	0€
Total		1306,44€	6532,20€

Tabla 7.1: Presupuesto del proyecto

Bibliografía

[1] "'API de geolocalización del estándar W3C.'." https://www.w3.org/TR/geolocation/.

[2] "'Problemas de privacidad de la API de geolocalización del estándar W3C'." https://escholarship.org/uc/item/0rp834wf.

[3] "'Flightradar24.'." https://www.flightradar24.com/.

[4] "'Marinetraffic.'." https://www.marinetraffic.com/.

[5] "'Strava.'." https://www.strava.com/.

[6] "'Leaflet.'." https://leafletjs.com/.

[7] "'Mobilefirst.'." https://developer.mozilla.org/es/docs/Glossary/Mobile_First.

[8] "'Progressive Web Application'." https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

[9] "'Service workers'." https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API.

[10] "'Visual studio code'." https://code.visualstudio.com/.

[11] "'Github'." https://github.com/.

[12] "'Github Actions'." https://github.com/features/actions.

[13] "'Heroku'." https://www.heroku.com/.

[14] "'Vercel'." https://vercel.com/.

[15] "'Sonarcloud'." https://www.sonarsource.com/products/sonarcloud/.

[16] "'ESLint'." https://eslint.org/.

[17] "'NodeJS'." https://nodejs.org/.

[18] "'npm'." https://www.npmjs.com/.

[19] "'Express'." https://expressjs.com/.

[20] "'Insomnia'." https://insomnia.rest/.

[21] "'Mocha'." https://mochajs.org/.

[22] "'Desarrollo guiado por comportamiento'." https://es.wikipedia.org/wiki/Desarrollo_guiado_por_comportamiento.

[23] "'Desarrollo guiado por pruebas'." https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas/.

[24] "'Chai'." https://www.chaijs.com/.

[25] "'Json Web Token'." https://jwt.io/.

[26] "'MongoDB'." https://www.mongodb.com/.

[27] "'MongoDB Atlas'." https://www.mongodb.com/atlas/database.

[28] "'Angular'." https://angular.io/.

[29] "'Single Page Application'." https://es.wikipedia.org/wiki/Single-page_application.

[30] "'Angular Routing'." https://angular.io/guide/routing-overview.

[31] "'Angular CLI'." https://angular.io/cli.

[32] "'Angular HTTP Client Module'." https://angular.io/api/common/http/HttpClientModule.

[33] "'Jasmine'." https://jasmine.github.io/.

[34] "'Karma'." https://karma-runner.github.io/latest/index.html.

[35] "'Bootstrap'." https://getbootstrap.com/.

[36] "'Openstreetmap'." https://www.openstreetmap.org/.

[37] "'Repositorio de la aplicación cliente'." https://github.com/ULL-TFGyMs-vblanco/TFG-2023-YagoPerezMolanes-FlyingTrack.

[38] "'Repositorio del servidor'." https://github.com/alu0101254678/TFG-2023-YagoPerezMolanes-Backend.

[39] "'Modelo Vista-Controlador'." https://developer.mozilla.org/es/docs/Glossary/MVC.