# Gestión de información georreferenciada y API para web y apps

Carlos Márquez Martín

**Resumen**— Hoy en día, el manejo de datos en las aplicaciones móvil son una parte muy importante en nuestra sociedad actual. El presente documento lleva la intención de proporcionar a los desarrolladores de sotware un sistema de manejo de información con el que estos puedan crear aplicaciones móviles o aplicaciones web sobre las rutas arqueológicas que se conocen hoy en día en todo el mundo. Para realizar este proyecto se han utilizado diferentes tecnologías para la creación tanto de la base de datos donde se almacenan los registros como para la API. La base de datos utilizada es MongoDB y el lenguaje de programación con el que se ha llevado a cabo el desarrollo de la API es NodeJS, haciendo uso de la librería Express entre otras. El objetivo del proyecto es el de crear una serie de endpoints en un intermediario (API) para poder hacer llamadas contra este y no directamente a la base de datos, con el peligro que ello conlleva. Por último, y no menos importante, se ha realizado una extensa documentación de todos los endpoints para que cualquier usuario o desarrollador de software pueda utilizarlos sin ningún tipo de dificultad.

**Palabras clave**— base de datos, API, endpoint, MongoDB, NodeJS, manejo de información.

**Abstract**— Today, data management in mobile applications is a very important part of our current society. This document is intended to provide software developers with an information management system with which they can create mobile applications or web applications on the archaeological routes that are known today throughout the world. To carry out this project, different technologies have been used to create both the database where the records are stored and the API. The database used is MongoDB and the programming language with which the API development has been carried out is NodeJS, making use of the Express library among others. The objective of the project is to create a series of endpoints in an intermediary (API) to be able to make calls against it and not directly to the database, with the danger that this entails. Last but not least, extensive documentation has been made for all endpoints so that any user or software developer can use them without any difficulty.

**Index Terms**— database, API, endpoint, MongoDB, NodeJS, information management.

## 1 Introducción - Contexto del trebajo

Actualmente, en el mundo existen muchos destinos arqueológicos. En atención a los yacimientos y monumentos existentes, estos son muy variados y datan de múltiples épocas. Se encuentran en diferentes estados de conservación. Los hay muy conocidos y otros completamente desconocidos, unos son fácilmente accesibles y otros menos. Es por esta razón que se ha decidido realizar este proyecto. El propósito de este es poder proporcionar a los usuarios una base de datos en la que puedan almacenar toda la información relativa a los destinos arqueológicos que existen y una API REST con la que puedan crear rutas, enviar datos sobre el uso de las rutas y crear análisis.

Los usuarios dispondrán de una aplicación desde la cuál puedan consultar datos y realizar acciones. No obstante, aunque este documento trata sobre la gestión de la información a partir de un sistema de información, que permite gestionar datos georeferenciados para el mantenimiento de rutas para las visitas arqueológicas, la implementación de las funciones de la API REST y la documentación de esta, este sofware pretende también ser válido para cualquier aplicación sobre rutas turísticas de todo tipo. Es decir, cualquier tipo de aplicación que necesite de esta API para guardar usuarios, rutas y puntos de interés, podrá disponer de este código y adaptarlo a sus necesidades. No es imprescindible que se trate de rutas arqueológicas.

Esta idea de proyecto, a parte de los objetivos técnicos que se verán a continuación, busca también la finalidad de contribuir a la protección de todos los destinos arqueológicos ya que, con esta aplicación, los usuarios pueden visualizar todos los datos necesarios sobre sus rutas desde su teléfono móvil. Por ende, no haría falta tener que construir enormes carteles informativos o señalizaciones en los yacimientos y monumentos, poniendo en riesgo estos lugares históricos.

## 2 OBJETIVOS

El objetivo de este proyecto es diseñar e implementar un sistema de información que sea capaz de gestionar datos georeferenciados para el mantenimiento de rutas para visitas arqueológicas. Para lograr este objetivo, se van a realizar las siguientes tareas:

- Diseño e implementación de la base de datos, donde se van a alojar los datos georeferenciados de los usuarios que utilizen dicha aplicación.
- Creación de una API REST para que los usuarios puedan interactuar con la base de datos tanto desde la web como de la app.
- Documentación de la API REST utilizando la documentación tipo swagger [1]. Se van a valorar diferentes librerías y se va a escoger la que mejor se adapte a las necesidades de este proyecto.
- Otro objetivo importante es el de añadir seguridad a los datos que se van a recoger. Se van a investigar diferentes maneras de hacerlo y se va a escoger la que mejor se adapte a nuestras necesidades.

## 3 METODOLOGÍA

En este proyecto se ha intentado seguir la metodología de programación extrema (XP). [2] Dicha metodología se diferencia de las tradicionales en que pone más énfasis en la adaptabilidad que en la previsibilidad. Sus defensores consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Se puede considerar la XP como la adopción de las mejores metodologías de desarrollo de acuerdo con lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.
Los valores originales de la XP son: la simplicidad, la comunicación, la retroalimentación (feedback), el coraje y, por último, el respeto.
Además de estos valores, en XP existen cinco principios llamados principios core [3]:

- Realimentación rápida (Rapid feedback)
- Asumir simplicidad (Assume simplicity): Don't Repeat Yourself
- Cambio incremental (Incremental change)
- Abrazar el cambio (Embracing change)
- Trabajo de calidad (Quality work)

En el caso de este proyecto, se ha intentado seguir todos estos valores y principio, sobretodo el de realimentación rápida (en este caso, la comunicación ha sido con el tutor), el de asumir simplicidad y el de presentar un trabajo de calidad.

## 4 PLANIFICACIÓN

La planificación de este proyecto ha constado de 5 fases. En este apartado se explica en qué ha consistido cada una de ellas y, al final de este documento, en el "Anexo 1", se puede encontrar una tabla en la que se ve claramente las tareas en las que se desglosa cada fase y el tiempo que duran tanto las tareas como las fases.

La primera fase es la de planificación y análisis del proyecto. En ella se han realizado actividades como la determinación del ámbito del proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados, su planificación temporal, la asignación de recursos a las diferentes etapas del proyecto y, por último, se ha analizado qué es lo que realmente debe hacer el software y cuáles son las funciones con las que este debe contar.

La segunda fase es la de diseño. En esta se han tomado las decisiones de diseño tanto de la base de datos del proyecto como de la estructura que tendrá este.

Después de definir las decisiones de diseño del proyecto, viene la fase de implementación de este.
Las tareas realizadas son las siguientes:

- Implementar los métodos GET, POST, PUT y DELETE para cada colección de la base de datos.
- Implementar las funciones de registro y autenticación de los usuarios en la API REST utilizando JSON Web Token.
- Refactorizar todo el código de la API REST utilizando el patrón de diseño de software Modelo-Vista-Controlador (MVC).
- Creación de las vistas de prueba para comprobar el correcto funcionamiento tanto del registro como la autenticación de los usuarios.
- Implementar las funciones necesarias para que un usuario autentificado pueda seleccionar diferentes puntos de interés y crear una ruta a partir de estos.
- Añadir más seguridad a las contraseñas que se almacenan en la base de datos.
- Buscar información sobre las diferentes librerías disponibles para la documentación tipo swagger de la API REST y escoger la que mejor se adapte a nuestras necesidades.
- Documentar la API REST.
- Testear los métodos GET, POST, PUT y DELETE con la aplicación POSTMAN.

La cuarta fase es la de testeo/pruebas. Se ha comprobado que todos los endpoints de la API funcionan correctamente y que la conexión de la API con la base de datos no presenta fallos.

Finalmente, en la última fase se han llevado a cabo las últimas modificaciones del proyecto y se ha preparado la documentación final del mismo.

## 5 DESARROLLO

Siguiendo la planificación explicada en el apartado anterior sobre la fase de implementación, se van a explicar a continuación los avances del proyecto respecto a la base de datos y a la API REST.

### 5.1 Diseño e implementación de la base de datos

La base de datos utilizada para la realización de este proyecto será MongoDB. Inicialmente se propuso utilizar MySQL pero, finalmente, se ha optado por la opción de MongoDB, que es un sistema de base de datos NoSQL, orientado a documentos y de código abierto. En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. [4]

El diseño final de la base de datos consta de cinco colecciones (tablas en MySQL). Estas son:

- usuarios
- rutas
- puntos de interés
- imágenes de la ruta
- puntuaciones de las rutas.

En la colección "usuarios" se almacena la información necesaria para poder autenticarse tanto en la aplicación como en la web. Puntos de interés son los distintos yacimientos arqueológicos que existen en España. En esta se guardan los datos de latitud y longitud para poderlos ubicar en el mapa. La colección de rutas se compone de diferentes puntos de interés seleccionados por el usuario. Es decir, el usuario podrá escoger diferentes puntos de interés y estos se guardarán en la colección "rutas". Será a partir de esta que se podrá crear la ruta óptima que debe realizar el usuario si quiere visitar todos los puntos de interés que ha seleccionado préviamente. Por último, la colección de imágenes y la de puntuaciones guardan, como su nombre indica, todas las imágenes de una ruta y las puntuaciones/valoraciones que hagan los usuarios desde la aplicación o desde la web.
Con respecto a los atributos que contiene cada una de las colecciones, en "usuarios" se guarda:

- Id aleatorio que crea automáticamente MongoDB.
- Email que proporcione el usuario. Este será único.
- Nombre de usuario.
- Contraseña encriptada.
- Fecha en la que se ha registrado. Esto no lo tiene que hacer el usuario. Esto se guarda automáticamente gracias a una función de la librería "mongoose" de MongoDB.

En "puntos de interés" se almacena:

- Id aleatorio que crea automáticamente MongoDB.
- Nombre del yacimiento.
- Latitud y longitud para poder ubicarlo en el mapa.
- Dirección del yacimiento.
- Descripción del yacimiento.
- Teléfono.
- URL de la página web del punto de interés.

La colección "rutas" se compone de un array de diccionarios en los que cada uno de ellos se recogen los datos de los diferentes puntos de interés seleccionados por el usuario.

Hasta aquí se han explicado las tres colecciones principales, con las cuales, la aplicación móvil o web ya podría empezar a funcionar perfectamente. Hasta ahora dispone de usuarios que pueden registrarse e iniciar sesión. Cuando estos estén registrados, pueden visualizar, crear, actualizar y eliminar rutas y puntos de interés.

Una vez implementadas estas funciones, se ha querido ampliar las funciones que un usuario puede llevar a cabo. Para esto se han tenido que implementar nuevas funciones en la API y se han tenido que crear nuevas colecciones en la base de datos. Estas nuevas funcionalidades van a permitir a los que utilicen esta aplicación:

- Subir varias imágenes de los puntos de interés y de las rutas.
- Puntuar las rutas que realicen los usuarios.

Para esto, se han creado dos colecciones nuevas. Una para almacenar todas las imágenes que pertenezcan a una ruta o a un punto de interés y otra para que los usuarios puedan puntuar las rutas. Esto último permitirá a los desarroladores que implementen la aplicación móvil o la aplicación web a implentar un sistema de puntuación por estrellas y hacer que la ruta con mejores puntuaciones se muestre en los primeros puestos, por ejemplo. Así se podría hacer un filtro para ordenar las rutas por puntuación.

### 5.1.2 Encriptación de contraseñas

Con la finalidad de proporcionar mayor seguridad a los datos que el usuario registra en la base de datos, se ha decidido no guardar la contraseña personal en texto plano, sino que, se ha utilizado un método de encriptación de contraseñas. La función de hashing utilizada ha sido "Bcrypt". [5] Está basada en el cifrado de Blowfish. Con esta librería se puede generar el hash de cualquier campo. Permite elegir el valor de *saltRounds*, que nos da el control sobre el coste de procesado de los datos. Cuanto más alto es este número, más tiempo requiere la máquina para calcular el hash asociado a la contraseña. Es importante a la hora de elegir este valor, seleccionar un número lo suficientemente alto como para que alguien que intente encontrar la password para un usuario por fuerza bruta, requiera tanto tiempo para generar todas los hash de las contraseñas posibles que no le compense. Y, por otra parte, debe ser lo suficientemente pequeño para no acabar con la paciencia del usuario a la hora de registrarse y de acceder. Por defecto, el valor saltRounds es 10.

Además de la encriptación, "Bcrypt" también nos proporciona una función de comparación de hash para cuando el usuario quiere hacer log-in en el sistema. Esta función lo que hace es: recoge la contraseña introducida por el usuario, le hace el hash y este lo compara con el de la base de datos. Devolverá True o False dependiendo del resultado.

La función "Bcrypt", como algoritmo de hasheado, tiene ventajas importantes sobre otros algoritmos de cifrado, como SHA-256 con salt. El coste en tiempo de cálculo del algoritmo bcrypt es mayor, por lo que las password generadas son mucho más seguras frente a ataques de fuerza bruta, ya que los atacantes necesitarían mucho más tiempo para probar cada una de las claves posibles.

### 5.1.3 Base de datos definitiva

Después de revisar la configuración de la base de datos y de elaborar nuevas funciones en la API, se ha decidido hacer una serie de cambios con respecto a las colecciones y con respecto a los datos que se almacenan y cómo se almacenan en cada una de ellas.
El principal problema que ha provocado que se hayan tenido que hacer cambios es que el planteamiento que se hizo al principio del proyecto no fue el correcto, ya que no se pensó de la manera correcta. Es decir, MongoDB es una base de datos No-SQL y, por falta de experiencia e información, la forma en la que se iban a almacenar los datos fue pensada como si fuese una base de datos relacional y SQL.
Precisamente, la razón por la cual se escogió MongoDB es porque este proyecto no requeria de grandes relaciones entre los datos. Por ejemplo, se ha querido hacer una función que almacene rutas y puntos de interés favoritos para cada usuario. Inicialmente, hizo una tabla intermedia que relacionase las rutas y los puntos con los usuarios. Después de investigar más a fondo se vió que esto no era necesario. Con una base de datos no relacional no hace falta crear una tabla que relacione el id del usuario con el id de las rutas y de los puntos. Es mucho más simple. Lo que se ha hecho es que, la colección de usuarios dispone de un campo de tipo Array en el cual se almacenan los id de las rutas y de los puntos que el usuario quiera agregar a su lista de favoritos.
Al igual que con el tema de las rutas y puntos favoritos, se ha hecho lo mismo para el almacenamiento de las imágenes de la rutas.

A parte de esto, se ha tomado la decisión de implementar un campo "validado" en las colecciones de los puntos de interés, las rutas y las imágenes. La intención de esto es que, la aplicación móvil o la web tengan un usuario validador, por ejemplo, que su función sea la de validar si los datos entrados en la base de datos son válidos para ser publicados. Con esta medida se pueden evitar dos problemas. El primero es que un usuario no podrá subir información e imágenes con la intención de publicar registros no apropiados. Y, el segundo es que, si el usuario sube una imagen que no posee la resolución correcta o que el tamaño de esta no se adaptará bien a la amplicación, entonces el usuario validador deberá descartar dicha imagen e informar al usuario del por qué del descarte.

En el "Anexo 2" se puede encontrar el esquema que representa todas las colecciones de la base de datos.

## 5.2 API REST

Para la creación de la API REST, se decidió que el lenguaje a utilizar para programarla sería NodeJS. Esta decisión se ha tomado en base a la facilidad de uso y al amplio conocimiento sobre este lenguaje por parte del desarrolador de sofware de este proyecto, facilitando así la implementación de las funcionalidades de la API REST.

### 5.2.1 Librerías utilizadas para la API

A medida que se avanza en la programación de la API, esta va requiriendo diferentes librerias que se guardan en el apartado de "*dependencies*" en el archivo *package.json*. A continuación, se van a listar las dependencias utilizadas hasta el momento, con una breve explicación de su uso:

- Express → permite estructurar una aplicación de una manera ágil, nos proporciona funcionalidades como el enrutamiento, opciones para gestionar sesiones y cookies, etc. [6]
- Body-parser → analiza el cuerpo de las peticiones y las devuelve en un objeto JSON.
- Bcrypt-nodejs → es una función de hashing de passwords basado en el cifrado de Blowfish. Lleva incorporado un valor llamado salt, que es un fragmento aleatorio que se usará para generar el hash asociado a la password, y se guardará junto con ella en la base de datos.

- Mongoose → permite crear esquemas para las colecciones/tablas de MongoDB.
- Moment → se utiliza para el manejo de fechas. En este proyecto es usado para el manejo de fechas de expiración de los tokens.
- Jwt-simple → se utilitza para manejar la autenticación en aplicaciones o web.
- Express-handlebars → es un motor de plantillas para crear vistes con etiquetes HTML.

### 5.2.2 Patrón de diseño MVC

Como se describe en la planificación de este proyecto, una de las tareas es la de refactorizar el código utilizando el patrón de diseño de software MVC.
Para ello se han creado tres carpetas: controladores, modelos y vistas. La carpeta "controladores" contiene tres ficheros, uno por cada colección. Dentro de estos se definen todos los métodos de petición necesarios para el correcto funcionamiento de la API.
La carpeta "modelos" contiene también tres ficheros (uno por colección), en los que se definen los esquemas de cada colección de la base de datos.
Por último, la carpeta "vistas" contiene un fichero "login.hbs" con la que podemos comprobar de manera gráfica el correcto funcionamiento de la autenticación por parte de los usuarios registrados en la base de datos.

### 5.2.3 Métodos de petición para API

En este apartado se ha hecho una pequeña modificación en la planificación ya que, se han implementado todos los métodos de petición en dos colecciones y no en las tres, tal y como se describe en la planificación del proyecto. Las colecciones implementadas al completo son la colección "usuarios" y la de "puntos de interés".
La colección "usuarios" contiene dos métodos GET (uno para listar todos los registros que haya en la tabla y otro para buscar por el campo ID), dos métodos POST (uno para el registro y otro para la autenticación), un método PUT (para editar usuarios registrados) y un método DELETE (para eliminar usuarios).
La colección "puntos de interés" contiene dos métodos GET (uno para listar todos los registros que haya en la tabla y otro para buscar por el campo ID), un método POST (para guardar un nuevo punto en la base de datos), un método PUT (para editar puntos de interés guardados en la base de datos) y un método DELETE (para eliminar puntos de interés).

### 5.2.4 JSON Web Token

Con la ayuda de la librería *jwt-simple* se ha creado un sistema de autenticación de usuarios mediante token. Dicho sistema consiste en que, cuando un usuario se registre en la aplicación, a este se le asigna un token con una fecha de expiración de este.

La ventaja que nos aporta este sistema es que, existen ciertas funciones de la aplicación que solo pueden ser ejecutadas por usuarios registrados y que posean un token válido. Por poner un ejemplo, un usuario no registrado podrá ver los puntos de interés y la información asociada a estos, pero, sin embargo, no podrá seleccionar varios de estos para crear una ruta.
Para conseguir crear este método de autenticación de usuarios, se ha creado un middleware que comprueba que en las cabeceras de las peticiones a la API viaja el token único del usuario que realiza la petición. Si este existe y es válido, entonces el usuario podrá ejecutar la función que haya requerido. Por el contrario, el sistema mandará un aviso de que no tiene autorización para realizar dicha acción.

### 5.2.5 Documentación API REST con Swagger

La documentación de una API se refiere al contenido técnico con instrucciones claras sobre cómo funciona una API, sus capacidades y cómo usarla.
No importa qué tan buena sea una API para crear y extender un servicio de software, ya que, esta podría ser inutilizable si los desarrolladores no pueden entender cómo funciona.
El objetivo de la documentación de esta API REST es el de trabajar como una fuente de referencia precisa capaz de describir la API en profundidad y actuar como herramienta de enseñanza y una guía para ayudar a los usuarios a familiarizarse con el software y poder utilizarlo.

Un manual completo que contiene toda la información necesaria para trabajar con una API específica, como funciones, argumentos, tipos de retorno y clases. El documento también incluye ejemplos para respaldar la información. Este debe ser fácilmente entendible para los usuarios o desarrolladores que requieran de su uso.

Para encontrar la librería que se adecuase mejor a nuestro proyecto, se ha buscado información en Internet sobre cómo documentar una API. Al hacerlo, el 90% de los resultados que aparecieron, todos hablaban sobre "swagger-jsdoc". Por tanto, se buscó información específicamente para esa librería. Y, finalmente, debido a la fácil implementación y al resultado final que ofrece al usuario (se puede ver en el Anexo 3 la interfaz final), se ha decidido utilizar "swagger-jsdoc" [7] para documentar la API REST de este proyecto.

Con dicha librería se puede documentar todos los *enpoints* de Express utilizando la especificación Swagger OpenAPI 3 sin escribir archivos YAML o JSON. Se puede escribir comentarios *jsdoc* encima de cada *enpoint* y la librería creará la interfaz de usuario swagger. En la Fig.1. se puede ver un ejemplo de cómo documentar el *Schema* de una colección de la base de datos.

```
/**
 * @swagger
 * components:
 *  schemas:
 *    Usuarios:
 *      type: object
 *      properties:
 *        email:
 *          type: string
 *          description: Email del usuario
 *        displayName:
 *          type: string
 *          description: Nombre del usuario
 *        password:
 *          type: string
 *          description: Contraseña del usuario
 *        rutas_favoritas:
 *          type: array
 *          description: Rutas favoritas del usuario
 *        puntos_favoritos:
 *          type: array
 *          description: Puntos favoritos del usuario
 *      required:
 *        - email
 *        - password
 */
```

Fig.1. – Documentación Schema Usuarios

Con los comentarios de la Fig.1. sobre el *Schema* de la colección "Usuariso", el resultado que se obtiene es el que se puede apreciar en la Fig.2. Esto se ha hecho con todas las colecciones de la base de datos.

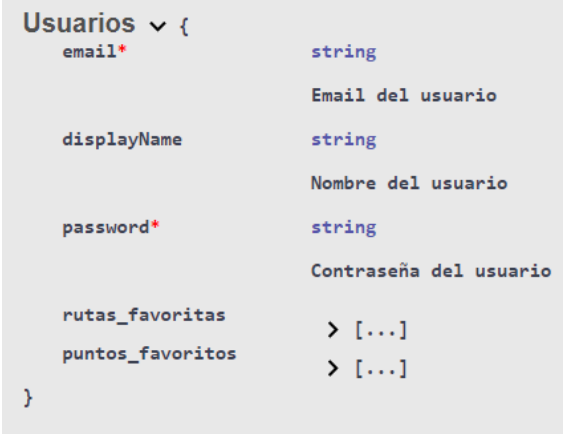



Fig.2. – Resultado documentación Schema Fig.1.

Ahora, en la Fig.3. se puede ver el ejemplo de cómo documentar un *enpoint*.

```
/**
 * @swagger
 * /api/punto:
 *   get:
 *     summary: return puntos de interés
 *     tags: [Punto]
 *     responses:
 *       200:
 *         description: todos los puntos de interés
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/Puntos'
 */
```

Fig.3. – Documentación Endpoint getPuntos().

Después de realizar la documentación de la Fig.3., en el "Anexo 3" se puede ver el resultado que se obtiene finalmente de la documentación de un *endpoint*.

Una de las características que ha sido determinante a la hora de escoger la librería "swagger-jsdoc" es que, además de documentar los Schemas de todas las colecciones y documentar todos los *endpoint* que ofrece la API, estos *endpoints* se pueden testear desde la propia interfaz web. La información que proporciona la documentación es la siguiente:

- URL del *endpoint* más una breve descripción de lo que devuelve o de la función que realiza.
- Parámetros necesarios (por ejemplo, id en el caso de querer borrar un registro).
- Códigos de respuesta que se pueden obtener al ejecutar la función con su respectiva descripción.
- Ejemplo de uso del Schema de la colección.

### 5.3 Creación de vistas

Para el testeo de la API se ha utilizado tanto el navegador web para las llamadas a los endpoints tipo GET como la aplicación POSTMAN [8], en la cual se han podido testear todos los endpoints de la API. No obstante, con el fin de implementar de forma completa el Model-Vista-Controlador, se han creado dos vistas. Una para probar el registro de un nuevo usuario y otra para probar el correcto funcionamiento del log-in. Estas dos vistas son muy sencillas pero suficientes tanto para comprobar que las funciones anteriores funcionan y para aprender a utilizar un motor de plantillas del cual se desconocía su existencia. Este es, el sistema de templates Handlebars (HBS) en NodeJS.

Existen diversos motores de plantillas para usar con NodeJS. Sin embargo, se ha escogido HBS por su sencillez a la hora de usarlo. No hay necesidad de usar otros lenguajes, como sí ocurre en otros motores de plantillas, y eso es una gran ventaja ya que, el propósito de esto no es hace una vista elaborada, sino una vista para comprobar el funcionamiento de funciones concretas.

### 5.4 Casos de uso

Caso de uso 1 – Usuario normal (Fig.4.)

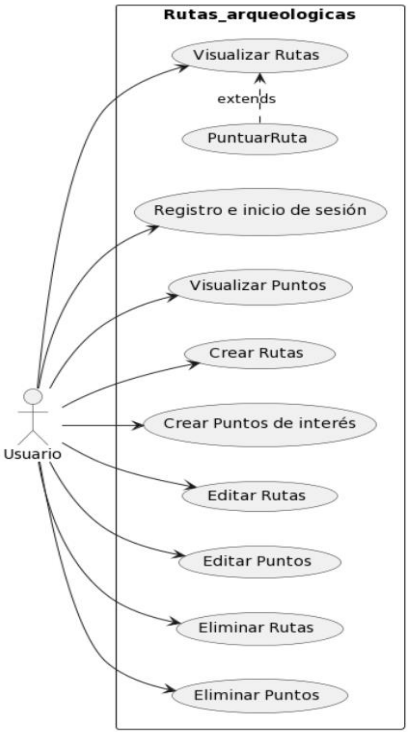



Fig.4. – Caso de uso usuario

En este primero caso de uso se ven reflejadas todas las funciones que un usuario normal registrado puede llevar a cabo.

Caso de uso 2 – Usuario validador (Fig.5.)

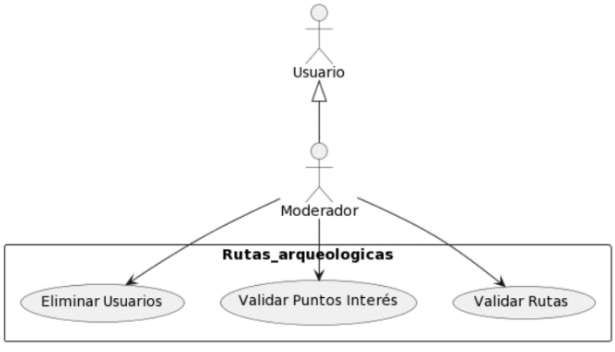



Fig.5. – Caso de uso usuario validador

En este segundo caso de uso tenemos a un usuario validador. Las funciones de este son, a parte de las que ya puede realizar un usuario normal, las de validar rutas, puntos de interés y eliminar usuarios. Es decir, funciones que no puede hacer el usuario del caso de uso 1.

### 5.5 Postman

Postman es una plataforma de API para que los desarrolladores diseñen, construyan, prueben e iteren sus API. Esta permite:

- Crear peticiones de APIs internas o de terceros.
- Elaborar test para validar el comportamiento de las APIs.
- Documentar APIs.

Conforme se iba avanzando el proyecto y se iban implementando los diferentes enpoints, hubo que buscar la manera de poder testear todos los endpoints que se iban desarrollando. Al principio no había problema, ya que, con las peticiones de tipo GET, había suficiente con realizarla en cualquier motor de búsqueda, como Google Chrome.
Sin embargo, para el resto de las peticiones (POST, PUT, DELETE) no era suficiente. Por esto, se empezó a utilizar POSTMAN para el testeo completo de la API REST.

Anteriormente en este artículo, se ha hablado de Swagger, que es la herramienta de software que se ha utilizado para documentar la API. Y, realmente, con Swagger también se podrían haber realizado todos estos test. Al igual que se podría haber utilizado POSTMAN para documentar. No obstante, se decidió utilizar las dos herramientas.

En el caso de los casos de prueba, POSTMAN es mucho más sencillo de usar que Swagger. Y, lo mismo pasa con el tema de la documentación. Swagger, a parte de tener una interfaz mucho más atractiva y legible para el usuario, es mucho más sencilla de implementar que en POSTMAN.

En las figuras del Anexo 4, se pueden ver tres ejemplos de test a dos endpoints diferentes. El primero corresponde a un endpoint de tipo GET (getPuntos()), que devuelve un array de todos los puntos de interés almacenados en la base de datos. El segundo corresponde a un endpoint de tipo POST (signUp()). Sirve para llevar a cabo el registro de un nuevo usuario. De este último se ha hecho el test de forma errónea para que se pueda ver que da error y, se ha hecho el test de forma correcta, es decir, el usuario se registra y se le asigna el token de autenticación.

## 5 RESULTADOS DEL PROYECTO

Al termino de este proyecto, cabe destacar que se han cumplido todos los objetivos que se plantearon al inicio de este, y que se han llevado a cabo todas las tareas que se reflejan en la planificación (Anexo 1). Es cierto que, ha habido tareas, que por falta de tiempo o por no hacer una buena estimación del tiempo necesario para realizar una tarea, han tenido que retrasarse. Sin embargo, estos retrasos no han afectado a la entre-

ga final del proyecto, ya que, al igual que hay algunas tareas que se han retrasado, hay otras en las que la estimación del tiempo ha sido excesiva y, esos huecos, se han podido ir rellenando con otras tareas nuevas.
Por lo tanto, los resultados de este proyecto son:

- Una base de datos no relacional con MongoDB.
- Una API REST con diferentes endpoints para extraer datos y comunicarse con la base de datos.
- Un método de auteticación mediante JSON Web Token.
- Documentación de la API con Swagger.

Al final de este documento, en la parte de los Anexos se puede encontrar:

- Anexo 1 → Planificación del proyecto
- Anexo 2 → Esquema de la BD NoSQL.
- Anexo 3 → Interfaz final de la documentación de la API REST con Swagger.
- Anexo 4 → Prueba de la API con el cliente de POSTMAN.

## 6. CONCLUSIONES

Durante la carrera de ingeniería informática se llevan a cabo decenas de proyecto en los que hay que desarrollar diferentes códigos en diferentes lenguajes de programación para hacer una página web o una aplicación móvil y, los datos de estas, siempre se extraen directamente de la base de datos, sin intermediario, y siempre utilizando bases de datos relacionales.
Pues bien, gracias a la realización de este proyecto se ha podido conocer diferentes herramientas y formas para manejar los datos que se almacenan en una base de datos.
Para empezar, el planteamiento y la elección de la base de datos ha cambiado por completo. Se ha pasado de una relacional a una no relacional, en la que los datos se almacenan en formato JSON y las relaciones entre tablas/colecciones es totalmente distinta. Se han podido apreciar las ventajas que supone trabajar con este tipo de base de datos. Estas son mucho más flexibles a la hora de crear esquemas de información, ofrecen mayor escalabilidad, es decir, pueden soportar mayores volúmenes de datos y añadir mayor capacidad añadiendo nuevos módulos de software sin necesidad de añadir nuevos servidores. Garantizan un alto rendimiento y, son muy funcionales, ya que cuentan con API exclusivas y proporcionan modelos de datos para trabajar con cada tipo de datos presentes en la base. [9]

Por otra parte, como se ha dicho, todos los proyectos de este tipo se han llevado a cabo sin intermediario. En este caso sí que se ha implementado una API, la cual recibe llamadas por parte de la aplicación y, a su vez, esta es la que extrae los datos almacenados en la base de datos. Y, tras llevar a cabo la realización de una API desde cero, se ha podido tomar consciencia de varias ventajas que puede proporcionar. Por un lado, tenemos el tema de la seguridad. Estas son seguras ya que crean una especie de barrera que permite el acceso solo a la información que forma parte de esa aplicación, no a todo el sistema. También aumenta la eficiencia de los sistemas y las aplicaciones. Las APIs contribuyen a mejorar el rendimiento de los sistemas, ya que no es necesario iniciar procesos de desarrollo desde cero al crear un servicio. Por último, el volumen de datos es menor porque cada API es específica para cada función, es decir, solo se insertan en el sistema los datos que son realmente necesarios para la tarea esperada. [10]
Para terminar, otro de los objetivos importantes de este proyecto era el de documentar la API para que cualquier desarrollador pueda utilizarla sin dificultades y, ese objetivo se ha cumplido con creces. Se ha conseguido hacer una documentación muy sencilla pero muy eficaz gracias a la librería "swagger-jsdoc".
La idea es seguir perfeccionando tanto la base de datos como la API REST con la intención de ir mejorando poco a poco la experiencia de uso de los usuarios finales. También sería bueno ponerse en contacto con los desarrolladores que han hecho la aplicación móvil para debatir la necesidad de mejorar o crear nuevas funcionalidades útiles para los usuarios y para el correcto funcionamiento del aplicativo móvil.

De cara a implementaciones futuras, sería bueno implementar un sistema de seguridad basado en roles de usuario. El objetivo sería crear diferentes roles y asignarle uno o varios de estos a los usuarios. Con este sistema lo que se puede llegar a conseguir es que, según el rol que tenga, un usuario solo podrá ver o ejecutar funciones permitidas para dicho rol.

## BIBLIOGRAFIA


[1] IBM Integration Bus, "Swagger", [Internet]. Disponible en: https://www.ibm.com/docs/es/integration-bus/10.0?topic=ssmkhh-10-0-0-com-ibm-etools-mft-doc-bi12018--htm

[2] Wikipedia, "Programación extrema", [Internet]. Disponible en: https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema

[3] Marvin López Mendoza, (2020, Sep. 18), "Extreme Programming: Qué es y cómo aplicarlo", [Internet]. Disponible en: https://openwebinars.net/blog/extreme-programming-que-es-y-como-aplicarlo/

[4] Wikipedia, "MongoDB", [Internet]. Disponible en: https://es.wikipedia.org/wiki/MongoDB

[5] Izertis, "Encriptación de password en NodeJS y MongoDB: bcrypt", [Internet]. Disponible en: https://www.izertis.com/es/-/blog/encriptacion-de-password-en-nodejs-y-mongodb-bcrypt.

[6] If geek then, "Qué es Express.JS y primeros pasos", [Internet]. Disponible en: https://ifgeekthen.nttdata.com/es/que-es-expressjs-y-primeros-pasos

[7] BRIKEV GITHUB, "express-jsdoc-swagger", [Internet]. Disponible en: https://brikev.github.io/express-jsdoc-swagger-docs/#/

[8] Postman, "Build APIs together", [Internet]. Disponible en: https://www.postman.com/

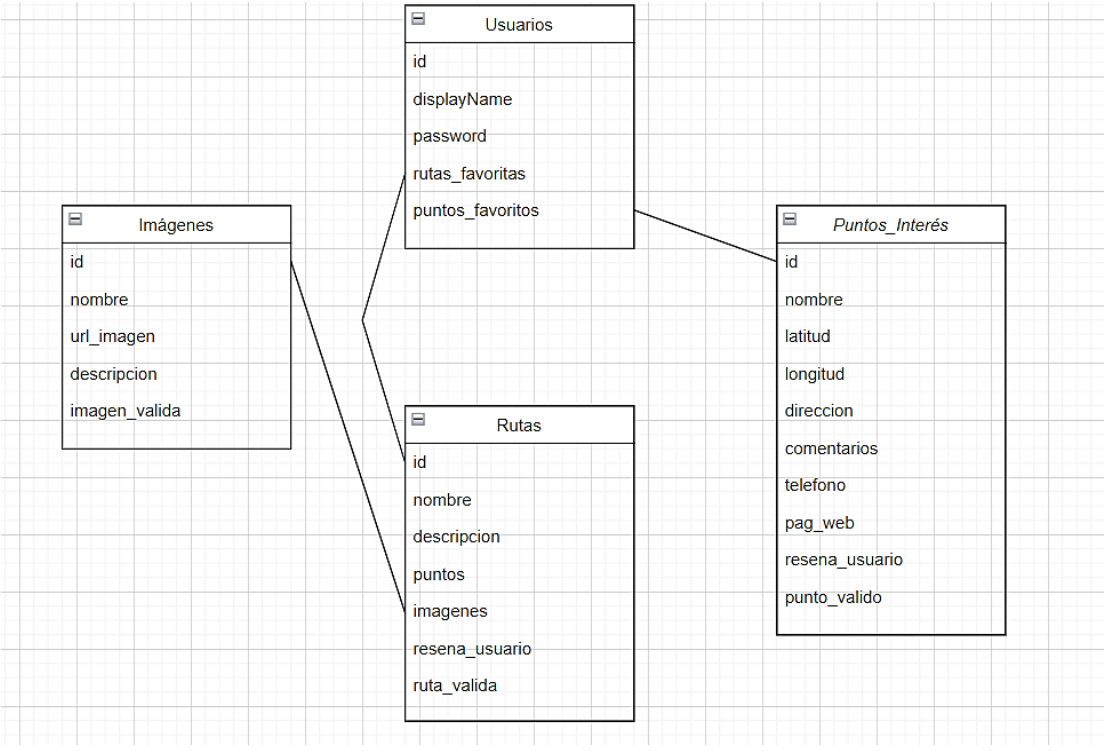
[9] Ayudaley, "Base de datos no relacional. ¿Qué es? Características y ejemplos", [Internet]. Disponible en: https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/#:~:text=Estas%20son%20las%20principales%20ventajas,Ofrecen%20una%20mayor%20escalabilidad.

[10] Zendesk, "¿Qué es una API? Ventajas y usos", [Internet]. Disponible en: https://www.zendesk.com.mx/blog/que-es-api/

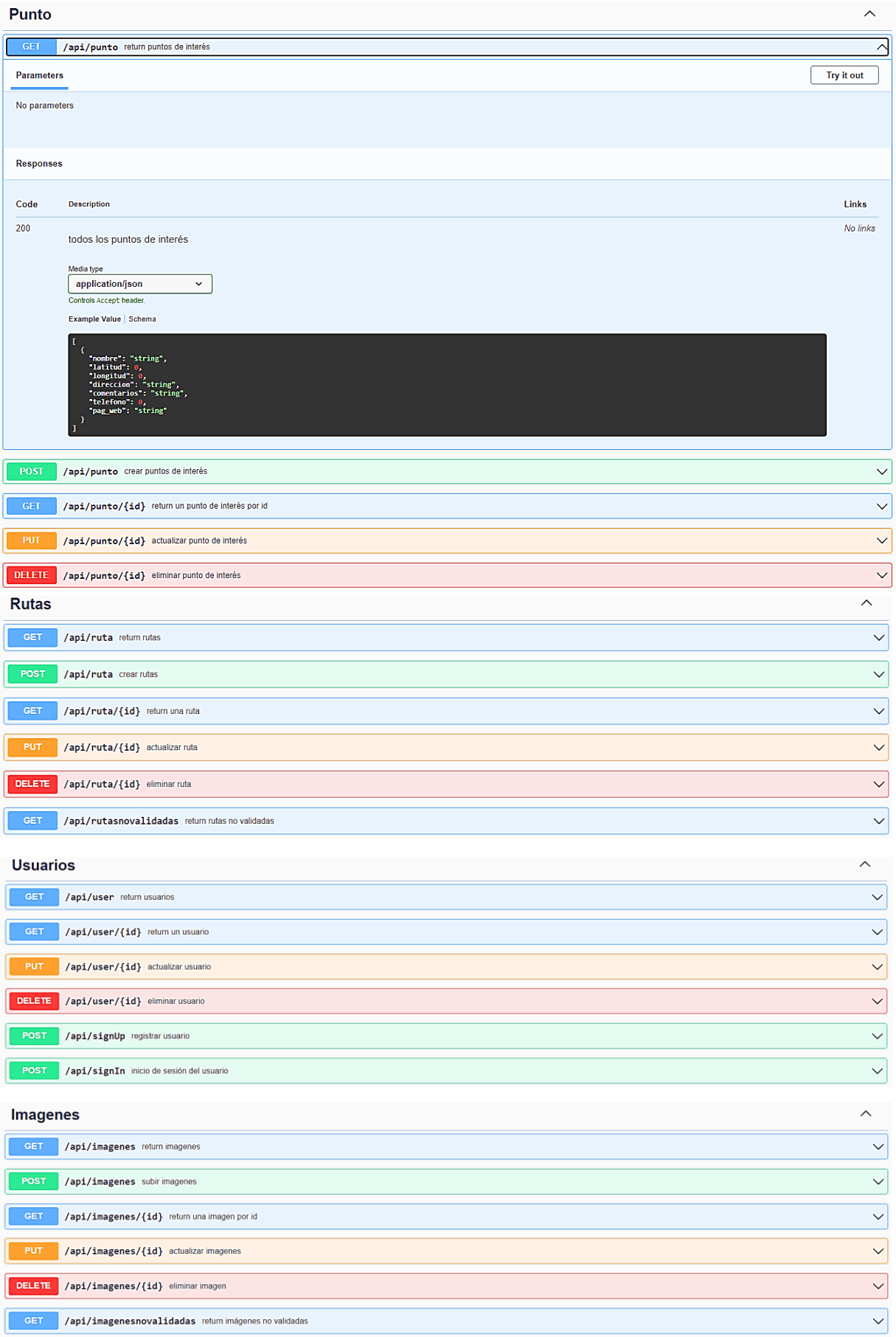
# ANEXO 1

| FASE | TAREAS | DURACIÓN TAREAS |
|---|---|---|
| Planificación y análisis | | 8 días |
| Diseño | Diseñar e implementar BD en MongoDB.<br><br>Empezar a programar la API REST con el lenguaje seleccionado. Creando los archivos necesarios para poder interactuar con la base de datos.<br><br>Buscar información sobre las diferentes librerías disponibles para la documentación tipo swagger de la API REST y escoger la que mejor se adapte a nuestras necesidades.<br><br>Decidir qué lenguaje de programación se va a utilizar para la implementación de la API REST. | 14 días |
| Implementación | Implementar los métodos GET, POST, PUT y DELETE para cada colección de la base de datos.<br><br>Implementar las funciones de registro y au-tenticación de los usuarios en la API REST uti-lizando JSON Web Token.<br><br>Refactorizar todo el código de la API REST utilizando el patrón de diseño de software Modelo-Vista-Controlador (MVC).<br>Creación de las vistas de prueba para com-probar el correcto funcionamiento tanto del registro como la autenticación de los usuarios.<br><br>Implementar las funciones necesarias para que un usuario autentificado pueda seleccionar diferentes puntos de interés y crear una ruta a partir de estos.<br><br>Añadir más seguridad a las contraseñas que se almacenan en la base de datos.<br><br>Documentar la API REST. | 60 días |
| Pruebas | Testear los métodos GET, POST, PUT y DELETE con la aplicación POSTMAN. | 15 días |
| Entrega final | Elaboración del informe final del proyecto.<br><br>Preparación de la presentación final para la defensa del proyecto. | 15 días |

# ANEXO 2


Usuarios
id
displayName
password
rutas_favoritas
puntos_favoritos
Imágenes
id
nombre
url_imagen
descripcion
imagen_valida
Rutas
id
nombre
descripcion
puntos
imagenes
resena_usuario
ruta_valida
Puntos_Interés
id
nombre
latitud
longitud
direccion
comentarios
telefono
pag_web
resena_usuario
punto_valido

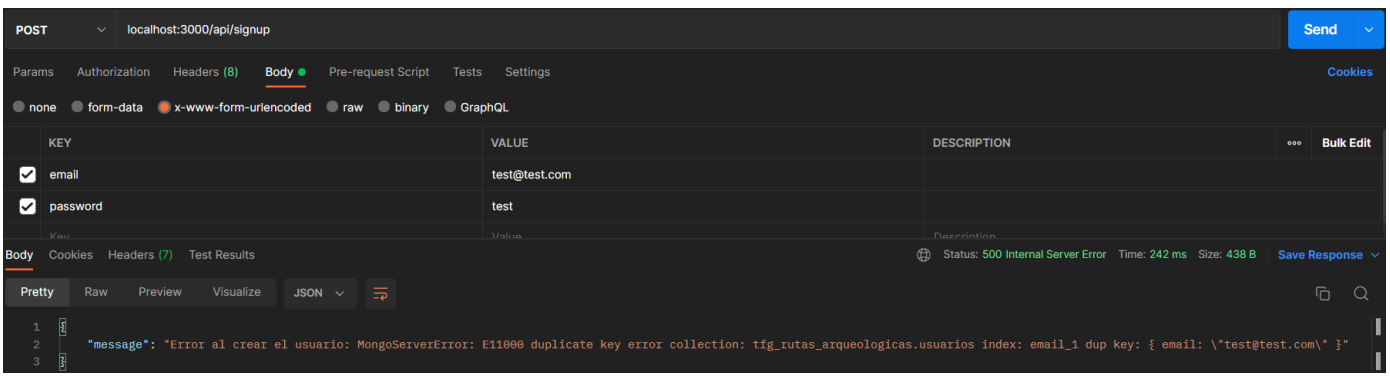
# ANEXO 3


Punto
GET /api/punto return puntos de interés
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
todos los puntos de interés
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
[
{
"nombre": "string",
"latitud": 0,
"longitud": 0,
"direccion": "string",
"comentarios": "string",
"telefono": 0,
"pag_web": "string"
}
]
POST /api/punto crear puntos de interés
GET /api/punto/{id} return un punto de interés por id
PUT /api/punto/{id} actualizar punto de interés
DELETE /api/punto/{id} eliminar punto de interés
Rutas
GET /api/ruta return rutas
POST /api/ruta crear rutas
GET /api/ruta/{id} return una ruta
PUT /api/ruta/{id} actualizar ruta
DELETE /api/ruta/{id} eliminar ruta
GET /api/rutasnovalidadas return rutas no validadas
Usuarios
GET /api/user return usuarios
GET /api/user/{id} return un usuario
PUT /api/user/{id} actualizar usuario
DELETE /api/user/{id} eliminar usuario
POST /api/signUp registrar usuario
POST /api/signIn inicio de sesión del usuario
Imagenes
GET /api/imagenes return imagenes
POST /api/imagenes subir imagenes
GET /api/imagenes/{id} return una imagen por id
PUT /api/imagenes/{id} actualizar imagenes
DELETE /api/imagenes/{id} eliminar imagen
GET /api/imagenesnovalidadas return imágenes no validadas

# ANEXO 4

- Test endpoint tipo GET (getPuntos()). Devuelve todos los puntos de interés que hay almacenados en la base de datos.


- Test endpoint tipo POST (signUp()). Como se puede ver, da mensaje de error porque el email proporcionado ya existe en la base de datos




- Test endpoint tipo POST (signUp()). En este caso, el usuario se registra y se le asigna un Token de autenticación.