

API REST ROUTES TRACKER

ADRIÁN ARROYO GÓMEZ



16 de diciembre de 2024

I.E.S RAFAEL ALBERTI

Contenido

[Registro de Usuario (POST /usuarios/register). 4](#_Toc185183438)

[Login de Usuario (POST /usuarios/login). 6](#_Toc185183439)

[GetUserById (GET /usuarios/{id}). 7](#_Toc185183440)

[GetAllUsers (GET /usuarios/) 9](#_Toc185183441)

[UpdateUserById (PUT /usuarios/{id}). 11](#_Toc185183442)

[DeleteUserByid (DELETE /usuarios/{id}) 13](#_Toc185183443)

[Registro de rutas (POST /rutas/) 14](#_Toc185183444)

[GetRutaById (GET /rutas/{id}) 16](#_Toc185183445)

[GetAllRutas (GET /rutas/) 17](#_Toc185183446)

[UpdateRutaById (PUT /rutas/{id}) 18](#_Toc185183447)

[DeleteRutaById (DELETE /rutas/{id}) 19](#_Toc185183448)

[Crear punto gps (POST /puntos\_gps/{ruta\_Id}) 20](#_Toc185183449)

[Obtener los puntos GPS de una ruta (GET /puntos\_gps/{ruta\_Id}) 21](#_Toc185183450)

[UpdatePuntoGpsById (PUT /puntos\_gps/{punto\_Id}) 22](#_Toc185183451)

[DeletePuntoGPSById (DELETE /puntos\_gps/{punto\_Id}) 23](#_Toc185183452)

# Idea del proyecto

Este proyecto consiste en la creación de una API REST segura, desarrollada con Spring Boot y usando Spring Security.  
La idea principal es una App que permite almacenar y registrar el seguimiento GPS de un vehículo usando la ubicación en tiempo real de un dispositivo móvil.

# Justificación del proyecto

Esta App nace para cubrir la necesidad de poder registrar tus salidas, rutas o viajes en vehículo, permitiendo consultarlas y compartirlas en cualquier momento.

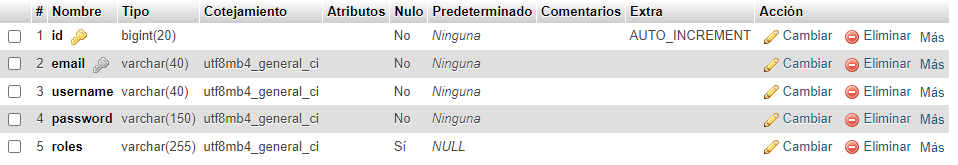
La idea principal es no tener la preocupación de encontrar o localizar de nuevo algún lugar o carretera por la cual has conducido, además de obtener un análisis de cada una de las rutas, un análisis general semanal, mensual etc.  
Pensada principalmente para el público motero, que sale a hacer rutas a la aventura sin un destino concreto y quieren tener un registro de esta.

Para ello, he creado esta API REST, la cual será implementada en un futuro, principalmente planteada para una App móvil para teléfonos Android. De momento, cubre las necesidades básicas en esta primera creación. Con el tiempo, se irá desarrollando e implementando nueva lógica de negocio, para hacerla más segura y robusta.

# Descripción de las tablas

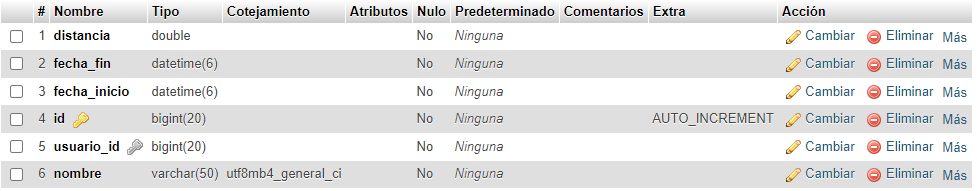
## Tabla Usuario

Contiene la información básica del usuario para poder registrarse. Se comunica con la tabla de Rutas mediante una relación uno a muchos que se verá mas adelante.

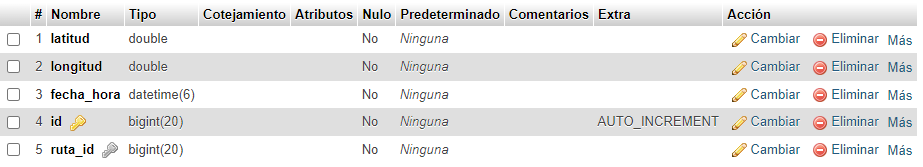


## Tabla Rutas

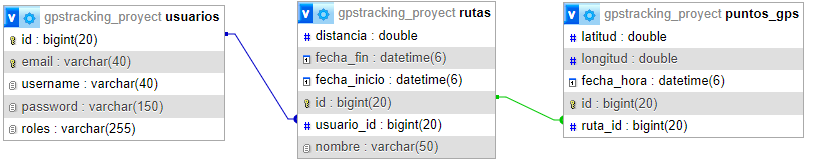
Esta tabla contiene la información esencial de cada ruta, como el nombre de la ruta, fechas y distancia. Además, está relacionada con la tabla Usuarios, almacenando la id del usuario al que pertenece cada ruta.  
El campo usuario le pongo @JsonIgnore en el Model del código, para que a la hora de realizar la petición no se muestre en bucle.



## Tabla Puntos\_GPS

Esta tabla contiene la información de cada punto que se obtiene del GPS del dispositivo. Está relacionada con la tabla Rutas, para asignarle la ID de la ruta a la que pertenece cada punto.

# Diagrama E-R



Como se puede observar, la relación entre estas 3 tablas es básicamente, de 1 a muchos.

Un usuario tiene muchas rutas, y una ruta concreta solo pertenece a un usuario.

Una ruta tiene muchos puntos GPS, pero un punto GPS solo pertenece a una ruta en concreto.

# Endpoints

## Tabla usuarios

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

* POST /usuarios -> Registra un nuevo usuario proporcionando un correo, contraseña y nombre de usuario. Responde con error si el usuario ya existe.
* GET /usuarios -> Obtiene un listado de la información de todos los usuarios.
* GET /usuarios/{id} -> Obtiene la información de un usuario por su ID. Solo puede acceder si el rol es **ADMIN** o el propio usuario.
* PUT /usuarios/{id} -> Actualiza la información de un usuario
* DELETE /usuarios/{id} -> Borra un usuario a través de su ID y todas sus rutas asociadas. Responde con error si el usuario no existe.

## Tabla rutas

Tabla

Descripción generada automáticamente

* POST /rutas -> Registra una nueva ruta asociada al usuario autenticado. ...
* GET /rutas -> Obtiene un listado de todas las rutas creadas por un usuario mediante su ID.
* GET /rutas/{id} -> Obtiene los detalles de una ruta específica mediante su ID.
* PUT /rutas/{id} -> Permite actualizar la información de una ruta, como el nombre
* DELETE /rutas/{id} -> Elimina una ruta mediante su ID, del usuario autenticado.

## Tabla puntos\_gps

Tabla

Descripción generada automáticamente con confianza media

* POST /puntos\_gps/{rutaId} -> Permite crear un nuevo punto GPS, que es pasado en el cuerpo de la petición, y lo añade a la Id de la ruta que se le pasa en la petición.
* GET /puntos\_gps/{rutaId} -> Obtiene todos los puntos asociados a una ruta específica por su ID.
* PUT /puntos\_gps/{puntoId} -> Actualiza un punto GPS mediante su ID. Se pasa el nuevo punto en el cuerpo de la petición.
* DELETE /puntos\_gps/{puntoId} -> Elimina un punto GPS mediante su ID.

# Lógica de negocio de la API

La lógica de negocio de esta API REST, está diseñada para gestionar rutas y puntos GPS asociados a estas rutas, asegurando que los usuarios puedan registrar, acceder, modificar y eliminar rutas y puntos GPS de manera segura y eficiente.

También se implementa una serie de reglas de acceso y validaciones que permiten asegurar que los usuarios solo puedan realizar operaciones sobre los recursos que les pertenecen o tienen permisos para gestionar.

Aspectos clave de la lógica de negocio (todos estos métodos pueden ser usados por el propio usuario, o por un administrador que siempre tiene acceso a todas las funciones):

## Gestión de usuarios

### Creación de usuarios

Un nuevo usuario puede registrarse en el sistema. Comprobaciones:

* Se comprueba que los campos no estén vacíos (nombre de usuario, email y contraseña).
* Se valida que la contraseña tenga al menos 8 caracteres.
* Se verifica que el nombre de usuario y el email no existan previamente en la base de datos. Si ya existen, se lanza una excepción **AlreadyExistsException**. Se codifica la contraseña antes de guardarla en la base de datos para garantizar la seguridad.

### Autenticación de usuarios

El sistema utiliza Spring Security para manejar la autenticación. El método **loadUserByUsername** permite que Spring Security obtenga los detalles de un usuario por su nombre de usuario para realizar el login. Se busca al usuario en la base de datos usando el nombre de usuario y se devuelve un **UserDetails** con los datos del usuario (como nombre de usuario, contraseña y roles).

### Obtener todos los usuarios

El método **getAllUsers** permite recuperar todos los usuarios registrados en el sistema. Este endpoint, está restringido solo a administradores, ya que proporciona acceso a la lista completa de usuarios.

### Obtener usuario por ID

Un usuario puede ver su información.

Si el usuario autenticado es el mismo que el que se quiere obtener (o ADMIN), se permite la operación.

### Actualizar usuario

Un usuario puede actualizar su información.

Se valida que los datos proporcionados para la actualización sean válidos y no generen problemas (por ejemplo, nombre de usuario o email ya registrados o contraseña no cumple con la longitud requerida). Si el usuario autenticado intenta actualizar su propia información o si es un administrador, la operación se permite.

### Eliminar usuario

Un usuario puede eliminar su cuenta.

Solo se permite si el usuario autenticado es el mismo que el usuario que se va a eliminar, o si el usuario autenticado tiene rol de administrador.

## Gestión de rutas

### Creación de rutas

Un usuario puede crear nuevas rutas.

Cada ruta debe tener un nombre único para evitar conflictos. Primero se valida que el nombre de la ruta sea válido, se comprueba si existe, y se inserta en la base de datos, asociado al usuario concreto.

### Obtención de rutas

Un usuario puede acceder únicamente a las rutas que le pertenecen.

Primero se valida que el usuario autenticado tenga acceso a la ruta solicitada. Si es así, se le devuelve la ruta, si no, se lanza una excepción **ForbiddenException**

### Actualización de rutas

Un usuario puede actualizar las rutas que le pertenece.

Primero se hacen las validaciones básicas, como comprobar nombre de la ruta que no esté vacío ni ya exista. Luego se actualiza con el nuevo nombre, ya que es lo único que se permite actualizar, y se guarda en la base de datos.

### Eliminación de rutas

Un usuario solo puede eliminar rutas que le pertenecen.

Se verifica que la ruta que el usuario quiera eliminar le pertenezca. Si es así, se elimina de la base de datos.

## Gestión de puntos GPS

### Creación de puntos

Un punto GPS debe ser asociado a una ruta específica, la cual debe pertenecer al usuario que lo crea.

Primero, el usuario proporciona la ID de la ruta a la que desea agregar el punto GPS. El sistema valida que esta ruta exista y que sea del propio usuario. Si es así, se crea el punto GPS, se asocia a dicha ruta, y se guarda en la base de datos.

### Obtención de puntos

Un usuario solo puede obtener los puntos GPS de las rutas que le pertenecen.

Se verifica que la ruta solicitada pertenezca al usuario autenticado. Si es válida, se devuelve la lista de puntos GPS que estén asociados a dicha ruta.

### Actualización de puntos

Un usuario puede actualizar los puntos GPS de las rutas que le pertenecen.

Primero el usuario proporciona el ID del punto GPS que desea actualizar. Luego se verifica que el punto GPS pertenece a una ruta del usuario. Si esto se cumple, se actualizan los campos del punto GPS con la nueva información proporcionada.

### Eliminación de puntos

Un usuario solo puede eliminar puntos GPS de las rutas que le pertenecen.

El usuario proporciona el ID del punto GPS que desea eliminar. Se verifica que el punto GPS pertenece a una ruta del usuario que hace la petición. Si la ruta es válida y el punto GPS pertenece a esa ruta, el punto GPS se elimina de la base de datos.

# Excepciones y códigos de estado

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media

# Restricciones de seguridad aplicadas

Se han implementado las siguientes restricciones de seguridad en la **API REST**:

* **Autenticación:** Todas las rutas de la API requieren que el usuario esté autenticado. Utilizamos el sistema de autenticación proporcionado por Spring Security para verificar que las credenciales sean correctas.
* **Ruta protegida:** Cada vez que un usuario intenta acceder a una ruta o punto GPS, se verifica que el usuario esté autenticado.
* **Excepción:** Si el usuario no está autenticado o sus credenciales son incorrectas, se lanza una excepción **UnauthorizedException** con el código de estado **401**.
* **Autorización:** Una vez que un usuario esté autenticado, se verifica si tiene permisos para acceder a los recursos solicitados.
* **Restricción de acceso:** Solo los usuarios propietarios de una ruta pueden acceder o modificar los puntos GPS asociados. Los usuarios no pueden acceder a recursos de otros usuarios.
* **Excepción:** Si un usuario intenta acceder a un recurso que no le pertenece, se lanza una excepción **ForbiddenException** con el código de estado **403**.
* **Roles de Administrador:** Los administradores tienen permisos especiales para acceder y gestionar recursos que pertenecen a otros usuarios.
* **Autorización para administradores:** Los administradores pueden acceder a cualquier recurso, incluso si no les pertenece, ya que tienen permisos globales.
* **Verificación de rol:** Antes de realizar cualquier operación, se verifica si el usuario es administrador. Si es así, se le otorgan permisos adicionales para realizar acciones como ver, modificar o eliminar recursos de otros usuarios.

## Archivo config

Enlace para ver de forma directa el contenido del [archivo de configuración](https://github.com/agadrian/Proyecto_GpsTracking/blob/master/src/main/kotlin/com/es/proyectgoDinAda_GPS_Tracking/security/SecurityConfig.kt) de seguridad de la API REST.

# Pruebas de los Endpoints con Insomnia

Antes de especificar en las rutas, dejar claro que, en cualquiera de ellas, se está controlando lo que se introduce como id en la petición. No lo pondré en todas porque sería repetir lo mismo una y otra vez. Ejemplo:

Captura de pantalla de un celular

Descripción generada automáticamenteRuta con id inexistente:

Ruta con id que no son números:Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamenteRuta con id negativa, obviamente no válida:

Ahora sí, pasamos a los métodos específicos.

## Registro de Usuario (POST /usuarios/register).

* Se envía un body con los parámetros del usuario correctamente:

Captura de pantalla de un celular

Descripción generada automáticamente

Devuelve la contraseña como nula para mayor seguridad (en realidad se inserta bien en la base de datos):



* Nombre, contraseña o email vacíos:

Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

* Contraseña no cumple con la longitud requerida (+8):

Captura de pantalla de un celular

Descripción generada automáticamente

* Usuario o email ya existentes:

Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

## Login de Usuario (POST /usuarios/login).

* Login con los datos correctos, devuelve el token:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

* Datos incorrectos (username, password):

Captura de pantalla de un celular

Descripción generada automáticamenteCaptura de pantalla de un celular

Descripción generada automáticamente

* Sin cuerpo

Captura de pantalla de un celular

Descripción generada automáticamente

## GetUserById (GET /usuarios/{id}).

* Con la ID y el token del usuario 1:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Sin token:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Con token de usuario 1 intentando acceder a usuario 2:

Captura de pantalla de un celular

Descripción generada automáticamente

* Con el token de un usuario con rol ADMIN:

Captura de pantalla de un celular

Descripción generada automáticamenteCaptura de pantalla de un celular

Descripción generada automáticamente

## GetAllUsers (GET /usuarios/)

* Con el token de un usuario ADMIN:

Texto

Descripción generada automáticamente

* Con el token de un usuairo USER:

Captura de pantalla de un celular

Descripción generada automáticamente

* Sin token o un token erróneo:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

## UpdateUserById (PUT /usuarios/{id}).

* Con el token del usuario 1 y el body con datos válidos:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* Captura de pantalla de un celular

  Descripción generada automáticamenteCon el token de usuario 2 y el body con datos válidos:
* Captura de pantalla de un celular

  Descripción generada automáticamenteCaptura de pantalla de un celular

  Descripción generada automáticamenteCon el token del usuario 1 y el body con datos inválidos (nombre o email que ya existan):
* Con el token de un usuario ADMIN a cualquier ID:

Captura de pantalla de un celular

Descripción generada automáticamente

## DeleteUserByid (DELETE /usuarios/{id})

* Token del usuario 3 intentando eliminar el usuario 1:

Captura de pantalla de un celular

Descripción generada automáticamente

* Token del usuario 3 intentando eliminar el usuario 3:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Token de usuario ADMIN (ID 4) eliminando cualquiera:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Registro de rutas (POST /rutas/)

* Con el token del usuario 1:

Captura de pantalla de un celular

Descripción generada automáticamente

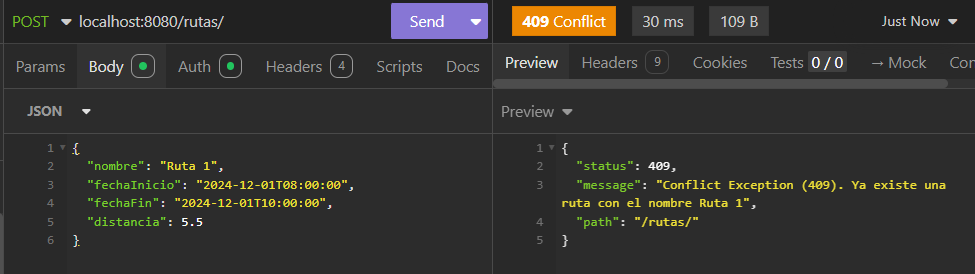
Se inserta con la id del usuario que lo crea, 1 en este caso:

* Sin token:

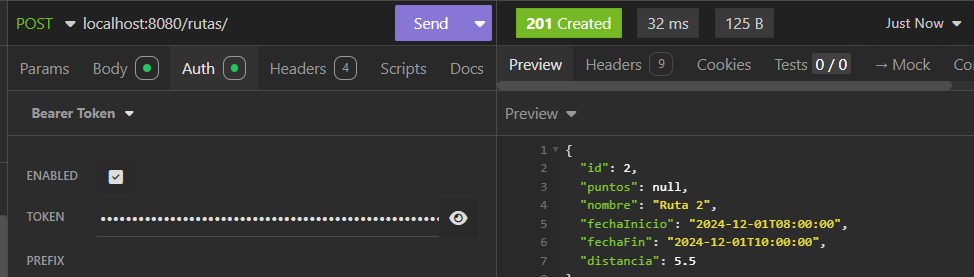
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

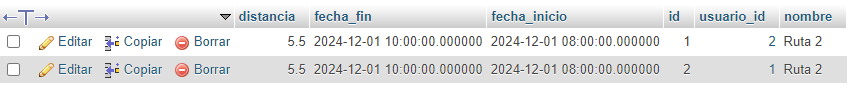
* Con token del usuario 1 y con el mismo nombre de ruta que otra existente:



* Con el token de otro usuario y el mismo nombre de ruta que otra que tenga otro usuario:



Deja crearla ya que se comprueba que el nombre de ruta no sea el mismo, pero para las rutas de cada usuario, no en general.



## GetRutaById (GET /rutas/{id})

* Con el token del usuario 1 obteniendo una ruta suya:

Captura de pantalla de un celular

Descripción generada automáticamente

* Interfaz de usuario gráfica, Aplicación

  Descripción generada automáticamenteCon el token del usuario 1 obteniendo una ruta que no es suya:
* Con el token de un usuario ADMIN:Captura de pantalla de un celular

  Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

## GetAllRutas (GET /rutas/)

* Captura de pantalla de un celular

  Descripción generada automáticamenteCon el token del usuario 1:

Devuelve todas sus rutas, en este caso solo tiene 1.

Cualquiera que lo haga obtiene su listado. No hay posible mensaje de fallo, únicamente si se hace sin token:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## UpdateRutaById (PUT /rutas/{id})

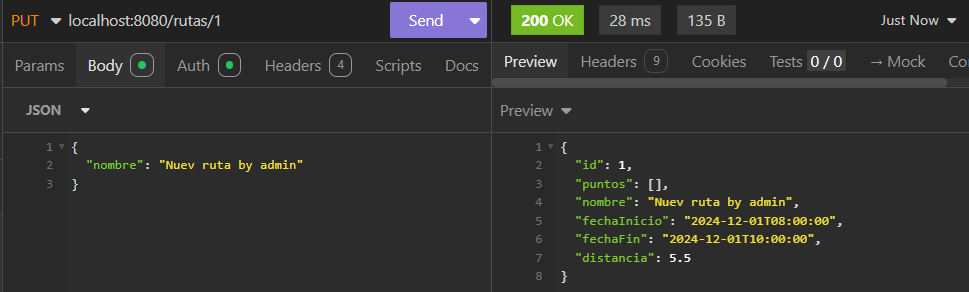
Tengo configurado para que solo se le pueda cambiar el nombre, por tanto veamos.

* Usando el token del usuario 1 y la id 2 (que es la suya) en la petición:

Captura de pantalla de un celular

Descripción generada automáticamente

* Usando el token del usuario 1 y la id1 (que no es la suya) en la petición:Captura de pantalla de un celular

  Descripción generada automáticamente
* Captura de pantalla de un celular

  Descripción generada automáticamenteCon el token de ADMIN y a cualquier id en la petición:

## DeleteRutaById (DELETE /rutas/{id})

* Con el token del usuario 1 borrando la ruta 1 que no es suya:

Captura de pantalla de un celular

Descripción generada automáticamente

* Con el token del usuario 1 borrando la ruta 2 que es suya:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El usuario con rol ADMIN al igual que anteriormente también puede eliminar a cualquier persona, no lo pongo todo de nuevo para no repetir lo mismo.

## Crear punto GPS (POST /puntos\_gps/{ruta\_Id})

* Crear un punto GPS con el token del usuario 1, y pasando la id de la ruta1, que es la ruta de este usuario en este caso:



Captura de pantalla de un celular

Descripción generada automáticamente

* Crear un punto GPS con el token del usuario 1, y pasando la id de la ruta 2, que no es la ruta de este usuario en este caso:

Captura de pantalla de un celular

Descripción generada automáticamenteInterfaz de usuario gráfica

Descripción generada automáticamente con confianza media

## Obtener los puntos GPS de una ruta (GET puntos\_gps/{ruta\_Id})

* Con el token del usuario 1 y la id 1 que es de una ruta suya:

Texto

Descripción generada automáticamente

* Con token del usuario 1 pero id de ruta que no es suya:

Captura de pantalla de un celular

Descripción generada automáticamente

## UpdatePuntoGPSById (PUT /puntos\_gps/{punto\_Id})

* Con el token del usuario 1 a una id de punto gps suyo:

Captura de pantalla de un celular

Descripción generada automáticamente

* Con el token del usuario 1, y la id del punto GPS de una ruta de otro usuario:

Captura de pantalla de un celular

Descripción generada automáticamente

* Con el token del usuario 1 y la id 36, que es un punto de otra ruta que pertenece a el usuario 1 también:



Captura de pantalla de un celular

Descripción generada automáticamente

## DeletePuntoGPSById (DELETE /puntos\_gps/{punto\_Id})

* Con el token del usuario 1 y la id de un punto que pertenece a una de sus rutas:

Captura de pantalla de un celular

Descripción generada automáticamente

* Con el token del usuario 1 y la id de un punto que no pertenece a una de sus rutas:

Captura de pantalla de un celular

Descripción generada automáticamente

# Tecnologías usadas

## Independencias

He utilizado las siguientes dependencias en mi proyecto:

* **spring-boot-starter-web**: para crear servicios RESTful utilizando Spring Boot.
* **spring-boot-starter-data-jpa**: para interactuar con la base de datos a través de JPA (Java Persistence API).
* **spring-boot-starter-oauth2-resource-server**: para manejar la seguridad de la API mediante OAuth2, protegiendo recursos del servidor.
* **jackson-module-kotlin**: para serializar y deserializar objetos Kotlin a JSON y viceversa.
* **spring-boot-devtools (developmentOnly)**: para acelerar el desarrollo, proporcionando recarga automática y otras herramientas útiles.
* **mysql-connector-j (runtimeOnly)**: para establecer conexiones con la base de datos MySQL.
* **spring-boot-configuration-processor (annotationProcessor):** para generar metadatos de configuración automáticamente.
* **spring-boot-starter-tomcat (providedRuntime)**: para usar Tomcat como servidor de aplicaciones embebido durante el despliegue.

Texto

Descripción generada automáticamenteEn el proyecto tengo alguna más, pero no han sido usadas de momento explícitamente:

## Software usado

* **IntelliJ IDEA**: como IDE para escribir el código en Kotlin y gestionar dependencias mediante Gradle
* **Insomnia**: para probar las peticiones HTTP y validar el correcto funcionamiento de los endpoints REST.
* **XAMPP**: para arrancar un servidor local de MySQL y realizar pruebas con la base de datos.

## Descripción y propósito tecnologías usadas

* **Spring Boot**: framework para construir aplicaciones basadas en microservicios. En este proyecto, lo usé para crear la API REST que gestiona la lógica del negocio y expone los endpoints creados.
* **Spring Security (OAuth2)**: para proteger los recursos de la API, implementando autenticación y autorización.
* **Kotlin**: lenguaje de programación principal, elegido por su sintaxis concisa, seguridad y capacidad de interoperar con Java.
* **Jackson**: para manejar la conversión entre objetos Kotlin y JSON en las solicitudes y respuestas.
* **MySQL**: sistema de gestión de bases de datos utilizado para almacenar los datos de la aplicación.
* **IntelliJ IDEA**: facilitó el desarrollo con herramientas avanzadas para Kotlin y Spring Boot.
* **Insomnia**: me permitió probar y verificar el comportamiento de los endpoints REST.
* **XAMPP**: fue útil para levantar un servidor MySQL local durante el desarrollo y pruebas.

# ¿Qué es una API REST?

Una API REST es una interfaz que permite la comunicación entre sistemas mediante HTTP, siguiendo principios que aseguran simplicidad, escalabilidad y eficiencia. Utiliza formatos como JSON para intercambiar datos.

## Principios básicos de una API REST

* **Cliente-Servidor:** Separación de responsabilidades entre cliente y servidor.
* **Sin estado :** Cada solicitud contiene toda la información necesaria para ser procesada.
* **Cacheable**: Las respuestas indican si pueden ser almacenadas en caché para optimizar el rendimiento.
* **Interfaz uniforme:** Uso de métodos **HTTP** estándar (GET, POST, PUT, DELETE) y **URIs** claros y concretos.
* **Arquitectura en capas:** Soporte para componentes intermedios como cachés o balanceadores de carga.

## Identificación de estos principios en mi implementación

* **Cliente-Servidor:** El cliente, en este caso yo mismo hago mis solicitudes HTTP con **Insomnia** hacia los endpoints del servidor, los cuales son manejados en los distintos **Controllers** y procesados en los **Services.**
* **Sin estado:** Cada solicitud incluye toda la información necesario para ser tratada correctamente, por lo que no se guarda ningún estado, cada llamada es independiente de la otra.
* **Interfaz uniforme:** Uso los métodos estándar **HTTP** (GET, POST, PUT, DELETE) en todos los Modelos de mi implementación. Lo hago de forma que la **URI** es clara, ya que es bastante explícita y con leerla se entiende con un simple vistazo.
* **Cacheable:** En este caso no ha sido implementada.
* **Arquitectura en capas:** La API ha sido distribuida en capas. Cuando el cliente hace una petición, se maneja de la siguiente forma:
  + Controlador -> Servicio -> Repositorio

Una vez el repositorio hace lo que se le pida, el proceso es al revés:

* + Repositorio -> Servicio -> Controlador

Y se le devuelve al cliente una respuesta.

# Ventajas de separación de responsabilidades

* **Escalabilidad:** Al separar cliente y servidor, cada componente puede escalarse de manera independiente según las necesidades de carga (por ejemplo, incrementar servidores para la API sin afectar la interfaz del cliente).
* **Mantenibilidad:** Facilita el mantenimiento y la actualización de cada componente sin interrumpir al otro. El cliente puede actualizar su diseño sin cambiar la lógica del servidor y viceversa.
* **Interoperabilidad:** Permite que diferentes clientes consuman la API del servidor sin necesidad de ser modificada para ello.
* **Reutilización del código**: La lógica del servidor puede ser utilizada por múltiples interfaces de cliente, lo que evita duplicar funcionalidades en diferentes plataformas.
* **Seguridad:** Centralizar la lógica de seguridad en el servidor permite un control más robusto y reduce el riesgo de vulnerabilidades en los clientes.