



API REST ROUTES TRACKER

ADRIÁN ARROYO GÓMEZ



16 DE DICIEMBRE DE 2024

I.E.S RAFAEL ALBERTI

ÍNDICE

Idea del proyecto.....	3
Justificación del proyecto	3
Descripción de las tablas	3
Tabla Usuario.....	3
Tabla Rutas	4
Tabla Puntos_GPS.....	4
Diagrama E-R.....	5
Endpoints.....	6
Tabla usuarios	6
Tabla rutas	7
Tabla puntos_gps	8
Lógica de negocio de la API.....	9
Gestión de usuarios	9
Creación de usuarios.....	9
Autenticación de usuarios	9
Obtener todos los usuarios.....	10
Obtener usuario por ID	10
Actualizar usuario.....	10
Eliminar usuario	10
Gestión de rutas	11
Creación de rutas.....	11
Obtención de rutas	11
Actualización de rutas.....	11
Eliminación de rutas.....	11
Gestión de puntos GPS	12
Creación de puntos	12
Obtención de puntos	12
Actualización de puntos.....	12
Eliminación de puntos.....	12
Excepciones y códigos de estado.....	13
Restricciones de seguridad aplicadas.....	14
Archivo config.....	14
Pruebas de los Endpoints con Insomnia	15
Registro de Usuario (POST /usuarios/register)	16

Login de Usuario (POST /usuarios/login)	18
GetUserById (GET /usuarios/{id})	19
GetAllUsers (GET /usuarios/)	21
UpdateUserById (PUT /usuarios/{id})	23
DeleteUserById (DELETE /usuarios/{id})	25
Registro de rutas (POST /rutas/)	26
GetRutaById (GET /rutas/{id})	28
GetAllRutas (GET /rutas/)	29
UpdateRutaById (PUT /rutas/{id})	30
DeleteRutaById (DELETE /rutas/{id})	31
Crear punto GPS (POST /puntos_gps/{ruta_id})	32
Obtener los puntos GPS de una ruta (GET puntos_gps/{ruta_id})	33
UpdatePuntoGPSById (PUT /puntos_gps/{punto_id})	34
DeletePuntoGPSById (DELETE /puntos_gps/{punto_id})	35
Tecnologías usadas	36
Dependencias	36
Software usado	37
Descripción y propósito tecnologías usadas	38
¿Qué es una API REST?	39
Principios básicos de una API REST	39
Identificación de estos principios en mi implementación	39
Ventajas de separación de responsabilidades	40

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 más adelante.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/> 1	id	bigint(20)			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
<input type="checkbox"/> 2	email	varchar(40)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 3	username	varchar(40)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 4	password	varchar(150)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 5	roles	varchar(255)	utf8mb4_general_ci		Sí	NULL			Cambiar Eliminar Más

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.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/> 1	distancia	double			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 2	fecha_fin	datetime(6)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 3	fecha_inicio	datetime(6)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 4	id	bigint(20)			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
<input type="checkbox"/> 5	usuario_id	bigint(20)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 6	nombre	varchar(50)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más

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.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/> 1	latitud	double			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 2	longitud	double			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 3	fecha_hora	datetime(6)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 4	id	bigint(20)			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
<input type="checkbox"/> 5	ruta_id	bigint(20)			No	Ninguna			Cambiar Eliminar Más

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

Método	Endpoint	Descripción
POST	/usuarios/register	Registrar nuevo usuario
POST	/usuarios/login	Hacer login con el usuario
GET	/usuarios/	Obtener todos los usuarios
GET	/usuario/{id}	Obtener usuario por ID
PUT	/usuarios/{id}	Actualizar usuario por ID
DELETE	/usuarios/{id}	Eliminar usuario por ID

- 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

Método	Endpoint	Descripción
POST	/rutas/	Registrar nueva ruta
GET	/rutas/	Obtener todas las rutas
GET	/rutas/{id}	Obtener ruta por ID
PUT	/rutas/{id}	Actualizar ruta por ID
DELETE	/rutas/{id}	Eliminar ruta por ID

- 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

Método	Endpoint	Descripción
POST	/puntos_gps/{rutald}	Registra y añade un nuevo punto GPS a una ruta
GET	/puntos_gps/{rutald}	Obtiene todos los puntos GPS de una ruta
PUT	/puntos_gps/{puntold}	Actualiza un punto GPS por su ID
DELETE	/puntos_gps/{puntold}	Elimina un punto GPS por su ID

- POST /puntos_gps/{rutald} -> 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/{rutald} -> Obtiene todos los puntos asociados a una ruta específica por su ID.
- PUT /puntos_gps/{puntold} -> Actualiza un punto GPS mediante su ID. Se pasa el nuevo punto en el cuerpo de la petición.
- DELETE /puntos_gps/{puntold} -> 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

Excepción	Código estado HTTP	Descripción	Mensaje
BadRequestException	400	Se lanza cuando la solicitud realizada por el cliente es incorrecta o no válida, como cuando faltan parámetros obligatorios en el cuerpo de la solicitud.	Bad request exception (400). + El mensaje concreto de cada caso
NotFoundException	404	Se lanza cuando no se encuentra un recurso solicitado, como cuando se intenta acceder a una ruta o un punto GPS que no existe.	Not Found Exception (404).+ El mensaje concreto de cada caso
ConflictException	409	Se lanza cuando se intenta crear o modificar un recurso que entra en conflicto con un recurso ya existente, como cuando se intenta registrar una ruta con un nombre que ya está en uso.	Conflict Exception (409). + El mensaje concreto de cada caso
AlreadyExistsException	409	Se lanza cuando se intenta crear o modificar un recurso que entra en conflicto con un recurso ya existente, como cuando se intenta registrar una ruta con un nombre que ya está en uso.	Conflict Exception (409). + El mensaje concreto de cada caso
UnauthorizedException	401	Se lanza cuando el usuario no está autenticado o no ha proporcionado credenciales válidas para acceder a un recurso.	Unauthorized Exception (401). + El mensaje concreto de cada caso
ForbiddenException	403	Se lanza cuando el usuario no tiene permisos suficientes para acceder a un recurso, incluso si está autenticado.	Forbidden Exception (403). + El mensaje concreto de cada caso
InternalServerError	500	Se lanza para capturar cualquier otro tipo de error no controlado por las excepciones anteriores. Esto puede ser un error de servidor inesperado.	InternalServerError (500). An unexpected error occurred (o algún otro personalizado en algún caso concreto)

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.

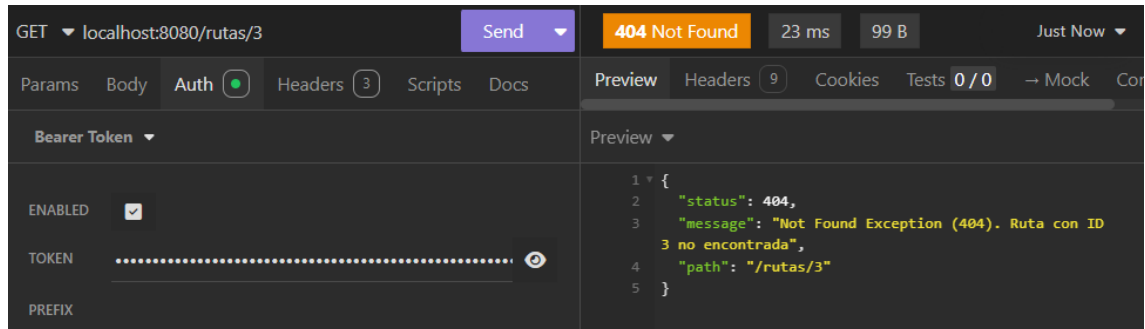
Archivo config

Enlace para ver de forma directa el contenido del [archivo de configuración](#) de seguridad de la API REST.

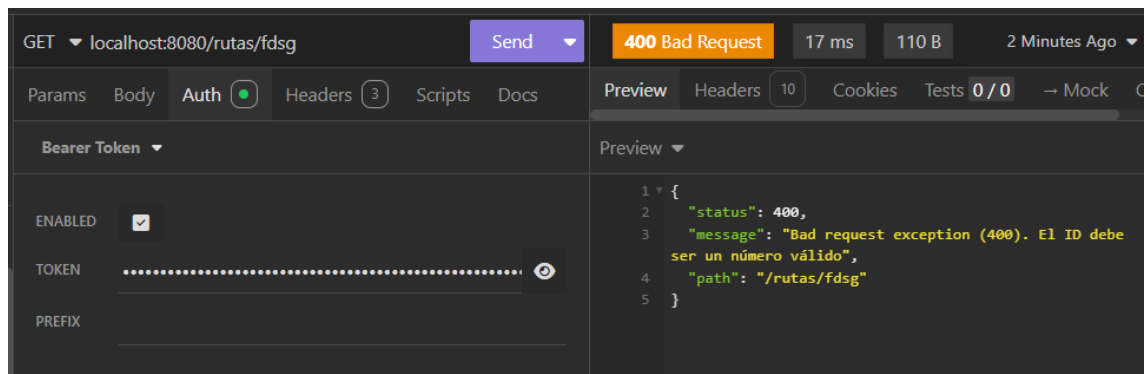
Pruebas de los Endpoints con Insomnia

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:

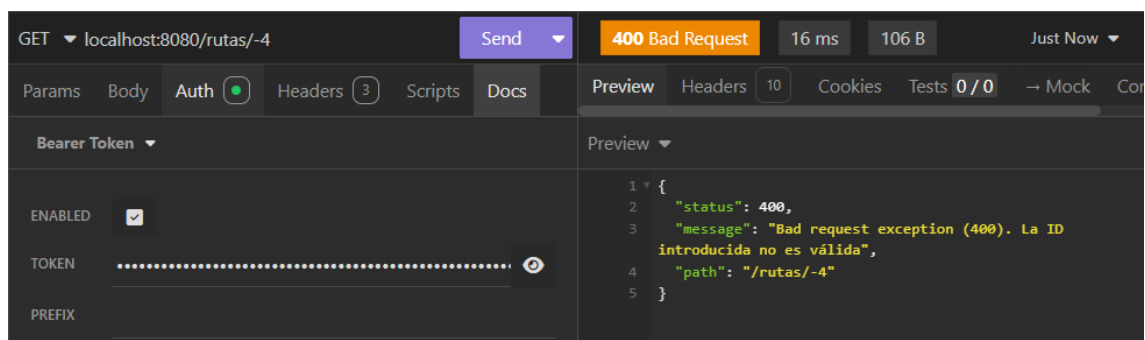
Ruta con id inexistente:



Ruta con id que no son números:



Ruta 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:

POST localhost:8080/usuarios/register Send 201 Created 640 ms 93 B Just Now

Params Body Auth Headers (4) Scripts Docs Preview Headers (9) Cookies Tests 0/0 → Mock Con

JSON

```
1 {
2   "username": "adri",
3   "password": "12341234",
4   "email": "adri@gmail.com",
5   "roles": "USER"
6 }
```

Preview

```
1 {
2   "id": 1,
3   "email": "adri@gmail.com",
4   "password": null,
5   "username": "adri",
6   "roles": "USER",
7   "rutas": []
8 }
```

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

	id	email	username	password	roles
<input type="checkbox"/>	1	adri@gmail.com	adri	\$2a\$10\$Xte10VyByDWDGdMFUEbDeYb3OYbmSvDHAMZa1viWIP...	USER

- Nombre, contraseña o email vacíos:

POST localhost:8080/usuarios/register Send 400 Bad Request 19 ms 126 B Just Now

Params Body Auth Headers (4) Scripts Docs Preview Headers (10) Cookies Tests 0/0 → Mock Con

JSON

```
1 {
2   "username": "",
3   "password": "12341234",
4   "email": "adri@gmail.com",
5   "roles": "USER"
6 }
```

Preview

```
1 {
2   "status": 400,
3   "message": "Bad request exception (400). El nombre
4     de usuario no puede estar vacío",
5   "path": "/usuarios/register"
6 }
```

POST localhost:8080/usuarios/register Send 400 Bad Request 40 ms 120 B Just Now

Params Body Auth Headers (4) Scripts Docs Preview Headers (10) Cookies Tests 0/0 → Mock Co

JSON

```
1 {
2   "username": "adri2",
3   "password": "",
4   "email": "adri@gmail.com",
5   "roles": "USER"
6 }
```

Preview

```
1 {
2   "status": 400,
3   "message": "Bad request exception (400). La
4     contraseña no puede estar vacía",
5   "path": "/usuarios/register"
6 }
```

POST localhost:8080/usuarios/register Send 400 Bad Request 44 ms 114 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 10 Cookies Tests 0/0 → Mock Cor

JSON Preview

```
1 {
2   "username": "adri2",
3   "password": "12341234",
4   "email": "",
5   "roles": "USER"
6 }
```

```
1 {
2   "status": 400,
3   "message": "Bad request exception (400). El email
4     no puede estar vacío",
5   "path": "/usuarios/register"
6 }
```

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

POST localhost:8080/usuarios/register Send 400 Bad Request 22 ms 129 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 10 Cookies Tests 0/0 → Mock Cor

JSON Preview

```
1 {
2   "username": "adri2",
3   "password": "1234",
4   "email": "adri2@gmail.com",
5   "roles": "USER"
6 }
```

```
1 {
2   "status": 400,
3   "message": "Bad request exception (400). La
4     contraseña debe tener mas de 8 caracteres",
5   "path": "/usuarios/register"
6 }
```

- Usuario o email ya existentes:

POST localhost:8080/usuarios/register Send 409 Conflict 21 ms 106 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

JSON Preview

```
1 {
2   "username": "adri",
3   "password": "12341234",
4   "email": "adri2@gmail.com",
5   "roles": "USER"
6 }
```

```
1 {
2   "status": 409,
3   "message": "Conflict Exception (409). El usuario
4     adri ya existe",
5   "path": "/usuarios/register"
6 }
```

POST localhost:8080/usuarios/register Send 409 Conflict 29 ms 126 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

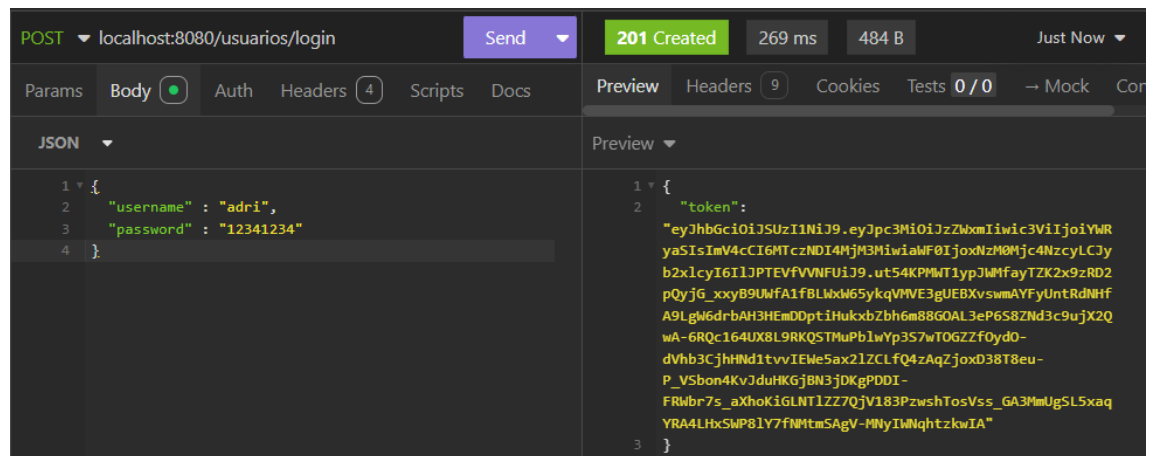
JSON Preview

```
1 {
2   "username": "adri2",
3   "password": "12341234",
4   "email": "adri@gmail.com",
5   "roles": "USER"
6 }
```

```
1 {
2   "status": 409,
3   "message": "Conflict Exception (409). El usuario
4     con email adri@gmail.com ya existe",
5   "path": "/usuarios/register"
6 }
```

Login de Usuario (POST /usuarios/login).

- Login con los datos correctos, devuelve el token:



POST localhost:8080/usuarios/login Send 201 Created 269 ms 484 B Just Now

Params Body Auth Headers (4) Scripts Docs Preview Headers (9) Cookies Tests 0/0 → Mock Cor

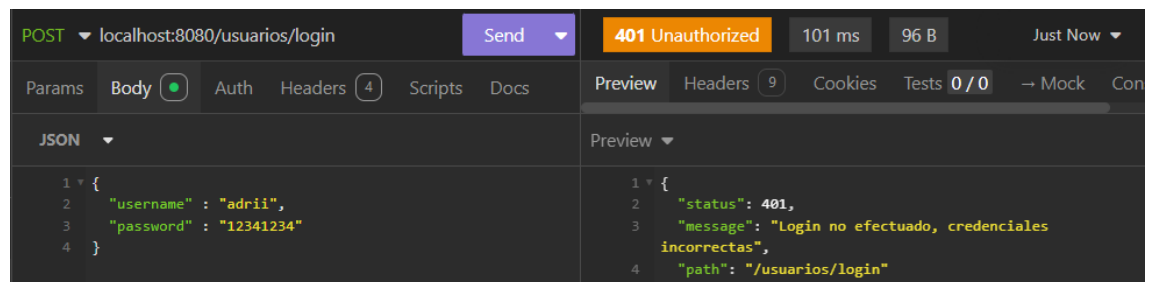
JSON

```
1 {
2   "username" : "adri",
3   "password" : "12341234"
4 }
```

Preview

```
1 {
2   "token":
3     "eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJzZWMiIiwic3ViIjoiYWRR
4     yaSIsImV4cCI6MTczNDI4MjM3MiwiaWF0IjoxNzA0Mjc4NzcyLCJy
5     b2xlc3I6IjE1PTEvVWVNFUI39.UT54KPMWt1ypJWt1fayTZK2x9zRD2
6     pQyJG_xxyB9UMfA1f8LWxW65ykqVMVE3gUEBXvswmAYfyUntRdHf
7     A9LgW6drbAH3HEmDDptIHukxbZbh6m88GOAL3eP6S8ZNd3c9urjX2Q
8     wA-6RQc164UX8L9RKQSTMuPblwYp3S7wTOGZZFoyd0-
9     dVhb3CjhHNd1tvvIEWe5ax2L7CLFQ4zAqZjoxD38T8eu-
10    P_VSbon4KvJduHKGj8N3jDKgPDDI-
11    FRWbr7s_aXhoKiGLNT1ZZ7QjV183PzswshTosVss_GA3MmUgSL5xaq
12    YRA4LHxSWP8LY7fNMtmSAgV-MNyIWNqhtzkWIA"
13 }
```

- Datos incorrectos (username, password):



POST localhost:8080/usuarios/login Send 401 Unauthorized 101 ms 96 B Just Now

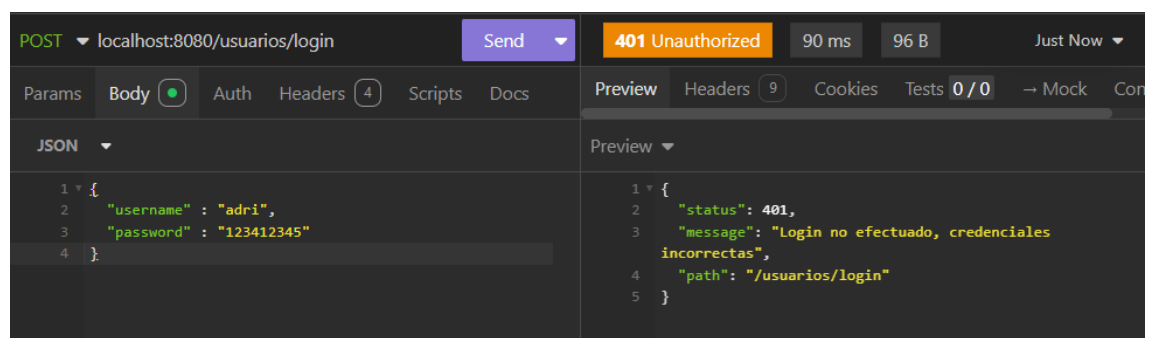
Params Body Auth Headers (4) Scripts Docs Preview Headers (9) Cookies Tests 0/0 → Mock Cor

JSON

```
1 {
2   "username" : "adrii",
3   "password" : "12341234"
4 }
```

Preview

```
1 {
2   "status": 401,
3   "message": "Login no efectuado, credenciales
4   incorrectas",
5   "path": "/usuarios/login"
6 }
```



POST localhost:8080/usuarios/login Send 401 Unauthorized 90 ms 96 B Just Now

Params Body Auth Headers (4) Scripts Docs Preview Headers (9) Cookies Tests 0/0 → Mock Cor

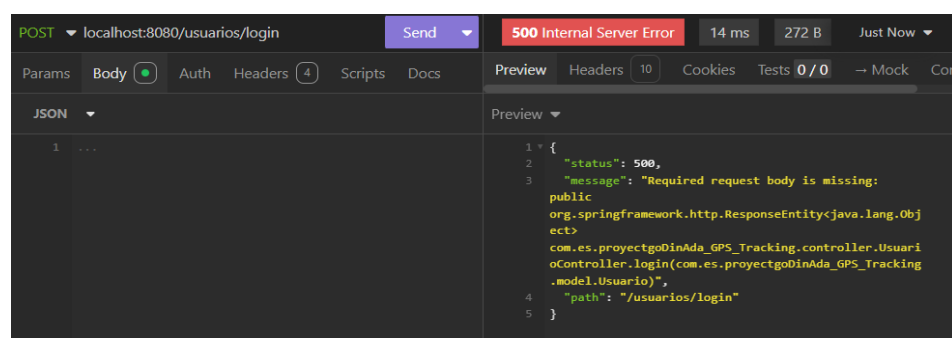
JSON

```
1 {
2   "username" : "adri",
3   "password" : "123412345"
4 }
```

Preview

```
1 {
2   "status": 401,
3   "message": "Login no efectuado, credenciales
4   incorrectas",
5   "path": "/usuarios/login"
6 }
```

- Sin cuerpo



POST localhost:8080/usuarios/login Send 500 Internal Server Error 14 ms 272 B Just Now

Params Body Auth Headers (4) Scripts Docs Preview Headers (10) Cookies Tests 0/0 → Mock Cor

JSON

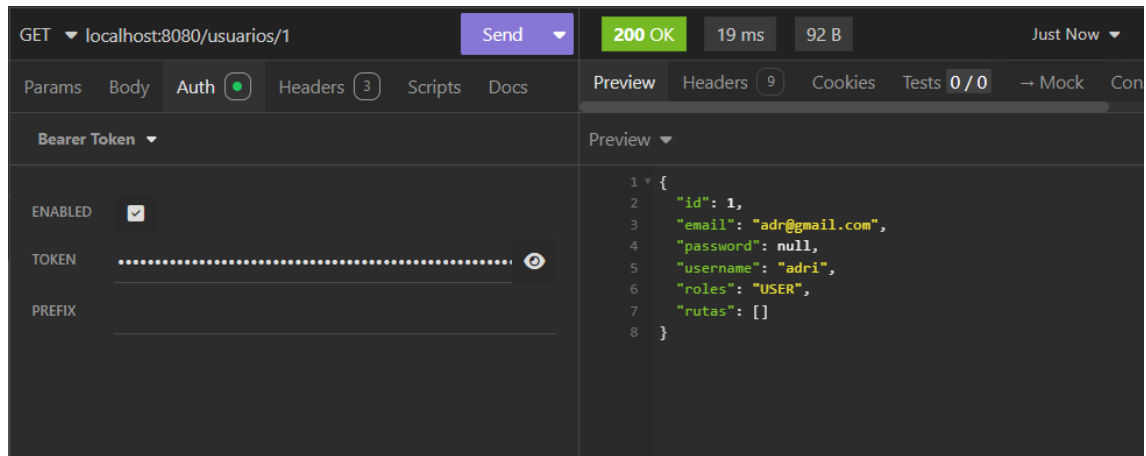
```
1 ...
```

Preview

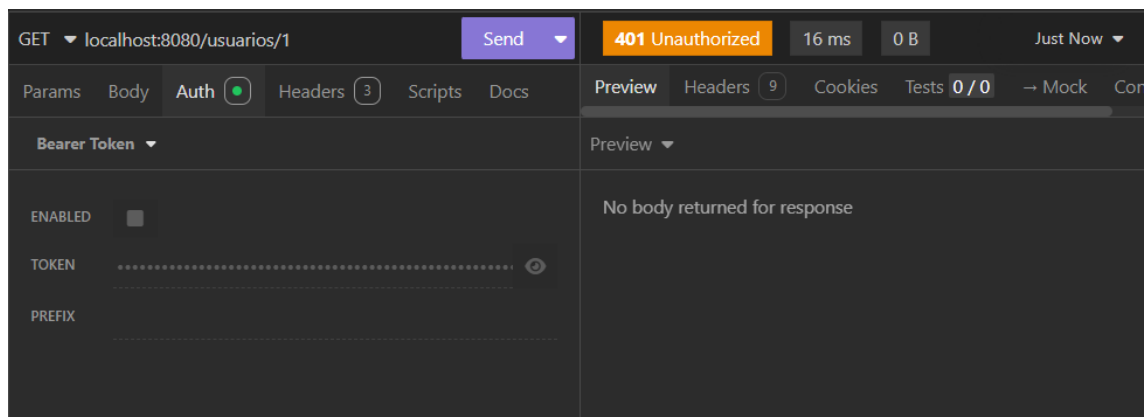
```
1 {
2   "status": 500,
3   "message": "Required request body is missing:
4     public
5     org.springframework.http.ResponseEntity<java.lang.Obj
6     ect>
7     com.es.proyectgoDinAda_GPS_Tracking.controller.Usuario
8     oController.login(com.es.proyectgoDinAda_GPS_Tracking
9     .model.Usuario)",
10   "path": "/usuarios/login"
11 }
```

GetUserById (GET /usuarios/{id}).

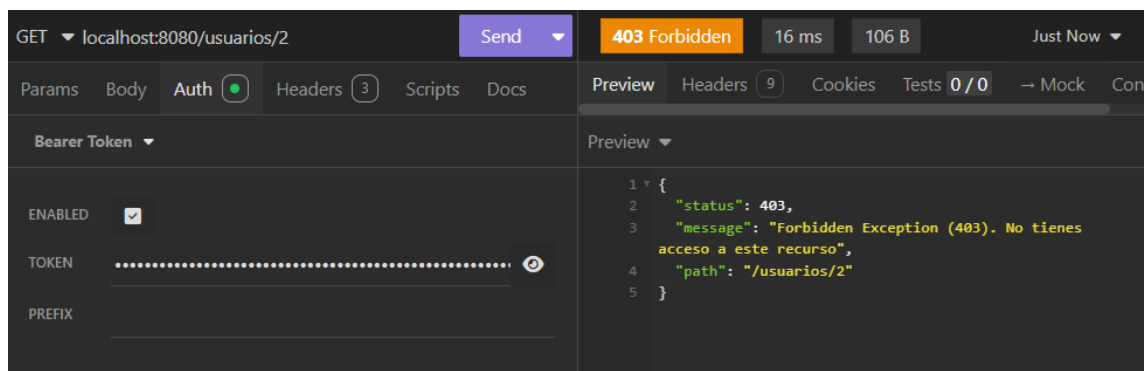
- Con la ID y el token del usuario 1:



- Sin token:



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



- Con el token de un usuario con rol ADMIN:

GET localhost:8080/usuarios/4 Send 200 OK 56 ms 96 B Just Now

Params Body Auth Headers 3 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock

Bearer Token

ENABLED ☒

TOKEN eyJhbGciOiJSUzI1Ni9.eyJpc3MiOiJzZWxmiwic3V

PREFIX

Preview

```
1 {
2   "id": 4,
3   "email": "admin@admin.com",
4   "password": null,
5   "username": "admin",
6   "roles": "ADMIN",
7   "rutas": []
8 }
```

GET localhost:8080/usuarios/2 Send 200 OK 22 ms 95 B Just Now

Params Body Auth Headers 3 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock

Bearer Token

ENABLED ☒

TOKEN eyJhbGciOiJSUzI1Ni9.eyJpc3MiOiJzZWxmiwic3V

PREFIX

Preview

```
1 {
2   "id": 2,
3   "email": "adri2@gmail.com",
4   "password": null,
5   "username": "adri2",
6   "roles": "USER",
7   "rutas": []
8 }
```

GetAllUsers (GET /usuarios/)

- Con el token de un usuario ADMIN:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8080/usuarios/
- Auth:** Bearer Token (ENABLED)
- Response Status:** 200 OK
- Response Time:** 28 ms
- Response Size:** 615 B
- Response Body (JSON):**

```
[
  {
    "id": 1,
    "email": "adr@gmail.com",
    "password": "$2a$10$dxpj7RGhf1peDyVwr5XyoeurChK8LUMSQ4nLwt0CSBo8BjF3vEGvm",
    "username": "adri",
    "roles": "USER",
    "rutas": []
  },
  {
    "id": 2,
    "email": "adri2@gmail.com",
    "password": "$2a$10$8rBMbuvH1XPtQ82ucu3dn.eav5VKN5cZUh49AYQGfPe.8CULCVPxq",
    "username": "adri2",
    "roles": "USER",
    "rutas": []
  },
  {
    "id": 3,
    "email": "adri3@gmail.com",
    "password": "$2a$10$HAq1EWh.mPPZap28nJuIEOmH1uXKVpD9K01bbhNnUI6nXGahVVple",
    "username": "adri3",
    "roles": "USER",
    "rutas": []
  }
]
```

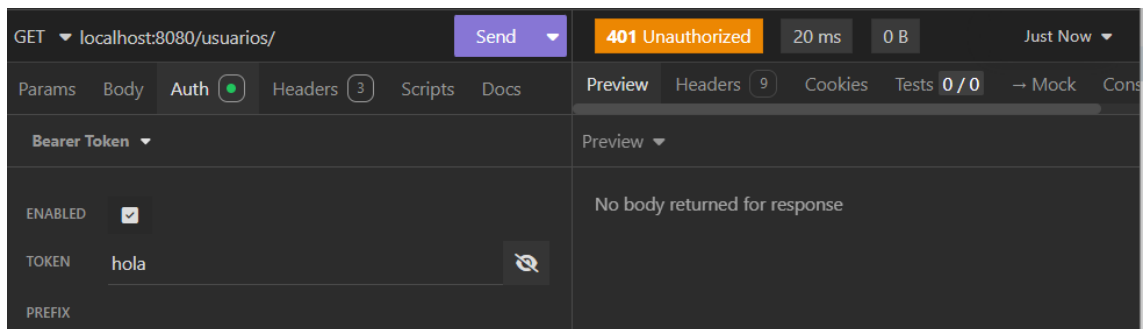
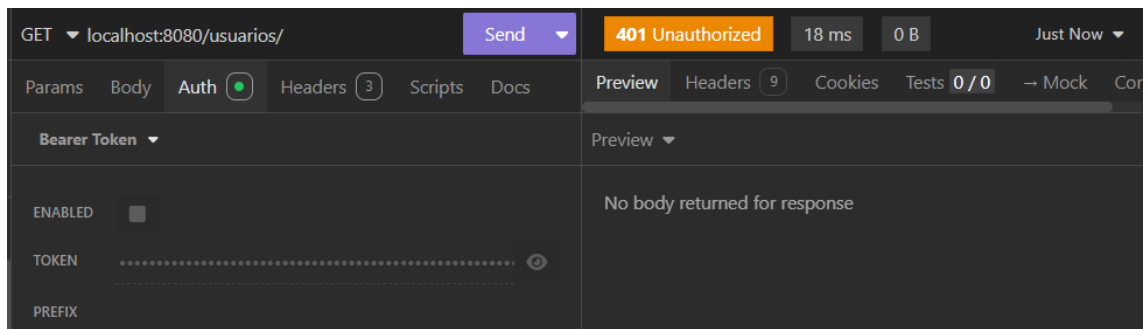
- Con el token de un usuario USER:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8080/usuarios/
- Auth:** Bearer Token (ENABLED)
- Response Status:** 403 Forbidden
- Response Time:** 20 ms
- Response Size:** 109 B
- Response Body (JSON):**

```
{
  "status": 403,
  "message": "Access Denied: No tienes permisos para acceder a este recurso.",
  "path": "/usuarios/"
}
```

- Sin token o un token erróneo:



UpdateUserById (PUT /usuarios/{id}).

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

PUT localhost:8080/usuarios/1 Send 200 OK 113 ms 164 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

JSON

```
1 {
2   "email": "adriUpdated@gmail.com",
3   "password": "12341234",
4   "username": "adrUpdated"
5 }
```

Preview

```
1 {
2   "id": 1,
3   "email": "adriUpdated@gmail.com",
4   "password":
5     "$2a$10$XVYUblW.RT3AIDd6fmmQboy/RDNckUuFoQ4UzyPHdGYsa
6     piDvzJd6",
7   "username": "adrUpdated",
8   "roles": "USER",
9   "rutas": []
10 }
```

- Con el token de usuario 2 y el body con datos válidos:

PUT localhost:8080/usuarios/1 Send 403 Forbidden 21 ms 106 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

JSON

```
1 {
2   "email": "adriUpdated2@gmail.com",
3   "password": "12341234",
4   "username": "adrUpdated2"
5 }
```

Preview

```
1 {
2   "status": 403,
3   "message": "Forbidden Exception (403). No tienes
4     acceso a este recurso",
5   "path": "/usuarios/1"
6 }
```

- Con el token del usuario 1 y el body con datos inválidos (nombre o email que ya existan):

PUT localhost:8080/usuarios/1 Send 409 Conflict 20 ms 114 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

JSON

```
1 {
2   "email": "adri2@gmail.com",
3   "password": "12341234",
4   "username": "adri2"
5 }
```

Preview

```
1 {
2   "status": 409,
3   "message": "Conflict Exception (409). El nombre de
4     usuario ya existe, use otro",
5   "path": "/usuarios/1"
6 }
```

PUT localhost:8080/usuarios/1 Send 409 Conflict 22 ms 137 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

JSON

```
1 {
2   "email": "adri2@gmail.com",
3   "password": "12341234",
4   "username": "adri2"
5 }
```

Preview

```
1 {
2   "status": 409,
3   "message": "Conflict Exception (409). El email ya
4     esta registrado en otra cuenta, por favor, use otro",
5   "path": "/usuarios/1"
6 }
```


- Con el token de un usuario ADMIN a cualquier ID:

The screenshot shows a REST client interface with a PUT request to `localhost:8080/usuarios/1`. The request body is a JSON object with email, password, and username. The response is a 200 OK status with a JSON object containing id, email, password, username, roles, and rutas.

```
PUT localhost:8080/usuarios/1
```

Send

200 OK 97 ms 166 B Just Now

Params Body Auth Headers 4 Scripts Docs

JSON

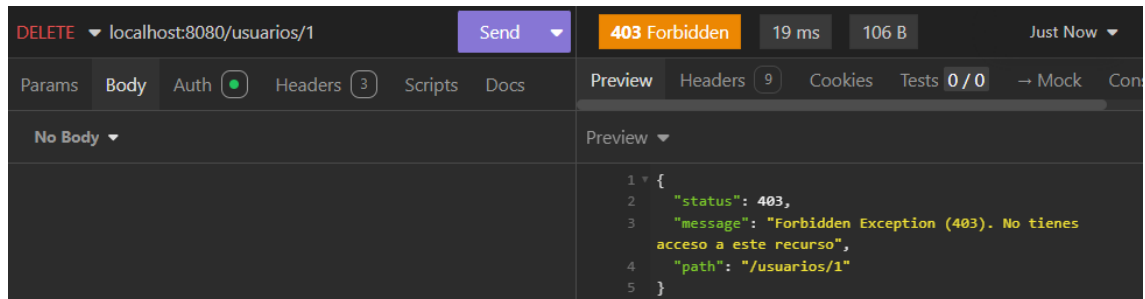
```
1 {
2   "email": "adri2ByAdmin@gmail.com",
3   "password": "12341234",
4   "username": "adriByAdmin"
5 }
```

Preview

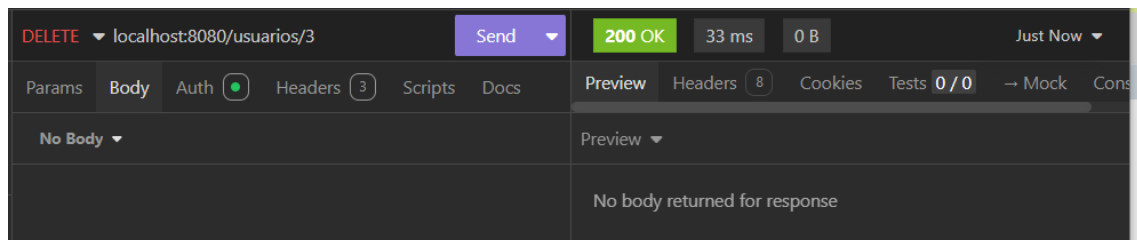
```
1 {
2   "id": 1,
3   "email": "adri2ByAdmin@gmail.com",
4   "password":
5     "$2a$10$6QStV9p2HnVPHpWCEq01e6AXkld6SGHXonWRao5MvaAY
6     FS3jFo/C",
7   "username": "adriByAdmin",
8   "roles": "USER",
9   "rutas": []
10 }
```

DeleteUserById (DELETE /usuarios/{id})

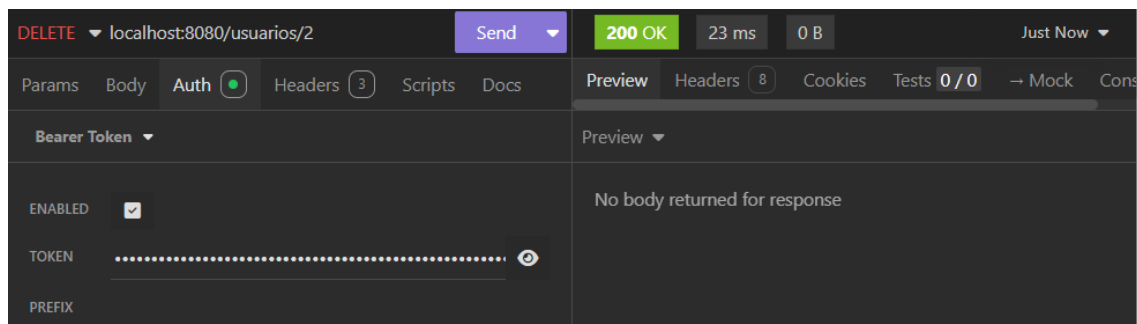
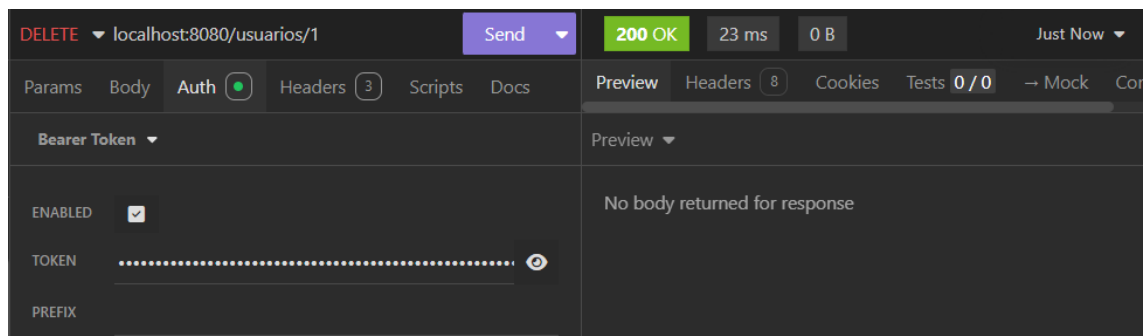
- Token del usuario 3 intentando eliminar el usuario 1:



- Token del usuario 3 intentando eliminar el usuario 3:

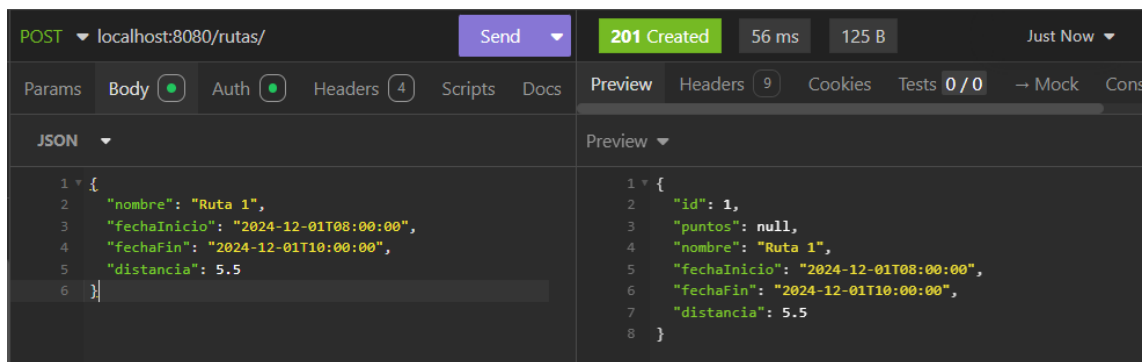


- Token de usuario ADMIN (ID 4) eliminando cualquiera:



Registro de rutas (POST /rutas/)

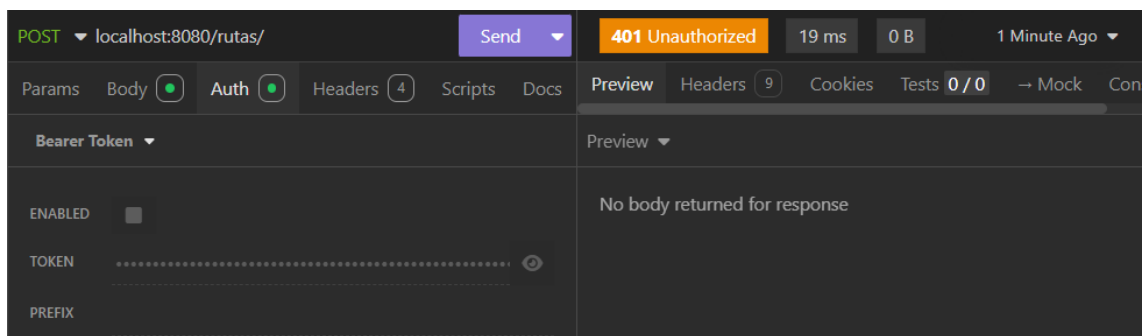
- Con el token del usuario 1:



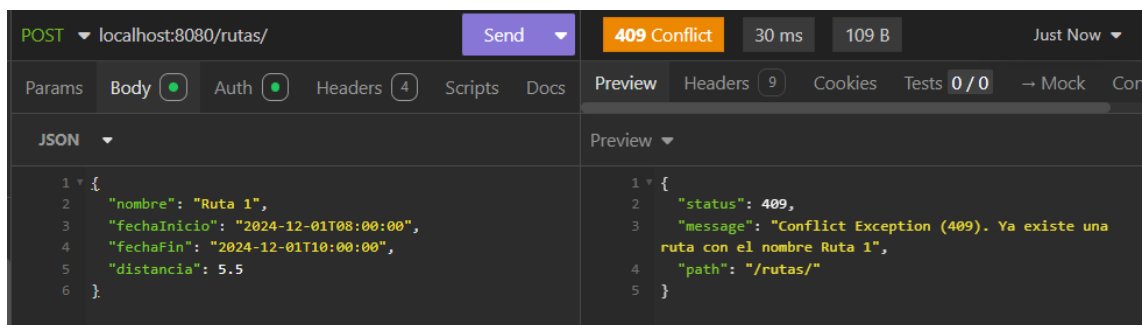
Se inserta con la id del usuario que lo crea, 1 en este caso:

	distancia	fecha_fin	fecha_inicio	id	usuario_id	nombre
<input type="checkbox"/>	5.5	2024-12-01 10:00:00.000000	2024-12-01 08:00:00.000000	1	1	Ruta 1

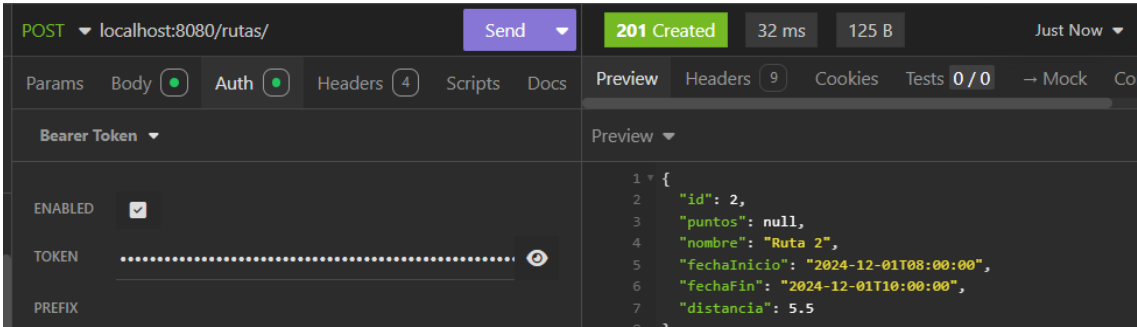
- Sin token:






- 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.

			distancia	fecha_fin	fecha_inicio	id	usuario_id	nombre
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	5.5	2024-12-01 10:00:00.000000	2024-12-01 08:00:00.000000	1	2 Ruta 2
<input type="checkbox"/>	Editar	Copiar	Borrar	5.5	2024-12-01 10:00:00.000000	2024-12-01 08:00:00.000000	2	1 Ruta 2

GetRutaById (GET /rutas/{id})

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

GET localhost:8080/rutas/2 Send 200 OK 29 ms 123 B 1 Minute Ago

Params Body Auth Headers (3) Scripts Docs

Bearer Token

ENABLED ☒

TOKEN

PREFIX

Preview

```
1 {
2   "id": 2,
3   "puntos": [],
4   "nombre": "Ruta 2",
5   "fechaInicio": "2024-12-01T08:00:00",
6   "fechaFin": "2024-12-01T10:00:00",
7   "distancia": 5.5
8 }
```

- Con el token del usuario 1 obteniendo una ruta que no es suya:

GET localhost:8080/rutas/1 Send 403 Forbidden 28 ms 100 B Just Now

Params Body Auth Headers (3) Scripts Docs

Bearer Token

ENABLED ☒

TOKEN

PREFIX

Preview

```
1 {
2   "status": 403,
3   "message": "Forbidden Exception (403). No tienes
4     acceso a esta ruta",
5   "path": "/rutas/1"
6 }
```

- Con el token de un usuario ADMIN:

GET localhost:8080/rutas/1 Send 200 OK 23 ms 123 B Just Now

Params Body Auth Headers (3) Scripts Docs

Bearer Token

ENABLED ☒

TOKEN

PREFIX

Preview

```
1 {
2   "id": 1,
3   "puntos": [],
4   "nombre": "Ruta 2",
5   "fechaInicio": "2024-12-01T08:00:00",
6   "fechaFin": "2024-12-01T10:00:00",
7   "distancia": 5.5
8 }
```

GET localhost:8080/rutas/2 Send 200 OK 20 ms 123 B Just Now

Params Body Auth Headers (3) Scripts Docs

Bearer Token

ENABLED ☒

TOKEN

PREFIX

Preview

```
1 {
2   "id": 2,
3   "puntos": [],
4   "nombre": "Ruta 2",
5   "fechaInicio": "2024-12-01T08:00:00",
6   "fechaFin": "2024-12-01T10:00:00",
7   "distancia": 5.5
8 }
```

GetAllRutas (GET /rutas/)

- Con el token del usuario 1:

GET localhost:8080/rutas/ Send 200 OK 17 ms 125 B Just Now

Params Body Auth Headers 3 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

Bearer Token

ENABLED ☒

TOKEN

PREFIX

Preview

```
1 [
2   {
3     "id": 2,
4     "puntos": [],
5     "nombre": "Ruta 2",
6     "fechaInicio": "2024-12-01T08:00:00",
7     "fechaFin": "2024-12-01T10:00:00",
8     "distancia": 5.5
9   }
10 ]
```

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:

GET localhost:8080/rutas/ Send 401 Unauthorized 15 ms 0 B Just Now

Params Body Auth Headers 3 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Cons

Bearer Token

ENABLED ☐

TOKEN

PREFIX

Preview

No body returned for response

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:

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8080/rutas/2
- Status:** 200 OK
- Time:** 36 ms
- Size:** 128 B
- Body (JSON):**

```
{  "nombre": "Nueva Rudta"}
```
- Preview (Response):**

```
{  "id": 2,  "puntos": [],  "nombre": "Nueva Rudta",  "fechaInicio": "2024-12-01T08:00:00",  "fechaFin": "2024-12-01T10:00:00",  "distancia": 5.5}
```

- Usando el token del usuario 1 y la id1 (que no es la suya) en la petición:

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8080/rutas/1
- Status:** 403 Forbidden
- Time:** 18 ms
- Size:** 100 B
- Body (JSON):**

```
{  "nombre": "Nueva Rudta"}
```
- Preview (Response):**

```
{  "status": 403,  "message": "Forbidden Exception (403). No tienes acceso a esta ruta",  "path": "/rutas/1"}
```

- Con el token de ADMIN y a cualquier id en la petición:

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8080/rutas/2
- Status:** 200 OK
- Time:** 27 ms
- Size:** 137 B
- Body (JSON):**

```
{  "nombre": "Nueva ruta by admin"}
```
- Preview (Response):**

```
{  "id": 2,  "puntos": [],  "nombre": "Nueva ruta by admin",  "fechaInicio": "2024-12-01T08:00:00",  "fechaFin": "2024-12-01T10:00:00",  "distancia": 5.5}
```

The screenshot shows a REST client interface with the following details:

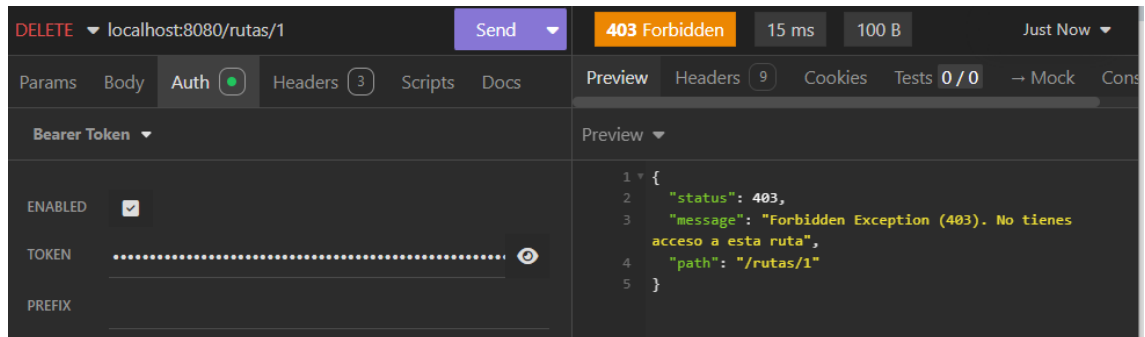
- Method:** PUT
- URL:** localhost:8080/rutas/1
- Status:** 200 OK
- Time:** 28 ms
- Size:** 135 B
- Body (JSON):**

```
{  "nombre": "Nuev ruta by admin"}
```
- Preview (Response):**

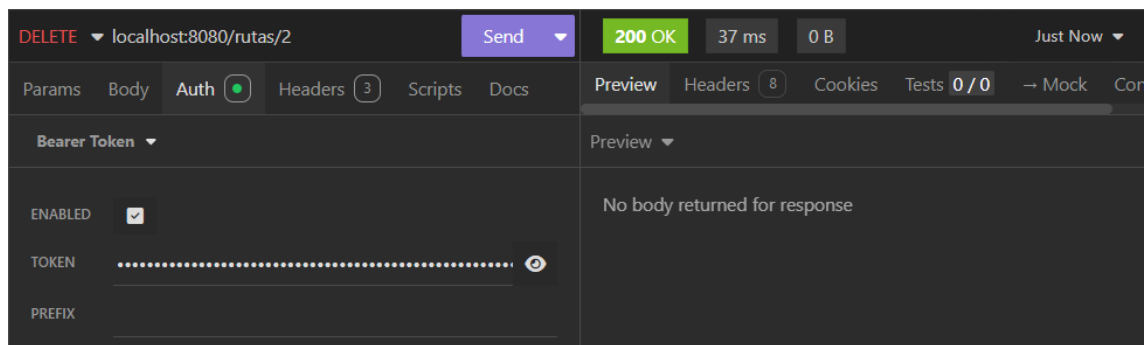
```
{  "id": 1,  "puntos": [],  "nombre": "Nuev ruta by admin",  "fechaInicio": "2024-12-01T08:00:00",  "fechaFin": "2024-12-01T10:00:00",  "distancia": 5.5}
```

DeleteRutaById (DELETE /rutas/{id})

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



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



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:

←T→

▼ distancia fecha_fin fecha_inicio id usuario_id nombre

☐ Editar Copiar Borrar

5.5 2024-12-01 10:00:00.000000 2024-12-01 08:00:00.000000 1 1 Ruta 1

POST localhost:8080/puntos_gps/1 Send 201 Created 57 ms 86 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Co

JSON Preview

```
1 {
2   "latitud": 40.7128,
3   "longitud": -74.0060
4 }
```

```
1 {
2   "id": 1,
3   "latitud": 40.7128,
4   "longitud": -74.006,
5   "fechaHora": "2024-12-15T18:46:34.620737"
6 }
```

- 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:

←T→

▼ distancia fecha_fin fecha_inicio id usuario_id nombre

☐ Editar Copiar Borrar

5.5 2024-12-01 10:00:00.000000 2024-12-01 08:00:00.000000 1 1 Ruta 1

☐ Editar Copiar Borrar

5.5 2024-12-01 10:00:00.000000 2024-12-01 08:00:00.000000 2 2 Ruta admin

☐ Editar Copiar Borrar

5.5 2024-12-01 10:00:00.000000 2024-12-01 08:00:00.000000 3 2 Ruta admin 2

POST localhost:8080/puntos_gps/2 Send 403 Forbidden 38 ms 105 B Just Now

Params Body Auth Headers 4 Scripts Docs Preview Headers 9 Cookies Tests 0/0 → Mock Co

JSON Preview

```
1 {
2   "latitud": 40.7128,
3   "longitud": -74.0060
4 }
```

```
1 {
2   "status": 403,
3   "message": "Forbidden Exception (403). No tienes
4   acceso a esta ruta",
5   "path": "/puntos_gps/2"
6 }
```

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:

GET localhost:8080/puntos_gps/1 Send 200 OK 28 ms 1690 B Just Now

Params Body Auth Headers 3 Scripts Docs

Bearer Token

ENABLED ☒

TOKEN

PREFIX

Preview

```
1 [
2   {
3     "id": 1,
4     "latitud": 40.7128,
5     "longitud": -74.006,
6     "fechaHora": "2024-12-15T18:46:34"
7   },
8   {
9     "id": 2,
10    "latitud": 0.0,
11    "longitud": -74.006,
12    "fechaHora": "2024-12-15T18:49:51"
13  },
14  {
15    "id": 3,
16    "latitud": 0.0,
17    "longitud": -74.006,
18    "fechaHora": "2024-12-15T18:49:52"
19  },
20  {
21    "id": 4,
22    "latitud": 0.0,
23    "longitud": -74.006,
24    "fechaHora": "2024-12-15T18:49:54"
25  },
26  {
27    "id": 5,
28    "latitud": 0.0,
29    "longitud": -74.006,
30    "fechaHora": "2024-12-15T18:49:55"
31  }
32 ]
```

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

GET localhost:8080/puntos_gps/2 Send 403 Forbidden 19 ms 105 B Just Now

Params Body Auth Headers 3 Scripts Docs

Bearer Token

ENABLED ☒

TOKEN

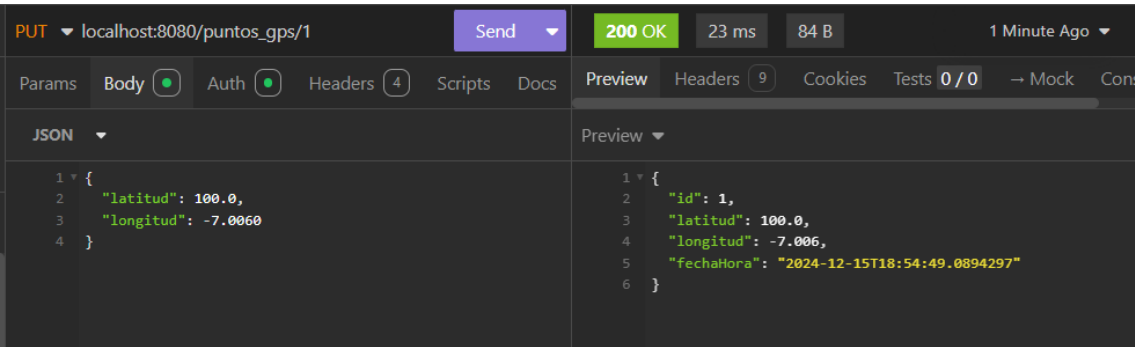
PREFIX

Preview

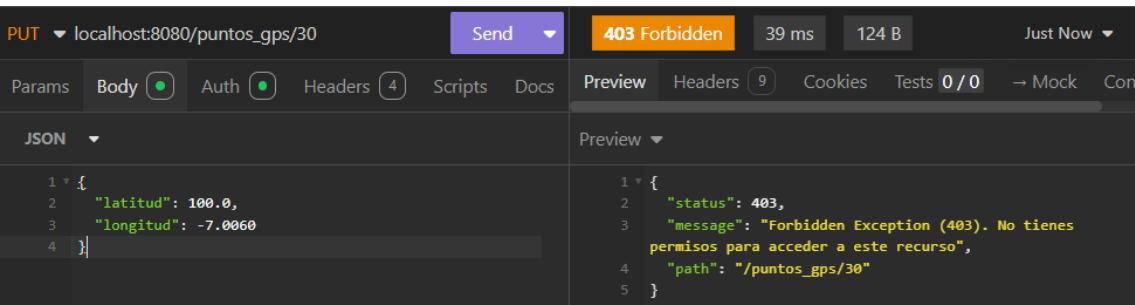
```
1 {
2   "status": 403,
3   "message": "Forbidden Exception (403). No tienes
4   acceso a esta ruta",
5   "path": "/puntos_gps/2"
6 }
```

UpdatePuntoGPSById (PUT /puntos_gps/{punto_Id})

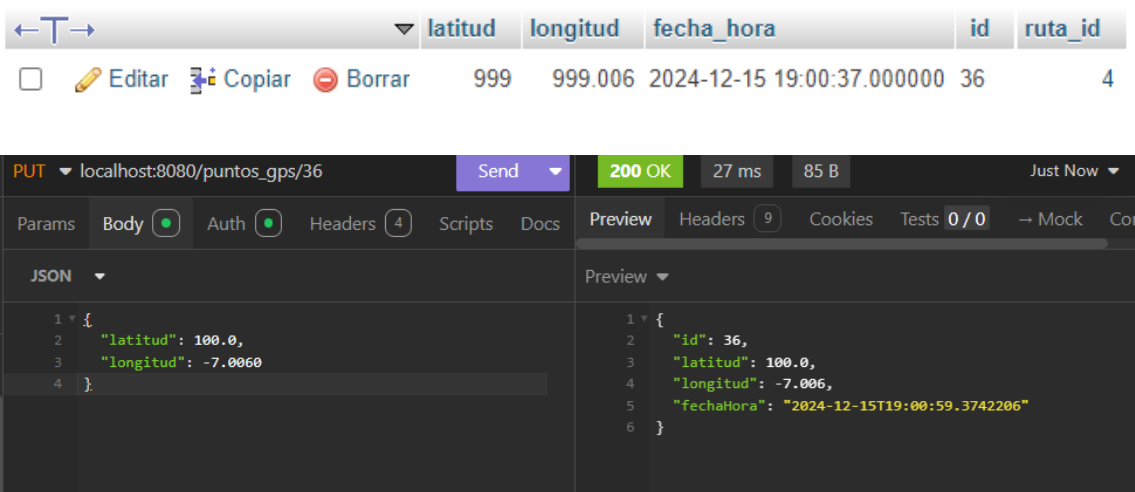
- Con el token del usuario 1 a una id de punto gps suyo:



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

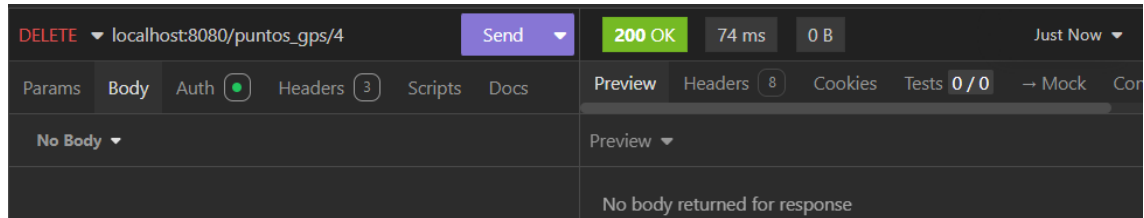


- Con el token del usuario 1 y la id 36, que es un punto de otra ruta que pertenece a el usuario 1 también:

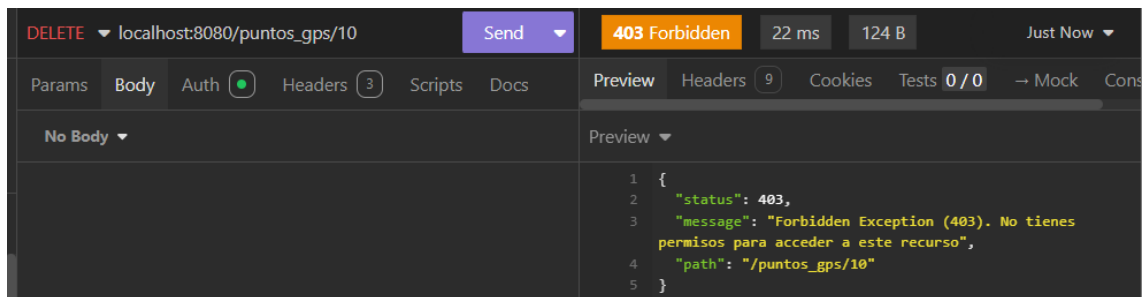


DeletePuntoGPSById (DELETE /puntos_gps/{punto_Id})

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



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




Tecnologías usadas

Dependencias

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.

En el proyecto tengo alguna más, pero no han sido usadas de momento explícitamente:

```
dependencies {  Edit Starters...  
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")  
    implementation("org.springframework.boot:spring-boot-starter-oauth2-resource-server")  
    implementation("org.springframework.boot:spring-boot-starter-web")  
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")  
    implementation("org.jetbrains.kotlin:kotlin-reflect")  
    developmentOnly("org.springframework.boot:spring-boot-devtools")  
    runtimeOnly("com.mysql:mysql-connector-j")  
    annotationProcessor("org.springframework.boot:spring-boot-configuration-processor")  
    providedRuntime("org.springframework.boot:spring-boot-starter-tomcat")  
    testImplementation("org.springframework.boot:spring-boot-starter-test")  
    testImplementation("org.jetbrains.kotlin:kotlin-test-junit5")  
    testRuntimeOnly("org.junit.platform:junit-platform-launcher")  
}
```

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 necesaria 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 -> RepositorioUna vez el repositorio hace lo que se le pida, el proceso es al revés:
 - Repositorio -> Servicio -> ControladorY 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.