PSP Microservicio de gestión de motos

Luna Flores Yanh Mauricio

DOCUMENTO BASE

1. Nombre del Proyecto

Microservicio de motos

2. Alcance

Desarrollo de un microservicio RESTful utilizando Spring Boot y Spring Data JPA que permite gestionar un catálogo de motos. El sistema expone endpoints HTTP para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre registros de motos almacenados en una base de datos relacional PostgreSQL. Cada registro contempla un identificador único por placa y atributos como marca, modelo, año, color, cilindraje, tipo y kilometraje. El microservicio puede ser consumido por clientes HTTP externos, como Postman o cualquier aplicación frontend, para la gestión y consulta de información en tiempo real.

3. Requisitos

- Java 17 instalado.
- Maven (o uso de mvnw.cmd incluido en el proyecto).
- Spring Boot 3.x con dependencias: Web, Data JPA, Validation, PostgreSQL Driver, Lombok.
- PostgreSQL en ejecución (según tu configuración actual, puerto 5433).
- Base de datos creada (por ejemplo microservice) y credenciales válidas en application.properties.
- Herramienta para pruebas HTTP (Postman).

4. Requerimientos Funcionales

RF01. Registrar una moto con los campos:
placa, marca, modelo, anio, color, cilindraje, tipo, kilometraje.

RF02. Consultar todas las motos registradas.

RF03. Consultar una moto por su placa.

RF04. Actualizar la información de una moto existente por placa.

RF05. Eliminar una moto por placa.

RF06. Validar datos obligatorios y rangos (ej. año, cilindraje, kilometraje).

RF07. Evitar duplicados de placa al crear registros.

RF08. Responder con códigos HTTP adecuados (201, 200, 204, 404, 409, 400).

5. Lista de Tareas

Configurar proyecto Spring Boot y conexión a PostgreSQL.

Definir entidad MotoEntity y mapeo de tabla.

Crear DTOs de request/response.

Implementar MotoRepository con JPA.

Implementar lógica de negocio en MotoService.

Implementar MotoController con endpoints CRUD.

Configurar validaciones con anotaciones (@NotBlank, @NotNull, @Min, @Max, etc.).

Probar endpoints con Postman (POST, GET, PUT, DELETE).

Verificar persistencia real en base de datos.

Documentar alcance, requisitos y evidencias de pruebas.

6. Criterios de Aceptación

CA01. Al hacer POST /api/motos con datos válidos, se crea un registro y responde 201.

CA02. Al hacer GET /api/motos, se obtiene la lista de motos registrada en la BD.

CA03. Al hacer GET /api/motos/{placa} de una placa existente, responde 200 con datos correctos.

CA04. Al hacer PUT /api/motos/{placa} de una placa existente, se actualizan los campos y responde 200.

CA05. Al hacer DELETE /api/motos/{placa}, elimina el registro y responde 204.

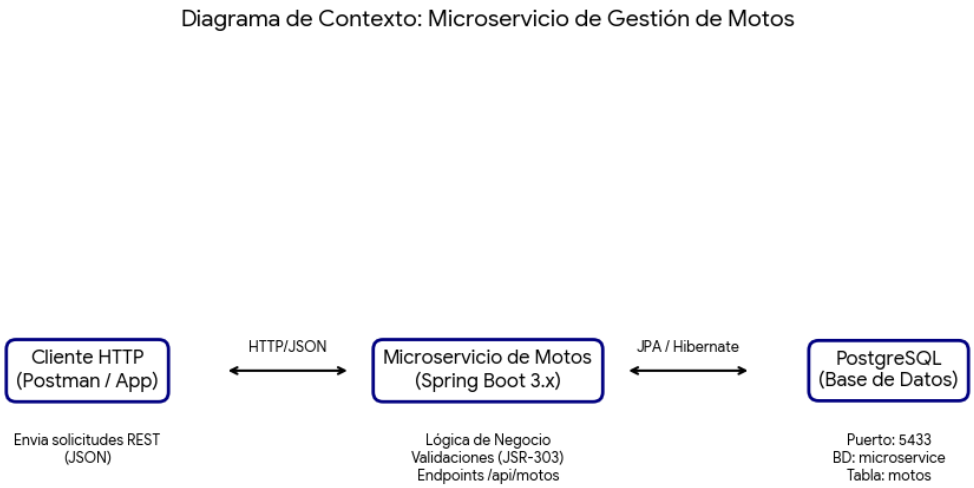
CA06. Si la placa no existe en GET/PUT/DELETE, responde 404.

CA07. Si se intenta crear una placa repetida, responde 409.

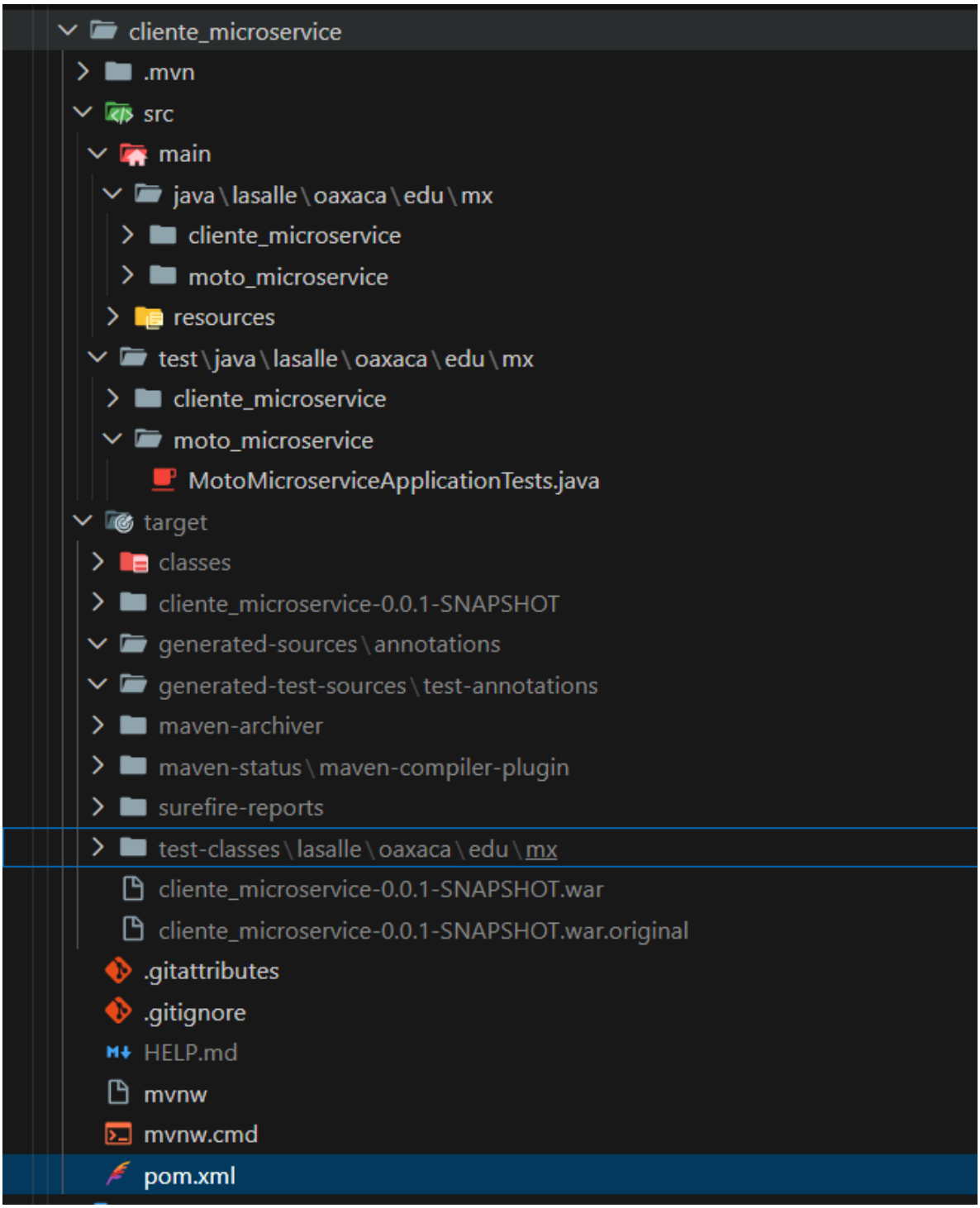
CA08. Si se envían datos inválidos o incompletos, responde 400.

CA09. Los cambios realizados mediante API se reflejan correctamente en PostgreSQL.

7. Diagrama de contexto



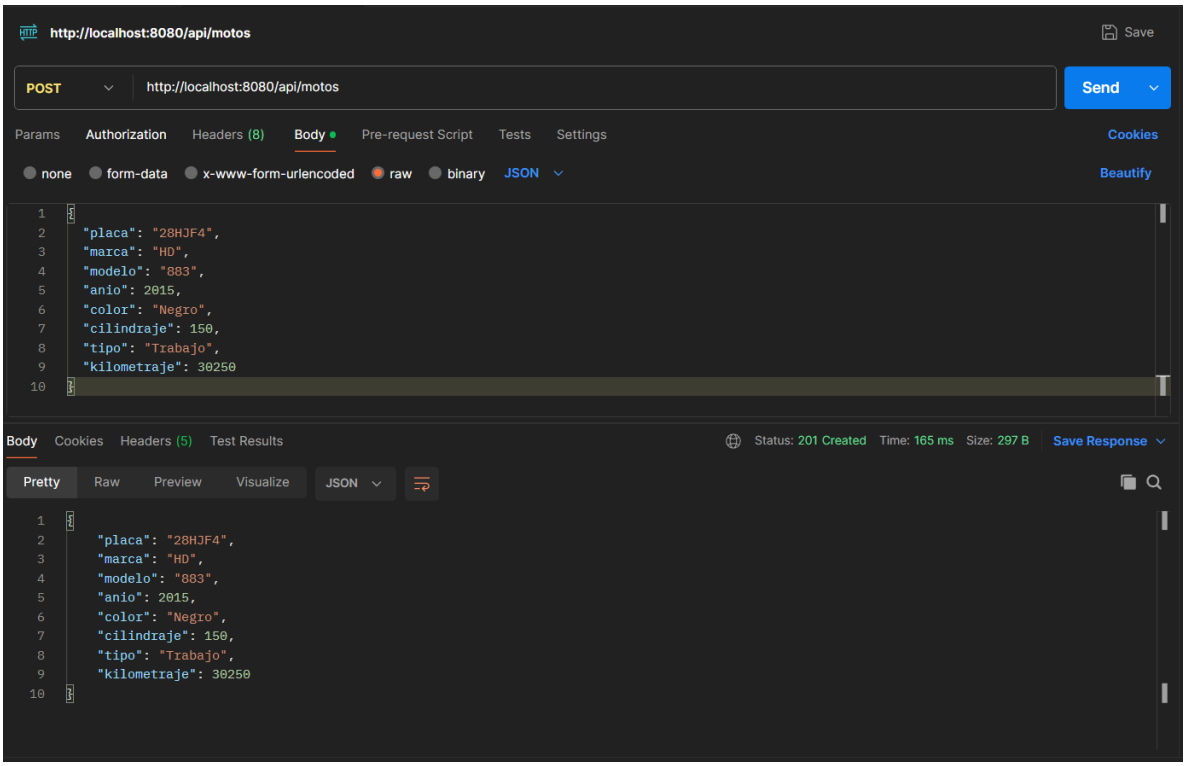
ESTRUCTURA DEL PROYECTO

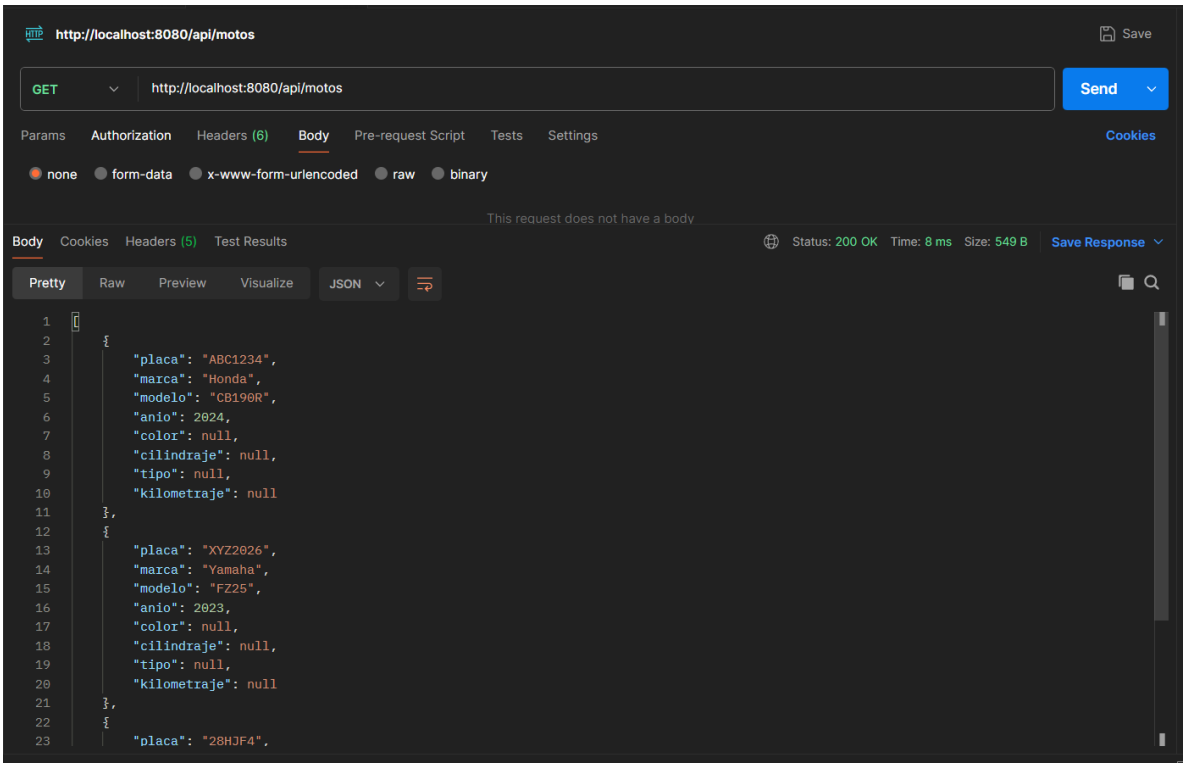


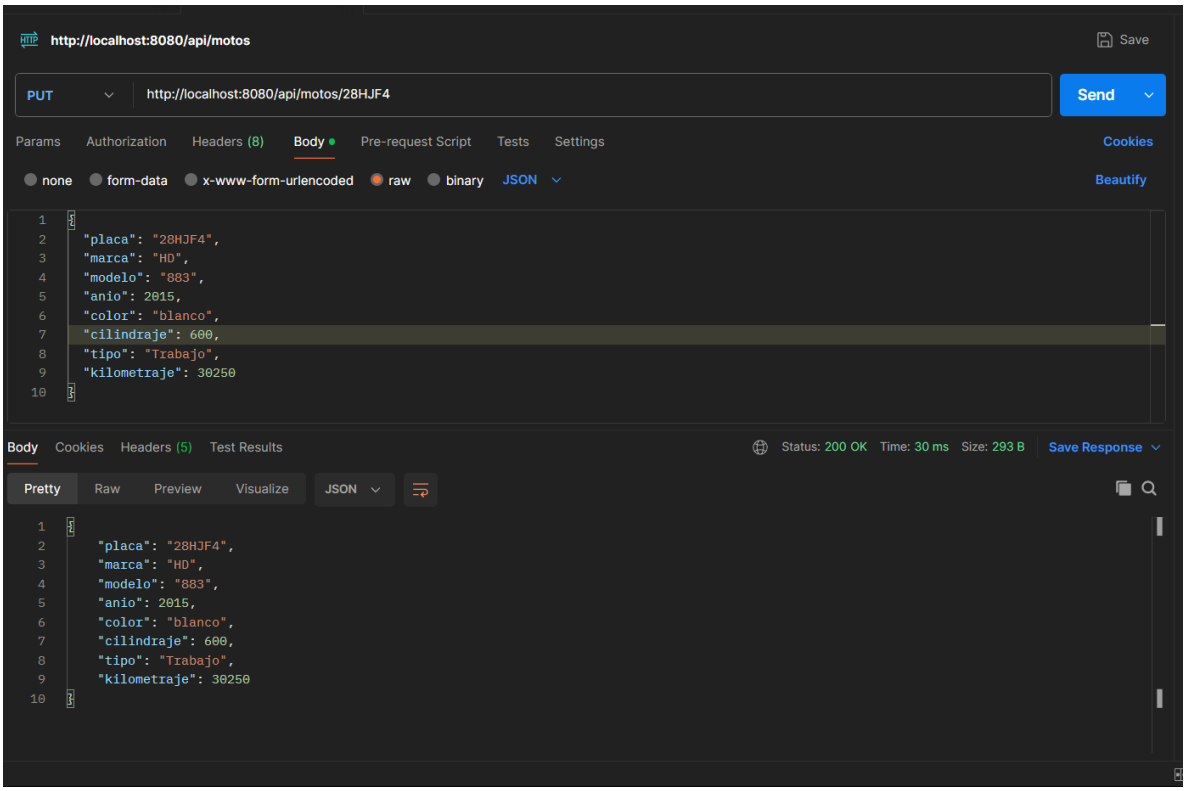
ARCHIVO APPLICATION.PROPERTIES

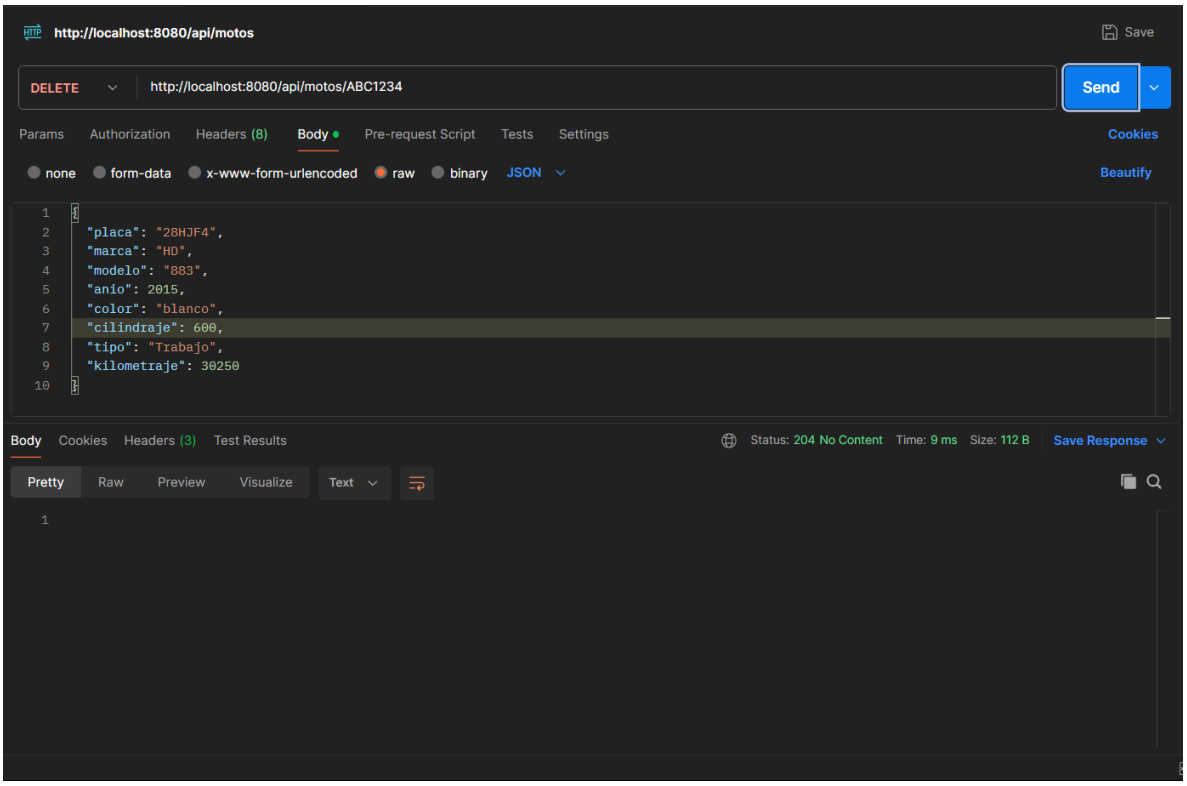
```
spring.application.name=moto_microservice

spring.datasource.url=jdbc:postgresql://localhost:5433/microservice
spring.datasource.username=postgres
spring.datasource.password=Lufy1386
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

PRUEBAS DE FUNCIONAMIENTO

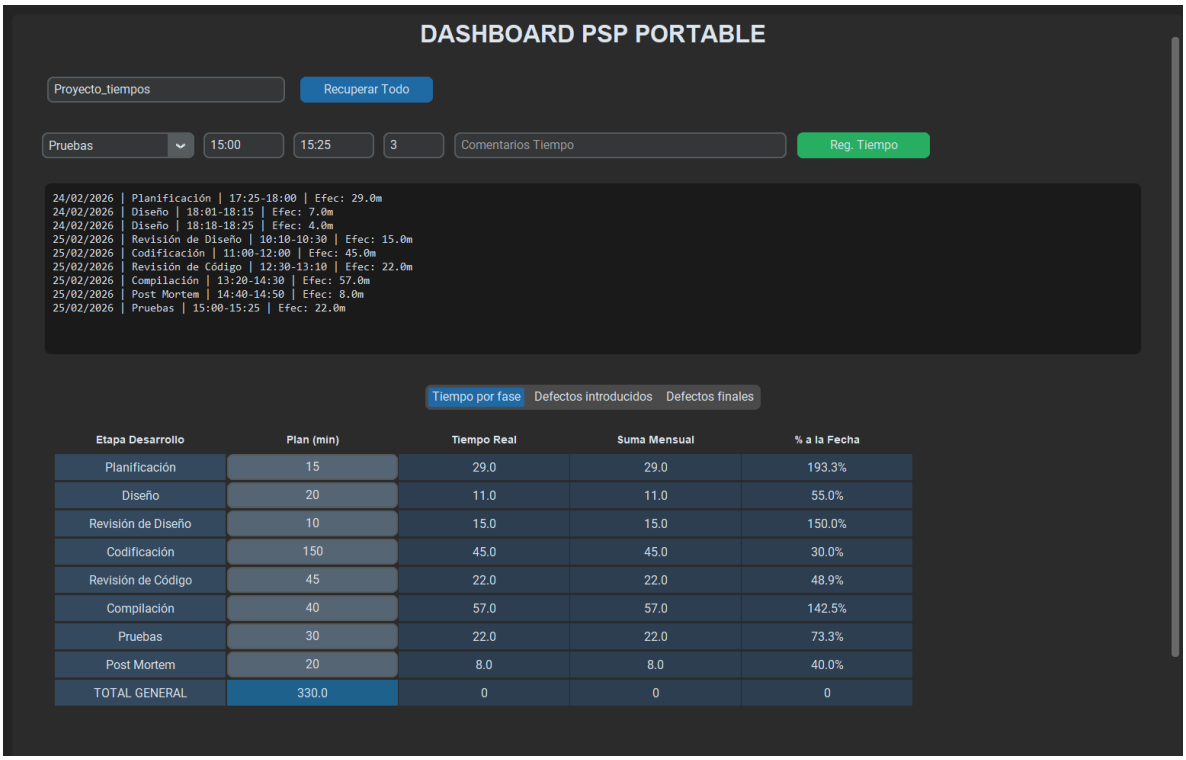





http://localhost:8080/api/motos
Save
PUT
http://localhost:8080/api/motos/28HJF4
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary JSON
Beautify
{
"placa": "28HJF4",
"marca": "HD",
"modelo": "883",
"anio": 2015,
"color": "blanco",
"cilindraje": 600,
"tipo": "Trabajo",
"kilometraje": 30250
}
Body Cookies Headers (5) Test Results
Status: 200 OK Time: 30 ms Size: 293 B
Save Response
Pretty Raw Preview Visualize JSON
{
"placa": "28HJF4",
"marca": "HD",
"modelo": "883",
"anio": 2015,
"color": "blanco",
"cilindraje": 600,
"tipo": "Trabajo",
"kilometraje": 30250
}


http://localhost:8080/api/motos
Save
DELETE
http://localhost:8080/api/motos/ABC1234
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary JSON
Beautify
{
"placa": "28HJF4",
"marca": "HD",
"modelo": "883",
"anio": 2015,
"color": "blanco",
"cilindraje": 600,
"tipo": "Trabajo",
"kilometraje": 30250
}
Body Cookies Headers (3) Test Results
Status: 204 No Content Time: 9 ms Size: 112 B
Save Response
Pretty Raw Preview Visualize Text

INFORME GENERAL DEL PSP



Etapa Desarrollo	Plan (min)	Tiempo Real	Suma Mensual	% a la Fecha
Planificación	15	29.0	29.0	193.3%
Diseño	20	11.0	11.0	55.0%
Revisión de Diseño	10	15.0	15.0	150.0%
Codificación	150	45.0	45.0	30.0%
Revisión de Código	45	22.0	22.0	48.9%
Compilación	40	57.0	57.0	142.5%
Pruebas	30	22.0	22.0	73.3%
Post Mortem	20	8.0	8.0	40.0%
TOTAL GENERAL	330.0	0	0	0

Postmortem

Se completó el microservicio CRUD de motos con Spring Boot, JPA y PostgreSQL, validado desde Postman. El principal incidente fue un error 500 por desajuste entre la entidad y la tabla (nuevos campos no existentes en BD), corregido al sincronizar el esquema. También hubo fallas de ejecución por ruta incorrecta y puerto 8080 ocupado, resueltas al estandarizar el arranque del proyecto.
Finalmente, se ajustó el test de contexto para evitar fallas por dependencias de repositorio en pruebas.
Resultado: API estable, conexión a BD funcional y operaciones CRUD operativas.

Conclusiones

Se desarrolló e integró exitosamente un microservicio RESTful para la gestión de motos, cumpliendo el objetivo de implementar operaciones CRUD con Spring Boot, Spring Data JPA y PostgreSQL. Durante el proceso se validó el funcionamiento de los endpoints mediante Postman y se aseguró la persistencia correcta de la información en base de datos.