

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

Diagrama de Contexto: Microservicio de Gestión de Motos



ESTRUCTURA DEL PROYECTO

The screenshot shows a file explorer window with the following project structure:

- cliente_microservice
 - .mvn
 - src
 - main
 - java\lasalle\oaxaca\edu\mx
 - cliente_microservice
 - moto_microservice
 - resources
 - test\java\lasalle\oaxaca\edu\mx
 - cliente_microservice
 - moto_microservice
 - MotoMicroserviceApplicationTests.java
 - target
 - classes
 - cliente_microservice-0.0.1-SNAPSHOT
 - generated-sources\annotations
 - generated-test-sources\test-annotations
 - maven-archiver
 - maven-status\maven-compiler-plugin
 - surefire-reports
 - test-classes\lasalle\oaxaca\edu\mx
 - cliente_microservice-0.0.1-SNAPSHOT.war
 - cliente_microservice-0.0.1-SNAPSHOT.war.original
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

ARCHIVO APPLICATION.PROPERTIES

```
1 spring.application.name=moto_microservice
2
3 spring.datasource.url=jdbc:postgresql://localhost:5433/microservice
4 spring.datasource.username=postgres
5 spring.datasource.password=Luffy1386
6 spring.datasource.driver-class-name=org.postgresql.Driver
7
8 spring.jpa.hibernate.ddl-auto=update
9 spring.jpa.show-sql=true
10 spring.jpa.properties.hibernate.format_sql=true
11 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

PRUEBAS DE FUNCIONAMIENTO

POST http://localhost:8080/api/motos

Send

Params Authorization Headers (8) Body **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary JSON

```

1 [
2   "placa": "28HJF4",
3   "marca": "HD",
4   "modelo": "683",
5   "anio": 2015,
6   "color": "Negro",
7   "cilindraje": 150,
8   "tipo": "Trabajo",
9   "kilometraje": 30250
10 ]

```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 165 ms Size: 297 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   "placa": "28HJF4",
3   "marca": "HD",
4   "modelo": "683",
5   "anio": 2015,
6   "color": "Negro",
7   "cilindraje": 150,
8   "tipo": "Trabajo",
9   "kilometraje": 30250
10 ]

```

GET http://localhost:8080/api/motos

Send

Params Authorization Headers (6) Body **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary

This request does not have a body

Status: 200 OK Time: 8 ms Size: 549 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "placa": "ABC1234",
4     "marca": "Honda",
5     "modelo": "CB190R",
6     "anio": 2024,
7     "color": null,
8     "cilindraje": null,
9     "tipo": null,
10    "kilometraje": null
11  },
12  {
13    "placa": "XYZ2026",
14    "marca": "Yamaha",
15    "modelo": "FZ25",
16    "anio": 2023,
17    "color": null,
18    "cilindraje": null,
19    "tipo": null,
20    "kilometraje": null
21  },
22  {
23    "placa": "28HJF4".

```

HTTP <http://localhost:8080/api/motos>

PUT <http://localhost:8080/api/motos/28HJF4>

Send Save

Params Authorization Headers (8) Body **JSON** Tests Settings

none form-data x-www-form-urlencoded raw binary Beautify

```
1
2   "placa": "28HJF4",
3   "marca": "HD",
4   "modelo": "883",
5   "anio": 2015,
6   "color": "blanco",
7   "cilindraje": 600,
8   "tipo": "Trabajo",
9   "kilometraje": 30250
10 
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 30 ms Size: 293 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2   "placa": "28HJF4",
3   "marca": "HD",
4   "modelo": "883",
5   "anio": 2015,
6   "color": "blanco",
7   "cilindraje": 600,
8   "tipo": "Trabajo",
9   "kilometraje": 30250
10 
```

HTTP <http://localhost:8080/api/motos>

DELETE <http://localhost:8080/api/motos/ABC1234>

Send Save

Params Authorization Headers (8) Body **JSON** Tests Settings

none form-data x-www-form-urlencoded raw binary Beautify

```
1
2   "placa": "28HJF4",
3   "marca": "HD",
4   "modelo": "883",
5   "anio": 2015,
6   "color": "blanco",
7   "cilindraje": 600,
8   "tipo": "Trabajo",
9   "kilometraje": 30250
10 
```

Body Cookies Headers (3) Test Results Status: 204 No Content Time: 9 ms Size: 112 B Save Response

Pretty Raw Preview Visualize Text

```
1
```

INFORME GENERAL DEL PSP

DASHBOARD PSP PORTABLE

The screenshot shows a dark-themed dashboard titled "DASHBOARD PSP PORTABLE". At the top, there are search and filter fields: "Proyecto_tiempos" (selected), "Recuperar Todo", "Pruebas" (selected), "15:00", "15:25", "3", "Comentarios Tiempo", and a green "Reg. Tiempo" button. Below these are two sections: a timeline of tasks and a detailed table of development phases.

Timeline of Tasks:

```

24/02/2026 | Planificación | 17:25-18:00 | Efec: 29.0m
24/02/2026 | Diseño | 18:01-18:15 | Efec: 7.0m
24/02/2026 | Diseño | 18:18-18:25 | Efec: 4.0m
25/02/2026 | Revisión de Diseño | 10:10-10:30 | Efec: 15.0m
25/02/2026 | Codificación | 11:00-12:00 | Efec: 45.0m
25/02/2026 | Revisión de Código | 12:30-13:10 | Efec: 22.0m
25/02/2026 | Compilación | 13:20-14:30 | Efec: 57.0m
25/02/2026 | Post Mortem | 14:40-14:50 | Efec: 8.0m
25/02/2026 | Pruebas | 15:00-15:25 | Efec: 22.0m

```

Table of Development Phases:

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.