# SPRING DATA -REST API-JPA

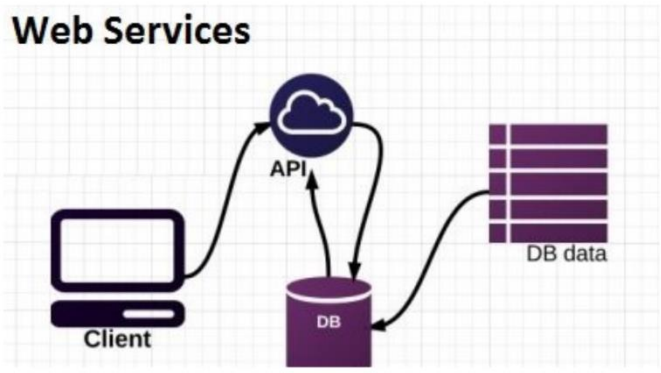
MIGUEL ANGEL RAMIREZ JUAREZ

JAVA ACADEMY

SEP/ 06/ 2024

## What is a Collection in Java?

The goal of this project is to create an inventory management system for a retail company using **Spring Boot** and **Spring Data JPA**. The system offers CRUD (Create, Read, Update, Delete) operations for managing product inventory, with the database being handled by **Spring Data JPA** to simplify interactions with a **MySQL** database. The REST API allows users to manage product information efficiently.




## Proposed Solution:

The solution uses **Spring Boot** and **Spring Data JPA** to interact with a **MySQL** database. The system is designed to handle CRUD operations through a REST API for managing products, making use of Spring Data JPA's simplified repository interface to reduce boilerplate code for database operations.

System Architecture:

1-Technologies Used:

Backend: Spring Boot, Spring Data JPA

Database: MySQL

Dependencies: Spring Web, Spring Data JPA, MySQL Driver, Lombok

Programming Language: Java

2-

REST Client (Postman/cURL) <---> REST Controller (ProductController) <---> Spring Data JPA Repository (ProductRepository) <---> MySQL Database

## Implemented Functionalities:

Add Product to Inventory:

Method: POST

Endpoint: /api/products

Description: Allows users to add new products to the inventory by specifying product details such as name, quantity, and price.

View All Products:

Method: GET

Endpoint: /api/products

Description: Returns a list of all available products in the inventory.

View Product by ID:

Method: GET

Endpoint: /api/products/{id}

Description: Retrieves details of a specific product by its unique ID.

Update Product Information:

Method: PUT

Endpoint: /api/products/{id}

Description: Updates information (quantity, price, etc.) for a specific product identified by its ID.

Delete Product:

Method: DELETE

Endpoint: /api/products/{id}

Description: Deletes a product from the inventory using its ID.

## CODE DIFERENTS FOR USIGN SPRING DATA:

```java
package com.hw1.springboot3restapijpacrud.entity;


import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;

@Data
@Entity
@Table(name = "Product")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nombre;
    private int cantidad;
    private String ubicacionAlmacen;
}
```

## Expected Results:

This system enables the retail company to:

Manage product inventory across multiple stores.

Easily add, update, and delete products in real-time.

Retrieve product information by ID or list all products.

Keep inventory up to date and accessible via REST API.

## POSTMAN:

- **GET /api/productos**:
  - Method: GET
  - URL: `http://localhost:8080/api/productos`
  - Click "Send" to retrieve all products.
- **GET /api/productos/{id}**:
  - Method: GET
  - URL: `http://localhost:8080/api/productos/{id}`
  - Replace `{id}` with an actual product ID.
  - Click "Send" to retrieve a specific product.
- **POST /api/productos**:
  - Method: POST
  - URL: `http://localhost:8080/api/productos`
  - Body: Select "raw" and "JSON" format, then provide the product details in JSON format, e.g.:

**JSON**

```
{
  "nombre": "Product Name",
  "cantidad": 10,
  "ubicacionAlmacen": "A1"
}
```

  - Click "Send" to create a new product.
- **PUT /api/productos/{id}**:
  - Method: PUT
  - URL: `http://localhost:8080/api/productos/{id}`

- Replace `{id}` with the product ID you want to update.
- Body: Select “raw” and “JSON” format, then provide the updated product details in JSON format.
- Click “Send” to update the product.
- **DELETE /api/productos/{id}**:
- Method: DELETE
- URL: `http://localhost:8080/api/productos/{id}`
- Replace `{id}` with the product ID you want to delete.
- Click “Send” to delete the product.

## Conclusions and Next Steps:

Ease of Use: Spring Data JPA provides built-in CRUD methods, reducing boilerplate code and speeding up development.

Scalability: The system can easily be extended to handle more complex business logic, such as product categorization and filtering.

Security: Authentication and role-based access control should be added to secure the API for production environments.

EXAMPLE OF WORK PUT PROCESS EN POSTMAN: