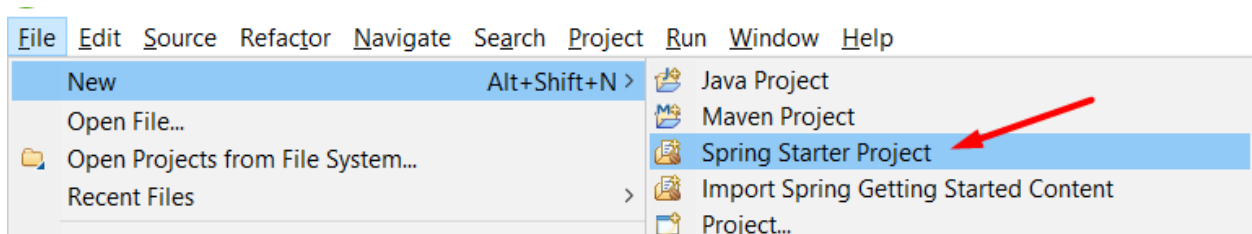


PROYECTO “crud-rest”

Paso 1: Crear el Proyecto en STS

Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**.



Completar los datos del proyecto:

- **Name:** producto-api
- **Type:** Maven
- **Java Version:** 17
- **Packaging:** Jar
- **Group:** com.example
- **Artifact:** producto-api

A screenshot of the 'New Spring Starter Project' dialog box in STS. The dialog contains various fields for project configuration. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' field contains 'producto-api'. The 'Use default location' checkbox is checked. The 'Location' field shows 'C:\Users\JR\Documents\workspace\producto-api'. The 'Type' is set to 'Maven', 'Packaging' to 'Jar', 'Java Version' to '17', and 'Language' to 'Java'. The 'Group' field contains 'com.example', 'Artifact' contains 'producto-api', 'Version' contains '0.0.1-SNAPSHOT', 'Description' contains 'Demo project for Spring Boot', and 'Package' contains 'com.example.productoapi'. At the bottom, there are options for 'Working sets' with 'Add project to working sets' unchecked and buttons for 'New...' and 'Select...'.

- **Dependencies:**
 - Spring Web (Para crear la API REST)
 - Spring Boot DevTools (Para recarga automática)
 - Lombok (Para reducir código repetitivo)

A screenshot of the 'Spring Boot Version' and 'Frequently Used' dependencies section in STS. The 'Spring Boot Version' is set to '3.4.2'. Below it, the 'Frequently Used' section shows a list of dependencies with checkboxes. The checked dependencies are 'Spring Boot DevTools' and 'Spring Web'. Other dependencies shown include 'H2 Database', 'Lombok', 'Spring Data JPA', and 'PostgreSQL Driver'.

Hacer clic en **Finish** para generar el proyecto.

Paso 2: Crear la Entidad Producto

Creamos una clase para representar un producto con ID, nombre y precio.

1. En el paquete **com.example.productoapi.model**, crear la clase **Producto.java**:



```
package com.example.productoapi.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor

public class Producto {

    private Long id;
    private String nombre;
    private double precio;
}
```

Explicación:

- **@Data**: Genera automáticamente getters, setters, toString, equals y hashCode.
- **@AllArgsConstructor**: Crea un constructor con todos los argumentos.
- **@NoArgsConstructor**: Crea un constructor vacío.

Paso 3: Crear una Lista para Simular una Base de Datos

En este ejercicio, no usaremos base de datos real. Guardaremos los datos en una lista en memoria.

1. En el paquete **com.example.productoapi.repository**, crear la clase **ProductoRepository.java**:



```
package com.example.productoapi.repository;

import com.example.productoapi.model.Producto;
import org.springframework.stereotype.Repository;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Repository
public class ProductoRepository {

    private final List<Producto> productos = new ArrayList<>();

    public List<Producto> findAll() {
        return productos;
    }

    public Optional<Producto> findById(Long id) {
        return productos.stream().filter(p -> p.getId().equals(id)).findFirst();
    }

    public Producto save(Producto producto) {
        productos.add(producto);
        return producto;
    }

    public boolean deleteById(Long id) {
        return productos.removeIf(p -> p.getId().equals(id));
    }
}
```

Explicación:

- **List<Producto> productos:** Simula una base de datos en memoria.
- **findAll():** Retorna todos los productos.
- **findById(Long id):** Busca un producto por ID.
- **save(Producto producto):** Guarda un nuevo producto en la lista.
- **deleteById(Long id):** Elimina un producto por ID.

Paso 4: Crear el Servicio ProductoService

El servicio maneja la lógica de negocio.

1. En el paquete **com.example.productoapi.service**, crear la clase **ProductoService.java**:



```
package com.example.productoapi.service;

import com.example.productoapi.model.Producto;
import com.example.productoapi.repository.ProductoRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class ProductoService {

    private final ProductoRepository productoRepository;

    public ProductoService(ProductoRepository productoRepository) {
        this.productoRepository = productoRepository;
    }

    public List<Producto> obtenerTodos() {
        return productoRepository.findAll();
    }

    public Optional<Producto> obtenerPorId(Long id) {
        return productoRepository.findById(id);
    }

    public Producto guardar(Producto producto) {
        return productoRepository.save(producto);
    }

    public boolean eliminar(Long id) {
        return productoRepository.deleteById(id);
    }
}
```

Explicación:

- Se inyecta **ProductoRepository** para acceder a los datos.
- **obtenerTodos()**: Retorna todos los productos.
- **obtenerPorId(Long id)**: Retorna un producto específico.
- **guardar(Producto producto)**: Guarda un producto.
- **eliminar(Long id)**: Elimina un producto.

Paso 5: Crear el Controlador REST

El controlador expone los endpoints REST.

1. En el paquete **com.example.productoapi.controller**, crear la clase **ProductoController.java**:



```
package com.example.productoapi.controller;

import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.productoapi.model.Producto;
import com.example.productoapi.service.ProductoService;

@RestController
@RequestMapping("/api/productos")
public class ProductoController {

    private final ProductoService productoService;

    public ProductoController(ProductoService productoService) {
        this.productoService = productoService;
    }

    @GetMapping
    public List<Producto> obtenerTodos() {
        return productoService.obtenerTodos();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Producto> obtenerPorId(@PathVariable Long id) {
        return productoService.obtenerPorId(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public Producto crearProducto(@RequestBody Producto producto) {
        return productoService.guardar(producto);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> eliminarProducto(@PathVariable Long id) {
        return productoService.eliminar(id) ?
            ResponseEntity.noContent().build() :
            ResponseEntity.notFound().build();
    }
}
```

Explicación:

- **@GetMapping:** Retorna todos los productos.
- **@GetMapping("/{id}"):** Busca un producto por ID.
- **@PostMapping:** Crea un producto.
- **@DeleteMapping("/{id}"):** Elimina un producto.
- **ResponseEntity.noContent().build():** Devuelve una respuesta HTTP con el código de estado 204 No Content, que significa que la operación fue exitosa pero no hay contenido en la respuesta.
- **ResponseEntity.notFound().build():** Devuelve una respuesta HTTP con el código de estado 404 Not Found, que significa que el elemento no fue encontrado o no existe.

Paso 6: Ejecutar y probar la API

Obtener todos los productos:

- **GET** http://localhost:8080/api/productos

The screenshot shows a Postman interface. At the top, a GET request is configured for the URL `http://localhost:8080/api/productos`. Below the request bar, the 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response with a status of '200 OK', a response time of '5 ms', and a size of '208 B'. The JSON data is as follows:

```
1 [
2   {
3     "id": 1,
4     "nombre": "Laptop",
5     "precio": 1200.5
6   }
7 ]
```

Crear un producto:

- **POST** http://localhost:8080/api/productos
- **Body (JSON):**

The screenshot shows a JSON body for a POST request, displayed next to a yellow icon of a document with a pencil. The JSON is as follows:

```
{
  "id": 1,
  "nombre": "Laptop",
  "precio": 1200.50
}
```

POST http://localhost:8080/api/productos Send

Params Authorization Headers (8) **Body** Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☐ JSON Beautify

```

1 {
2   "id": 1,
3   "nombre": "Laptop",
4   "precio": 1200.50
5 }
6

```

Body Cookies Headers (5) Test Results 200 OK • 98 ms • 206 B • Save Response ⋮

{} JSON Preview Visualize ⋮

```

1 {
2   "id": 1,
3   "nombre": "Laptop",
4   "precio": 1200.5
5 }

```

Obtener un producto por ID:

- **GET** http://localhost:8080/api/productos/1

GET http://localhost:8080/api/productos/1 Send

Params Authorization Headers (6) **Body** Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body Cookies Headers (5) Test Results 200 OK • 11 ms • 206 B • Save Response ⋮

{} JSON Preview Visualize ⋮

```

1 {
2   "id": 1,
3   "nombre": "Laptop",
4   "precio": 1200.5
5 }

```

Eliminar un producto:

- **DELETE** http://localhost:8080/api/productos/1

DELETE http://localhost:8080/api/productos/1 Send

Params Authorization Headers (6) **Body** Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body Cookies Headers (3) Test Results 204 No Content • 4 ms • 112 B • Save Response ⋮

Raw Preview Visualize ⋮

```

1

```