Ejercicios de API REST con Spring Boot + MySQL para Tienda Deportiva

📌 1. Configuración Inicial del Proyecto Spring Boot

(Basado en: Tutorial CRUD con Spring Boot y MySQL)

Pasos:

1. **Crear proyecto** en Spring Initializr con:
    - Dependencias: **Spring Web**, **Spring Data JPA**, **MySQL Driver**.

```
Group: com.tiendadeportiva
Artifact: api-productos
```

2. **Configurar** `application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/tienda_deportiva
spring.datasource.username=root
spring.datasource.password=tu_contraseña
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

3. **Estructura de paquetes**:

```
src/
├── main/
│   ├── java/
│   │   └── com.tiendadeportiva/
│   │       ├── model/
│   │       ├── repository/
│   │       ├── controller/
│   │       └── ApiProductosApplication.java
```

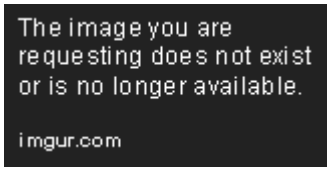
📌 2. Entidad `Producto` con JPA

Objetivo: Mapear la tabla `productos` de MySQL a una clase Java.

```
package com.tiendadeportiva.model;

import jakarta.persistence.*;

@Entity
```

```java
@Table(name = "productos")
public class Producto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nombre;

    @Column(nullable = false)
    private String descripcion;

    @Column(nullable = false)
    private double precio;

    @Column(nullable = false)
    private int stock;

    @ManyToOne
    @JoinColumn(name = "id_categoria")
    private Categoria categoria;

    // Getters y Setters
    // Constructor vacío
}
```

Imagen de la estructura:



(Diagrama UML: Atributos y relación con `Categoria`).

📌 3. Repositorio y Controlador

Ejercicio: Implementar el CRUD para productos.

1. **Interfaz `ProductoRepository`**:

```java
package com.tiendadeportiva.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.tiendadeportiva.model.Producto;

public interface ProductoRepository extends JpaRepository<Producto, Long> {
    List<Producto> findByCategoriaNombre(String nombre);
}
```

2. **Clase `ProductoController`**:

```java
package com.tiendadeportiva.controller;

import org.springframework.web.bind.annotation.*;
import com.tiendadeportiva.repository.ProductoRepository;

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

    @Autowired
    private ProductoRepository productoRepository;

    @GetMapping
    public List<Producto> listarProductos() {
        return productoRepository.findAll();
    }

    @PostMapping
    public Producto crearProducto(@RequestBody Producto producto) {
        return productoRepository.save(producto);
    }
}
```

📌 4. Ejercicios Prácticos

◇ Ejercicio 1: Endpoint para Productos por Categoría

Tarea:

- Crear un endpoint `GET /api/productos/categoria/{nombre}` que filtre productos por categoría (ej: "running").
- Usar `@Query` en el repositorio.

Solución:

```java
// En ProductoRepository
@Query("SELECT p FROM Producto p WHERE p.categoria.nombre = :nombre")
List<Producto> findByCategoria(@Param("nombre") String nombre);
```

◇ Ejercicio 2: Actualización de Stock

Tarea:

- Crear un endpoint `PUT /api/productos/{id}/stock` que reciba `{"cantidad": 5}` y actualice el stock.
- Validar que el stock no sea negativo.

Solución:

```
@PutMapping("/{id}/stock")
public ResponseEntity<Producto> actualizarStock(
    @PathVariable Long id,
    @RequestBody Map<String, Integer> body
) {
    Producto producto = productoRepository.findById(id).orElseThrow();
    int nuevoStock = producto.getStock() + body.get("cantidad");
    if (nuevoStock < 0) {
        return ResponseEntity.badRequest().build();
    }
    producto.setStock(nuevoStock);
    return ResponseEntity.ok(productoRepository.save(producto));
}
```

📌 5. Frontend con jQuery

Ejercicio: Mostrar productos en HTML usando AJAX.

```
<!DOCTYPE html>
<html>
<head>
    <title>Tienda Deportiva</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <h1>Productos</h1>
    <div id="lista-productos"></div>

    <script>
        $(document).ready(function() {
            $.get("http://localhost:8080/api/productos", function(data) {
                data.forEach(producto => {
                    $("#lista-productos").append(`
                        <div>
                            <h3>${producto.nombre}</h3>
                            <p>Precio: ${producto.precio}€</p>
                        </div>
                    `);
                });
            });
        });
    </script>
</body>
</html>
```

📌 6. Despliegue y Pruebas

1. **Ejecutar la aplicación**:

```
mvn spring-boot:run
```

2. **Probar endpoints** con Postman:

```
curl -X GET http://localhost:8080/api/productos
```

3. **Compartir Collection de Postman**: Instrucciones de uso de Postman
    - Crear pruebas para cada endpoint.
    - Guardar la colección.
    - Exportar la colección y subirla al repositorio o compartirla directamente.

📌 Recursos Adicionales

- Documentación de Spring Data JPA