



UNIVERSIDAD DE LAS FUERZAS ARMADAS –ESPE-  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN



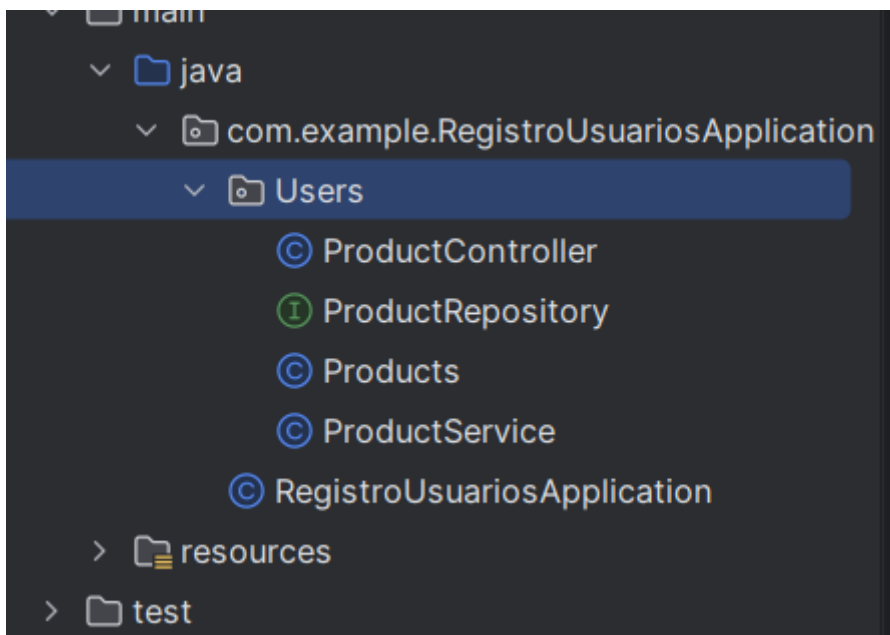
DESARROLLO WEB PARA INTEGRACIÓN

NOMBRES: ANDRANGO JESSICA , MOGROVEJO SEBASTIAN

FECHA: 18/08/2023

PRUEBA TERCER PARCIAL

**Modelo de Datos:** Crea una entidad **Product** con atributos como id, nombre, descripción, precio, stock, etc. Utiliza anotaciones de JPA para mapear la entidad a la base de datos.



```
public class Products {  
  
    @Id  
  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(unique = true)
```

```

private String name;

private String descripcion;

private String precio;

private String stock;


public Products() {

}

public Products(Long id, String name, String descripcion
, String precio, String stock) {

    this.id = id;

    this.name = name;

    this.descripcion = descripcion;

    this.precio = precio;

    this.stock = stock;
}

```

**Repositorio:** Crea un repositorio ProductRepository que extienda JpaRepository para acceder a la base de datos y realizar operaciones CRUD sobre los productos

```

package com.example.RegistroUsuariosApplication.Users;

import
org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.Optional;

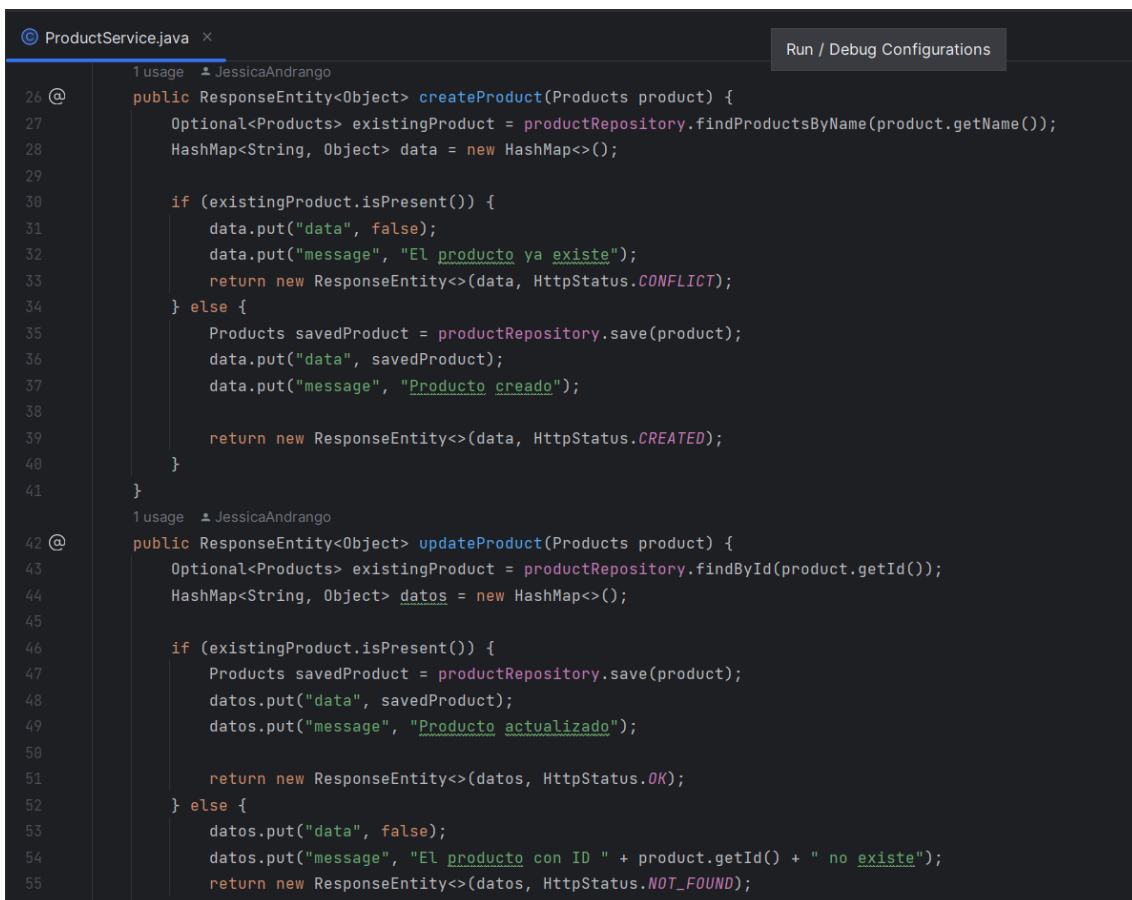
@Repository

public interface ProductRepository extends
JpaRepository<Products, Long> {

```

```
Optional<Products> findProductsByName(String name);  
}
```

**Servicio:** Crea un servicio ProductService que utilice ProductRepository para implementar los métodos de CRUD (createProduct, getProduct, updateProduct, deleteProduct, etc.).



```
ProductService.java x Run / Debug Configurations  
1 usage JessicaAndrango  
26 @  
27 public ResponseEntity<Object> createProduct(Products product) {  
28     Optional<Products> existingProduct = productRepository.findProductsByName(product.getName());  
29     HashMap<String, Object> data = new HashMap<>();  
30  
31     if (existingProduct.isPresent()) {  
32         data.put("data", false);  
33         data.put("message", "El producto ya existe");  
34         return new ResponseEntity<>(data, HttpStatus.CONFLICT);  
35     } else {  
36         Products savedProduct = productRepository.save(product);  
37         data.put("data", savedProduct);  
38         data.put("message", "Producto creado");  
39  
40         return new ResponseEntity<>(data, HttpStatus.CREATED);  
41     }  
42 }  
43  
44 1 usage JessicaAndrango  
45 @  
46 public ResponseEntity<Object> updateProduct(Products product) {  
47     Optional<Products> existingProduct = productRepository.findById(product.getId());  
48     HashMap<String, Object> datos = new HashMap<>();  
49  
50     if (existingProduct.isPresent()) {  
51         Products savedProduct = productRepository.save(product);  
52         datos.put("data", savedProduct);  
53         datos.put("message", "Producto actualizado");  
54  
55         return new ResponseEntity<>(datos, HttpStatus.OK);  
56     } else {  
57         datos.put("data", false);  
58         datos.put("message", "El producto con ID " + product.getId() + " no existe");  
59         return new ResponseEntity<>(datos, HttpStatus.NOT_FOUND);  
60     }  
61 }
```

**Controlador:** Crea un controlador ProductController con rutas para cada operación CRUD (/api/products). Utiliza anotaciones como @RestController, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

```

ProductController.java
8  @RestController
9  @RequestMapping(path = "api/v1/products")
10 public class ProductController {
11
12     5 usages
13     private final ProductService userService;
14
15     no usages  ⚡ JessicaAndrango
16     @Autowired
17     public ProductController(ProductService userService) { this.userService = userService; }
18
19     no usages  ⚡ JessicaAndrango
20     @GetMapping
21     public ResponseEntity<List<Products>> getAllProducts() {
22         List<Products> products = userService.getProducts();
23         return new ResponseEntity<>(products, HttpStatus.OK);
24     }
25
26     no usages  ⚡ JessicaAndrango
27     @PostMapping
28     public ResponseEntity<Object> crearUser(@RequestBody Products user) { return userService.createProduct(user); }
29
30     no usages  ⚡ JessicaAndrango
31     @PutMapping
32     public ResponseEntity<Object> actualizarUser(@RequestBody Products user) { return userService.updateProduct(user); }
33
34     no usages  ⚡ JessicaAndrango
35     @DeleteMapping(path = "{userId}")
36     public ResponseEntity<Object> eliminarUser(@PathVariable("userId") Long userId) {
37         return userService.deleteProduct(userId);
38     }
39

```

**Pruebas:** Utiliza herramientas como Postman o cURL para realizar pruebas sobre la API que has creado. Envía solicitudes POST, GET, PUT y DELETE para crear, leer, actualizar y eliminar productos

## GET

GET	http://localhost:8090/api/v1/products	Send	Status: 200 OK	Size: 2 Bytes	Time: 17 ms
Query	Headers 2	Auth	Body 1	Tests	Pre Run
JSON	XML	Text	Form	Form-encode	GraphQL Binary
			Response	Headers 4	Cookies Results
			1	[]	

## POST

POST http://localhost:8090/api/v1/products Send

Query Headers<sup>2</sup> Auth **Body<sup>1</sup>** Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```

1 {
2   "id": 1,
3   "name": "Reloj",
4   "descripcion": "Grande",
5   "precio": "254",
6   "stock": "Disponible"
7 }

```

**Response** Headers<sup>4</sup> Cookies Results Docs

```

1 {
2   "data": {
3     "id": 1,
4     "name": "Reloj",
5     "descripcion": "Grande",
6     "precio": "254",
7     "stock": "Disponible"
8   },
9   "message": "Producto creado"
10 }

```

Status: **201 Created** Size: **119 Bytes** Time: **227 ms**

## PUT

PUT http://localhost:8090/api/v1/products Send

Query Headers<sup>2</sup> Auth **Body<sup>1</sup>** Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```

1 {
2   "id": 1,
3   "name": "Reloj",
4   "descripcion": "Grande y mediano",
5   "precio": "254",
6   "stock": "Disponible"
7 }

```

**Response** Headers<sup>4</sup> Cookies Results Docs

```

1 {
2   "data": {
3     "id": 1,
4     "name": "Reloj",
5     "descripcion": "Grande y mediano",
6     "precio": "254",
7     "stock": "Disponible"
8   },
9   "message": "Producto actualizado"
10 }

```

Status: **200 OK** Size: **134 Bytes** Time: **75 ms**

## DELETE

DELETE http://localhost:8090/api/v1/products/1 Send

Query Headers<sup>2</sup> Auth **Body<sup>1</sup>** Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```

1 {
2   "data": true,
3   "message": "El producto con id 1 ha sido eliminado"
4 }

```

**Response** Headers<sup>4</sup> Cookies Results Docs

Status: **200 OK** Size: **64 Bytes** Time: **440 ms**

**Consumo de Productos:** Implementa una función en el controlador que permita obtener todos los productos disponibles y devuelve la lista de productos en formato JSON.

```

@GetMapping
public ResponseEntity<List<Products>> getAllProducts() {
    List<Products> products = userService.getProducts();
}

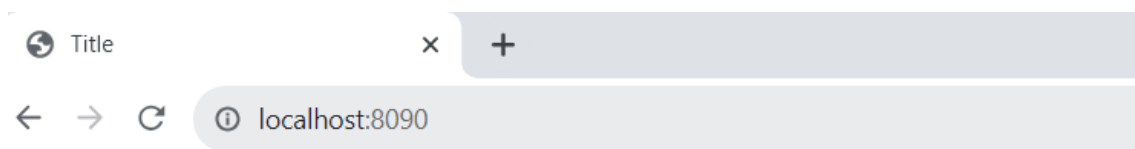
```

```
return new ResponseEntity<>(products, HttpStatus.OK);
}
```

**Crea una vista (puede ser una página HTML simple) que consuma esta API utilizando JavaScript y muestra la lista de productos en la tienda virtual.**

```
<> index.html x
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <h1>Productos en la Tienda Virtual</h1>
9      <ul id="productList"></ul>
10
11     <script>
12         async function fetchProducts() {
13             try {
14                 const response = await fetch('/api/products');
15                 const products = await response.json();
16
17                 const productList = document.getElementById('productList');
18
19                 products.forEach(product => {
20                     const listItem = document.createElement('li');
21                     listItem.innerHTML = `
22                         Nombre: ${product.name}, Precio: ${product.price}
23                         <button onclick="addToCart(${product.id})">Agregar al Carrito</button>
24                         <button onclick="showDetails(${product.id})">Ver Detalles</button>
25                         <button onclick="updateProduct(${product.id})">Actualizar Producto</button>
26                     `;
27                     productList.appendChild(listItem);
28                 });
29             } catch (error) {
30                 console.error('Error al obtener la lista de productos:', error);
            }
        }
    </script>
</body>
</html>
```

**Interacción con la Tienda:**



# Productos en la Tienda Virtual

