Manual de Usuario

Sistema de Gestión de Repuestos Automotrices

Mateo Benjamín Ortiz de Leon
IN5CM
Perito en Informática

Introducción

Este manual explica cómo funciona el sistema de gestión de repuestos automotrices. El proyecto está desarrollado con Spring Boot y permite administrar empleados, proveedores, repuestos y ventas a través de una API REST.

¿Qué es una API REST?

Una API REST es como un puente que permite que diferentes aplicaciones se comuniquen entre sí. En este caso, nuestra API permite crear, leer, actualizar y eliminar información de la base de datos usando peticiones HTTP (GET, POST, PUT, DELETE).

Estructura del Proyecto

1. Entity

Las entidades son clases que representan las tablas de la base de datos. Cada entidad tiene:

- • Atributos que corresponden a las columnas de la tabla
- • Anotaciones JPA que definen cómo se mapean a la base de datos
- • Getters y setters para acceder y modificar los datos

Ejemplo:

```
@Entity
@Table(name = "Empleados")
public class Empleado {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer idEmpleado;

    @NotBlank(message = "El nombre no puede estar vacío")
    private String nombreEmpleado;
    // ... más campos
}
```

Las anotaciones como @NotBlank validan que los datos sean correctos antes de guardarlos.

2. Repository (Repositorios)

Los repositorios son interfaces que se encargan de la comunicación con la base de datos. Spring Boot genera automáticamente los métodos necesarios (findAll, findById, save, delete).

```
@Repository
public interface EmpleadoRepository extends JpaRepository<Empleado, Integer> {
    // Spring Boot genera automáticamente los métodos
}
```

3. Service (Servicios)

Los servicios contienen la lógica de negocio. Hay dos partes:

- • Interface: Define qué métodos debe tener el servicio
- • Implementación: Contiene el código que ejecuta cada método

```
@Service
public class EmpleadoServiceImplements implements EmpleadoService {

    @Override
    public Empleado updateEmpleado(Integer id, Empleado empleado) {
        // 1. Buscar el empleado existente
        Empleado empleadoExistente = empleadoRepository.findById(id).orElse(null);

        // 2. Si no existe, devolver null
        if(empleadoExistente == null) return null;

        // 3. Actualizar los campos
        empleadoExistente.setNombreEmpleado(empleado.getNombreEmpleado());
        // ... actualizar más campos

        // 4. Guardar los cambios
        return empleadoRepository.save(empleadoExistente);
    }
}
```

4. Controller (Controladores)

Los controladores manejan las peticiones HTTP. Definen los endpoints (URLs) y procesan las solicitudes.

```
@RestController
@RequestMapping("/api/empleados")
public class EmpleadoController {

    @GetMapping("/{id}")
    public ResponseEntity<Object> getEmpleadoById(@PathVariable Integer id) {
        Empleado empleado = empleadoService.getEmpleadoById(id);
        if (empleado == null) {
            return ResponseEntity.status(404).body("Empleado no encontrado");
        }
        return ResponseEntity.ok(empleado);
    }
}
```

Flujo de una Petición

Cuando haces una petición en Postman, el flujo es el siguiente:

1. Postman envía la petición HTTP
  ↓
2. El Controller recibe la petición
  ↓
3. El Controller llama al Service
  ↓
4. El Service llama al Repository
  ↓
5. El Repository ejecuta la consulta en MySQL
  ↓
6. Los datos regresan por el mismo camino
  ↓
7. Postman recibe la respuesta en JSON

Ejemplo práctico: Crear un empleado

1. En Postman envías un POST a /api/empleados con este JSON:

```
{
    "nombreEmpleado": "Juan",
    "apellidoEmpleado": "Pérez",
    "puestoEmpleado": "Vendedor",
    "emailEmpleado": "juan@empresa.com"
}
```

2. El EmpleadoController recibe la petición

3. Valida que los campos no estén vacíos (@NotBlank)

4. Llama a empleadoService.saveEmpleado()

5. El servicio llama a empleadoRepository.save()

6. MySQL guarda el empleado y genera un ID automático

7. El empleado con su nuevo ID regresa al controlador

8. El controlador devuelve HTTP 201 CREATED con el empleado

Entidades y sus Relaciones

Empleados

Almacena la información de los empleados que trabajan en la tienda.

Campos: ID, nombre, apellido, puesto, email

Proveedores

Almacena la información de los proveedores que surten los repuestos.

Campos: ID, nombre, teléfono, dirección, email

Repuestos

Almacena la información de los repuestos disponibles para venta.

Campos: ID, nombre, categoría, precio compra, precio venta, proveedor

Relación: Un repuesto pertenece a un proveedor (@ManyToOne)

Ventas

Registra las ventas realizadas.

Campos: ID, fecha, cantidad, total, empleado, repuesto

Relaciones:

- • Una venta es realizada por un empleado (@ManyToOne)
- • Una venta incluye un repuesto (@ManyToOne)

Sistema de Validaciones

El sistema valida que los datos sean correctos antes de guardarlos en la base de datos.

Tipos de validaciones implementadas:

- • @NotBlank: El campo no puede estar vacío (para texto)
- • @NotNull: El campo no puede ser nulo
- • @Email: Valida que el email tenga formato correcto
- • @Min: El número debe ser mayor o igual al valor especificado
- • @DecimalMin: Para números decimales, debe ser mayor al mínimo

¿Qué pasa si los datos son inválidos?

Si intentas enviar datos incorrectos, el sistema devuelve un error 400 Bad Request con un mensaje explicando qué está mal.

Ejemplo de error:

```
{
    "nombreEmpleado": "El nombre no puede estar vacío",
    "emailEmpleado": "El email debe ser válido"
}
```

Lo mismo aplica para:

- • /api/proveedores
- • /api/repuestos
- • /api/ventas

Manejo de Errores

El sistema maneja diferentes tipos de errores:

Error 400 - Bad Request

Ocurre cuando los datos enviados son inválidos.

Ejemplo: Campo vacío, email inválido, cantidad negativa

Error 404 - Not Found

Ocurre cuando intentas buscar algo que no existe.

Ejemplo: Buscar empleado con ID 999 que no existe

Error 500 - Internal Server Error

Ocurre cuando hay un problema en el servidor.

Ejemplo: Error de conexión con la base de datos

Conexión rechazada

Si Postman muestra "Connection refused", significa que:

- • La aplicación Spring Boot no está corriendo
- • Solución: Iniciar la aplicación desde IntelliJ

Configuración del Proyecto

application.properties

Este archivo contiene la configuración de la aplicación:

```
spring.datasource.url=jdbc:mysql://localhost:3306/DBRepuestosAutomotriz_in5cm
spring.datasource.username=root
spring.datasource.password=1234
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Cómo Usar el Sistema

1. Iniciar la aplicación

1. • Abre el proyecto en IntelliJ IDEA
2. • Asegúrate de que MySQL esté corriendo
3. • Ejecuta RepuestosYAutomotricesApplication.java
4. • Espera el mensaje: "Started RepuestosYAutomotricesApplication"

2. Probar en Postman

5. • Abre Postman
6. • Crea una petición GET a: http://localhost:8080/api/empleados
7. • Dale "Send"
8. • Deberías ver la lista de empleados

3. Crear un registro

9. • Cambia el método a POST
10. • En Headers agrega: Content-Type = application/json
11. • En Body (raw - JSON) escribe los datos
12. • Depues "Send"