

“Informe Proyecto Perfulandia”

Sección:001D

Docente: Viviana Poblete

Integrantes: Javier Albornoz

Vicente Hormazabal

Diego Neira

Índice

“Informe Proyecto Perfulandia”	1
Índice	2
Introducción	3
Diagrama de arquitectura de microservicios	4
Estructura del proyecto	5
DEPENDENCIAS:	6
COMPONENTES IMPLEMENTADOS:	7
Base de datos	14
Implementación de los servicios	16
Implementación de vistas que permitan desplegar petición get de cada servicio.	19
Git – Git Hub	22
Conclusión	25

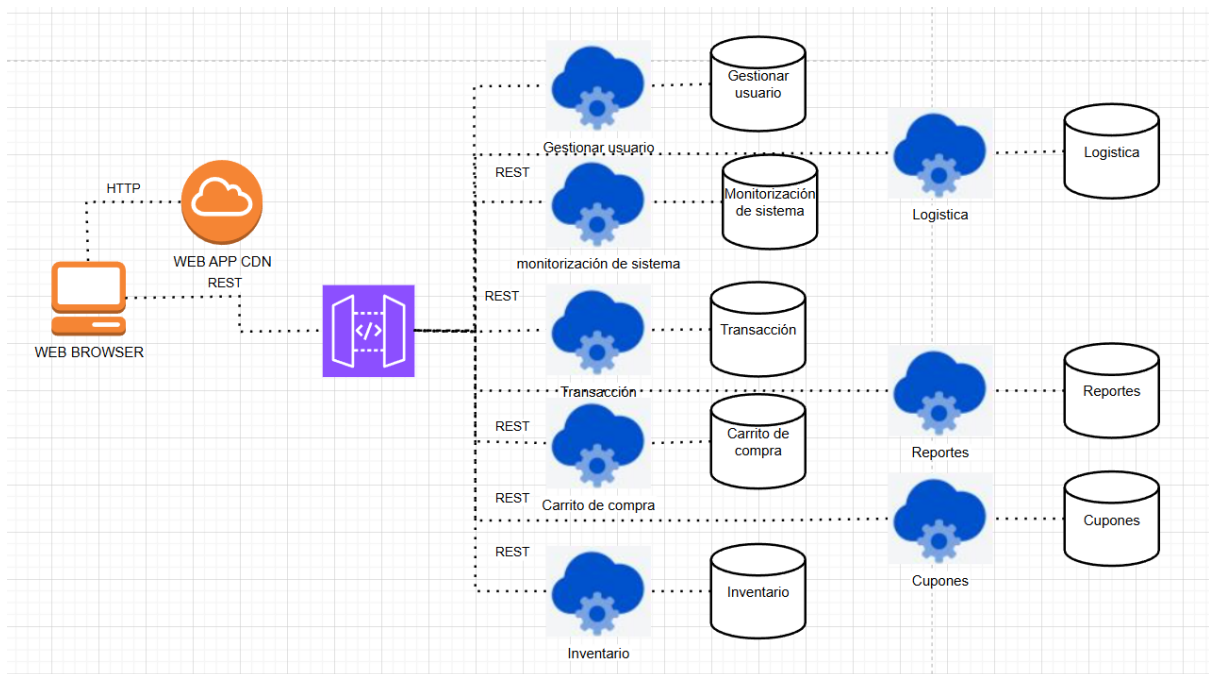
Introducción

En la anterior evaluación nos presentaron el caso de perfulandia, una empresa emergente que destaca por ofrecer productos de alta calidad a precios competitivos,

Perfulandia tiene un problema, su rápido crecimiento ha revelado las limitaciones de su sistema actual de software monolítico. El sistema monolítico de perfulandia a empezado a presentar problemas de rendimiento y de disponibilidad que ponen en riesgo las operaciones diarias y sobretodo la satisfacción del cliente.

En este informe detallaremos nuestra solución ofrecida a perfulandia la cual está basada en un sistema de Microservicios, detallaremos Nuestro diagrama de arquitectura de microservicios , toda la estructura de nuestro proyecto ,git y github, la implementación de nuestros servicios, la implementación de las 3 vistas y lo más importante la base de datos.

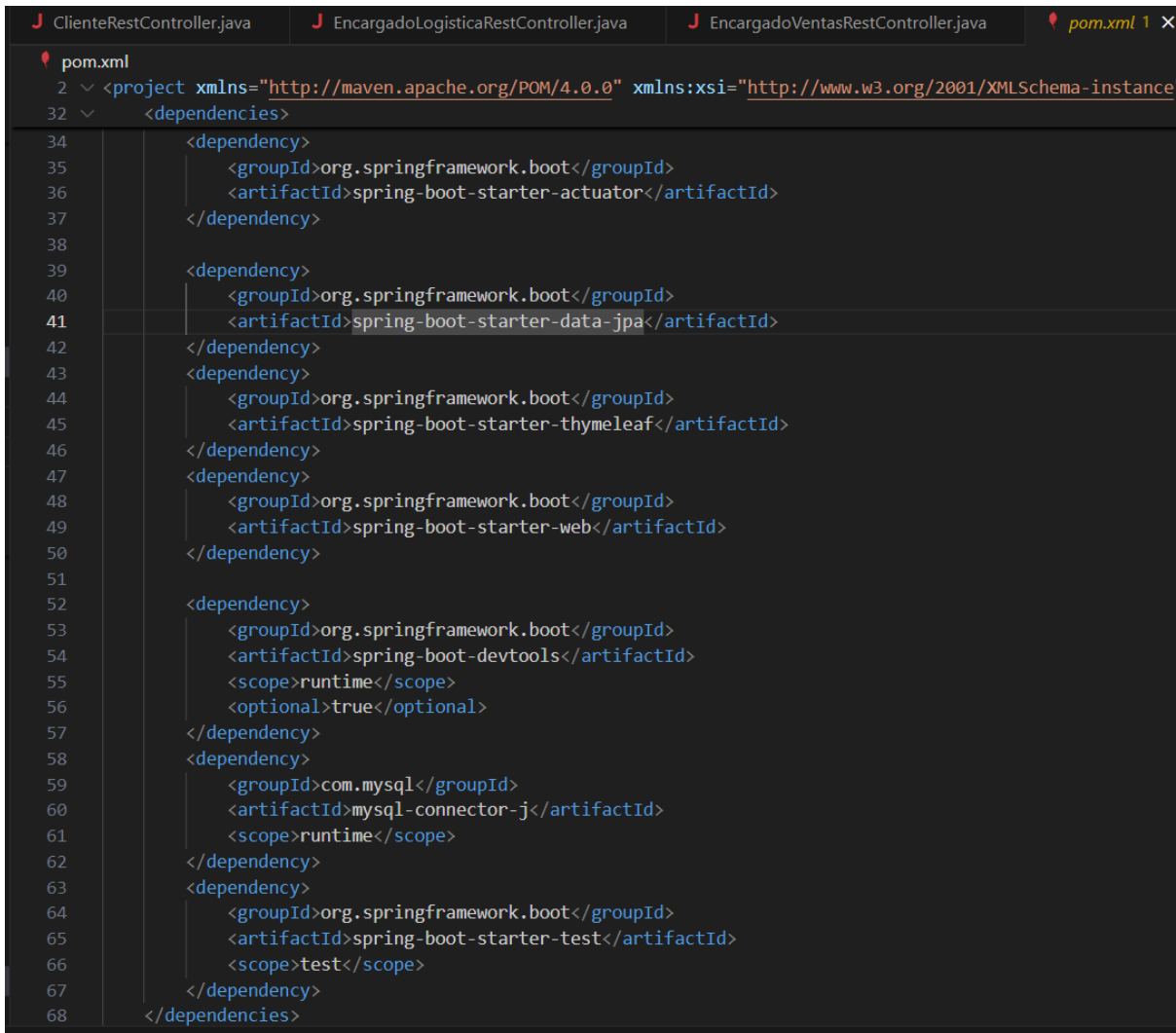
Diagrama de arquitectura de microservicios



En el diagrama de arquitectura de microservicios podemos visualizar los servicios con sus respectivas bases de datos, en nuestro diagrama aparecen 8 bases de datos, pero por el momento solo se crearon 3 (por rúbrica), también podemos ver cómo son dirigidos por api gateway que facilita distintos servicios.

Estructura del proyecto

DEPENDENCIAS:



```
1 pom.xml
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
32 <dependencies>
34 <dependency>
35 <groupId>org.springframework.boot</groupId>
36 <artifactId>spring-boot-starter-actuator</artifactId>
37 </dependency>
38
39 <dependency>
40 <groupId>org.springframework.boot</groupId>
41 <artifactId>spring-boot-starter-data-jpa</artifactId>
42 </dependency>
43 <dependency>
44 <groupId>org.springframework.boot</groupId>
45 <artifactId>spring-boot-starter-thymeleaf</artifactId>
46 </dependency>
47 <dependency>
48 <groupId>org.springframework.boot</groupId>
49 <artifactId>spring-boot-starter-web</artifactId>
50 </dependency>
51
52 <dependency>
53 <groupId>org.springframework.boot</groupId>
54 <artifactId>spring-boot-devtools</artifactId>
55 <scope>runtime</scope>
56 <optional>true</optional>
57 </dependency>
58 <dependency>
59 <groupId>com.mysql</groupId>
60 <artifactId>mysql-connector-j</artifactId>
61 <scope>runtime</scope>
62 </dependency>
63 <dependency>
64 <groupId>org.springframework.boot</groupId>
65 <artifactId>spring-boot-starter-test</artifactId>
66 <scope>test</scope>
67 </dependency>
68 </dependencies>
```

Aca en el archivo pom se pueden ver las dependencias que se estan utilizando, entre las cuales estan:

1. **Spring Boot DevTools**, sirve para:
 - Mejorar la experiencia de desarrollo, reinicia automáticamente la aplicación al detectar cambios en el código, desactiva la caché de plantillas como thymeleaf.
2. **Spring Web**:, sirve para:
 - Construir aplicaciones web y apis restful.

- Incluye el soporte para controladores: (@Controller, @RestController), para rutas: (@GetMapping, @PostMapping), y manejo de solicitudes http.
- Usa internamente Spring MVC.

3. **Thymeleaf**, sirve para:

Es un motor de plantillas que genera html desde el backend.
Se integra muy bien con Spring Boot
Permite insertar datos en la vista usando th:text y th:each.

4. **Spring data JPA**, sirve para:

- Simplifica el acceso y gestión de bases de datos relacionales.
- Permite usar interfaces sin necesidad de escribir código sql.
- Puede mapear clases java a tablas con @Entity, @Id.

5. **MySQL Driver**, sirve para:

- Es el conector JDBC que permite que tu aplicación Java se comuniquen con una base de datos MySQL.

COMPONENTES IMPLEMENTADOS:

Tenemos 3 microservicios los cuales son **Usuarios, Productos y sucursales**.

Dentro de ellos tenemos los componentes:

1. **Entities:** Contiene las clases modelo que representan las entidades persistentes en la base de datos. Cada entidad está anotada con @Entity y cuenta con atributos.

Ejemplo:

```

src > main > java > com > javier > springboot > ejemplo > perfulandia > entities > Cliente.java > Cliente > password
1 package com.javier.springboot.ejemplo.perfulandia.entities;
2
3 import jakarta.persistence.Entity;
4
5 @Entity
6 public class Cliente extends Usuario {
7
8     private String direccion;
9     private String comuna;
10    private String telefono;
11    private String email;
12    private String password;
13
14    public Cliente(String rut, String nombre, String apellido, String direccion, String comuna, String telefono, String email, String password) {
15        super(rut, nombre, apellido, direccion, comuna, telefono, email, password);
16        this.direccion = direccion;
17        this.comuna = comuna;
18        this.telefono = telefono;
19        this.email = email;
20        this.password = password;
21    }
22    public Cliente() {
23    }
24
25    public String getDireccion() {
26        return direccion;
27    }
28
29    public void setDireccion(String direccion) {
30        this.direccion = direccion;
31    }
32
33    public String getComuna() {
34        return comuna;
35    }
36
37    public void setComuna(String comuna) {

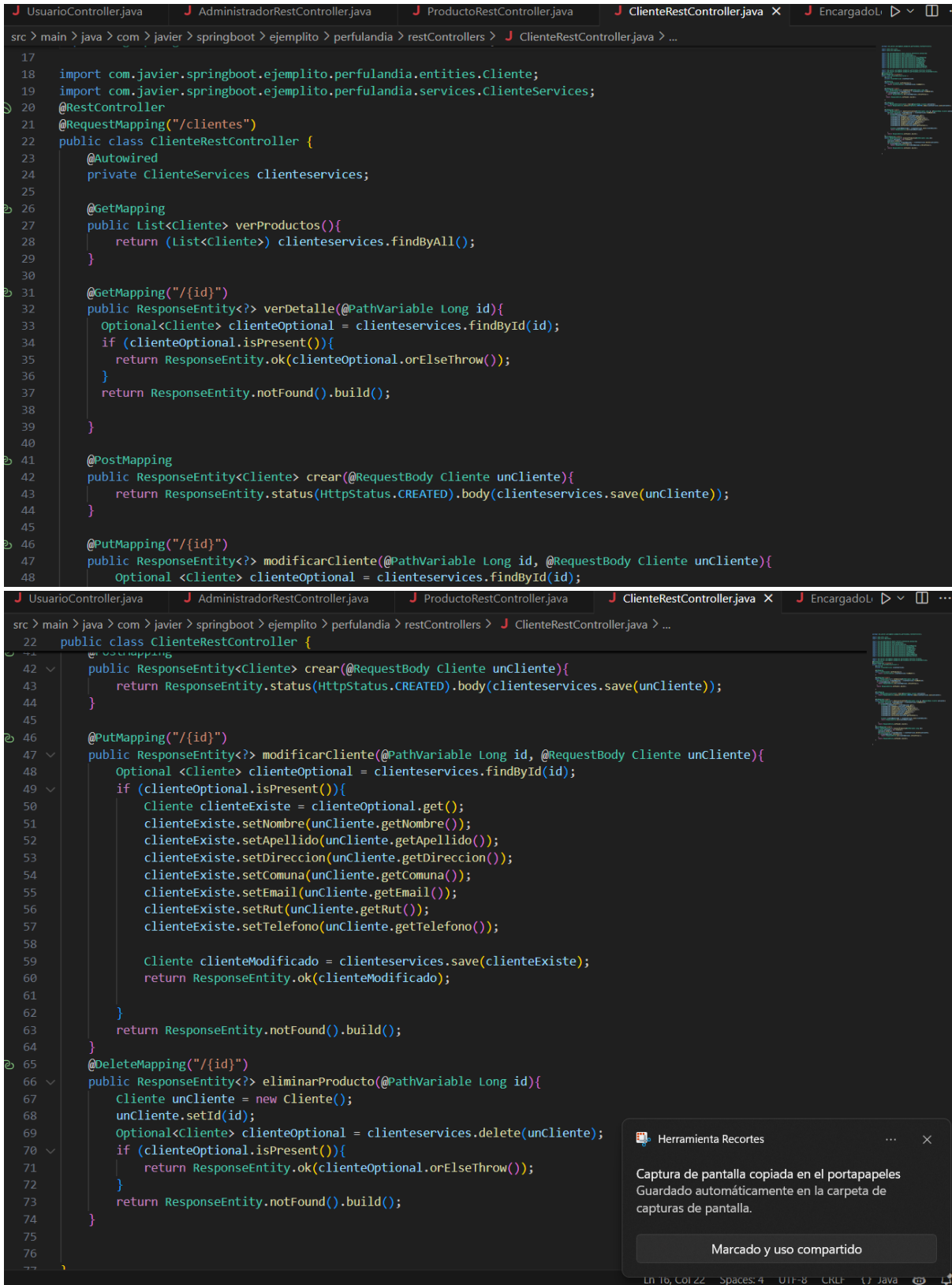
```

2. Controllers: Este package se usa para manejar las vistas HTML.

Ejemplo:

```
J UsuarioController.java x J AdministradorRestController.java J ProductoRestController.java J ClienteRestController.java J EncargadoL...
src > main > java > com > javier > springboot > ejemplito > perfulandia > controllers > J UsuarioController.java > Language Support for Java(TM) by Red Hat > UsuarioControll
6 import org.springframework.web.bind.annotation.GetMapping;
7
8 import com.javier.springboot.ejemplito.perfulandia.repositories.AdministradorRepository;
9 import com.javier.springboot.ejemplito.perfulandia.repositories.ClienteRepository;
10 import com.javier.springboot.ejemplito.perfulandia.repositories.EncargadoLogisticaRepository;
11 import com.javier.springboot.ejemplito.perfulandia.repositories.EncargadoVentasRepository;
12 import com.javier.springboot.ejemplito.perfulandia.repositories.GerenteRepository;
13
14
15 @Controller
16 public class UsuarioController {
17
18     @Autowired
19     private ClienteRepository clienteRepository;
20
21     @Autowired
22     private GerenteRepository gerenteRepository;
23
24     @Autowired
25     private AdministradorRepository administradorRepository;
26
27     @Autowired
28     private EncargadoVentasRepository encargadoVentasRepository;
29
30     @Autowired
31     private EncargadoLogisticaRepository encargadoLogisticaRepository;
32
33     @GetMapping("/cliente")
34     public String mostrarUsuarios(Model model) {
35         model.addAttribute(attributeName:"clientes", clienteRepository.findAll());
36         model.addAttribute(attributeName:"gerentes", gerenteRepository.findAll());
37         model.addAttribute(attributeName:"administrador", administradorRepository.findAll());
38         model.addAttribute(attributeName:"ventas", encargadoVentasRepository.findAll());
39         model.addAttribute(attributeName:"logisticas", encargadoLogisticaRepository.findAll());
40         return "cliente";
41     }
42 }
```

3. **RestController:** Aquí se implementan los endpoints REST utilizando `@RestController`, para permitir la comunicación con el frontend.
Ejemplo:



```

17
18 import com.javier.springboot.ejemplito.perfulandia.entities.Cliente;
19 import com.javier.springboot.ejemplito.perfulandia.services.ClienteServices;
20 @RestController
21 @RequestMapping("/clientes")
22 public class ClienteRestController {
23     @Autowired
24     private ClienteServices clienteservices;
25
26     @GetMapping
27     public List<Cliente> verProductos(){
28         return (List<Cliente>) clienteservices.findAll();
29     }
30
31     @GetMapping("/{id}")
32     public ResponseEntity<?> verDetalle(@PathVariable Long id){
33         Optional<Cliente> clienteOptional = clienteservices.findById(id);
34         if (clienteOptional.isPresent()){
35             return ResponseEntity.ok(clienteOptional.orElseThrow());
36         }
37         return ResponseEntity.notFound().build();
38     }
39
40
41     @PostMapping
42     public ResponseEntity<Cliente> crear(@RequestBody Cliente unCliente){
43         return ResponseEntity.status(HttpStatus.CREATED).body(clienteservices.save(unCliente));
44     }
45
46     @PutMapping("/{id}")
47     public ResponseEntity<?> modificarCliente(@PathVariable Long id, @RequestBody Cliente unCliente){
48         Optional <Cliente> clienteOptional = clienteservices.findById(id);
49         if (clienteOptional.isPresent()){
50             Cliente clienteExiste = clienteOptional.get();
51             clienteExiste.setNombre(unCliente.getNombre());
52             clienteExiste.setApellido(unCliente.getApellido());
53             clienteExiste.setDireccion(unCliente.getDireccion());
54             clienteExiste.setComuna(unCliente.getComuna());
55             clienteExiste.setEmail(unCliente.getEmail());
56             clienteExiste.setRut(unCliente.getRut());
57             clienteExiste.setTelefono(unCliente.getTelefono());
58
59             Cliente clienteModificado = clienteservices.save(clienteExiste);
60             return ResponseEntity.ok(clienteModificado);
61         }
62         return ResponseEntity.notFound().build();
63     }
64
65     @DeleteMapping("/{id}")
66     public ResponseEntity<?> eliminarProducto(@PathVariable Long id){
67         Cliente unCliente = new Cliente();
68         unCliente.setId(id);
69         Optional<Cliente> clienteOptional = clienteservices.delete(unCliente);
70         if (clienteOptional.isPresent()){
71             return ResponseEntity.ok(clienteOptional.orElseThrow());
72         }
73         return ResponseEntity.notFound().build();
74     }
75
76
77

```

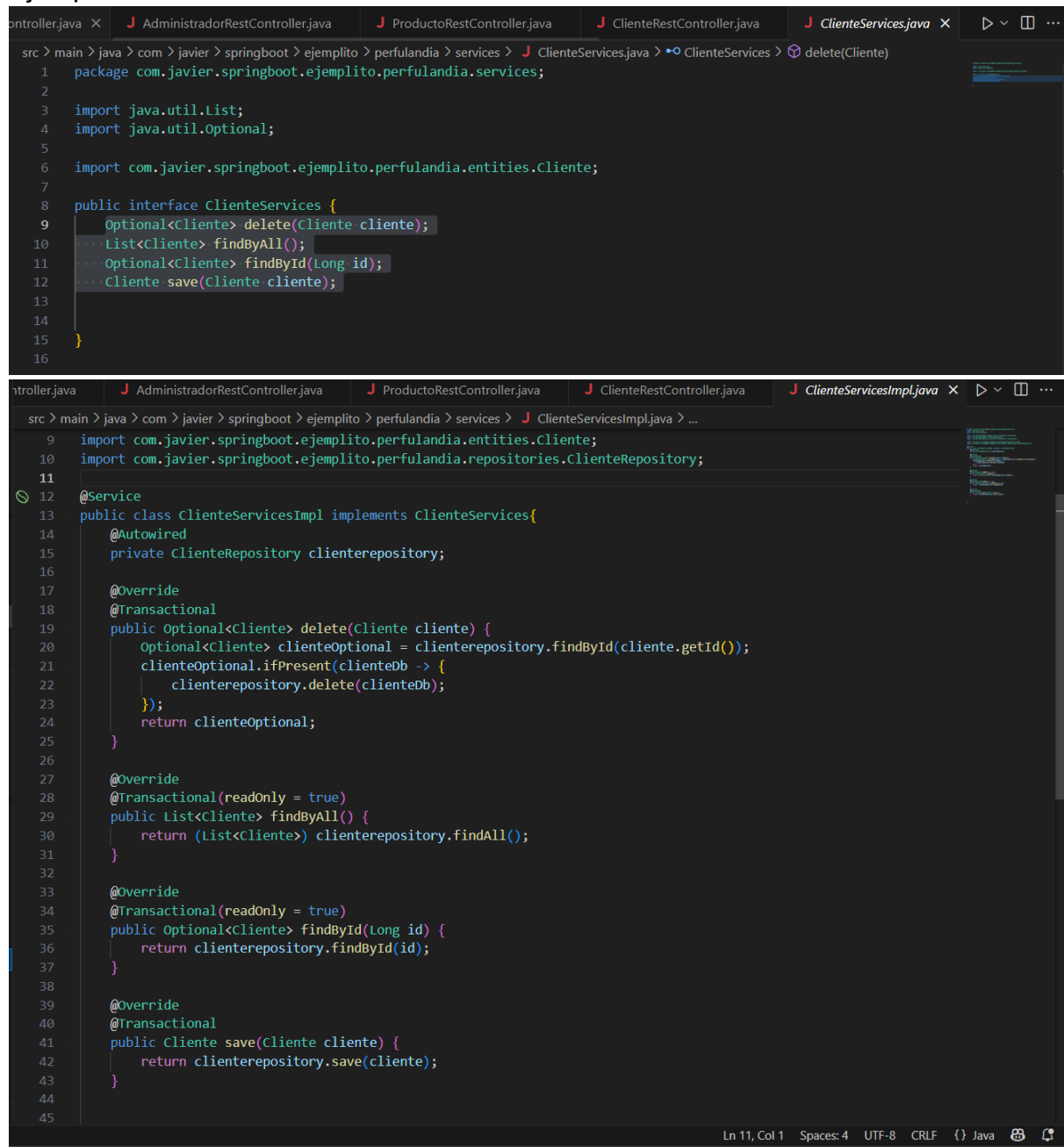
4. Repositories: Contiene interfaz que extiende JpaRepository, permitiendo así la interacción con la base de datos mediante operaciones crud.

Ejemplo:

```
ontroller.java | AdministradorRestController.java | ProductoRestController.java | ClienteRestController.java | ClienteRepository.java x | ...
src > main > java > com > javier > springboot > ejemplo > perfulandia > repositories > ClienteRepository.java > Language Support for Java(TM) by Red Hat > ClienteReposit
1 package com.javier.springboot.ejemplo.perfulandia.repositories;
2
3 import org.springframework.data.repository.CrudRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.javier.springboot.ejemplo.perfulandia.entities.Cliente;
7 @Repository
8 public interface ClienteRepository extends CrudRepository<Cliente, Long>{
9
10 }
11
```

5. **Services:** Este paquete encapsula la lógica de negocio, implementando la interacción entre los controladores y el repositorio.

Ejemplo:



```

src > main > java > com > javier > springboot > ejemplito > perfulandia > services > J ClienteServices.java > ClienteServices > delete(Cliente)
1 package com.javier.springboot.ejemplito.perfulandia.services;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import com.javier.springboot.ejemplito.perfulandia.entities.Cliente;
7
8 public interface ClienteServices {
9     Optional<Cliente> delete(Cliente cliente);
10    List<Cliente> findByAll();
11    Optional<Cliente> findById(Long id);
12    Cliente save(Cliente cliente);
13
14
15 }
16

ntroller.java J AdministradorRestController.java J ProductoRestController.java J ClienteRestController.java J ClienteServicesImpl.java
src > main > java > com > javier > springboot > ejemplito > perfulandia > services > J ClienteServicesImpl.java > ...
9 import com.javier.springboot.ejemplito.perfulandia.entities.Cliente;
10 import com.javier.springboot.ejemplito.perfulandia.repositories.ClienteRepository;
11
12 @Service
13 public class ClienteServicesImpl implements ClienteServices{
14     @Autowired
15     private ClienteRepository clienterepository;
16
17     @Override
18     @Transactional
19     public Optional<Cliente> delete(Cliente cliente) {
20         Optional<Cliente> clienteOptional = clienterepository.findById(cliente.getId());
21         clienteOptional.ifPresent(clienteDb -> {
22             clienterepository.delete(clienteDb);
23         });
24         return clienteOptional;
25     }
26
27     @Override
28     @Transactional(readOnly = true)
29     public List<Cliente> findByAll() {
30         return (List<Cliente>) clienterepository.findAll();
31     }
32
33     @Override
34     @Transactional(readOnly = true)
35     public Optional<Cliente> findById(Long id) {
36         return clienterepository.findById(id);
37     }
38
39     @Override
40     @Transactional
41     public Cliente save(Cliente cliente) {
42         return clienterepository.save(cliente);
43     }
44
45

```

6. HTML (Vistas): Sirve para mostrar los datos. Se accede a este a través del controlador.

Ejemplo:

```

cliente.html x UsuarioController.java AdministradorRestController.java ProductoRestController.java ClienteRestController.java
src > main > resources > templates > cliente.html > html > body.p-4 > div.container > table.table.table-striped.table-bordered > tbody > tr > td
1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Usuarios del Sistema</title>
7 <!-- Bootstrap CDN -->
8 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
9 <style>
10 body {
11     background-color: #2c2f33; /* Gris oscuro */
12     color: #f1f1f1;
13 }
14 h1, h2 {
15     color: #ffffff;
16 }
17 .table {
18     background-color: #3b3f44;
19     color: #ffffff;
20 }
21 .table th, .table td {
22     border-color: #6c757d;
23 }
24 .table-striped > tbody > tr:nth-of-type(odd) {
25     background-color: #2f3338;
26 }
27 .table-striped > tbody > tr:nth-of-type(even) {
28     background-color: #3b3f44;
29 }
30 </style>
31 </head>
32 <body class="p-4">
33 <div class="container">
34 <h1 class="text-center mb-4">Usuarios del Sistema</h1>
35
36 <!-- Clientes -->
37 <h2 class="text-primary">Clientes</h2>

```

```

cliente.html x UsuarioController.java AdministradorRestController.java ProductoRestController.java ClienteRestController.java
src > main > resources > templates > cliente.html > html > body.p-4 > div.container > table.table.table-striped.table-bordered > tbody > tr > td
31 </head>
32 <body class="p-4">
33 <div class="container">
34 <h1 class="text-center mb-4">Usuarios del Sistema</h1>
35
36 <!-- Clientes -->
37 <h2 class="text-primary">Clientes</h2>
38 <table class="table table-striped table-bordered">
39 <thead class="table-dark">
40 <tr>
41 <th>ID</th>
42 <th>Nombre</th>
43 <th>Apellido</th>
44 <th>Email</th>
45 </tr>
46 </thead>
47 <tbody>
48 <tr th:each="cliente : ${clientes}">
49 <td th:text="${cliente.id}"></td>
50 <td th:text="${cliente.nombre}"></td>
51 <td th:text="${cliente.apellido}"></td>
52 <td th:text="${cliente.email}"></td>
53 </tr>
54 </tbody>
55 </table>
56
57 <!-- Gerentes -->
58 <h2 class="text-success">Gerentes</h2>
59 <table class="table table-striped table-bordered">
60 <thead class="table-dark">
61 <tr>
62 <th>ID</th>
63 <th>Nombre</th>
64 <th>Apellido</th>
65 <th>Email</th>
66 </tr>

```

```

src > main > resources > templates > cliente.html > html > body.p-4 > div.container > table.table.table-striped.table-bordered > tbody > tr > td
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
32 <body class="p-4">
33   <div class="container">
34     <table class="table table-striped table-bordered">
47       <tbody>
51         <tr>
52           <td th:text="{cliente.apellido}"></td>
53           <td th:text="{cliente.email}"></td>
54         </tr>
55       </tbody>
56     </table>
57
58     <!-- Gerentes -->
59     <h2 class="text-success">Gerentes</h2>
60     <table class="table table-striped table-bordered">
61       <thead class="table-dark">
62         <tr>
63           <th>ID</th>
64           <th>Nombre</th>
65           <th>Apellido</th>
66           <th>Email</th>
67         </tr>
68       </thead>
69       <tbody>
70         <tr th:each="gerente : ${gerentes}">
71           <td th:text="{gerente.id}"></td>
72           <td th:text="{gerente.nombre}"></td>
73           <td th:text="{gerente.apellido}"></td>
74           <td th:text="{gerente.email}"></td>
75         </tr>
76       </tbody>
77     </table>
78
79     <!-- Administradores -->
80     <h2 class="text-danger">Administradores</h2>
81     <table class="table table-striped table-bordered">
82       <thead class="table-dark">

```

```

src > main > resources > templates > cliente.html > html > body.p-4 > div.container > table.table.table-striped.table-bordered > tbody > tr > td
32 <body class="p-4">
33   <div class="container">
78     <!-- Administradores -->
79     <h2 class="text-danger">Administradores</h2>
80     <table class="table table-striped table-bordered">
81       <thead class="table-dark">
82         <tr>
83           <th>ID</th>
84           <th>Nombre</th>
85           <th>Apellido</th>
86           <th>Email</th>
87         </tr>
88       </thead>
89       <tbody>
90         <tr th:each="admin : ${administrador}">
91           <td th:text="{admin.id}"></td>
92           <td th:text="{admin.nombre}"></td>
93           <td th:text="{admin.apellido}"></td>
94           <td th:text="{admin.email}"></td>
95         </tr>
96       </tbody>
97     </table>
98
99     <!-- Encargados de Ventas -->
100    <h2 class="text-warning">Encargados de Ventas</h2>
101    <table class="table table-striped table-bordered">
102      <thead class="table-dark">
103        <tr>
104          <th>ID</th>
105          <th>Nombre</th>
106          <th>Apellido</th>
107          <th>Email</th>
108        </tr>
109      </thead>
110      <tbody>
111        <tr th:each="ventas : ${ventas1}">

```

```

<?xml version="1.0" encoding="UTF-8"?>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
  <body class="p-4">
    <div class="container">
      <table class="table table-striped table-bordered">
        <tbody>
          <tr th:each="ventas : ${ventas}">
            <td th:text="${ventas.id}"></td>
            <td th:text="${ventas.nombre}"></td>
            <td th:text="${ventas.apellido}"></td>
            <td th:text="${ventas.email}"></td>
          </tr>
        </tbody>
      </table>
      <!-- Encargados de Logística -->
      <h2 class="text-info">Encargados de Logística</h2>
      <table class="table table-striped table-bordered">
        <thead class="table-dark">
          <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Apellido</th>
            <th>Email</th>
          </tr>
        </thead>
        <tbody>
          <tr th:each="logistica : ${logisticas}">
            <td th:text="${logistica.id}"></td>
            <td th:text="${logistica.nombre}"></td>
            <td th:text="${logistica.apellido}"></td>
            <td th:text="${logistica.email}"></td>
          </tr>
        </tbody>
      </table>
    </div>
  </body>
</html>

```

Base de datos

Utilizamos MySQL (Es un sistema de gestión de bases de datos relacional de código abierto. Entre sus funciones principales están: almacenar, organizar y recuperar datos de forma eficiente utilizando el lenguaje SQL) como motor de base de datos, conectado localmente mediante Spring Data JPA.

Estructura de base de datos: Cada microservicio (Usuarios, Productos, Sucursales) tendrá su propia base de datos para así asegurar la modularidad y eficiente escalabilidad.

Tablas:

Tabla de Usuario/ Administrador:

	id	apellido	comuna	direccion	email	nombre	password	rut	telefono
▶	301	Navarro	NULL	NULL	elena.navarro@example.com	Elena	NULL	NULL	NULL
	302	Herrera	NULL	NULL	tomas.herrera@example.com	Tomás	NULL	NULL	NULL
	303	Castro	NULL	NULL	camila.castro@example.com	Camila	NULL	NULL	NULL
	304	Molina	NULL	NULL	francisco.molina@example.com	Francisco	NULL	NULL	NULL
	305	González	NULL	NULL	matias.gonzalez@example.com	Matías	NULL	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla de Productos:

	id	descripcion	nombre	precio
▶	1	Lámpara LED con brazo flexible	Lámpara flexible	14990
	2	Mochila resistente al agua de 20 litros	Mochila deportiva	29990
	3	Juego de cubiertos de acero inoxidable, 24 piezas	Set de cubiertos	19990
	4	Auriculares Bluetooth con cancelación de ruido	Auriculares inalámbricos	49990
	5	Taza de acero inoxidable para mantener la tem...	Taza térmica	9990
*	NULL	NULL	NULL	NULL

Tabla de Sucursales:

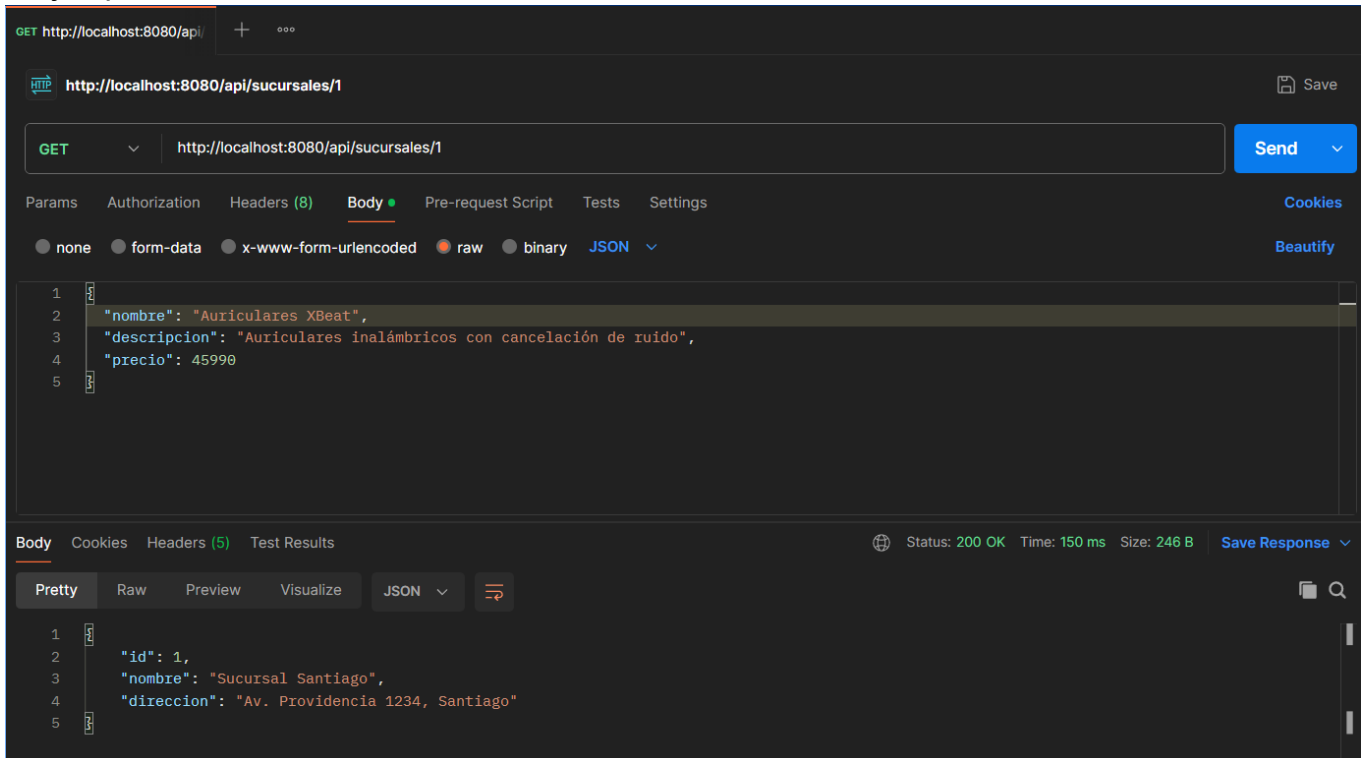
	id	direccion	nombre
▶	1	Av. Providencia 1234, Santiago	Sucursal Santiago
	2	Calle Valparaíso 567, Viña del Mar	Sucursal Viña del Mar
	3	O'Higgins 890, Concepción	Sucursal Concepción
	6	Av. Apoquindo 4567	Sucursal Las Condes
*	NULL	NULL	NULL

Implementación de los servicios

En este proyecto se utilizan los servicios de:

1. GET
2. POST
3. PUT
4. DELETE

1.Ejemplo de GET:



The screenshot shows a REST client interface with a GET request to `http://localhost:8080/api/sucursales/1`. The response is a JSON object with the following structure:

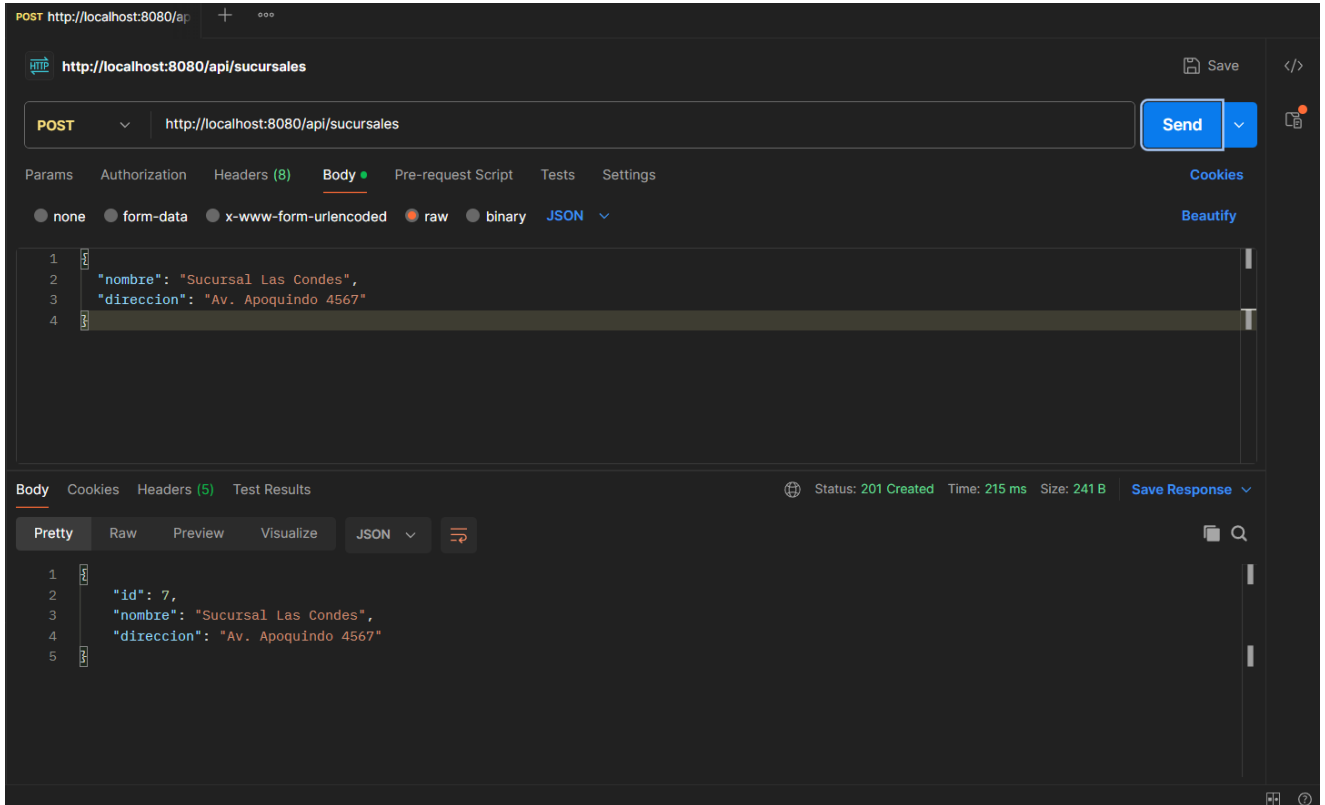
```
1 {
2   "nombre": "Auriculares XBeat",
3   "descripcion": "Auriculares inalámbricos con cancelación de ruido",
4   "precio": 45990
5 }
```

The bottom section shows the response body in a pretty-printed JSON format:

```
1 {
2   "id": 1,
3   "nombre": "Sucursal Santiago",
4   "direccion": "Av. Providencia 1234, Santiago"
5 }
```

Additional details visible in the interface include a status of 200 OK, a time of 150 ms, and a size of 246 B.

2.Ejemplo de POST:



POST http://localhost:8080/api/sucursales

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "nombre": "Sucursal Las Condes",
3   "direccion": "Av. Apoquindo 4567"
4 }
```

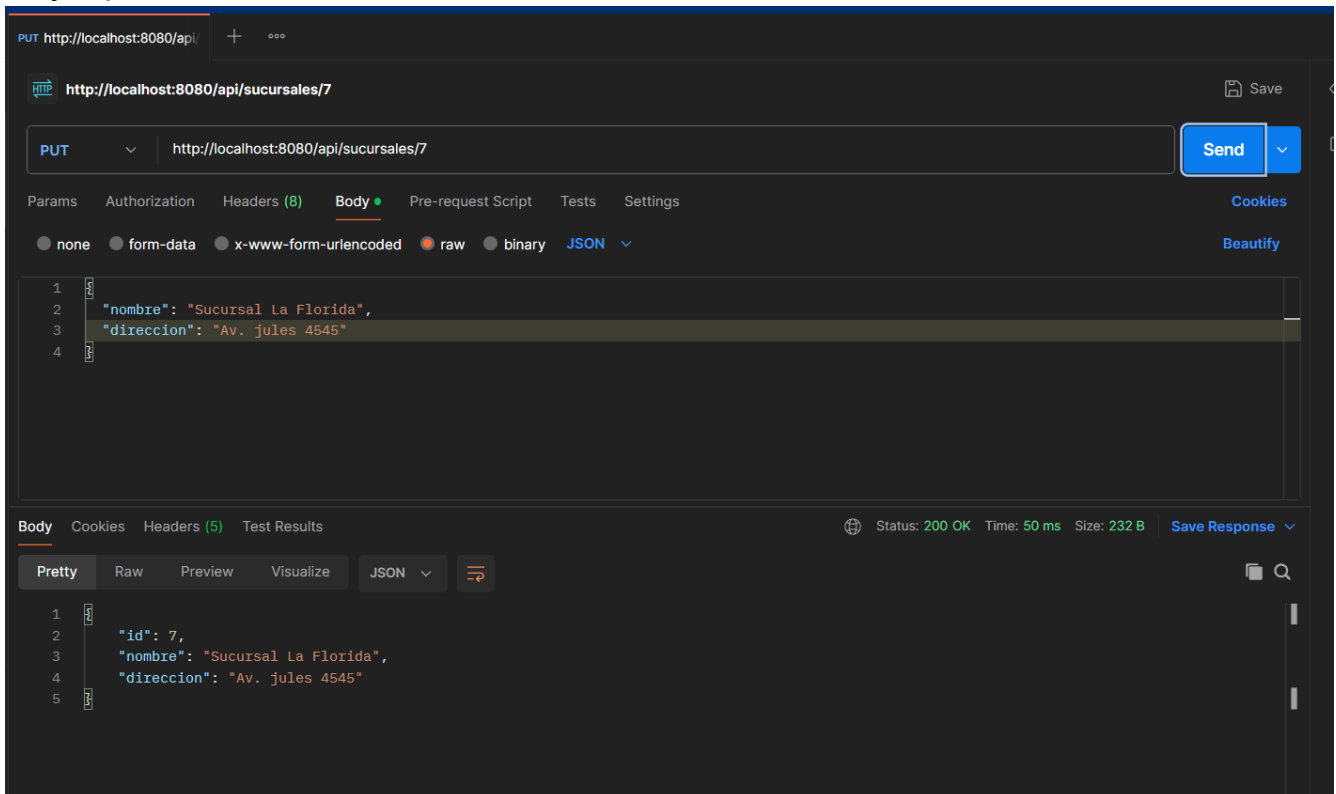
Body Cookies Headers (5) Test Results

Status: 201 Created Time: 215 ms Size: 241 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 7,
3   "nombre": "Sucursal Las Condes",
4   "direccion": "Av. Apoquindo 4567"
5 }
```

3.Ejemplo de PUT:



PUT http://localhost:8080/api/sucursales/7

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "nombre": "Sucursal La Florida",
3   "direccion": "Av. jules 4545"
4 }
```

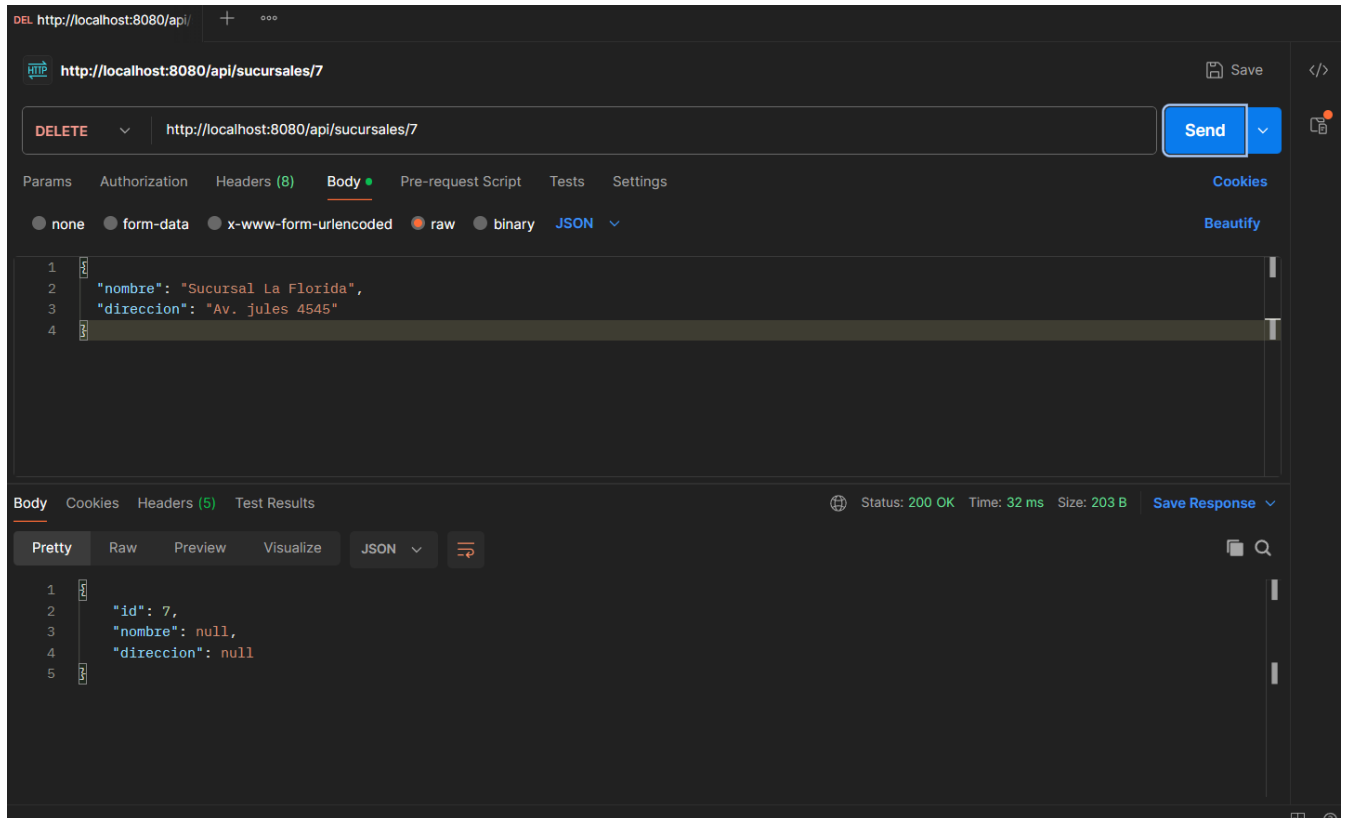
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 50 ms Size: 232 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 7,
3   "nombre": "Sucursal La Florida",
4   "direccion": "Av. jules 4545"
5 }
```

4. Ejemplo de DELETE:



The screenshot displays a REST client interface with a DELETE request to `http://localhost:8080/api/sucursales/7`. The request body is a JSON object: `{ "nombre": "Sucursal La Florida", "direccion": "Av. jules 4545" }`. The response status is `200 OK` with a time of `32 ms` and a size of `203 B`. The response body is a JSON object: `{ "id": 7, "nombre": null, "direccion": null }`.

Request:

```
DELETE http://localhost:8080/api/sucursales/7
```

Body:

```
{
  "nombre": "Sucursal La Florida",
  "direccion": "Av. jules 4545"
}
```

Response:

```
{
  "id": 7,
  "nombre": null,
  "direccion": null
}
```

Vista Usuarios:

Usuarios del Sistema

Clientes

ID	Nombre	Apellido	Email
2	Juan	Pérez	juan.perez@email.com
101	Carlos	Muñoz	carlos.munoz@example.com
102	Lucía	Pérez	lucia.perez@example.com
103	Javier	Rojas	javier.rojas@example.com
104	Fernanda	López	fernanda.lopez@example.com

Gerentes

ID	Nombre	Apellido	Email
201	Paula	González	paula.gonzalez@example.com
202	Andrés	Salazar	andres.salazar@example.com
203	María	Contreras	maria.contreras@example.com
204	Rodrigo	Vega	rodrigo.vega@example.com

Administradores

ID	Nombre	Apellido	Email
----	--------	----------	-------

Administradores

ID	Nombre	Apellido	Email
301	Elena	Navarro	elena.navarro@example.com
302	Tomás	Herrera	tomas.herrera@example.com
303	Camila	Castro	camila.castro@example.com
304	Francisco	Molina	francisco.molina@example.com
305	Matías	González	matias.gonzalez@example.com

Encargados de Ventas

ID	Nombre	Apellido	Email
401	Natalia	Silva	natalia.silva@example.com
402	Diego	León	diego.leon@example.com
403	Valentina	Araya	valentina.araya@example.com
404	Matías	Fernández	matias.fernandez@example.com

Encargados de Logística

ID	Nombre	Apellido	Email
501	Ignacio	Paredes	ignacio.paredes@example.com
502	Sofía	Carrasco	sofia.carrasco@example.com

Vista de Productos:

ID Producto	Nombre	Descripción	Precio
1	Lámpara flexible	Lámpara LED con brazo flexible	14990.0
2	Mochila deportiva	Mochila resistente al agua de 20 litros	29990.0
3	Set de cubiertos	Juego de cubiertos de acero inoxidable, 24 piezas	19990.0
4	Auriculares inalámbricos	Auriculares Bluetooth con cancelación de ruido	49990.0
5	Taza térmica	Taza de acero inoxidable para mantener la temperatura	9990.0

Vista de Sucursales:



ID	Nombre	Dirección
1	Sucursal Santiago	Av. Providencia 1234, Santiago
2	Sucursal Viña del Mar	Calle Valparaíso 567, Viña del Mar
3	Sucursal Concepción	O'Higgins 890, Concepción
6	Sucursal Las Condes	Av. Apoquindo 4567

Git – Git Hub

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio
$ cd perfulandia

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git init
Reinitialized existing Git repository in C:/Users/javie/OneDrive/Escritorio/perfulandia/.git/

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git config user.name javier14-2

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git config user.email javierluciano14@gmail.com

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ |
```


Primero vamos a la ruta donde se encuentra el proyecto.


Luego inicializamos el repositorio con las respectivas credenciales.

Personal access tokens (classic)

Generate new token ▾

Tokens you have generated that can be used to access the [GitHub API](#).

 Make sure to copy your personal access token now. You won't be able to see it again!


✓ ghp_I6hgRLbyOfs90SDXImqHXtZVWF1aX10d00Rq 

Delete

holachicos — repo

Last used within the last week

Delete

 This token has no expiration date.

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Creamos el token desde github

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git remote add origin https://github.com/havier14-2/Seccion-001D-Mayo-27.git

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
  add

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git add .
warning: in the working copy of 'pom.xml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/main/resources/application.properties', LF will be replaced by CRLF the next time Git touches it

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git commit -m "Implementación de servicios REST"
[master 313d796] "Implementación de servicios REST"
 30 files changed, 1431 insertions(+), 18 deletions(-)
 create mode 100644 src/main/java/com/javier/springboot/ejemplito/perfulandia/controllers/UsuarioController.java
 create mode 100644 src/main/java/com/javier/springboot/ejemplito/perfulandia/repositories/AdministradorRepository.java
 create mode 100644 src/main/java/com/javier/springboot/ejemplito/perfulandia/repositories/ClienteRepository.java
 create mode 100644 src/main/java/com/javier/springboot/ejemplito/perfulandia/repositories/EncargadoLogisticaRepository.java
 create mode 100644 src/main/java/com/javier/springboot/ejemplito/perfulandia/repositories/EncargadoVentasRepository.java
 create mode 100644 src/main/java/com/javier/springboot/ejemplito/perfulandia/repositories/GerenteRepository.java
```

Ahora se vincula la carpeta local con el repositorio con el comando `git remote add origin`, se preparan los archivos para ser subidos con `git add .` y se le agrega un commit con `git commit -m`

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio/perfulandia (master)
$ git push -u origin master
Enumerating objects: 83, done.
Counting objects: 100% (83/83), done.
Delta compression using up to 16 threads
Compressing objects: 100% (63/63), done.
Writing objects: 100% (83/83), 19.76 KiB | 1.98 MiB/s, done.
Total 83 (delta 26), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (26/26), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/havier14-2/Seccion-001D-Mayo-27/pull/new/master
remote:
To https://github.com/havier14-2/Seccion-001D-Mayo-27.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Y ejecutamos el comando `git push -u origin master`.

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio
$ git clone https://github.com/havier14-2/Seccion-001D-Mayo-27.git
Cloning into 'Seccion-001D-Mayo-27'...
remote: Enumerating objects: 83, done.
remote: Counting objects: 100% (83/83), done.
remote: Compressing objects: 100% (37/37), done.
remote: Total 83 (delta 26), reused 83 (delta 26), pack-reused 0 (from 0)
Receiving objects: 100% (83/83), 19.76 KiB | 505.00 KiB/s, done.
Resolving deltas: 100% (26/26), done.

javie@GearSixth MINGW64 ~/OneDrive/Escritorio
$ cd Seccion-001D-Mayo-27
```

Ejecutamos comando git clone para clonar el repositorio y subir todas las carpetas al mismo repositorio.

luego cd para acceder al repositorio clonado.

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio/Seccion-001D-Mayo-27 (master)
$ cp -r ~/OneDrive/Escritorio/perfulandia .

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/Seccion-001D-Mayo-27 (master)
$ cp -r ~/OneDrive/Escritorio/productos .

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/Seccion-001D-Mayo-27 (master)
$ cp -r ~/OneDrive/Escritorio/sucursales .

javie@GearSixth MINGW64 ~/OneDrive/Escritorio/Seccion-001D-Mayo-27 (master)
$ git init
Reinitialized existing Git repository in C:/Users/javie/OneDrive/Escritorio/Seccion-001D-Mayo-27/.git/
```

copiamos las carpetas en el clon con el comando cp -r ~.

luego inicializamos el repositorio con el comando git init.

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio/Seccion-001D-Mayo-27 (master)
$ git push -u origin master
Enumerating objects: 63, done.
Counting objects: 100% (63/63), done.
Delta compression using up to 16 threads
Compressing objects: 100% (44/44), done.
Writing objects: 100% (62/62), 9.38 KiB | 961.00 KiB/s, done.
Total 62 (delta 8), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local object.
To https://github.com/havier14-2/Seccion-001D-Mayo-27.git
  313d796..b6ceb96 master -> master
branch 'master' set up to track 'origin/master'.
```

Ejecutamos comando git add .. luego git commit -m para agregar un commit, y finalmente git push -u origin master para subir los cambios.

Conclusión

Gracias a este proyecto pudimos aplicar lo aprendido sobre la arquitectura de microservicios utilizando Spring boot, uno de los mejores frameworks que trabajan las empresas. Separando en 3 grandes módulos el proyecto, se



permitirá el fácil mantenimiento, una eficiente escalabilidad en el sistema y un despliegue independiente de cada módulo.

Gracias a la implementación de APIs RESTful bien definidas logramos cumplir los estándares modernos de desarrollo de software, en este proyecto trabajamos con Git y GitHub las cuales son clave en la actualidad.

Este proyecto presenta una base sólida para futuras ampliaciones.