

Desarrollo de un Back-End utilizando Java y Spring

Juan Manuel Zapata Forero

Desarrollo de aplicaciones web

Especialización en Ingeniería de Software

© Universidad Internacional de La Rioja (UNIR)

Ingeniero Javier Díaz Díaz

08 de diciembre de 2024

## Tabla de Contenido

<b>Introducción:</b> .....	3
<b>Desarrollo:</b> .....	4
<b>Creación de los microservicios</b> .....	4
<b>Base de datos:</b> .....	9
<b>Pruebas de funcionamiento:</b> .....	9
<b>Conclusiones:</b> .....	13

## Tablas de ilustraciones

Ilustración 1. Estructura proyecto “microservice-persona” .....	4
Ilustración 2. Estructura proyecto microservice-registro .....	5
Ilustración 3. Configuración consumo microservicio-persona .....	6
Ilustración 4. Estructura de carpetas microservice-eureka y microservice-gate way.....	7
Ilustración 5. Registro de microservicios en eureka .....	8
Ilustración 6. Microservicios direccionados por el gateway .....	8
Ilustración 7. Bases de datos MySQL y datos de prueba .....	9
Ilustración 8. Consultas desde thunder client. ....	9
Ilustración 9. Consulta por ID.....	10
Ilustración 10. Consulta microservicio persona por documento .....	10
Ilustración 11. Consulta microservicio-persona por apartamento .....	11
Ilustración 12. Consulta todos los registros microservicio-persona .....	11
Ilustración 13. Borrado de registros. ....	11
Ilustración 14. Creación de personas .....	12
Ilustración 15 Creación de apartamentos .....	12
Ilustración 16. Consumo del microservicio-personas desde el microservicio-registro .....	13

## **Introducción:**

En este trabajo se ponen en práctica los conocimientos adquiridos sobre Java y la herramienta Spring Boot, en la implementación de un backend.

Se crea el backend para una aplicación de registro de los habitantes de los apartamentos de un conjunto residencial, dejando abierta la posibilidad de incluir los microservicios para la gestión de mascotas, espacios comunes, empleados de conjunto y la comunicación con un microservicio de control de acceso.

En el siguiente repositorio se cargaron los archivos de la actividad:

<https://github.com/juanchoz24/MicroServicesAct1>

A continuación, se expone el desarrollo y metodología de trabajo para el desarrollo de la actividad requerida.

## Desarrollo:

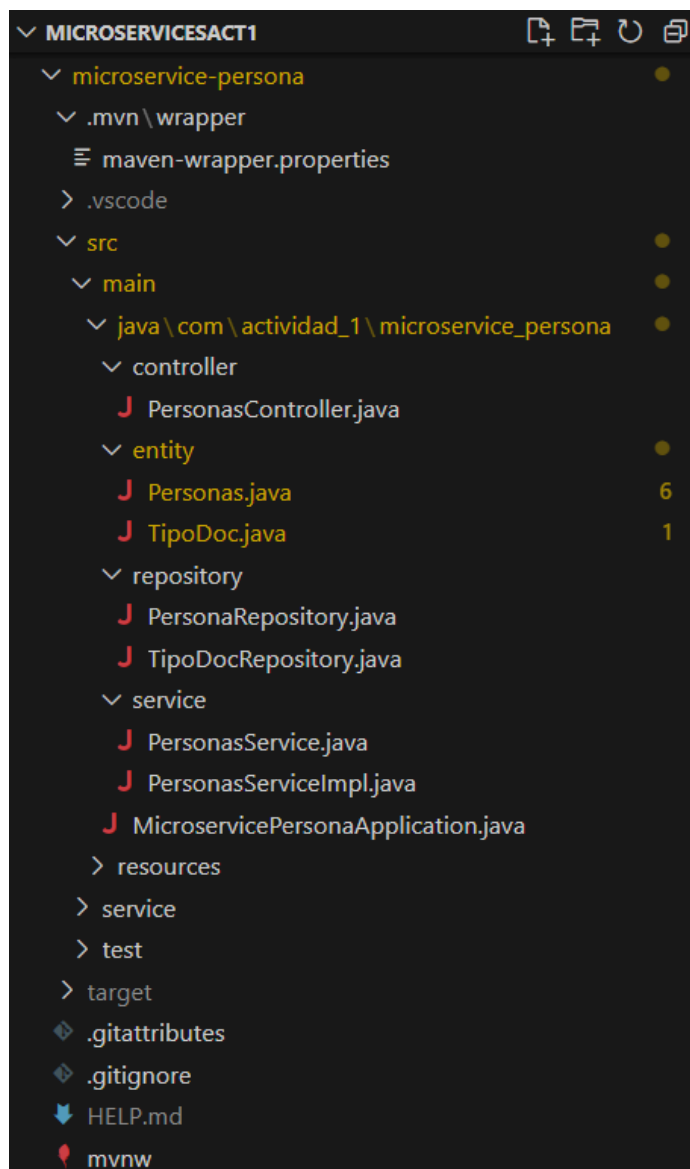
En primer lugar, se realizó el análisis de los requerimientos para el desarrollo de la actividad, mediante la lectura del material con las instrucciones y las pautas para la realización de la actividad.

### Creación de los microservicios

Para el desarrollo de la actividad, inicialmente se creó la carpeta “MICROSERVICESACT1”, en la cual con la herramienta web Spring Initiaizr se dispusieron los microservicios creados:

- microservice-persona.

Ilustración 1. Estructura proyecto “microservice-persona”



Fuente: Elaboración propia.

Se crearon las carpetas controller, entity, repository y service.

En la carpeta controller se definió el controlador rest con los verbos CRUD para el microservicio.

- Método para verbo Create: savePersonas.
- Métodos para verbo Read: findAllPersonas, findById, findByDoc y findByApartamento
- Método para el verbo Update: actualizarPersona
- Método para el verbo Delete: deleteById

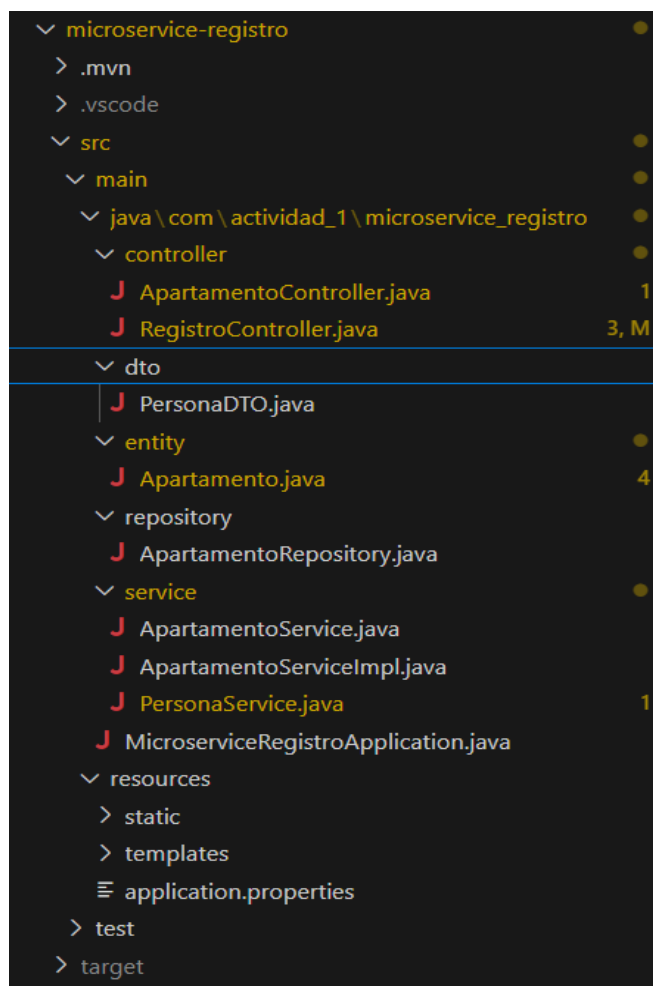
En la carpeta entity se definieron las clases Persona y TipoDoc, con la relación muchos a uno para el tipo de documento, con el objetivo de que esta última sea una tabla paramétrica.

En la carpeta repository se definieron las interfaces para la gestión de las respectivas consultas de las entidades.

En la carpeta service se definen los servicios y la implementación de los mismos.

- Microservice-registro

Ilustración 2. Estructura proyecto microservice-registro



Fuente: Elaboración propia.

De forma análoga al microservicio anterior, se define la estructura de carpetas controller, entity, repository y service, pero se agregó la carpeta dto, la cual alberga el objeto de transporte para la respuesta al consumo del microservicio persona.

Adicionalmente, se crearon clases de servicio y controlador para el consumo del microservicio persona.

*Ilustración 3. Configuración consumo microservicio-persona*

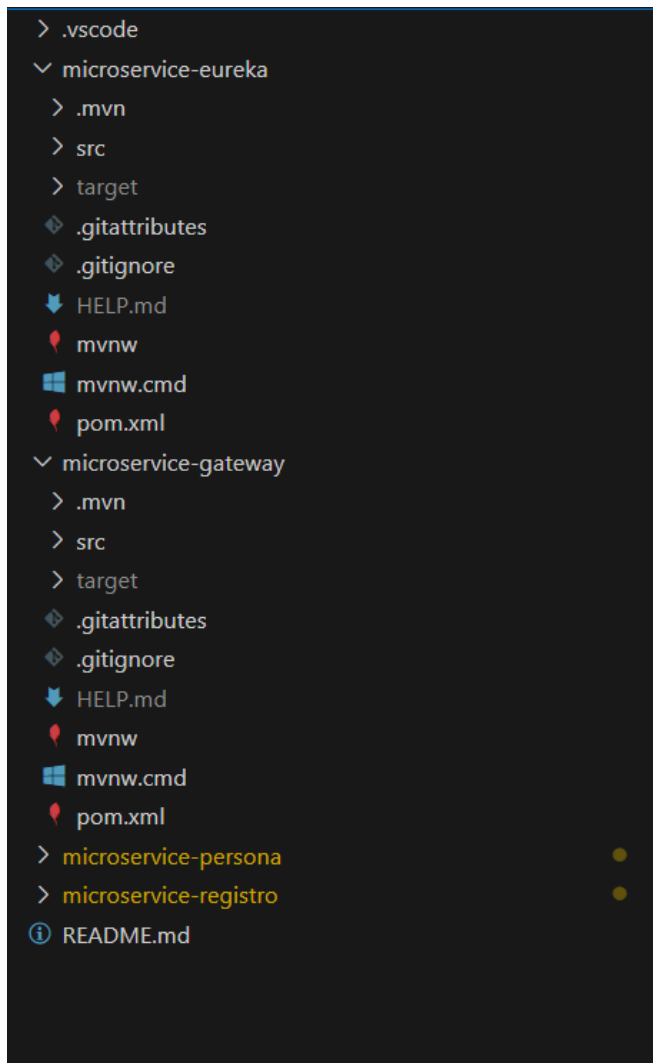
```
microservice-registro > src > main > java > com > actividad_1 > microservice_registro > service > PersonaService.java > PersonaService > getPersonas(String)

6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.http.ResponseEntity;
8  import org.springframework.stereotype.Service;
9  import org.springframework.web.client.RestTemplate;
10 import org.springframework.web.util.UriComponentsBuilder;
11
12 import com.actividad_1.microservice_registro.dto.PersonaDTO;
13
14 @Service
15 public class PersonaService {
16
17     @Autowired
18     RestTemplate restTemplate;
19
20     public List<PersonaDTO> getPersonas(String apartamento){
21         String url = UriComponentsBuilder
22             .fromHttpUrl("http://localhost:8080/api/persona/v1/buscarapto")
23             .queryParams(name:"apartamento", apartamento)
24             .toUriString();
25
26         ResponseEntity<PersonaDTO[]> response = restTemplate.getForEntity(url, responseType:PersonaDTO[].class);
27         PersonaDTO[] productDTO = response.getBody();
28         List<PersonaDTO> m = Arrays.asList(productDTO);
29
30         return m;
31     }
32 }
33
```

Fuente: Elaboración propia.

- Microservice eureka y microservice gateway

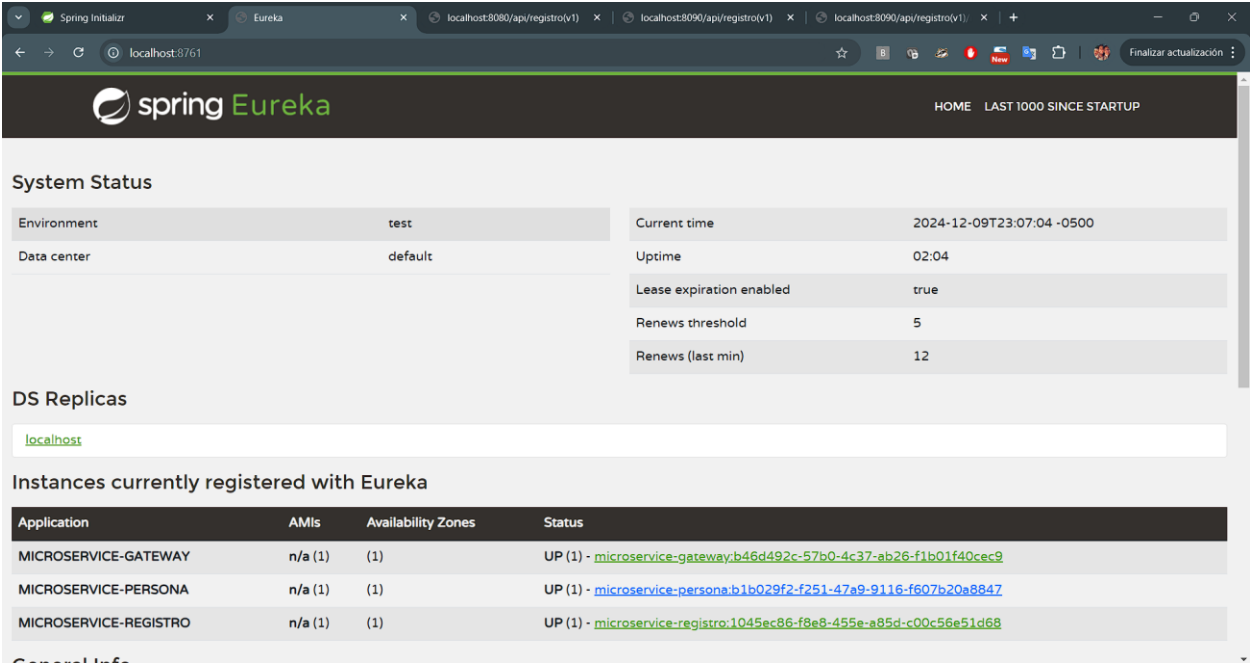
Ilustración 4. Estructura de carpetas microservice-eureka y microservice-gate way



Fuente: Elaboración propia.

Una vez se arranca el microservicio-eureka y posteriormente los demás, se evidencia como se registran los microservicios.

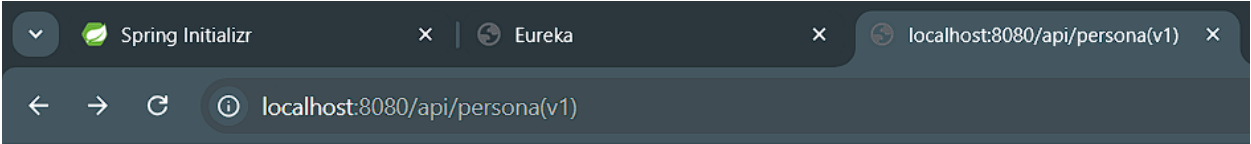
Ilustración 5. Registro de microservicios en eureka



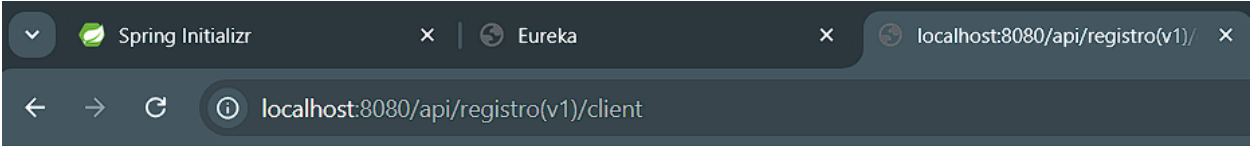
Fuente: Elaboración propia.

En cada uno de los microservicio se configuro el archivo application.properties, con la configuración de conexión a la base de datos de los dos microservicios principales, así como los clientes y servidor para eureka y el Gateway.

Ilustración 6. Microservicios direccionados por el gateway



Personas



Registro

Fuente: Elaboración propia.



## Base de datos:

Una vez se ejecutan los dos microservicios principales, se crean las dos bases de datos en MySQL y se insertan datos de prueba:

Ilustración 7. Bases de datos MySQL y datos de prueba

Navigator

SCHEMAS

Filter objects

desarrollo2024

registropersonas

Tables

personas

tipos\_doc

Views

Stored Procedures

Functions

registroresidentes

Tables

apartamentos

Views

Stored Procedures

Functions

sys

Administration

Schemas

Information

Query 1

personas

personas

apartamentos - Table

apartamentos

personas

Limit to 1000 rows

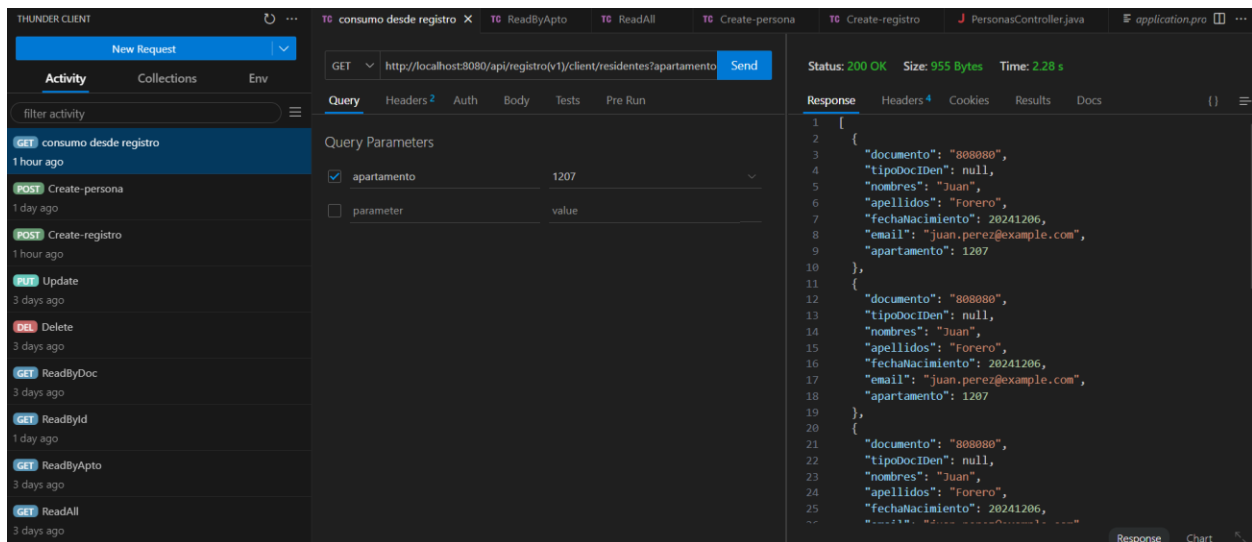
</

Fuente: Elaboración propia.

## Pruebas de funcionamiento:

Una vez se crearon y configuraron los cuatro microservicios, con el uso de thunderclient se elaboraron las consultas get, post, put y delete para los dos microservicios.

Ilustración 8. Consultas desde thunder client.



The screenshot shows the Thunder Client interface. The 'Activity' pane on the left lists several requests. The main area displays a GET request to `http://localhost:8080/api/registro(v1)/client/residentes/apartamento` with a query parameter `apartamento=1207`. The response is a JSON object with the following structure:

```
{
  "documento": "808080",
  "tipoDocId": null,
  "nombres": "Juan",
  "apellidos": "Forero",
  "fechaNacimiento": 20241206,
  "email": "Juan.perez@example.com",
  "apartamento": 1207
}
```

Fuente: Elaboración propia.

Se realizaron test del funcionamiento de los verbos CRUD para los dos microservicios

*Ilustración 9. Consulta por ID*

The screenshot displays two REST client test results. The top test is for 'ApartamentoController.java' with a GET request to 'http://localhost:8080/api/registro(v1)/buscarid/{id}'. The response is a JSON object with status 200 OK, size 81 Bytes, and time 75 ms. The JSON body contains: { "id": "1", "torre": "1", "numApartamento": "1207", "deposito": "402", "parqueadero": 250 }. The bottom test is for 'PersonasController.java' with a GET request to 'http://localhost:8080/api/persona(v1)/buscarid/{id}'. The response is a JSON object with status 200 OK, size 154 Bytes, and time 2.46 s. The JSON body contains: { "id": 3, "documento": null, "tipoDocIDen": null, "nombres": "Juan", "apellidos": "Pérez", "fechaNacimiento": 0, "email": "juan.perez@example.com", "apartamento": null }.

Fuente: Elaboración propia.

*Ilustración 10. Consulta microservicio persona por documento*

The screenshot displays a REST client test result for 'PersonasController.java' with a GET request to 'http://localhost:8080/api/persona(v1)/buscar/{documento}'. The response is a JSON object with status 200 OK, size 150 Bytes, and time 335 ms. The JSON body contains: { "id": 6, "documento": "123", "tipoDocIDen": "CC", "nombres": "Juan", "apellidos": "Zapata", "fechaNacimiento": 2024126, "email": "123@123.com", "apartamento": null }.

Fuente: Elaboración propia.

Ilustración 11. Consulta microservicio-persona por apartamento

TC ReadById TC consumo desde registro TC ReadByApto X TC ReadAll TC Create-persona TC Create-registro J PersonasController.java

GET http://localhost:8080/api/persona(v1)/buscarapto?apartamento=1207 Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

apartamento	1207
parameter	value

Status: 200 OK Size: 1003 Bytes Time: 28 ms

Response Headers 5 Cookies Results Docs

```
2 {
3   "id": 27,
4   "documento": "808080",
5   "tipoDocIDen": null,
6   "nombres": "Juan",
7   "apellidos": "Forero",
8   "fechaNacimiento": 20241206,
9   "email": "juan.perez@example.com",
10  "apartamento": 1207
11 },
12 {
13   "id": 28,
14   "documento": "808080",
15   "tipoDocIDen": null,
16   "nombres": "Juan",
17   "apellidos": "Forero",
18   "fechaNacimiento": 20241206,
19   "email": "juan.perez@example.com",
20   "apartamento": 1207
21 },
```

Fuente: Elaboración propia.

Ilustración 12. Consulta todos los registros microservicio-persona

TC ReadById TC consumo desde registro TC ReadByApto TC ReadByDoc TC ReadAll X TC Create-persona TC Create-registro

GET http://localhost:8080/api/persona(v1)/buscartodos Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter	value
-----------	-------

Status: 200 OK Size: 4.38 KB Time: 59 ms

Response Headers 5 Cookies Results Docs

```
1 [
2   {
3     "id": 1,
4     "documento": null,
5     "tipoDocIDen": null,
6     "nombres": "Juan",
7     "apellidos": "Pérez",
8     "fechaNacimiento": 0,
9     "email": "juan.perez@example.com",
10    "apartamento": null
11  },
12  {
13    "id": 2,
14    "documento": null,
15    "tipoDocIDen": null,
16    "nombres": "Juan",
17    "apellidos": "Pérez",
18    "fechaNacimiento": 0,
19    "email": "juan.perez@example.com",
20    "apartamento": null
21  },
22 ]
```

Fuente: Elaboración propia.

Ilustración 13. Borrado de registros.

TC ReadByDoc TC ReadAll TC Delete X J PersonasServiceImpl.java TC Create-persona TC Create-registro J PersonasController.java

DELETE http://localhost:8080/api/persona(v1)/borrar/{id} Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

id	123
parameter	value

Status: 204 No Content Size: 0 Bytes Time: 93 ms

Response Headers 3 Cookies Results Docs

```
1
```

Fuente: Elaboración propia.

Ilustración 14. Creación de personas

The screenshot shows a REST client interface with a tab for 'TC Create-persona'. The request is a POST to 'http://localhost:8080/api/persona(v1)/save'. The response is a 200 OK status with a size of 171 Bytes and a time of 708 ms. The response body is a JSON object representing a person.

```
POST http://localhost:8080/api/persona(v1)/save
```

```
{
  "documento": "11111",
  "tipoDocIden": "CC",
  "nombres": "Angelina",
  "apellidos": "Jolie",
  "fechaNacimiento": 20001206,
  "email": "angelina.perez@example.com",
  "torre": 4,
  "apartamento": 107
}
```

Fuente: Elaboración propia.

Ilustración 15 Creación de apartamentos

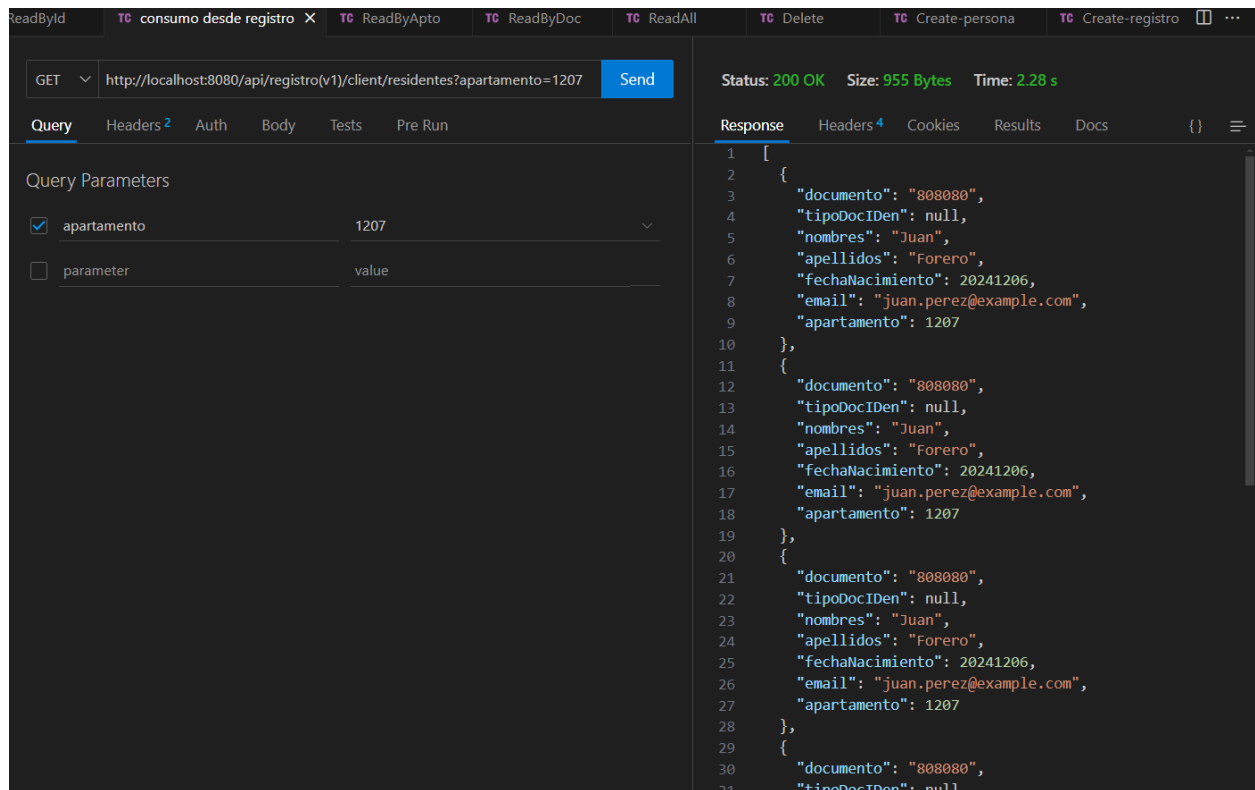
The screenshot shows a REST client interface with a tab for 'TC Create-registro'. The request is a POST to 'http://localhost:8080/api/registro(v1)/save'. The response is a 200 OK status with a size of 81 Bytes and a time of 193 ms. The response body is a JSON object representing an apartment.

```
POST http://localhost:8080/api/registro(v1)/save
```

```
{
  "torre": "1",
  "numApartamento": "1207",
  "deposito": "402",
  "parqueadero": 250
}
```

Fuente: Elaboración propia.

Ilustración 16. Consumo del microservicio-personas desde el microservicio-registro



Fuente: Elaboración propia.

## Conclusiones:

- La implementación de microservicios independientes permite el escalamiento de la aplicación, creando nuevos microservicios que se conecten y aumenten las prestaciones y servicios inicialmente especificados.
- El uso de del microservicio Gateway, permite desde una misma url, recibir las distintas peticiones y direccionarlas al microservicio correspondiente, facilitando la gestión de dominios.
- Es posible obtener desde un microservicio, datos de la base de datos de otro, para almacenarlos y/o procesarlos en la propia base de datos del otro microservicio, que incluso puede estar en otro motor y/o tipo de base de datos.