Laboratorio 04 ARSW: REST API Blueprints

María Paula Rodríguez Muñoz
Juan Andrés Suárez
Juan Pablo Nieto
Tomás Felipe Ramírez

1. Introducción

En este laboratorio se implementó una API REST para la gestión de Blueprints utilizando Java 21 y Spring Boot 3.3.x. La aplicación permite crear, consultar y actualizar blueprints, almacenando la información en una base de datos PostgreSQL y documentando los endpoints mediante Swagger/OpenAPI.

2. Instrucciones para ejecutar el proyecto

2.1. 1. Levantar la base de datos PostgreSQL

Se utilizó PostgreSQL en Docker con el siguiente comando:

```
docker run -d --name postgres-blueprints \
-e POSTGRES_DB=blueprints \
-e POSTGRES_USER=postgres \
-e POSTGRES_PASSWORD=postgres \
-p 5432:5432 postgres
```

2.2. 2. Compilar y ejecutar la aplicación

Desde el directorio del proyecto:

```
mvn clean install
mvn spring-boot:run
```

2.3. 3. Acceder a Swagger UI

Abrir en el navegador:

```
http://localhost:8080/swagger-ui/index.html
```

Desde allí se pueden probar todos los endpoints de la API.

3. Evidencia de consultas en Swagger UI

3.1. Creación de Blueprint (POST)

Se realizó una petición POST al endpoint:

```
/api/v1/blueprints
```

Obteniendo como respuesta el código HTTP 201 Created.

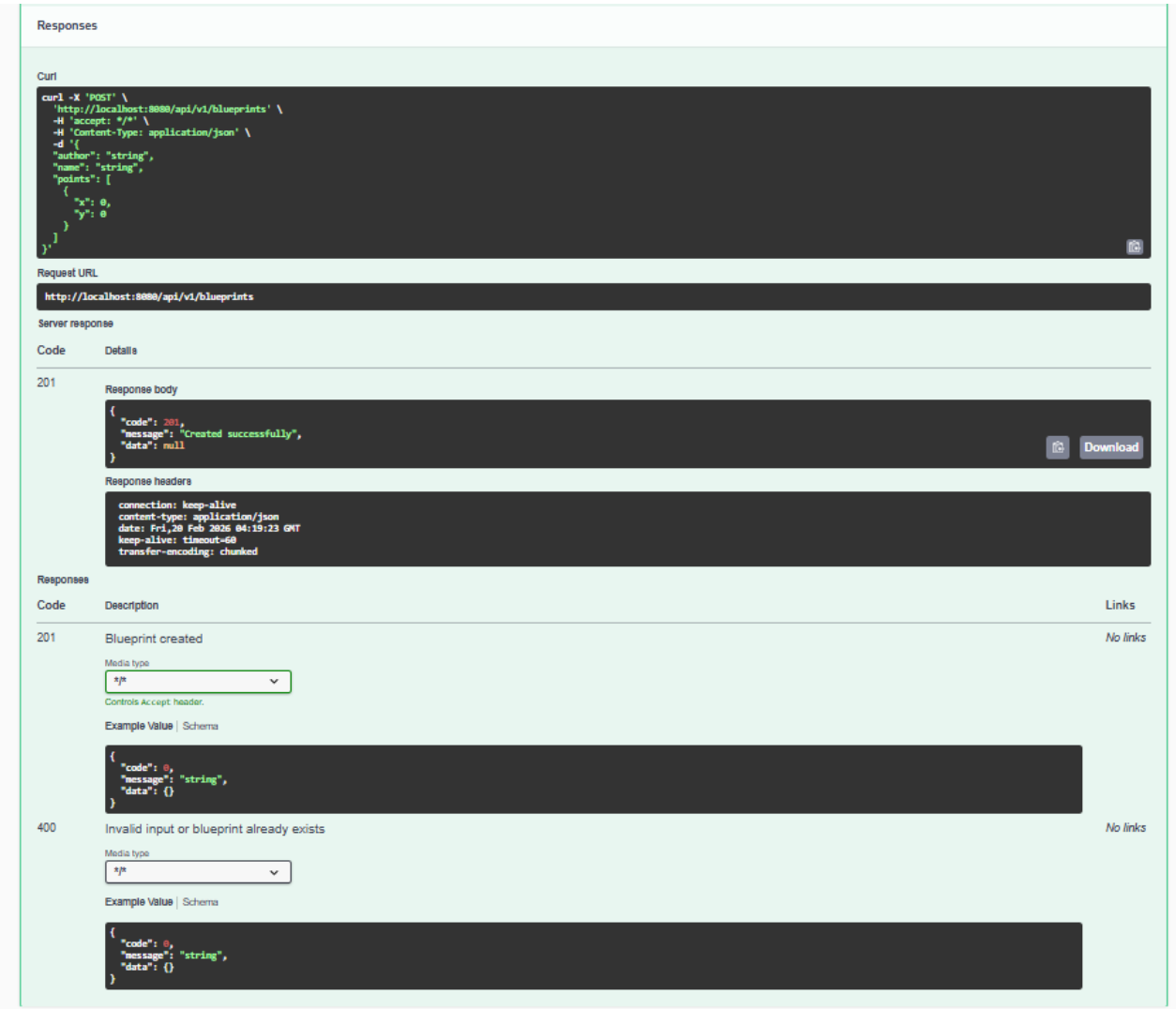

Figura 1: Evidencia creación de blueprint en Swagger

3.2. Consulta de Blueprints por autor (GET)

Se realizó una petición GET al endpoint:

`/api/v1/blueprints/{author}`

Obteniendo como respuesta el código HTTP 200 OK.

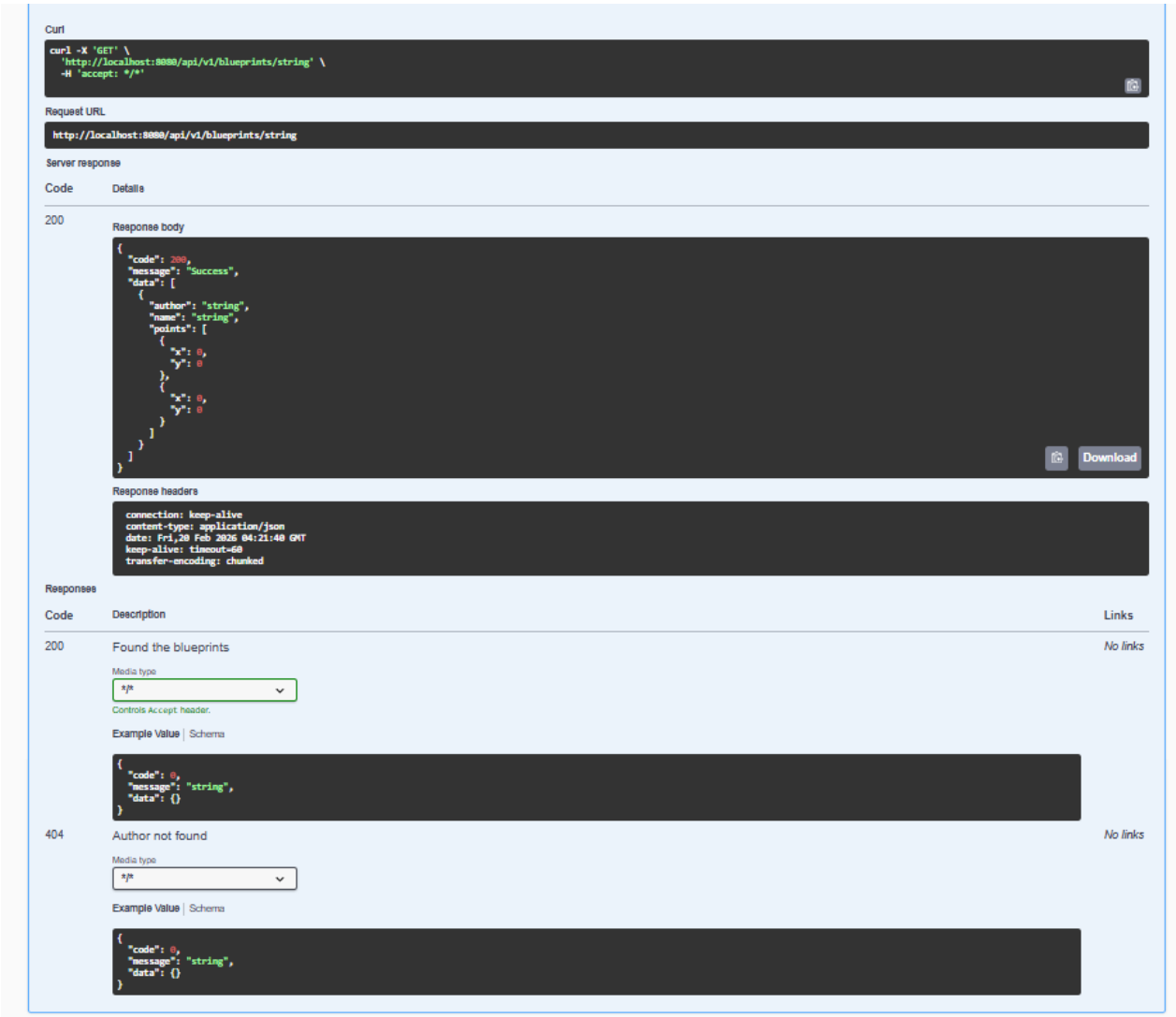

Figura 2: Evidencia consulta por autor en Swagger

4. Evidencia de persistencia en base de datos

Luego de realizar las peticiones desde Swagger, se verificó directamente en PostgreSQL que la información fue almacenada correctamente.

4.1. Tabla blueprint

```
PS D:\arsw\lab 4\REST-API-Blueprints> docker exec -it postgres-blueprints psql -U postgres -d blueprints
psql (16.12)
Type "help" for help.

blueprints=# SELECT * FROM blueprints;
  author  |  name
----------+---------
 john     | kitchen
 testuser | testbp
 string   | string
 Juan     | string
(4 rows)

blueprints=#
```

Figura 3: Registros almacenados en tabla blueprint

5. Verificación y Calidad del Proyecto

5.1. Análisis de calidad con SonarQube

Se ejecutó análisis estático de código mediante SonarQube utilizando el siguiente comando:

```
mvn clean verify sonar:sonar \
-Dsonar.projectKey=arsw-2024-1-g6-2_to-do-code-REST-API-Blueprints-
    lab \
-Dsonar.login=tokent
```

El análisis permitió evaluar métricas de calidad y cobertura de pruebas del proyecto.

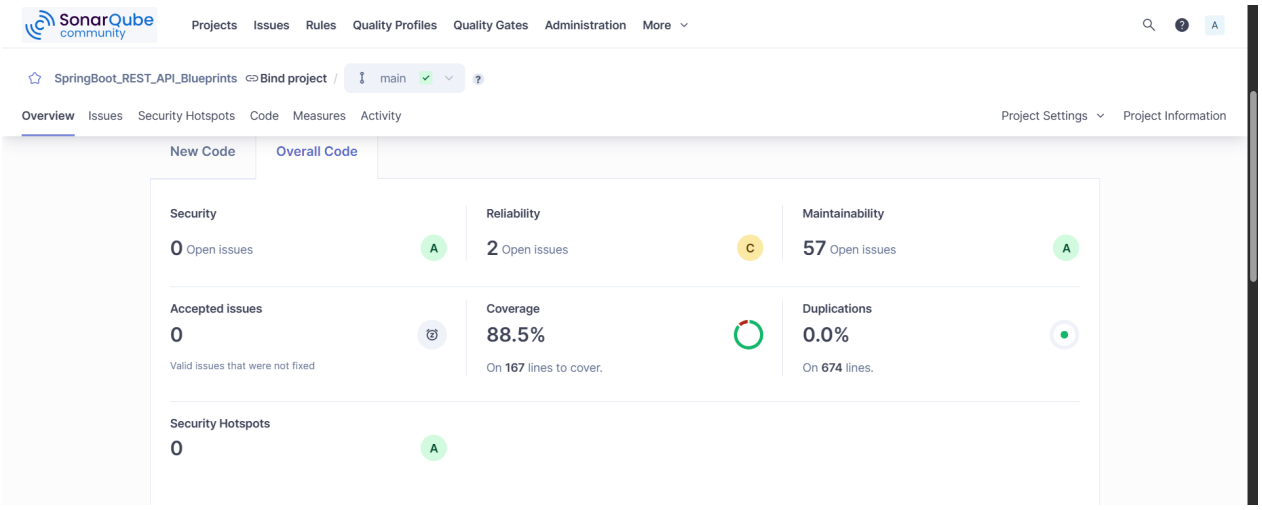

Figura 4: Métricas de calidad y cobertura en SonarQube

5.2. Cobertura de pruebas con JaCoCo

Se utilizó JaCoCo para generar el reporte de cobertura de pruebas del proyecto mediante el siguiente comando:

```
mvn clean test jacoco:report
```

El reporte generado permite visualizar el porcentaje de cobertura por clases y métodos.

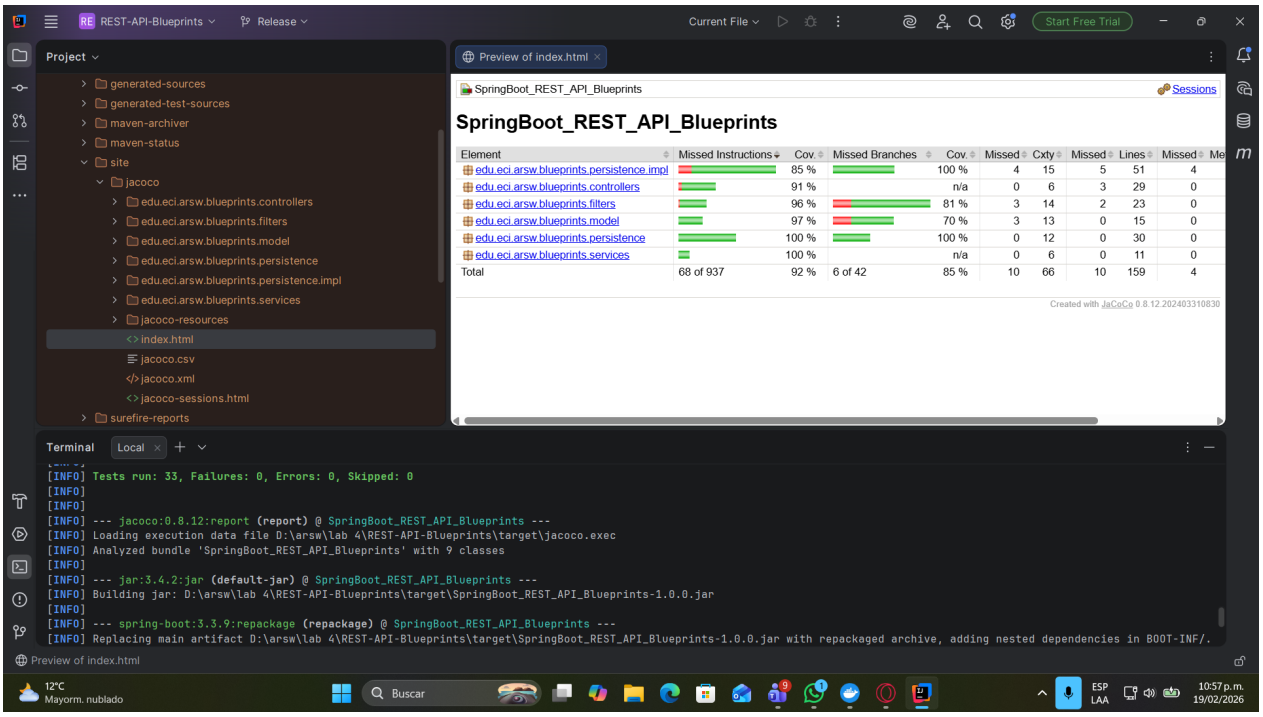

Figura 5: Reporte de cobertura generado por JaCoCo

6. Diagramas

6.1. Diagrama de Componentes

El siguiente diagrama muestra la arquitectura por capas implementada en la API, evidenciando la separación entre controlador, servicio, persistencia y base de datos.

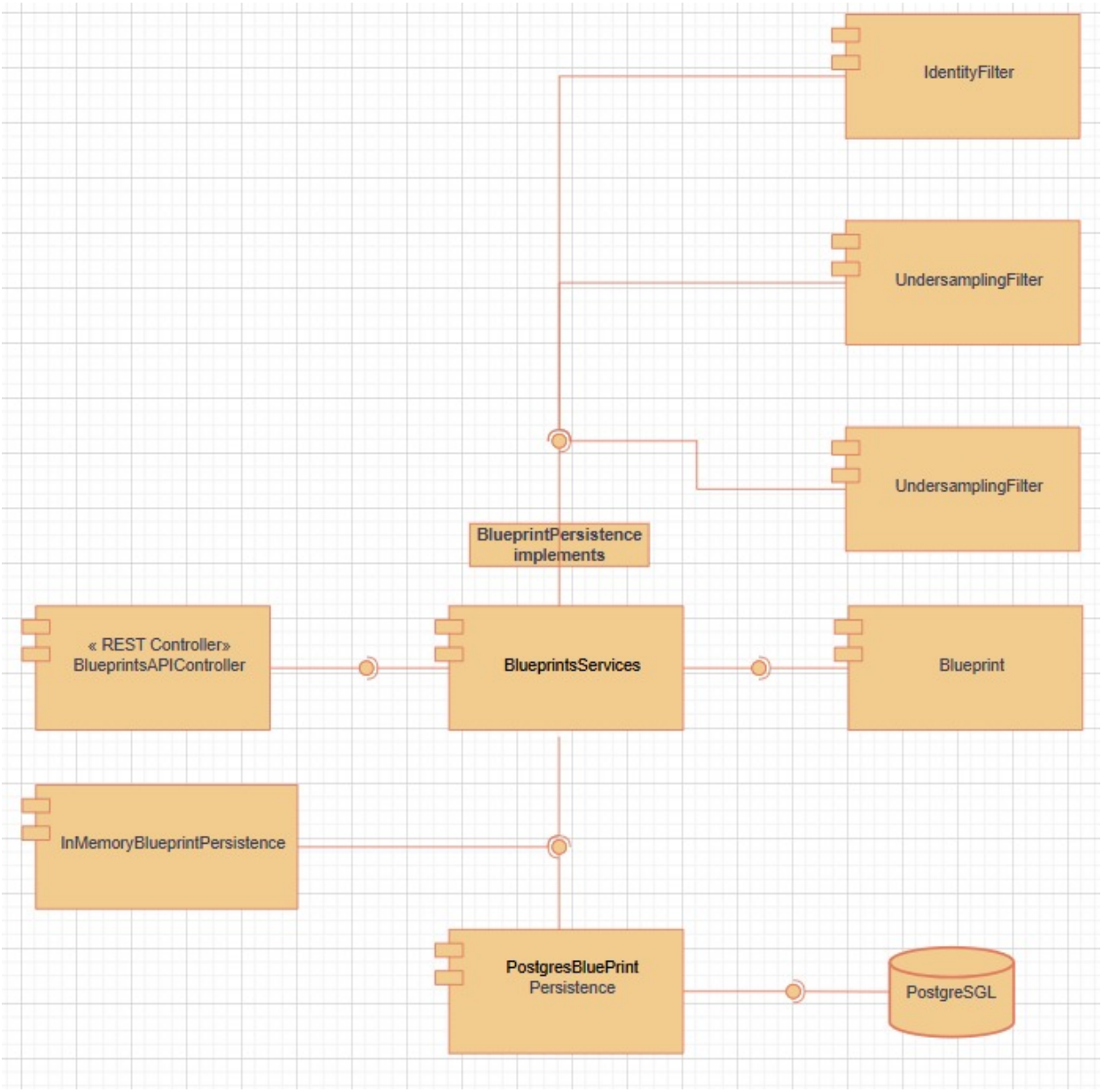

Figura 6: Diagrama de componentes de la API Blueprints

6.2. Diagrama de Flujo

El siguiente diagrama representa el flujo general de una petición HTTP desde el cliente hasta la persistencia en base de datos y la generación de la respuesta.

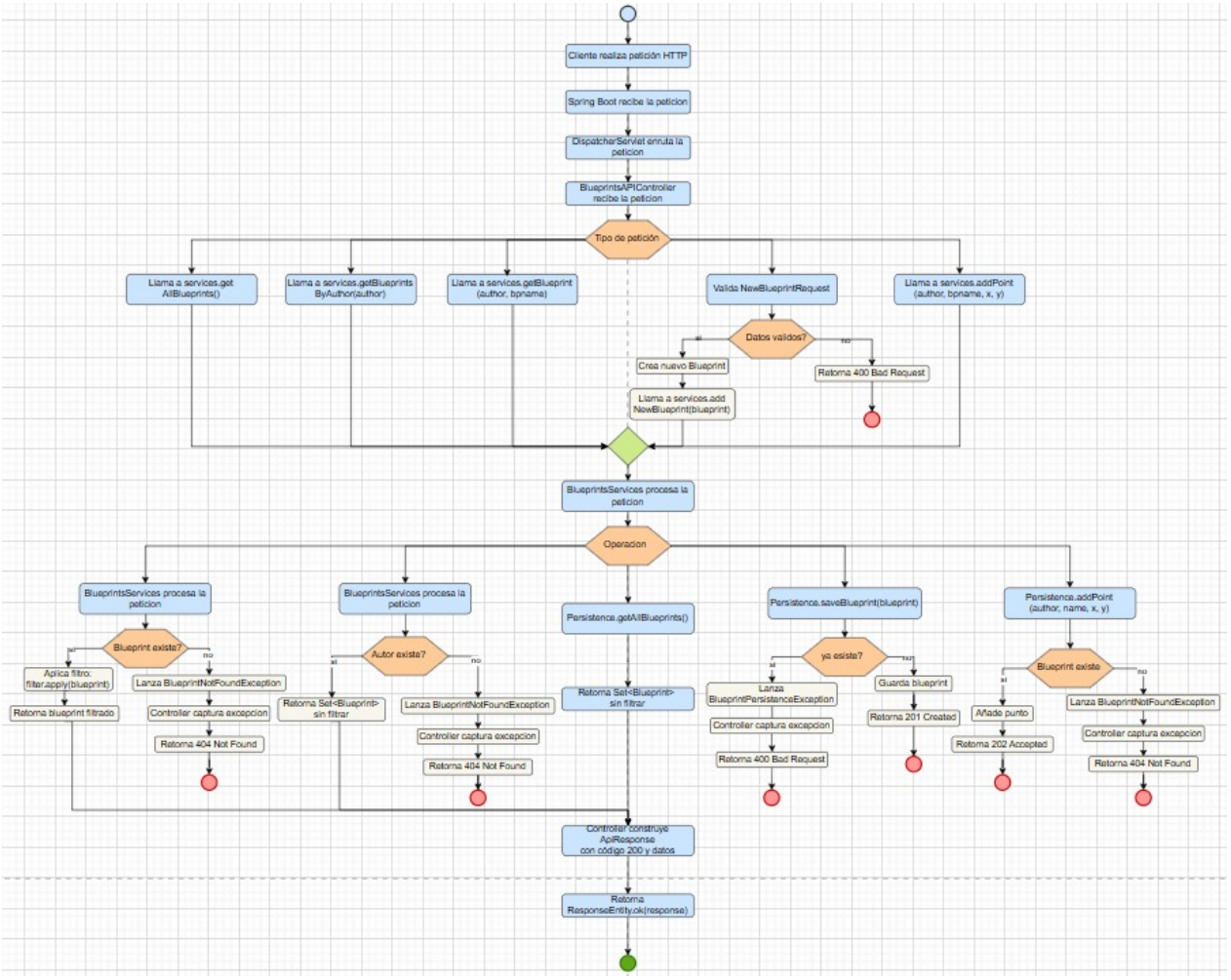

Figura 7: Flujo de procesamiento de una petición en la API

7. Buenas prácticas aplicadas

Durante el desarrollo del laboratorio se aplicaron las siguientes buenas prácticas:

- Versionamiento de API mediante el prefijo `/api/v1`.
- Uso correcto de códigos HTTP (200, 201, 400, 404).
- Implementación de una clase `ApiResponse` para estandarizar respuestas.
- Manejo global de excepciones con `@RestControllerAdvice`.
- Separación por capas (controlador, servicio, persistencia).

8. Conclusiones

- Se implementó una API REST siguiendo una arquitectura por capas, garantizando separación de responsabilidades y mejor mantenibilidad.
- La integración con PostgreSQL mediante Docker permitió un entorno de ejecución estable y reproducible.
- Las pruebas realizadas en Swagger UI confirmaron el correcto funcionamiento de los endpoints.
- Las consultas directas a la base de datos evidenciaron la correcta persistencia de la información.
- El análisis con SonarQube y la cobertura generada con JaCoCo permitieron evaluar la calidad y robustez del código.
- La documentación mediante diagramas facilita la comprensión de la arquitectura y el flujo de procesamiento.