

INFORME DE LABORATORIO

REST API Blueprints

Presentado por:

Miguel Ángel Monroy Cárdenas

Ana Gabriela Fiquitiva Poveda

Ingeniería de Sistemas

Universidad Escuela Colombiana de Ingeniería

Profesor:

Diego Andrés Triviño González

Febrero 2026

Materia: Arquitecturas de Software (ARSW)

Tema: API REST

1. Introducción

Este informe documenta la evolución de la aplicación BlueprintsRESTAPI. Se ha pasado de una implementación básica en memoria a una API que utiliza persistencia relacional con PostgreSQL, garantizando la integridad de los planos y sus puntos coordenados a través de un diseño basado en capas.

2. Arquitectura y Diseño del Sistema

2.1. Capas del Software

Se mantiene la separación de intereses para facilitar el mantenimiento:

- Controladores: Manejan el protocolo HTTP y transforman las peticiones en llamadas al servicio.
- Servicios: Orquestan la lógica de negocio y aplican los filtros de planos.
- Persistencia (JPA/Hibernate): Gestiona el almacenamiento de las entidades Blueprint y Point en la base de datos PostgreSQL.

2.2. Estandarización de Respuestas (ApiResponse<T>)

Se implementó una clase genérica para uniformar la comunicación con el cliente:

```
public class ApiResponse<T> {  
  
    private String message;  
  
    private T data;  
  
    private int status;  
  
}
```

Esto permite que cualquier respuesta (éxito o error) tenga una estructura predecible para el front-end.

3. Implementación de Base de Datos

Se migró el almacenamiento volátil a PostgreSQL. La configuración en el archivo application.yml define la conexión y el comportamiento de Hibernate.

Script SQL / DDL

La base de datos consta de dos tablas principales:

1. blueprints: Almacena el nombre del plano y el autor.
 2. points: Almacena las coordenadas (x, y) asociadas a un ID de plano mediante una llave foránea (Relación uno a muchos).
-

4. Documentación con Swagger/OpenAPI

Se habilitó la interfaz interactiva para pruebas de la API.

- Ruta de acceso: <http://localhost:8080/swagger-ui/index.html>
- Funcionalidad: Permite realizar peticiones GET, POST y PUT, visualizando los esquemas de datos en tiempo real.

5. Buenas Prácticas Aplicadas

1. Inyección de Dependencias: Uso de `@Autowired` y `@Service` para desacoplar componentes.
2. Manejo de Excepciones: Implementación de un `GlobalExceptionHandler` que captura errores de "Recurso no encontrado" y retorna un código 404 estructurado.
3. Código Limpio (Clean Code): Nombres de variables significativos y métodos de una sola responsabilidad en los servicios.
4. RESTful: Uso de Verbos HTTP y sustantivos en Plural siguiendo las buenas practicas para el uso de REST

6. Instrucciones de Ejecución

1. Requisitos: Tener instalado Java 21 y una instancia de PostgreSQL corriendo.
2. Configuración: Ajustar las credenciales en `src/main/resources/application.yml`.
3. Compilación: Ejecutar `mvn clean install`.
4. Ejecución: Iniciar con `mvn spring-boot:run` o ejecutando el JAR generado.

7. Evidencias y Resultados

7.1. Consultas en Swagger UI

(Aquí debes insertar la captura de pantalla de la interfaz de Swagger mostrando el listado de endpoints).

7.2. Verificación en Base de Datos

(Aquí debes insertar la captura de pantalla de tu cliente SQL —DBeaver, pgAdmin— mostrando los datos en las tablas blueprints y points).

Conclusión: La API cumple con los requisitos de robustez, documentación y persistencia exigidos, proporcionando una base sólida para futuras expansiones del sistema de gestión de planos.