

Laboratorio Componentes y Conectores Parte 1

Sebastián Cardona, Laura Gil, Zayra Gutiérrez

**Ingeniero de Sistemas
Javier Toquica Barrera**

Universidad Escuela Colombiana de ingeniería Julio Garavito

Contenido

Introducción	3
Desarrollo del Laboratorio.....	4
Parte 1	4
Componentes y conectores	9
Pruebas con POSTMAN.....	19
Conclusiones	26

Introducción

El presente documento describe la implementación del laboratorio de Componentes y Conectores, cuyo objetivo principal es aplicar principios de inyección de dependencias y uso del framework Spring para la gestión y manipulación de planos arquitectónicos. En este laboratorio, se aborda el desarrollo de un sistema para gestionar planos arquitectónicos de una prestigiosa compañía de diseño, utilizando una arquitectura basada en componentes y conectores.

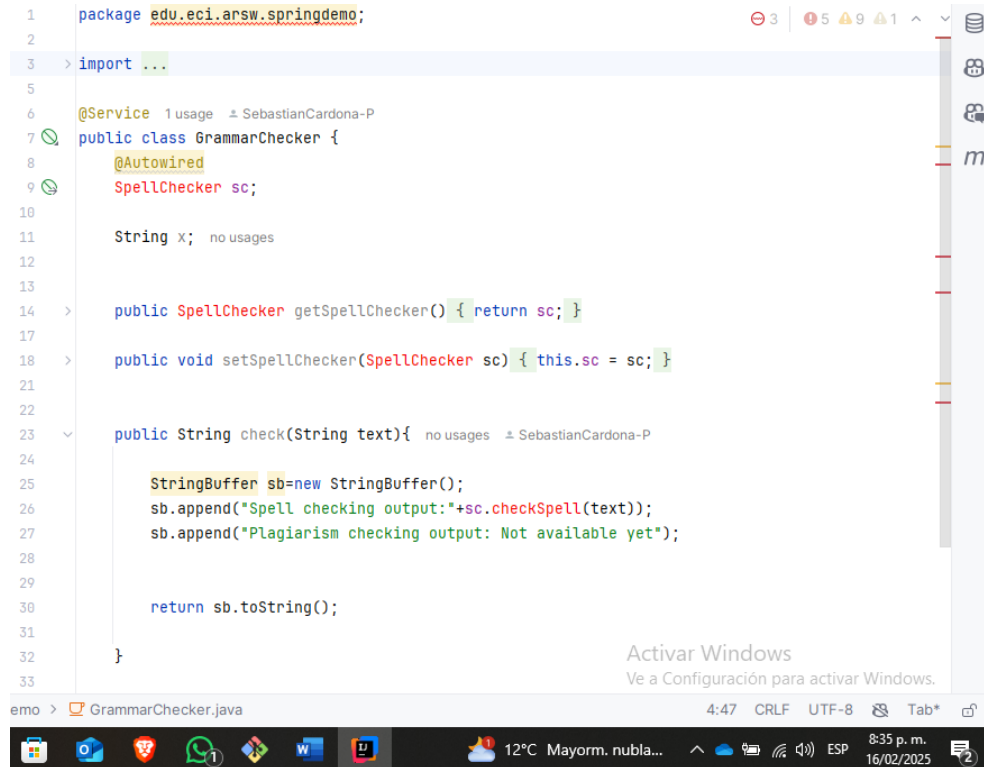
La meta principal es diseñar y construir la capa lógica de la aplicación, aprovechando el framework Spring para facilitar la inyección de dependencias y la configuración a través de anotaciones. Además, se implementa un esquema de persistencia en memoria que permite el almacenamiento y consulta de planos. Como parte del desarrollo, se crean dos tipos de filtros para optimizar los datos almacenados: el filtrado de redundancias, que elimina puntos consecutivos duplicados, y el filtrado de submuestreo, que reduce el número de puntos intercaladamente. Estos filtros se aplican durante las operaciones de consulta, permitiendo reducir el tamaño de los planos arquitectónicos y mejorar la eficiencia del sistema.

Toda la implementación del ejercicio se encuentra disponible en el siguiente repositorio de GitHub: [ARSW-Lab04](#).

Desarrollo del Laboratorio

Parte 1

1. Abra las fuentes del proyecto



```
1 package edu.eci.arsw.springdemo;
2
3 > import ...
4
5
6 @Service 1 usage SebastianCardona-P
7 public class GrammarChecker {
8     @Autowired
9     SpellChecker sc;
10
11     String x; no usages
12
13
14 > public SpellChecker getSpellChecker() { return sc; }
15
16
17 > public void setSpellChecker(SpellChecker sc) { this.sc = sc; }
18
19
20
21
22
23 > public String check(String text){ no usages SebastianCardona-P
24
25     StringBuffer sb=new StringBuffer();
26     sb.append("Spell checking output:"+sc.checkSpell(text));
27     sb.append("Plagiarism checking output: Not available yet");
28
29
30     return sb.toString();
31
32 }
33
```

emo > GrammarChecker.java 4:47 CRLF UTF-8 Tab* 8:35 p. m. 16/02/2025

El proyecto consta de un revisor de ortografía, cuya implementación puede ser en inglés o en español

2. Revise el archivo de configuración de Spring ya incluido en el proyecto (src/main/resources). El mismo indica que Spring buscará automáticamente los 'Beans' disponibles en el paquete indicado.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"

       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework
                           http://www.springframework.org/schema/context http://www.springframework.org/schema/con
">

    <context:component-scan base-package="edu.eci.arsw" />

</beans>
```

Activar Windows
Ve a Configuración para activar Windows.

1:1 CRLF UTF-8 4 spaces

12°C Mayorm. nubla... 8:34 p. m. 16/02/2025

3. Haciendo uso de la configuración de Spring basada en anotaciones marque con las anotaciones `@Autowired` y `@Service` las dependencias que deben inyectarse, y los 'beans' candidatos a ser inyectados -respectivamente-:
Esto se logra colocando la anotación de `@Autowired` en la interfaz a inyectar, de esta manera se indica a el framework que se debe inyectar la dependencia de `SpellChecker`

```
@Service 1 usage SebastianCardona-P
public class GrammarChecker {
    @Autowired
    SpellChecker sc;

    String x; no usages

    public SpellChecker getSpellChecker() { return sc; }

    public void setSpellChecker(SpellChecker sc) { this.sc = sc; }

    public String check(String text){ no usages SebastianCardona-P
        StringBuffer sb=new StringBuffer();
        sb.append("Spell checking output:"+sc.checkSpell(text));
        sb.append("Plagiarism checking output: Not available yet");

        return sb.toString();
    }
}
```

Activar Windows
Ve a Configuración para activar Windows.

GrammarChecker.java 4:47 CRLF UTF-8 Tab*

12°C Mayorm. nubla... 8:39 p. m. 16/02/2025

Seguido de esto, se pone la notación de @Service en la implementación en específico que queramos inyectar, ya sea en SpanishSpellChecker o EnglishSpellChecker, esto se hace para indicarle a el framework cual de las dos implementaciones de inyectará

```
@Service no usages SebastianCardona-P
public class SpanishSpellChecker implements SpellChecker {

    @Override no usages SebastianCardona-P
    public String checkSpell(String text) {
        return "revisando (" + text + ") con el verificador de sintaxis del español";
    }
}
```

Activar Windows
Ve a Configuración para activar Windows.

SpanishSpellChecker.java 1:1 CRLF UTF-8 Tab*

Lluvia para mañana 8:41 p. m. 16/02/2025

4. Haga un programa de prueba, donde se cree una instancia de GrammarChecker mediante Spring, y se haga uso de la misma:

```
> /.../
package edu.eci.arsw.springdemo.ui;

> import ...

/**
 *
 * @author hcadavid
 */
public class Main { no usages SebastianCardona-P

    public static void main(String a[]) { no usages SebastianCardona-P
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");
        GrammarChecker gc = ac.getBean(GrammarChecker.class);
        System.out.println(gc.check("la la la "));
    }
}

Activar Windows
Ve a Configuración para activar Windows.
```

1:1 CRLF UTF-8 4 spaces

5. Modifique la configuración con anotaciones para que el Bean 'GrammarChecker' ahora haga uso de la clase SpanishSpellChecker (para que a GrammarChecker se le inyecte EnglishSpellChecker en lugar de SpanishSpellChecker. Verifique el nuevo resultado.

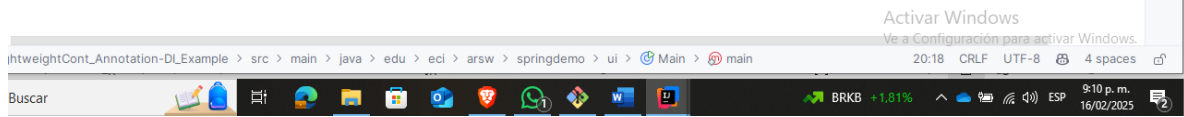
Para esto simplemente se pone la notación de @Service en la implementación que queramos inyectar

Ejecutando el programa con la inyección del chequeador en español

```
"C:\Program Files\Java\jdk-22.0.2\bin\java.exe" ...
Spell checking output:revisando (la la la ) con el verificador de sintaxis del españolPlagiarism checking output: Not available yet
Feb 16, 2025 9:07:21 P. M. org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
ing_LightweightCont_Annotation-DLExample > src > main > java > edu > eci > arsw > springdemo > ui > Main > main
20:18 CRLF UTF-8 4 spaces
```

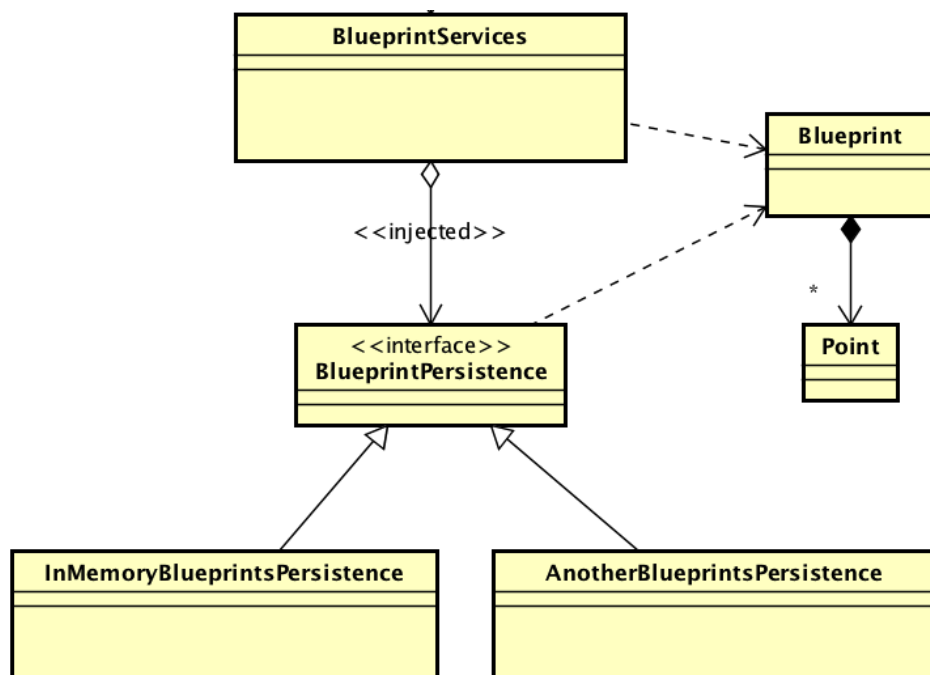
Ejecutando el programa con la inyección del chequeador en inglés

Spell checking output:Checked with english checker:la la la Plagiarism checking output: Not available yet
Process finished with exit code 0



Componentes y conectores

En este ejercicio se va a construir un modelo de clases para la capa lógica de una aplicación que permita gestionar planos arquitectónicos de una prestigiosa compañía de diseño.



1. Configure la aplicación para que funcione bajo un esquema de inyección de dependencias, tal como se muestra en el diagrama anterior.

Lo anterior requiere:

- Agregar las dependencias de Spring.

```

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>3.3.4</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies> Edit Starters...
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>

  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>

```

- Agregar la configuración de Spring.
- Configurar la aplicación -mediante anotaciones- para que el esquema de persistencia sea inyectado al momento de ser creado el bean 'BlueprintServices'.

```

/**
 * Servicio para la gestión de Blueprints.
 * Proporciona métodos para agregar, recuperar y filtrar planos.
 */
@Service 2 usages ± ZayraGS1403 +1
public class BlueprintsServices {
    |
    @Autowired
    BlueprintsPersistence bpp;

    /**
     * Implementación en memoria de la persistencia de planos.
     * Almacena los planos en un mapa en memoria.
     */
    @Service ± LaaSofiaa +1
    public class InMemoryBlueprintPersistence implements BlueprintsPersistence{

        private final Map<Tuple<String,String>,Blueprint> blueprints=new HashMap<>(); 7 usages

```

2. Complete las operaciones `getBlueprint()` y `getBlueprintsByAuthor()`. Implemente todo lo requerido de las capas inferiores (por ahora, el esquema de persistencia disponible 'InMemoryBlueprintPersistence') agregando las pruebas correspondientes en 'InMemoryPersistenceTest'.

getBlueprint()

```
/**
 * Obtiene un blueprint por autor y nombre.
 * @param author Nombre del autor.
 * @param bprintname Nombre del blueprint.
 * @return El blueprint correspondiente.
 * @throws BlueprintNotFoundException si no se encuentra el blueprint.
 */
@Override 6 usages 1 LaaSofiaa +1
public Blueprint getBlueprint(String author, String bprintname) throws BlueprintNotFoundException {
    String authorTrimmed = author.trim();
    String nameTrimmed = bprintname.trim();
    Blueprint bp = blueprints.get(new Tuple<>(authorTrimmed, nameTrimmed));
    if (bp == null) {
        throw new BlueprintNotFoundException("Blueprint not found: " + author + " - " + bprintname);
    }
    return bp;
}
```

nce > impl > InMemoryBlueprintPersistence > getBlueprintsByAuthor

78:1 CRLF UTF-8 4 spaces



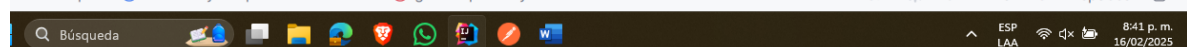
getBlueprintsByAuthor()

```
/**
 * Obtiene todos los blueprints de un autor específico.
 * @param author Nombre del autor.
 * @return Un conjunto de blueprints del autor.
 * @throws BlueprintNotFoundException si el autor no tiene blueprints.
 */
@Override 3 usages 1 LaaSofiaa +1
public Set<Blueprint> getBlueprintsByAuthor(String author) throws BlueprintNotFoundException {
    String authorTrimmed = author.trim();
    Set<Blueprint> result = new HashSet<>();
    for (Map.Entry<Tuple<String, String>, Blueprint> entry : blueprints.entrySet()) {
        if (entry.getKey().getElem1().equals(authorTrimmed)) {
            result.add(entry.getValue());
        }
    }

    if (result.isEmpty()) {
        throw new BlueprintNotFoundException("No blueprints found for author: " + author);
    }
    return result;
}
```

nce > impl > InMemoryBlueprintPersistence > getBlueprintsByAuthor

78:1 CRLF UTF-8 4 spaces



Pruebas correspondientes

The image shows a screenshot of an IDE with two windows. The top window displays the source code for `InMemoryPersistenceTest`. The code includes several test methods: `saveAndRetrieveBlueprint()`, `getNonExistingBlueprintThrowsException()`, `getBlueprintsByAuthor()`, and `getBlueprintsByNonExistingAuthor()`. The bottom window shows the 'Run' output, indicating that all 8 tests passed successfully in 45 ms. The output also lists the execution time for each individual test method.

```
void saveAndRetrieveBlueprint() throws Exception {
    Blueprint bp = new Blueprint( author: "Javier", name: "ARWS", new Point[]{new Point( x: 10, y: 10), new Point( x: 20, y: 20)});
    persistence.saveBlueprint(bp);
    assertEquals(bp, persistence.getBlueprint( author: "Javier", bprintname: "ARWS"));
}

@Test
void getNonExistingBlueprintThrowsException() {
    assertThrows(BlueprintNotFoundException.class, () -> persistence.getBlueprint( author: "Desconocido", bprintname: "NoBlueprint"));
}

@Test
void getBlueprintsByAuthor() throws Exception {
    Blueprint bp1 = new Blueprint( author: "Alice", name: "IETI1", new Point[]{new Point( x: 10, y: 10)});
    Blueprint bp2 = new Blueprint( author: "Alice", name: "IETI2", new Point[]{new Point( x: 20, y: 20)});
    persistence.saveBlueprint(bp1);
    persistence.saveBlueprint(bp2);

    Set<Blueprint> blueprints = persistence.getBlueprintsByAuthor("Alice");
    assertEquals( expected: 2, blueprints.size());
}

@Test
void getBlueprintsByNonExistingAuthor() {
    assertThrows(BlueprintNotFoundException.class, () -> persistence.getBlueprintsByAuthor("NoAutor"));
}
```

Run InMemoryPersistenceTest x

Tests passed: 8 of 8 tests - 45 ms

Process finished with exit code 0

- ✓ InMemoryPersistenceTest (edu.ecl.arsw.blueprints.test.persistence.impl) 45 ms
- ✓ getBlueprintsByAut 29 ms
- ✓ saveExistingBlueprin 4 ms
- ✓ getAllBlueprints() 3 ms
- ✓ saveNewAndLoadTe 2 ms
- ✓ blueprintNamesShou 1 ms
- ✓ getBlueprintsByNonI 2 ms
- ✓ getNonExistingBlue 2 ms

3. Haga un programa en el que cree (mediante Spring) una instancia de `BlueprintServices`, y rectifique la funcionalidad del mismo: registrar planos, consultar planos, registrar planos específicos, etc.

Se realizó la implementación de los siguientes métodos en la clase `BlueprintsServices`

```
/**
 * Agrega un nuevo blueprint al sistema.
 * @param bp El blueprint a agregar.
 * @throws UnsupportedOperationException si el blueprint ya existe.
 */
public void addNewBlueprint(Blueprint bp){ 1 usage  ± ZayraGS1403 +1
    try{
        bpp.saveBlueprint(bp);
    }catch (Exception ex){
        ex.printStackTrace();
        throw new UnsupportedOperationException("User already exists!");
    }
}
```

ces > BlueprintsServices > addNewBlueprint 29:13 CRLF UTF-8 4 spaces 9:08 p. m. 16/02/2025

```
/**
 * Obtiene todos los blueprints disponibles, aplicando el filtro correspondiente.
 * @return Un conjunto de todos los blueprints filtrados.
 */
public Set<Blueprint> getAllBlueprints(){ 1 usage  ± ZayraGS1403
    return filter.filterBlueprints(bpp.getAllBluePrints());
}

/**
 * Obtiene un blueprint específico basado en el autor y el nombre.
 * @param author Autor del blueprint.
 * @param name Nombre del blueprint.
 * @return El blueprint filtrado.
 * @throws BlueprintNotFoundException si no se encuentra el blueprint.
 */
public Blueprint getBlueprint(String author,String name) throws BlueprintNotFoundException{ 1 usage  ± ZayraGS1403
    return filter.filterPlain(bpp.getBlueprint(author, name));
}
```

ices > BlueprintsServices > addNewBlueprint 29:13 CRLF UTF-8 4 spaces 9:08 p. m. 16/02/2025

```
/**
 * Obtiene todos los blueprints de un autor específico.
 * @param author Autor de los blueprints.
 * @return Un conjunto de blueprints filtrados del autor.
 * @throws BlueprintNotFoundException si el autor no tiene blueprints.
 */
public Set<Blueprint> getBlueprintsByAuthor(String author) throws BlueprintNotFoundException{ 1 usage  ± ZayraGS1403
    return filter.filterBlueprints(bpp.getBlueprintsByAuthor(author));
}
```

ces > BlueprintsServices > addNewBlueprint 29:13 CRLF UTF-8 4 spaces 9:08 p. m. 16/02/2025

Respectiva implementación en la clase BlueprintController, en la cual creamos un Rest Controller para poder realizar las respectivas pruebas en Postman

```
/**
 * Controlador REST para gestionar los blueprints.
 */
@RestController  ± ZayraGS1403 +1
@RequestMapping("/blueprints")
public class BlueprintController {

    @Autowired
    private final BlueprintsServices bps;

    /**
     * Constructor de BlueprintController.
     * @param bps Servicio de blueprints inyectado.
     */
    public BlueprintController(BlueprintsServices bps) {  ± ZayraGS1403
        this.bps = bps;
    }

    /**
     * Obtiene un blueprint específico por autor y nombre.
     * @param author Nombre del autor del blueprint.
     * @param name Nombre del blueprint.
     * @return ResponseEntity con el blueprint solicitado o un error si no se encuentra.
     */
}

Activar Windows
Ver configuración de Windows.
```

BluePrintController 18:1 CRLF UTF-8 4 spaces

Resultado 9:32 p. m. 16/02/2025

```
/**
 * Obtiene un blueprint específico por autor y nombre.
 * @param author Nombre del autor del blueprint.
 * @param name Nombre del blueprint.
 * @return ResponseEntity con el blueprint solicitado o un error si no se encuentra.
 */
@GetMapping("/{author}/{name}")  ± ZayraGS1403
public ResponseEntity<?> getBlueprint(@PathVariable("author") String author, @PathVariable("name") String name) {
    try {
        return ResponseEntity.ok(bps.getBlueprint(author, name));
    } catch (BlueprintNotFoundException e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Blueprint not found: " + author + "/" + name);
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred.");
    }
}
}
```

controller > BluePrintController > registerBlueprint 77:47 CRLF UTF-8 4 spaces

Búsqueda 9:11 p. m. 16/02/2025

```
/**
 * Obtiene todos los blueprints de un autor específico.
 * @param author Nombre del autor.
 * @return ResponseEntity con el conjunto de blueprints o un error si no se encuentran.
 */
@GetMapping("/{author}") // ZayraGS1403
public ResponseEntity<?> getBlueprintByAuthor(@PathVariable("author") String author) {
    try {
        Set<Blueprint> blueprints = bps.getBlueprintsByAuthor(author);
        return ResponseEntity.ok(blueprints);
    } catch (BlueprintNotFoundException e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Blueprints not found for author: " + author);
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred.");
    }
}
}
```

ntroller > BluePrintController > registerBlueprint 77:47 CRLF UTF-8 4 spaces

Búsqueda 9:12 p. m. 16/02/2025

```
/**
 * Registra un nuevo blueprint en el sistema.
 * @param bp Objeto Blueprint a registrar.
 * @return ResponseEntity con el estado de la operación.
 */
@PostMapping // ZayraGS1403
public ResponseEntity<?> registerBlueprint(@RequestBody Blueprint bp){
    HashMap<String, Object> response = new HashMap<>();
    try {
        bps.addNewBlueprint(bp);
        response.put("status", "success");
        return ResponseEntity.status(HttpStatus.OK).body(response);
    } catch (Exception e) {
        response.put("error", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }
}
}
```

troller > BluePrintController > registerBlueprint 77:47 CRLF UTF-8 4 spaces

Búsqueda 9:12 p. m. 16/02/2025

```
/**
 * Obtiene todos los blueprints disponibles en el sistema.
 * @return Conjunto de blueprints.
 */
@GetMapping // ZayraGS1403
public Set<Blueprint> getAllBlueprints(){
    return bps.getAllBlueprints();
}
}
```

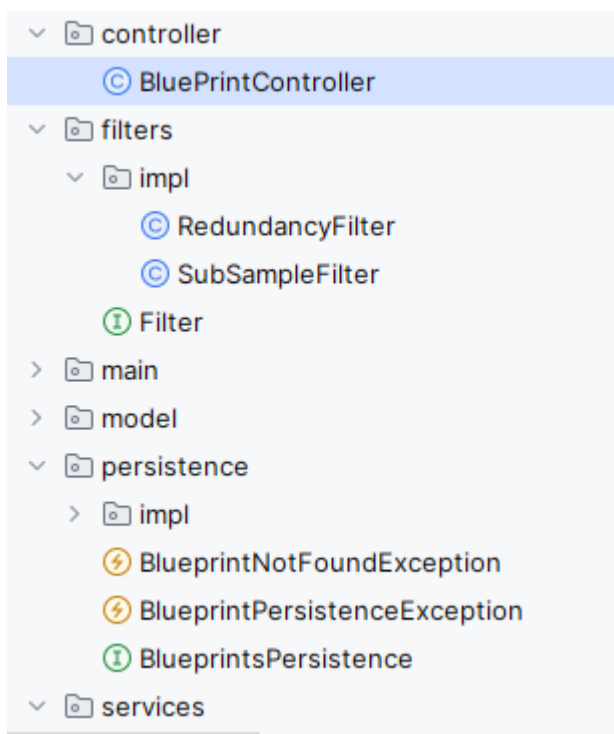
roller > BluePrintController > registerBlueprint 77:47 CRLF UTF-8 4 spaces

Búsqueda 9:12 p. m. 16/02/2025

Por último se crea la implementación de la clase main, la cual ayudará a inicializar la instancia de BlueprintServices y a probar todos sus métodos

4. Se quiere que las operaciones de consulta de planos realicen un proceso de filtrado, antes de retornar los planos consultados. Dichos filtros lo que buscan es reducir el tamaño de los planos, removiendo datos redundantes o simplemente submuestrando, antes de retornarlos. Ajuste la aplicación (agregando las abstracciones e implementaciones que considere) para que a la clase BlueprintServices se le inyecte uno de dos posibles 'filtros' (o eventuales futuros filtros). No se contempla el uso de más de uno a la vez:

Para ello creamos una interfaz Filter y sus implementaciones:



Interfaz del filtrado

```
/*
 * Interfaz para la aplicación de filtros sobre los blueprints.
 */
public interface Filter {
    public abstract Blueprint filterPlain(Blueprint blueprint);
    public Set<Blueprint> filterBlueprints(Set<Blueprint> blueprints);
}
```

- (A) Filtrado de redundancias: suprime del plano los puntos consecutivos que sean repetidos.

```
//@Service
public class RedundancyFilter implements Filter { no usages ± ZayraGS1403 +1

    /**
     * Filtra un blueprint eliminando puntos consecutivos duplicados.
     * @param blueprint Blueprint a filtrar.
     * @return Blueprint filtrado sin puntos redundantes.
     */
    @Override 3 usages ± ZayraGS1403
    public Blueprint filterPlain(Blueprint blueprint){

        List<Point> originalPoints = blueprint.getPoints();
        // Si solo hay un punto, no hay nada que filtrar
        if (blueprint.getPoints().size() <= 1) {
            return blueprint;
        }

        List<Point> filteredPoints = new ArrayList<>();
        filteredPoints.add(originalPoints.get(0));

        for (int i = 1; i < originalPoints.size(); i++) {
            Point prev = originalPoints.get(i-1);
            Point current = originalPoints.get(i);

            if (!(prev.getX() == current.getX() && prev.getY() == current.getY())) {
                filteredPoints.add(current);
            }
        }
    }
}
```

```
// Convertimos la lista a un array de puntos
Point[] resultPoints = filteredPoints.toArray(new Point[0]);
return new Blueprint(blueprint.getAuthor(), blueprint.getName(), resultPoints);
}

/**
 * Filtra un conjunto de blueprints aplicando el filtro de redundancia a cada uno.
 * @param blueprints Conjunto de blueprints a filtrar.
 * @return Conjunto de blueprints filtrados.
 */
@Override 2 usages ± ZayraGS1403
public Set<Blueprint> filterBlueprints(Set<Blueprint> blueprints){
    Set<Blueprint> newBlueprintSet = new HashSet<>();

    for(Blueprint i: blueprints){
        newBlueprintSet.add(filterPlain(i));
    }
    return newBlueprintSet;
}
}
```

- (B) Filtrado de submuestreo: suprime 1 de cada 2 puntos del plano, de manera intercalada.

```

/**
 * Filtra un blueprint eliminando uno de cada dos puntos.
 * @param blueprint Blueprint a filtrar.
 * @return Blueprint filtrado con la mitad de los puntos originales.
 */
@Override 3 usages 1 ZayraGS1403
public Blueprint filterPlain(Blueprint blueprint) {

    List<Point> originalPoints = blueprint.getPoints();
    // Si solo hay un punto, no hay nada que filtrar
    if (blueprint.getPoints().size() <= 1) {
        return blueprint;
    }

    // Crear un nuevo array con la mitad de los puntos (eliminando 1 de cada 2)
    Point[] filteredPoints = new Point[(originalPoints.size() + 1) / 2];
    for (int i = 0, j = 0; i < originalPoints.size(); i += 2, j++) {
        filteredPoints[j] = originalPoints.get(i);
    }

    return new Blueprint(blueprint.getAuthor(), blueprint.getName(), filteredPoints);
}

/**
 * Filtra un conjunto de blueprints aplicando el submuestreo a cada uno.
 * @param blueprints Conjunto de blueprints a filtrar.
 * @return Conjunto de blueprints filtrados.
 */
@Override 2 usages 1 ZayraGS1403
public Set<Blueprint> filterBlueprints(Set<Blueprint> blueprints) {
    Set<Blueprint> newBlueprintSet = new HashSet<>();

    for (Blueprint i : blueprints) {
        newBlueprintSet.add(filterPlain(i));
    }

    return newBlueprintSet;
}

```

main > java > edu > eci > arsw > blueprints > filters > impl > SubSampleFilter 13:14 CRLF UTF-8 4 spaces

```

/**
 * Filtra un conjunto de blueprints aplicando el submuestreo a cada uno.
 * @param blueprints Conjunto de blueprints a filtrar.
 * @return Conjunto de blueprints filtrados.
 */
@Override 2 usages 1 ZayraGS1403
public Set<Blueprint> filterBlueprints(Set<Blueprint> blueprints) {
    Set<Blueprint> newBlueprintSet = new HashSet<>();

    for (Blueprint i : blueprints) {
        newBlueprintSet.add(filterPlain(i));
    }

    return newBlueprintSet;
}

```

main > java > edu > eci > arsw > blueprints > filters > impl > SubSampleFilter 13:14 CRLF UTF-8 4 spaces

5. Agregue las pruebas correspondientes a cada uno de estos filtros, y pruebe su funcionamiento en el programa de prueba, comprobando que sólo cambiando la posición de las anotaciones -sin cambiar nada más-, el programa retorne los planos filtrados de la manera (A) o de la manera (B).

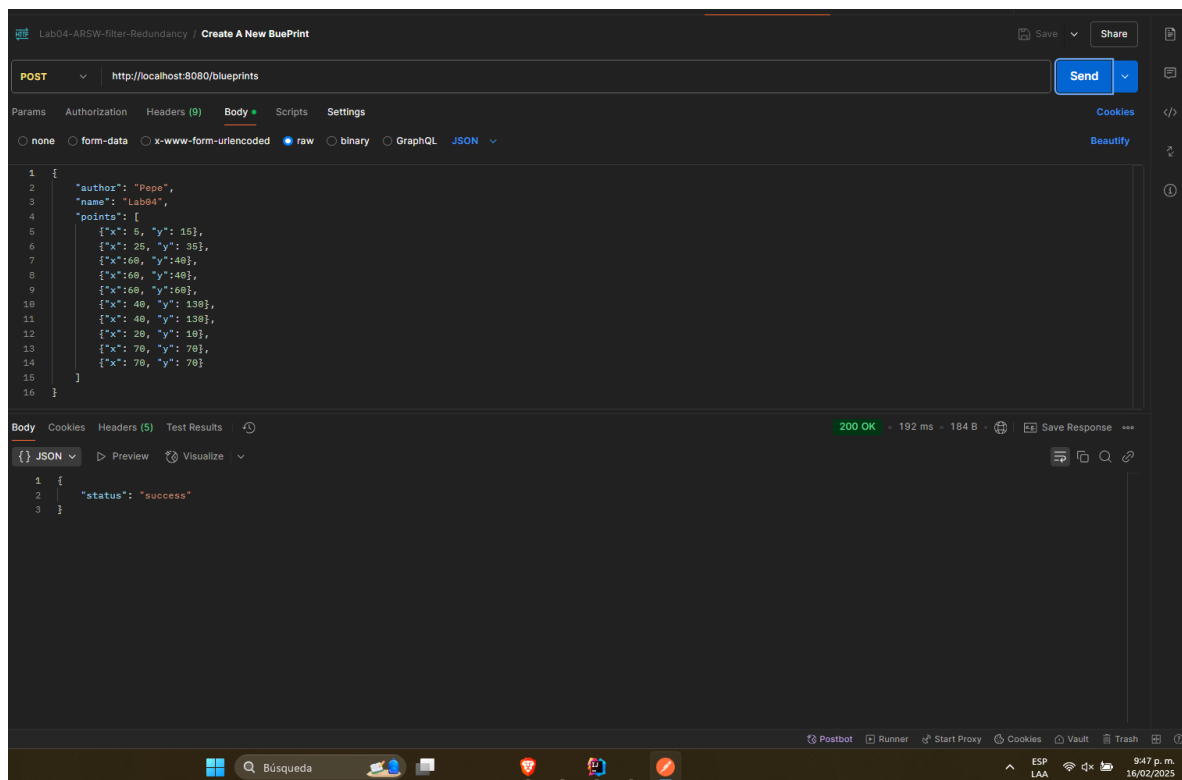
Pruebas con POSTMAN

Colección Filtro “REDUNDANCY”

El primer paso es correr el código con el comando *mvn spring-boot:start*

Posteriormente abrimos postman

Crear un nuevo blueprint:



Lab04-ARSW-filter-Redundancy / Create A New Blueprint

POST http://localhost:8080/blueprints

Body

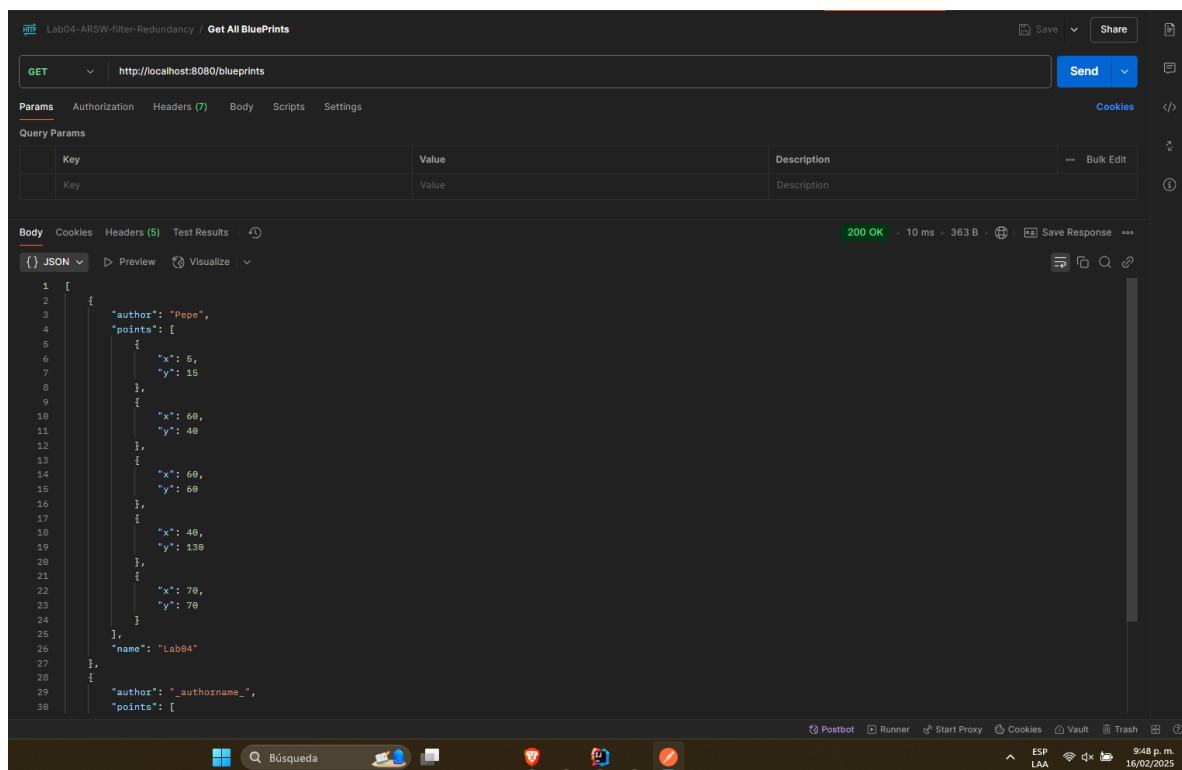
```
1 {
2   "author": "Pepe",
3   "name": "Lab04",
4   "points": [
5     {"x": 5, "y": 15},
6     {"x": 25, "y": 35},
7     {"x": 60, "y": 40},
8     {"x": 60, "y": 40},
9     {"x": 60, "y": 60},
10    {"x": 40, "y": 130},
11    {"x": 40, "y": 130},
12    {"x": 20, "y": 10},
13    {"x": 70, "y": 70},
14    {"x": 70, "y": 70}
15  ]
16 }
```

Body

```
1 {
2   "status": "success"
3 }
```

200 OK - 192 ms - 184 B

Consultar todos los blueprints



Lab04-ARSW-filter-Redundancy / Get All BluePrints

GET http://localhost:8080/blueprints

Body

```
1 [
2   {
3     "author": "Pepe",
4     "points": [
5       {
6         "x": 5,
7         "y": 15
8       },
9       {
10        "x": 60,
11        "y": 40
12      },
13      {
14        "x": 60,
15        "y": 60
16      },
17      {
18        "x": 40,
19        "y": 130
20      },
21      {
22        "x": 70,
23        "y": 70
24      }
25    ],
26     "name": "Lab04"
27   },
28   {
29     "author": "_authorname_",
30     "points": [

```

200 OK - 10 ms - 363 B

Consultar blueprint por nombre y autor

Lab04-ARSW-filter-Redundancy / Get By Author and BluePrint name

GET http://localhost:8080/blueprints/Pepe/Lab04

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

200 OK · 7 ms · 287 B

JSON Preview Visualize

```
1 {
2   "author": "Pepe",
3   "points": [
4     {
5       "x": 5,
6       "y": 15
7     },
8     {
9       "x": 60,
10      "y": 40
11     },
12     {
13       "x": 60,
14       "y": 60
15     },
16     {
17       "x": 40,
18       "y": 130
19     },
20     {
21       "x": 70,
22       "y": 70
23     }
24   ],
25   "name": "Lab04"
26 }
```

Postbot Runner Start Proxy Cookies Vault Trash

Búsqueda

ESP LAA 9:49 p. m. 16/02/2025

Consultar blueprints de un autor

Lab04-ARSW-filter-Redundancy / Get BluePrint By Author

GET http://localhost:8080/blueprints/Pepe

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

200 OK · 5 ms · 289 B

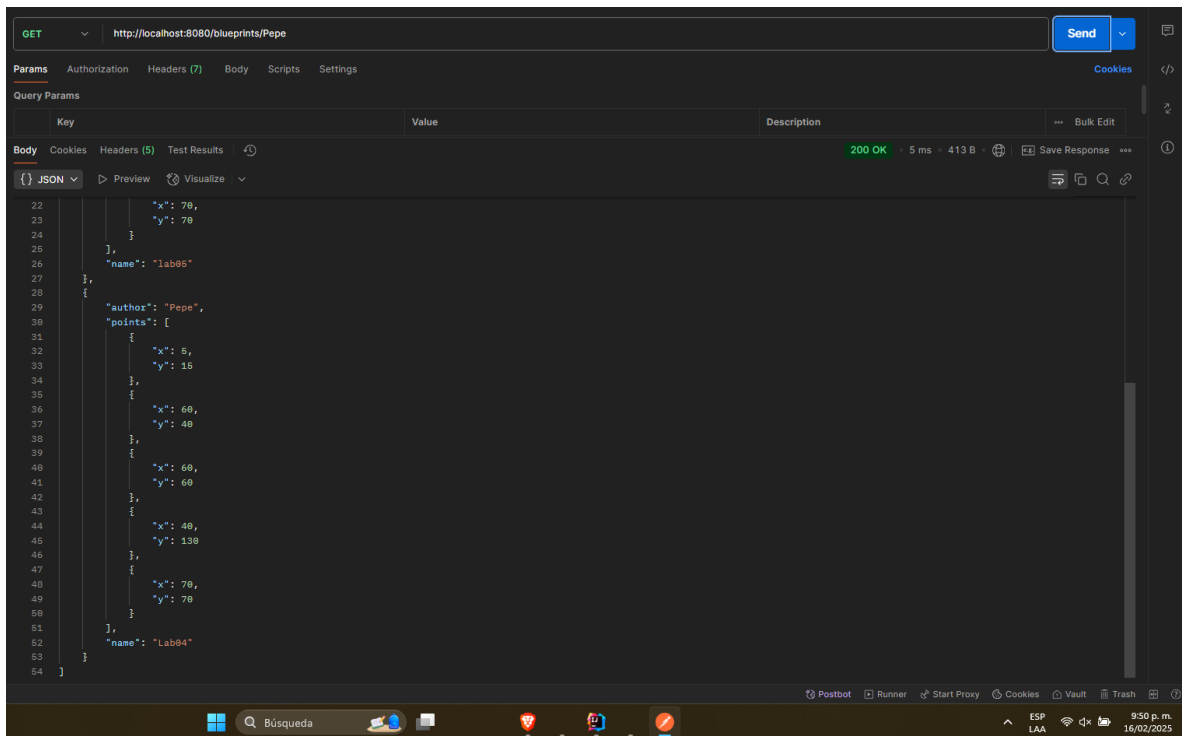
JSON Preview Visualize

```
1 [
2   {
3     "author": "Pepe",
4     "points": [
5       {
6         "x": 5,
7         "y": 15
8       },
9       {
10        "x": 60,
11        "y": 40
12       },
13       {
14        "x": 60,
15        "y": 60
16       },
17       {
18        "x": 40,
19        "y": 130
20       },
21       {
22        "x": 70,
23        "y": 70
24       }
25     ],
26     "name": "Lab04"
27   }
28 ]
```

Postbot Runner Start Proxy Cookies Vault Trash

Búsqueda

ESP LAA 9:49 p. m. 16/02/2025



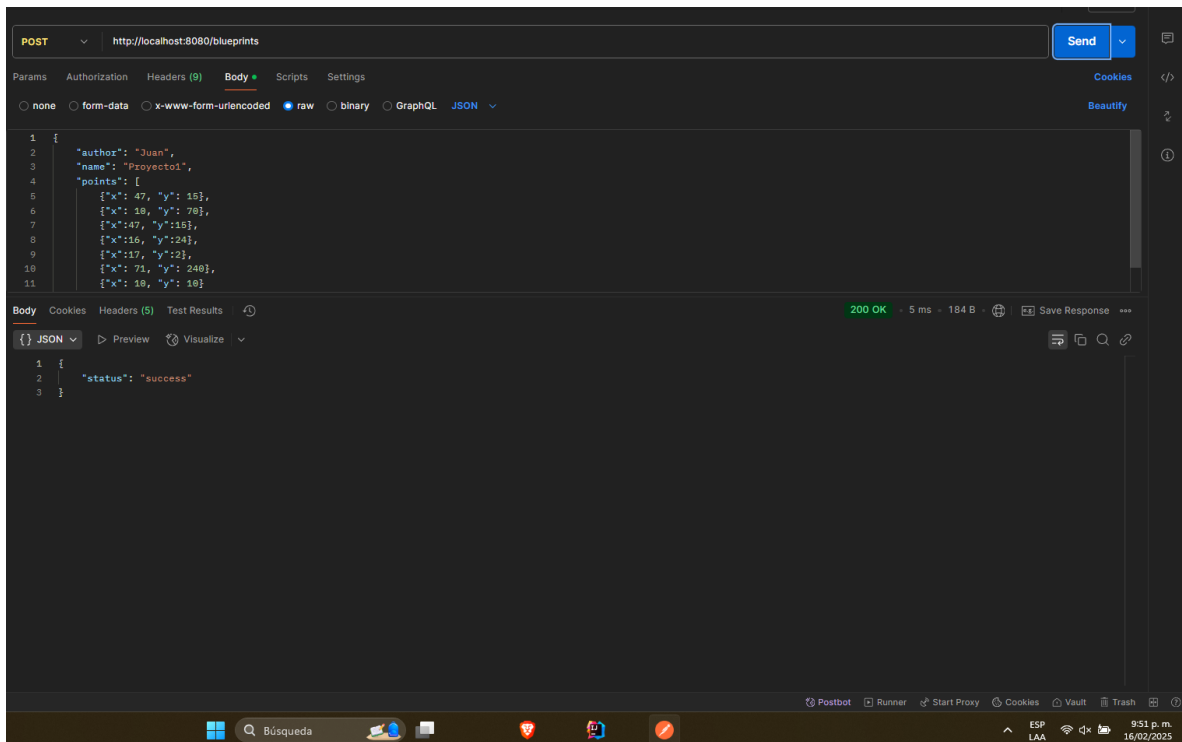
Se puede evidenciar como el filtro en realidad funciona, pues al crear un nuevo blueprint con dos puntos seguidos iguales, elimina uno de ellos.

Colección Filtro “SubSample”

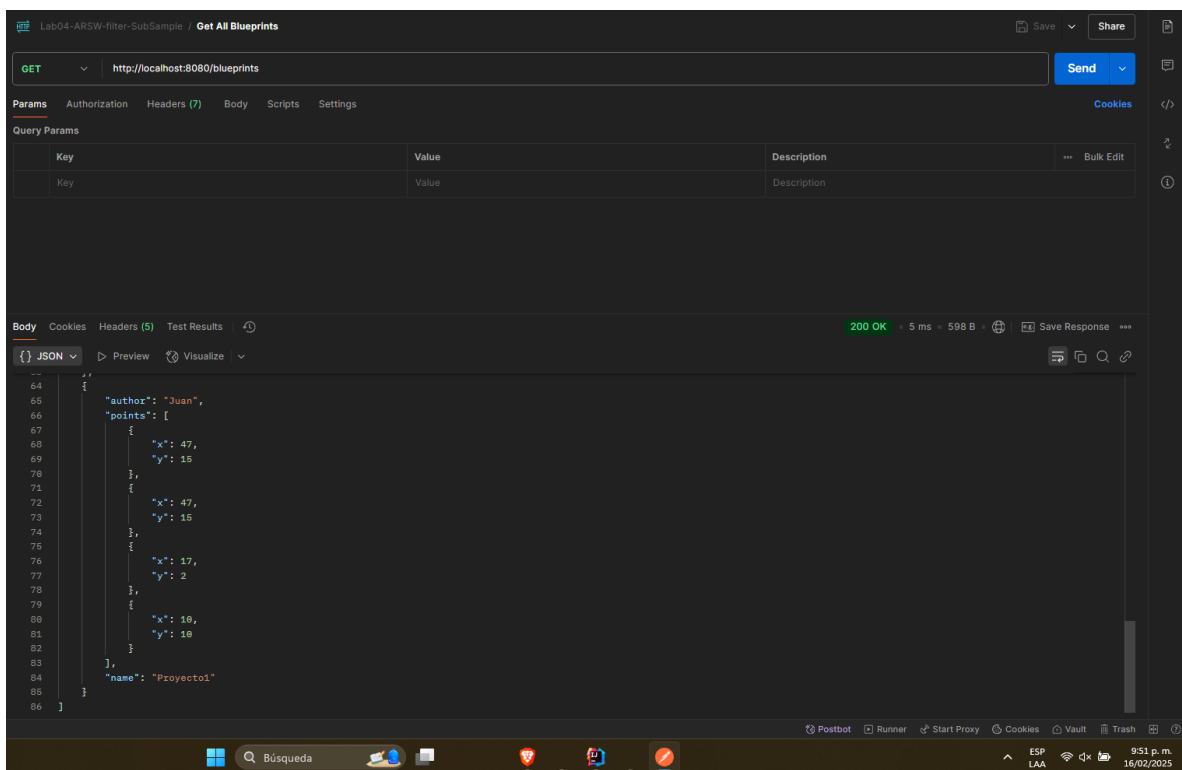
El primer paso es correr el código con el comando *mvn spring-boot:start*

Posteriormente abrimos postman

Crear un nuevo blueprint:



Consultar todos los blueprints



Consultar blueprint por nombre y autor

Lab04-ARSW-filter-SubSample / **Ge BluePrint By Author And Name** Save Share

GET ▼ Send ▼

Params Authorization Headers (7) Body Scripts Settings Cookies />

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results 200 OK 4 ms 274 B Save Response ++

{} JSON Preview Visualize ▼

```
1 {
2   "author": "Juan",
3   "points": [
4     {
5       "x": 47,
6       "y": 15
7     },
8     {
9       "x": 47,
10      "y": 15
11     },
12     {
13       "x": 17,
14       "y": 2
15     },
16     {
17       "x": 10,
18       "y": 10
19     }
20   ],
21   "name": "Proyecto1"
22 }
```

Postbot Runner Start Proxy Cookies Vault Trash

Búsqueda

ESP LAA 9:51 p. m. 16/02/2025

Consultar blueprints de un autor

Lab04-ARSW-filter-SubSample / **Get BluePrint By Author** Save Share

GET ▼ Send ▼

Params Authorization Headers (7) Body Scripts Settings Cookies />

Body Cookies Headers (5) Test Results 200 OK 4 ms 387 B Save Response ++

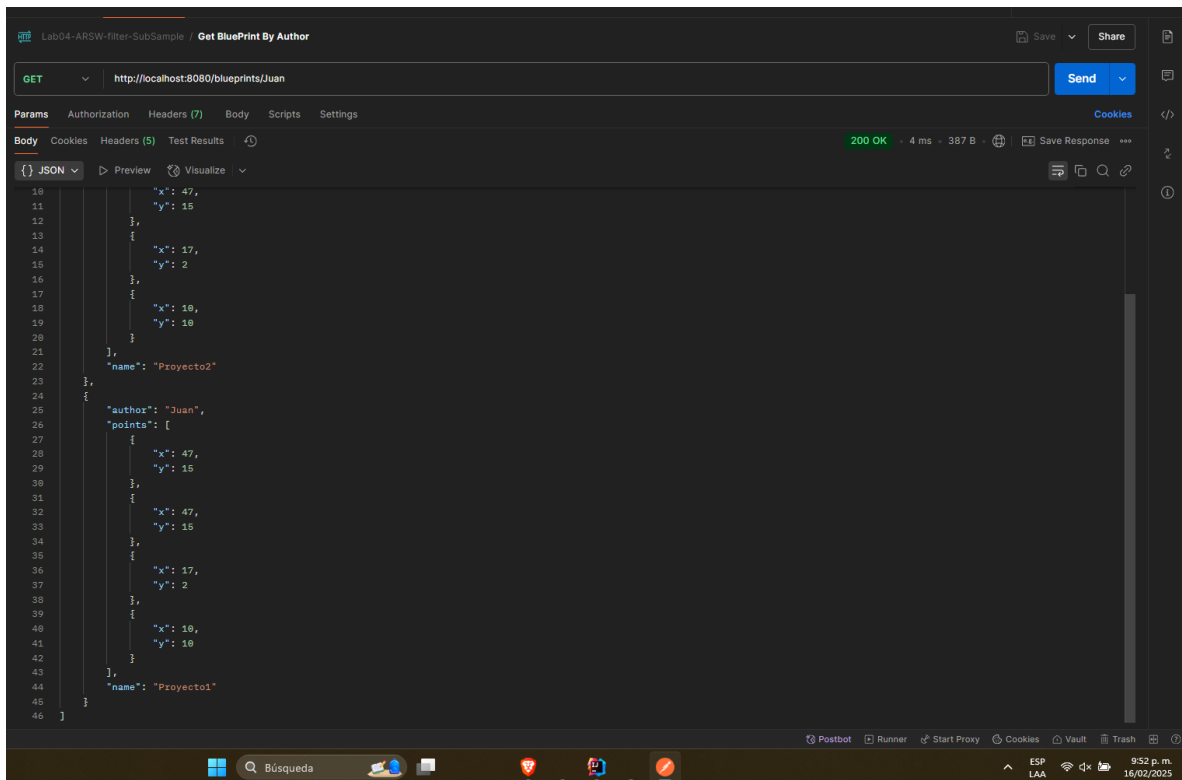
{} JSON Preview Visualize ▼

```
1 [
2   {
3     "author": "Juan",
4     "points": [
5       {
6         "x": 47,
7         "y": 15
8       },
9       {
10        "x": 47,
11        "y": 15
12      },
13      {
14        "x": 17,
15        "y": 2
16      },
17      {
18        "x": 10,
19        "y": 10
20      }
21    ],
22    "name": "Proyecto2"
23  },
24  {
25    "author": "Juan",
26    "points": [
27      {
28        "x": 47,
29        "y": 15
30      },
31      {
32        "x": 47,
33        "y": 15
34      },
35      {
36        "x": 17,
37        "y": 2
38      }
39    ]
40  }
41 ]
```

Postbot Runner Start Proxy Cookies Vault Trash

Búsqueda

ESP LAA 9:52 p. m. 16/02/2025



En este caso el filtro funciona correctamente, pues al consultar una blueprint con ese filtro se logra evidenciar como solo se muestran los puntos intercalados y no todos.

Lo mejor es que este cambio de filtros se logra relativamente fácil, simplemente se debe poner la anotación de `@Service` en la implementación del filtro que queremos inyectar, por lo que, si en un futuro se crea otro filtro, no es difícil inyectarlo

Las colecciones usadas para las pruebas en postman están puestas en este repositorio

Conclusiones

La aplicación de la inyección de dependencias mediante Spring permite desarrollar sistemas más modulares y mantenibles, facilitando la sustitución y prueba de componentes sin afectar el resto de la aplicación.

La arquitectura implementada posibilita la expansión del sistema mediante la adición de nuevos filtros y servicios sin alterar la estructura existente.

La implementación de pruebas en Postman valida correctamente la funcionalidad del sistema, asegurando que los filtros operen según lo esperado.

El uso de persistencia en memoria simplifica el desarrollo y prueba de la aplicación sin necesidad de una base de datos externa; sin embargo, para entornos de producción se recomienda integrar un sistema de persistencia más robusto.

La implementación basada en Spring demuestra la importancia de utilizar frameworks adecuados para la gestión eficiente de aplicaciones empresariales.