



## Laboratorio 5

API REST para la gestión de planos.

url repositorio:

[https://github.com/ARSW2023-2/ARSW\\_Lab05.git](https://github.com/ARSW2023-2/ARSW_Lab05.git)

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Squad:

Inside Out

Profesor:

Javier Iván Toquica Barrera

Curso:

ARSW – 1

Fecha De Entrega:

22-09-2023

# I. Introducción

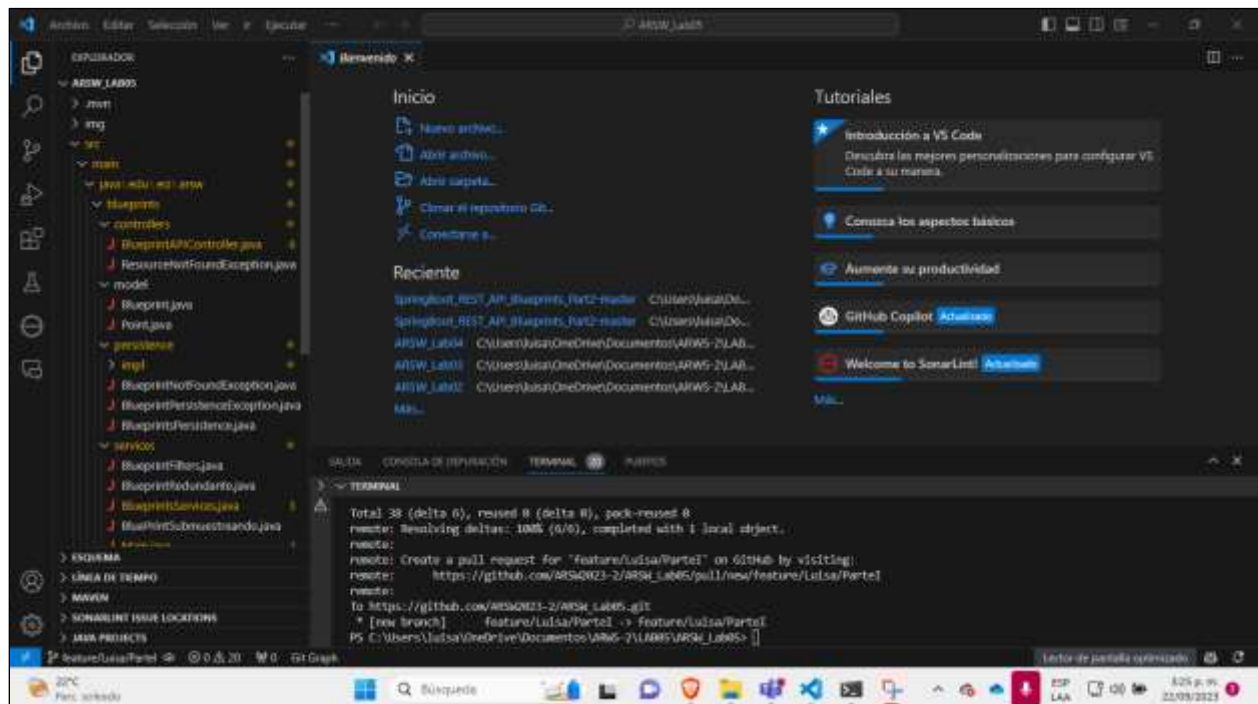
Este informe detalla la creación y desarrollo del componente BlueprintsRESTAPI, el cual forma parte de un proyecto dedicado a la gestión de planos arquitectónicos para una renombrada compañía de diseño. El propósito fundamental de este componente radica en proporcionar una interfaz de programación de aplicaciones (API) altamente versátil y de plataforma independiente, permitiendo la administración centralizada de planos arquitectónicos.

La construcción del BlueprintsRESTAPI se enmarca en la arquitectura de servicios web REST, permitiendo la comunicación efectiva entre sistemas a través del protocolo HTTP. Para lograrlo, se hace uso de Spring Boot, un entorno de desarrollo que simplifica la configuración y el despliegue de aplicaciones basadas en Spring Framework. Además, se emplea Spring MVC para definir servicios REST y se aplica el principio de Inversión de Dependencias.

## II.Desarrollo del laboratorio

## Parte I

1. Integre al proyecto base suministrado los Beans desarrollados en el ejercicio anterior. Sólo copie las clases, NO los archivos de configuración. Rectifique que se tenga correctamente configurado el esquema de inyección de dependencias con las anotaciones `@Service` y `@Autowired`.



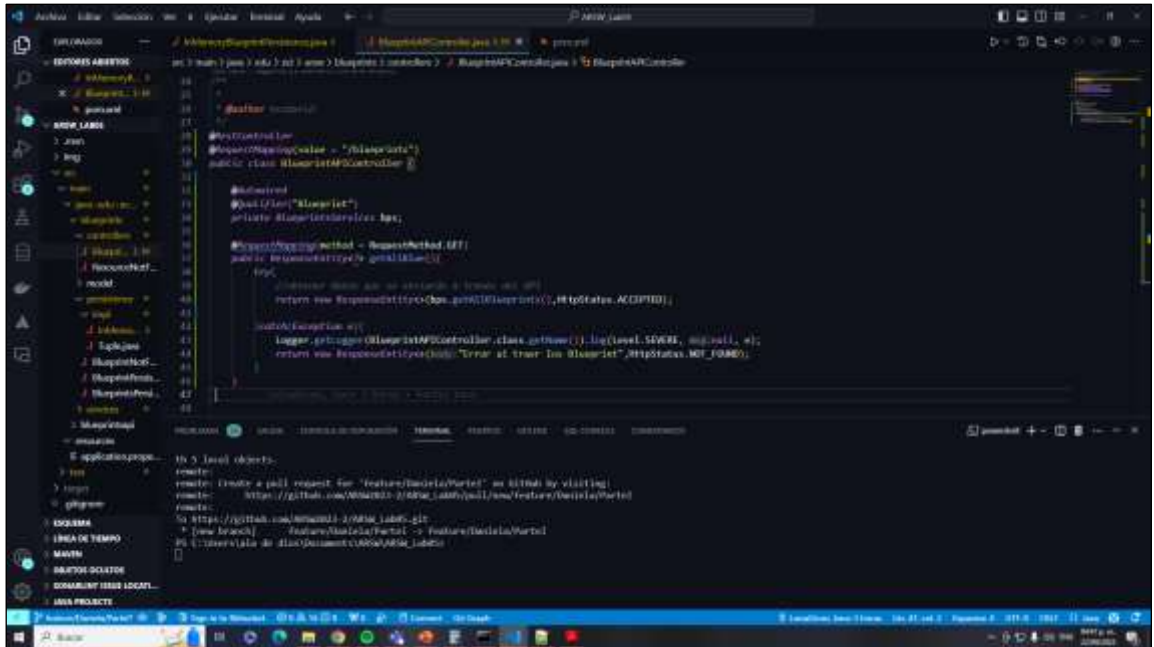
- ## InMemoryBlueprintPersistence

- ```
@RestController
@RequestMapping(value = "/url-raiz-recurso")
public class XXController {

    @RequestMapping(method = RequestMethod.GET)
    public ResponseEntity<?> manejadorGetRecursoXX(){
        try {
            //obtener datos que se enviarán a través del API
            return new ResponseEntity<>(data,HttpStatus.ACCEPTED);
        } catch (XXException ex) {
            Logger.getLogger(XXController.class.getName()).log(Level.SEVERE, null, ex);
            return new ResponseEntity<>("Error bla bla bla",HttpStatus.NOT_FOUND);
        }
    }
}
```

- Haga que en esta misma clase se inyecte el bean de tipo BlueprintServices (al cual, a su vez, se le inyectarán sus dependencias de persistencia y de filtrado de puntos).

### BlueprintApiController



4. Verifique el funcionamiento de la aplicación lanzando la aplicación con maven:

```

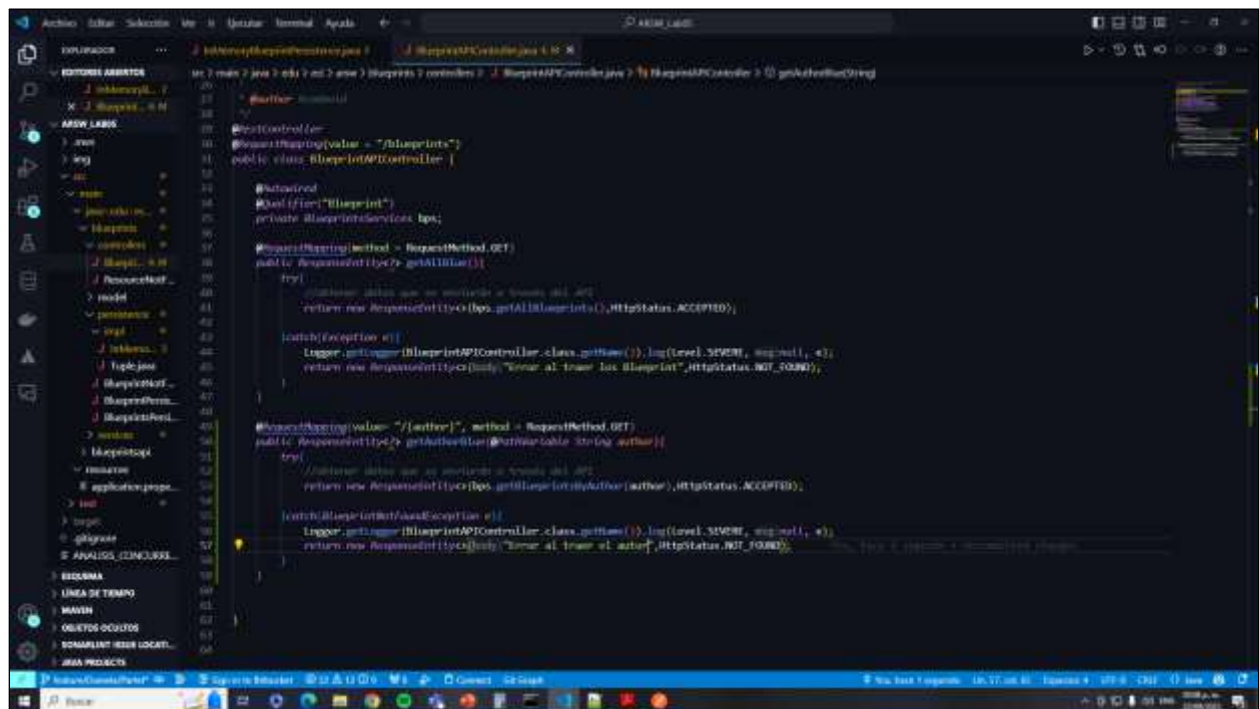
$ mvn compile
$ mvn spring-boot:run

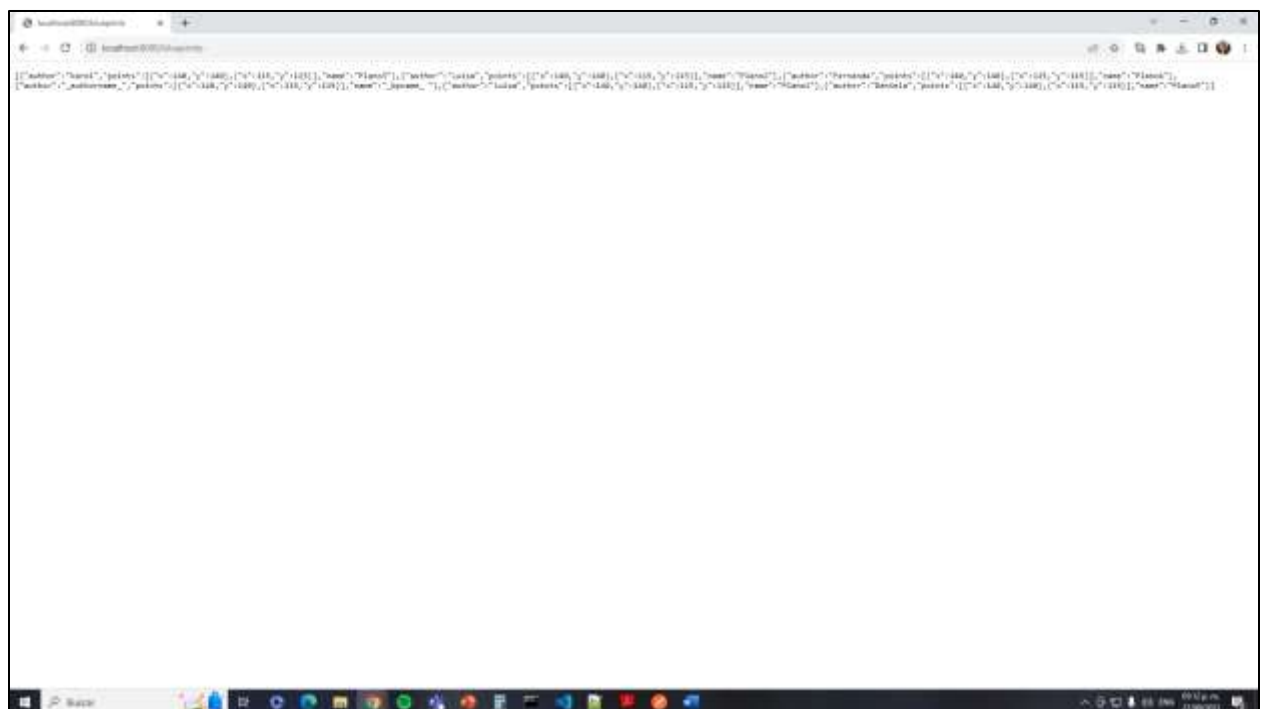
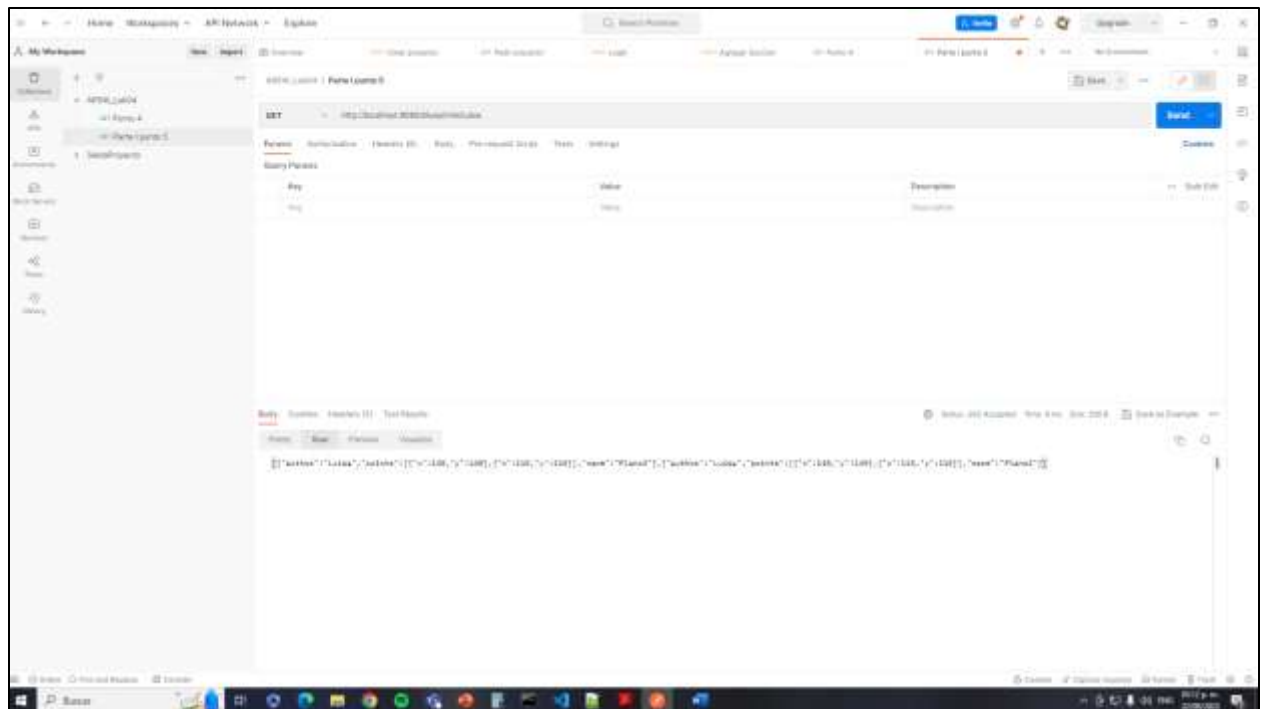
```

Y luego enviando una petición GET a: <http://localhost:8080/blueprints>. Rectifique que, como respuesta, se obtenga un objeto JSON con una lista que contenga el detalle de los planos suministrados por defecto, y que se haya aplicado el filtrado de puntos correspondiente.

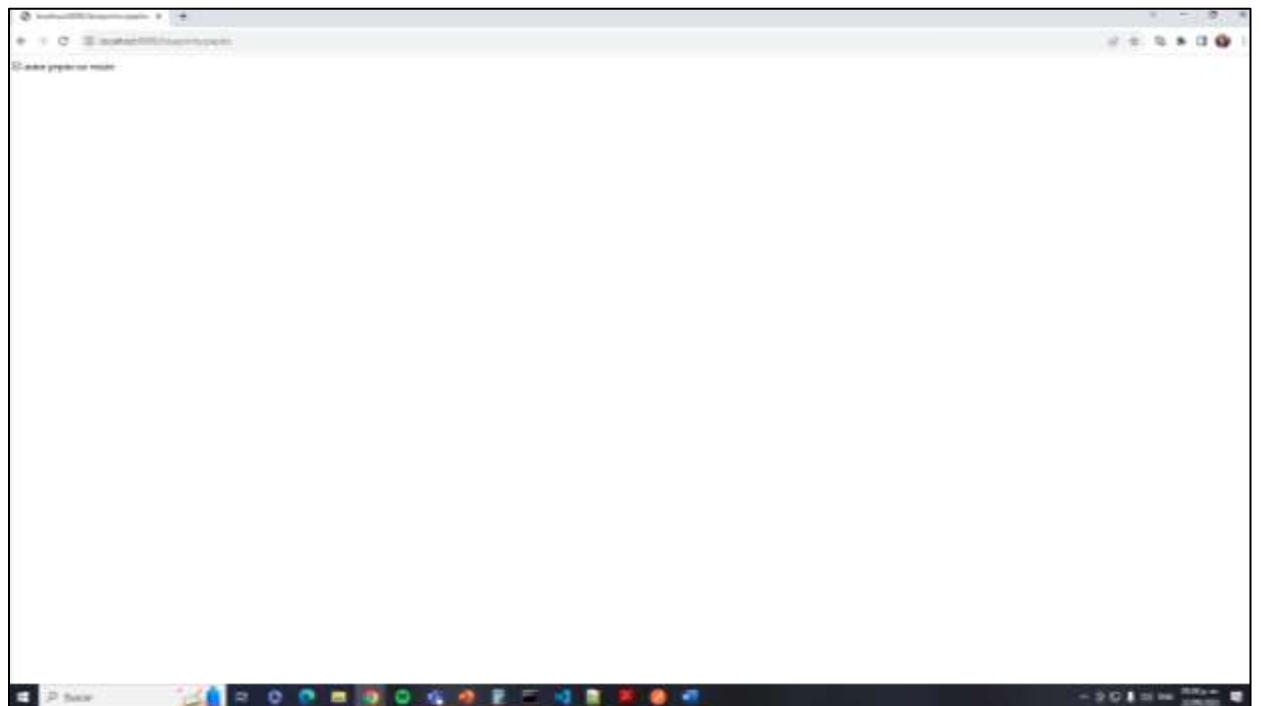
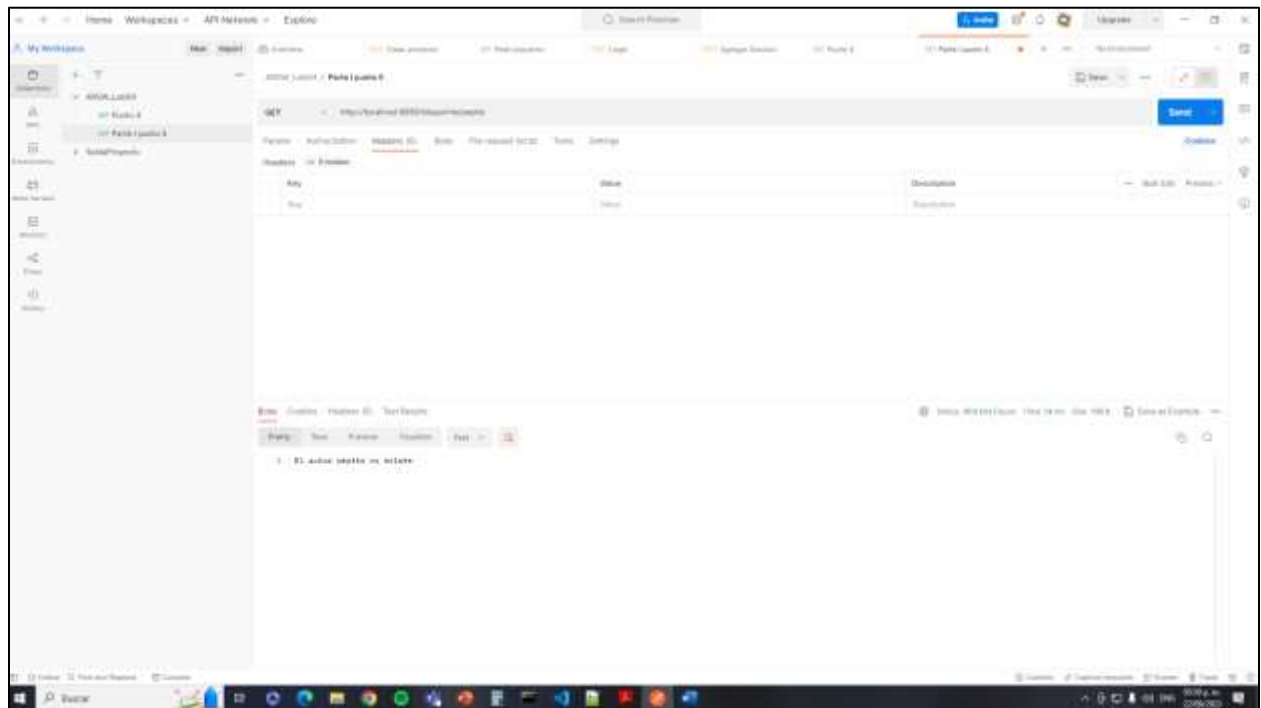


- ## BlueprintApiController





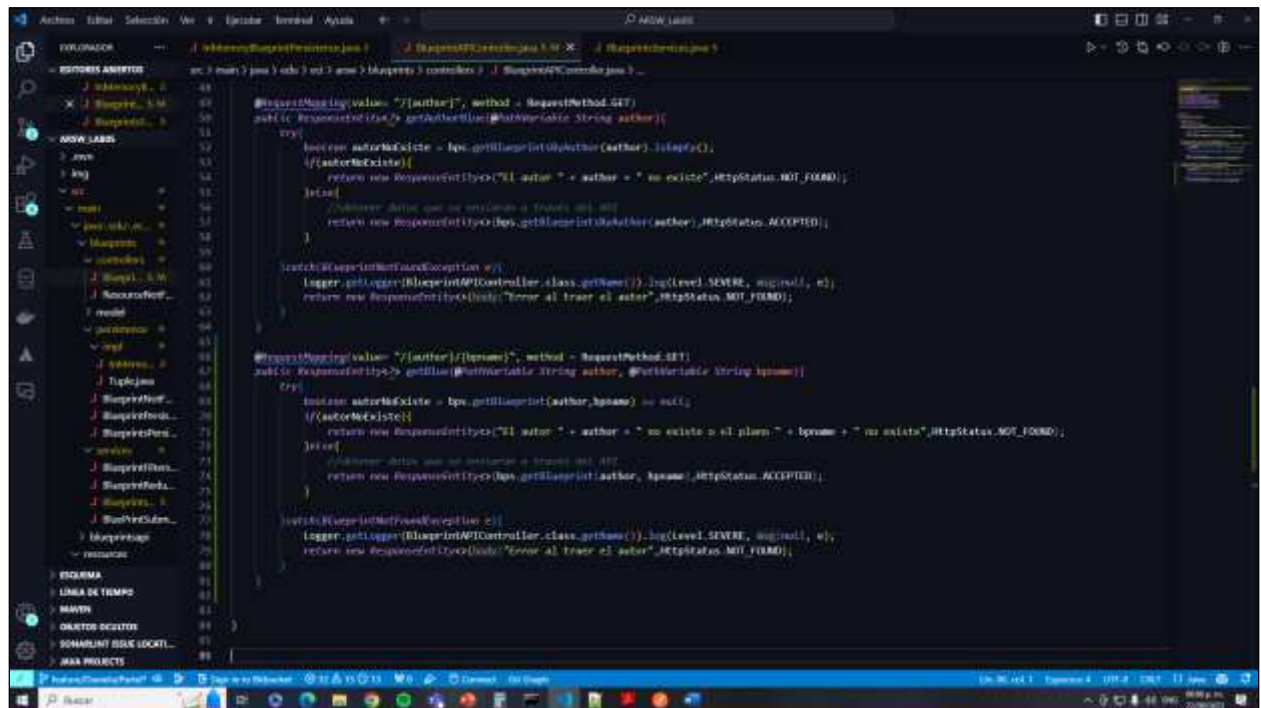
- Si el autor no existe



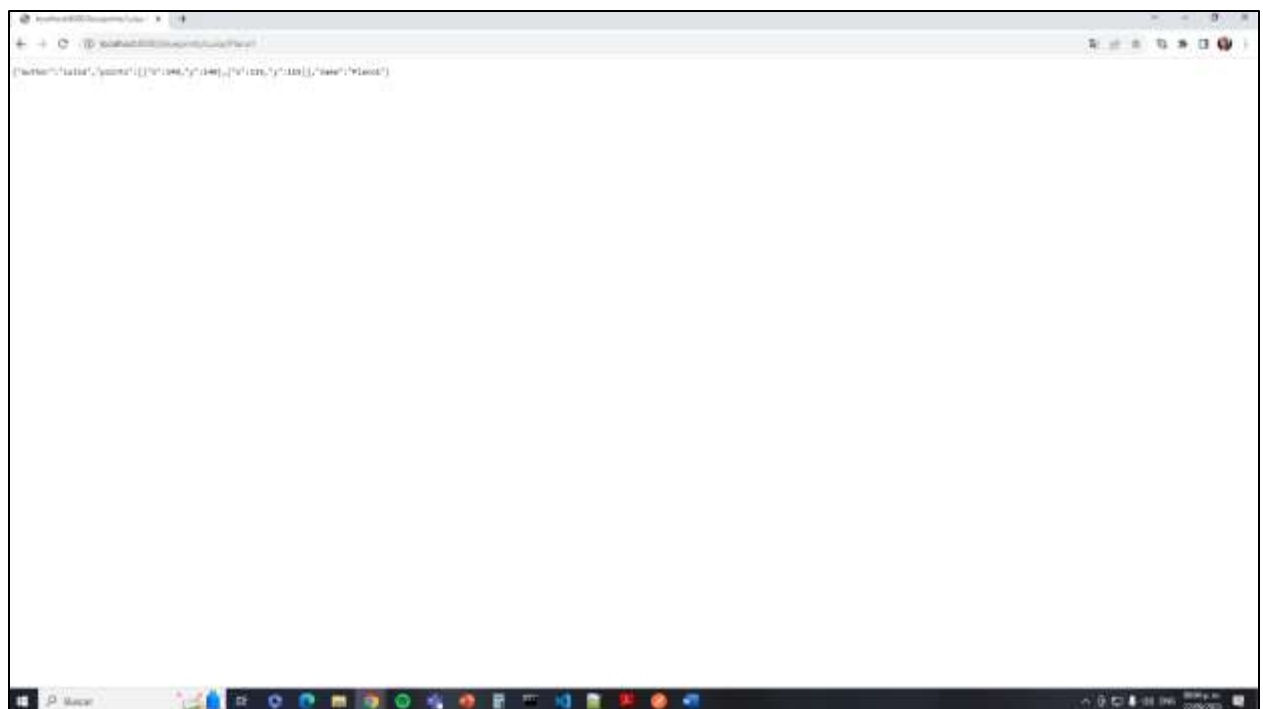
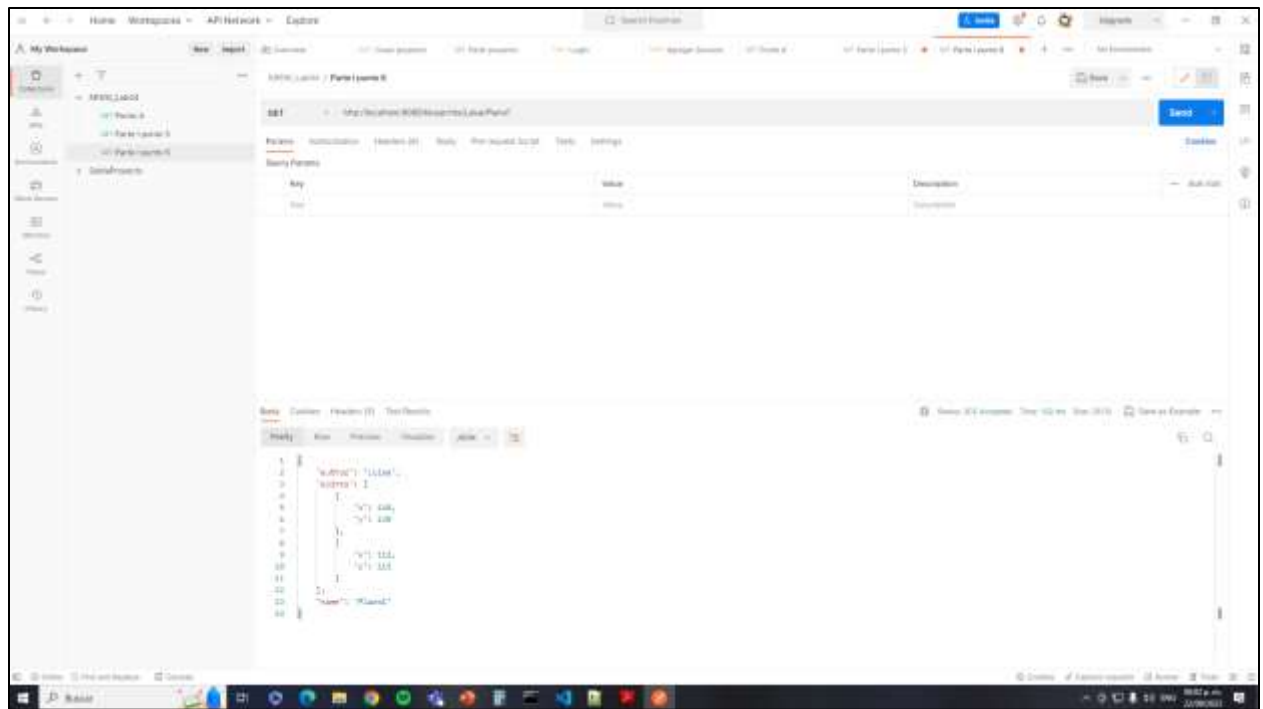


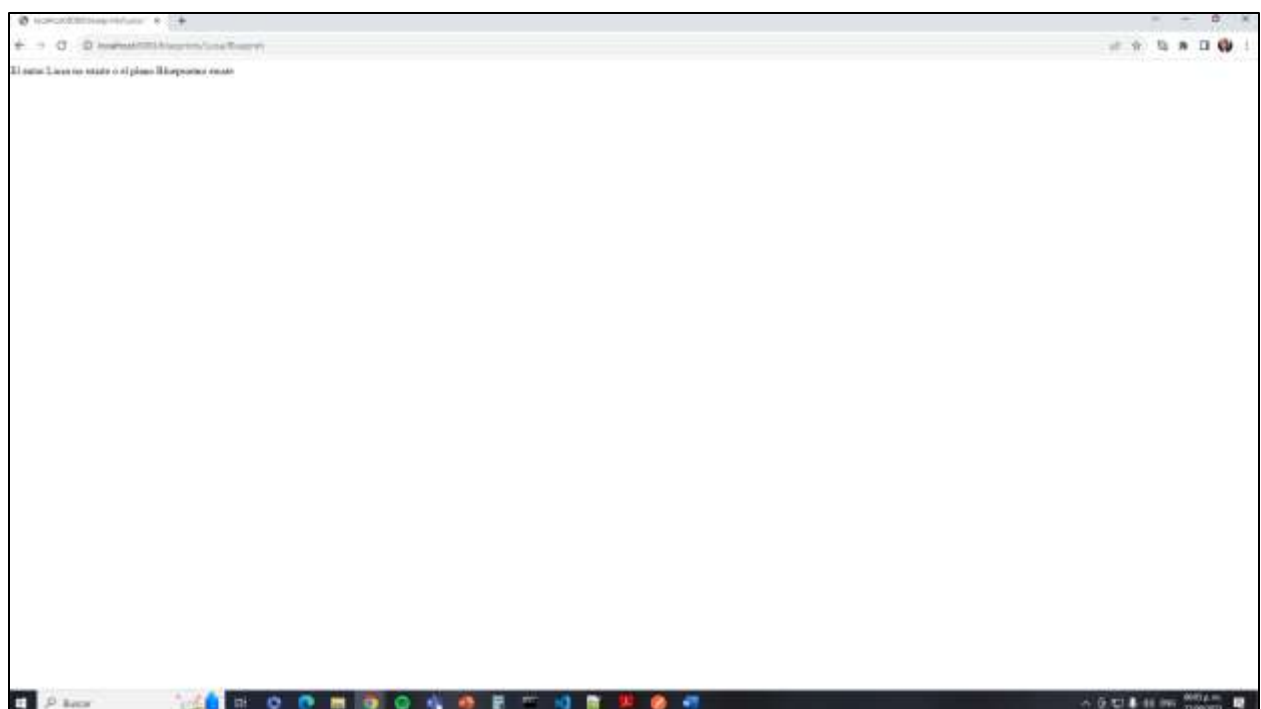
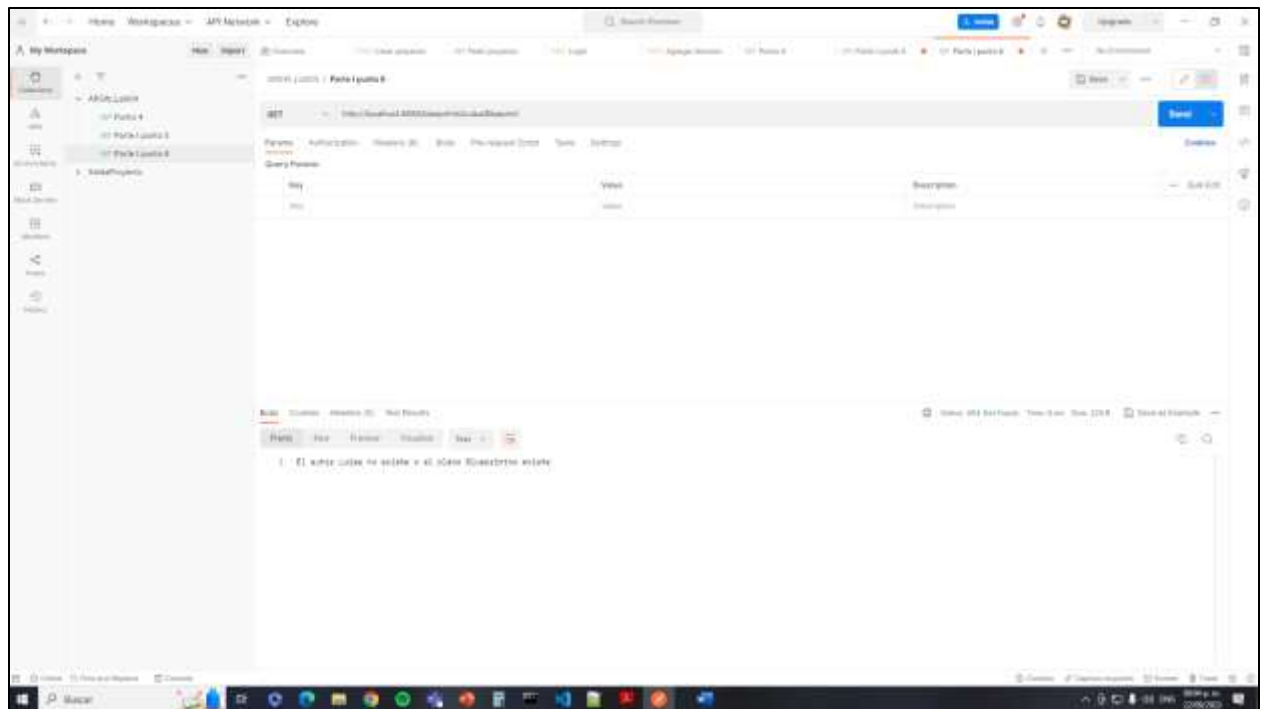
6. Modifique el controlador para que ahora, acepte peticiones GET al recurso `/blueprints/{author}/{bpname}`, el cual retorne usando una representación JSON sólo UN plano, en este caso el realizado por `{author}` y cuyo nombre sea `{bpname}`. De nuevo, si no existe dicho autor, se debe responder con el código de error HTTP 404.

## BlueprintApiController



```
1  package com.springboot.controllers;
2
3  import org.springframework.http.HttpStatus;
4  import org.springframework.web.bind.annotation.*;
5
6  @RequestMapping(value = "/blueprints", method = RequestMethod.GET)
7  public class BlueprintApiController {
8
9      // Endpoint para obtener un blueprint por autor
10     @RequestMapping(value = "/{author}", method = RequestMethod.GET)
11     public ResponseEntity<Object> getBlueprint(@PathVariable String author) {
12         try {
13             boolean autorExiste = lps.getBlueprint(author).isEmpty();
14             if (autorExiste) {
15                 return new ResponseEntity<>("El autor " + author + " no existe", HttpStatus.NOT_FOUND);
16             } else {
17                 // Devolver datos que se relacionan a través del API
18                 return new ResponseEntity<Object>(lps.getBlueprint(author), HttpStatus.ACCEPTED);
19             }
20         } catch (RuntimeException e) {
21             logger.getLogger(BlueprintApiController.class.getName()).log(Level.SEVERE, null, e);
22             return new ResponseEntity<Object>("Error al traer el autor", HttpStatus.NOT_FOUND);
23         }
24     }
25
26     // Endpoint para obtener un blueprint por autor y nombre
27     @RequestMapping(value = "/{author}/{bpname}", method = RequestMethod.GET)
28     public ResponseEntity<Object> getBlueprint(@PathVariable String author, @PathVariable String bpname) {
29         try {
30             boolean autorExiste = lps.getBlueprint(author, bpname) != null;
31             if (autorExiste) {
32                 return new ResponseEntity<Object>("El autor " + author + " no existe o el plano " + bpname + " no existe", HttpStatus.NOT_FOUND);
33             } else {
34                 // Devolver datos que se relacionan a través del API
35                 return new ResponseEntity<Object>(lps.getBlueprint(author, bpname), HttpStatus.ACCEPTED);
36             }
37         } catch (RuntimeException e) {
38             logger.getLogger(BlueprintApiController.class.getName()).log(Level.SEVERE, null, e);
39             return new ResponseEntity<Object>("Error al traer el autor", HttpStatus.NOT_FOUND);
40         }
41     }
42 }
```



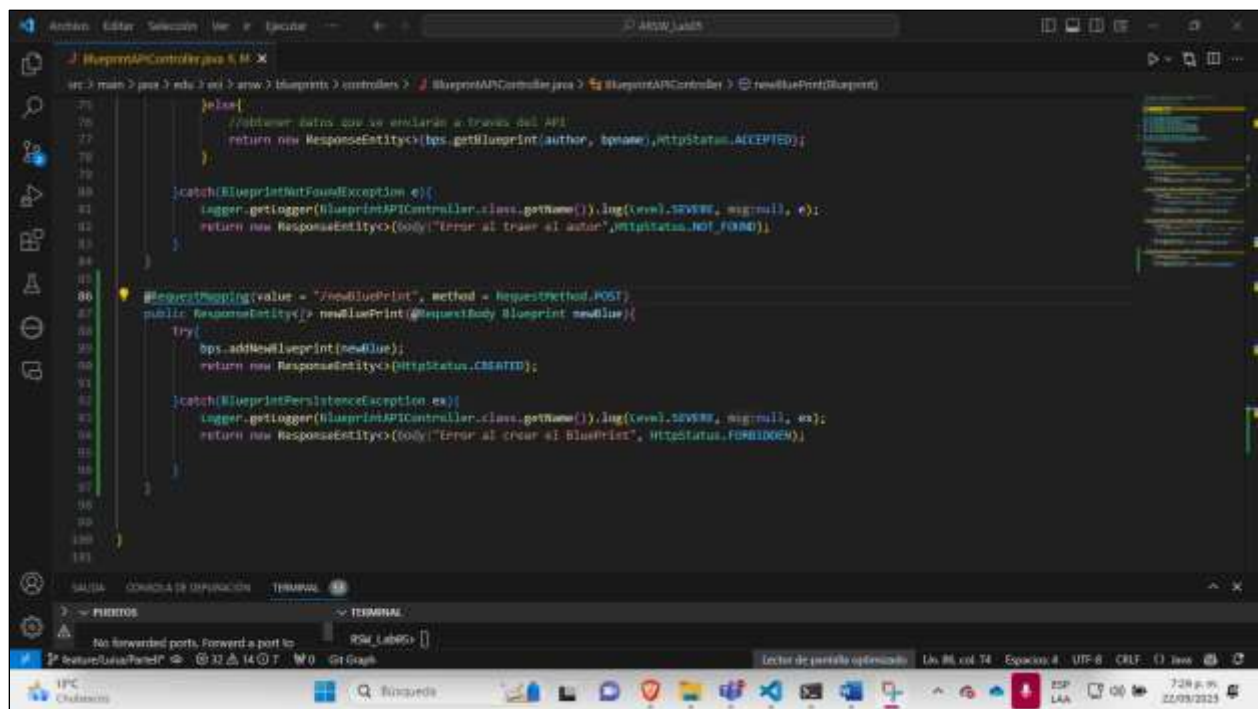


## Parte II

1. Agregue el manejo de peticiones POST (creación de nuevos planos), de manera que un cliente http pueda registrar una nueva orden haciendo una petición POST al recurso 'planos', y enviando como contenido de la petición todo el detalle de dicho recurso a través de un documento jSON. Para esto, tenga en cuenta el siguiente ejemplo, que considera -por consistencia con el protocolo HTTP- el manejo de códigos de estados HTTP (en caso de éxito o error):

```
@RequestMapping(method = RequestMethod.POST)
public ResponseEntity<?> manejadorPostRecursoXX(@RequestBody TipoXX o){
    try {
        //registrar dato
        return new ResponseEntity<>(HttpStatus.CREATED);
    } catch (XXException ex) {
        Logger.getLogger(XXController.class.getName()).log(Level.SEVERE, null, ex);
        return new ResponseEntity<>("Error bla bla bla",HttpStatus.FORBIDDEN);
    }
}
```

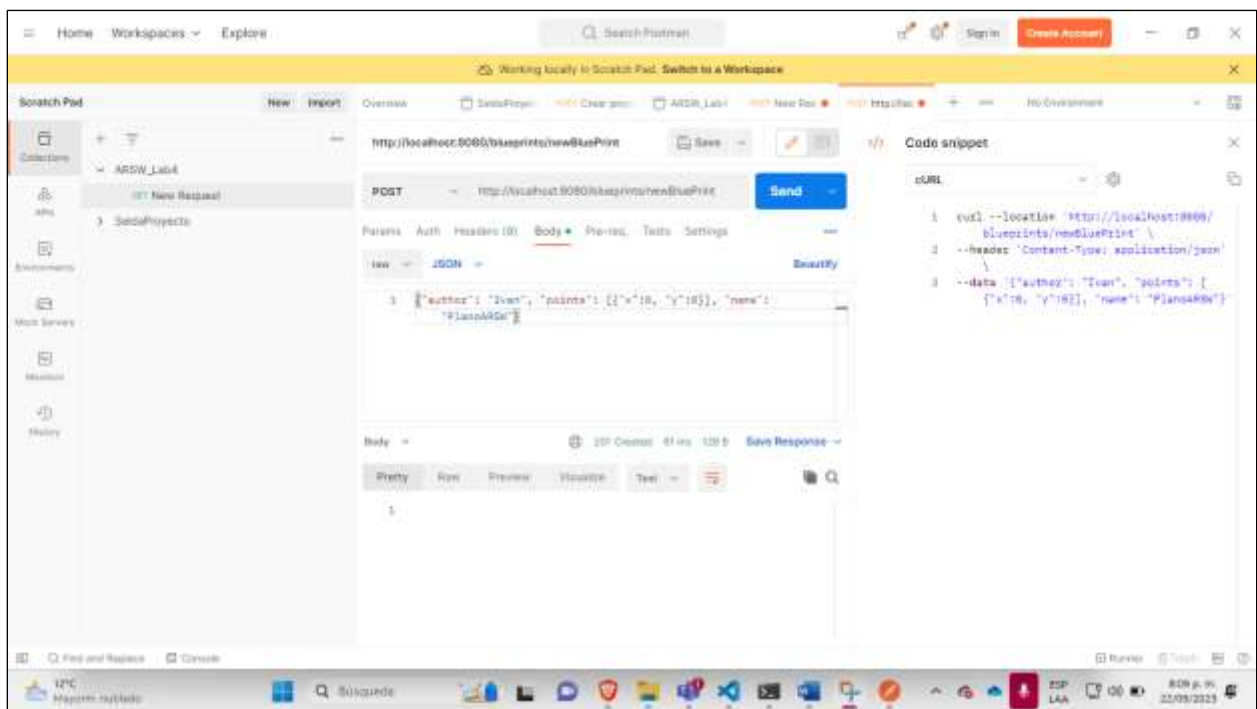
BlueprintAPIController



- Para probar que el recurso 'planos' acepta e interpreta correctamente las peticiones POST, use el comando curl de Unix. Este comando tiene como parámetro el tipo de contenido manejado (en este caso JSON), y el 'cuerpo del mensaje' que irá con la petición, lo cual en este caso debe ser un documento JSON equivalente a la clase Cliente (donde en lugar de {ObjetoJSON}, se usará un objeto JSON correspondiente a una nueva orden:

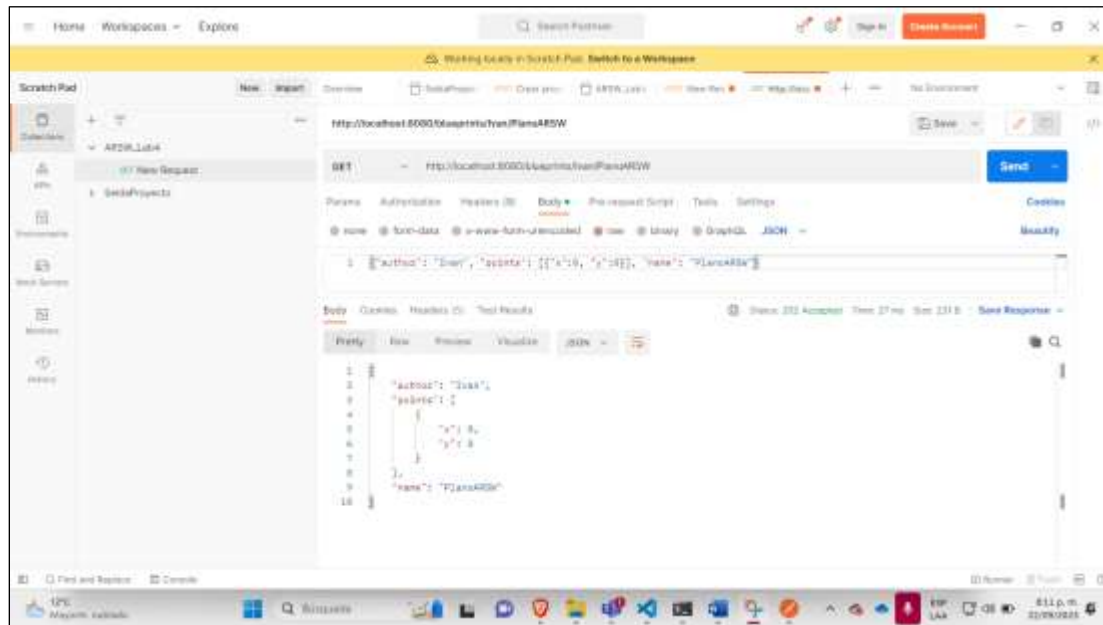
```
$ curl -i -X POST -HContent-Type:application/json -HAccept:application/json  
http://URL_del_recurso_ordenes -d '{ObjetoJSON}'
```

Con lo anterior, registre un nuevo plano (para 'diseñar' un objeto JSON, puede usar [esta herramienta](#)):



Nota: puede basarse en el formato JSON mostrado en el navegador al consultar una orden con el método GET.

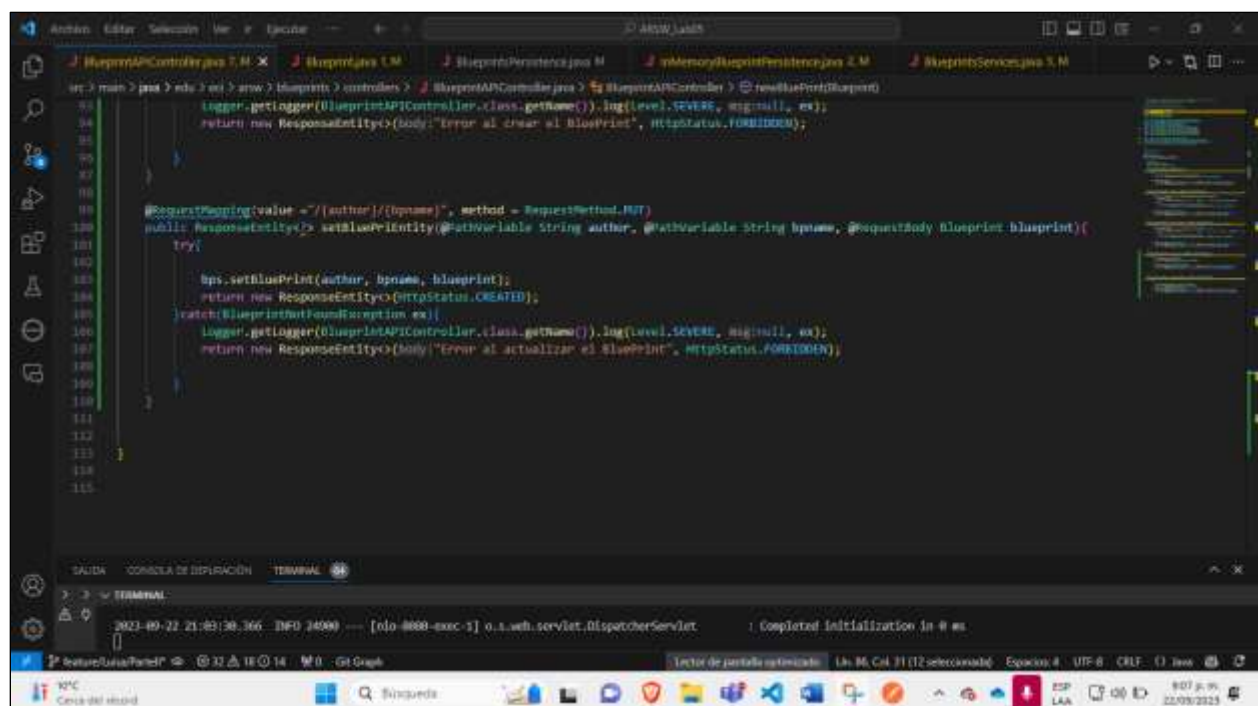
3. Teniendo en cuenta el autor y nombre del plano registrado, verifique que el mismo se pueda obtener mediante una petición GET al recurso '/blueprints/{author}/{bpname}' correspondiente.



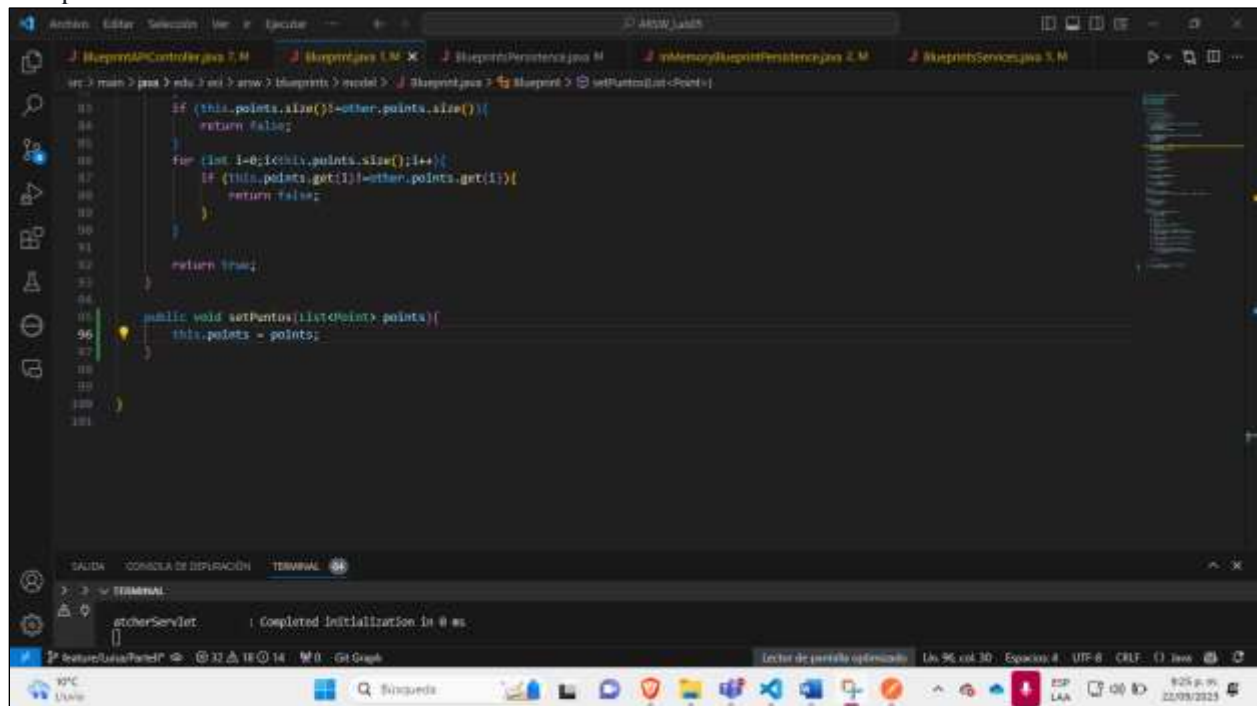
4. Agregue soporte al verbo PUT para los recursos de la forma '/blueprints/{author}/{bpname}', de manera que sea posible actualizar un plano determinado.

Código de las clases que se modificaron para que funcionara el PUT

BlueprintApiController



## Blueprint

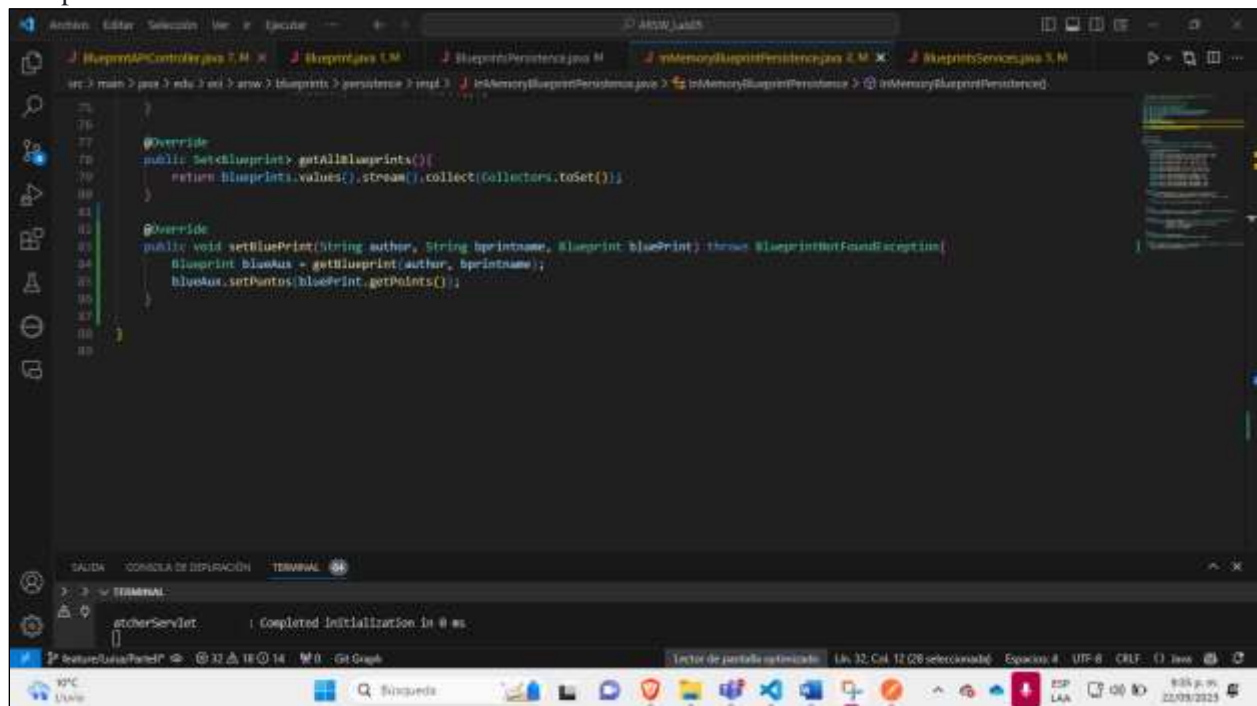


```
src > main > java > edu > uoi > arsw > blueprints > model > Blueprint.java > Blueprint > setPoints(List<Point>)
```

```
83     if (this.points.size() != other.points.size()) {
84         return false;
85     }
86     for (int i = 0; i < this.points.size(); i++) {
87         if (this.points.get(i) != other.points.get(i)) {
88             return false;
89         }
90     }
91     return true;
92 }
93
94 public void setPoints(List<Point> points) {
95     this.points = points;
96 }
97
98 }
```

```
gatoServerlet : Completed initialization in 0 ms.
```

## BlueprintsPersistence



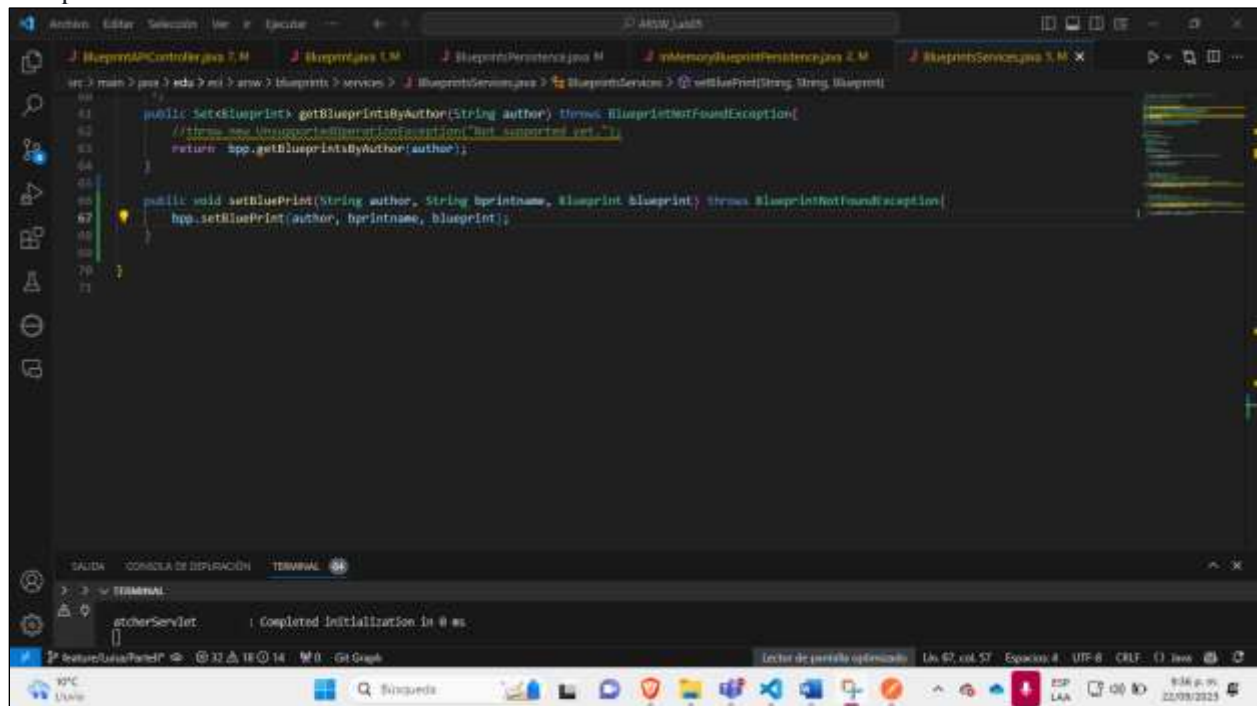
```
src > main > java > edu > uoi > arsw > blueprints > persistence > impl > InMemoryBlueprintPersistence.java > InMemoryBlueprintPersistence > InMemoryBlueprintPersistence()
```

```
76 }
77
78 @Override
79 public Set<Blueprint> getAllBlueprints() {
80     return blueprints.values().stream().collect(Collectors.toSet());
81 }
82
83 @Override
84 public void setBlueprint(String author, String blueprintname, Blueprint blueprint) throws BlueprintNotFoundException {
85     Blueprint blueprintx = getBlueprint(author, blueprintname);
86     blueprintx.setPoints(blueprint.getPoints());
87 }
88
89 }
```

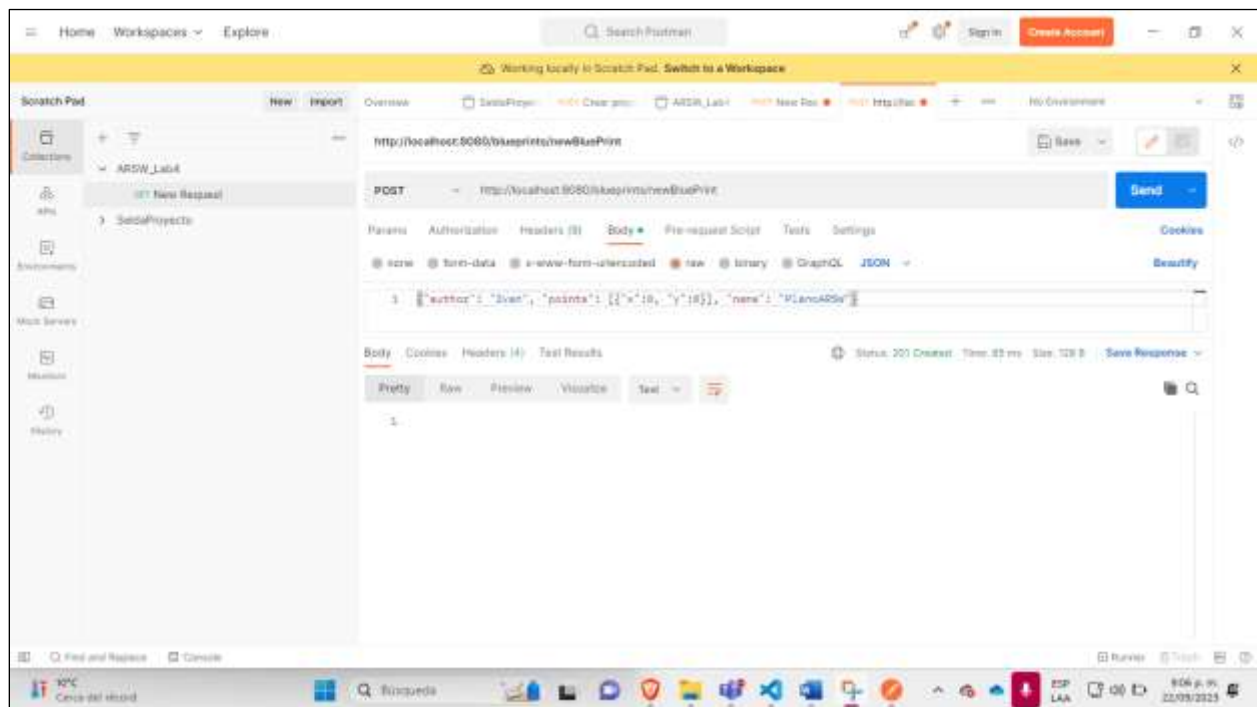
```
gatoServerlet : Completed initialization in 0 ms.
```



## BlueprintsServices

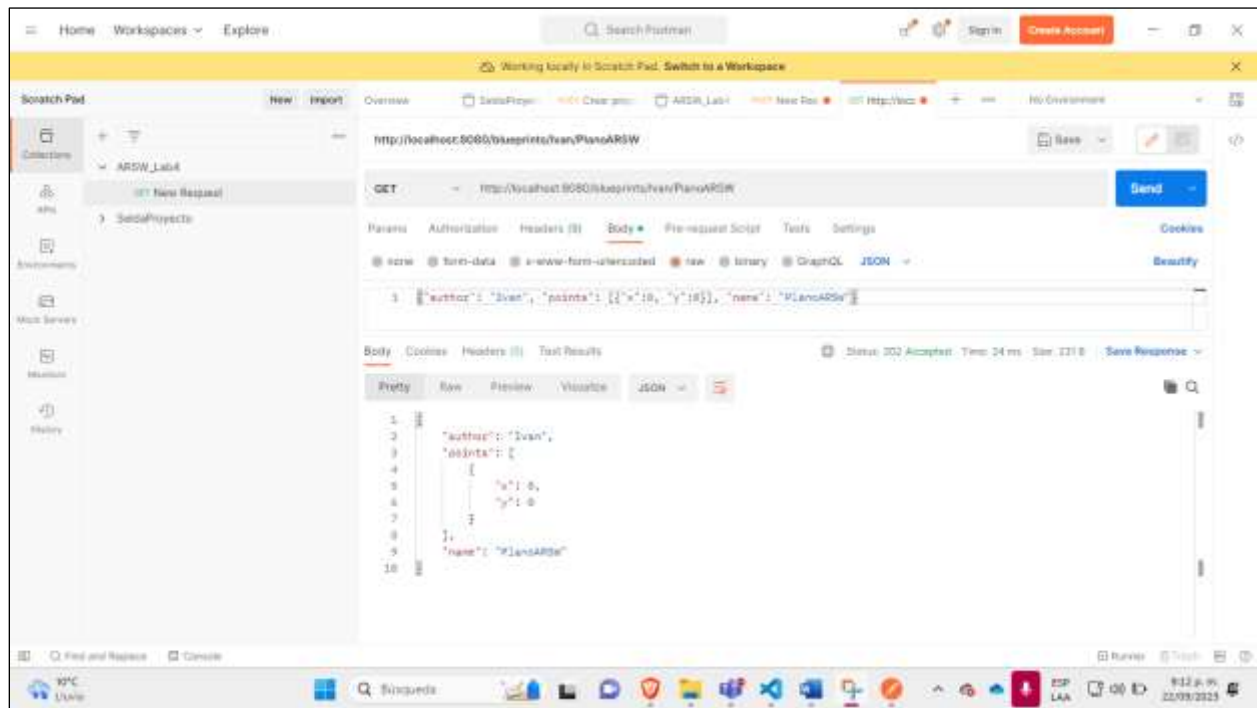


Primero hicimos un POST

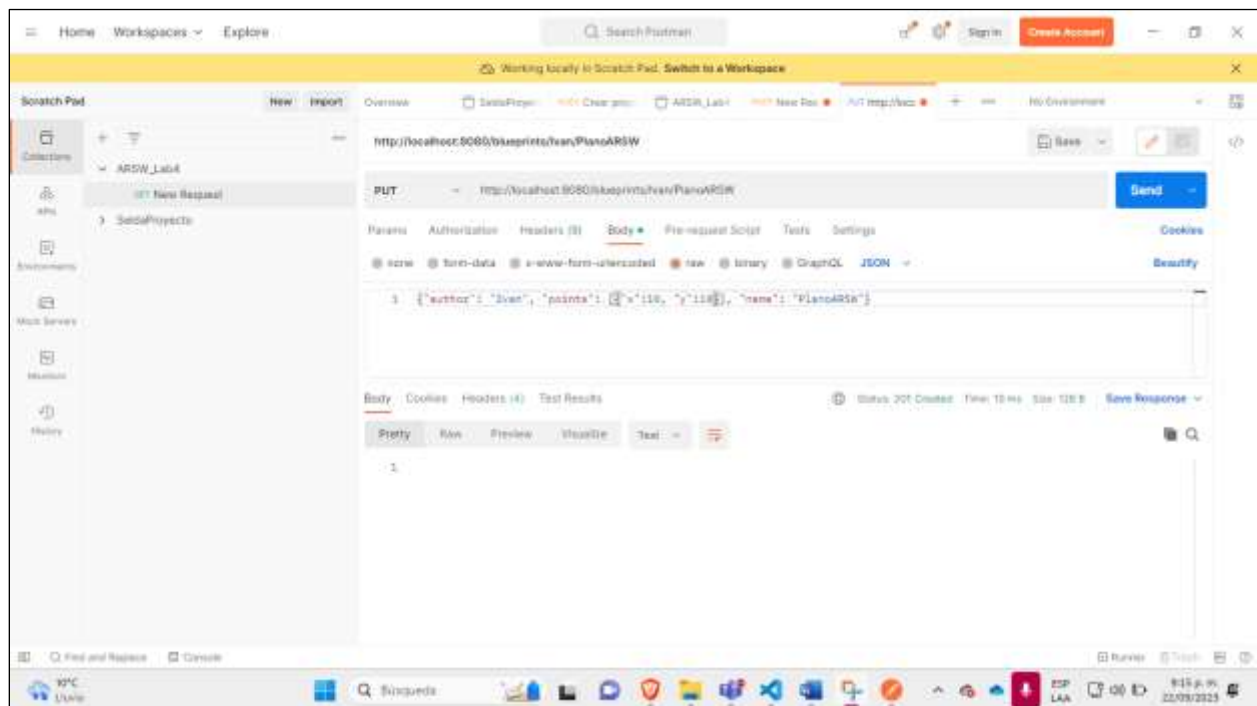




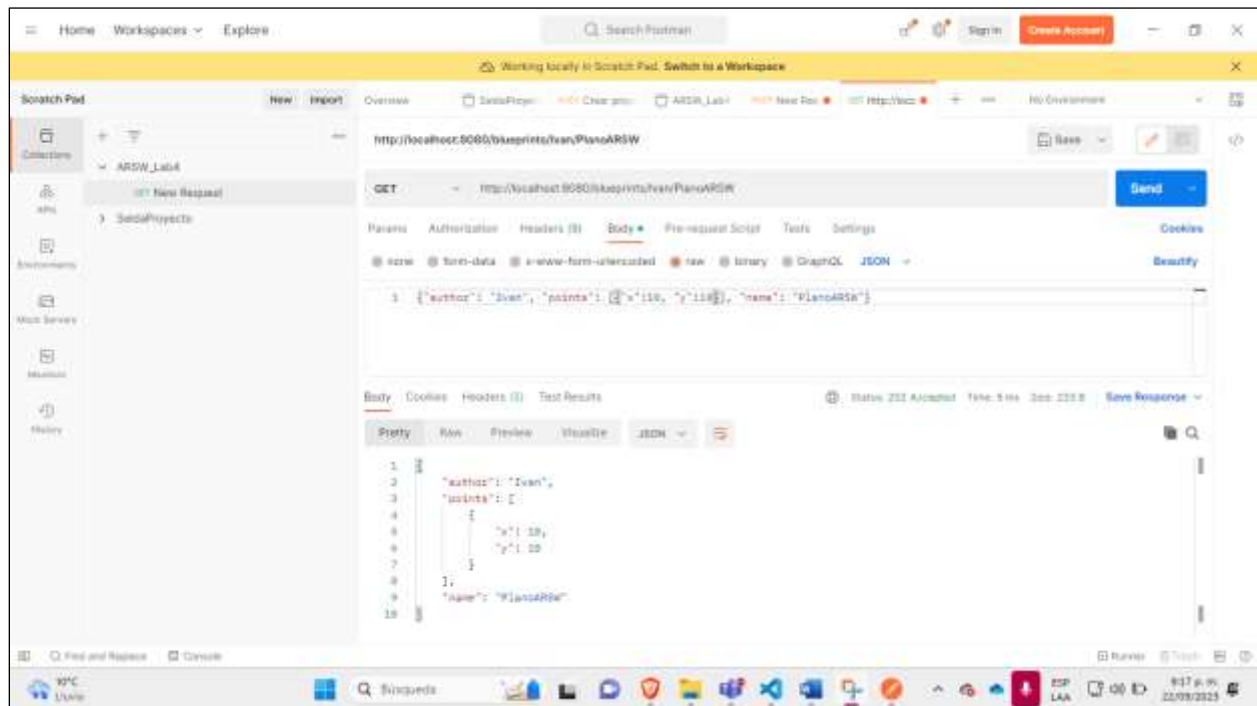
Consultamos con un GET



Modificamos con PUT



Consultamos con GET que los datos hayan modificado



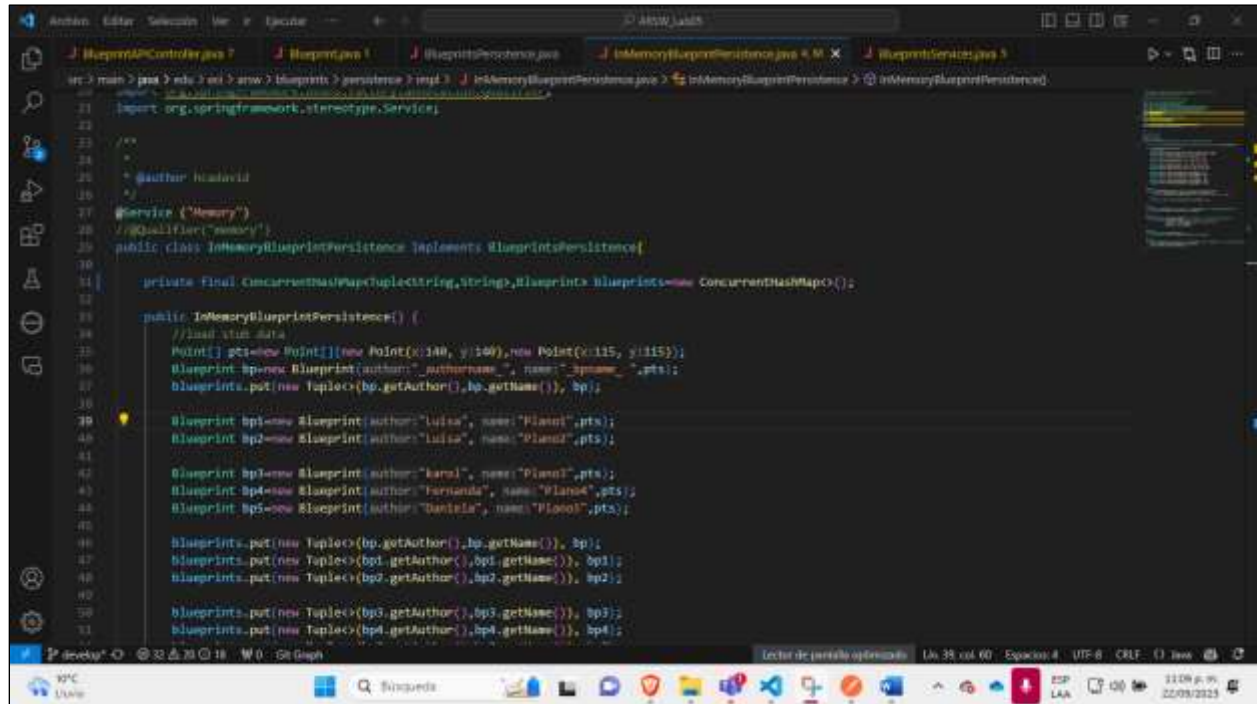
### Parte III

El componente BlueprintsRESTAPI funcionará en un entorno concurrente. Es decir, atenderá múltiples peticiones simultáneamente (con el stack de aplicaciones usado, dichas peticiones se atenderán por defecto a través múltiples de hilos). Dado lo anterior, debe hacer una revisión de su API (una vez funcione), e identificar:

- ¿Qué condiciones de carrera se podrían presentar?  
Cuando un usuario intente insertar, actualizar o consultar los Blueprints y al mismo tiempo otros usuarios estén tratando de realizar alguna de las tres acciones mencionadas anteriormente la información se vería afectada, ya que se puede dar el caso de que esta no sea la correcta en tiempo real.
- ¿Cuáles son las respectivas regiones críticas?  
En este caso las regiones críticas se presentarían cuando se añaden y eliminan Blueprints

Ajuste el código para suprimir las condiciones de carrera. Tengan en cuenta que simplemente sincronizar el acceso a las operaciones de persistencia/consulta DEGRADARÁ SIGNIFICATIVAMENTE el desempeño de API, por lo cual se deben buscar estrategias alternativas.

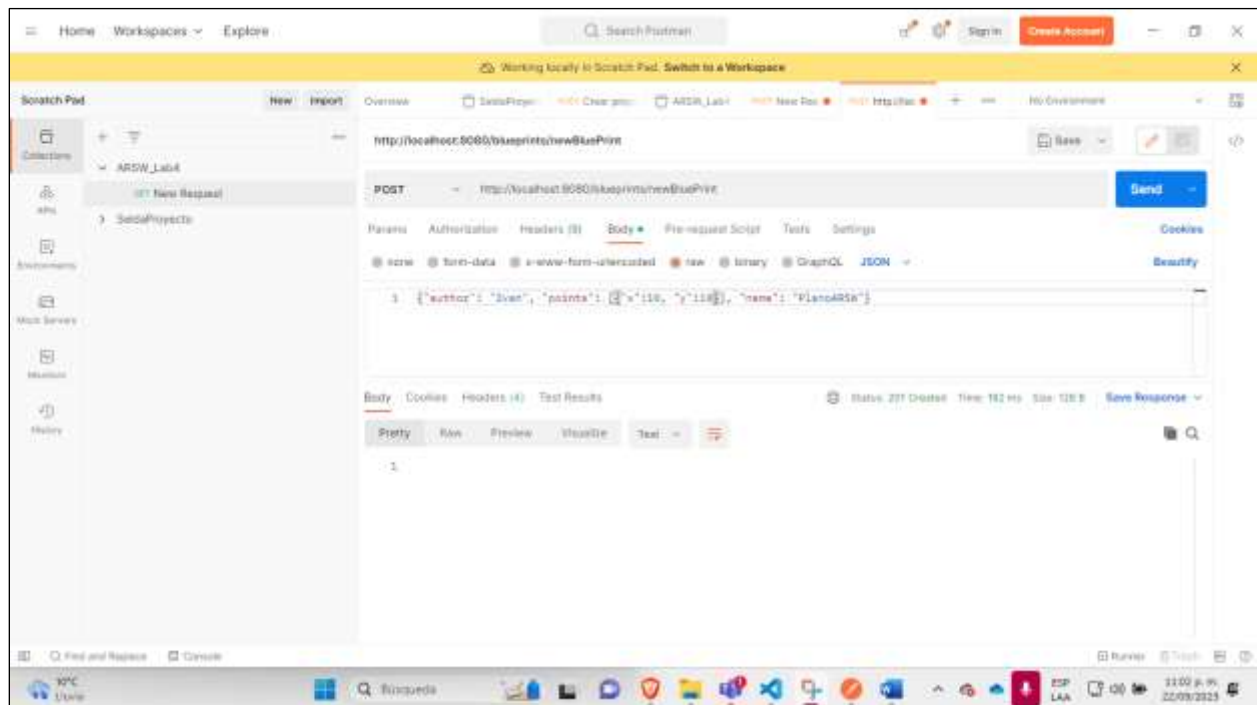
## Código (InMemoryBlueprintPersistence)



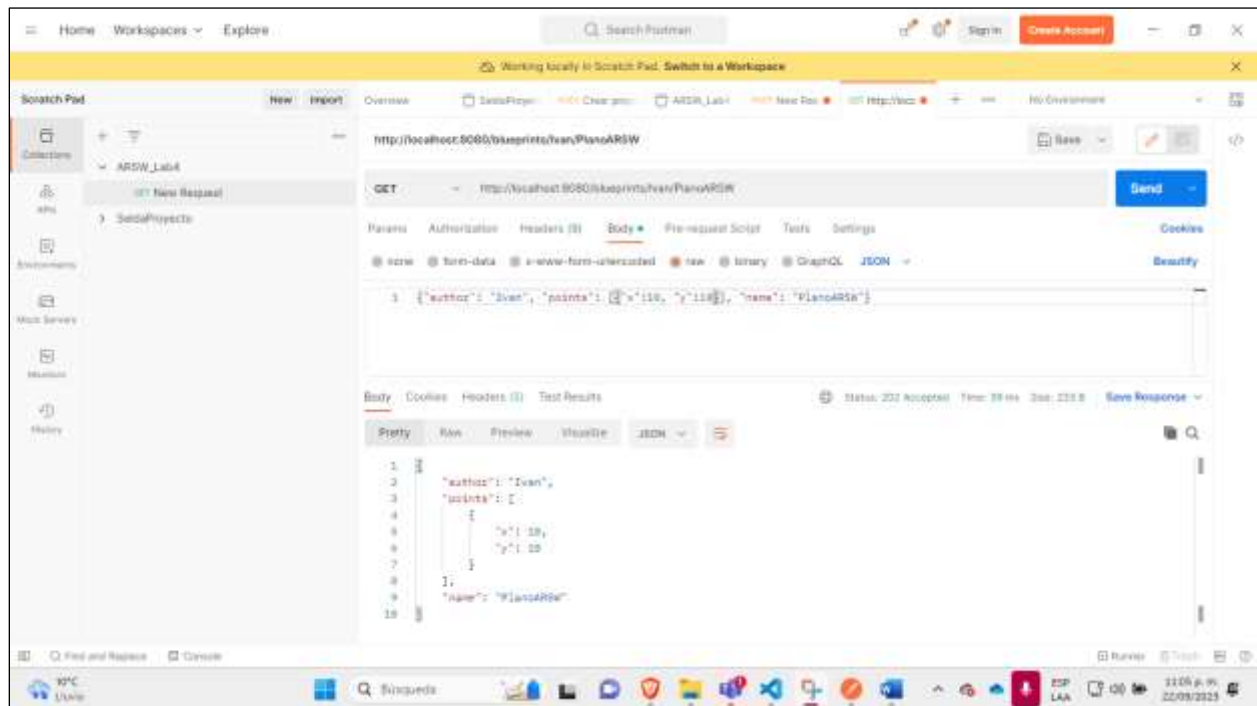
```
1 import org.springframework.stereotype.Service;
2
3 /**
4  *
5  * @author hcardia
6  */
7 @Service("Memory")
8 // @Qualifier("memory")
9 public class InMemoryBlueprintPersistence implements BlueprintPersistence {
10
11     private final ConcurrentHashMap<String, String> blueprints = new ConcurrentHashMap<>();
12
13     public InMemoryBlueprintPersistence() {
14         //load xtin data
15         Point[] pts = new Point[] { new Point(x:100, y:100), new Point(x:115, y:115) };
16         Blueprint bp1 = new Blueprint(author:"_author_", name:"_pname_", pts);
17         blueprints.put(new Tuple<>(bp.getAuthor(), bp.getName()), bp);
18
19         Blueprint bp2 = new Blueprint(author:"Luiza", name:"Plano1", pts);
20         Blueprint bp3 = new Blueprint(author:"Luiza", name:"Plano2", pts);
21
22         Blueprint bp4 = new Blueprint(author:"karsi", name:"Plano1", pts);
23         Blueprint bp5 = new Blueprint(author:"fernanda", name:"Plano2", pts);
24         Blueprint bp6 = new Blueprint(author:"Daniela", name:"Plano3", pts);
25
26         blueprints.put(new Tuple<>(bp.getAuthor(), bp.getName()), bp);
27         blueprints.put(new Tuple<>(bp1.getAuthor(), bp1.getName()), bp1);
28         blueprints.put(new Tuple<>(bp2.getAuthor(), bp2.getName()), bp2);
29
30         blueprints.put(new Tuple<>(bp3.getAuthor(), bp3.getName()), bp3);
31         blueprints.put(new Tuple<>(bp4.getAuthor(), bp4.getName()), bp4);
32     }
33 }
```

Probamos que la aplicación sigue funcionando después de las modificaciones del código:

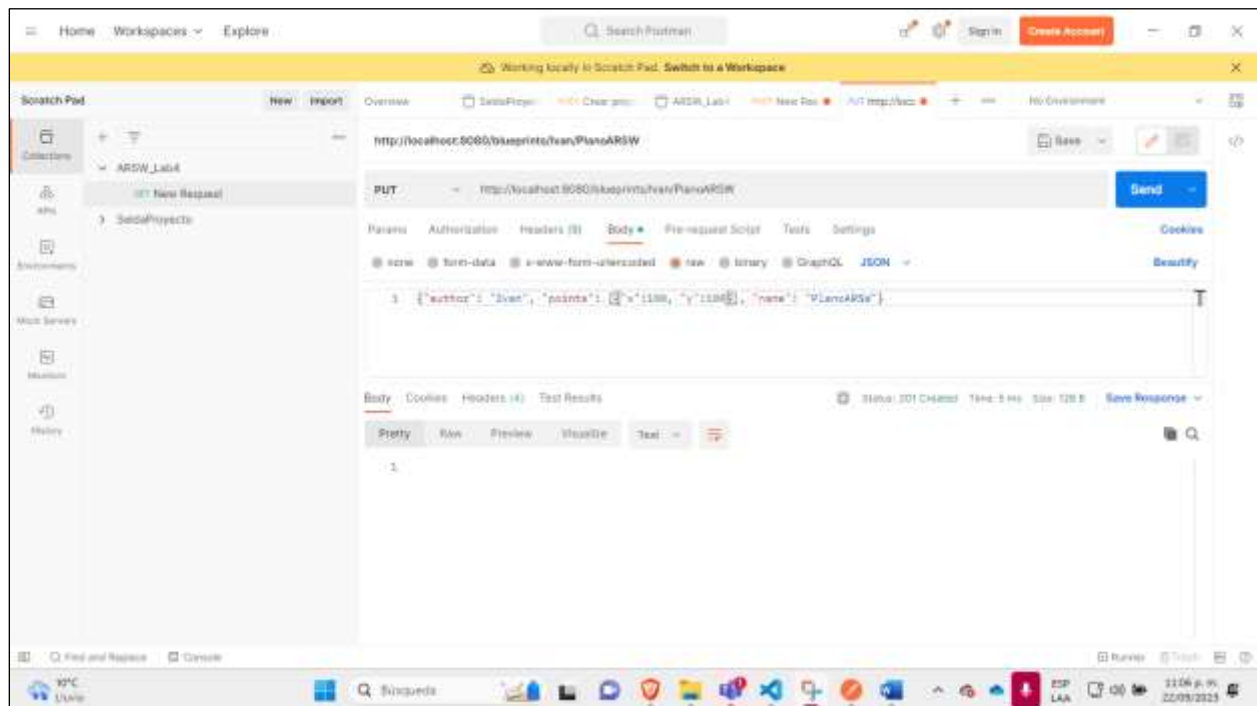
## POST



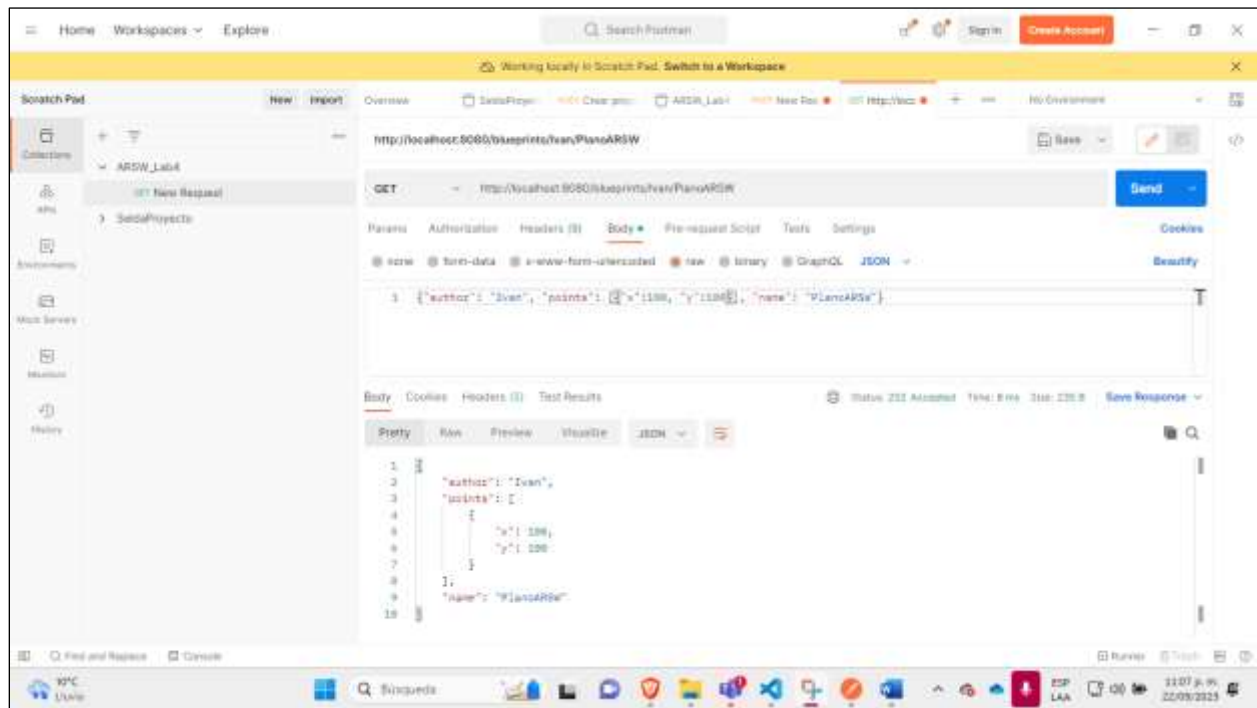
GET



PUT



GET para comprobar que el PUT SI Funciono



**Respuesta:** Para esto se modificó en la clase InMemoryBlueprintPersistence el tipo del atributo por ConcurrentHashMap ya que este nos permite realizar una concurrencia total sobre los accesos y una alta concurrencia para las actualizaciones

### III. Conclusiones

- En este laboratorio, se desarrolló una API REST llamada BlueprintsRESTAPI para gestionar planos arquitectónicos. Esta API se basa en la arquitectura de servicios web REST, utilizando Spring Boot y Spring MVC para crear un sistema de gestión centralizado de planos arquitectónicos.
- Se configuró la API para ofrecer el recurso "/blueprints" que, al recibir una petición GET, devuelve en formato JSON la lista de todos los planos existentes, aplicando el filtrado de puntos correspondiente.
- Se agregó soporte para peticiones POST que permiten registrar nuevos planos a través del recurso 'planos', utilizando JSON para enviar los detalles del plano en la solicitud HTTP.
- Se habilitó el verbo PUT para los recursos de la forma '/blueprints/{author}/{bpname}', lo que permite actualizar un plano específico mediante una solicitud HTTP PUT.
- Se reconoció la importancia de que la API funcione en entornos concurrentes y se identificaron posibles condiciones de carrera.