

## **Laboratorio 5**

**Sebastián Cardona, Laura Gil, Zayra Gutiérrez**

**Ingeniero de Sistemas**

**Javier Toquica Barrera**

**Universidad Escuela Colombiana de ingeniería Julio Garavito**

**2025-1**

# Contenido

<b>Introducción .....</b>	<b>3</b>
<b>Desarrollo del Laboratorio .....</b>	<b>4</b>
<b>    Parte II.....</b>	<b>4</b>
<b>    Parte III .....</b>	<b>6</b>
<b>    Documentación con SWAGGER.....</b>	<b>8</b>
<b>Conclusiones .....</b>	<b>13</b>

# Introducción

En este laboratorio, se procedió a implementar una API REST para la gestión de los planos utilizando el recurso Spring Boot. Se incorporaron las operaciones de registrar, modificar y consultar los planos a través de las peticiones HTTP con los métodos POST, GET y PUT. Se introdujeron elementos de concurrencia para garantizar la integridad de los datos en un entorno multiusuario. Se detectaron situaciones de carrera y se introdujeron mecanismos para evitar inconsistencias como el uso de ConcurrentHashMap y de métodos atómicos. Se documentó finalmente la API utilizando Swagger, con el fin de poder probarla o integrarla con otras aplicaciones.

Aquí podrás ver la implementación en el siguiente enlace:

<https://github.com/ZayraGS1403/ARSW-Lab05.git>

# Desarrollo del Laboratorio

## 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):

```
/**
 * Registra un nuevo blueprint en el sistema.
 * @param bp Objeto Blueprint a registrar.
 * @return ResponseEntity con el estado de la operación.
 */
@PostMapping no usages  ZayraGS1403 +1
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 (BlueprintPersistenceException e) {
        response.put("error", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }
}
```

2. 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:

**Comando para ejecutar desde consola CMD:**

```
curl.exe -i -X POST -H "Content-Type: application/json" -H "Accept: application/json" -d '{"author":"Juan\\", "name":"NuevoPlano\\", "points":[{"x":50, "y":50}, {"x":60, "y":60}]}' http://localhost:8080/blueprints
```

```
C:\Users\laura\Desktop\ARWS\Laboratorios\ARSW-Lab05>curl.exe -i -X POST -H "Content-Type: application/json" -H "Accept: application/json" -d '{"author":"Juan","name":"NuevoPlano","points":[{"x":50,"y":50},{"x":60,"y":60}]}' http://localhost:8080/blueprints
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 04 Mar 2025 17:35:52 GMT

{"status":"success"}
C:\Users\laura\Desktop\ARWS\Laboratorios\ARSW-Lab05>
```

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.*

**Comando para ejecutar desde consola CMD:**

**curl.exe -i -X GET <http://localhost:8080/blueprints/Juan/NuevoPlano>**

```
C:\Users\laura\Desktop\ARWS\Laboratorios\ARSW-Lab05>curl.exe -i -X GET "http://localhost:8080/blueprints/Juan/NuevoPlano"
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 04 Mar 2025 17:38:02 GMT

{"author":"Juan","points":[{"x":50,"y":50}],"name":"NuevoPlano"}
C:\Users\laura\Desktop\ARWS\Laboratorios\ARSW-Lab05>
```

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

```
/**
 * Actualiza un blueprint existente.
 * @param author Nombre del autor.
 * @param name Nombre del blueprint.
 * @param blueprint Blueprint actualizado.
 * @return ResponseEntity con el blueprint actualizado o un error si no se encuentra.
 */
@PutMapping("/{author}/{name}") no usages SebastianCardona-P
public ResponseEntity<> updateBlueprint(@PathVariable("author") String author, @PathVariable("name") String name, @RequestBody Blueprint blueprint) {
    try {
        return ResponseEntity.ok(bps.updateBlueprint(author, name, blueprint));
    } 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.");
    }
}
```

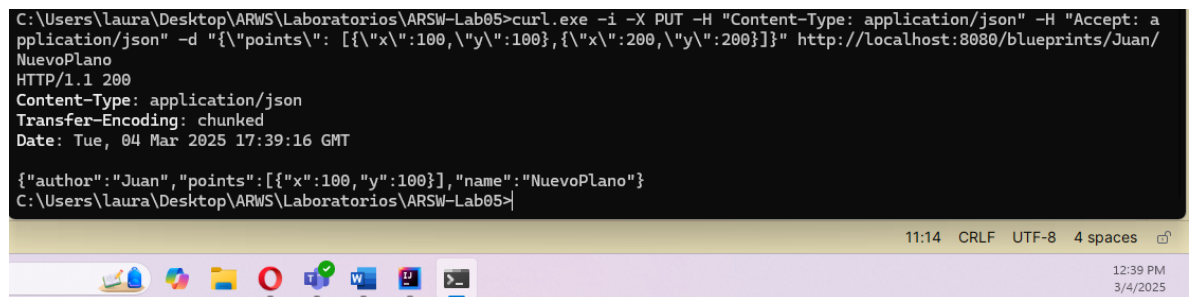
```
@Override 1 usage SebastianCardona-P
public Blueprint updateBlueprint(String author, String name, Blueprint blueprint) throws BlueprintNotFoundException {
    String authorTrimmed = author.trim();
    String nameTrimmed = name.trim();
    Tuple<String, String> key = new Tuple<>(authorTrimmed, nameTrimmed);

    blueprint.setAuthor(authorTrimmed);
    blueprint.setName(nameTrimmed);
    if (blueprints.replace(key, blueprint) == null) {
        throw new BlueprintNotFoundException("Blueprint not found: " + author + " - " + name);
    }
    return blueprint;
}
```

```
public Blueprint updateBlueprint(String author, String name, Blueprint blueprint) throws BlueprintNotFoundException {
    return filter.filterPlain(bpp.updateBlueprint(author, name, blueprint));
}
```

**Comando para ejecutar desde consola CMD:**

**curl.exe -i -X PUT -H "Content-Type: application/json" -H "Accept: application/json" -d '{"points": [{"x":100,"y":100}, {"x":200,"y":200}]}' http://localhost:8080/blueprints/Juan/NuevoPlano**



```
C:\Users\laura\Desktop\ARWS\Laboratorios\ARSW-Lab05>curl.exe -i -X PUT -H "Content-Type: application/json" -H "Accept: application/json" -d '{"points": [{"x":100,"y":100}, {"x":200,"y":200}]}' http://localhost:8080/blueprints/Juan/NuevoPlano
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 04 Mar 2025 17:39:16 GMT

{"author": "Juan", "points": [{"x":100,"y":100}], "name": "NuevoPlano"}
C:\Users\laura\Desktop\ARWS\Laboratorios\ARSW-Lab05>
```

## 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?*

Las condiciones de carrera pueden producirse cuando simultáneamente varios hilos

acceden y modifican la estructura de datos compartida sin la debida sincronización.

En este caso, el mapa de blueprints podría estar sujeto a inconsistencias en los escenarios siguientes:

- La adición simultánea de un mismo blueprint: dos hilos podrían añadir el mismo blueprint al mismo tiempo.
  - La modificación y la lectura concurrente: un hilo podría leer un blueprint mientras que otro lo modifica, dando lugar a datos inconsistentes.
  - La eliminación y la lectura simultáneas: un hilo podría eliminar un blueprint mientras este es consultado por otro hilo, desencadenando errores.
- *¿Cuáles son las respectivas regiones críticas?*

Las regiones críticas donde pueden ocurrir condiciones de carrera son:

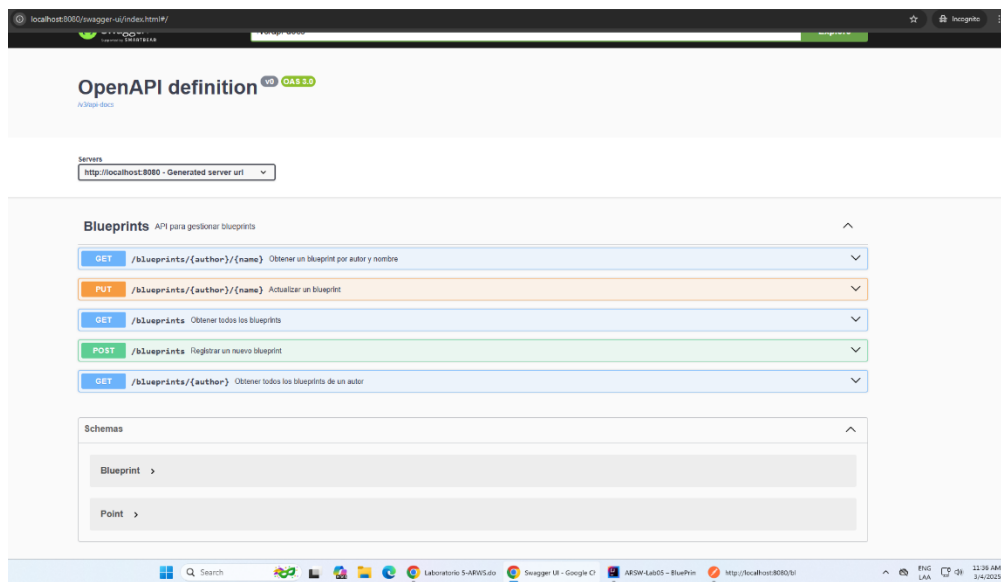
- Método saveBlueprint: Se necesita garantizar que no se sobrescriba un blueprint existente.
  - Método updateBlueprint: Se debe asegurar que la actualización solo ocurra si el blueprint ya existe.
  - Método getBlueprintsByAuthor y getAllBlueprints: Se requiere evitar la lectura de datos inconsistentes si hay modificaciones simultáneas.
- *Solución*

Para evitar estas condiciones de carrera sin afectar el rendimiento, se aplicaron las siguientes soluciones:

- Uso de ConcurrentHashMap: Maneja acceso concurrente sin bloqueos pesados.
- putIfAbsent en saveBlueprint: Garantiza que la inserción sea atómica y evita sobrescrituras.

- replace en updateBlueprint: Asegura que un blueprint solo se actualice si ya existe.
- Copia segura en operaciones de lectura: getAllBlueprints() y getBlueprintsByAuthor() devuelven copias de los datos para evitar inconsistencias.

## Documentación con SWAGGER



1. Obtener un BluePrint por autor y nombre



GET /blueprints/{author}/{name} Obtener un blueprint por autor y nombre

Devuelve un blueprint basado en su autor y nombre.

Parameters

Name	Description
author * required string (path)	Marcos
name * required string (path)	EdificioC

Execute
Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/blueprints/marcos/edificioC' \
  -H 'accept: */*'

```

Request URL

```
http://localhost:8080/blueprints/marcos/edificioC

```

Server response

Code Details

200

Response body

```
{
  "author": "marcos",
  "points": [
    {
      "x": 0,
      "y": 100
    },
    {
      "x": 100,
      "y": 100
    }
  ],
  "name": "edificioC"
}

```

## 2. Actualizar un Blueprint

PUT /blueprints/{author}/{name} Actualizar un blueprint

Modifica los datos de un blueprint existente.

Parameters

Name	Description
author * required string (path)	Marcos
name * required string (path)	EdificioC

Request body required
application/json

```
{
  "author": "Marcos",
  "points": [
    {
      "x": 0,
      "y": 22
    }
  ],
  "name": "edificioC"
}

```

Execute
Clear

Execute

Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:8080/blueprints/Marcos/Edificio' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "author": "Marcos",
    "points": [
      {
        "x": 0,
        "y": 22
      },
      {
        "x": 100,
        "y": 100
      }
    ],
    "name": "EdificioC"
  }'
```

Request URL

http://localhost:8080/blueprints/Marcos/EdificioC

Server response

Code

Details

200

Response body

```
{
  "author": "Marcos",
  "points": [
    {
      "x": 0,
      "y": 22
    }
  ],
  "name": "EdificioC"
}
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Tue, 04 Mar 2025 16:35:26 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	No links

Media type

\*/\*

### 3. Obtener todos los Blueprints

GET

/blueprints

Obtener todos los blueprints

Devuelve una lista de todos los blueprints registrados en el sistema.

Parameters

No parameters

Cancel

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/blueprints' \
  -H 'accept: */*' \
```

Request URL

http://localhost:8080/blueprints

Server response

Code

Details

200

Response body

```
{
  "author": "Juan",
  "points": [
    {
      "x": 100,
      "y": 100
    }
  ],
  "name": "EdificioA"
},
{
  "author": "Juan",
  "points": [
    {
      "x": 10,
      "y": 24
    },
    {
      "x": 100,
      "y": 100
    }
  ],
  "name": "EdificioB"
}
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Tue, 04 Mar 2025 16:39:33 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

POST http://localhost:8080/blueprints

Server response

Code Details

200

Response body

```
{
  "name": "EdificioA",
  "author": "Juan",
  "points": [
    {
      "x": 10,
      "y": 24
    },
    {
      "x": 100,
      "y": 100
    }
  ]
},
{
  "name": "EdificioB",
  "author": "Marcos",
  "points": [
    {
      "x": 0,
      "y": 22
    }
  ]
},
{
  "name": "EdificioC"
}
]
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Tue, 04 Mar 2025 16:19:33 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	No links

Media type: \*/\*

Controls Accept header: Example Value | Schema

```
{
  "author": "string",
  "points": [
    {
      "x": 0,
      "y": 0
    }
  ]
},
{
  "name": "string"
}
]
```

#### 4. Registrar un nuevo Blueprint

POST /blueprints Registrar un nuevo blueprint

Añade un nuevo blueprint al sistema.

Parameters

No parameters

Request body **required** application/json

```
{
  "author": "Felipe",
  "points": [
    {
      "x": 70,
      "y": 248
    },
    {
      "x": 100,
      "y": 100
    }
  ]
},
{
  "name": "EdificioPP"
}
]
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/blueprints' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "author": "Felipe",
    "points": [
      {
        "x": 70,
        "y": 248
      },
      {
        "x": 100,
        "y": 100
      }
    ]
  },
  {
    "name": "EdificioPP"
  }
]'
```

Request URL

JSON input:

```
{
  "y": 100
},
{
  "name": "EdificioPP"
}
}
```

Buttons: [Execute](#) [Clear](#)

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/blueprints' \
  -H 'Accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "author": "Felipe",
    "points": [
      {
        "x": 70,
        "y": 248
      },
      {
        "x": 100,
        "y": 100
      }
    ],
    "name": "EdificioPP"
  }'
```

Request URL: <http://localhost:8080/blueprints>

Server response

Code	Details
200	<p>Response body</p> <pre>{   "status": "success" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Tue, 04 Mar 2025 16:41:28 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	OK	No links

Media type:

Controls: [Accept header](#)

## 5. Obtener todos los Blueprints de un autor

GET /blueprints/{author} Obtener todos los blueprints de un autor

Devuelve una lista de blueprints pertenecientes a un autor específico.

Parameters

Name	Description
author * required	
string (path)	<input type="text" value="Felipe"/>

Buttons: [Execute](#) [Clear](#)

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/blueprints/Felipe' \
  -H 'Accept: */*' \
  -H 'Content-Type: */*' \
  -d ''
```

Request URL: <http://localhost:8080/blueprints/Felipe>

Server response

Code	Details
200	<p>Response body</p> <pre>{   "author": "Felipe",   "points": [     {       "x": 70,       "y": 248     },     {       "x": 100,       "y": 100     }   ],   "name": "EdificioPP" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Tue, 04 Mar 2025 16:41:54 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	OK	No links

Media type:

# Conclusiones

- Se logró implementar una API REST funcional con soporte para creación, consulta y actualización de planos, asegurando la correcta manipulación de los datos.
- Se identificaron y solucionaron problemas de concurrencia mediante estructuras de datos seguras y operaciones atómicas, garantizando la consistencia en accesos simultáneos.
- La documentación con Swagger permitió mejorar la usabilidad y comprensión de la API, facilitando su integración y pruebas.
- Se comprendió la importancia de manejar correctamente las regiones críticas en aplicaciones concurrentes, asegurando un desempeño eficiente sin comprometer la integridad de los datos.