ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

UNIVERSIDAD

Laboratorio 5 - Blue Prints parte 2

Andrés Felipe Arias Ajiaco

Cesar David Amaya Gómez

Johan Sebastián Gracia Martínez

Sebastián David Blanco Rodríguez

Universidad Escuela Colombiana de ingeniería Julio Gravito

Arquitecturas de software

Ing. Javier Iván Toquica Barrera

1 de marzo de 2024

Introducción

Este laboratorio tiene como objetivo desarrollar un componente BlueprintsRESTAPI para gestionar los planos arquitectónicos de una compañía de diseño. El componente debe ofrecer un medio estandarizado e independiente de la plataforma para gestionar los planos de forma centralizada. El diseño del componente se basa en el uso de Spring (con SpringMVC y SpringBoot) para implementar el API REST.

La primera parte del laboratorio implica integrar los beans desarrollados previamente en el proyecto base, configurar la inyección de dependencias con las anotaciones @Service y @Autowired, y modificar el bean de persistencia para inicializarlo con planos por defecto. Además, se debe configurar el recurso /blueprints para que al hacer una petición GET retorne todos los planos en formato JSON.

En la segunda parte del laboratorio se agrega el manejo de peticiones POST para la creación de nuevos planos. Se debe permitir que un cliente HTTP registre una nueva orden haciendo una petición POST al recurso planos, enviando el detalle del recurso en formato JSON. Se debe implementar el manejo de códigos de estados HTTP para indicar el éxito o error de la operación.

Finalmente, en la tercera parte del laboratorio se aborda el funcionamiento concurrente del componente BlueprintsRESTAPI. Se debe identificar posibles condiciones de carrera y regiones críticas en el código existente, así como implementar soluciones para suprimir estas condiciones de carrera

En resumen, este laboratorio se enfoca en desarrollar un API REST para gestionar planos arquitectónicos, integrando diferentes componentes de Spring y garantizando un funcionamiento eficiente y concurrente del mismo.

Desarrollo del Laboratorio

- **Parte I**

i. 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.

ii. Modifique el bean de persistecia 'InMemoryBlueprintPersistence' para que por defecto se inicialice con al menos otros tres planos, y con dos asociados a un mismo

```
public InMemoryBlueprintPersistence() {

    Point[] pts=new Point[]{new Point( x: 140,  y: 140),new Point( x: 115,  y: 115)};
    Blueprint bp=new Blueprint( author: "_authorname_",  name: "_bpname_ ",pts);
    Blueprint bp1 = new Blueprint( author: "andres",  name: "andres_p1",pts);
    Blueprint bp2 = new Blueprint( author: "andres",  name: "andres_p2",pts);
    Blueprint bp3 = new Blueprint( author: "Sebastian",  name: "sebastian_p1",pts);
    blueprints.put(new Tuple<>(bp.getAuthor(),bp.getName()), bp);
    blueprints.put(new Tuple<>(bp1.getAuthor(),bp1.getName()), bp1);
    blueprints.put(new Tuple<>(bp2.getAuthor(),bp2.getName()), bp2);
    blueprints.put(new Tuple<>(bp3.getAuthor(),bp3.getName()), bp3);
}
```

En esta parte agregamos los tres mapas extras, en los que nos piden que al menos dos sean del mismo autor de esta manera al tener inyectado InMemoryBlueprintPersistence en BlueprintsSevices crea los planos.

iii. Configure su aplicación para que ofrezca el recurso "/blueprints", de manera que cuando se le haga una petición GET, retorne -en formato jSON- el conjunto de todos los planos. Para esto:

- Modifique la clase BlueprintAPIController teniendo en cuenta el siguiente ejemplo

de controlador REST hecho con SpringMVC/SpringBoot:

- Haga que en esta misma clase se inyecte el de tipo BlueprintServices

```java
@RestController
public class BlueprintAPIController {

    @Autowired
    BlueprintsServices blueprintsServices;

    new *
    @RequestMapping(value = "/blueprints",method = RequestMethod.GET)
    public ResponseEntity<?> getBluePrints() {
        Set<Blueprint> blueprintSet = null;

        try {
            blueprintSet = blueprintsServices.getAllBlueprints();
            return new ResponseEntity<>(blueprintSet, HttpStatus.ACCEPTED);
        } catch (Exception ex) {
            Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, msg: null, ex);
            return new ResponseEntity<>( body: "Error bla bla bla", HttpStatus.NOT_FOUND);
        }
    }
}
```

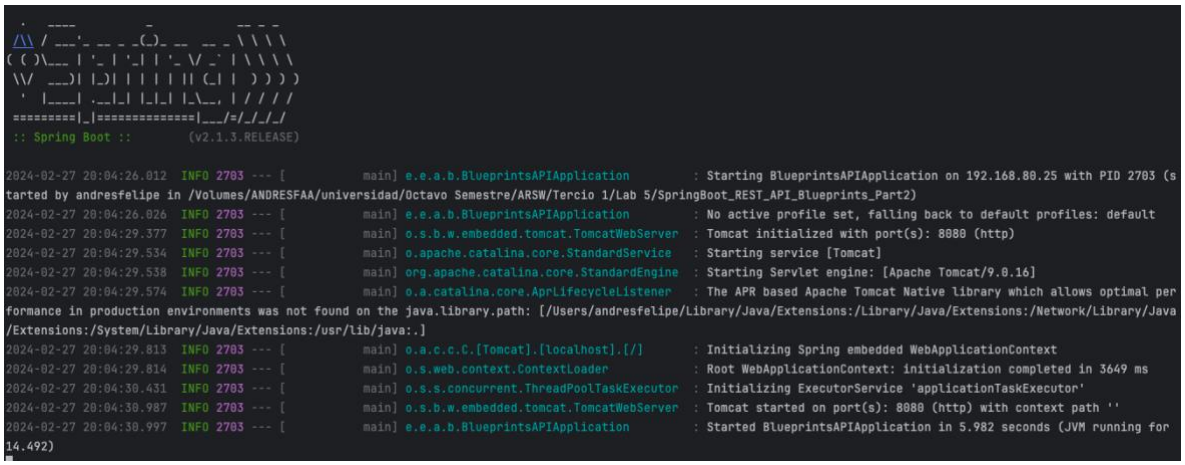
Inyectamos BlueprintsServices en el controlador de esta manera al ser instanciado el controlador, Spring boot instanciara las dependencias requeridas en blueprintsServcies y posteriormente las inyectara.

iv. Verifique el funcionamiento de a aplicación lanzando la aplicación con maven:

- mvn compile

```
andresfelipe@192 SpringBoot_REST_API_Blueprints_Part2 % mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< edu.eci.pdsw.examples:blueprints-api >----------------
[INFO] Building Blueprints_API 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:2.6:resources (default-resources) @ blueprints-api ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- compiler:3.1:compile (default-compile) @ blueprints-api ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 15 source files to /Volumes/ANDRESFAA/universidad/Octavo Semestre/ARSW/Tercio 1/Lab 5/SpringBoot_REST_API_Blueprints_Part2/target/classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.553 s
[INFO] Finished at: 2024-02-27T19:02:23-05:00
[INFO] ------------------------------------------------------------------------
andresfelipe@192 SpringBoot_REST_API_Blueprints_Part2 %
```

- 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 suministados por defecto, y que se haya aplicado el filtrado de puntos correspondiente.

v. Modifique el controlador para que ahora, acepte peticiones GET al recurso /blueprints/{author}, el cual retorne usando una representación jSON todos los planos realizados por el autor cuyo nombre sea {author}. Si no existe dicho autor, se debe responder con el código de error HTTP 404. Para esto, revise en la documentación de Spring, sección 22.3.2, el uso de @PathVariable. De nuevo, verifique que al hacer una petición GET -por ejemplo- a recurso http://localhost:8080/blueprints/juan, se obtenga en formato jSON el conjunto de planos asociados al autor 'juan'.

```java
@RequestMapping(value = "/blueprints/{Author}",method = RequestMethod.GET)
public ResponseEntity<?> getBluePrintsByAuthor(@PathVariable String Author) {
    Set<Blueprint> blueprintSet = null;
    try {
        blueprintSet = blueprintsServices.getBlueprintsByAuthor(Author);
        if(blueprintSet.isEmpty()) {
            return new ResponseEntity<>( body: "404 Not Found", HttpStatus.NOT_FOUND);
        }else {
            return new ResponseEntity<>(blueprintSet, HttpStatus.ACCEPTED);
        }
    } catch (Exception ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, msg: null, ex);
        return new ResponseEntity<>( body: "Error" + ex.getMessage(), HttpStatus.NOT_FOUND);
    }
}
```

En esta parte modificamos el controlador agregándole un método con tipo de petición GET, que nos permite obtener los planos de un determinado autor, agregándole la condición de que, si no existe un plano con ese autor, arroje un error de tipo HTTP 404.

En esta parte comprobamos que funcione esta nueva implementación, pidiéndole los planos de Sebastián.

En este caso probamos con un autor que no existe, y de esta manera comprobamos que funciona el error HTTP 404.

vi. 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

```java
@RequestMapping(value = "/blueprints/{Author}/{bpname}",method = RequestMethod.GET)
public ResponseEntity<?> getBluePrintsByAuthor(@PathVariable String Author, @PathVariable String bpname) {
    Blueprint blueprint = null;
    try {
        blueprint = blueprintsServices.getBlueprint(Author,bpname);
        if(blueprint == null) {
            return new ResponseEntity<>( body: "404 Not Found", HttpStatus.NOT_FOUND);
        }else {
            return new ResponseEntity<>(blueprint, HttpStatus.ACCEPTED);
        }
    } catch (Exception ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, msg: null, ex);
        return new ResponseEntity<>( body: "Error" + ex.getMessage(), HttpStatus.NOT_FOUND);
    }
}
```

En esta parte modificamos el controlador agregándole un método con tipo de petición GET, que nos permite obtener un plano de un determinado autor, agregándole la condición de que, si no existe un plano con ese autor, arroje un error de tipo HTTP 404.

En esta parte comprobamos que funcione esta nueva implementación, pidiéndole el plano andres_p1 de andres.

En este caso probamos con un plano que no existe, y de esta manera comprobamos que funciona el error HTTP 404.

- **Parte II**

i. 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):

```java
@RequestMapping(value = "/blueprints/agregar",method = RequestMethod.POST)
public ResponseEntity<?> createNewBluePrints(@RequestBody Blueprint bp) {
    try {
        blueprintsServices.addNewBlueprint(bp);
        return new ResponseEntity<>( body: "El plano se agrego correctamente",HttpStatus.CREATED);
    } catch (Exception ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE,  msg: null, ex);
        return new ResponseEntity<>( body: "Error" + ex.getMessage(), HttpStatus.FORBIDDEN);
    }
}
```

Agregamos un nuevo método en el que se puede agregar nuevos planos mediante la petición POST.

ii. 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}'

```
andresfelipe@192 SpringBoot_REST_API_Blueprints_Part2 % curl -i -X POST -H "Content-Type: application/json" -H "Accept: application/json" http://localhost:8080/blueprints/agregar -d '{"author":"felipe","points":[{"x":140,"y":140}],"name":"plano4"}'

HTTP/1.1 201
Content-Type: application/json;charset=UTF-8
Content-Length: 32
Date: Wed, 28 Feb 2024 02:22:16 GMT

El plano se agrego correctamente%
andresfelipe@192 SpringBoot_REST_API_Blueprints_Part2 %
```

Usando el commando curl -i -X POST -H "Content-Type: application/json" -H "Accept: application/json" http://localhost:8080/blueprints/agregar -d '{"author":"felipe","points":[{"x":140,"y":140}],"name":"plano4"}', Podemos observer que

el plano se agregó correctamente

iii. 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.

Teniendo en cuenta el punto anterior comprobamos si se agregó el plano de Felipe, usamoslocalhost:8080/blueprints/felipe/plano4 para verificar si definitivamente se agregó de manera correcta.

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

```
new *
@RequestMapping(value = "/blueprints/{Author}/{bpname}",method = RequestMethod.PUT)
public ResponseEntity<?> updateBluePrint(@PathVariable String author, @PathVariable String bpname, @RequestBody List<Point> points) {
    try {
        blueprintsServices.updateBluePrint(author,bpname,points);
        return new ResponseEntity<>( body: "El plano se actualizo correctamente",HttpStatus.OK);
    } catch (Exception ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, msg: null, ex);
        return new ResponseEntity<>( body: "Error" + ex.getMessage(), HttpStatus.FORBIDDEN);
    }
}
```

En controlador se agregó un nuevo método para actualizar un plano mediante la petición PUT.

- **Parte III**

i. ¿Qué condiciones de carrera se podrían presentar?

Cuando necesitamos consultar o agregar nuevos planos, podemos experimentar un escenario donde accedemos al HashMap donde se almacenan los planos.

```
@Component
@Qualifier("inMemoryBluePrintPersistence")
public class InMemoryBlueprintPersistence implements BlueprintsPersistence{

    12 usages
    private final Map<Tuple<String,String>,Blueprint> blueprints=new ConcurrentHashMap<>();
```

ii. ¿Cuáles son las respectivas regiones críticas?

Dado que el almacenamiento persistente de los planos no es seguro para hilos (thread-safe), surge una condición de carrera cuando dos solicitudes intentan agregar un plano con el mismo nombre y autor al mismo tiempo.

```
@Override
public void saveBlueprint(Blueprint bp) throws BlueprintPersistenceException {
    if (blueprints.containsKey(new Tuple<>(bp.getAuthor(),bp.getName()))){
        throw new BlueprintPersistenceException("The given blueprint already exists: "+bp);
    }
    else{
        blueprints.putIfAbsent(new Tuple<>(bp.getAuthor(),bp.getName()), bp);
    }
}
```

Para asegurar el funcionamiento thread-safe, vamos a cambiar el tipo de HashMap a ConcurrentHashMap, ya que esta clase garantiza un funcionamiento thread-safe. En cuanto a la condición de carrera, vamos a sustituir el método put() por el método putIfAbsent(), que previene la condición de carrera.

Conclusiones

- El laboratorio se centra en el desarrollo de un componente BlueprintsRESTAPI utilizando el framework Spring, específicamente SpringMVC y SpringBoot, para implementar un API REST que gestiona planos arquitectónicos. La elección de Spring sugiere un enfoque robusto y ampliamente utilizado para construir aplicaciones.

- El objetivo principal es proporcionar un medio estandarizado e independiente de la plataforma para gestionar planos arquitectónicos de forma centralizada. Esto implica la integración de beans, configuración de inyección de dependencias y la implementación de recursos que permitan la obtención y creación de planos a través de peticiones HTTP.

- La última parte del laboratorio aborda aspectos concurrentes del componente BlueprintsRESTAPI. Se destaca la importancia de identificar y solucionar posibles condiciones de carrera y regiones críticas en el código existente, subrayando la necesidad de garantizar un funcionamiento eficiente y concurrente del API para manejar múltiples solicitudes simultáneas de manera efectiva.

Bibliografía

- http://www.drdobbs.com/web-development/soa-web-services-and-restful-systems/199902676?pgno=1 (sólo la página 1).

- SOAP vs REST: https://stackify.com/soap-vs-rest/