# API REST BLUEPRINTS PARA LA GESTIÓN DE PLANOS – LAB 05

**AUTORES:**

YORKS GOMEZ – CESAR VÁSQUEZ

**PROFESOR:**

JAVIER IVAN TOQUICA

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

ARQUITECTURA DE SOTFWARE – GRUPO # 1

INGENIERÍA DE SISTEMAS

BOGOTÁ D.C.

2023

# INTRODUCCIÓN

En este laboratorio gestionaremos los planos arquitectónicos de una prestigiosa compañía de diseño, la idea de este API es ofrecer un medio estandarizado e 'independiente de la plataforma' para que las herramientas que se desarrollen a futuro para la compañía puedan gestionar los planos de forma centralizada. Utilizaremos la transferencia de estado representacional (REST) donde los componentes débilmente acoplados se comunicarán a través de interfaces por protocolos web estándar.

Aplicando los criterios de evaluación, diseño, funcionalidad y análisis de concurrencia de forma correcta para entender, usamos las herramientas apropiadas para la solución de este laboratorio.

## DESARROLLO

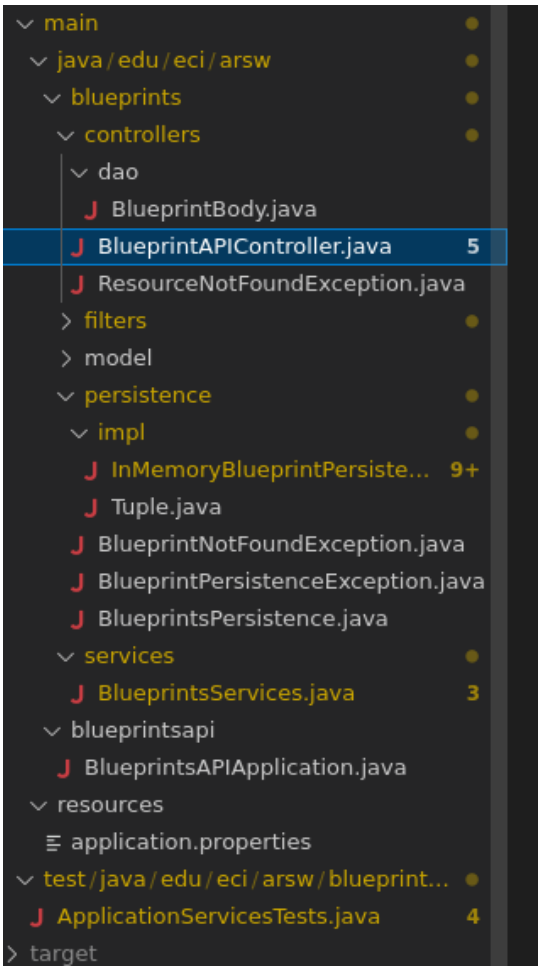
### PARTE 1

Durante esta sección integramos lo que hicimos en el laboratorio pasado al actual, por lo tanto, copiamos las clases pertinentes y las conectamos con @Service y @Autowired

```
 */
@Service("BlueprintsServices")
public class BlueprintsServices {
```

```
/**
 *
 * @author hcadavid
 */
@RestController
@RequestMapping(value = "/blueprints")
public class BlueprintAPIController {

    @Autowired
    BlueprintsServices services;
```

Aquí podemos ver las clases agregadas




Posteriormente adicionamos algunos planos por defecto a InMemoryBlueprintPersistance

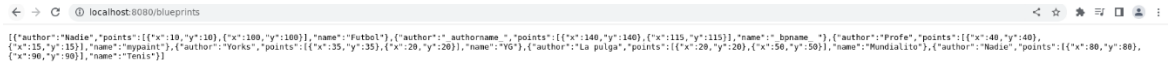
```
public InMemoryBlueprintPersistence() {
    //load stub data
    Point[] pts=new Point[]{new Point(x: 140, y: 140),new Point(x: 115, y: 115)};
    Blueprint bp=new Blueprint(author: "_authorname_", name: "_bpname_ ",pts);
    blueprints.put(new Tuple<>(bp.getAuthor(),bp.getName()), bp);
    // Adicionando más
    Point[] pts0 = new Point[]{new Point(x: 40, y: 40),new Point(x: 15, y: 15)};
    Blueprint bp0 = new Blueprint(author: "Profe", name: "mypaint",pts0);
    blueprints.put(new Tuple<>(bp0.getAuthor(),bp0.getName()), bp0);
    Point[] pts1 = new Point[]{new Point(x: 0, y: 0),new Point(x: 10, y: 10)};
    /*
    Blueprint bp1 = new Blueprint("Cesar", "CV",pts1);
    blueprints.put(new Tuple<>(bp1.getAuthor(),bp1.getName()), bp1);
    */
    Point[] pts2 = new Point[]{new Point(x: 35, y: 35),new Point(x: 20, y: 20),new Point(x: 20,y: 20)};
    Blueprint bp2 = new Blueprint(author: "Yorks", name: "YG",pts2);
    blueprints.put(new Tuple<>(bp2.getAuthor(),bp2.getName()), bp2);
    Point[] pts3 = new Point[]{new Point(x: 20, y: 20),new Point(x: 50, y: 50)};
    Blueprint bp3 = new Blueprint(author: "La pulga", name: "Mundialito",pts3);
    blueprints.put(new Tuple<>(bp3.getAuthor(),bp3.getName()), bp3);
    Point[] pts4 = new Point[]{new Point(x: 10, y: 10),new Point(x: 100, y: 100)};
    Blueprint bp4 = new Blueprint(author: "Nadie", name: "Futbol",pts4);
    blueprints.put(new Tuple<>(bp4.getAuthor(),bp4.getName()), bp4);
    Point[] pts5 = new Point[]{new Point(x: 80, y: 80),new Point(x: 90, y: 90)};
    Blueprint bp5 = new Blueprint(author: "Nadie", name: "Tenis",pts5);
    blueprints.put(new Tuple<>(bp5.getAuthor(),bp5.getName()), bp5);

}
```

Además, adicionamos las etiquetas correspondientes al API Controller, que adiciona una

ruta base /blueprints, y el método get() que gestiona el resultado de esta ruta por defecto, la misma retorna todos los planos correspondientes.

```java
 */
@RestController
@RequestMapping(value = "/blueprints")
public class BlueprintAPIController {

    @Autowired
    BlueprintsServices services;

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

Ejecutando la consulta a /blueprints obtenemos la lista de todos los planos




Después, como nos lo pide el laboratorio, adicionamos la ruta /blueprints/{author} de la siguiente manera

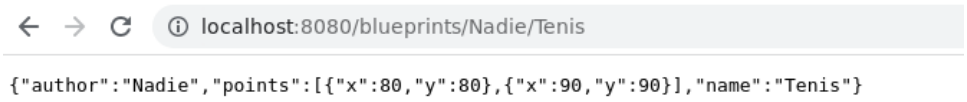
```java
@RequestMapping(method = RequestMethod.GET, value = "/{author}")
public ResponseEntity<?> getByAuthor(@PathVariable String author) {
    try {
        //obtener datos que se enviarán a través del API
        return new ResponseEntity<>(services.getBlueprintsByAuthor(author),HttpStatus.ACCEPTED);
    } catch (BlueprintNotFoundException ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, null, ex);
        return new ResponseEntity<>(body: "Error 404 Blueprint not found",HttpStatus.NOT_FOUND);
    }
}
```

Ejecutando esta consulta obtenemos el siguiente resultado (en el caso del autor Nadie)


Finalmente manejamos la ruta /blueprints/{author}/{bpname} de la siguiente manera

```java
@RequestMapping(method = RequestMethod.GET, value = "/{author}/{bpname}")
public ResponseEntity<?> getByAuthorAndName(@PathVariable String author, @PathVariable String bpname) {
    try {
        //obtener datos que se enviarán a través del API
        return new ResponseEntity<>(services.getBlueprint(author, bpname),HttpStatus.ACCEPTED);
    } catch (BlueprintNotFoundException ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, null, ex);
        return new ResponseEntity<>(body: "Error 404 Blueprint not found",HttpStatus.NOT_FOUND);
    }
}
```

Y ejecutando con el autor Nadie y el plano Tenis obtenemos




Así damos como finalizada la primera parte del laboratorio

**PARTE 2**

En esta segunda sección empleamos los métodos POST y PUT con el fin de seguir desarrollando nuestro “mini-crud”.

Comenzamos trabajando el POST, en el que se nos pide crear un plano, lo hacemos de la siguiente manera

```java
@RequestMapping(method = RequestMethod.POST, value = "/create")
public ResponseEntity<?> create(@RequestBody BlueprintBody body) {
    try {
        Point[] pts = new Point[]{new Point(body.getX1(),body.getY1()),new Point(body.getX2(), body.getY2())};
        Blueprint bp = new Blueprint(body.getAuthor(), body.getName(), pts);
        services.addNewBlueprint(bp);

        return new ResponseEntity<>(HttpStatus.CREATED);
    } catch (Exception ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, null, ex);
        return new ResponseEntity<>(body: "Could not create blueprint",HttpStatus.NOT_FOUND);
    }
}
```

Y ejecutando el POST obtenemos


Lo cual es una salida correcta, y lo verificamos usando el autor Kevin y el plano ola. Obtenemos la siguiente salida


Dicha salida es correcta, con lo que los planos se están creando correctamente.

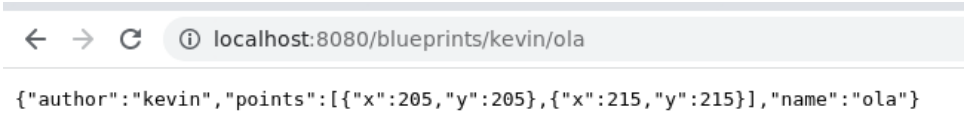
Adicionalmente se nos solicita crear un putt para modificar los planos, por lo que creamos el siguiente método

```
@RequestMapping(method = RequestMethod.PUT, value = "/{author}/{bpname}")
public ResponseEntity<?> update(@RequestBody BlueprintBody body, @PathVariable String author, @PathVariable String bpname) {
    try {
        services.updateBlueprint(author, bpname, body);
        return new ResponseEntity<>(HttpStatus.CREATED);
    } catch (Exception ex) {
        Logger.getLogger(BlueprintAPIController.class.getName()).log(Level.SEVERE, null, ex);
        return new ResponseEntity<>(body: "Could not update blueprint",HttpStatus.NOT_FOUND);
    }
}
```

Y ejecutando la siguiente consulta obtenemos un cambio correcto

```
edhawk@edhawk:~$ curl -i -X PUT -HContent-Type:application/json -HAccept:application/json http://localhost:8080/blueprints/kevin/ola -d '{"x1":205,"y1":205,"x2":220}'
HTTP/1.1 201
Content-Length: 0
Date: Wed, 08 Mar 2023 03:53:35 GMT

edhawk@edhawk:~$
```

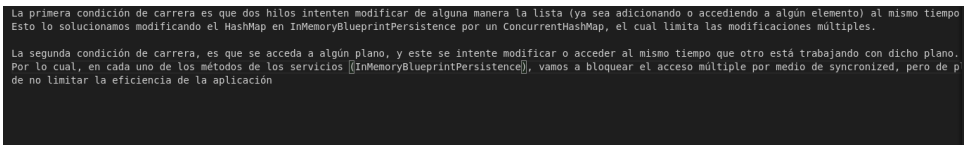
Allí estamos modificando los puntos del plano recién creado en el POST, podemos ver las modificaciones utilizando nuestro método GET




Por lo tanto, el funcionamiento es correcto

**Parte 3**

Se adiciona el análisis pertinente



```
La primera condición de carrera es que dos hilos intenten modificar de alguna manera la lista (ya sea adicionando o accediendo a algún elemento) al mismo tiempo
Esto lo solucionamos modificando el HashMap en InMemoryBlueprintPersistence por un ConcurrentHashMap, el cual limita las modificaciones múltiples.

La segunda condición de carrera, es que se acceda a algún plano, y este se intente modificar o acceder al mismo tiempo que otro está trabajando con dicho plano.
Por lo cual, en cada uno de los métodos de los servicios (InMemoryBlueprintPersistence), vamos a bloquear el acceso múltiple por medio de syncronized, pero de p
de no limitar la eficiencia de la aplicación
```

Modificamos el tipo de HashMap

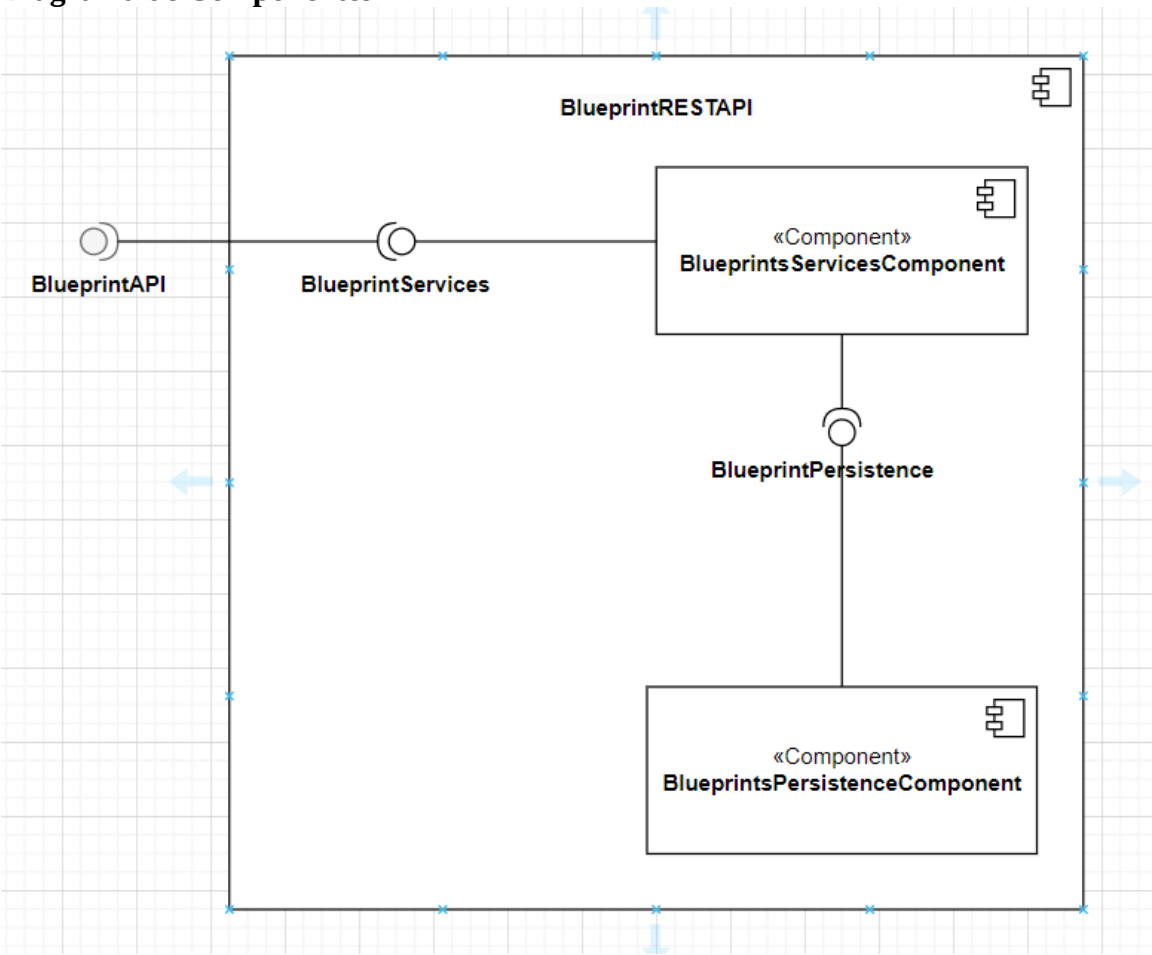
```
private final Map<Tuple<String,String>,Blueprint> blueprints=new ConcurrentHashMap<>();
```

Adicionamos los syncronized necesarios

```
 @Override
 public Blueprint getBlueprint(String author, String bprintname) throws BlueprintNotFoundException {
     Blueprint bp= blueprints.get(new Tuple<>(author, bprintname));

     synchronized(bp) {
         if (bp==null){
             throw new BlueprintNotFoundException(message: "Blueprint no se pudo encontrar");
         }

         filter.filter(bp);
         return bp;
     }

 }

@Override
public Set<Blueprint> getBlueprintsByAuthor(String author) throws BlueprintNotFoundException {
    Set<Tuple<String, String>> keys = blueprints.keySet();
    HashSet bps = new HashSet();
    synchronized(keys) {
        for (Tuple t : keys){
            if (author.equals(t.o1)){
                bps.add(blueprints.get(new Tuple<>(author,(String)t.o2)));
            }
        }
        if(bps.isEmpty()){
            throw new BlueprintNotFoundException("El autor: " + author + " No existe");
        }
        return bps;
    }

}
@Override
public Set<Blueprint> getAllBlueprints() throws BlueprintNotFoundException {
    Set<Tuple<String, String>> keys=blueprints.keySet();
    HashSet bps=new HashSet();
    synchronized(keys) {
        for (Tuple t : keys){
            bps.add(blueprints.get(new Tuple<>((String)t.o1,(String)t.o2)));
            Blueprint bp = blueprints.get(new Tuple<>((String)t.o1,(String)t.o2));

            synchronized(bp) {
                filter.filter(bp);
                bps.add(bp);
            }

        }
        return bps;
    }
}
```

Con lo que esta API podría ser concurrente sin ningún problema

## Diagrama de Componentes



## CONCLUSIONES

- Aprendimos a usar de una mejor manera las herramientas para el desarrollo. En nuestro caso REST, fue útil para nuestro sistema software.
- Gestionamos los planos arquitectónicos de una manera efectiva y eficiente.
- Mejoramos la definición de región crítica y condiciones de carrera aplicándolas en el laboratorio.
- Se puede hacer la combinación de la arquitectura de software (interfaces, componentes, conectores, etc) con los de la arquitectura de red (portabilidad, administración de ancho de banda, patrones, etc).