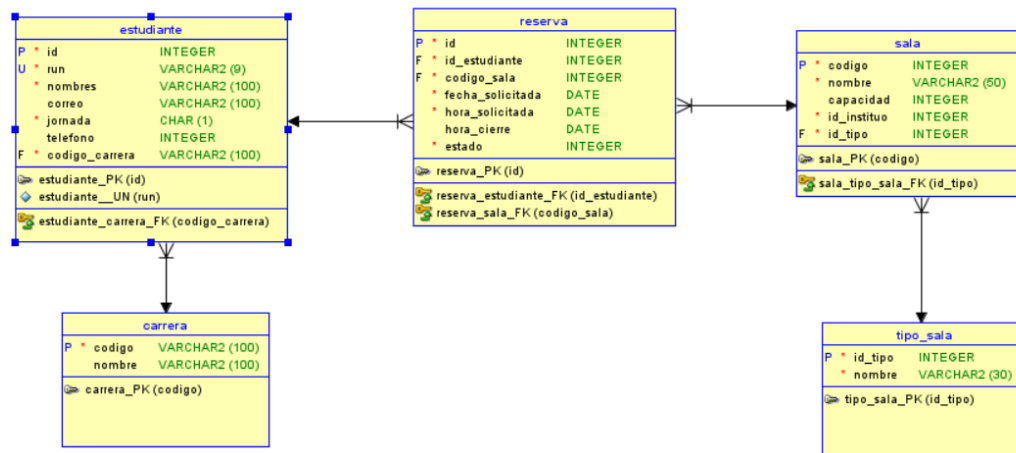


Guía práctica y ejercicios

Proyecto reserva de salas biblioteca

Implementar HATEOAS

En esta guía, implementaremos las dependencias de HATEOAS en el proyecto llamado "Salas Bibliotecas".



Contexto del caso

Este proyecto consiste en un sistema de reserva de salas desarrollado en Spring. El sistema permite que los estudiantes reserven salas disponibles.

Estructura de la Base de Datos

La estructura de la base de datos incluye las siguientes tablas:

- Carrera, Estudiante, Reserva, Sala, Tipo de Sala

Paso 1: Añadir Dependencias de HATEOAS

<https://spring.io/projects/spring-hateoas>

Primero, necesitas añadir las dependencias necesarias en tu archivo **pom.xml**.

```

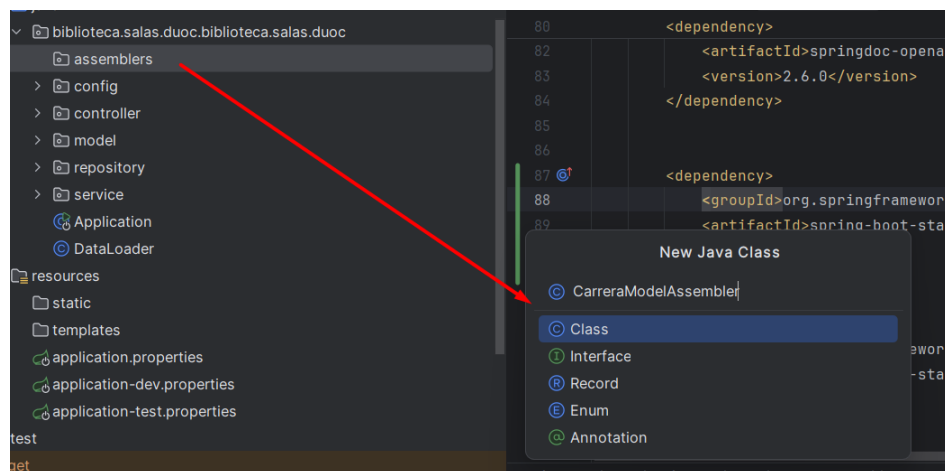
86
87
88     <groupId>org.springframework.boot</groupId>
89     <artifactId>spring-boot-starter-hateoas</artifactId>
90 </dependency>
91
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>

```

Paso 2: Crear un ensamblador de recursos

Un ensamblador de recursos se encarga de crear enlaces HATEOAS para una entidad.

- Crear la carpeta **assemblers** y el modelo **CarreraModelAssembler**
 - (ver archivo [carreraModelAssembler.txt](#))

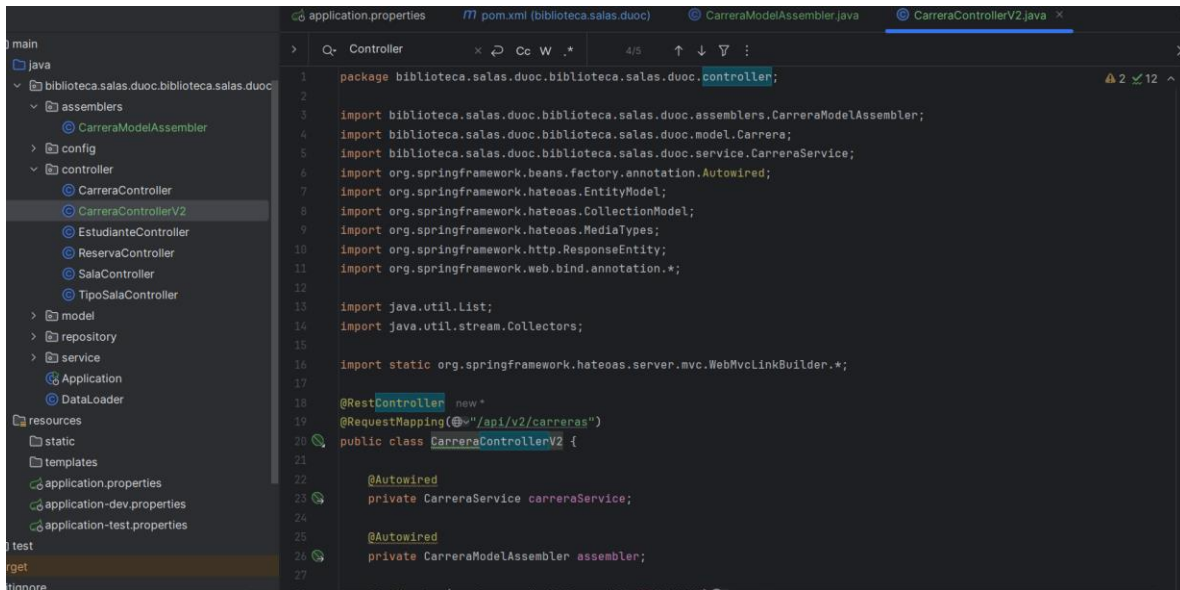


Paso3: Reglas de versión con y sin HATEOAS

Vamos a implementar dos versiones del controlador **CarreraController**. La primera versión no incluirá HATEOAS ❌, mientras que la segunda versión sí lo hará ✅.

- ✅ Para acceder a la versión con HATEOAS, se deben realizar las llamadas a <http://localhost:8080/api/v2/...>, utilizando los controladores que incorporan esta funcionalidad.

Crear el controlador **CarreraControllerV2** (ver archivo [CarreraControllerV2.txt](#))

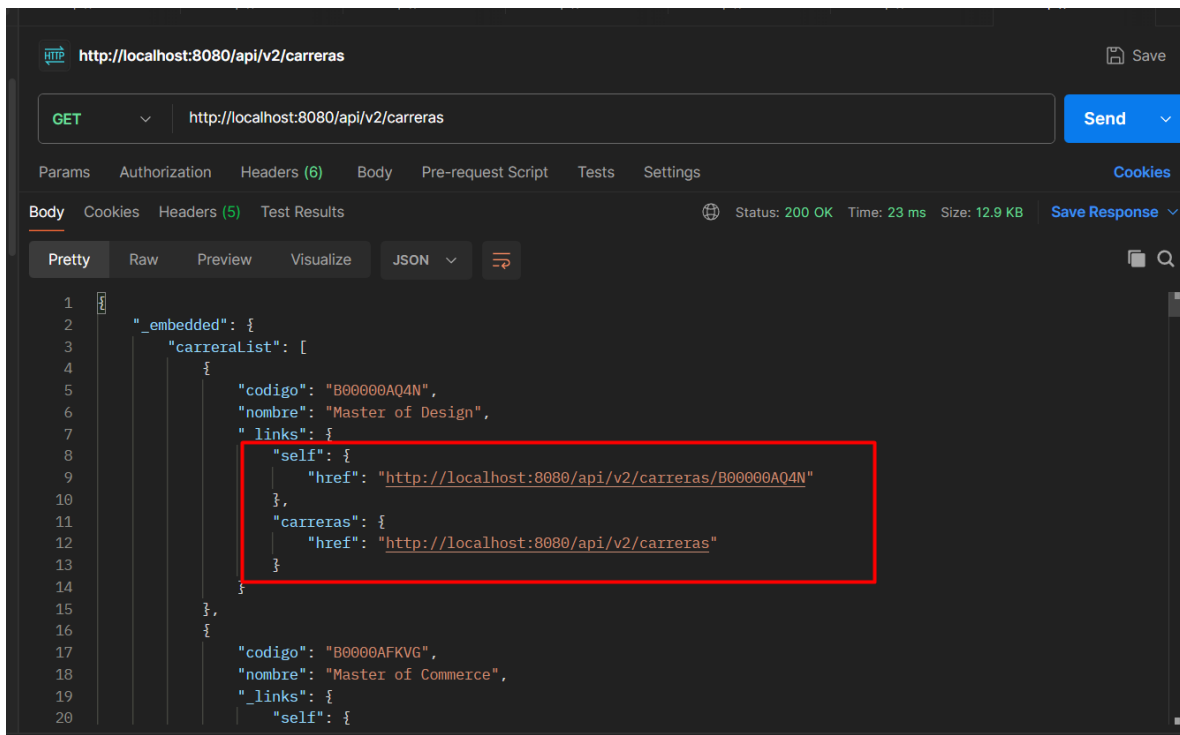


```

1 package biblioteca.salas.duoc.biblioteca.salas.duoc.controller;
2
3 import biblioteca.salas.duoc.biblioteca.salas.duoc.assemblers.CarreraModelAssembler;
4 import biblioteca.salas.duoc.biblioteca.salas.duoc.model.Carrera;
5 import biblioteca.salas.duoc.biblioteca.salas.duoc.service.CarreraService;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.hateoas.EntityModel;
8 import org.springframework.hateoas.CollectionModel;
9 import org.springframework.hateoas.MediaType;
10 import org.springframework.http.ResponseEntity;
11 import org.springframework.web.bind.annotation.*;
12
13 import java.util.List;
14 import java.util.stream.Collectors;
15
16 import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;
17
18 @RestController
19 @RequestMapping("/api/v2/carreras")
20 public class CarreraControllerV2 {
21
22     @Autowired
23     private CarreraService carreraService;
24
25     @Autowired
26     private CarreraModelAssembler assembler;
27
28 }
    
```

Probar la API con HATEOAS

Al ejecutar tu aplicación, los endpoints de **CarreraControllerV2** ahora devolverán respuestas **HATEOAS** con enlaces que permiten a los clientes navegar dinámicamente por la API.



```

1 {
2   "_embedded": {
3     "carreraList": [
4       {
5         "codigo": "B00000A04N",
6         "nombre": "Master of Design",
7         "_links": {
8           "self": {
9             "href": "http://localhost:8080/api/v2/carreras/B00000A04N"
10          },
11           "carreras": {
12             "href": "http://localhost:8080/api/v2/carreras"
13           }
14         }
15       },
16       {
17         "codigo": "B00000AFKVG",
18         "nombre": "Master of Commerce",
19         "_links": {
20           "self": {
21             "href": "http://localhost:8080/api/v2/carreras/B00000AFKVG"
22           },
23           "carreras": {
24             "href": "http://localhost:8080/api/v2/carreras"
25           }
26         }
27       }
28     ]
29   }
30 }
    
```

Con esto ya estamos entregando una api con HATEOAS



Guía paso a paso: Añadir métodos personalizados en HATEOAS

Agregar 5 métodos personalizados en el controlador **ReservaControllerV2** con HATEOAS.

Métodos Personalizados

1. Obtener todas las reservas de una sala específica.

Demostración ejercicio 1

Obtener todas las reservas de una sala específica.

Primero, añadimos el método en el servicio **ReservaService** que recuperará todas las reservas asociadas a un código de sala específico.



```
12 public class ReservaService {  
34     return reservaRepository.findByEstudianteId(idEstudiante);  
35 }  
36  
37 public List<Reserva> findBySalaCodigo(Integer codigoSala) {  
38     return reservaRepository.findBySalaCodigo(codigoSala);  
39 }  
40 }
```

- Función **findBySalaCodigo** solicita un código y retorna una lista de reserva.
- La interface **ReservaRepository** con el metodo **findBySalaCodigo**, buscará por el código la sala.

Añadir el Método en ReservaRepository

Luego, añadimos el método correspondiente en el repositorio **ReservaRepository**, ya que el repositorio se encargará de llamar a la base de datos mediante el ORM.

- No hay queries, ya que el ORM interpreta la solicitud.

```

package ...

> import ...

public interface ReservaRepository extends JpaRepository<Reserva, Integer> {
    List<Reserva> findByEstudianteId(Integer idEstudiante);
    List<Reserva> findBySalaCodigo(Integer codigoSala);
    List<Reserva> findByFechaSolicitada(Date fecha);
    List<Reserva> findByEstado(Integer estado);
    List<Reserva> findByFechaSolicitadaBetween(Date start, Date end);
    long countByEstudianteId(Integer idEstudiante);
}

```

Añadir el Método en el Controlador ReservaControllerV2

Finalmente, añadimos el método en el controlador **ReservaControllerV2** para manejar la solicitud HTTP y devolver las reservas en formato HATEOAS.

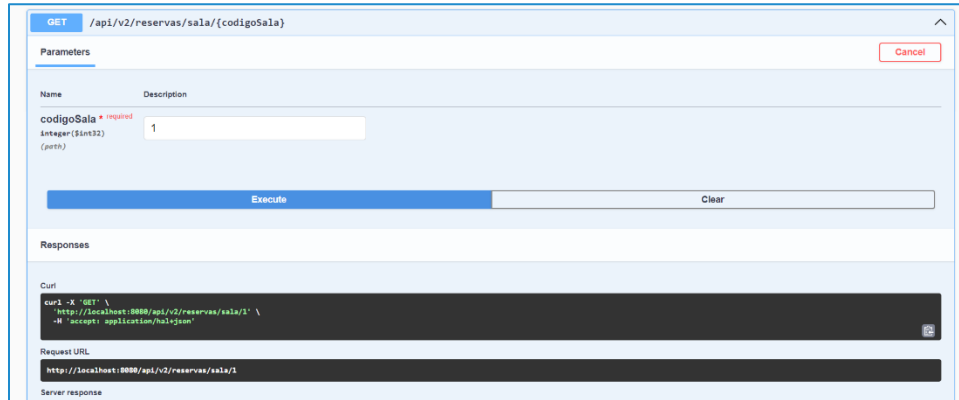
```

// Método personalizado para obtener reservas por sala
@GetMapping(value = @"/sala/{codigoSala}", produces = MediaType.HAL_JSON_VALUE)
public CollectionModel<EntityModel<Reserva>> getReservasBySala(@PathVariable Integer codigoSala) {
    List<EntityModel<Reserva>> reservas = reservaService.findBySalaCodigo(codigoSala).stream()
        .map(Assembler::toModel)
        .collect(Collectors.toList());

    return CollectionModel.of(reservas,
        LinkTo(methodOn(ReservaControllerV2.class).getReservasBySala(codigoSala)).withSelfRel());
}

```

Probar api rest nueva con HATEOAS



GET /api/v2/reservas/sala/{codigoSala}

Parameters

Name	Description
codigoSala * required	

Integer(\$int32) (path)

1

Execute Clear

Responses

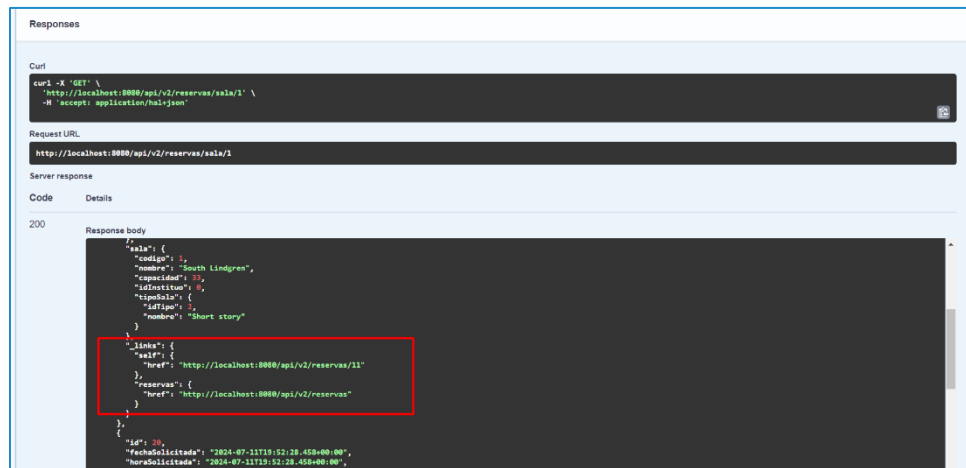
Curl

```
curl -X GET 'http://localhost:8080/api/v2/reservas/sala/1' \
-H 'accept: application/hal+json'
```

Request URL

http://localhost:8080/api/v2/reservas/sala/1

Server response



Responses

Curl

```
curl -X GET 'http://localhost:8080/api/v2/reservas/sala/1' \
-H 'accept: application/hal+json'
```

Request URL

http://localhost:8080/api/v2/reservas/sala/1

Server response

Code Details

200

Response body

```
{
  "sala": {
    "codigo": 1,
    "nombre": "South Lindgren",
    "capacidad": 33,
    "descripcion": 9,
    "tipoSala": {
      "idTipo": 2,
      "nombre": "Short story"
    }
  },
  "links": {
    "self": {
      "href": "http://localhost:8080/api/v2/reservas/sala/1"
    },
    "reservas": {
      "href": "http://localhost:8080/api/v2/reservas"
    }
  }
},
{
  "id": 20,
  "fechaSolicitud": "2024-07-11T13:52:28.458+00:00",
  "horaSolicitud": "2024-07-11T13:52:28.458+00:00"
}
```

Ejercicios para el estudiante

Ya están creados estos cuatros reportes con HATEOAS, analiza el caso y puebas los endpoints.

- Obtener todas las reservas en una fecha específica.
- Obtener todas las reservas con un estado específico.
- Obtener todas las reservas entre dos fechas.
- Obtener el total de reservas realizadas por un estudiante.

Desarrolla esos cinco reportes faltantes

- Obtener todas las reservas de un estudiante en una fecha específica.
- Obtener todas las reservas de una sala en un estado específico.
- Obtener todas las reservas de un estudiante entre dos fechas.
- Obtener todas las reservas de una sala entre dos fechas.

5. Obtener el total de reservas realizadas en una sala específica.

