Spring framework 5 - REST

Configuración

Para habilitar el soporte para REST se debe incluir el módulo Spring MVC a través de la siguiente dependencia:

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring provee una serie de herramientas de programación llamadas **DevTools**, para utilizarlas se debe incluir la siguiente dependencia:

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-devtools</artifactId>
	<scope>runtime</scope>
	<optional>true</optional>
</dependency>
```

Al ejecutar la aplicación, se mantendrá en ejecución hasta que se detenga de forma explícita.

Spring MVC

Spring MVC es un módulo que soporta el patrón de diseño modelo vista controlador:

-**Model**: Representa un **POJO** de **Java** que transporta los datos

-**View**: Representa la visualización de los datos que contiene el *modelo*

-**Controller**: Controla el flujo de la información entre el modelo y la vista, mantiene el modelo y la vista separados.

Anotaciones

A continuación se presentan las anotaciones importantes al trabajar con **Spring MVC**:

-**@Controller**: **Stereotype** que define a una clase como un controlador de **Spring**

-**@RestController**: Meta anotación que representa a(**@Controller** y **@ResponseBody**)

-**@RequestMapping**: Se puede aplicar a nivel de clase y método y permite mapear peticiones **HTTP** a métodos llamados **handler methods**.

-**@GetMapping**: Es una anotación compuesta que actúa como **@RequestMapping**(method=RequestMethod.GET)

-**@PostMapping**: Es una anotación compuesta que actúa como **@RequestMapping**(method=RequestMethod.POST)

-**@PutMapping**: Es una anotación compuesta que actúa como **@RequestMapping**(method=RequestMethod.PUT)

-**@DeleteMapping**: Es una anotación compuesta que actúa como **@RequestMapping**(method=RequestMethod.DELETE)

-**@PatchMapping**: Es una anotación compuesta que actúa como **@RequestMapping**(method=RequestMethod.PATCH)

-**@PathVariable**: Indica que un parámetro de un método debe tomar el valor de alguno de la URL, es conocido como **path param**.

-**@RequestParam**: Indica que un parámetro de un método debe tomar el valor de un **request parameter**, es conocido como **query param**

-**@ResponseStatus**: Se aplica en métodos y excepciones y define el status HTTP que se devolverá

-**@Service**: Es utilizado por los **controllers** y contiene la lógica de negocio de la aplicación

@Controller, @RestController y @RequestMapping

A continuación se muestra un ejemplo sobre el uso de **@Controller** y **@RestController** :

```
@RestController
@RequestMapping("/roles")
public class RoleController {
	//handler methods
}
```

@RequestMapping define que las peticiones sobre la URL /roles serán interceptadas por los **handler methods** definidos en la clase. En este ejmplo se puede reemplazar **@RestController** por **@Controller** y tendrá la misma funcionalidad.

@RequestMapping en métodos

@RequestMapping se puede aplicar tanto en clases como en métodos. Veamos el siguiente ejemplo:

```
@RestController
@RequestMapping("/roles")
public class RoleController {
	@Autowired
	private RoleService roleService;


	@RequestMapping(value = "/{id}", method =
RequestMethod.GET)
	public ResponseEntity<List<Role>> getById(.....) {
		return new ResponseEntity<>(roleService.findAll(),
	HttpStatus.OK);
	}
}
```

En el ejemplo anterior, el método **getAll** atenderá las peticiones hacia la **URL** "/roles/{id}" sobre el método **HTTP** GET.

@PathVariable

En el ejemplo anterior, se explicó cómo generar una URL con base en múltiples **@RequestMapping**. Ahora se mostrará como añadir **@PathVariable**:

```
@RestController
@RequestMapping("/roles")
public class RoleController {
	@Autowired
	private RoleService roleService;


	@RequestMapping(value = "/{id}", method =
RequestMethod.GET)
	public ResponseEntity<Role>
getById(@PathVariable("id") int id) {
		return new
ResponseEntity<>(roleService.findById(id),
	HttpStatus.OK);
	}
}
```

En el ejemplo anterior, el método **getAll** atenderá las peticiones hacia la URL "/roles/{id}" sobre el método HTTP GET, una petición de ejemplo sería http://localhost:8080/roles/1.

En este ejemplo el valor 1 sería el valor de la **@PathVariable** "{id}" y se asignará en el parámetro **id** del método getById gracias a la anotación **@PathVariable**.

@GetMapping

Es posible simplificar el **endpoint** anterior a través de @GetMapping como se muestra a continuación:

```
@GetMapping(value = "/{id}")
public ResponseEntity<Role> getById(@PathVariable("id") int
id) {
	return new ResponseEntity<>(roleService.findById(id),
HttpStatus.OK);
}
```

Las anotaciones **@PostMapping**, **@PutMapping**, **@PatchMapping**, etc. funcionan exactamente igual.

@RequestParam

@RequestParam nos permite obtener **query params** desde la URL. Veamos un ejemplo:

```
@GetMapping
public ResponseEntity<List<Role>> getAll
(@RequestParam("page") int page,
	@RequestParam("size") int size) {

	log.info("page {} size {} ", page, size);
	return new ResponseEntity<>
	(roleService.findAll(), HttpStatus.OK);
}
```

En el ejemplo anterior el método **getAll** atenderá las peticiones hacia la URL "/roles" sobre el método HTTP GET. Una petición de ejemplo sería http://localhost:8080/roles?page=1&size=10.

@ResponseStatus

@ResponseStatus permite aplicar a métodos y excepciones para definir el **status http** a devolver. A continuación se muestra un ejemplo aplicado a una excepción:

```
@ResponseStatus(code=HttpStatus.NOT_FOUND,
reason="Resource not found")
public class ResourceNotFoundException extends
RuntimeException{

 private static final long serialVersionUID =
8668589127062335507L;

}
```

En caso de que se arroje una ResourceNotFoundException se devolverá un status HTTP 404.

Otras clases útiles

Algunas otras clases útiles al trabajar con REST son:

-**ResponseEntity**: Permite devolver un contenido, **status** y **headers** HTTP.

-**HttpStatus**: Enumeración que contiene los **status** HTTP y su descripción.

-**ResponseStatusException**: En caso de que no se desee crear una excepción propia, es posible arrojar una ResponseStatusException para definir el status HTTP, ejemplo:

```
throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
"Resource not found");
```

Configuraciones útiles

A continuación se presentan algunas configuraciones útiles al trabajar con Spring MVC:

```
server.tomcat.threads.max=2
server.port=8081
server.servlet.context-path=/rest
```