**Prácticas Sesión 4.**

**Práctica 1.**

Usando como base el proyecto principios-rest creado con Spring Boot vamos a:

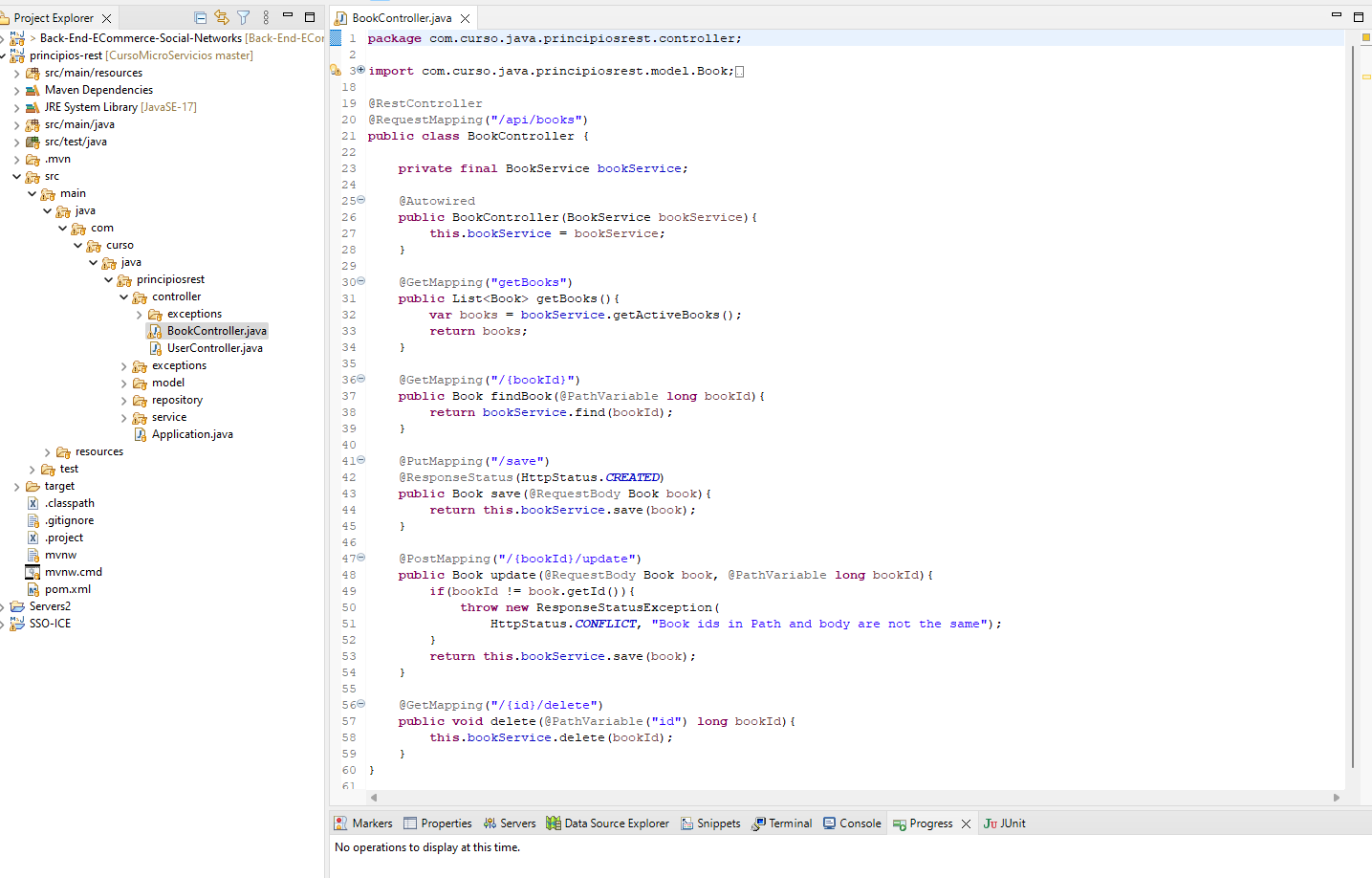
* Corregir algunos endpoints para que sigan los principios Rest establecidos.
  + Aplicar HTTP methods correctamente.
  + URL de endpoints según el estándar.
  + Status codes retornados según las mejores prácticas.
    - Si no devuelve nada, retornar 204
    - Si guarda un registro (como los post) retornar un 201 (Created)
    - Si no encuentra un registro retornar un 404

**Práctica 2.**

* Generar una documentación del API usando Open API/Swagger
  + Agregar las siguiente dependencia
* <dependency>
* <groupId>org.springdoc</groupId>
* <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
* <version>2.4.0</version>
* </dependency>
  + Acceder al URL <http://localhost:8080/swagger-ui/index.html>

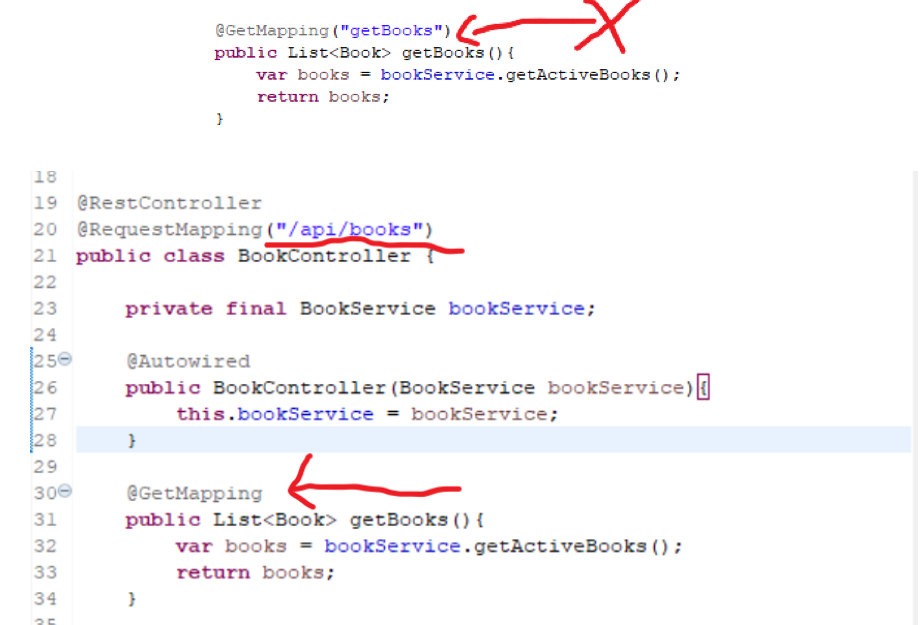
**Práctica Guíada.**

Abra el Proyecto de principios REST de la práctica en la sesión 4 o bajarlo del git que su profesor le comparta.



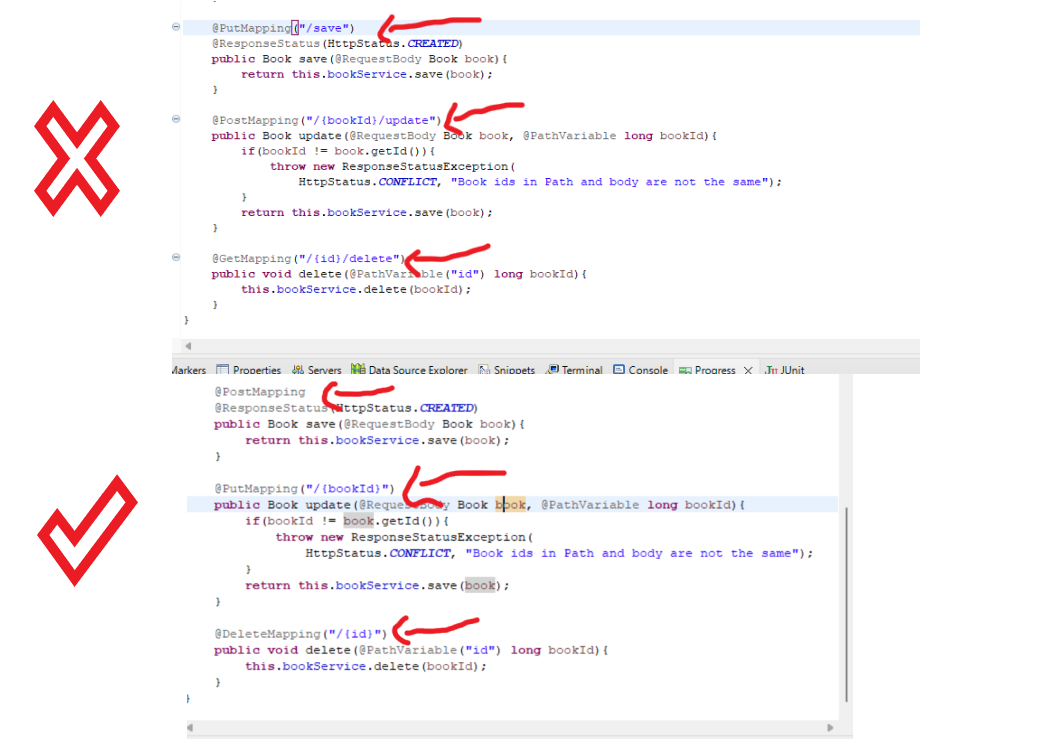
Observe detenidamente y de acuerdo a la parte teórica revise que puede mejorar para hacer más legibles y con las mejores prácticas los rest end points, la denominación y estructuración adecuadas de los recursos.

Primero con BookController.java :



No utilizar verbos para una operación como getBooks, ya la entidad tiene books en el request mapping, eliminar el getBooks dejando solo la anotación de @GetMapping como se muestra en la figura de abajo, al estar listando los libros y haber solo un get sin parámetros ya se toma como listado de Books el único get sin parámetros.

**BookController.**



Para el @PutMapping el ejemplo que está mal no requiere el /save ya que es una operación de update de CRUD donde solo se requiere el {bookId} y se sabe que la única operación que funciona con @PutMapping es el Update.

El @PostMapping no requiere ninguna nomenclatura o ruta ya que el @RequestBody donde viene todo el objeto Book para agregarse se le envía el método save del servicio inyectado en el controller.

Para el caso del @DeleteMapping solo necesitamos indicarle el {id} a eliminar, por lo tanto el /delete del método delete de arriba sobra.

**UserController.**



Podemos ver varios errores como poner verbos en los puntos finales : listBooks, remove, getUsers, getUser, update-user, etc.

Cuando se elimina un registro se puede devolver el código 204 o HttpStatus,NO\_CONTENT, para removerle a un usuario un libro solo se pone el {userId}, después en este caso /books/ y el id del libro : /{book\_id}.

ResponseEntity<Book> y poner el código de respuesta en lugar de HttpStatus.

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

@Autowired

public BookController(BookService bookService){

this.bookService = bookService;

}

@GetMapping

public List<Book> getBooks(){

var books = bookService.getActiveBooks();

return books;

}

@GetMapping("/{bookId}")

public **ResponseEntity<Book>** findBook(@PathVariable long bookId){

Book book = bookService.find(bookId);

**return ResponseEntity.ok(book);**

}

@PostMapping

public **ResponseEntity<Book>** save(@RequestBody Book book){

Book savedBook = this.bookService.save(book);

**return ResponseEntity.status(HttpStatus.CREATED).body(savedBook);**

}

@PutMapping("/{bookId}")

public **ResponseEntity<Book>** update(@RequestBody Book book, @PathVariable long bookId){

if(bookId != book.getId()){

throw new ResponseStatusException(

**HttpStatus.CONFLICT, "Book ids in Path and body are not the same");**

}

Book updatedBook = this.bookService.save(book);

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping("/{id}")

public **ResponseEntity<Void>** delete(@PathVariable("id") long bookId){

this.bookService.delete(bookId);

**return ResponseEntity.noContent().build();**

}

}

Resumen para operaciones básicas :

201 : Created : POST

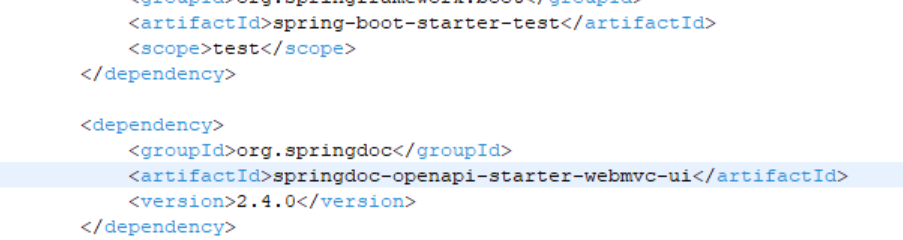
200 : OK : UPDATED

204 : NO\_CONTENT : DELETE

Normalmente para consultas se usa el HttpStatus.OK o ResponseEntity.OK.

Práctica 2.

Poner en las dependencias del archivo POM.xml está para la documentación de Swagger.



Hacemos un Maven Update del proyecto.

Entramos en esta dirección : <http://localhost:8080/swagger-ui/index.html>

