Spring Boot

Repaso módulo Spring Boot

// Conceptos básicos

Arquitectura Multicapa

Cuando creamos nuestro proyecto, la estructura debe representar la arquitectura multicapa elegida. Cada capa debe incluirse en un paquete específico, con el nombre de la misma.

Ojo: No existe un modelo único y dependiendo del proyecto tendremos más o menos capas.

Podemos incluir las siguientes capas:

- **Controller**: Se encarga de atender una request desde el momento en que es solicitada hasta la generación de la response y su transmisión.

  Lo que hace un controlador es llamar a una o más funciones de la capa de servicio. También gestiona la deserialización de una solicitud y la serialización de la respuesta, a través de la capa DTO.

  Cada clase de controlador debe estar marcada con la anotación **@RestController**.

  En cada Controller se especifican los **endpoint** a exponer al cliente.

- **DTO (Data Transfer Object)**: Se usan para transferir información. En Spring dentro del @RestController podemos retornar directamente un DTO y el framework se encarga de transformarlo a formato JSON.

  Ojo: Los DTOs son objetos planos, no deben tener lógica, solo se usan para enviar y recibir datos. No debemos olvidarnos los getters y setters (eso es mucho muy importante)

  

- **Service**: La capa de servicio es el lugar donde ubicamos cualquier operación de lógica de negocio que uno o más controladores necesiten.

Cada clase de servicio debe marcarse con la anotación **@Service**.

- **Repository**: Es la capa de persistencia de datos. Cada clase del repositorio debe estar marcada con la anotación **@Repository**.

  Esta capa se encarga de buscar en los datos guardados información para el service. Contiene funciones SQL, trabaja con colecciones de objetos. Por debajo podemos tener la capa DAO, que ofrece el crud jpa y trabaja con un objeto (a fines prácticos y por simplicidad para nosotrxs repository y dao serán sinónimos).

Estructura del proyecto

La estructura de un proyecto debería verse similar a la que se puede apreciar en la Imagen 1, donde en cada uno de esos paquetes crearemos las clases correspondientes.

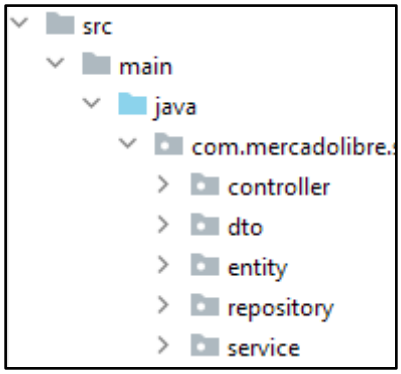

Imagen 1: Estructura del proyecto.

// Nociones a tener en cuenta

- Cada **endpoint** puede ser principalmente de los métodos GET, POST, PUT y DELETE (ver especificaciones mas detalladas de cuando utilizar cada uno). Dentro de Spring Boot se da el método especificado a través de las notaciones **@GetMapping, @PostMapping, @PutMapping y @DeleteMapping**.

- Para modelar recursos y asignaciones para diferentes verbos HTTP con Spring Boot, usamos la anotación **@RequestMapping**. Es aplicable a nivel de clase y a nivel de método, por lo que podemos construir nuestras API's de una manera sencilla. Para hacerlo aún más simple, el framework proporciona variantes como **@PostMapping**, **@GetMapping**, etc..

- Si queremos pasar el cuerpo de una solicitud (request) a un método dentro del controlador, usamos la anotación **@RequestBody**. Si usamos una clase personalizada, Spring Boot intentará deserializarla, utilizando el tipo pasado al método.

- La anotación **@RestController** le dice a Spring que este es un componente especializado modelado de un controlador REST. Es una combinación de @Controller y @ResponseBody, que le indica a Spring que coloque el resultado de los métodos como cuerpo de una respuesta HTTP.

- Por defecto en Spring Boot (y si no se indica lo contrario), la respuesta se serializará como **JSON** y se incluirá en el cuerpo de la respuesta (response).

- Spring **@Autowired** es una de las anotaciones más habituales cuando trabajamos con Spring ya que se trata de la anotación que permite inyectar unas dependencias con otras. Una vez que los objetos están creados, la anotación Spring **@Autowired** se encarga de construir las relaciones entre los distintos elementos.
- Cuando usamos la inyección de dependencias (por ejemplo con @Autowired) el tipo del atributo que vamos a inyectar va a ser una interfaz y no un tipo concreto que se haya implementado.

// Manejo de errores

- **@ControllerAdvice**: La anotación @ControllerAdvice nos permite consolidar múltiples @ExceptionHandlers dispersos de antes, en un solo componente global de manejo de errores.

- **@ExceptionHandler**: Esta anotación, como su nombre lo indica, proporciona una forma de definir métodos en un controlador en particular para manejar específicamente las excepciones lanzadas por dichos métodos.

- **@ResponseStatus:** vincula una excepción particular a un estado de respuesta HTTP específico. Entonces, cuando Spring detecta esa excepción en particular, genera una respuesta HTTP con la configuración definida allí. Marca un método o clase de excepción con el **código de estado** y el **mensaje** de motivo que debe devolverse. El código de estado se aplica a la respuesta HTTP cuando se invoca el método del controlador o siempre que se lanza la excepción especificada.