Tema 9

Dependencias en Spring Boot

Spring Boot permite el desarrollo de aplicaciones Java basadas en Spring.

1.Lombok

Lombok es una librería que reduce la cantidad de código repetitivo en Java

Anotaciones principales:

- @Getter y @Setter: Generan automáticamente los métodos getter y setter.
- @NoArgsConstructor y @AllArgsConstructor: Generan constructores sin y con argumentos.
- @Data: Combina @Getter, @Setter, @ToString, @EqualsAndHashCode y @RequiredArgsConstructor.

2.Spring Web

Spring Web proporciona herramientas para desarrollar **servicios web RESTful** en Spring Boot.

Anotaciones principales:

- @RestController: Indica que la clase es un controlador REST.
- @RequestMapping: Define rutas para los controladores.
- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping: Manejan solicitudes HTTP específicas.

3.Spring Data JPA

Spring Data JPA simplifica la interacción con bases de datos mediante el uso de **Java Persistence API**

Principales anotaciones:

- @Entity: Marca una clase como una entidad de base de datos.
- @Table(name = "nombre_tabla"): Define la tabla de la base de datos.
- @Id: Indica la clave primaria.
- @Column: Personaliza las propiedades de la columna en la BD.
- @GeneratedValue(strategy = GenerationType.IDENTITY): Genera automáticamente los valores de la clave primaria.
- @Repository: Marca una interfaz como un repositorio de JPA.

3.1. Configuración de la conexión a la BD en application.properties

Para conectar Spring Boot con una base de datos, se configuran estas propiedades:

```
# Conexión a la BD
spring.datasource.url=jdbc:mysql://localhost:3306/mi_base_de_datos
spring.datasource.username=usuario
spring.datasource.password=contraseña
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Configuración de Hibernate
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.j
+a.show-sql=true  # Muestra las consultas SQL en la consola
```

Opciones para `spring.jpa.hibernate.ddl-auto`

Con esta propiedad se gestionan las tablas en la BD.

None → No hace cambios en la BD.

Validate → Verifica que la BD coincide con las entidades, pero no modifica nada.

Update → Modifica la BD para adaptarse a las entidades sin borrar datos.

Create → Crea las tablas desde cero en cada inicia, en este caso se borran los datos.

Create-drop → Funciona como create pero borra las tablas al apagar la aplicación.

4.Hibernate

Hibernate es una **implementación de JPA** que gestiona la persistencia de datos en bases de datos relacionales.

Anotaciones Spring

Las anotaciones son esenciales para la configuración y el manejo de dependencias, lo que facilita la implementación de aplicaciones de manera eficiente y organizada.

1. @RestController

Esta anotación se utiliza para definir una clase como un controlador en una aplicación web de Spring Boot. Combina las funcionalidades de @Controller y @ResponseBody, lo que significa que los métodos dentro de la clase retornarán directamente datos en formato JSON o XML.

Ejemplo:

```
@RestController
public class MiControlador {
    @GetMapping("/mensaje")
    public String mensaje() {
        return "Hola, mundo!";
    }
}
```

2. @Autowired

Se usa para la inyección de dependencias en Spring. Permite que Spring resuelva automáticamente las dependencias necesarias en una clase sin necesidad de instanciarlas manualmente.

Ejemplo:

```
@Service
public class MiServicio {
    public String obtenerDato() {
        return "Dato importante";
    }
}

@RestController
public class MiControlador {
    @Autowired
    private MiServicio miServicio;

    @GetMapping("/dato")
    public String obtenerDato() {
        return miServicio.obtenerDato();
    }
}
```

3. @GetMapping

Se usa para manejar solicitudes HTTP GET en un controlador de Spring.

Ejemplo:

```
@GetMapping("/usuarios")
public List<Usuario> obtenerUsuarios() {
    return usuarioService.listarUsuarios();
}
```

4. @PathVariable

Permite extraer valores de la URL y usarlos como parámetros en un método.

Ejemplo:

```
@GetMapping("/usuarios/{id}")
public Usuario obtenerUsuario(@PathVariable Long id) {
    return usuarioService.buscarPorId(id);
}
```

5. @PostMapping

Maneja solicitudes HTTP POST, generalmente utilizadas para enviar datos al servidor.

Ejemplo:

```
@PostMapping("/usuarios")
public Usuario crearUsuario(@RequestBody Usuario usuario) {
    return usuarioService.guardar(usuario);
}
```

6. @RequestBody

Se usa para mapear el cuerpo de una solicitud HTTP en un objeto Java.

Ejemplo:

```
@PostMapping("/productos")
public Producto agregarProducto(@RequestBody Producto producto) {
    return productoService.agregar(producto);
}
```

7. @PutMapping

Maneja solicitudes HTTP PUT para actualizar recursos.

Ejemplo:

```
@PutMapping("/usuarios/{id}")
public Usuario actualizarUsuario(@PathVariable Long id, @RequestBody Usuario usuario) {
    return usuarioService.actualizar(id, usuario);
}
```

8. @DeleteMapping

Maneja solicitudes HTTP DELETE para eliminar recursos.

Ejemplo:

```
@DeleteMapping("/usuarios/{id}")
public void eliminarUsuario(@PathVariable Long id) {
    usuarioService.eliminar(id);
}
```

9. @Service

Marca una clase como un servicio dentro del patrón de diseño MVC en Spring. Se utiliza en la capa de lógica de negocio.

Ejemplo:

```
@Service
public class UsuarioService {
    public List<Usuario> listarUsuarios() {
        return usuarioRepository.findAll();
    }
}
```