Material descargable

egg

SPRING FRAMEWORK

Repasemos: Capa de Servicios

Los **servicios** son componentes especializados cuya función es gestionar la **lógica de negocio** de la aplicación. Se encargan de ejecutar las funcionalidades necesarias para que la aplicación cumpla con los requisitos del usuario.

En general, cada entidad debe contar con su propio **servicio**, el cual gestionará la lógica específica de dicho objeto o componente.

Consideraciones al crear un servicio

Al desarrollar un servicio, es importante tener en cuenta lo siguiente:

- **Declaración de la clase**
  - La clase debe estar marcada con la anotación `@Service`.
- **Importación de dependencias**
  - Es necesario importar la clase del repositorio correspondiente. En **VSCode**, la importación se realiza con:
    `import org.springframework.stereotype.Service;`
- **Definición de los métodos**
  - Se deben implementar los métodos que contienen la lógica de negocio.
  - También es recomendable incluir métodos básicos para **crear, eliminar, modificar o consultar información** de manera eficiente y precisa.
- **Gestión de dependencias:** Se deben importar todas las clases necesarias con: `import org.springframework…;`
  - Para acceder a los métodos del repositorio, es necesario instanciar un atributo de la clase repositorio correspondiente.
- **Inyección de dependencias**
  - Al declarar atributos en la clase, se utiliza la anotación `@Autowired` para que el servidor de aplicaciones se encargue de inicializar la variable automáticamente.

Algunas de las anotaciones a utilizar en nuestras entidades

ANOTACIÓN	USO	ADICIONAL
@Service	Declara una clase como servicio.	Requiere un atributo de la clase repositorio con la que se relaciona.
@Autowired	Permite la inyección automática de dependencias.	Se utiliza en atributos de tipo Repositorio para inicializarlos sin necesidad de instanciarlos manualmente.
@Transactional	Se usa antes de los métodos para manejar transacciones.	Si el método no lanza excepciones, se confirma (commit) la transacción. Para consultas, se debe agregar (readOnly = true).
@Async	Permite la ejecución de un método en paralelo.	Se usa, por ejemplo, para métodos que envían notificaciones.
@Lob @Basic	Permiten la carga de archivos en la base de datos.	Se recomienda establecer el parámetro fetch = FetchType.LAZY.

Métodos proporcionados por Spring Data JPA

Los repositorios de **Spring Data JPA** heredan métodos predefinidos de las interfaces `JpaRepository` y `CrudRepository`. Estos permiten realizar operaciones básicas de **CRUD** de manera eficiente.

Método	Descripción
`save()`	Guarda o actualiza una entidad en la base de datos.
`findById(id)`	Busca una entidad por su identificador único y devuelve un `Optional<T>`.

`getReferenceById(id)`	Alternativa a `findById()`, pero con diferencias en su comportamiento.
`findAll()`	Obtiene todas las entidades de un tipo determinado.
`deleteById(id)`	Elimina una entidad por su identificador único.
`count()`	Devuelve la cantidad total de entidades de un tipo.
`existsById(id)`	Verifica si una entidad con el identificador dado existe en la base de datos.

Uso de `Optional` en Spring Data JPA

La clase **`Optional`**, introducida en **Java 8**, es un contenedor que puede o no contener un valor no nulo. Su uso ayuda a evitar **NullPointerException** cuando se trabaja con valores que podrían ser nulos.

Ejemplo: Modificación de un autor en la base de datos

```
@Transactional
 public void modificarAutor(String nombre, String id){

     Optional<Autor> respuesta = autorRepositorio.findById(id);

     if (respuesta.isPresent()) {
         Autor autor = respuesta.get();
         autor.setNombre(nombre);
         autorRepositorio.save(autor);
     }
 }
```

Explicación del código

1. **Búsqueda del autor por ID**
   - Se usa `findById(id)`, que retorna un `Optional<Autor>`.
   - Si el autor existe en la base de datos, `Optional` contendrá su valor; de lo contrario, estará vacío.
2. **Verificación de existencia**
   - `isPresent()` verifica si el `Optional` contiene un autor.
   - Si el autor existe, se obtiene su instancia con `get()`.
3. **Modificación y guardado**
   - Se actualiza el **nombre** del autor y se guarda con `save()`.

El uso de **servicios** permite una mejor organización del código al separar la lógica de negocio de la lógica de persistencia. Además, la utilización de **Optional** en los repositorios ayuda a manejar de manera segura los valores nulos, evitando excepciones innecesarias.

Para más detalles sobre `Optional`, puedes consultar la **documentación oficial de Java**.