Material descargable

egg

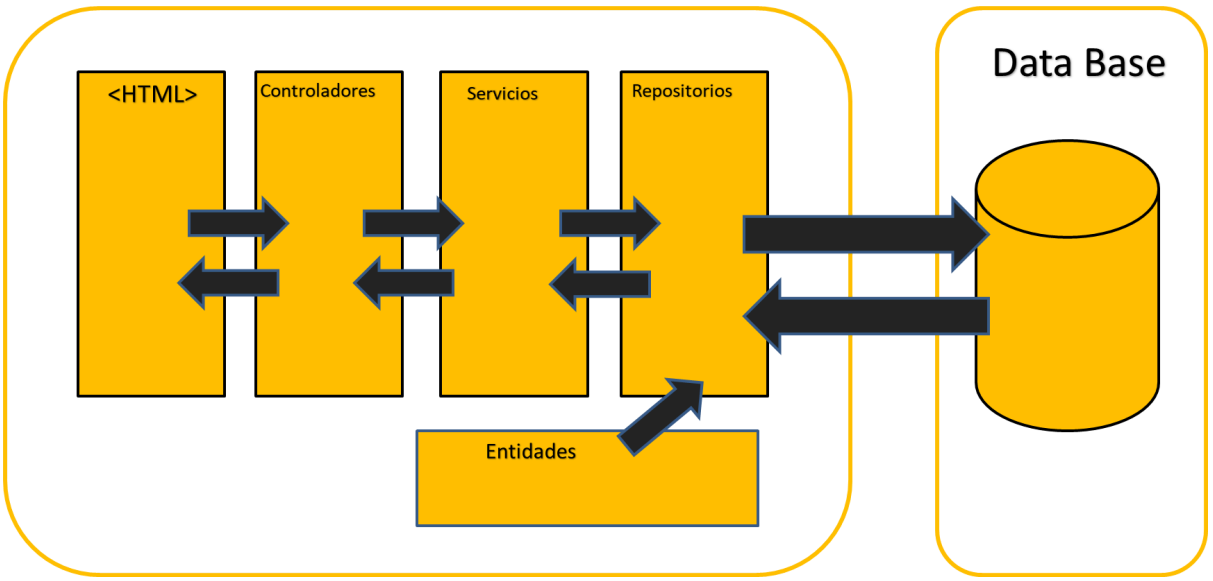
SPRING FRAMEWORK

Arquitectura en Capas: Organizando la Lógica del Software

La **arquitectura en capas** es un enfoque fundamental en el desarrollo de software que estructura el código en distintos niveles de abstracción, cada uno con responsabilidades claramente definidas.

Capas de una Aplicación

- **Capa de Interfaz de Usuario:** Es la capa superior donde residen los componentes que interactúan directamente con el usuario. Aquí se encuentran los elementos **HTML**, interfaces gráficas y cualquier otro elemento visible para el usuario final.
- **Capa de Comunicación (o de Control):** Gestiona la comunicación entre las capas superiores e inferiores. Aquí intervienen los **controladores**, que actúan como intermediarios entre la interfaz de usuario y el resto del sistema. Su función principal es recibir las solicitudes del usuario, invocar los servicios necesarios y devolver una respuesta adecuada.
- **Capa de Servicios:** Contiene la **lógica de negocio** de la aplicación. Define reglas y operaciones que determinan el comportamiento del sistema. Los servicios encapsulan esta lógica y coordinan diferentes componentes de la aplicación.
- **Capa de Acceso a Datos:** Maneja la interacción con la **base de datos**. Aquí se encuentran las **entidades**, que representan las tablas de la base de datos en forma de objetos, y los **repositorios**, que proporcionan métodos para realizar operaciones **CRUD** (Crear, Leer, Actualizar, Eliminar).



Ventajas de la Arquitectura en Capas

- **Separación de preocupaciones**: Cada capa tiene una responsabilidad específica, lo que facilita la comprensión y mantenimiento del código.
- **Modularidad**: Se pueden desarrollar, probar y desplegar capas de forma independiente, mejorando la flexibilidad y escalabilidad.
- **Reutilización de código**: Facilita la reutilización de componentes y la interoperabilidad dentro del sistema.
- **Escalabilidad**: Se pueden escalar capas específicas según sea necesario para mejorar el rendimiento.
- **Desarrollo colaborativo**: Permite que diferentes equipos trabajen en áreas específicas sin afectar otras partes del sistema.

Spring y el Uso de Anotaciones

Spring implementa la arquitectura en capas mediante el patrón **Modelo-Vista-Controlador (MVC)** y el uso de **anotaciones**, que definen el comportamiento y funcionalidad de los componentes.

Las anotaciones deben declararse antes del **atributo, método o clase** al que afectan.

Ejemplo de anotación en una clase

```
7
8   @Repository
9   public interface EditorialRepositorio extends JpaRepository<Editorial,String> {
0
1   }
2
```

Aquí, se indica que la interfaz **es un repositorio**, lo que permite que Spring maneje automáticamente las operaciones sobre la base de datos.

Ejemplo de anotación en un atributo:

```
@Id
private Long isbn;
```

En este caso, se especifica que el atributo **isbn** funcionará como el identificador (**id**) de la tabla.

Capa de Acceso a Datos en Spring

Las **entidades y repositorios** permiten transformar el modelo relacional de la base de datos en un modelo de objetos dentro de la aplicación.

Entidades en Spring

Las **entidades** modelan los objetos del dominio y utilizan anotaciones de **JPA** para definir su comportamiento en la base de datos.

Consideraciones al crear entidades

- Declarar la clase con la anotación `@Entity`.
- Importar la unidad de persistencia (`jakarta.persistence.*`).
- Definir los atributos de la entidad.
- Utilizar las anotaciones necesarias para mapear correctamente los atributos.
- Implementar un **constructor vacío** y los **getters/setters**.

Anotaciones comunes en entidades

ANOTACIÓN	USO	ADICIONAL
@Entity	Declara una clase como entidad (Futura tabla)	@Table(name="loqquiera") {Si no lo uso, por defecto la tabla se va a llamar como la clase Entity} Esta anotación es de JPA.
@Id	Declara al atributo como Identificador. Debe ser único.	Obligatorio en cada entidad.
@GeneratedValue	Declara que el atributo va a ser "autogenerado"	Se puede usar IDENTITY, SEQUENCE, AUTO, TABLE.
@GenericGenerator	Para declarar la estrategia de generar el Id	
@Enumerated	Mapea al atributo con un objeto del tipo Enum.	Entre () debo escribir el criterio con el que va a usar el dato: • (EnumType.STRING) • (EnumType.ORDINAL)
@ManyToOne **@OneToOne** **@ManyToMany** **@OneToMany**	Declara que hay relaciones con el atributo. El atributo es del tipo "entidad"(otra clase)Prestar atención a la cardinalidad de la relación.	En conveniente poner @JoinColumn(name ="id") luego de anunciar la relación, para evitar "tablas intermedias".
@Inheritance	Anotación a utilizar en clase padre abstracta si implemento herencia	@Entity @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS) __Debo dejar marcada la estrategia de comportamiento
@Temporal	Todo dato que sea del tipo DATE debe utilizar esta anotación	Entre () debo escribir el criterio con el que va a usar el dato: • (TemporalType.TIME) • (TemporalType.DATE) • (TemporalType.TIMESTAMP)

Consideraciones al usar `enum` en entidades

- Se recomienda agrupar las clases `enum` en un paquete separado.

- No llevan anotaciones en su declaración.
- En la entidad, se deben mapear con `@Enumerated(EnumType.STRING)`.

```
public enum Turno{
    DIA,NOCHE
}
```

Repositorios en Spring

Los **repositorios** actúan como interfaces entre el modelo de objetos y la base de datos. Son responsables de **crear, modificar, eliminar y buscar objetos** del dominio.

Spring Data JPA simplifica el acceso a la base de datos proporcionando una implementación automática para las interfaces que extienden `JpaRepository`.

```
7   @Repository
8   public interface EditorialRepositorio extends JpaRepository<Editorial,String> {
9
10  }
11
```

Aquí, `EditorialRepository` gestiona la entidad `Editorial`, cuya clave primaria es de tipo `String`.

Métodos Personalizados en Repositorios

Si se requiere una consulta específica, se pueden definir métodos adicionales dentro del repositorio.

```
@Query("SELECT l FROM Libro l WHERE l.titulo = :titulo")
public Libro buscarPorTitulo(@Param("titulo") String titulo);
```

Consideraciones sobre las Queries en Repositorios

- Si una consulta tiene errores de sintaxis, la aplicación no compilará correctamente.

- Es importante verificar que los nombres de las columnas coincidan con los declarados en la entidad.
- Se pueden definir métodos adicionales según las necesidades del sistema.

Anotaciones comunes en repositorios

ANOTACIÓN	USO	ADICIONAL
@Repository	Declara una clase como repositorio. Es una interface,no una clase	Debemos indicar de quien extiende y manipula: ***extends JpaRepository<Clase,TipodatoID>***
@Query	Utilizado antes de un método que realizará una consulta en una clase Repository ,incluye la consulta en la base de datos	@Query(" La consulta con formato MYSQL") La consulta debe estar entre ""
@Param	Utilizado en los métodos declarados previamente como QUERY	@Param(" nombrecolumna") Tipo de dato + nombre variable.

Spring proporciona un conjunto de herramientas poderosas para la **persistencia de datos**, facilitando la interacción con bases de datos de manera eficiente.

Para más información, consulta la documentación oficial de Spring Data: 🔗 **Spring Data JPA Documentation**