¿Qué diferencia hay entre @Component, @Service y @Repository?

@Component es el estereotipo principal el cual indica que es una clase con anotación component o Bean de Spring, por tanto @Repository(Acceso a BD) y @Service(Capa con lógica de negocio) son especializaciones de @Component para casos concretos, el cual también puede anotar los beans con la anotación de @Component, pero al anotarlas con @Repository, @Service o @Controller se obtienen beneficios adicionales.

@Service

```java
@Service
public class UserServiceImpl implements UserService{

    @Autowired
    private UserRepository repository;
```

@Repository

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long>{

    boolean existsByEmail(String email);
}
```

¿Cómo se implementa la inyección de dependencias en Spring?

La utilización de inyección de dependencias en Spring es muy importante, debido que esta permite desacoplar las clases y delegar diferentes responsabilidades a sus debidas dependencias necesarias, actualmente tenemos 3 formas de realizarlo:

- Inyección por Setter

```java
public class SimpleMovieLister {

    // the SimpleMovieLister has a dependency on the MovieFinder
    private MovieFinder movieFinder;

    // a setter method so that the Spring container can inject a MovieFinder
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // business logic that actually uses the injected MovieFinder is omitted...
}
```

- Inyección mediante @Autowired

```
@Service
public class UserServiceImpl implements UserService{

    @Autowired
    private UserRepository repository;

    @Autowired
    private UsuarioMapper mapper;
```

- Inyección de dependencia basada en constructor

```
private final UserServiceImpl userServiceImpl;


// Constructor para inyectar UserServiceImpl
public UserController(UserServiceImpl userServiceImpl) {
    this.userServiceImpl = userServiceImpl;
}
```

¿Qué es el ciclo de vida de un Bean en Spring?

Se conoce el ciclo de vida de un Bean la representación de las etapas que Spring realiza, desde la creación de un objeto hasta que este se destruye, a la vez nos permite la personalización de los comportamientos para su creación, uso y finalmente su destrucción, lo que permite que las aplicaciones sean modulares y reduce el acoplamiento estrecho entre componentes.

Etapas:

- Instanciación
- Inyecciones de dependencias
- Inicialización
- Utilización de Bean
- Destrucción

¿Qué diferencia hay entre un DTO y un Entity?

Las entidades en Spring se utilizan para la representación de datos en BD gestionada por JPA/Hibernate y está directamente relacionada con el modelo de persistencia, pero mientras tanto los DTO son utilizados para la representación y transporte de datos tanto enviados como los recibidos de un cliente, para el funcionamiento correcto en la asignación de datos entre una entidad y un DTO, será importante y necesario la definición de atributos. Reconocer la diferencia entre un DTO y una Entity es fundamental en el desarrollo con Spring Boot, principalmente cuando trabajamos con capas (como controller, service, repository).

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "usuarios")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String nombre;

    @Column(nullable = false, unique = true)
    private String email;

    @Enumerated(EnumType.STRING)
    private Rol rol;

    private LocalDateTime fechaRegistro = LocalDateTime.now();

}
```

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserCreateDto {

    @NotBlank
    private String nombre;

    @Email
    @NotBlank
    private String email;

    @NotNull
    private Rol rol;

}
```

```java
@Data
public class UserDto {

    private Long id;
    private String nombre;
    private String email;
    private String rol;
    private LocalDateTime fechaRegistro;

}
```

¿Cómo se maneja la paginación en Spring Data JPA?

La paginación en Spring se puede implementar de una manera muy fácil, mediante una clase Page<T>, el cual este nos devolverá un método findAll(), esto nos permitirá obtener resultados directamente desde nuestros métodos de repositorios sin necesidad de realizar una lógica manual.

Repositorio

git clone https://github.com/Jairo090101/Prueba_SpringBoot