Spring framework 5 -Spring data

Conceptos básicos

Todos los *Repository* reciben 2 genéricos:

-T : Representa la entidad JPA
-ID: Representa el tipo de dato de la llave primaria

El *Repository* más general es **CRUDRepository** seguido de **PagingAndSortingRepository** y al final se encuentra **JPARepository**, cuando no se necesite la funcionalidad completa de **JPARepository** o **PagingAndSortingRepository** es posible solo utilizar **CRUDRepository.**

CRUDRepository

La interfaz **CRUDRepository** contiene la funcionalidad básica para realizar un CRUD:

-**save**(): Guarda uno o varios objetos.

-**findOne**(): Busca un elemento basado en su llave primaria.

-**findAll**(): Devuelve un iterable de todos los registros de la base de datos.

-**count**(): Devuelve el conteo de los registros de la base de datos.

-**delete**(): Borra el elemento recibido como parámetro.

-**exists**(): Verifica si existe un elemento correspondiente a la llave primaria especificada.

PagingAndSortingRepository

PagingAndSortingRepository provee toda la funcionalidad de **CRUDRepository** más los siguientes métodos:

-**findAll**(Sort sort): Devuelve todas las entidades ordenadas por la opción especificada.

-**findAll**(Pageable p): Devuelve todas las entidades de acuerdo a las opciones de paginación especificadas.

JPARepository

JPARepository provee toda la funcionalidad de **PagingAndSortingRepository** más los siguientes métodos:

-List<T> **findAll**(): Parecido al definido en CRUDRepository pero este método devuelve un objeto **List** en lugar de un **Iterable.**

-**findAll**(Sort sort): Parecido al definido en CRUDRepository pero ordena los elementos de acuerdo al objeto *Sort.*

-**flush**(): Hace *flush* de todas las tareas pendientes a la base de datos.

-**saveAndFlush**(): Guarda las entidades y hace *flush* inmediatamente.

-**deleteInBatch**(): Borra una lista de entidades en modo *batch.*

Query Methods

En caso de que la operación que se desea realizar no se encuentre disponible en los métodos heredados del repositorio, se deberá definir un método como los siguientes:

-find...By...() :
-read...By...() :
-query...By...() :
-count...By...() :
-get...By()... :

Es posible agregar cláusulas como *distinct*, condiciones que incluyan **And** y **Or**. **By** actua como delimitador para especificar el campo.

Ejemplos de *Query Methods*

A continuación se muestran algunos ejemplos:

```
public interface PersonRepository extends Repository<Person, Long> {

    List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

    List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);

    List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

    List<Person> findByLastnameIgnoreCase(String lastname);

    List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

    List<Person> findByLastnameOrderByFirstnameAsc(String lastname);

    List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

No es necesario implementar los métodos, al seguir la convención de nombres, Spring Data creará la implementación por nosotros.

Query Methods

Notas importantes:

-Las operaciones se pueden concatenar, así que se pueden utilizar múltiples AND y OR. También esta soportado el uso de operadores como **Between**, **LessThan**, **GreaterThan** y **Like**.

-Los métodos soportan IgnoreCase para propiedades individuales **findByLastnameIgnoreCase**()

-Es posible aplicar un ordenamiento agregando **OrderBy**. En ese caso, es posible incluir **Asc** o **Desc**.

Limitando los resultados

Es posible limitar el número de resultados que se obtendrán utilizando *first* y *top*. A continuación se muestran algunos ejemplos:

```
public interface UserRepository extends CrudRepository<User, Integer>  {
    User findFirstByOrderByLastnameAsc();
    User findTopByOrderByAgeDesc();
    Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);
    Slice<User> findTop3ByLastname(String lastname, Pageable pageable);
    List<User> findFirst10ByLastname(String lastname, Sort sort);
    List<User> findTop10ByLastname(String lastname, Pageable pageable);
}
```

Los métodos mostrarán solo los resultados especificados en el método.

Custom queries

Si se desea ejecutar un *query* que no esté soportado por los *query methods,* es posible utilizar la anotación **@Query**. A continuación se muestra un ejemplo:

```
public interface UserRepository extends CrudRepository<User, Integer> {
    @Query("SELECT u.username FROM User u ")
    public List<String> findAllUserNames();

}
```

DEVS4J
Hecho por Devs para Devs

Paginación

Para habilitar paginación, nuestro *Repository* debe heredar de **PagingAndSortingRepository** o de **JPARepository,** como se muestra a continuación:

```
public interface UserRepository extends JpaRepository<User, Integer> { m
}
```

Una vez hecho esto, se deben ejecutar los métodos que reciben un objeto de tipo **Pageable** y que devuelvan uno de tipo **Page**<T> como el siguiente:

```
Page<T> findAll(Pageable pageable)
```

Para utilizarlo, crearemos el objeto de tipo *Pageable* del siguiente modo:

```
Page<User> users = userRepository.findAll(PageRequest.of(1, 10));
```

El método anterior busca todos los usuarios y solicita la primera página con un tamaño de 10 registros.

Ordenamiento

Es posible determinar el ordenamiento a través de un parámetro de tipo **Sort** definido en **PagingAndSortingRepository:**

```
Iterable<T> findAll(Sort sort);
```

Para utilizarlo, crearemos el objeto de tipo *Sort* del siguiente modo:

```
List<User> users = userRepository.findAll(Sort.by(Sort.Direction.ASC,"username"));
```

Como se puede ver, es posible determinar si el orden será de forma ascendente o descendente, además de una lista de parámetros.

Ordenamiento y paginación

Para realizar paginación y ordenamiento utilizaremos el método que recibe el objeto de tipo *Pageable* del siguiente modo:

```
Page<User> users = userRepository.findAll(PageRequest.of(1, 10, Sort.Direction.ASC, "username"));
```

El método *of* de la clase **PageRequest** está sobrecargado para recibir tanto los parámetros de paginación como el ordenamiento.

Notas

Spring data no solo soporta JPA, también se puede utilizar con:

-LDAP	-REST	-CouchBase
-MongoDB	-Cassandra	-ElasticSearch
-Redis	-Solr	-etc.