1. Spring-Data

L'extension "**Spring-Data**" permet (entre autre) de :

- **générer automatiquement des composants "DAO / Repository" modernes** (utilisables avec des technologies SQL , NO-SQL ou orientées graphes telles que JPA , MongoDB , Neo4J)
- accélérer le temps de développement (l'interface suffit souvent, la classe d'implémentation sera générée dynamiquement par introspection et selon certaines conventions).
- standardiser le format des composants "DAO/Repository" : mêmes méthodes fondamentales.
  On parle alors en termes de "composants DAO consistants" → des automatismes sont possibles (tests en partie automatique , ….) .

1.1. Spring-data-commons

"**Spring-data-commons**" est la partie centrale de Spring-data sur laquelle pourra se greffer certaines extensions (pour jpa , pour mongo , …) .

"Spring-data-commons" est essentiellement constituée de **3 interface**s : **Repository** , **CrudRepository** et **PagingAndSortingRepository** .

- **Repository<T,ID>** n'est qu'une interface de marquage dont toutes les autres héritent.
- **CrudRepository<T,ID>** standardise les méthodes fondamentales (findOne , findAll , save , delete, …)
- **PagingAndSortingRepository<T,ID>** étend CrudRepository en ajoutant des méthodes supportant le tri et la pagination.

Méthodes fondamentales de **CrudRepository**<T ,ID extends Serializable> :

`<S extends T> S save(S entity);`	Sauvegarde l'entité (au sens saveOrUpdate) et retourne l'entité (éventuellement ajustée/modifiée dans le cas d'une auto-incrémentation ou autre).
`T findOne(ID primaryKey);`	Recherche par clef primaire
`Iterable<T> findAll();`	Recherche toutes les entités (du type courant/considéré)
`Long count();`	Retourne le nombre d'entités existantes

`void delete(T entity);`
 	
`void delete(ID primaryKey);`
 	
`void deleteAll();`
	Supprime une (ou plusieurs) entités

`boolean exists(ID primaryKey);`	Test l'existence d'une entité

Variantes de quelques méthodes (surchargées) au sein de CrudRepository :

`<S extends T> Iterable<S> save(Iterable<S> entities);`	Sauvegarde une liste d'entités
`Iterable<T> findAll(Iterable<ID> ids );`	Recherche toutes les entités (du type courant/considéré) ayant les Ids demandés
`void delete(Iterable< ? Extends T> entities)`	Supprime une liste d'entités

Rappel : java.util.**Collection**<E> et java.util.**List**<E> héritent de **Iterable**<E>

Fonctionnalité "**tri**" apportée en plus par l'interface **PagingAndSortingRepository** :

```
...
   Iterable<Personne> personnesTrouvees =
   personnePaginationRep.findAll(new Sort(Sort.Direction.DESC, "nom"));
   …
```

où **org.springframework.data.domain.Sort** est spécifique à Spring-data .

Fonctionnalité "**pagination**" apportée en plus par l'interface **PagingAndSortingRepository** :

```
public void testPagination() {
   assertEquals(10, personnePaginationRep.count());
   Page<Personne> pageDePersonnes =
   // 1re page de résultats et 3 résultats max.
   personnePaginationRep.findAll(new PageRequest(1, 3));
   assertEquals(1, pageDePersonnes.getNumber());
   assertEquals(3, pageDePersonnes.getSize());// la taille d'une page
   assertEquals(10, pageDePersonnes.getTotalElements());
   assertEquals(4, pageDePersonnes.getTotalPages());
   assertTrue(pageDePersonnes .hasContent());
   ...
  }
```

Avec comme types précis :

org.springframework.data.domain.Page<T>

et **org.springframework.data.domain.PageRequest** implémentant l'interface **org.springframework.data.domain.Pageable**

1.2. Spring-data-jpa

Dépendance maven :

```
<dependencies>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
  </dependency>
<dependencies>
```

Exemple de version : **1.12.4.RELEASE**

Activation en xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jpa="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

  <jpa:repositories  base-package="com.acme.repositories" />

</beans>
```

Activation en java-config :

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories
...
class Config {}
```

Exemple d'interface de DAO/Jpa avec **JpaRepository** :

```
interface UserRepository extends JpaRepository<User, Long> {
   List<User> findByLastname(String lastname);
}
```

La classe d'implémentation sera générée automatiquement (si *@EnableJpaRepositories* ou si *<jpa:repositories base-package="..." />*)

il suffit d'une injection via **@Autowired** ou **@Inject** pour accéder au composant DAO généré .

Conventions de noms sur les méthodes de l'interface :

`find`…By, `read`…By, `query`…By, `get`…By and `count`…By,

Exemples :

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

// Enables the distinct flag for the query
List<User> **findDistinctPeopleByLastnameOrFirstname**(String lastname, String firstname);
List<User> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

// Enabling ignoring case for an individual property
List<User> **findByLastnameIgnoreCase**(String lastname);

// Enabling ignoring case for all suitable properties
List<User> **findByLastnameAndFirstnameAllIgnoreCase**(String lastname, String firstname);

// Enabling static ORDER BY for a query
List<User> **findByLastnameOrderByFirstnameAsc**(String lastname);
List<User> **findByLastnameOrderByFirstnameDesc**(String lastname);

methodNameWithKeyWords(?1,$2,...)

Keyword	Sample	JPQL snippet
And	findByLastname**And**Firstname	… where x.lastname = ?1 **and** x.firstname = ?2
Or	findByLastname**Or**Firstname	… where x.lastname = ?1 **or** x.firstname = ?2
Is,Equals	FindByFirstname, findByFirstname**Is**, findByFirstname**Equals**	… where x.firstname **=** ?1
Between	findByStartDate**Between**	… where x.startDate **between** ?1 **and** ?2
LessThan	findByAge**LessThan**	… where x.age **<** ?1
LessThanEqual	findByAge**LessThanEqual**	… where x.age **<=** ?1
GreaterThan	findByAge**GreaterThan**	… where x.age **>** ?1
GreaterThanEqual	findByAge**GreaterThanEqual**	… where x.age **>=** ?1
After	findByStartDate**After**	… where x.startDate **>** ?1
Before	findByStartDate**Before**	… where x.startDate **<** ?1

Keyword	Sample	JPQL snippet
IsNull	findByAge**IsNull**	… where x.age **is null**
IsNotNull, NotNull	findByAge(Is)**NotNull**	… where x.age **not null**
Like	findByFirstname**Like**	… where x.firstname **like** ?1
NotLike	findByFirstname**NotLike**	… where x.firstname **not like** ?1
StartingWith	findByFirstname**StartingWith**	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstname**EndingWith**	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstname**Containing**	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAge**OrderBy**Lastname**Desc**	… where x.age = ?1 **order by** x.lastname **desc**
Not	findByLastname**Not**	… where x.lastname **<>** ?1
In	findByAge**In**(Collection<Age> ages)	… where x.age **in** ?1
NotIn	findByAge**NotIn**(Collection<Age> age)	… where x.age **not in** ?1
True	findByActive**True**()	… where x.active **= true**
False	findByActive**False**()	… where x.active **= false**
IgnoreCase	findByFirstname**IgnoreCase**	… where UPPER(x.firstame) = UPPER(?1)

Paramétrage par défaut de JpaRepositories :

CREATE_IF_NOT_FOUND (default) combines CREATE and USE_DECLARED_QUERY

→ **on peut donc éventuellement personnaliser l'implémentation des méthodes.**

Utilisation de @NamedQuery à coté de @Entity (ou <named-query ...> dans orm.xml) :

Dans orm.xml (référencé par META-INF/persistence.xml ou …) :

```
<named-query name="User.findByLastname">
  <query>select u from User u where u.lastname = ?1</query>
</named-query>
```

et/ou dans la classe d'entité persistante :

```
@Entity
@NamedQuery(name = "User.findByEmailAddress",
  query = "select u from User u where u.emailAddress = ?1")
public class User {

}
```

```
public interface UserRepository extends JpaRepository<User, Long>
{
  List<User> findByLastname(String lastname);

  User findByEmailAddress(String emailAddress);
}
```

Utilisation (un peu radicale) de @Query (de Spring Data) dans l'interface :

Sémantiquement peu être un trop peu radical pour une interface !!!

Exemple :

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

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}
```

==> et encore beaucoup d'autres possibilités / options dans la **doc de référence de spring-data** .

1.3. Spring-data-mongo

1.4. Spring-data-Neo4J