

# Spring Data JPA

---

adriencaubel.fr

# Table des matières

1. Introduction
  1. Rappels
2. Présentation de Spring Data JPA
  1. Son objectif
  2. Une couche d'abstraction
3. Initialiser un projet avec Spring Data JPA
  1. Dépendance Maven
  2. Le fichier application.properties
4. L'interface Repository
  1. L'interface Repository
  2. Créer notre Repository
5. Derived Query Methods
  1. Objectifs
  2. Respecter les conventions
  3. Depuis @Query
6. Pagination

# Introduction

# Rappels

- Nous avons étudié les concepts suivants
  - La notion `@Entity`
  - Les associations `@OneToMany` ... et `mappedBy` (bidirectionnelle)
  - L'héritage `@Inheritance`
  - L'optimisation des lectures ( `Fetch.LAZY` / `Fetch.EAGER` )
  - L'API Criteria
  - Les projections

# Présentation de Spring Data JPA

# Son objectif

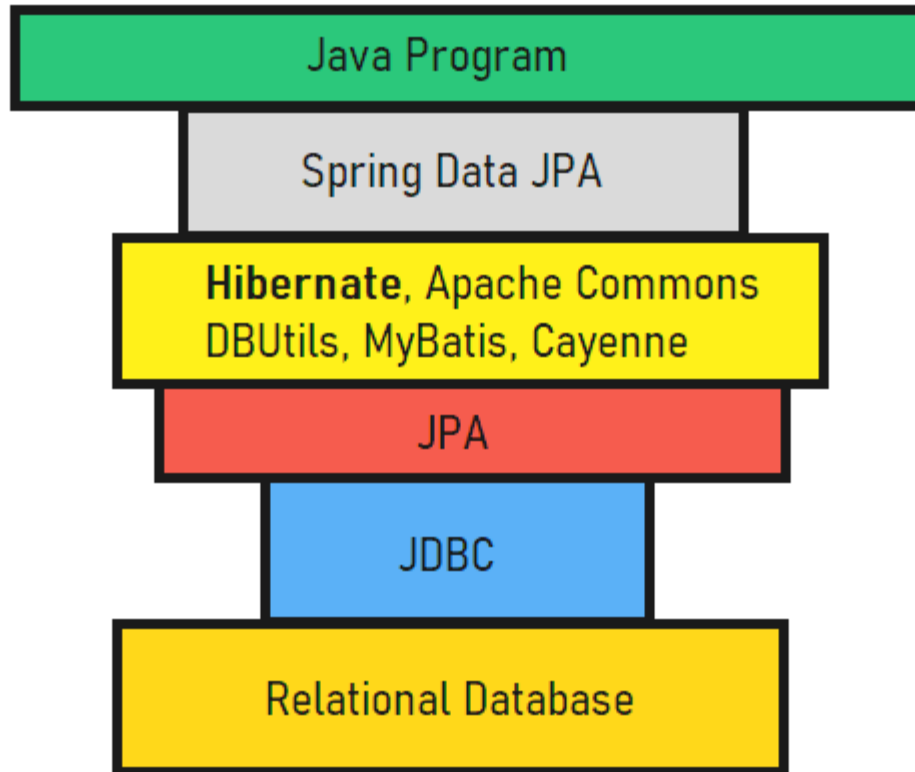
## 🚫 Objectif

Implementing a data access layer for an application can be quite cumbersome. Too much boilerplate code has to be written to execute the simplest queries. Add things like pagination, auditing, and other often-needed options, and you end up lost.

Spring Data JPA est une couche d'abstraction :

- Pagination et tri intégrés
- Requêtes avancées via JPQL, SQL natif, Criteria API et Specifications
- Gestion des transactions automatique

# Une couche d'abstraction



# Initialiser un projet avec Spring Data JPA

# Dépendance Maven

Dans un projet Maven, pour utiliser Spring Data pour une base de données relationnelles avec JPA, il faut déclarer la dépendance suivante

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>4.0.2</version>
</dependency>
```

Et préciser un le driver utilisé

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
</dependency>
```

## Conséquences

- Embarque automatiquement la dépendance `hibernate-core`
- configuration de la base de données dans `application.properties`

# Le fichier application.properties

## ⓘ application.properties

In a Spring Boot application, the application.properties file is a key configuration file that helps to customize various aspects of the application. It is typically located in the src/main/resources directory of your project. This file allows you to set properties that configure the behavior of your Spring Boot application, such as database connections, server settings, logging levels and more.

## Pour la base de données

```
spring.datasource.url=jdbc:mysql://localhost:3306/app
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=update
```

```
# Afficher les logs
spring.jpa.show-sql=true
```

```
# Formater le SQL (plus lisible)
spring.jpa.properties.hibernate.format_sql=true
```

```
# Afficher les valeurs des paramètres (?)
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.orm.jdbc.bind=TRACE
```

# L'interface Repository

# L'interface Repository

```
public interface CrudRepository<T, ID>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);
    Optional<T> findById(ID primaryKey);
    Iterable<T> findAll();
    long count();
    void delete(T entity);
    boolean existsById(ID primaryKey);

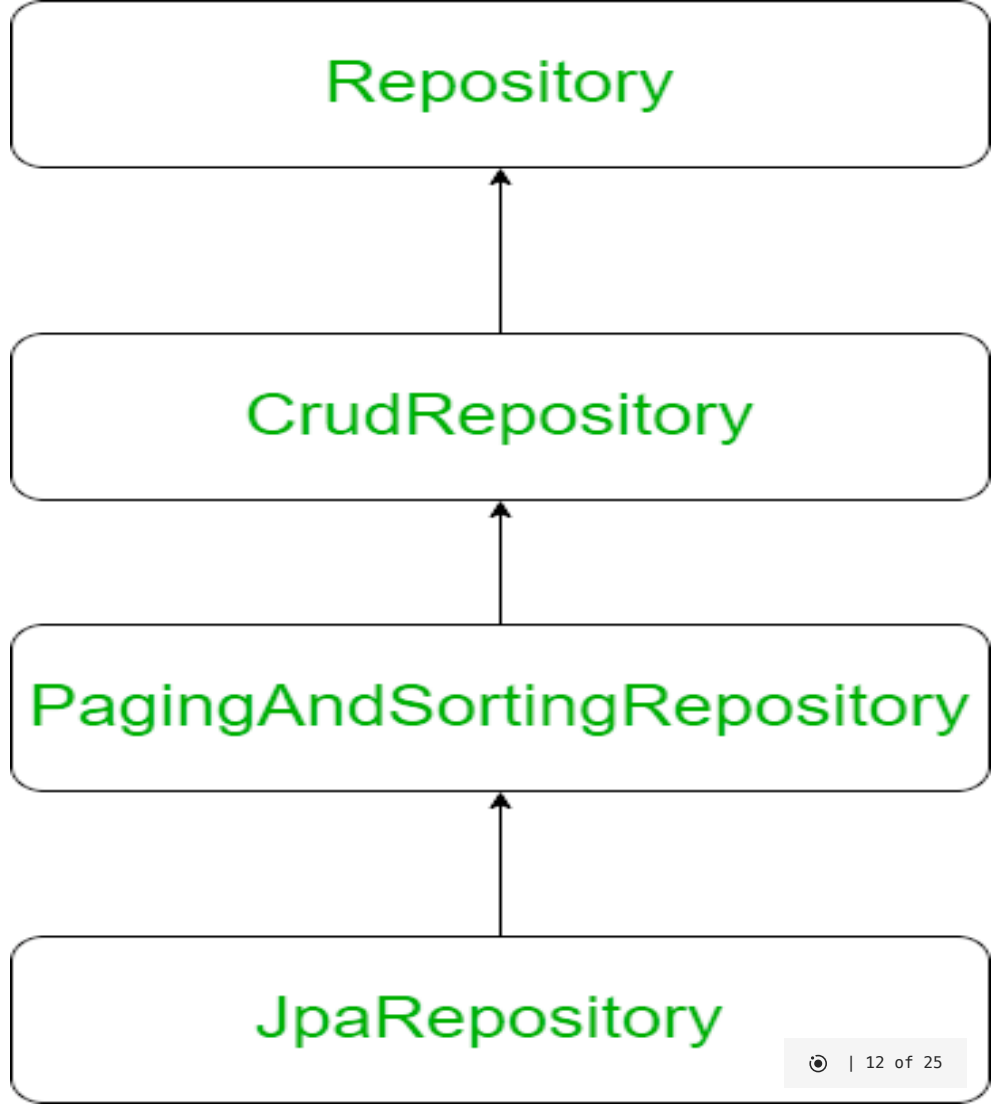
    // ... more functionality omitted.
}
```

En plus de `CrudRepository` pour la pagination nous avons `PagingAndSortingRepository`

```
public interface PagingAndSortingRepository<T, ID>

    Iterable<T> findAll(Sort sort);

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



# Créer notre Repository

```
public interface StudentRepository extends JpaRepository<Student, Long> {  
    // pas besoin de coder les opérations de CRUD  
}
```

Nous n'avons pas besoin de coder les opérations de CRUD, elles sont héritées. Mais une application a souvent besoin de requêtes supplémentaires.

# Derived Query Methods

# Objectifs

<https://docs.spring.io/spring-data/jpa/reference/repositories/query-methods-details.html>

Spring va traduire le nom de la méthodes en requête SQL. Supposons une classe `Produit` qui dispose du champs `nom` et puis nous définissons la méthode suivante

```
public interface ProduitRepository extends JpaRepository<Produit, Long> {  
    List<Produit> findByNom(String nom);  
}
```

Spring Data JPA implémente automatiquement cette méthode sans aucune écriture SQL manuelle en

```
SELECT * FROM produit WHERE nom = :nom
```

# Respecter les conventions

Nous devons juste prêter garde à respecter les conventions définies par Spring Data JPA.

# Depuis @Query

Également, lorsque la requête devient plus complexe, nous pouvons soit :

- écrire une requête JPQL avec `@Query`

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query("select u from User u where u.emailAddress = ?1")  
    User findByEmailAddress(String emailAddress);  
}
```

- écrire une requête SQL avec `@NativeQuery`

```
@NativeQuery(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1")  
User findByEmailAddress(String emailAddress);
```

# Pagination

# Pageable et Page

<https://docs.spring.io/spring-data/jpa/reference/repositories/query-methods-details.html#repositories.special-parameters>

Via l'interface `PagingAndSortingRepository` nous bénéficions directement de `Page<T>`  
`findAll(Pageable pageable)`

```
Page<User> findAll(Pageable pageable);
```

```
// Utilisation  
Pageable pageable = PageRequest.of(0, 10);  
Page<Employee> page = employeeRepository.findAll(pageable);
```

- Dans la première ligne, nous avons créé une `PageRequest` de 10 employés et demandé la première page (0). La demande de page a été transmise à `findAll` afin d'obtenir une page d'employés en réponse.
- Si nous voulons accéder à l'ensemble de pages suivant, nous pouvons augmenter le numéro de page à chaque fois.

```
PageRequest.of(1, 10);  
PageRequest.of(2, 10);  
PageRequest.of(3, 10);
```

# Specifications

# Une abstraction sur l'API Criteria

```
public interface CustomerRepository extends CrudRepository<Customer, Long>, JpaSpecificationExecutor<Customer> {  
    }  
}
```

- `JpaSpecificationExecutor` nous donne la méthode `List<T> findAll(Specification<T> spec)`

Et l'interface `Specification` est la suivante

```
public interface Specification<T> {  
    Predicate toPredicate(Root<T> root,  
                          CriteriaQuery<?> query,  
                          CriteriaBuilder builder);  
}
```

# Avec seulement JPA

```
LocalDate today = new LocalDate();

CriteriaBuilder builder = em.getCriteriaBuilder();
CriteriaQuery<Customer> query = builder.createQuery(Customer.class);
Root<Customer> root = query.from(Customer.class);

Predicate hasBirthday = builder.equal(root.get(Customer_.birthday), today);
Predicate isLongTermCustomer = builder.lessThan(root.get(Customer_.createdAt), today.minusYears(2));

query.where(builder.and(hasBirthday, isLongTermCustomer));

em.createQuery(query.select(root)).getResultList();
```

# Avec Spring Data JPA

```
public CustomerSpecifications {  
  
    public static Specification<Customer> customerHasBirthday() {  
        return new Specification<Customer> {  
            public Predicate toPredicate(Root<T> root, CriteriaQuery query, CriteriaBuilder cb) {  
                return cb.equal(root.get(Customer_.birthday), today);  
            }  
        };  
    }  
  
    public static Specification<Customer> isLongTermCustomer() {  
        return new Specification<Customer> {  
            public Predicate toPredicate(Root<T> root, CriteriaQuery query, CriteriaBuilder cb) {  
                return cb.lessThan(root.get(Customer_.createdAt), new LocalDate.minusYears(2));  
            }  
        };  
    }  
}
```

# Utilisation dans les services

```
customerRepository.findAll(CustomerSpecifications.hasBirthday());  
customerRepository.findAll(CustomerSpecifications.isLongTermCustomer());
```

Et on peut les combiner

```
customerRepository.findAll(where(customerHasBirthday()).and(isLongTermCustomer()));
```

# Chaîner les spécifications

```
public class FeedbackSpecification {  
    public static Specification<Feedback> filter(Long trainingId, Long sessionId) {  
        Specification<Feedback> specification = Specification.where(null);  
        if (trainingId != null) { specification = specification.and(belongsToTraining(trainingId)); }  
        if (sessionId != null) { specification = specification.and(belongsToSession(sessionId)); }  
        return specification;  
    }  
  
    public static Specification<Feedback> belongsToTraining(Long trainingId) {  
        return (root, query, criteriaBuilder) ->  
            criteriaBuilder.equal(root.get("sessionEnrollment").get("session").get("training").get("id"), trainingId);  
    }  
  
    public static Specification<Feedback> belongsToSession(Long sessionId) {  
        return (root, query, criteriaBuilder) ->  
            criteriaBuilder.equal(root.get("sessionEnrollment").get("session").get("id"), sessionId);  
    }  
}
```

```
public Page<Feedback> findAll(Long trainingId, Long sessionId, Pageable pageable) {  
    Specification<Feedback> specification = FeedbackSpecification.filter(trainingId, sessionId);  
    return feedbackRepository.findAll(specification, pageable);  
}
```