Hinane Aymane Rapport G5 IIR emsi centre

JpaRepository is a JPA (JAVA Persistence API ) , c’est une extension de repository. Elle contient toute les api pour les requetes CRUD basic et aussi les api pour la pagination et le trie

Create 4 packages as listed below and create some classes and interfaces inside these packages as seen in the below image

* entity
* repository
* service
* controller

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

Graphical user interface, text, application, Word, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Create a simple interface and name the interface as StudentRepository. This interface is going to extends the JpaRepository

interface JpaRepository<T,ID>

* ***T:****Domain type that repository manages (Generally the Entity/Model class name)*
* ***ID:****Type of the id of the entity that repository manages (Generally the wrapper class of your @Id that is created inside the Entity/Model class)*

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Some of the most important methods that are available inside the JpaRepository are given below

**Method 1: saveAll():**Saves all given entities.

**Syntax:**

<S extends T> List<S> saveAll(Iterable<S> entities)

**Parameters**: Entities, keeping note that they must not be null nor must it contain null.

**Return Type**: the saved entities; will never be null. The returned Iterable will have the same size as the Iterable passed as an argument.

**Exception Thrown:** It throws [IllegalArgumentException](https://www.geeksforgeeks.org/how-to-solve-illegalargumentexception-in-java/) in case the given entities or one of its entities is null.

**Method 2: getById():**Returns a reference to the entity with the given identifier. Depending on how the JPA persistence provider is implemented this is very likely to always return an instance and throw an EntityNotFoundException on first access. Some of them will reject invalid identifiers immediately.

**Syntax:**

T getById(ID id)

**Parameters**: id – must not be null.

**Return Type:** a reference to the entity with the given identifier.

**Method 3: flush():**Flushes all pending changes to the database.

**Syntax:**

void flush()

**Method 4: saveAndFlush():**Saves an entity and flushes changes instantly.

**Syntax:**

<S extends T> S saveAndFlush(S entity)

**Parameters:**The entity to be saved. Must not be null.

**Return Type**: The saved entity

**Method 5: deleteAllInBatch():**Deletes the given entities in a batch which means it will create a single query. This kind of operation leaves JPAs first-level cache and the database out of sync. Consider flushing the EntityManager before calling this method.

**Syntax:**

void deleteAllInBatch(Iterable<T> entities)

***Parameters****: The e*ntities to be deleted, must not be null.

**Implementation:**Let us consider a Spring Boot application that manages a Department entity with JpaRepository. The data is saved in the H2 database. We use a RESTful controller.