Spring Data Jpa et une première entité JPA

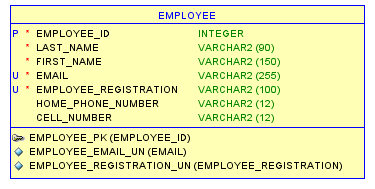
# Démonstration 1 du module 2

|  |
| --- |
| Les objectifs de cette démonstration :   * Déclaration d’une source de données * Déclaration d’une entité JPA * Création d’un Repository * Validation par les tests unitaires |

Déroulement

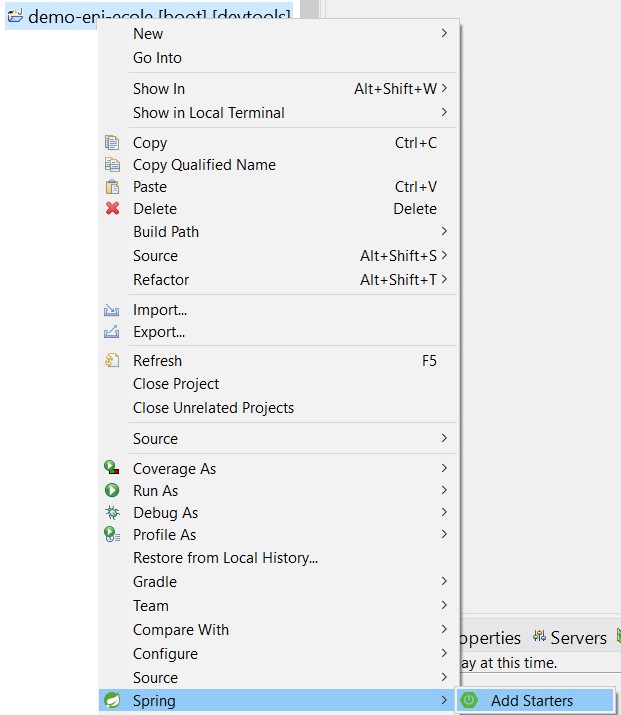
# Contexte

* Continuer l’application précédente
* Notre application va se compléter au fil des démonstrations, en représentant une partie du cœur de métier de l’ENI Ecole.
* Dans cette itération, il faut mettre en place la base de données :
  + Configuration de la source de données
  + Mise en place des annotations JPA pour que la classe Employe devienne une entité
  + Déclaration de l’interface EmployeRepository qui hérite de JpaRepository
  + Validation des comportements avec des tests unitaires
* Spring Boot utilise une base de données embarquées pour faire tourner les tests unitaires : H2
* La déclaration des classes métiers, BO, Repository et RestController est en français par défaut.
  + Pour la base de données, nous faisons le choix de mettre les informations en anglais
  + Voici le schéma de la table qui sera créée :

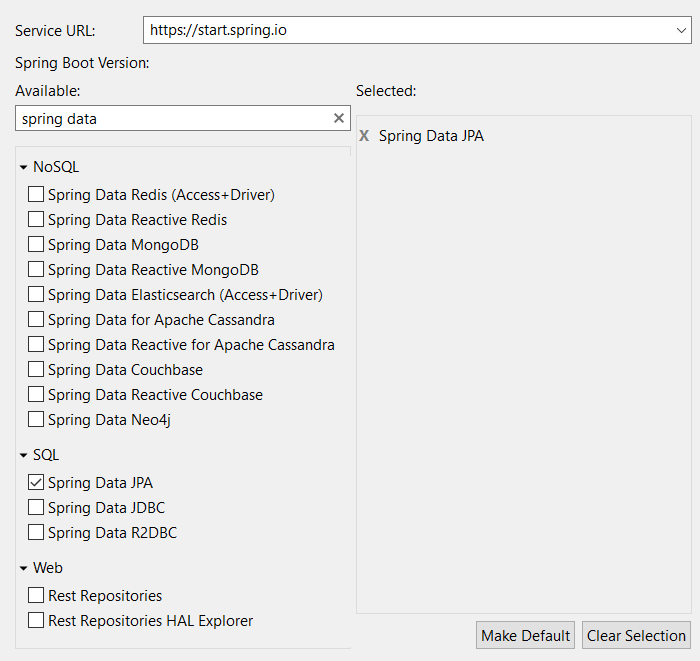


# Configuration du projet Spring Boot

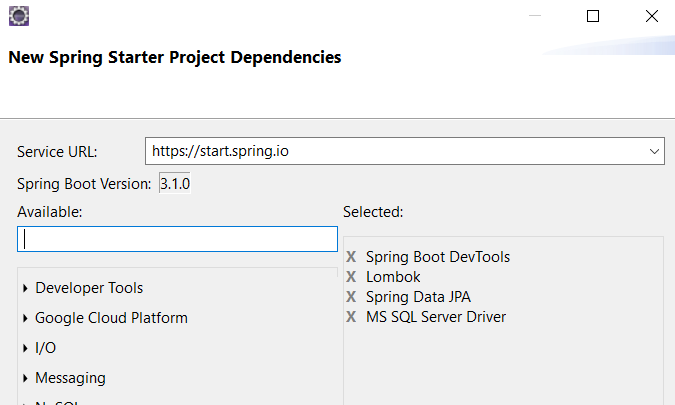
* Il faut ajouter les starters :
  + Pour Spring Data Jpa
  + Pour la base de données officielles : Microsoft SQL Server
  + Il faudra configurer aussi la base de données embarquées : H2 pour les tests unitaires.
* Dans Eclipse ; faire :
  + Clic droit sur le projet
  + Sélectionner Spring 🡪 Add Starters



* Commencer à filtrer sur Spring Data



* + Remarquer que cela expose tous les starters de Spring Data pour les sources de données : NoSQL, SQL et Web
  + Dans ce premier cas, il faut sélectionner SQL : Spring Data JPA
* Sélectionner aussi, le driver JDBC de la base de données : MS SQL Server Driver
* Recocher les starters précédents, pour qu’ils ne soient pas supprimés du fichier build.gradle
  + Spring Boot Dev Tools
  + Lombok



* Enfin, clic droit sur le projet
  + Gradle 🡪 Refresh Gradle Project



## Configuration H2 dans build.gradle

* Ajout du driver JDBC de H2
* Déclaration de cette source de données pour les tests unitaires

…

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

compileOnly 'org.projectlombok:lombok'

developmentOnly 'org.springframework.boot:spring-boot-devtools'

runtimeOnly 'com.microsoft.sqlserver:mssql-jdbc'

annotationProcessor 'org.projectlombok:lombok'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

//Ajout de la dépendance de la datasource H2 pour les tests unitaires sur JPA

testImplementation 'com.h2database:h2'

}

…

* Clic droit sur le projet
  + Gradle 🡪 Refresh Gradle Project

# Configuration de la source de données

* Pour le moment, nous allons surtout manipuler notre couche d’accès aux données au travers des tests unitaires et donc de la base de données embarquées H2
* Mais en prévision de la mise en place de la couche métier et des API, nous allons configurer dès maintenant l’utilisation de la base de données relationnelles Microsoft SQL Server
* Il est possible de la configurer la source de données via :
  + Le fichier application.properties qui est le choix par défaut
  + Ou via le fichier application.yml avec la grammaire YAML correspondante
  + Pour cette démonstration, nous vous fournissons les 2 choix

## Déclaration dans application.properties

#Connexion à la base de données

spring.datasource.url=jdbc:sqlserver://localhost;databasename=DEMO\_ENI\_ECOLE;integratedSecurity=false;encrypt=false;trustServerCertificate=false

spring.datasource.username=sa

spring.datasource.password=Pa$$w0rd

#Détail des requêtes pour le développement

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

## Déclaration dans application.yml

#Connection to DB

spring:

datasource:

url: jdbc:sqlserver://localhost;databasename=DEMO\_ENI\_ECOLE;integratedSecurity=false;encrypt=false;trustServerCertificate=false

username: sa

password: Pa$$w0rd

#Options to DB

jpa:

show-sql: true

hibernate:

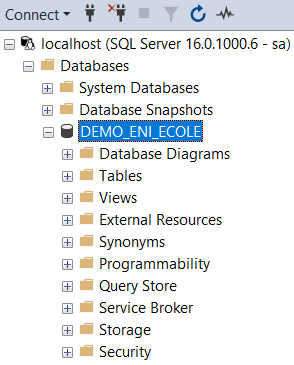
ddl-auto: create

Pour Spring Boot; il n’y a pas de différence entre ces 2 configurations.

Il s’agit plus d’un choix de développeur entre la grammaire des fichiers de properties et de YAML

Par défaut, si vous définissez les 2 fichiers, il utilisera application.properties

* Lancer votre Microsoft SQL Server Management Studio
  + Créer une nouvelle base de données : DEMO\_ENI\_ECOLE



# Déclaration d’une entité JPA : Employe

* Ajouter les annotations de JPA pour que la classe métier Employe devienne une entité
  + Il faut qu’elle corresponde à la table :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* + Les annotations nécessaires sont :
    - @ [jakarta](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta).[persistence](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta.persistence).Entity
    - @[jakarta](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta).[persistence](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta.persistence).Table 🡪 pour préciser le nom de la table en base de données
    - @[jakarta](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta).[persistence](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta.persistence).Id 🡪 pour préciser la clef primaire
    - @[jakarta](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta).[persistence](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta.persistence).GeneratedValue 🡪 pour préciser la stratégie de génération. IDENTITY pour notre cas
    - @[jakarta](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta).[persistence](eclipse-javadoc:%E2%98%82=demo-spring-jpa/C:%5C/Users%5C/abaille%5C/.gradle%5C/caches%5C/modules-2%5C/files-2.1%5C/jakarta.persistence%5C/jakarta.persistence-api%5C/3.1.0%5C/66901fa1c373c6aff65c13791cc11da72060a8d6%5C/jakarta.persistence-api-3.1.0.jar=/gradle_used_by_scope=/main,test=/%3Cjakarta.persistence).Column 🡪 pour préciser le nom des colonnes, l’unicité ou non, la nullité ou non et la taille

**package** fr.eni.demo.bo;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@EqualsAndHashCode(of= {"immatriculation"})

@ToString

@Builder

/\*\*

\*

\* **@author** Eni Ecole

\* Voici une classe BO (Business Object)

\* Respectant le design pattern POJO (Plained Old Java Object)

\* Définie avec Lombok

\* Elle est maintenant déclarée comme une entité JPA

\*/

@Entity

@Table(name="EMPLOYEE")

**public** **class** Employe {

//Attributs

//Clef primaire -- générée par la base avec IDENTITY

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "EMPLOYEE\_ID")

**private** Integer id;

//Précision sur le nom de la colonne -> name=

//Cette colonne est non nulle -> nullable = false

//La taille de la colonne en base est de 90 -> length = 90

@Column(name = "LAST\_NAME",nullable = **false**, length = 90)

**private** String nom;

@Column(name = "FIRST\_NAME",nullable = **false**, length = 150)

**private** String prenom;

//La valeur de l’email est unique -> unique = true

@Column(nullable = **false**, unique = **true**, length = 255)

**private** String email;

@Column(name="EMPLOYEE\_REGISTRATION", nullable = **false**, unique = **true**, length = 100)

**private** String immatriculation;

@Column(name="HOME\_PHONE\_NUMBER", length = 12)

**private** String numDom;

@Column(name="CELL\_NUMBER", length = 12)

**private** String numPortable;

}

# Création de EmployeRepository

* A présent, dans la couche DAL, nous n’avons plus besoin de déclarer des interfaces DAO et leurs implémentations
* Il nous suffit de déclarer des interfaces héritant de JpaRepository
  + Et préciser l’entité JPA associée et son type de clef primaire
* Supprimer l’interface et la classe d’implémentation du package DAL
* Créer l’interface EmployeRepository qui hérite de JpaRepository<Employe,Integer>

Il n’est pas nécessaire de poser d’annotation sur ces interfaces

Spring Boot détermine qu’il faut créer l’implémentation et injecter une instance dans le contexte de Spring

# Création de tests unitaires pour valider le Repository

Spring Boot nous fournit un contexte de tests unitaires pour les JpaRepository.

* Pour cela, il faut utiliser l’annotation :

@org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest

* Et utiliser un « mock » de l’EntityManager de JPA qui est :

org.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManager

Dans les tests unitaires, nous allons vérifier les comportements de base :

* Création d’un employé
* Trouver un employé par son identifiant
* Mise à jour d’un employé
* Suppression de l’employé
* Récupération de tous les employés

**package** fr.eni.demo.dal;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** **static** org.junit.jupiter.api.Assertions.*assertNull*;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManager;

**import** fr.eni.demo.bo.Employe;

**import** lombok.extern.slf4j.Slf4j;

@Slf4j

@DataJpaTest

**public** **class** TestEmployeRepository {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

EmployeRepository repository;

@Test

**public** **void** test\_save() {

**final** Employe entiteASauver = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Appel du comportement

**final** Employe entite = repository.save(entiteASauver);

***log***.info(entite.toString());

*assertThat*(entite.getId()).isGreaterThan(0);

}

@Test

**public** **void** test\_findById() {

// Contexte de la DB

**final** Employe entite = Employe

.*builder*()

.nom("DAUTAIS")

.prenom("Servane")

.email("sdautais@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_09397")

.numPortable("06XXXXXXXX")

.build();

// Contexte de la DB

entityManager.persist(entite);

entityManager.flush();

***log***.info(entite.toString());

*assertThat*(entite.getId()).isGreaterThan(0);

// Appel du traitement

**int** id = entite.getId();

**final** Optional<Employe> op = repository.findById(id);

//Vérification que l'Optional contienne une entité

*assertThat*(op.isPresent()).isTrue();

//Récupération de l'entité

**final** Employe entiteDB = op.get();

***log***.info(entiteDB.toString());

//Validation de l'entité

*assertThat*(entiteDB.getId()).isEqualTo(id);

*assertThat*(entiteDB).isEqualTo(entite);

}

@Test

**public** **void** test\_update() {

// Contexte de la DB

**final** Employe entite = Employe

.*builder*()

.nom("LACHESNAIS")

.prenom("Frédéric")

.email("fdelachesnais@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_15009")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build();

// Contexte de la DB

entityManager.persist(entite);

entityManager.flush();

*assertThat*(entite.getId()).isGreaterThan(0);

***log***.info("ORIGINE : " + entite.toString());

entite.setNom("DELACHESNAIS");

// Appel du traitement

**final** Employe entiteDB = repository.save(entite);

*assertThat*(entiteDB.getId()).isEqualTo(entite.getId());

*assertThat*(entiteDB.getNom()).isEqualTo(entite.getNom());

***log***.info("MISE A JOUR : " + entiteDB.toString());

}

@Test

**public** **void** test\_delete() {

**final** Employe entite = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Contexte de la DB

entityManager.persist(entite);

entityManager.flush();

***log***.info(entite.toString());

*assertThat*(entite.getId()).isGreaterThan(0);

// Appel du comportement

repository.delete(entite);

// Vérification que l'entité a été supprimée

Employe entityDB = entityManager.getEntityManager().find(Employe.**class**, entite.getId());

*assertNull*(entityDB);

}

// Vérifier que s’il n’y a pas d’enregistrement en base, nous remontons vide

@Test

**public** **void** test\_find\_all\_is\_empty() {

**final** List<Employe> lstEntites = repository.findAll();

*assertThat*(lstEntites).isEmpty();

}

// Retrouver l’ensemble du jeu de données

@Test

**public** **void** test\_findAll() {

// Contexte de la DB

jeuDeDonnees();

**final** List<Employe> lstEntites = repository.findAll();

*assertThat*(lstEntites.size()).isGreaterThan(0);

***log***.info(lstEntites.toString());

}

// Création d’un jeu de données pour simuler la base

**private** **void** jeuDeDonnees() {

**final** List<Employe> lstEntites = **new** ArrayList<>();

lstEntites.add(Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build());

lstEntites.add(Employe

.*builder*()

.nom("DAUTAIS")

.prenom("Servane")

.email("sdautais@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_09397")

.numPortable("06XXXXXXXX")

.build());

lstEntites.add(Employe

.*builder*()

.nom("DELACHESNAIS")

.prenom("Frédéric")

.email("fdelachesnais@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_15009")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build());

// Contexte de la DB

lstEntites.forEach(e -> {

entityManager.persist(e);

});

entityManager.flush();

}

}

Les tests fonctionnent indépendamment les uns des autres.

* Si vous les exécuter, ils seront tous verts.

# Modification de la couche métier (BLL)

En supprimant l’interface et la classe DAO de la couche DAL. Nous avons mis en erreur la couche métier.

* Les services de la couche métier doivent utiliser les repository de la couche DAL
* Dans EmployeServiceImpl injecter EmployeRepository au lieu du EmployeDAO

Voici les correspondances entre les méthodes du DAO (CRUD) et celles du Repository de Spring Data JPA :

|  |  |
| --- | --- |
| DAO (CRUD) | Repository (Spring Data JPA) |
| create | save |
| read | findById(Entite.class, ID) : Optional<Entite.class> |
| findAll | findAll |
| update | save |
| delete | delete |

A cette étape nous ne savons pas encore comment gérer la récupération d’un employé par son immatriculation

L’immatriculation est une spécificité du métier. Il nous faudra voir comment ajouter ce type de requête particulière avec Spring Data JPA

Nous le verrons dans la suite du cours. Pour que les tests unitaires nous servent de vérification, nous allons forcer celui-ci à tomber en erreur en retournant l’employé en paramètre par défaut. Nous pourrons ainsi vérifier son bon fonctionnement, lorsque nous aurons la solution. Nous indiquons aussi un TODO pour nous rappeler ce qu’il reste à faire

**package** fr.eni.demo.bll;

**import** java.util.List;

**import** org.springframework.stereotype.Service;

**import** fr.eni.demo.bo.Employe;

**import** fr.eni.demo.dal.EmployeRepository;

**import** lombok.AllArgsConstructor;

//Permet de faire injecter la couche DAL associée

@AllArgsConstructor

@Service

**public** **class** EmployeServiceImpl **implements** EmployeService {

**private** EmployeRepository employeRepository;

@Override

**public** **void** ajouter(Employe employe) {

// Validation des données de l'employé avant sauvegarde

**if** (employe == **null**) {

**throw** **new** RuntimeException("L'employé n'est pas renseigné");

}

validerImmatriculation(employe);

validerChaineNonNulle(employe.getNom(), "Vous devez renseigner le nom");

validerChaineNonNulle(employe.getPrenom(), "Vous devez renseigner le prénom");

validerChaineNonNulle(employe.getEmail(), "Vous devez renseigner un email");

employeRepository.save(employe);

}

@Override

**public** List<Employe> chargerTousEmployes() {

**return** employeRepository.findAll();

}

**private** **void** validerChaineNonNulle(String chaine, String msgErreur) {

**if** (chaine == **null** || chaine.isBlank())

**throw** **new** RuntimeException(msgErreur);

}

**private** **void** validerImmatriculation(Employe employe) {

// Valider que l'immatriculation n'est pas nule ou vide

validerChaineNonNulle(employe.getImmatriculation(), "L'immatriculation n'a pas été renseignée");

// Immatriculation doit être unique

//**TODO** A compléter quand nous serons créés une requête spécifique pour Spring Data Jpa

Employe employeDB = **null**;

**if** (employeDB != **null**) {

**throw** **new** RuntimeException("L'immatriculation doit être unique");

}

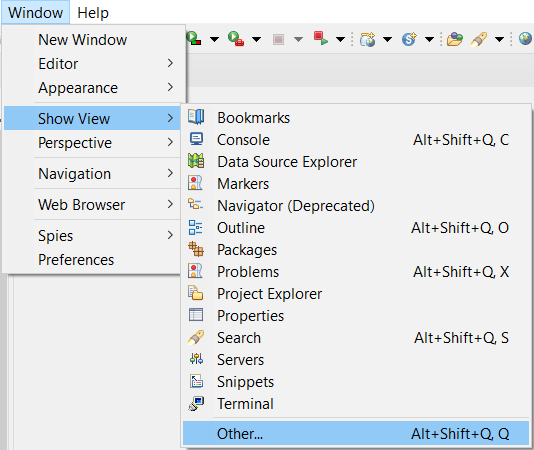
}

}

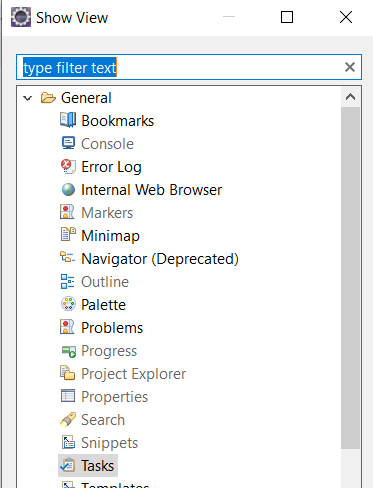
Eclipse vous indique que le bloc if devient inutile. Nous le laissons comme ça pour le moment.

Avantage du TODO

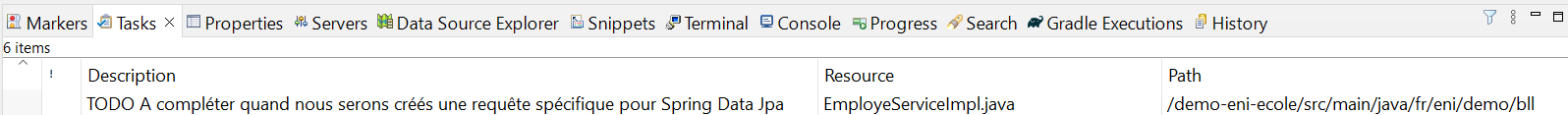
* Nous pouvons les faire afficher en sélectionnant l’onglet : Tasks
* Window 🡪 Show View 🡪 Other …



* General 🡪 Tasks



* Vous avez un nouvel onglet, qui permet de voir et d’accéder aux TODOs



Lors d’un livrable au client, il ne doit plus avoir de TODO

# Modification des tests unitaires du service

* Dès qu’une mise à jour d’une méthode est faite ; il faut mettre à jour le test unitaire associé
  + Il faut aussi par défaut les faire retourner pour valider les non-régression
* Les tests doivent refléter les mises à jour et continuer à prouver le fonctionnement
* Dans l’itération précédente, la couche d’accès aux données était un bouchon
  + Nous devons donc reprendre les tests unitaires et utiliser Mockito pour gérer le bouchon du EmployeRepository
* Nous aurons pour le moment un test qui sera en échec celui de la recherche d’un employé par son immatriculation
  + Il permettra de confirmer qu’il faut mettre à jour ce code
* Il faut injecter EmployerRepository à la place d’EmployeDAO
* Et changer les méthodes du CRUD d’origine par les méthodes de CrudRepository et JpaRepository

**package** fr.eni.demo.bll;

**…**

**import** java.util.Optional;

…

**import** fr.eni.demo.bo.Employe;

**import** fr.eni.demo.dal.EmployeRepository;

**class** TestEmployeService {

**private** EmployeService employeService;

@Mock // Injection d'un Mock du EmployeRepository

**private** EmployeRepository employeRepository;

@BeforeEach

**void** init() {

MockitoAnnotations.*openMocks*(**this**);

employeService = **new** EmployeServiceImpl(employeRepository);

}

@Test

**void** test01\_ajouter\_tousParametresValides() {

**int** id = 1;

Employe employe = Employe

.*builder*()

.id(id)

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_Ecole\_012892")

.numDom("0299XXXXXX")

.numPortable("0699XXXXXX")

.build();

// Définir le comportement du Repository mocké

*when*(employeRepository.findById(id)).thenReturn(Optional.*of*(employe));

// Comportemnet à valider

employeService.ajouter(employe);

// Vérification de l'ajout dans la liste des employés

Optional<Employe> op = employeRepository.findById(id);

*assertNotNull*(op);

*assertThat*(op.isPresent()).isTrue();

**final** Employe employeDB = op.get();

*assertNotNull*(employeDB);

*assertThat*(employe.getImmatriculation()).isEqualTo(employeDB.getImmatriculation());

*assertThat*(employe.getEmail()).isEqualTo(employeDB.getEmail());

*assertThat*(employe.getNom()).isEqualTo(employeDB.getNom());

*assertThat*(employe.getPrenom()).isEqualTo(employeDB.getPrenom());

*assertThat*(employe.getNumDom()).isEqualTo(employeDB.getNumDom());

*assertThat*(employe.getNumPortable()).isEqualTo(employeDB.getNumPortable());

}

…

// Immatriculation Unique - la validation métier l'interdit

@Test

**void** test\_ajouter\_immatriculationUnique() {

Employe employe1 = Employe

.*builder*()

.id(1)

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_Ecole\_012892")

.numDom("0299XXXXXX")

.numPortable("0699XXXXXX")

.build();

employeService.ajouter(employe1);

// Définir le comportement du Repository avec findByImmatriculation

//**TODO** A compléter quand nous serons créés une requête spécifique pour Spring Data Jpa

//when(employeRepository.findByImmatriculation("ENI\_Ecole\_012892")).thenReturn(employe1);

Employe employe2 = Employe

.*builder*()

.id(2)

.nom("GOBIN")

.prenom("Stephane")

.email("sgobin@campus-eni.fr")

.immatriculation("ENI\_Ecole\_012892")

.numDom("0288XXXXXX")

.build();

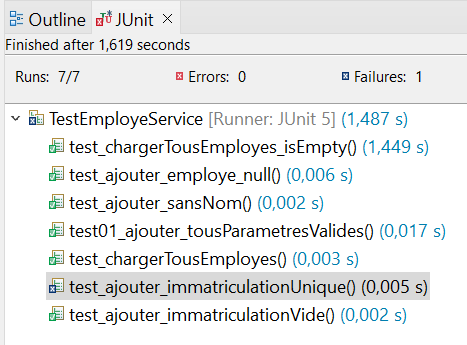
*assertThrows*(RuntimeException.**class**, () -> employeService.ajouter(employe2));

}

…

}

* Lors de l’exécution, les tests passent sauf celui de l’unicité de l’immatriculation



* Dans Tasks nous voyons nos 2 TODOs

