
# Documentation Technique : Service Desk

Victor Marfil
Lilian Miesiac
Quentin Monnet
Luca Grunenwald

15.01.2020

# TABLE DE MATIERE

# Description des types de classes

## 1. Classes « entité » (edu.uha.miage.core.entity)

Ces classes représentent les tables de notre base de données. Nous avons donc « recopié » le MCD dans ces classes. Lors de la première initialisation de l'application, les tables correspondantes seront créées.

## 2. Classes « controller » (edu.uha.miage.web.controller)

Ces classes permettent de gérer les différentes requêtes qui sont faites sur l'application. On y retrouve tous les GET et POST possibles de l'application.

## 3. Classes « demo » (edu.uha.miage):

Ces classes ont pour but de peupler la base donnée avec des valeurs initiales. On y retrouve par exemple la création du compte administrateur.

## 4. Classes « repository » et « service / serviceImpl » (edu.uha.miage.core.repository & edu.uha.miage.core.service)

Ces classes ont pour but de permettre de faire des requêtes à la base de données sans utiliser de requête SQL brute nous-même mais en utilisant plutôt des noms de fonctions faisant office de requête SQL.

## 5. Classes « configuration » (edu.uha.miage.config)

Ces classes permettent de configurer le fonctionnement de l'application. On y retrouve notamment les règles de sécurité (WebSecurityConfigProd.java) et la gestion des droits utilisateurs (SecurityUser.java)

# Classes « entité »

## 1. Fonctionnement

Le fonctionnement des classes entités est très simple. Il suffit d'ajouter des variables de classes pour que directement le champ correspondant à cette variable soit créé dans la base de données lors du lancement de l'application.

Il est possible d'ajouter des annotations au-dessus des variables pour leur donner des contraintes (de taille par exemple pour une String avec Size).

Ce sont ces classes qui seront manipulées dans les formulaires de l'application.

## 2. Méthodes

Les méthodes de ces classes sont uniquement des getters et des setters pour les différentes variables de classes.

Les variables de classes représentent les champs de notre base de données.

Ces getters et setters seront notamment utilisés plus loins lors de l'utilisation des classes entités dans les templates thymeleaf.

# Classes « controllers »

## 1. Fonctionnement

Les classes « controllers » ont pour but de créer le cheminement logique de l'application web. En ajoutant un « RequestMapping » nous pouvons définir un chemin d'accès aux différentes méthodes de la classe.

## 2. Méthodes

Les méthodes des classes « controllers » gèrent la logique et le bon déroulement de l'application. Par exemple, dans « DemandeController » on retrouve une fonction findAll(Model model) ayant pour URL « /demandes ».

Lorsque l'utilisateur cliquera sur un lien redirigeant vers « /demandes », alors le code contenu dans la fonction « findAll(Model model) » de « DemandeController » sera appelé.

En retour de méthode se trouve la page HTML qui va être rendu (dans le cas exemple « demande/viewDemandes.html » situé dans le dossier « template ».

Le paramètre de type Model dans les différentes fonctions permet notamment d'ajouter des variables pour les utilisés dans le template Thymeleaf.

Il existe aussi d'autres paramètres pour les méthodes. Dans ces paramètres ont retrouve notamment l'annotation "@Valid" et une variable de type BindingResult (et la variable de type Model en plus).

Ces deux premiers paramètres ont pour but de valider l'entrée utilisateur. Spring va automatiquement vérifier que les contraintes mises sur les variables de la classe entité soit respectées. Il est possible de vérifier s'il y a des erreurs avec la variable de type BindingResult et sa méthode "hasErrors()". Un exemple serait dans FonctionController.java avec sa méthode "edited".

# Classes « demo »

## 1. Fonctionnement

Les classes demo permettent de peupler la base de données pour avoir un jeu de données initial lors du lancement de l'application. Elles sont notamment utilisées pour créer le compte et la personne administrateur.

## 2. Méthodes

Les classes « demo » héritent toutes de CommandLineRunner. Elles ont donc par défaut une méthode nommée « run ». Cette méthode sera appelée automatiquement lors du lancement de l'application.

En général, on y retrouve aussi une autre méthode permettant d'ajouter dans la base de donnée les enregistrements en vérifiant leurs non présences au préalable.

Pour choisir dans quel ordre exécuter ces « demo », il faut ajouter une annotation « Order » ainsi que le nombre représentant l'ordre dans lequel il va être exécuté. Par exemple, dans notre projet le classe « DemoRole » sera exécuté en premier, suivi « DemoDepartement », etc.

Il est important de définir un bon ordre car certaines entités doivent au préalable avoir des informations concernant d'autres entités.

Par exemple, pour pouvoir créer un compte il faut au préalable avoir la personne qui va être le titulaire du compte.

# Classes « repository » et « service / serviceImpl »

## 1. Fonctionnement

Les classes « repository » sont les classes qui vont réellement exécuter des requêtes à la base de données (SELECT, UPDATE, etc.) bien qu'elles soient cachées par des noms de fonctions.

Les interfaces « service » permettent l'utilisation de ces repository mais ne représentent qu'une classe abstraite ne proposant pas d'implémentation.

Les classes « serviceImpl » permettent d'implémenter les interfaces « service » et utilisent les classes « repository » pour accéder à la base de données.

Dans les classes « controllers », nous utilisons en général des « Autowired » avec des variables de classes de type « service ». Spring ira chercher une implémentation (classes « serviceImpl ») de ces interfaces automatiquement.

On peut ensuite simplement les utiliser pour récupérer / insérer des informations dans la base de données.

## 2. Méthodes

En général, nous retrouvons les méthodes « findAll() » qui permettent de récupérer la liste de tous les enregistrements pour un type d'entité donnée.

Il arrive parfois que nous ayons des fonctions plus complexe tel que « findByNomAndPrenomAndEmailAndAdresse » dans « PersonneServiceImpl.java ».

Cette fonction, ira chercher les enregistrements de « Personne » ayant pour nom, prénom, email et adresse les valeurs données à la fonction.

Ces fonctions ont un nommage très précis décris par la documentation suivante : https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation

# Classes « configuration »

## 1. Fonctionnement

Ces classes héritent pour la plupart de classes pré établi par Spring.

La classe « SecurityUser » par exemple dérive de UserDetails (donnée par Spring) et permet de donner des droits aux utilisateurs.

La classe « SpringWebConfig » dérive de WebMvcConfigurer et permet de choisir dans quel répertoire se trouvent les différentes ressources tel que les fichiers CSS et JS.

La classe « StorageProperties » n'héritent d'aucune classe et ne sert que de configuration pour les classes nécessitant de l'upload de fichier (par exemple ServiceController.java où est fait l'upload de fichier pour donner une image à notre service).

Les classes « WebSecurityConfigProd » et « WebSecurityConfigDev » dérivent de la classe proposé par Spring « WebSecurityConfigurerAdapter » et permettent de définir les règles d'accès au différentes ressources du site selon les rôles des utilisateurs.

## 2. Méthodes

Les méthodes sont pour toutes (sauf StorageProperties) des méthodes héritées de leur classe parent.

Les méthodes de StorageProperties permettent de récupérer et définir le chemin d'accès du fichier où seront mis les fichiers télécharger.