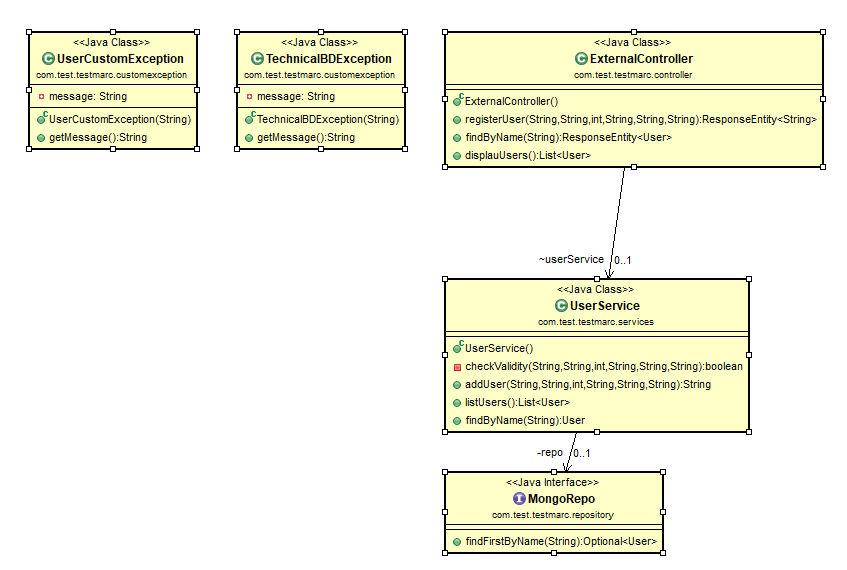
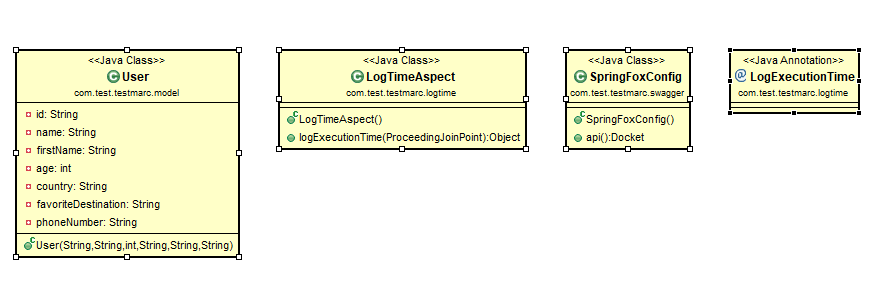
# Documentation du projet de test pour Air France

Marc Rubin le 30/05/2021

# Diagramme UML de l’application :



# Choix d’architecture :

Pour ce projet, j’ai choisi d’utiliser le Spring boot qui permet d’intégrer de manière simple les principaux composants du framework Spring qui répondent aux attentes de ce projet : Spring web, Spring Data, Spring AOP. Il me permet également de lancer l’application dans un serveur Tomcat embarqué pour la facilité de développement.

Concernant la partie des tests, je vais utiliser Junit et Mockito qui sont des standards du test unitaire.

Finalement pour la base de données, j’ai choisi une base NoSQL car il n’y a pas de données relationnelles et en particulier mongoDB pour son stockage des données en document qui permet une grande flexibilité en cas d’ajout de champs utilisateurs. Il est facilement intégrable avec spring data dans le cadre de ce projet.

# Details de conception :

Cette application possède une architecture en trois couches :

La couche d’exposition des services rest utilisant Spring Web et exposant deux endpoint dans la classe ExternalController :

/registerUser pour l’enregistrement d’un nouvel utilisateur et

/displayUser/{name} pour afficher les détails d’un utilisateur suite à une recherche via son nom.

Le service d’enregistrement renvoi un code http 200 en cas de succès avec une phrase indiquant le nom de l’utilisateur, un code 403 en cas d’entrées non valide ou un code 500 si la base n’a pas pu être jointe.

Le service de recherche et d’affichage des détails renverra un code 200 et la représentation JSON de la classe utilisatuer correspondante ou un code 404 si l’utilisateur n’a pas été trouvé.

Ce contrôleur fait appel à la couche service via la classe UserService à fin de manipuler des données utilisateur. Lors de l’enregistrement on utilisera la méthode addUser qui va dans un premier temps vérifier l’intégrité des données utilisateur fournies avec sa méthode privée checkValidity ^puis faire appel à la couche Data pour enregistrer l’utilisateur. Lors de l’affichage des détails, c’est la méthode findByName qui sera appelée.

Les méthodes de la classe UserService sont annotées avec @LogExecutionTime ce qui permet d’inscrire dans les logs le démarrage, la fin et le temps d’exécution de cette méthode.

Enfin la couche data est composée d’un repository MongoRepo qui permet de faire le lien avec la base de donnée embarquée MongoDB elle exécute les enregistrements et recherche de la classe User en base.

# Tests Unitaires:

Des tests unitaires sur les trois couches de conception vues plus haut peuvent être mis en place.

Pour la couche d’exposition des services Rest, j’ai remplacé les tests unitaires par l’ajout de swagger-ui, une interface utilisateur qui permet de bien décrire les services exposés et de les tester en direct via l’interface, j’aurai également pu ajouter des tests unitaires sur cette couche en réalisant des appels Rest à l’aide de Junit.

Pour la couche service, j’ai réalisé quelques cas de tests qui peuvent être étendus mais comprennent les principales attentes de validation des entrées utilisateur lors de l’enregistrement ainsi que les cas de recherche d’utilisateur existant ou non, pour effectuer ces tests, j’ai mocké la base MongoDB embarquée pour ne tester que les services. Il existe une autre méthode de test consistant à utiliser la base embarqué à la place d’un mock mais dans ce cas d’intégration de la base, on réalise un test qui s’apparente plutôt à du test EndToEnd au lieu d’unitaire.

Pour la couche Data, je n’ai pas effectué de test en considérant que la base embarqué ne sera pas utilisée dans un contexte de recette ou production, ces tests ne me paraissent pas pertinents (la base embaquée n’étant pas produit officiel MongoDB, il peut y avoir quelques différences de comportement).

# Quelque unes des améliorations possibles:

Optimisation AOP : dans ce contexte très précis ou on utilise peu l’AOP avec des Joinpoint qui correspondent uniquement à des méthodes et dans un contexte Spring, l’utilisation de Spring AOP me parait adapté avec l’aide d’annotations AspectJ, mais dans un application plus importante avec beaucoup d’appels, il sera plus préférable de passer entièrement avec une solution AspectJ comportant une compilation du code AOP qui sera 8 à 35 fois plus rapide en terme de vitesse d’exécution que la solution actuelle s’exécutant au runtime.  
Dans ce projet, j’ai uniquement appliqué les logs sur les méthodes de la couche service, mais on peux annoter toutes les méthodes de l’application de la même manière (dans les couches controller rest et data).

Définition du user : l’utilisateur peux être étendu avec beaucoup plus de champs, email, adresse, date de naissance, je suis resté sur une modélisation simpliste qui correspond aux attentes du projet (chacun des champs nécessitant une vérification personnalisée).

Vérification des champs utilisateur : Lors de la vérification des entrées utilisateur, une solution efficace que je voulais mettre en oeuvre sur le numéro de téléphone est la vérification des champs par des Regex avec les classes Pattern et Matcher, je suis revenu sur ma décision car je ne voulais pas bloquer l’enregistrement de l’utilisateur pour le champ téléphone facultatif, cela relève plus des contrôles fait dans la partie UI, je pourrais passer le numéro à null et réaliser l’enregistrement mais ce type de décision fonctionnel est généralement décrit dans un document de spécification.  
J’ai laissé cette partie de côté pour une économie de temps.

Optimisation du fonctionnement de la recherche utilisateur : la recherche actuelle se base sur le nom de l’utilisateur, si deux utilisateurs on le même nom, elle ne renverra jamais le deuxième, j’ai pris cette décision dans le contexte du projet car je voulais une utilisation simple du projet permettant l’enregistrement d’un utilisateur et sa recherche par son nom, dans un cas réel d’utilisation, soit les critère de recherche seront plus poussés (nom + prénom + date de naissance…) et permettant d’arriver à un utilisateur unique, soit on utilisera l’id de l’utilisateur pour la recherche, n’ayant pas installé Compass sur l’ordinateur de développement de ce projet, je n’avais pas d’accès à la base Mongo embarquée et aux Id des utilisateurs (d’où la création du 3ème endpoint affichant le contenu de la base). Je suis donc resté sur une implémentation simpliste de la recherche en considérant que 2 utilisateurs n’auront pas le même nom.