Équida

Doc Technique

MARTIN Justine
BOTTON Léa

Table des matières

1 Contexte et présentation

A Présentation du contexte

B Choix techno

i MySql

ii Spring Boot

iii Ionic

iv Gradle

Gradle est un "build automation system" (moteur de production). Il est un équivalent plus récent et complet à Maven. Il possède de meilleure performances, possède un bon support dans de nombreux IDE et permet d'utiliser de nombreux dépots, dont ceux de Maven.

C Organisation du projet

i Git et branches

Branches

Nommenclature

ii Les différents dossiers

Doc

SQL

Sources

iii Trello

D Interactions entre les différentes parties du projet

i Les différentes parties

Le projet Équida est composé de 2 applications. Une l'application web, qui est également l'application principale, une application mobile qui est à usage principal des utilisateurs. Les 2 applications s'appuient sur la même base de données. L'application web y est directement connectée. L'application mobile, elle, passe par une API. En effet, si celle ci se connecterait directement à la base de données comme c'est le cas pour l'application web, une personne mal intentionnée serait en mesure de décompiler l'application mobile afin d'obtenir les identifiants de la base de données. L'utilisation de cette API empêche donc notamment ce problème de sécurité.

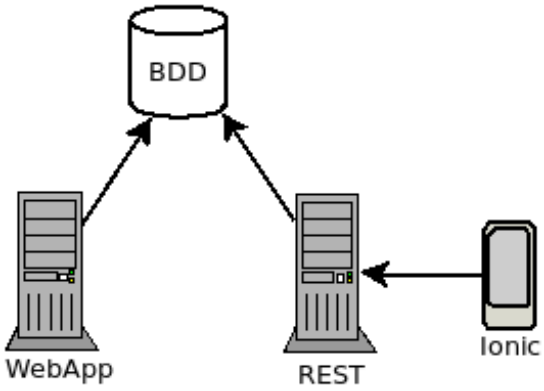

FIGURE 1.1 – La connexion à la BDD selon le projet

L'API ainsi que l'application web utilisent sur le Framework Spring Boot. Ces 2 applications font donc parti de 2 projets différents, "webapp" pour la partie web et "rest" pour l'api. Celles si demandant un code identique pour les Services, les Entités ainsi que les Repository, le choix a donc été fait de faire un projet commun dénomé "core" dans lequel on peut retrouver tout le code qui sera commun aux 2 autres parties, non seulement concernant les éléments cités plus haut mais également concernant les exceptions ou certains outils.

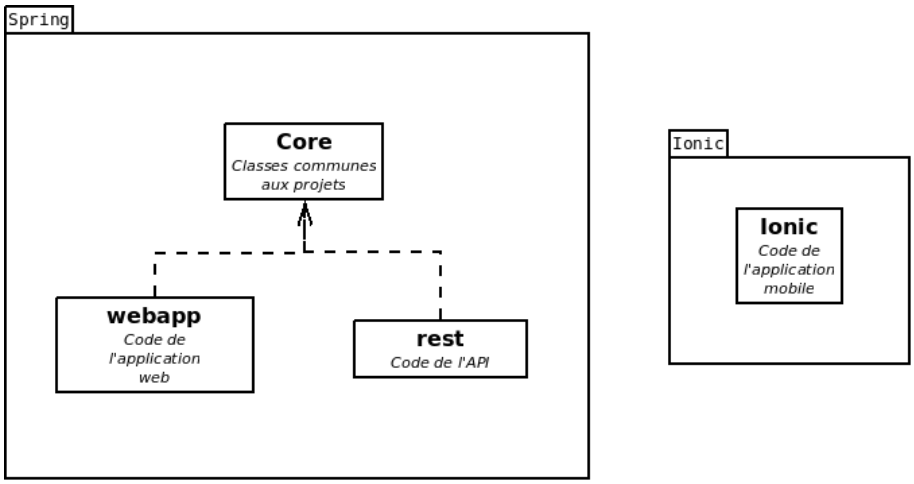

FIGURE 1.2 – Les dépendances entre les projets

ii Configuration Gradle

Pour gérer correctement les différents projets basés sur Spring, leur dépendances ainsi que leur configuration nous avons donc utilisé Gradle comme mentionné plus haut. Dans le dossier "src/Spring" on retrouve le "build.gradle" qui se charge de configurer tout le projet. On peut observer la configuration suivante pour tout les projets.

```
allprojects {
    apply plugin : 'java'
    apply plugin : 'io.spring.dependency-management'
    apply plugin : 'org.springframework.boot'

    ext {
        springBootVersion = '2.1.3.RELEASE'
    }

    repositories {
        mavenCentral()
        jcenter()
        maven {
            url 'https://plugins.gradle.org/m2/'
        }
    }

    dependencies {
        implementation 'org.springframework.boot:spring-boot-starter'
        implementation 'org.springframework.boot:spring-boot-starter-web'
        implementation 'org.springframework.boot:spring-boot-starter-actuator'
        implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
        implementation 'org.springframework.boot:spring-boot-starter-security'

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

FIGURE 1.3 – Configuration Gradle de tous les projets

On définit donc la version de Spring à utiliser, en plus des dépendences commune à chaque projet (spring-boot-starter-web, spring-boot-starter-data-jpa, ...). On va par la suite définir les dépendances uniques à chaque projet.

```
project(':core') {
    jar {
        enabled = true
    }

    dependencies {
        implementation 'com.h2database:h2'
        implementation 'mysql:mysql-connector-java'
    }
}

project(':rest') {
    dependencies {
        implementation project(':core')
    }
}

project(':webApp') {
    dependencies {
        implementation project(':core')

        implementation 'org.springframework.boot:spring-boot-starter-freemarker'
    }
}
```

FIGURE 1.4 – Configuration Gradle individuelle des projets

De même, concernant le projet core, on active uniquement la compilation en jar (comme une lib) et non pas en jar bootable (comme c'est le cas lorsque l'on utilise Spring Boot).

D'autre scripts "build.gradle" se trouvent dans chaque dossiers du projet, cependant, ceux ci ne configurent que le nom du projet à l'issue du build, la version du JDK utilisée ainsi que le package de base du projet.

2 Core

A Organisation des packages

L'organisation des packages se déroule comme suit :

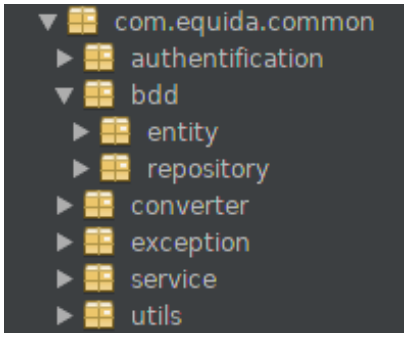

FIGURE 2.1 – Packages de Core

authentification Contient toutes le classes relatives à l'authentification

bdd Contient toutes les classes relative à la base de données

- **entity** Contient toutes les classes Métiers
- **repository** Contient toutes les classes qui héritent de CrudRepository

converter Contient toutes les classes qui héritent de AttributeConverter

exception Contient toutes les Exceptions

service Contient toutes les classes Services

utils Contient quelques classes utiles (Sha256PasswordEncoder, DateUtils, ...)

B Exemple d'Entity

C Exemple de Repository

D Exemple de Service

E Exemple d'exception (NotFoundException)

Le module Core permet de fournir certaines exceptions. Actuellement elles sont au nombre de 4.

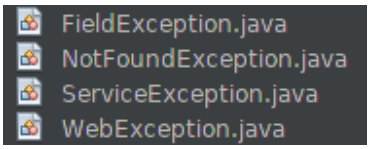

FIGURE 2.2 – Les différentes exceptions

FieldException Permet de signaler une erreur sur un champs d'un formulaire alors que l'information saisie est valide d'un point de vue HTML. On peut par exemple citer un sire qui n'existe pas dans la base de données, un login qui existe déjà, ...

NotFoundException Permet, notamment, de signaler que l'enregistrement demandé n'existe pas dans la base de données ou bien que l'utilisateur n'est pas autorisé à le voir, comme c'est le cas avec un cheval qui appartient à un autre client par exemple.

ServiceException Permet de signaler une erreur dans le comportement du Service. Elles sont surtout utilisés pour signaler qu'un champs requis vaut null.

WebException Permet d'encapsuler une exception pour ne pas gêner l'affichage utilisateur. Lors d'un ServiceException, par exemple, l'envoie d'un WebException permet d'afficher une page d'erreur personnalisée.

Le code des exceptions est plutot simple et court. Voici, par exemple, le code pour NotFoundException.

```java
@ResponseStatus(value=HttpStatus.NOT_FOUND)
public class NotFoundException extends Exception {

    public NotFoundException() {
        super("La page demandée n'existe pas.");
    }

    public NotFoundException(String msg) {
        super(msg);
    }

}
```

FIGURE 2.3 – Code de NotFoundException

On y définit un simple message d'erreur et on change le code de réponse HTTP, grace à l'annotation ResponseStatus, pour qu'il envoie le code 404 et affiche la page adéquate.

F Authentification

Ce package permet de gérer les différentes classes communes à l'authentification de l'utilisateur pour ce qui est du module Rest et WebApp. On y retrouve 2 classes :

AuthentificationService Cette classe implémente l'interface "UserDetailsService". Elle permet pour un login donné d'aller récupérer l'utilisateur correspondant dans la base de données et retourne un objet de type "AuthentificatedUser".

AuthentificatedUser Représente l'utilisateur actuellement connecté. Cette classe implémente l'interface "UserDetails". On y retient notamment la classe Compte qui est la classe métier de la table "COMPTE" dans la base de données. Grace à cet attribut on pourra facilement obtenir les informations sur l'utilisateur connecté, par le biais de la méthode "Utilisateur getUtilisteur()".

L'authentification sera alors géré de manière différente selon le module. Dans le cas du module WebApp il s'agira d'une connexion par le biais d'une page web tandis que pour l'api Rest on utilisera Basic Authentification.

G Utils

i DateUtils

Cette classe permet de gérer certaines fonctionnalités au niveau des dates. On peut par exemple citer la méthode "boolean isBetween(Date, Date, Date)" qui permet de savoir si la dernière Date est comprise entre les 2 premières. Cette classe permet également d'aficher une date fournit en parametre au format "jour/mois/année" grâce à la méhode "String format(Date)"

ii Sha256PasswordEncoder

Cette classe permet de gérer le hachage des mots de passes. Elle implémente l'interface "PasswordEncoder". Celle ci fournit une méthode publique "String encode(CharSequence)" retournant en String le hachage, en sha256 dans notre cas, de la chaine de caractère fournit en paramètre. Cette classe est fournit à Spring Security afin de vérifier les informations saisies par l'utilisateur lors d'une connexion. Elle est également utilisé manuellement lors de l'enregistrement d'un nouveau client.

3 Application web

A Organisation des packages

i Packages

ii Resources

B Parler configuration de l'application

i application.properties

ii Configuration par le code

C Fichiers resources

i Freemarker

Page de base

Page d'erreur

Autre

ii Assets (images, js, ...)

D Parler authentification

i Gestion template et controller

ii Interceptor

E Exemple Route

F Exemple Form

G Classe InputOutputAttribute

H Exemple Controller

4 Application mobile

A API REST

i Organisation des packages

B Ionic

i Organisation des packages

5 Ressources

Github du projet : https://github.com/justine-martin-study/Equida
Trello : https://trello.com/b/jrKixhpu/equida-spring