Coffee-and-Co-Manager (Backend)

Installation

Dans l'archive qui vous a été fournie, vous trouverez un coffee-and-co-manager.war . Vous pouvez le lancer depuis le dossier avec java -jar coffee-and-co-manager.war .

Si vous souhaitez générer votre propre jar, faites un mvn clean package depuis la racine du projet. Pour l'exécuter, faites un java -jar target\coffee-and-co-manager.war .

Si vous souhaitez vous connecter en tant qu'admin sur le backend, un compte admin est disponible. Les identifiants sont email=admin@admin.fr et password=admin. Les identifiants doivent être passé à l'endpoint /login sous forme de formulaire (form-data).

Si vous souhaitez vous connecter en tant qu'user sur le backend, un compte user est disponible. Les identifiants sont email=user@user.fr et password=user.

Description fonctionnelle

Principe

Coffee-and-Co-Manager est l'application permettant de gérer des machines à café (et autres). On peut se connecter dessus, un administrateur peut créer des comptes, on peut récupérer les informations des utilisateurs, mais aussi ajouter et supprimer des machines à café, voir la liste des machines, changer leurs configurations et leurs stocks.

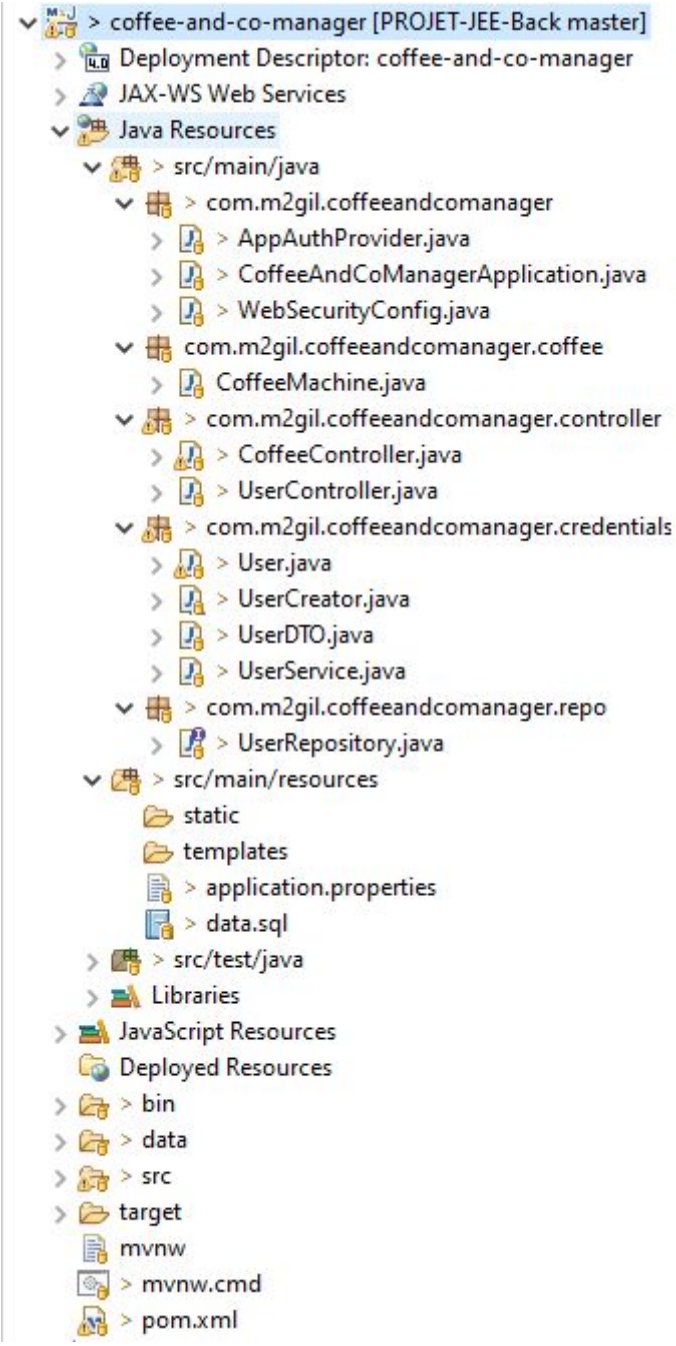
Description des API

Endpoints	Paramètres	Description	Rôles		
			Admin	User	Anonyme
POST /login	RequestBody : email, password	Permet de se Log In, avec ces identifiants			
POST /logout		Permet de se Log Out			
GET /user		Recupère les informations simples de l'utilisateur identifié courant			
GET /user/byEmail/{userEmail}	PathVariable : userEmail	Renvoie les informations de l'utilisateur de mail {userEmail}			
GET /user/byName/{userName}	PathVariable : userName	Renvoie les informations de l'utilisateur de nom {userName}			
GET /user/list		Récupère toutes les infos simples de tous les utilisateurs			
GET /user/info		Recupère les informations entières de l'utilisateur identifié courant. Inutile en pratique, je pense.			
GET /user/name		Retourne le nom de l'utilisateur courant			
GET /user/password		Retourne le mot de passe hashé de l'utilisateur courant			
GET /user/role		Retourne le role de l'utilisateur courant ("admin" ou "user")			
POST /user/register	RequestBody : User (email, username, password, role)	Permet d'enregistrer un nouvel utilisateur en BDD. Des règles sur les différents attributs du User sont instaurées (email, mdp, pas de double User en BDD, etc)			
PUT /user/modify	RequestBody : User (email, username, password, role)	Permet de modifier un User de la BDD. L'user en BDD est identifié par son email. On ne peut donc pas modifier son email. Seules les modifications valides sont autorisées. Si une modification sur un champ n'est pas valide, cette modification sur ce champ est ignorée (les autres modifications valides sont effectives).			
DELETE /user/delete/{email}	PathVariable : String email	Supprime l'utilisateur en BDD dont l'email vaut la chaine passée dans le corps de la requête.			
GET /data		Permet de récupérer toutes les machines à café.			
POST /data	RequestBody : CoffeeMachine (products, port)	Ajoute la Machine dans le Controller. Permet à la machine d'envoyer des données.			
POST /data/addMachine	RequestBody : String machineName	Inclue la machine dans l'application. Elle sera ajoutée lorsque elle effectuera son prochain appel à /data (POST)			
DELETE /data/deleteMachine/{machineName}	PathVariable : String machineName	Exclue la machine de l'application.			
GET /data/getConfig/{machineName}	PathVariable : machineName	Récupère la configuration de la machine nommée {machineName} (le nom correspond au nom mis dans l'attribut "port" des machines, récupérable lors d'un appel à /data (GET))			
POST /data/setConfig/{machineName}	PathVariable : machineName RequestBody : String mConf	Change la configuration de la machine nommée {machine name} (voir endpoint au dessus), pour la configuration mConf. La configuration mConf change en fonction des machines, c'est pourquoi on peut envoyer une String ou un objet en JSON.			
POST /data/addProduct/{productName}/to/{nomMachine}	PathVariable : String productName, String nomMachine	Ajoute un produit, de nom {productName}, dans la machine de nom {nomMachine} (ce nom équivaut à l'adresse IP + port)			
POST /data/addProduct/{productName}/to/{nomMachine}/quantity/{qty}	PathVariable : String productName, String nomMachine, int qty	Ajoute un nombre {qty} de produit, de nom {productName}, dans la machine de nom {nomMachine} (ce nom équivaut à l'adresse IP + port)			
POST /data/removeProduct/{productName}/to/{nomMachine}	PathVariable : String productName, String nomMachine	Retire un produit, de nom {productName}, dans la machine de nom {nomMachine} (ce nom équivaut à l'adresse IP + port)			
POST /data/removeProduct/{productName}/to/{nomMachine}/quantity/{qty}	PathVariable : String productName, String nomMachine, int qty	Retire un nombre {qty} de produit, de nom {productName}, dans la machine de nom {nomMachine} (ce nom équivaut à l'adresse IP + port)			

Cette image est disponible ici :
https://drive.google.com/file/d/1SZdo8cisaXg6iqoBFnoakY7lM_3geXnu/view?usp=sharing
Le tableau est disponible dans l'archive de rapport. Mais aussi ici :
https://docs.google.com/spreadsheets/d/180jL5G6UdzZSFIgnwgbixiNizlkGtybheZoRpo0kel0/edit?usp=sharing

Description technique

Description des packages



- coffeeandcomanager : Configuration de l'application, avec Spring Boot et Spring Security.
- coffee : Package ayant un rapport avec les machines à café
- controller : Controllers de l'application
- credentials : Package ayant un rapport avec l'authentification et les utilisateurs (User)

- repo : Interface permettant de communiquer avec la base de données et de récupérer les informations de la table contenant les utilisateurs
- src/main/resources : Dossier resources. Le script data.sql permet d'initialiser la BDD H2, et application.properties permet de définir les options de configuration nécessaires pour y accéder.

Bibliothèques utilisées

- Spring Security : Ajoute la couche sécurité à l'application
- Spring Data JPA : Ajoute la couche ORM à l'application et permet d'interagir connexion avec la base de données
- H2 : La base de données embarquée utilisée. C'est une base de données volatile, ce qui signifie qu'elle se réinitialise lors de l'arrêt de l'application. Néanmoins, on l'utilise via des fichiers, donc la BDD est persistante.
- Lombok : Automatise les tâches répétitives.
- Swagger : On a un Swagger de l'application. Néanmoins, on ne peut pas se connecter, donc on ne peut presque pas essayer les méthodes des controllers.

Plugin maven utilisés

- Spring Boot Maven Plugin : Embarque des commandes Maven

Choix techniques

Spring Security a été choisi car c'est une des technologies imposées par le prof. De plus, c'est un framework complet de contrôle d'accès, puissant et personnalisable, et un standard dans la sécurisation d'application Spring.

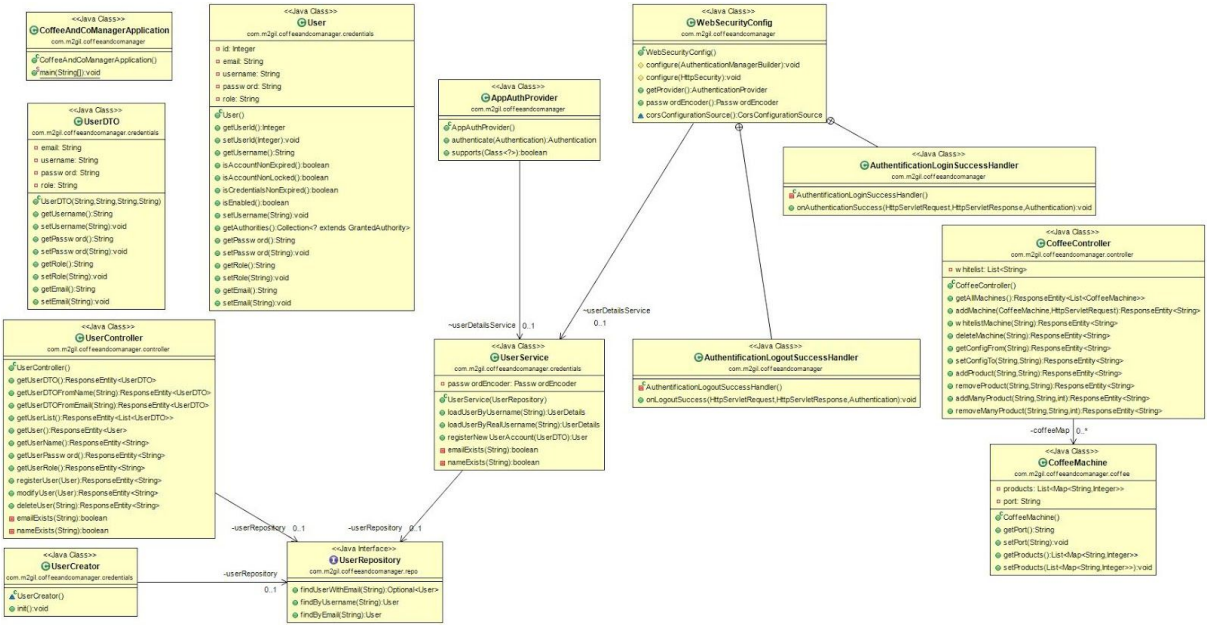
De la même façon, Spring Data JPA est un standard nous permettant d'implémenter la couche JPA.

Lombok a été choisi car la librairie permet de simplifier le développement en automatisant certains aspects de la programmation.

La base de données H2 a été choisi car c'est une base de données volatile, et l'apprentissage semblait intéressant. Il s'avère qu'il s'agit une base de données légère et rapide, et facile à manipuler. Elle est en plus embarquée dans l'application, ce qui facilite le déploiement de l'application. Le fait d'avoir en plus une console sur un endpoint personnalisable a pu énormément accélérer le développement de l'application.

Les utilisateurs sont stockées en base de données, leur mots de passe sont hashés pour la sécurité. Néanmoins, les machines à cafés sont gardées en mémoire dans le contrôleur des machines à café. Pour une application sur laquelle on stockerai beaucoup de machines, il faudra transitionner vers un stockage en BDD, en implémentant et utilisant un repository personnalisé.

Diagramme UML



Gestion des erreurs

Les erreurs sont normalement toutes gérées, l'application n'est pas censé s'arrêter. Néanmoins, les warning, les avertissements ne sont pas gérés : je pense en particulier à la récupération de la machine à café fournie et à ses valeurs pour chaque produits valant des entiers positifs, négatifs ou une String. Pour un entier négatif, le gestionnaire affichera qu'il y a 0 quantité du produit de disponible, donc pas d'erreur. Pour une String, le gestionnaire ne pourra pas parser l'objet reçu, affichera un warning sur la sortie standard et n'enregistrera pas la machine en mémoire.

Le problème qui nous as bloqué au final repose sur les dernières méthodes du CoffeeController. Les méthodes permettant de modifier les produits sur les machines sont inaccessibles sur l'application Front. On peut appeler ces quatres méthodes (addProduct, addManyProduct, removeProduct et removeManyProduct) via PostMan, mais on a une erreur 403 lorsqu'on le fait via Angular.