CAMARA Fatoumata-**M2 CCN** 2022/2023
MEHRABIJEY HOUNABADI Milad-**M2 IL** ISTIC

Compte rendu TP TAA

I-Description de l'API

L'API est une interface qui permet au étudiants de prendre rendez-vous avec des profs. Pour ce faire, on doit pouvoir ajouter, supprimer, modifier, afficher des profs et des étudiants dans la base de données à travers leurs informations respectives. On doit également pouvoir faire de même avec les rendez-vous entre ces deux groupes d'utilisateurs de notre API.

II-Le choix de l'architecture

Dans ce projet nous avons:

- Un package Config : Il est indépendant et contient la configuration swagger
- Un package aspects : Il est indépendant et contient la classe des aspects
- Un package Model (metier) : Il contient les classes entités.
- Un package service(DAO) : Il contient des interfaces DAO
- Un package Web (REST) : Il contient les Controllers.
- Un package DTO : pour mapper les objets Rendez-vous.
- Une base de données MySQL

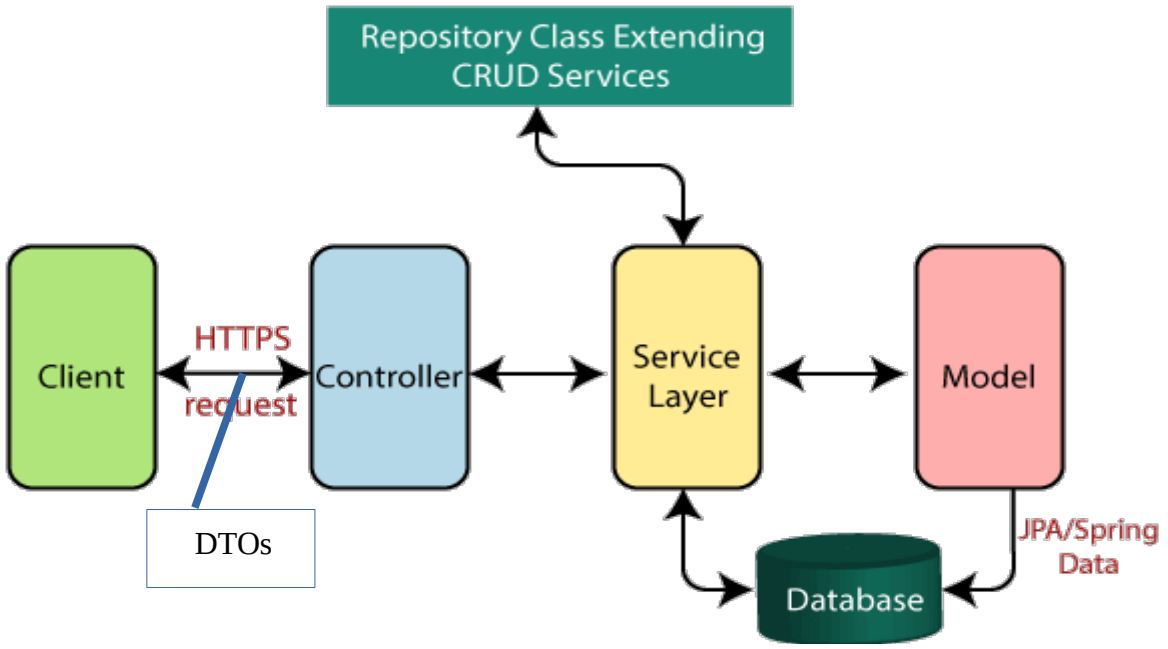

Figure1: Architecture du Back-end de l'API revisitée .

DAO (Data access object) :
nos interfaces DAO sont contenues dans la couche **''Service''.** Elles implémentent toutes **''JpaRepository''**, qui contient des méthodes d'accès à la base de données y compris les Opérations CRUD. Les annotations **@Repository** et **@Component** sont utilisées pour indiquer à Spring le rôle de communicant de ces interfaces avec la base de données.

DTO (Data transfert object) :
Nous avons une seule classe DTO (RdvDTO), qui a pour but de mapper nos rendez-vous en des objets qui respectent la structure de données de retour au format JSON des APIs. Cette classe prend , une date et un temps (au format String qui seront parsées ensuite), un id professeur et un id étudiant.

REST(Web) :

Cette couche comprend nos Controllers, trois classes associées respectivement aux étudiants(EtdController), aux professeurs(ProfController) et aux rendez-vous(RdvController). ils contiennent des méthodes implémentées qui agissent directement dans la base de données via des opérations d'insertion, de modification, de suppression et de récupération des données(create, delete, get-by-Id/emal, update). L'utilisation de @controller permet de gérer et valider les opérations. Les annotations @RequestBody et @Requesparam ont étés utilisées en fonction de si on liait le corps de la requête aux paramètres de la méthode ou si on liait les paramètres de la requête à un paramètre de la méthode du Controller.

Dépendances : L'injection de dépendances dans nos controllers par l'annotation **@Autowire** respecte le Pattern Singleton et s'assure qu'il n'y a qu'une seule instance utilisée des classes respectives.

III-Diagramme UML de la partie métier(Model)

Le package Domaine est composé de 4 classes concrètes visibles ci-dessous . Les classes EtdJPA et ProfJPA héritent de la classe UserJPA avec quelques attributs spécifiques à chacune des classes filles. Le « D-type » est généré

automatiquement dans la base de données comme marqueur de distinction .Le choix de la table unique pour professeurs et étudiants est motivé par l'obtention d'une visualisation d'ensemble facilitant les manipulations.
L'UML montre également les types de relations entre les différentes classes.

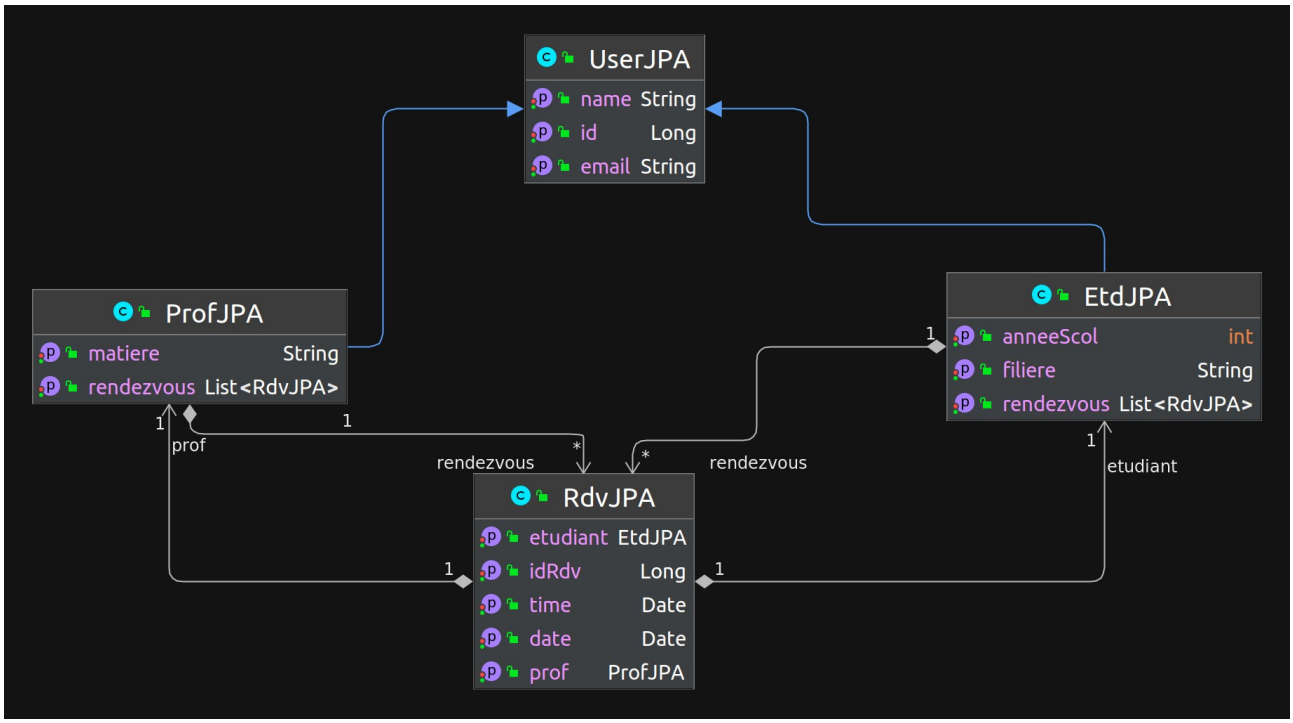

Figure2 : Diagramme des classes du package Model metier.
Les utilisateurs (UserJPA) y compris prof (ProfJPA) et étudiants (EtdJPA) peuvent avoir zéro à plusieurs rendez-vous(@OneToMany). Un à plusieurs rendez-vous peuvent concerner un étudiant et un prof à la fois(@ManyToOne).

IV-Description des classes

Nous avons utilisé l'annotation **@Entity** pour nos 4 classes du package Domaine, il permet à Spring de reconnaître et modéliser nos classes java en tables (entités) dans la base de données.

1-La classe UserJPA

Classe entité et première table de la base de données, c'est la classe mère de EtdJPA et ProfJPA, pour signifier cela, nous avons utilisé l'annotation

@Inheritance et avons précisé notre stratégie d'une table unique pour les 2 classes filles. **@Inheritance(strategy=InheritanceType.*SINGLE_TABLE*)**
elle définie les attributs communs à tous les utilisateurs (professeurs et étudiants) à savoir, un identifiant (id) généré automatiquement, un émail et un nom (name).

A)-La classe EtdJPA

Représente une classe entité fille de la classe ''UserJPA'', dont elle hérite, en plus des attributs hérités, elle définie deux autres attributs qui sont : filière et année scolaire(anneeScol) pour les étudiants.

B)-La classe ProfJPA

En tant que deuxième classe entité fille, elle hérite des attributs de UserJPA dans un constructeur et définie à son tour l'attribut ''matière'' en plus pour le professeur.

2-La classe RdvJPA

C'est la deuxième table de la base de données, classe entité, elle définie les attributs d'un rendez-vous entre professeurs et étudiants qui sont : un identifiant de rendez-vous(idRdv) généré automatiquement par les annotations @Id et @GeneratedValue, une date , une heure, un identifiant prof et un identifiant étudiant.

V-Exemples de requêtes

Les requêtes sont à tester dans Postman Swagger et via les formulaires(Front).
Précision : Les requetes sont toutes mappées sur (POST/@PostMapping), en lieu et place du @GetMapping, @PutMapping, @DeleteMapping. Cela pour ne pas avoir à dupliquer les méthodes pour Postman et pour le formulaire(front).

-Création d'un rendez-vous

localhost:8080/rdvcontroller/createRdv?date=2022-10-10&time=13:20:30&idProf=37&idEtd=36

-Suppression d'un rendez-vous

(POST)-→ localhost:8080/rdvcontroller/delete?idRdv=176

-Retrouver un rendez-vous par son identifiant

(POST)-→ localhost:8080/rdvcontroller/get-by-idRdv?idRdv=183

-Modification d'un rendez-vous

(POST)-→ localhost:8080/rdvcontroller/updaterdv?idRdv=66&date=2022-10-26&time=13:20:30

-Création d'un étudiant

(POST) → localhost:8080/etdcontroller/create?
name=Mamaman&email=maman@yahoo.fr&anneeScol=2022&filiere=Geomorpho

-Création d'un professeur

POST)-→ localhost:8080/profcontroller/create
name="Siddy"&email="sidy@yahoo.fr"&matiere="economie"

-Suppression d'un étudiant

(POST)-→ localhost:8080/etdcontroller/delete?id=185

-Suppression d'un professeur

(POST)-→ localhost:8080/profcontroller/delete?id=160

-Retrouver un étudiant par son adresse email

(POST)-→ localhost:8080/etdcontroller/get-by-email?email=gnamantoure@y.fr

-Retrouver un étudiant par son adresse email

(POST)-→ localhost:8080/profcontroller/get-by-email?email=sidy@yahoo.fr

-Modification d'un étudiant
(POST)-→ localhost:8080/etdcontroller/update?
id=36&email=mamatoure@fanta&name=MANIGBE

Modification d'un professeur

(POST)-→ localhost:8080/profcontroller/update?
id=37&email=sidymamoudou@yahoo&name=siddy diallo

VI-Front-end

La partie front-end se trouve dans le sous package « static » du package « ressources » du projet. Il est constitué de formulaires qui font appels aux méthodes des classes controllers pour effectuer les opérations CRUD.

-Création d'un rendez-vous → http://localhost:8080/CreateRdv.html

-Modification d'un rendez-vous → http://localhost:8080/UpdateRdv.html

-Suppression d'un rendez-vous → http://localhost:8080/DeleteRdvz.html

-Retrouver un rendez-vous par son identifiant →

http://localhost:8080/GetRdvById.html

-Création d'un étudiant → http://localhost:8080/CreateStudent.html

-Suppression d'un étudiant → http://localhost:8080/DeleteStudent.html

-Retrouver un étudiant par son adresse email -->

http://localhost:8080/GetStudentByemail.html

-Création d'un professeur → localhost:8080/CreateTeacher.html

-Suppression d'un professeur → http://localhost:8080/DeleteTeacher.html

-Modification d'un professeur → http://localhost:8080/UpdateTeacher

Utilisation de L'API :
Connexion à une base de données, run le « SampleDataJpaApplication ».
Ensuite, tester les requetes soit via swagger, postman ou les formulaires.
Swagger → http://localhost:8080/swagger-ui.html