Rapport projet M1 MIAGE – semestre 1

Sous la responsabilité de :

Yohan Boichut et Frédéric Moal

Réaliser par :

Houssam YACOUBI

Hassane BARRY

Ahmed Zakaria CHIKHI

Soumia Hadjar BRAHMI

Sommaire :

I. Introduction:

Dans le cadre de notre première année en master méthode informatique appliquée à la gestion d'entreprise, nous devons réaliser un projet nous permettant de mettre en pratique nos connaissances et nos compétences au travers d'un cahier de charges ayant pour finalité la conception et le développement d'un système de gestion de transports en commun.

En particulier, nous voulons un système permettant de gérer les abonnements des clients et leurs trajets. Un client doit valider son titre de transport à chaque changement de bus/ tram. Un titre est valide pendant une heure. Ce projet sera développé sur l'année complète.

Très passionné par le développement, notre groupe composé de Hassane BARRY, Houssam YACOUBI, Soumia BRAHMI et Ahmed CHIKHI avons saisi cet intérêt commun pour atteindre l'objectif attendu de ce projet. Sous la tutelle de Yohan BOICHUT & Frédéric MOAL.

II. OBJECTIFS ET CONTRAINTES

1. Fonctionnalités :

L'objectif initial de notre projet consiste à concevoir une application permettant de gérer les abonnement et tickets des utilisateurs en se basant sur pattern vus en cours, à savoir créer les Classes correspondantes et en ajoutant d'autres si nécessaire. Le point capital dans la réalisation de ce projet concerne la partie algorithmique. Notre application devrait être implémenté capable de répondre à la problématique, en optimisant au maximum ses performances. Enfin, il sera nécessaire de concevoir l'application qui devra proposer une interface simple et intuitive, ainsi qu'une aide interactive pour guider les utilisateurs novices.

Les fonctionnalités intransigeantes que devait contenir notre projet c'était les suivantes :

1. *Pouvoir s'inscrire à la plate-forme avec un nom et un mot de passe*
2. *Se désinscrire si l'utilisateur le souhaite*
3. *Se connecter à son espace personnel (et par la même occasion se déconnecter)*
4. *Souscrire à un abonnement de façon annuel ou mensuel*
5. *Commander un nombre de tickets précis (on peut acheter un ticket ou par pack de dix)*
6. *Valider un titre de transport sur un terminal dédié*

Heureusement, nous avons réussi faire marcher toutes ces contraintes bien avant le délai malgré plusieurs contraintes.

2. Contraintes & complications :

Premièrement les contraintes structurelles, nous disposons d'un repo git que nous devions utiliser pour l'ensemble du projet :

1. Code source commenté
2. Fichiers de configuration
3. Jeux de données de test/démo pour la validation du modèle

Notre travail est un projet Maven groupid:artifactId où : fr.miage.orleans.projetm1s2:groupe5. Le déploiement des bases de données s'est fait automatiquement par un docker-compose.

Deuxièmement, sachant que nous vivons les jours et mois les plus joyeux depuis la crise sanitaire que traverse le pays. Notre groupe a eu énormément de complications. D'abord, à aucun moment nous avons pu nous réunir pour travailler autour d'une table ronde, M.Yacoubi était confiné au Maroc, M. Barry a été à Nantes de base pour son alternance mais après le discours du président il a dû y rester, puis Mlle Brahmi était confinée avec sa famille à Marseille et M.Chikhi est resté à Orléans avec un travail à mi-temps. Ensuite, Mlle Brahmi a eu des soucis avec sa machine au niveau du git ce qui a obligé notre travail à ce qu'il ne soit pas fluide comme il aurait pu l'être. Par ailleurs nous avons eu des complications de communication par rapport à l'avancement de chacun malgré le fait que nous discutions par message, malheureusement nous n'avons jamais pu faire un appel vidéo tous réuni vu que chacun avait ses propres contraintes.

III. Planification du projet

Nos diverses expériences professionnelles et personnelles au cours de ces années d'études, nous ont permis d'aiguiser notre capacité à organiser la planification du projet, ainsi nous nous sommes départager les taches comme ci-dessous :

Tout d'abord nous avons réfléchi ensemble sur la compréhension du sujet. Puis nous avons rédigé la liste des besoins puis les contraintes ensemble, en découpant le projet en deux parties (La partie MySQL et MongoDB).

Pour la partie MySQL, elle permet de gérer l'inscription et la désinscription de l'utilisateur à la plateforme, ainsi la connexion et la déconnexion à son espace personnelle.

Ceux qui laisse la gestion des souscriptions à des abonnement mensuel et annuel, l'achat des tickets et la validation des titres de transport pour la partie MongoDB.

Ainsi Hassane Barry a réalisé :

- *Les inscriptions et connexion des utilisateurs avec une base sql*
- *Le déploiement de la base de données sql dans container docker avec docker compose*

Zakaria Chikhi a réalisé :

- *Implémentation de la façade transport et de ces différentes fonctions ainsi sa gestion d'erreurs(exceptions).*
- *Le déploiement de la base de données mongo dans container docker avec docker compose.*

- *La création d'un client mongo pour l'interface de l'application et la gestion de l'instanciation du client.*

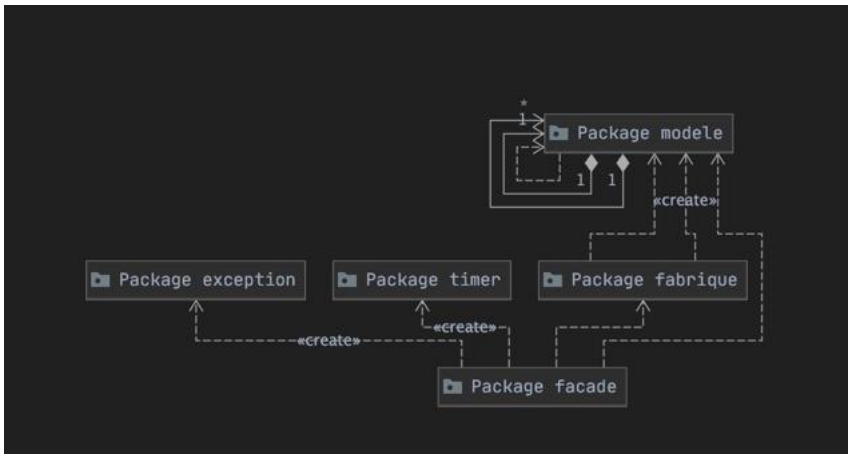
Houssam YACOUBI a réalisé :

- *Implémentation du modèle du projet et de ces différentes classes (Abonnement, Ticket, gestionAbonnement) ainsi les différentes fabriques pour chaque classe.*
- *Rédaction du rapport*

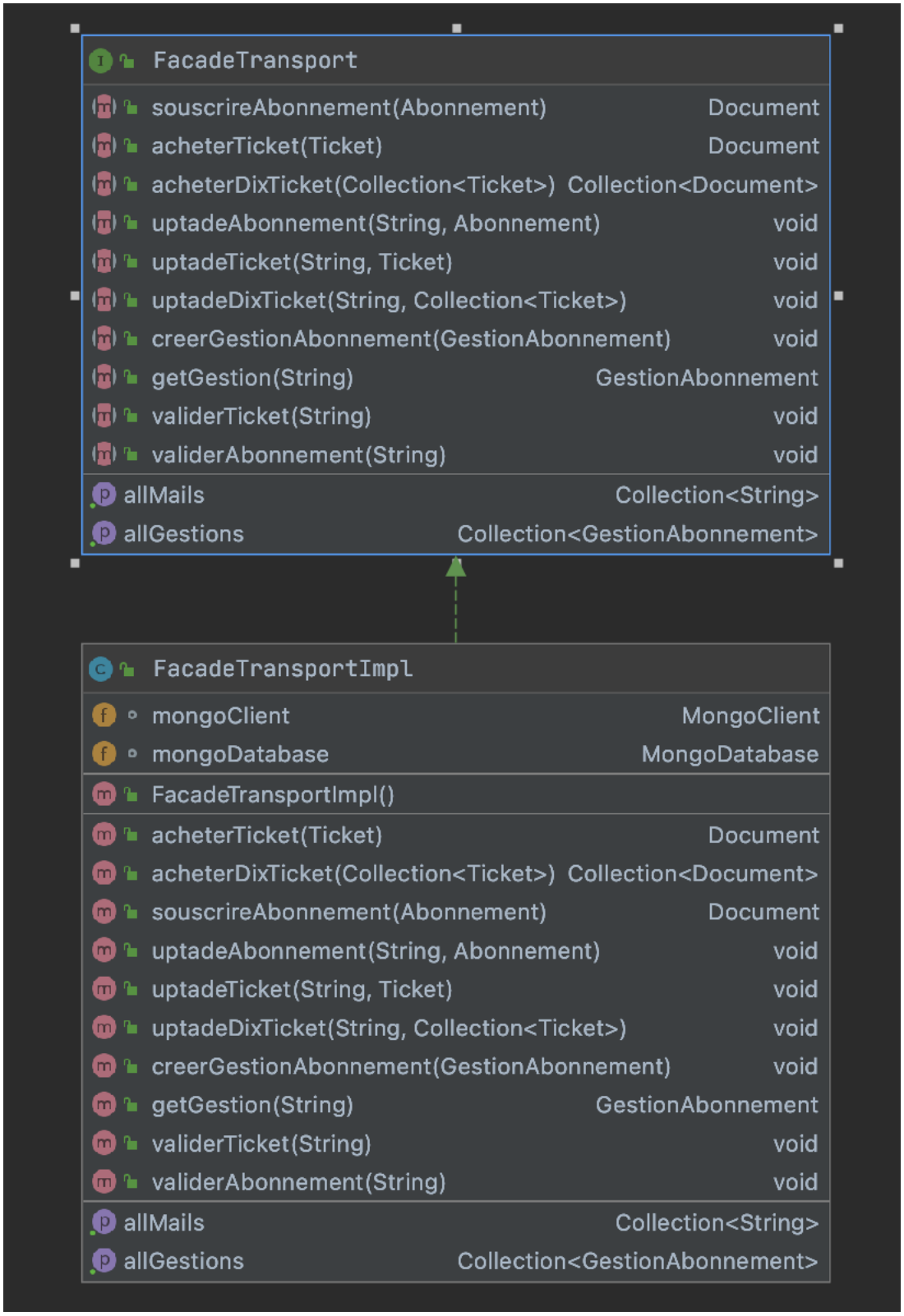
Bien sûr Nous nous sommes mis d'accord sur *La conception du modèle conceptuel des données sur papier.*

IV. Choix techniques et de conceptions :

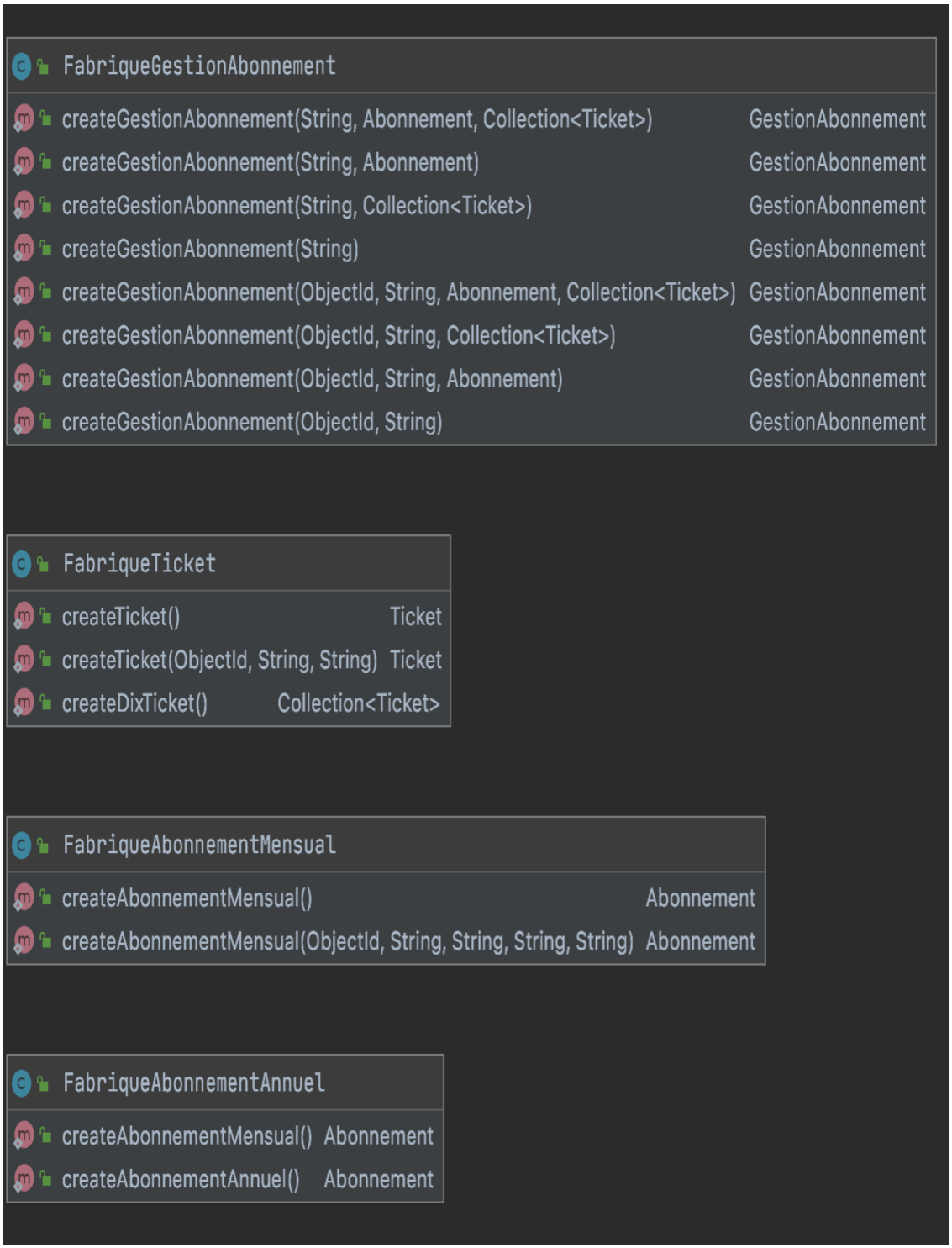
Tout d'abord, nous avons opté pour l'utilisation du design pattern Façade pour réaliser ce projet. Ainsi, nous avons l'uml ci-dessous qui représente le diagramme de **la partie mongo**, aussi les packages utiliser tel que timer, fabrique, exception, modele et façade :



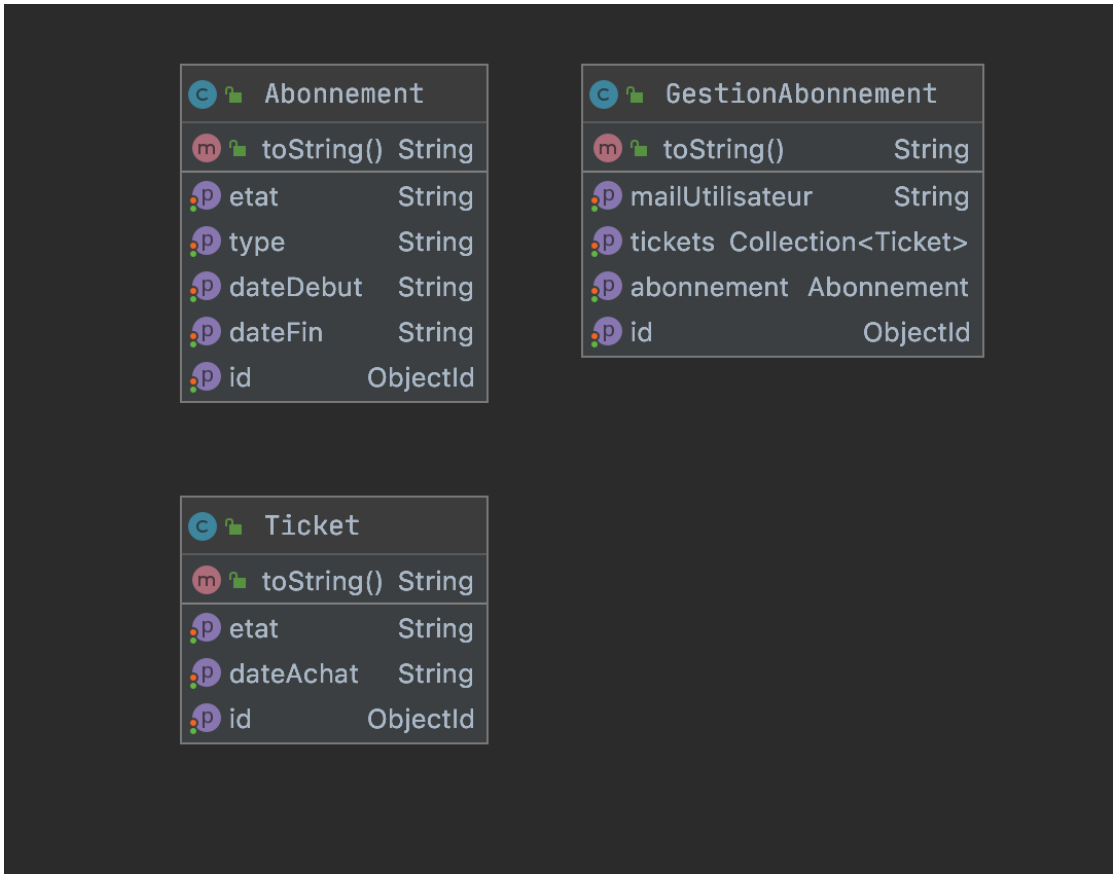
En outre nous retrouvons dans le package **facade** l'interface FacadeTransport qui permet d'acheter tout type de ticket, souscrire à un abonnement ainsi que valider son titre de transport. Cette interface est implémentée par facadeTransportImpl. Au final pour ce package, Nous avons ce diagramme qui résume toutes les méthodes :


FacadeTransport
souscrireAbonnement(Abonnement) Document
acheterTicket(Ticket) Document
acheterDixTicket(Collection<Ticket>) Collection<Document>
uptadeAbonnement(String, Abonnement) void
uptadeTicket(String, Ticket) void
uptadeDixTicket(String, Collection<Ticket>) void
creerGestionAbonnement(GestionAbonnement) void
getGestion(String) GestionAbonnement
validerTicket(String) void
validerAbonnement(String) void
allMails Collection<String>
allGestions Collection<GestionAbonnement>
FacadeTransportImpl
mongoClient MongoClient
mongoDatabase MongoDatabase
FacadeTransportImpl()
acheterTicket(Ticket) Document
acheterDixTicket(Collection<Ticket>) Collection<Document>
souscrireAbonnement(Abonnement) Document
uptadeAbonnement(String, Abonnement) void
uptadeTicket(String, Ticket) void
uptadeDixTicket(String, Collection<Ticket>) void
creerGestionAbonnement(GestionAbonnement) void
getGestion(String) GestionAbonnement
validerTicket(String) void
validerAbonnement(String) void
allMails Collection<String>
allGestions Collection<GestionAbonnement>

Ensuite, nous avons le package **fabrique** où tapissent toutes les classes permettant la création d'un ticket simple ou ticket de dix, la fabrication des deux types d'abonnement (mensuel & annuel) et finalement la création de gestion d'abonnement. Nous retrouvons diagramme à la fin :



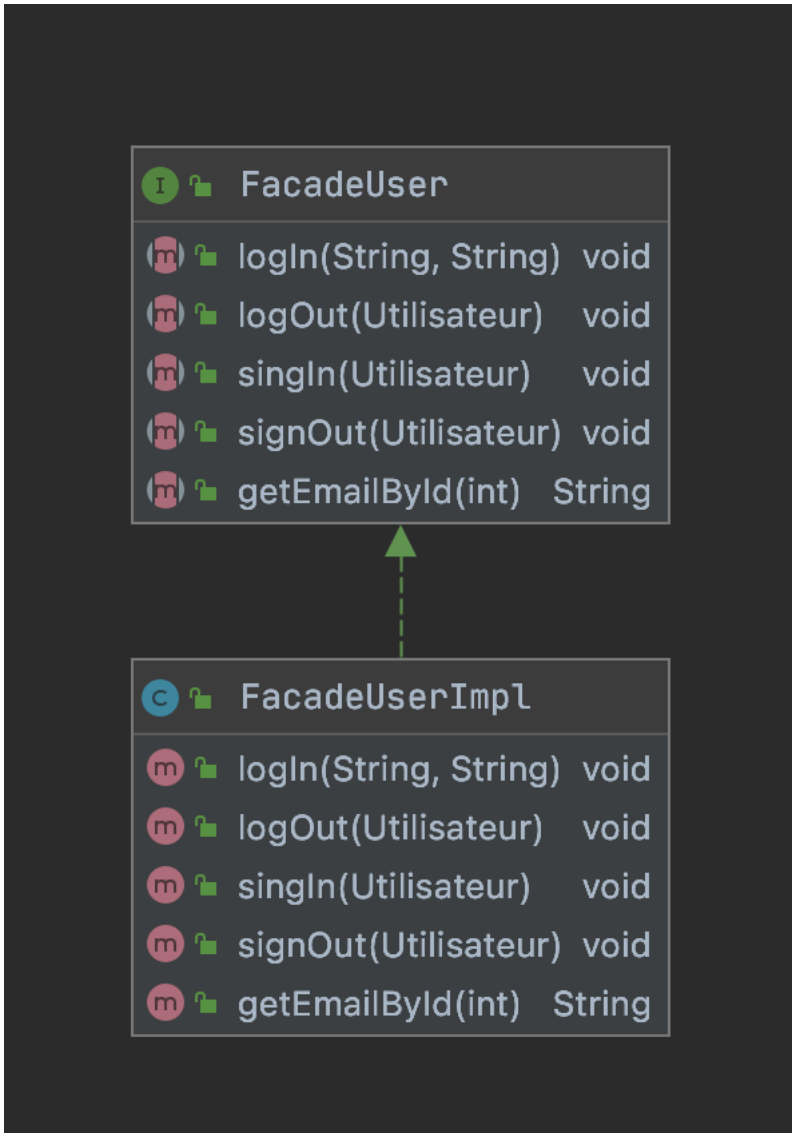
Pour le package **modele**, nous avons les trois classes Abonnement, gestion d'abonnement et ticket avec leurs méthodes :



Quant au package **exception**, nous avons toutes les exceptions que ça soit pour la non-validation du ticket ou de l'abonnement, si le mail est déjà dans la collection ou s'il n'a pas pu être retrouver.

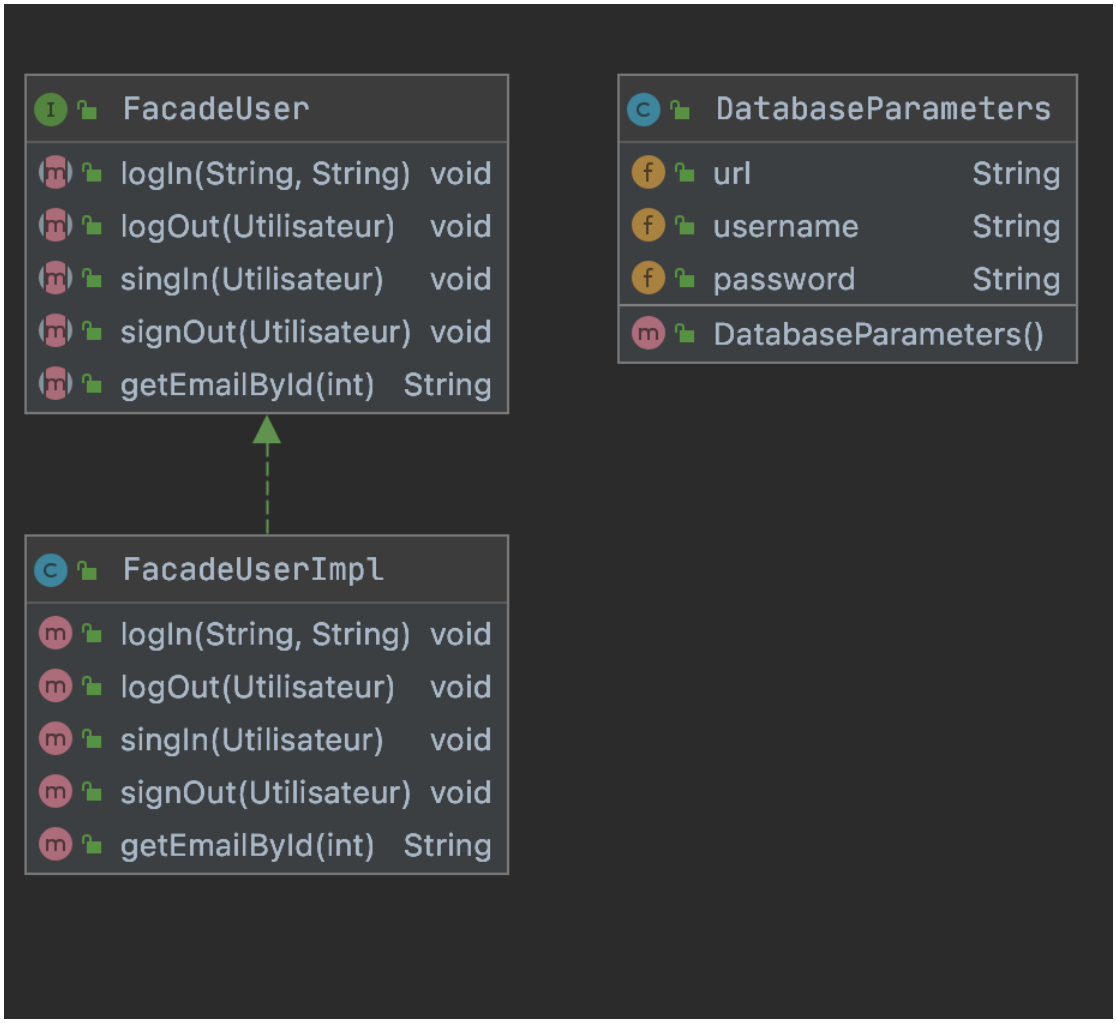
Finalement **timer** contient une seule classe TimerTicket qui hérite de TimerTask qui permet au ticket d'avoir une heure de validation.

Par la suite, nous avons la partie authentification où nous avons deux packages **modele** et **facade**. Dans le package facade nous avons l'interface FacadeUser qui est implémentée par FacadeUserImpl :


FacadeUser
logIn(String, String) void
logOut(Utilisateur) void
singIn(Utilisateur) void
signOut(Utilisateur) void
getEmailById(int) String
FacadeUserImpl
logIn(String, String) void
logOut(Utilisateur) void
singIn(Utilisateur) void
signOut(Utilisateur) void
getEmailById(int) String

Aussi dans le package modele nous avons la classe DataBaseParameters qui permet de parametrer l'accès à la base de données avec un url, username et un passeword.

Pour au final avoir le diagramme suivant :



Quant au déploiement de la base de données mongo il faudra suivre les étapes suivantes :

- Acceder au repertoire dockerCompose :
  cd DockerCompose
- Telecharger l'image dans un container :
  docker-compose up -d
- Acceder à un client Mongo:
  docker exec -it mongoTransportServer mongo
- Acceder à la base gestion de transport
  use gestionTransport
- Afficher la collection transport
  db.transport.find()

Aussi pour la base de données sql :

- Acceder au repertoire dockerCompose **cd DockerCompose**

*Telecharher l'image dans un container:
docker-compose up -d
*Acceder à un client mysql:
docker exec -it mysqlcompose_gesttrans_db_1 mysql -uroot -p
use gettrans
*Créer la table pour la première utilisation:
CREATE TABLE utilisateur (id smallint(5) unsigned NOT NULL AUTO_INCREMENT, nom varchar(100) NOT NULL, prenom varchar(60) NOT NULL, adresse varchar(200) DEFAULT NULL, username varchar (20) DEFAULT NULL, email varbinary(100) DEFAULT NULL, password varchar (20) DEFAULT NULL, date_naissance date DEFAULT NULL, PRIMARY KEY (id), UNIQUE KEY ind_uni_email (email));

V. *Architecture logicielle*

Pour ce projet, nous avons eu recours au design pattern Facade. Une façade est une classe qui procure une interface simple vers un sous-système complexe de parties mobiles. Les fonctionnalités proposées par la façade seront plus limitées que si vous interagissiez directement avec le sous-système, mais vous pouvez vous contenter de n'inclure que les fonctionnalités qui intéressent votre client.

Une façade se révèle très pratique si votre application n'a besoin que d'une partie des fonctionnalités d'une librairie sophistiquée parmi les nombreuses qu'elle propose.

Par exemple, une application qui envoie des petites vidéos de chats comiques sur des réseaux sociaux peut potentiellement utiliser une librairie de conversion vidéo professionnelle. Mais la seule chose dont vous ayez réellement besoin, c'est d'une classe dotée d'une méthode encoder (fichier, format). Après avoir créé cette classe et intégré la librairie de conversion vidéo, votre façade est opérationnelle.

VI. Conclusion

Tout d'abord, nous tenons à remercier tout particulièrement et à témoigner toute notre reconnaissance aux personnes suivantes, pour leur dévouement et leur soutient dans la concrétisation de ce projet : nos professeurs du module : « Methodes Avancées de Conception » M. Fréderic Moal et M. Yohan Boichut qui grâce à eux nous avons acquis de nouvelles compétences en conception et développement, compétences indispensables lors de la réalisation de ce projet.

En espérant, pouvoir terminer ce projet au deuxième semestre en présentiel.