Guide d'installation et de maintenance

PRUNIER Lucas, MEHR Ambre,
MALO Clotilde, BASSET Stéphane
C2

2024-2025

Sommaire

1 Introduction 3
1.1 Présentation de l'application 3
1.2 Objectif du guide 3
2. Guide d'installation 4
2.1. Déploiement de l'API 4
2.2. Installation du client 6
3. Guide de maintenance 7
3.1. Partie client 7
3.1.1. Le Model/ logique 7
3.1.2. La vue-modèle 8
3.1.3. La vue 9
3.1.4. Le network / réseau 9
3.2. Partie Serveur 11
3.2.1. Le contrôleur 11
3.2.2. Le stockage 12
3.2.3. Le service 12
3.3. Comment procéder pour faire une fonctionnalité ? 14
3.4. La base de données 15
3.4.1. MCD et MLD 15

1 Introduction

1.1 Présentation de l'application

L'application PARE est une application d'aide à la répartition des enseignements entre les enseignants tout au long du semestre destiné au personnel de l'IUT. Qui a pour objectif de faciliter le placement des modules pour répondre aux contraintes de chacun.
Grâce à PARE le corps enseignant, a la possibilité de positionner les modules sur des plages de semaine en prenant compte des contraintes.
PARE espère vous fournir un accès et une gestion plus simple pour le positionnement des heures du corps enseignant de l'IUT.
Cette application est créée à destination d'IUT-Corp et a été réalisé sous forme de sprint dans lesquels nous avons fait valider chaque fonctionnalité à Matthieu Simonet, notre interlocuteur chez IUT-Corp.

1.2 Objectif du guide

Ce guide est là pour vous aider dans l'installation et dans la maintenance de l'application en vous donnant toutes les clés pour vous faciliter les choses. Vous retrouverez dans les prochaines pages du guide les détails de chaque étape.

2. Guide d'installation

2.1. Déploiement de l'API

Le prérequis pour pouvoir déployer l'API est d'avoir Docker (configurer pour des systèmes Linux) sur la machine qui va être utilisé pour la déployer.
Pour cela, il faut commencer par récupérer l'image Docker de l'API.
Il existe une version disponible sur Github, mais vous pouvez aussi créer la vôtre en recompilant le projet depuis Visual Studio.

Récupérer de l'image Docker

Depuis Github

Dans la partie « Release » sur Github, cliquez sur le tag et affichez les fichiers disponibles.
Le fichier « sae5-page-api.tgz » est l'image Docker de l'API. Téléchargez-le et vous pourrez l'envoyer sur votre serveur Docker.

Compiler depuis Visual Studio

Pour cela, lancer le projet API dans Visual Studio en sélectionnant Release, le projet API et la configuration Container (Dockerfile).

Il est nécessaire d'avoir Docker installé et en fonctionnement sur la machine pour pouvoir lancer le projet.
Quand tout est en place, on lance le projet en cliquant sur la flèche verte creuse. Le Swagger, interface permettant d'envoyer des requêtes à l'API, devrait s'ouvrir dans votre navigateur internet.

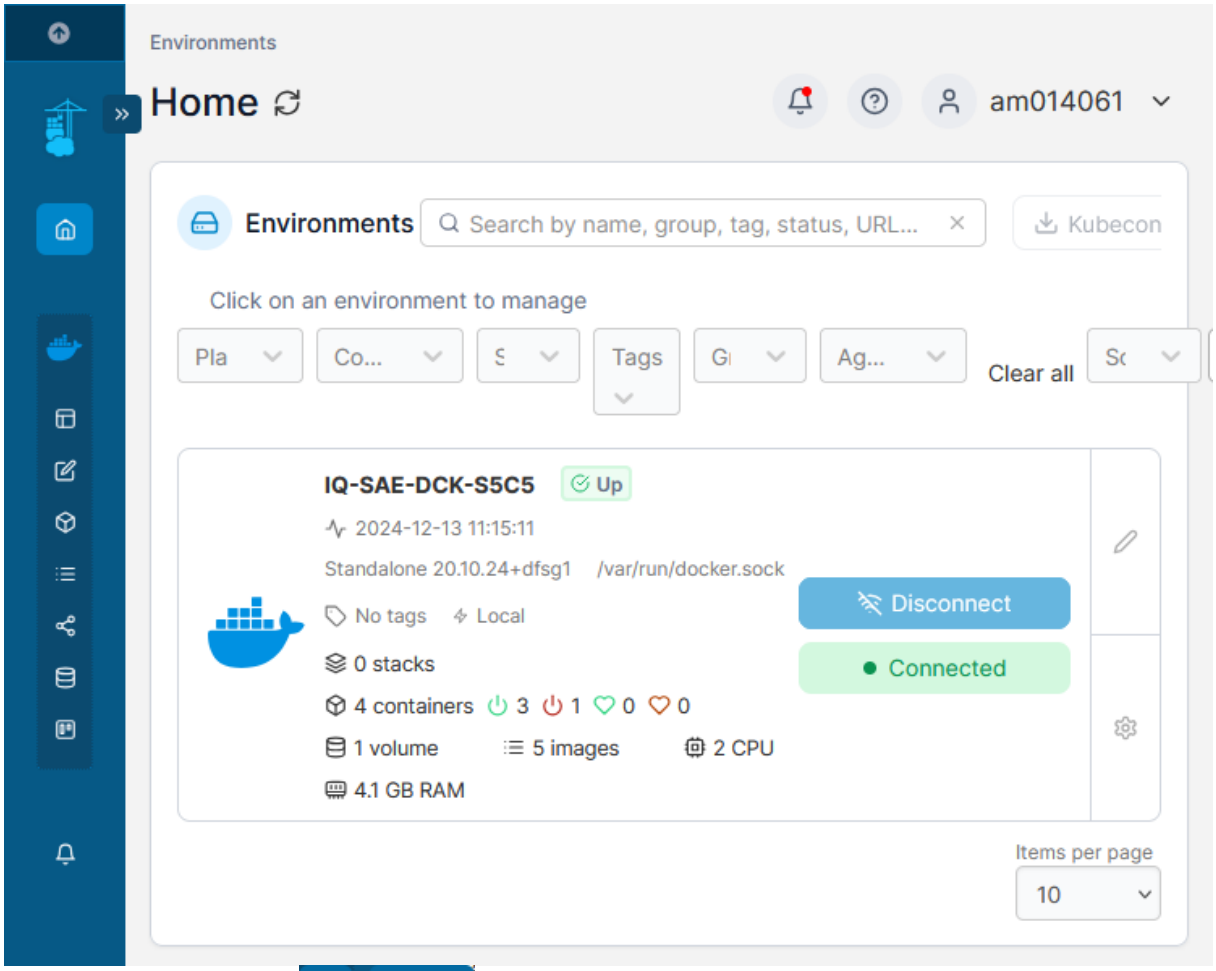
Une fois tout cela fait, vous pouvez fermer Visual Studio et ouvrir un terminal WSL, et vous placer dans le répertoire de votre choix.

```
ambre@Nocta-MB-W11:~$ docker image list
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
api          latest    860f6837fa52   24 minutes ago   251MB
ambre@Nocta-MB-W11:~$ docker save api:latest | gzip > PARE-Api.tgz
```

La commande « docker image list » affiche la liste des images docker sur le poste.
On sauvegarde ensuite l'image souhaité avec la commande « docker save api:latest | gzip > PARE-Api.tgz » qui nous donne un fichier PARE-Api.tgz, qui est l'image à envoyer sur le serveur Docker pour la suite des étapes.

Transférer l'image sur Docker

Connectez vous à votre environnement Docker, soit via ligne de commande, soit par Portainer.
Ici, nous montrerons comment faire la suite des étapes sur le réseau de l'IUT, en se connectant sur Portainer.
Sélectionnez l'environnement sur lequel vous voulez déployer l'API.



Cliquez sur « Images » 

Puis sur « Import »
Sélectionnez le fichier d'image Docker, et cliquez sur « Upload ».

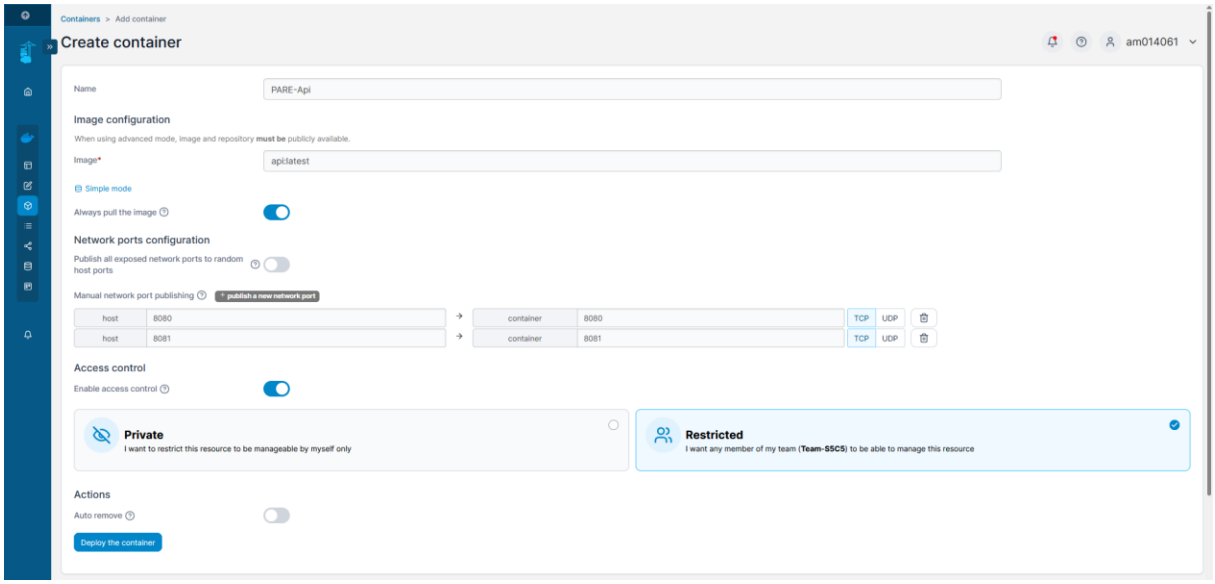
Créer un conteneur avec l'image

Cliquez sur « Containers » 

, puis sur « Add container ».
Remplissez la page avec les informations suivantes :

- Image en mode avancé : api:latest
- Manual network publishing :
  - 8080 -> 8080 TCP
- 8081 -> 8081 TCP

Puis, cliquez sur « Deploy the container ».

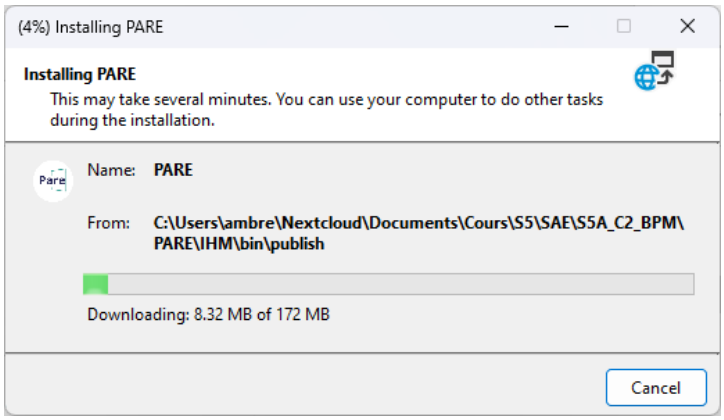


Voici l'API déployée !

Notez bien l'URL de l'API et les ports utilisés.
Le port 8081 sera le seul utile, permettant la connexion HTTPS à l'API.

2.2. Installation du client

Ouvrez le dossier publish vous trouverez un fichier « setup.exe », double-cliquez dessus et laissez l'installeur se faire tout seul.

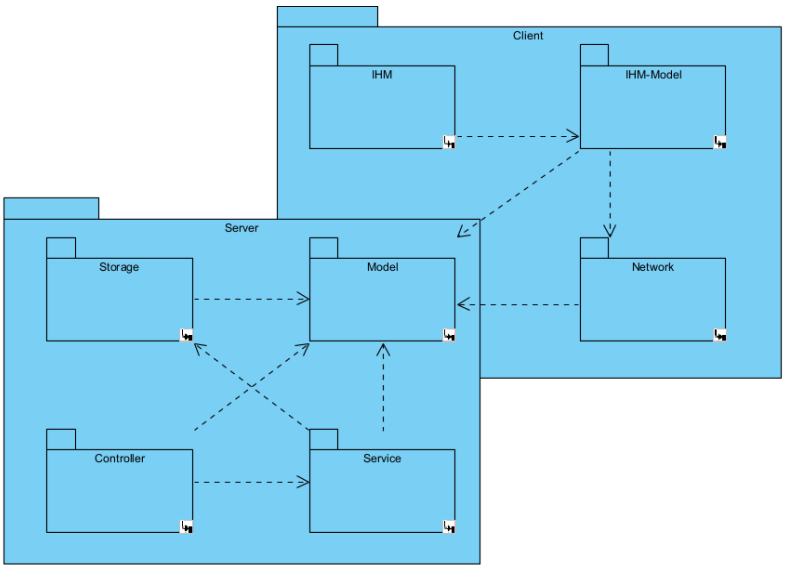


Une fois terminé, l'application se lance toute seul.
Vous pourrez également la retrouvez sur votre ordinateur au nom de PARE.

Pour avoir accès à l'API qui est déjà déployé en ouvrant le client il faudra se trouver sur le réseau de l'IUT (sur place ou en passant par un VPN). Si vous avez déployé l'API sur votre propre serveur alors il faudra simplement modifier l'url de l'api dans le fichier networkConfig.json qui se trouve dans le projet Network afin de renvoyer vers sa nouvelle adresse.

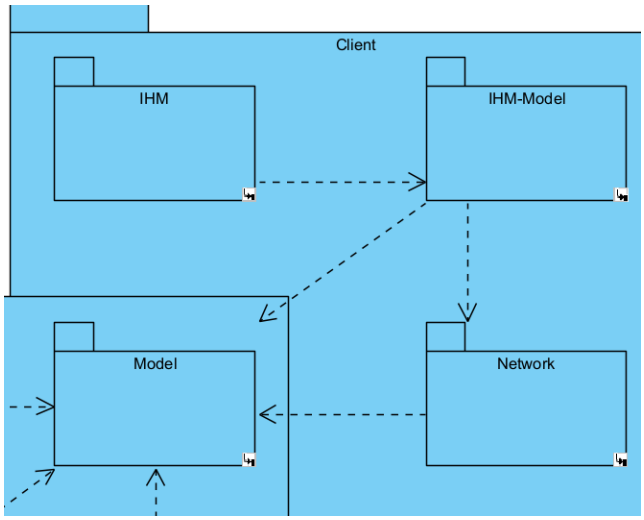
3. Guide de maintenance

Notre application est une application Client-Serveur divisée de cette façon :



3.1. Partie client

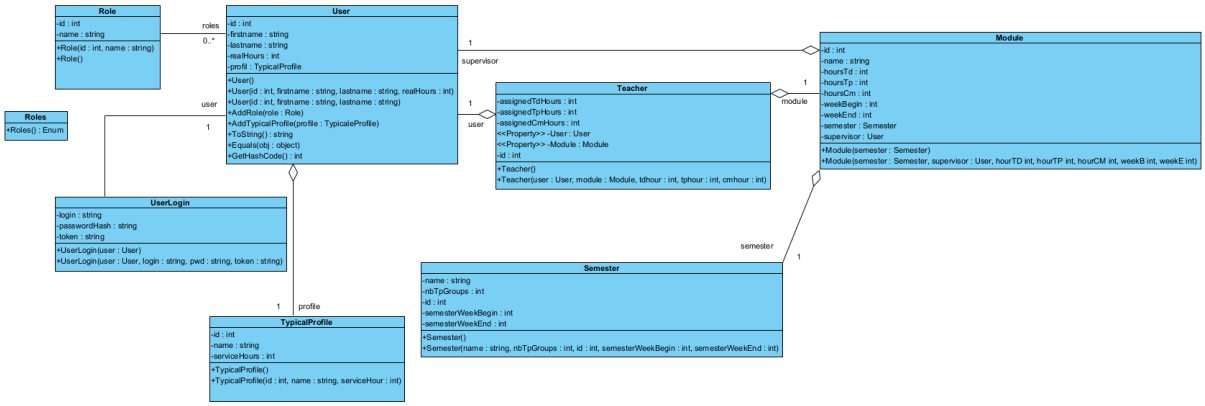
La structure globale de l'application est composée en plusieurs partie : le Model, la View, le View-Model (MVVM) aussi traduit en logique, vue et vue-modèle.
Dans notre application on les a nommés : IHM pour la vue, IHM-Model pour la vue-modèle, Model pour la logique.
En plus du MVVM, nous avons également une classe Network pour la partie réseau qui communique avec l'API.



3.1.1. Le Model/ logique

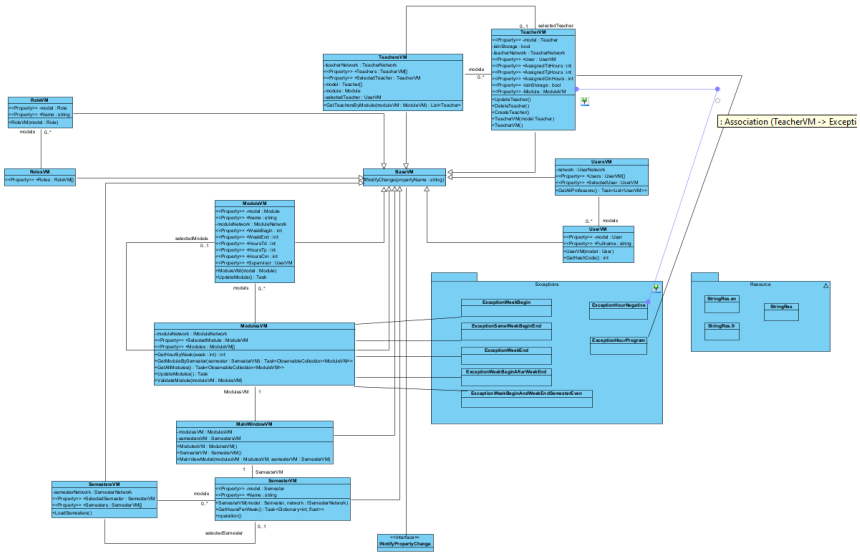
Le modèle représente la logique métier de l'application et la façon dont les données sont manipulées et traitées. Il contient les classes et les fonctionnalités nécessaires pour gérer les données de l'utilisateur : classe User, avec les données liées à l'utilisateur (rôle : classe Role, profil type : classe TypicalProfile).
On retrouve également les données liées aux modules : classe Module (aussi appelé ressources ou bien cours) avec une liaison entre le module et l'utilisateur par un enseignant :

classe Teacher (attention un enseignant n'est pas un utilisateur mais un enseignant sur un cours, si un professeur intervient dans plusieurs modules alors il aura plusieurs objet enseignant).
Un module a également un responsable de ressource appelé « supervisor » qui est de type « User ».
Un module intervient sur un semestre : classe Semester et à chaque semestre on a un nombre de groupe de TP d'étudiants.



3.1.2. La vue-modèle

Dans la vue-modèle on fait le lien entre la partie modèle et la partie vue, c'est cette classe qui va se charger de faire l'appel au réseau pour récupérer les données pour que la vue puisse l'utiliser.
Certaines propriétés peuvent être ajoutés dans cette partie pour faciliter la récupération côté vue.
On retrouve également un dossier Exceptions dans lequel on retrouve les exceptions que l'on utilise.
Mais aussi, un dossier Ressource dans lequel on retrouve un dictionnaire de ressource avec nos textes d'exceptions en français et en anglais. Ceux-ci sont alors traduits selon la langue du système sur votre machine.

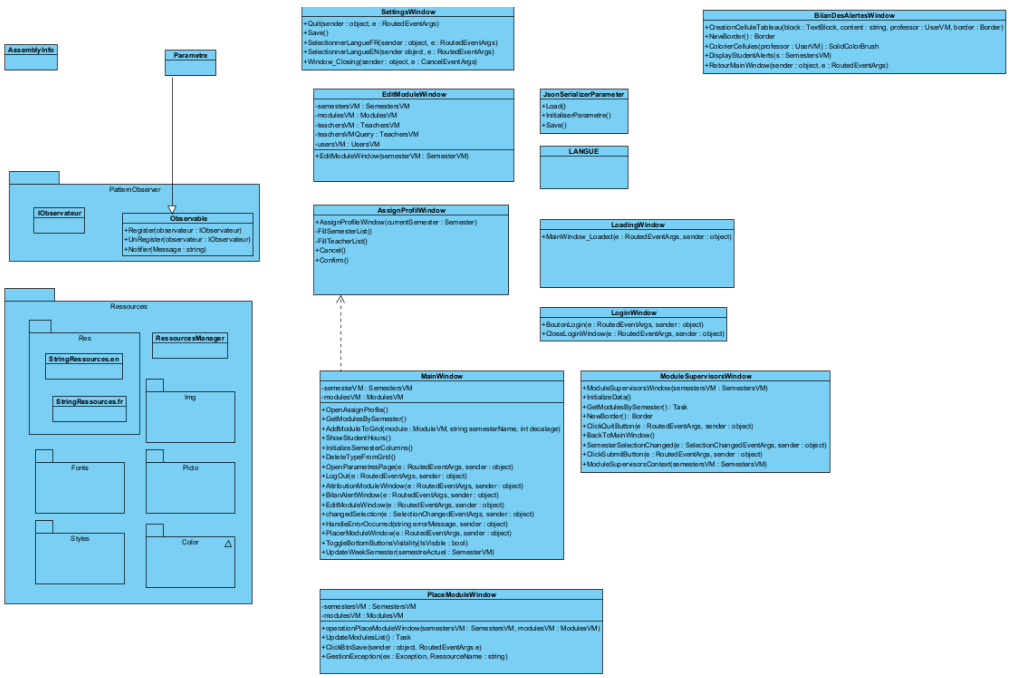


3.1.3. La vue

Dans la vue on retrouve toutes les fenêtres intervenant dans l'application mais également toutes les Ressources contenant les styles et couleurs utilisés dans l'application qui sont tous regroupés ici pour pouvoir être modifiés facilement.

Dans ces ressources on retrouve également les polices et images utilisées. Cette partie ressources contient un dossier « Res » dans lequel on retrouve tous les textes ajoutés dans l'application (en français et en anglais).

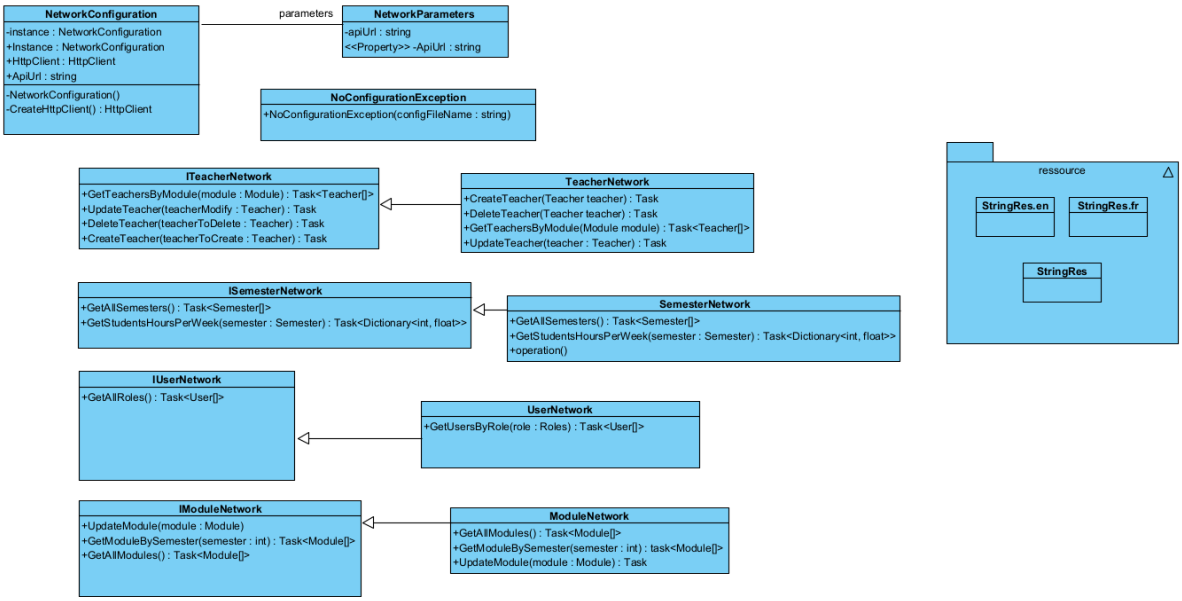
On retrouve aussi la partie de gestion de la langue dans la vue avec le pattern observeur (dossier PaternObserver) qui permet de récupérer dynamiquement les changements faits, mais aussi une classe qui permet de sérialiser et désérialiser en json. On retrouve également une énumération langue qui contient les langues que l'on gère : français et anglais.
Les textes que l'on ajoute directement dans la vue sont alors traduits selon la langue choisie dans l'application.



3.1.4. Le network / réseau

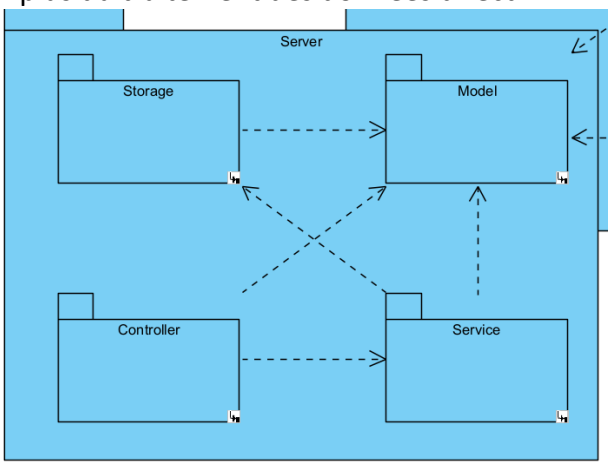
La partie réseau gère la récupération des données et donc tous les appels à l'API.
On retrouve alors la partie configuration avec « networkConfig » qui contient le lien de l'API. C'est donc ici qu'il faut modifier le lien si l'API est déployée sur un autre serveur. Il y a également NetworkConfiguration qui permet de définir la base des requêtes et de configurer la façon dont on communique avec le serveur (et la sécurité avec laquelle on communique) et NetworkParameters qui récupère et permet de sauvegarder le lien de l'API que l'on met dans networkConfig.

Cette partie contient donc également toutes les interfaces et toutes les classes utilisées pour communiquer directement avec la partie serveur.



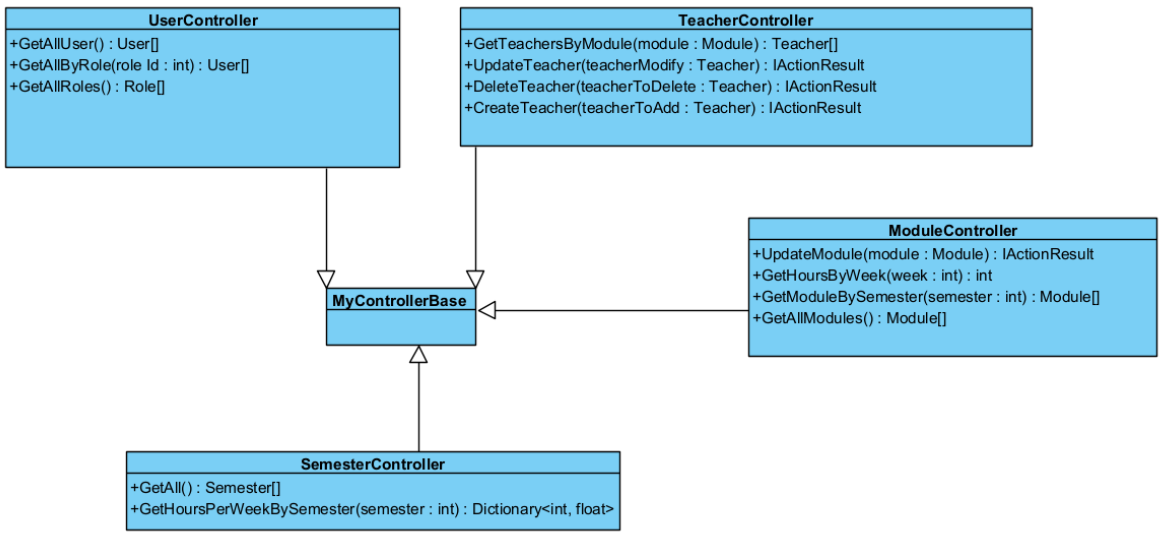
3.2. Partie Serveur

La partie serveur est composé de 4 parties : on retrouve encore le Model qui est la partie logique, il s'agit de la même partie que pour le côté Client.
Elle est également composée du controller (contrôleur) qui s'occupe d'être la porte d'entrée pour le client, il s'agit de la partie avec laquelle le client communique.
On peut aussi retrouver la partie storage (stockage) dans laquelle on retrouve tous les appels à la base de données avec toutes les requêtes et la conversion du retour de la base de données en un objet.
Enfin, il y a le service qui fait le lien entre le controller et le storage et qui permet de faire toutes les opérations en plus du traitement des données direct.



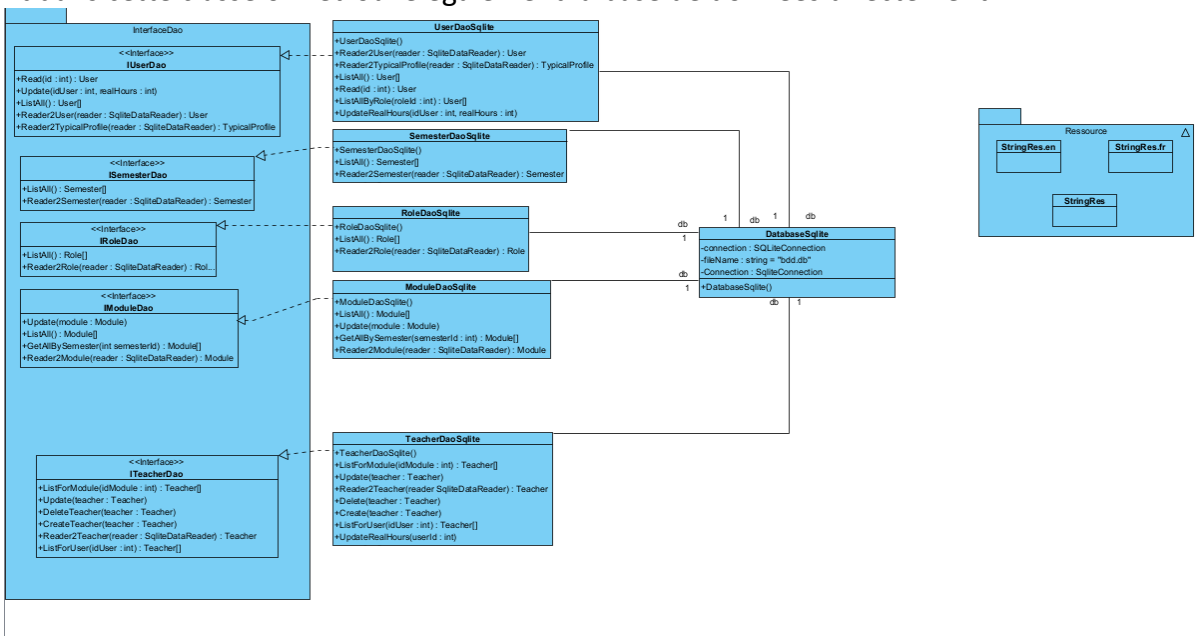
3.2.1. Le contrôleur

Le contrôleur comprend une base « MyControllerBase » dans laquelle on retrouve les initialisations d'instance de service pour éviter de les recréer dans chaque méthode des contrôleurs. On retrouve ensuite 4 contrôleurs dédié chacun à un objet : utilisateur (UserController), semestre (SemesterController), enseignant lié à un module et à un utilisateur (TeacherController), module (ModuleController).



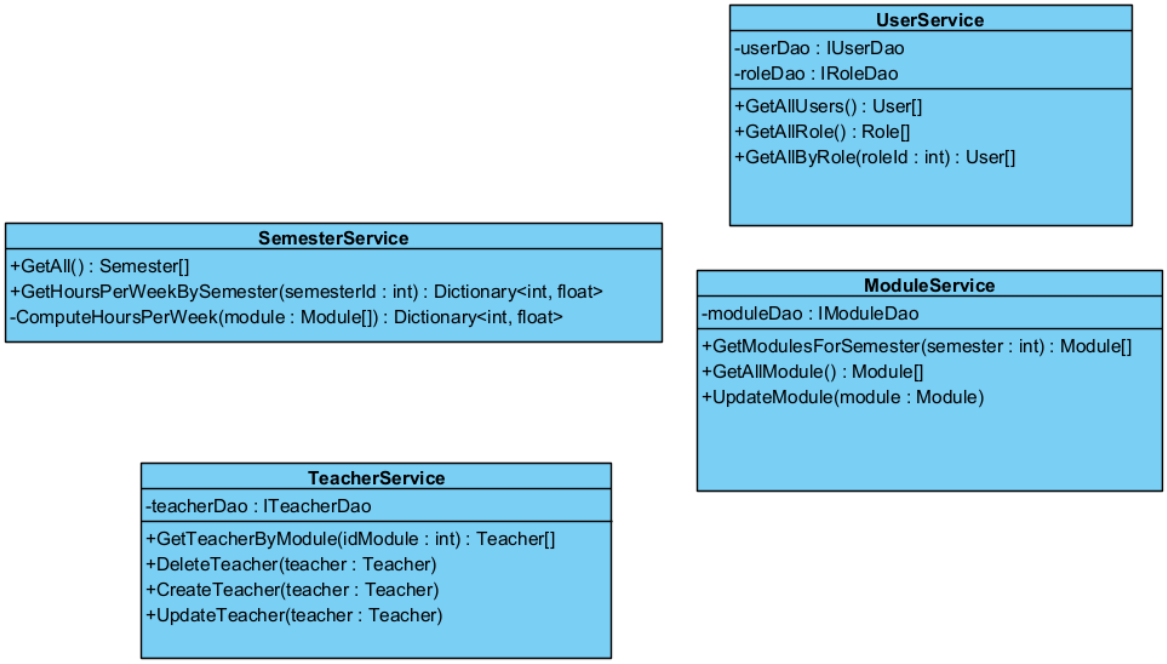
3.2.2. Le stockage

Le stockage contient des interfaces dédiées à chaque objet : module (IModuleDao), rôle (IRoleDao), semestre (ISemesterDao), enseignant lié à un module et à un utilisateur (ITeacherDao), utilisateur (IUserDao).
Les interfaces permettent de faciliter le travail de séparation des couches, si un jour il y a besoin de changer la technologie de la base de données le travail sera plus simple.
On a ensuite une classe correspondant à chaque interface mais sous forme de DaoSqlite.
Et dans cette classe on retrouve également la base de données directement.

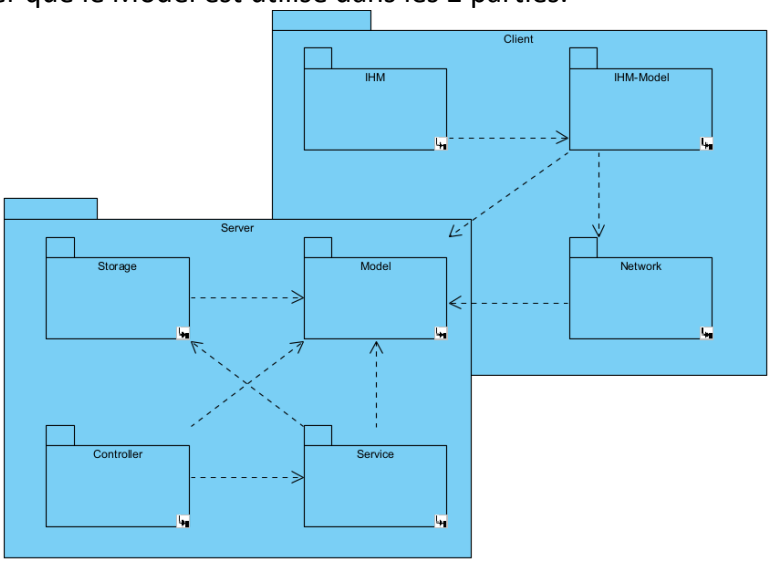


3.2.3. Le service

Le service contient également 4 services dédié chacun à un objet : utilisateur (UserService), semestre (SemesterService), enseignant lié à un module et à un utilisateur (TeacherService), module (ModuleService).
On l'utilise pour communiquer au stockage depuis le contrôleur et pour faire quelques opérations en plus sur ces données si nécessaires.



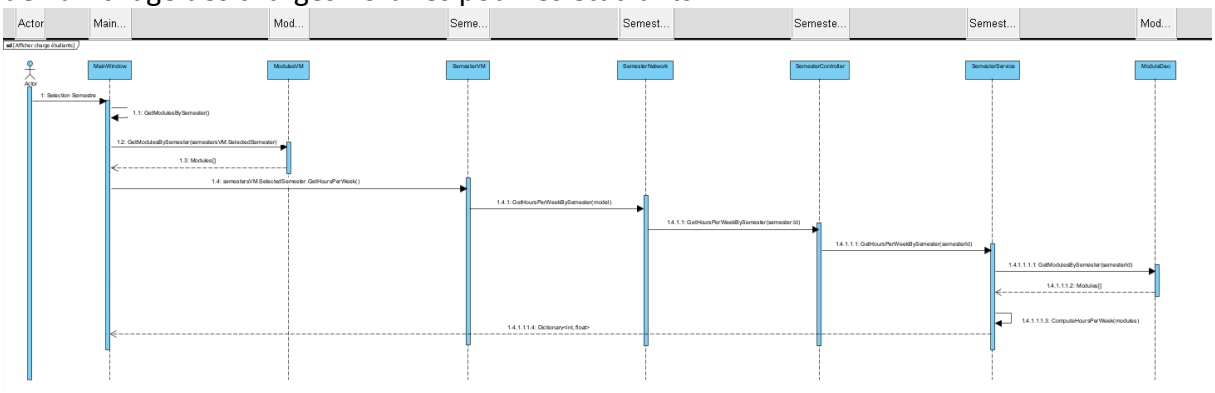
Voici le diagramme de package complet de l'application avec la partie client et la partie serveur. A noter que le Model est utilisé dans les 2 parties.



3.3. Comment procéder pour faire une fonctionnalité ?

Pour faire une fonctionnalité qui doit récupérer ou mettre à jour/ajouter/supprimer quelques choses dans l'application à partir d'une fenêtre, il faudra passer par :
IHM → IHM-Model (qui utilise le Model) → Network → Controller → Service → Storage

Voici un exemple d'une fonctionnalité que l'on a réalisé pour montrer le processus, il s'agit de l'affichage des charges horaires pour les étudiants :

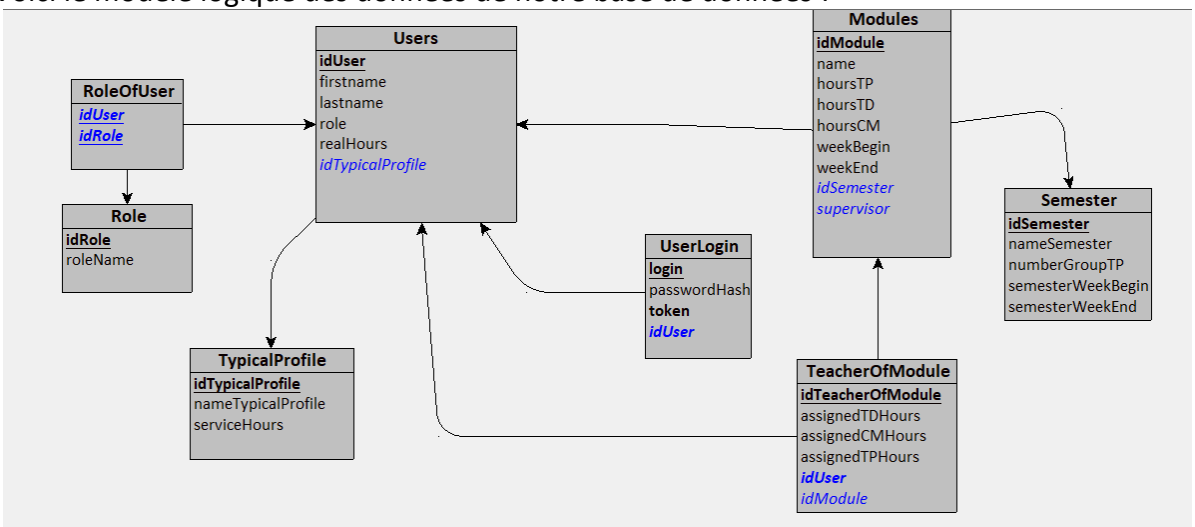


3.4. La base de données

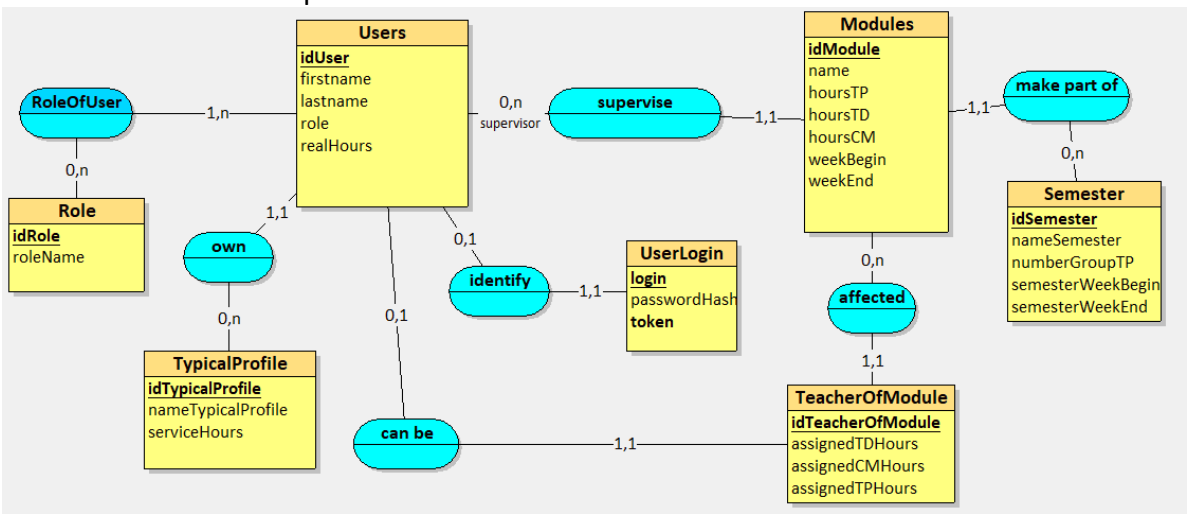
Notre base de données est faite en SQLite.

3.4.1. MCD et MLD

Voici le modèle logique des données de notre base de données :



Et voici le modèle conceptuel des données de notre base de données :



3.4.2. Script de création des tables

```
CREATE TABLE IF NOT EXISTS "TeacherOfModule" (
   "idTeacherOfModule" INTEGER,
   "assignedTDHours"   INTEGER,
   "assignedTPHours"   INTEGER,
   "assignedCMHours"   INTEGER,
   "idUser"    INTEGER,
   "idModule"  INTEGER,
   PRIMARY KEY("idTeacherOfModule"),
   FOREIGN KEY("idUser") REFERENCES "Users"("idUser"),
   FOREIGN KEY("idModule") REFERENCES "Modules"("idModule")
);
CREATE TABLE IF NOT EXISTS "TypicalProfile" (
```

```
    "idTypicalProfile"  INTEGER,
    "nameTypicalProfile"    TEXT,
    "serviceHours"  INTEGER,
    PRIMARY KEY("idTypicalProfile")
);
CREATE TABLE IF NOT EXISTS "Role" (
    "idRole"    INTEGER,
    "roleName"  TEXT,
    PRIMARY KEY("idRole")
);
CREATE TABLE IF NOT EXISTS "RoleOfUser" (
    "idUser"    INTEGER,
    "idRole"    INTEGER,
    FOREIGN KEY("idUser") REFERENCES "Users"("idUser"),
    FOREIGN KEY("idRole") REFERENCES "Role"("idRole")
);
CREATE TABLE IF NOT EXISTS "Users" (
    "idUser"    INTEGER,
    "firstname" TEXT,
    "lastname"  TEXT,
    "realHours" INTEGER,
    "idTypicalProfile"  INTEGER,
    PRIMARY KEY("idUser"),
    FOREIGN KEY("idTypicalProfile") REFERENCES "TypicalProfile"("idTypicalProfile")
);
CREATE TABLE IF NOT EXISTS "UserLogin" (
    "login" TEXT,
    "passwordHash"  TEXT,
    "token" TEXT,
    "idUser"    INTEGER,
    PRIMARY KEY("login"),
    FOREIGN KEY("idUser") REFERENCES "Users"("idUser")
);
CREATE TABLE IF NOT EXISTS "Modules" (
    "idModule"  INTEGER,
    "name"  TEXT,
    "hourTP"    INTEGER,
    "hourTD"    INTEGER,
    "hourCM"    INTEGER,
    "weekBegin" INTEGER,
    "weekEnd"   INTEGER,
    "idSemester"    INTEGER,
    "supervisor"    INTEGER,
    PRIMARY KEY("idModule"),
    FOREIGN KEY("idSemester") REFERENCES "Semester"("idSemester"),
    FOREIGN KEY("supervisor") REFERENCES "Users"("idUser")
);
CREATE TABLE IF NOT EXISTS "Semester" (
    "idSemester"    INTEGER,
    "nameSemester"  TEXT,
    "numberGroupTp" INTEGER,
    "SemesterWeekBegin" INTEGER,
    "SemesterWeekEnd"   INTEGER,
    PRIMARY KEY("idSemester" AUTOINCREMENT)
);
```

3.4.3. Script d'insertion de données

Voici les données que nous avons utilisé pour notre application :

```
INSERT INTO "TeacherOfModule" VALUES (1,1,2,1,4,5);
```

```
INSERT INTO "TeacherOfModule" VALUES (2,2,2,2,2,4);
INSERT INTO "TeacherOfModule" VALUES (3,6,8,3,5,4);
INSERT INTO "TeacherOfModule" VALUES (4,4,2,2,5,3);
INSERT INTO "TeacherOfModule" VALUES (5,2,4,3,4,2);
INSERT INTO "TeacherOfModule" VALUES (6,5,3,2,3,1);
INSERT INTO "TeacherOfModule" VALUES (7,5,2,3,2,1);
INSERT INTO "TypicalProfile" VALUES (1,'Maitre de conference',192);
INSERT INTO "TypicalProfile" VALUES (2,'Professeur 2nd degre',348);
INSERT INTO "Role" VALUES (1,'Admin');
INSERT INTO "Role" VALUES (2,'Chef de departement');
INSERT INTO "Role" VALUES (3,'Enseignant');
INSERT INTO "RoleOfUser" VALUES (1,1);
INSERT INTO "RoleOfUser" VALUES (2,2);
INSERT INTO "RoleOfUser" VALUES (3,3);
INSERT INTO "RoleOfUser" VALUES (4,3);
INSERT INTO "RoleOfUser" VALUES (5,3);
INSERT INTO "RoleOfUser" VALUES (2,3);
INSERT INTO "Users" VALUES (1,'Stephano','Bassot',NULL,NULL);
INSERT INTO "Users" VALUES (2,'Chirstophe','Cruz',59,1);
INSERT INTO "Users" VALUES (3,'Mehdi','Ment-souris',300,2);
INSERT INTO "Users" VALUES (4,'Patricia','Menissier',400,2);
INSERT INTO "Users" VALUES (5,'François','Tas-vingt',192,1);
INSERT INTO "UserLogin" VALUES
('stephanoBassot@test.fr','f3ce2c5929df0756e37992b5d6d0ce3afe7ef6e39985f00c894ed8ee37b34298',NULL,1);
INSERT INTO "UserLogin" VALUES
('ChirstopheCruz@test.fr','a75c246ef9f290376e0da3ae5939b06732b476a60ab199abafd9f48d7271f544',NULL,2);
INSERT INTO "UserLogin" VALUES
('MehdiMentsouris@test.fr','744bea40aeb7d7e98966836fec34272dded0e623f24bd63ce013112a3855175c',NULL,3);
INSERT INTO "UserLogin" VALUES
('PatriciaMenissier@test.fr','ee5df4560d05575ef4c37edb732fd340f5fd5d1a6028aa9e28d84c54a9f913cf',NULL,4);
INSERT INTO "UserLogin" VALUES
('FrançoisTasvingt@gmail.com','6f15f31e41b3cffbd18babc9a86f66965c2683be529f976e566053129180e09e',NULL,5);
INSERT INTO "Modules" VALUES (1,'R1.01 Math',10,5,5,37,40,1,2);
INSERT INTO "Modules" VALUES (2,'R1.02 Bdd',4,2,3,38,39,1,3);
INSERT INTO "Modules" VALUES (3,'R1.03 Architecture',2,4,2,40,47,1,4);
INSERT INTO "Modules" VALUES (4,'R1.04 Graphe',10,8,5,39,41,1,2);
INSERT INTO "Modules" VALUES (5,'R1.05 Ppp',2,1,1,40,46,1,4);
INSERT INTO "Modules" VALUES (6,'R3.06 Qualité dev',5,6,4,37,42,3,3);
INSERT INTO "Modules" VALUES (7,'R2.01 Anglais',10,8,0,3,5,2,2);
INSERT INTO "Modules" VALUES (8,'R4.01 Dev web',10,8,6,10,11,4,4);
INSERT INTO "Modules" VALUES (9,'R5.01 Dev avancé',6,2,4,36,38,5,3);
INSERT INTO "Modules" VALUES (10,'R6.01 Architecture',8,6,10,3,13,6,NULL);
INSERT INTO "Modules" VALUES (11,'R4.02 Qualité de développement',20,8,2,9,10,4,NULL);
INSERT INTO "Modules" VALUES (12,'R4.03 Qualité et au-delà du relationnel',10,8,4,10,11,4,NULL);
INSERT INTO "Modules" VALUES (13,'R4.04 Méthodes d''optimisation',10,6,2,11,12,4,NULL);
INSERT INTO "Modules" VALUES (14,'R4.05 Anglais',4,6,0,3,7,4,NULL);
INSERT INTO "Modules" VALUES (15,'R4.06 Communication interne',4,6,4,3,7,4,NULL);
INSERT INTO "Modules" VALUES (16,'R4.07 PPP',4,6,2,3,7,4,NULL);
INSERT INTO "Modules" VALUES (17,'R4.A.08 Virtualisation',4,6,2,3,7,4,NULL);
INSERT INTO "Modules" VALUES (18,'R4.A.09 Management avancé des SI',4,6,2,3,7,4,NULL);
INSERT INTO "Modules" VALUES (19,'R4.A.10 Complément web',4,6,2,3,7,4,NULL);
INSERT INTO "Modules" VALUES (20,'R4.A.11 Développement pour applications mobiles',4,6,2,3,7,4,NULL);
INSERT INTO "Modules" VALUES (21,'R4.A.12 Automates et langages',4,6,2,3,7,4,NULL);
INSERT INTO "Modules" VALUES (22,'S4.01',4,6,2,3,7,4,NULL);
INSERT INTO "Semester" VALUES (1,'Semestre 1',6,36,48);
INSERT INTO "Semester" VALUES (2,'Semestre 2',6,2,14);
INSERT INTO "Semester" VALUES (3,'Semestre 3',5,36,48);
INSERT INTO "Semester" VALUES (4,'Semestre 4',6,2,14);
INSERT INTO "Semester" VALUES (5,'Semestre 5',6,36,48);
INSERT INTO "Semester" VALUES (6,'Semestre 6',6,2,18);
```

3.5. Technologies et logiciels utilisés

Le client est fait en C# et WPF. Le serveur est une API Rest faite en C# avec ASP.NET CORE. La base de données est en SQLite. Un SDK 8.0 est utilisé.

Nous avons utilisé comme logiciels :

- Visual Studio : pour le développement du client et de l'API
- DB Browser for SQLite : pour la création de la base de données et l'insertion des données
- Docker Desktop : pour le déploiement de l'API
- Visual Paradigm : pour la conception