Master en informatique 1ère Année
Specialité Science et Technologie du Logiciel
Université Pierre Et Marie Curie

PROJET STL

« Clickeur » pédagogique sur smartphone :

partie serveur

Auteurs :

MEGHARI Aghiles

ARIB Yasssine

Encadrant :

Pr.SIGAUD Olivier

2016-2017

Contents

List of Figures

1 Introduction

Diverses formations de l'UPMC distribuent en amphi des « clikeurs » qui permettent aux étudiants de répondre à des questions posées pendant le cours, puis à l'enseignant de visualiser en direct l'ensemble des réponses, voire de les afficher en temps réel dans ses transparents. Ces solutions « propriétaires » ont un coût significatif et ne peuvent pas être customisées en fonction du besoin de chacun. Nous souhaitons proposer un système aux propriétés similaires, mais qui soit ouvert et qui repose sur l'utilisation du smartphone des étudiants. Le projet d'ensemble est découpé en 3 sous-parties.

Partie Étudiant Cette partie est réalisée par un autre binôme, qui utilise l'api fournie par le serveur pour récupérer les informations nécessaire pour son fonctionnement. L'objectif de cette partie est de réaliser la partie de l'application accessible aux étudiants. En pratique, les étudiants se connecteront via leur smartphone sur une page web spécifique de l'application, puis ils choisiront dans une liste l'Ue concerné, puis la séance à fin de répondre au questions posées par l'enseignant.

Partie Enseignant L'objectif de cette partie également réalisée par un autre binôme est de réaliser la partie de l'application accessible aux enseignants. En pratique, les enseignants se connecteront à l'application depuis un ordinateur sur une application hébergée à l'UPMC. Chaque enseignant pourra créer des Ues, des séances relatives aux ues précédemment créent ainsi que des questions pour chaque séance, et avoir un visuel concernant les statistiques des réponses sous forme d'histogrammes.

Partie Serveur C'est la partie qui nous intéresse le plus dans ce rapport car nous sommes les responsables de sa réalisation. L'objectif de cette partie est de réaliser la partie serveur et base de données du projet, gérer le logique métier ainsi que de faire liens entre les deux autres parties du projet (parti étudiant et enseignant). Ses principales taches sont de : traiter les requêtes, mettre à jour de la base de données, et répondre au client concerné.

2 Phase d'analyse

2.1 Description du fonctionnement général de l'application

Authentification: À la connexion sur l'application, l'utilisateur devra saisir son identifiant et son mot de passe. La vérification de l'identité de l'utilisateur se fera en interrogeant la base de données de l'université. L'utilisateur devra accéder à l'application pendant deux heures (durée de la cour) sans avoir à se ré-authentifier.

Création d'UE et de séances: Un enseignant peut créer une UE en saisissant un nom et un code d'Ue. Il peut créer une séance ayant un nom associé à une UE, en sélectionnant l'UE. On vérifie qu'on ne crée pas une UE qui existe déjà, ni une séance qui existe déjà pour l'UE choisie.

Inscriptions aux Ues: Un étudiant peut s'inscrire à une Ue à laquelle il n'est pas déjà inscrit en cliquant dessus. La liste des Ues se trouve dans la page d'accueil.

Saisie des nouvelles questions: L'enseignant doit choisir une UE et une séance. Il peut alors saisir des questions. Pour saisir une question, il doit saisir un label (champ textuel de quelques lignes max), puis saisir au moins deux propositions possibles dont au moins une est vraie, si ce n'est pas le cas l'opération est annulée. Pour cette seconde saisie, on affiche une liste de champs pour les propositions, l'enseignant valide quand il a fini. On lui propose alors de saisir une nouvelle question ou de revenir au menu principal.

Réponses aux questions : Une fois connectés, les étudiants devront choisir parmi les Ues auxquelles ils sont inscrits et dont il existe des questions ouvertes. Puis ils devront choisir une séance de cette UE. Puis ils devront choisir une question parmi celles de cette séance. Les questions parmi lesquelles choisir apparaît avec un numéro : Q1, Q2, Q3... Leur écran fera alors apparaître le libellé de la question et la liste des réponses possibles. Ils devront sélectionner une ou plusieurs réponses selon le type de la question (question à choix unique ou multiple, le type de la question est déterminé à sa création en calculant le nombre de propositions ayant un verdict vrai).

Système de réponses par tour: Imaginons le cas suivant : l'enseignant veut tester si les étudiants ont assimilé un point précis du cours, alors il leur propose de répondre une première fois (premier tour) à une question, il s'aperçoit en regardant les statistiques que les étudiants n'ont pas bien répondues. Il décide de ré-expliquer puis évalue une deuxième fois (deuxième tour) les étudiants. en visionnant les statistiques par tour, l'enseignant peut monitorer en temps réel la compréhension des étudiants. C'est dans cet objectif qu'a été développé cette fonctionnalité.

Publication et masquage des questions: Par défaut, des questions qui viennent d'être saisies sont masquées. L'enseignant qui les a saisies (et personne d'autre) peut choisir de les publier, ce qui a pour effet de les rendre accessibles au choix des étudiants. Il peut ensuite choisir de masquer à nouveau une question qu'il a publiée.

Suppression et mise à jour des questions: L'enseignant qui les a saisies (et personne d'autre) peut choisir de supprimer ou de mettre à jour des questions. S'il supprime une question, toutes les réponses d'étudiants à cette question stockées dans la base sont supprimées.

Consultation des réponses des étudiants: L'enseignant peut choisir une question et visualiser l'ensemble des réponses reçues à cette question : on lui affiche pour chaque choix possible le nombre de réponses reçues correspondantes.

Consultation des réponses par étudiant: L'enseignant peut alors sélectionner cet étudiant et visualiser ses réponses à toutes les questions (ou son taux de réponses correctes sur l'ensemble des questions).

2.2 Diagramme de class métier [1]

Le diagramme de classes constitue l'un des pivots essentiels de la modélisation. En effet, ce diagramme permet de donner la représentation statique du système à développer. Cette représentation est centrée sur les concepts de classe et d'association. Chaque classe se décrit par les données dont elle est responsable pour elle-même et vis-à-vis d'autres classes.

En se basant sur le cahier des charges, nous avons recensé 8 classes permettant clairement de définir les différentes tables de la base de donnée utilisées dans la réalisation du projet.

Users et Roles: La table users encapsule les données de l'utilisateur indifféremment du fait qu'il soit étudiant ou enseignant. La distinction se fait dans la table roles, effectivement on associe un rôle à chaque utilisateur.
l'attribution des rôles se fait en fonction de l'identifiant de l'utilisateur. Si l'identifiant est une chaîne de caractère numérique de longueur 7 alors il s'agit d'un étudiant, si ce n'est pas le cas alors c'est un enseignant.

Ues : Elle encapsule les informations relatives aux Ues, elle possède plusieurs relations :

- `avec un utilisateur ayant le rôle d'enseignant (1..N)`: c'est une relation d'appartenance autrement dit l'enseignant est l'unique propriétaire de cette Ue.
- `avec un utilisateur ayant le rôle d'étudiants (N..N)`: c'est une relation d'inscription. Une Ue peut avoir plusieurs inscrits et un étudiant peut s'inscrire à plusieurs Ues.
- `avec une séance (1..N)`: c'est une relation d'appartenance de plusieurs séances à une Ue. **Remarque:** la séance est l'équivalente d'un cour, cette table porte le nom sessions dans le schéma ci-dessus.

Questions,Propositions et Responses: La table question contient l'intitulé d'une question créée par l'enseignant, propositions contiennent un intitulé et un verdict et appartiennent à une question, Responses modélise le choix d'un utilisateur d'une ou plusieurs propositions relatives à une question donnée. Les relations liant tables sont décrites comme suit

- `Questions avec Propositions (1..N)`: C'est une relation décrivant l'appartenance de plusieurs propositions à une question.
- `Users avec Responses (1..N)`: un utilisateur peut choisir plusieurs propositions d'une question donnée ce qui permet de dire qu'un utilisateur peut avoir plusieurs réponses pour une question.
- `Sessions avec Questions (1..N)`: décrit l'appartenance de plusieurs questions à une séance.

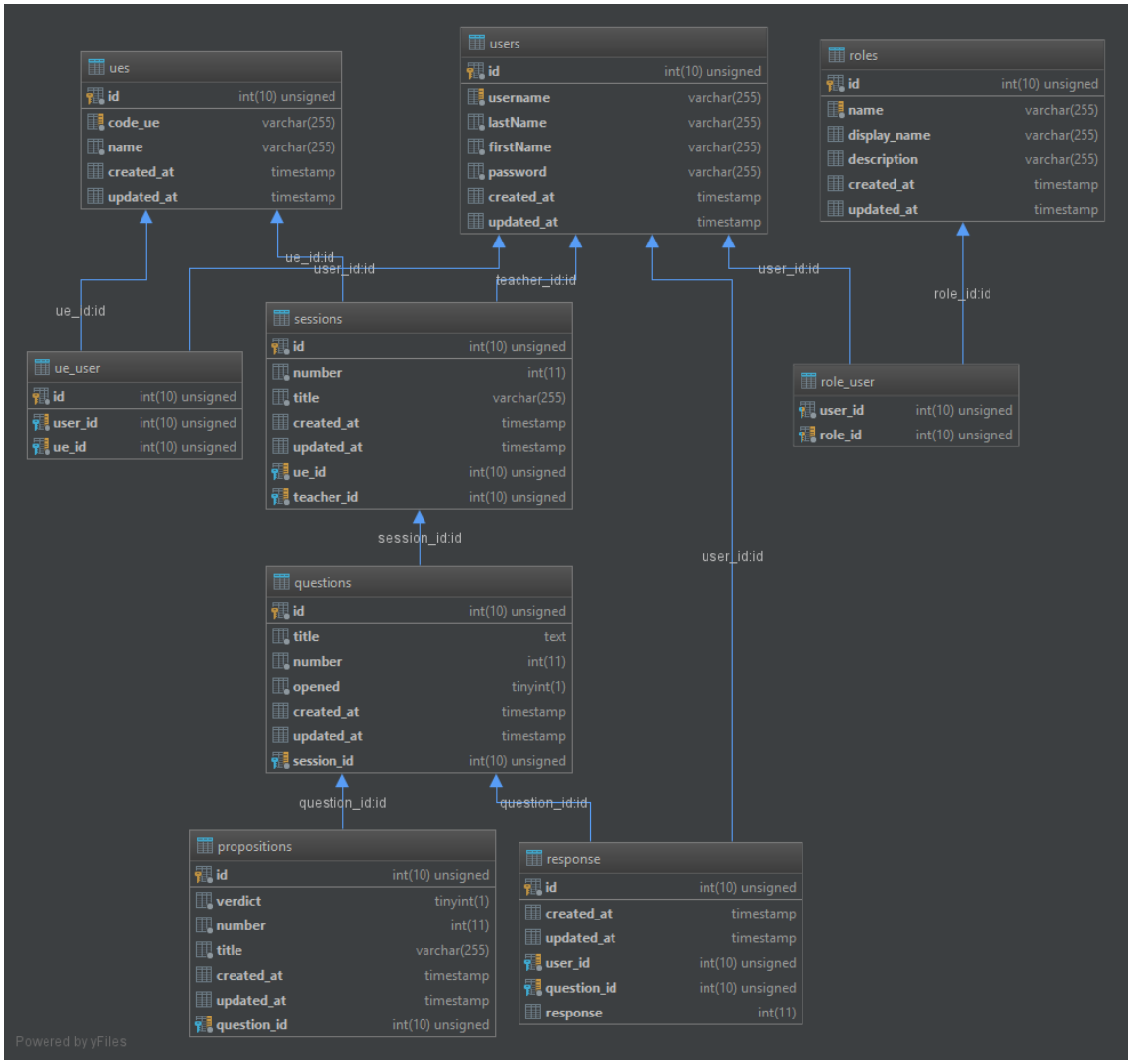

Figure 1: Diagramme de classe métier de notre application

3 Choix d'implémentation

Pour tout développement d'applications, il est nécessaire de choisir les technologies et outils adéquats pour faciliter la réalisation. Dans ce chapitre nous allons présenter ceux que nous avons choisis afin d'en justifier le choix.

3.1 Architecture

En commençons à travailler sur le projet, la question de la communication entre notre application serveur et les deux autres clients s'est rapidement posée, on s'est retrouvé face à deux choix d'implémentations possibles.

Le premier choix consiste à développer une application monolithique où les couches communiquent entre elles via des appels de méthodes. c'est-à-dire travailler à six sur une base de code commune aux trois binômes. ce qui rend le développement des trois parties fortement couplé entre elles. mais aussi ferait perdre de l'autonomie à chaque binôme, autrement dit cette méthode nous obligerait à développer notre application dans un cadre restreint où l'on dépend beaucoup plus les uns des autres. Néanmoins cette approche n'a pas que des inconvenants car l'architecture et le déploiement de l'application sont plus simple, comparer à la deuxième approche décrite ci-dessous.

Le deuxième choix consiste à développer trois applications distinctes qui communiquent entre elles à travers le protocole HTTP via une API REST, le but étant d'isoler chaque partie métier de notre application pour la transformer en un service indépendant exposant sa propre interface. Chacune des parties de notre projet est devenue une application indépendante, ce qui présente plusieurs avantages:

- Chaque application a une base de code plus petite, ce qui la rend plus lisible et facilite sa maintenance.
- Chaque application étant indépendante du reste, elle peut évoluer plus rapidement. Si le contrat d'interface est constant, les autres applications ne seront pas impactées par les changements internes. De plus, toujours grâce à cette indépendance, il est possible d'écrire chaque application dans le langage qui correspond le mieux au binôme dédié à son développement.

Nous avons fait le choix de nous orienter vers la solution à trois applications. Ce qui implique de définir un contrat pour la communication entre le serveur et les deux clients. On est rapidement confronté à la problématique du « design d'API ». Ce point constitue un enjeu majeur, dans la mesure où une API mal conçue crée plus de problèmes qu'elle n'en résout comparé à une application monolithique.

3.2 REST [2]

Pour une bonne conception de l'API serveur, nous avons suivi les principes et recommandations de l'architecture REST, ce qui implique de respecter les contraintes suivantes:

l'URI comme identifiant des ressources REST se base sur les URIs (Uniform Resource Identifier) afin d'identifier une ressource. Ainsi notre application se doit de construire ses URIs (et donc ses URL) de manière précise, en tenant compte des contraintes REST. Il est nécessaire de prendre en compte la hiérarchie des ressources et la sémantique des URLs pour les éditer
Quelques exemples de construction d'URIs:

- Liste des Ues: "`/api/ues`".
- Affichage d'une Ue: "`/api/ues/{ue_id}`".
- Toutes les séances d'une Ue: "`/api/ues/{ue_id}/sessions`".

En construisant correctement les URIs, il est possible de les trier, de les hiérarchiser et donc d'améliorer la compréhension du système.
L'URL suivante peut alors être décomposée logiquement :

- La séance ayant l'id 30: "`/api/sessions/30`".
- Toutes les seance par Ue (Ue ayant l'id 5): "`/api/ues/5/sessions`".
- L'Ue ayant l'id 5: "`/api/ues/5`".
- Toutes les Ues: "`/api/ues`".

Les verbes HTTP comme identifiant des opérations La seconde règle d'une architecture REST est d'utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource. Ainsi, généralement pour une ressource, il y a 4 opérations possibles (CRUD) et HTTP propose les verbes correspondants:

Opération	Verbe HTTP
Créer (create)	POST
Afficher (read)	GET
Mettre à jour (update)	PUT
Supprimer (delete)	DELETE

Exemple d'URLs pour une ressource donnée (une UE par exemple) :

Opération	Conforme aux règles de REST		Non conforme aux règles de REST	
Créer	`POST`	`/api/ues`	`GET`	`/api/ues/create`
Afficher	`GET`	`/api/ues/{ue_id}`	`GET`	`/api/ues/display/{ue_id}`
Mettre à jour	`PUT`	`/api/ues/{ue_id}`	`POST`	`/api/ues/edit/{ue_id}`
Supprimer	`DELETE`	`/api/ues/{ue_id}`	`GET`	`/api/ues/delete/{ue_id}`

les réponses HTTP comme représentation des ressources Il est important d'avoir à l'esprit que la réponse envoyée n'est pas une ressource, c'est la représentation d'une ressource. Ainsi, une ressource peut avoir plusieurs représentations dans des formats divers : `HTML`, `XML`, `CSV`, `JSON`, etc. Dans notre cas les réponses retournées au client sont exclusivement en `JSON`, étant donné que les deux clients qui utilisent notre API sont écrit en Javascript le format `JSON` est le plus approprié pour les échanges, principalement parce qu'il est nativement reconnu par Javascript en ne nécessite pas d'être parsé.
Un exemple de JSON que retourne l'API (le JSON ci-dessous représentante les statistiques pour la question ayant l'id 108):
`GET /api/stat/question/108`


```
{
  "question": {
    "id": 108,
    "title": "Quel est le format renvoyer par l'api du
       serveur ?",
    "number": 3,
    "opened": true,
    "session_id": 36,
    "propositions": [
      {
        "id": 322,
        "verdict": 1,
        "number": 1,
        "title": "JSON",
        "stat": {
          "responses_count": 1,
          "users": [
            {
              "id": 37,
              "firstName": "Cruz",
              "lastName": "Botsford",
              "username": "3651667"
            }
          ]
        }
      },
      {
        "id": 323,
        "verdict": 0,
        "number": 2,
        "title": "XML",
        "stat": {
          "responses_count": 2,
          "users": [
            {
              "id": 35,
              "firstName": "Lane",
```

```
                "lastName": "Zieme",
                "username": "3508917"
              },
              {
                "id": 40,
                "firstName": "Quincy",
                "lastName": "Barrows",
                "username": "3550518"
              }
            ]
          }
        },
        {
          "title": "sans opinion",
          "number": 0,
          "verdict": 0,
          "stat": {
            "responses_count": 0,
            "users": []
          }
        }
      ]
    }
}
```


Authentification: [3] Historiquement, pour remplir ce rôle, les cookies sont utilisés et retournés par le serveur au client suite à une authentification réussie. Ces mêmes cookies accompagnent chaque requête du client afin de l'identifier. Les sessions sont maintenues côté serveur (stockées en mémoire, sur le système de fichiers ou encore en base de données) avec par conséquent une problématique de charge à gérer. D'autres solutions existent telles que HTTP Basic Authentication (les identifiants de l'utilisateur doivent cependant transiter dans chaque requête).

C'est un des sujets les plus souvent abordé quand on parle de REST : comment authentifier une requête sachant qu'une API REST se doit d'être stateless (pas de session côté serveur) ? La réponse est très simple et est massivement utilisée par des APIs renommées (Google, Yahoo, etc.) : le jeton d'authentification.

Chaque requête est envoyée avec un jeton (token) passé en paramètre de la requête. Ce jeton temporaire est obtenu en envoyant une première requête d'authentification puis en le combinant avec nos requêtes.

Pour des questions de Sécurité nous avons choisi d'utiliser un système de jeton normalisé et standardisé par la RFC 7519 [4] le **JWT** (JSON Web Tokens). Le principe est simple, après s'être authentifié, le serveur génère un hash de plus de 100 caractères qui servira de signature pendant une certaine durée (2h dans notre cas, ce qui correspond à la durée d'un cour). Ce token va ensuite transiter dans chaque

requête entre le client et le serveur. Pas de cookie ni de session serveur. De plus, cette solution est applicable pour tout type de plate-forme et peut permettre à une API d'être utilisée par des applications natives, des applications web... Cette approche a l'avantage de pouvoir être utilisée avec de nombreuses technologies. Les tokens s'autosuffisent, ils contiennent toute l'information nécessaire à l'identification d'un utilisateur. Ils peuvent aussi bien être utilisés dans une URL que dans un header HTTP.

Ainsi, on peut construire le scénario suivant :

1. Demande d'accès à une ressource `GET /api/ues/5/sessions`
   Le serveur refuse l'accès à la ressource

```
{
  "error": {
    "message": "token not provided",
    "status_code": 400,
    }
}
```

2. Demande d'authentification
   `POST /api/auth?username=nobel&password=password`
   Le serveur nous donne un jeton

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
     eyJzdWIiOjQ4LCJpc3MiOiJodHRwOlwvXC9hcGkuZGV2XC9
     hcGlcL2F1dGgiLCJpYXQiOjE0OTUzNjY5ODIsImV4cCI6MTQ
     5NTM3Nzc4MiwibmJmIjoxNDk1MzY2OTgyLCJqdGkiOiI4N2
     ExNWVlNGZmZThlZWE3MThhNzQxM2I3ODE4YTg1NiJ9.F5EE2
     TnJd83ezFjRe3j5TE_MiiWBTB6ktXZ24uyy83U"
}
```

3. Demande d'accès à une ressource avec un jeton valide
   `GET api/ues/5/sessions?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.`
   `eyJzdWIiOjQ4LCJpc3MiOiJodHRwOlwvXC9hcGkuZGV2XC9hcGlcL2F1dGgiLCJpYXQiOj`
   `E0OTUzNjY5ODIsImV4cCI6MTQ5NTM3Nzc4MiwibmJmIjoxNDk1MzY2OTgyLCJqdGkiOiI4`
   `N2ExNWVlNGZmZThlZWE3MThhNz QxM2I3ODE4YTg1NiJ9.F5EE2TnJd83ezFjRe3j5TE_`
   `MiiWBTB6ktXZ24uyy83U`

4. Le serveur nous donne accès a la ressource demandée

```
{
  "ue": {
    "id": 5,
    "code_ue": "4I506",
    "name": "Composants (CPS)",
```

```
    "sessions": [
      {
        "id": 31,
        "number": 1,
        "title": "Introduction et design patterns des
            composants.",
        "ue_id": 5,
        "teacher_id": 43
      },
      {
        "id": 32,
        "number": 2,
        "title": "Langage de specification.",
        "ue_id": 5,
        "teacher_id": 43
      }
    ]
  }
}
```

3.3 Technologies utilisées

PHP C'est un langage de programmation conçu pour permettre la création d'applications dynamiques, le plus souvent développées pour le Web. PHP est le plus souvent couplé à un serveur Apache bien qu'il puisse être installé sur la plupart des serveurs HTTP. Ce couplage permet de récupérer des informations issues d'une base de données, d'un système de fichiers (contenu de fichiers et de l'arborescence) ou plus simplement des données envoyées par le navigateur afin d'être interprétées ou stockées pour une utilisation ultérieure.

Laravel C'est est un framework web open-source écrit en PHP respectant le principe MVC et entièrement développé en POO. Il est devenu à la suite de sa version 4 l'un des frameworks PHP les plus utilisés et les plus reconnus au monde. Créé par Taylor Otwel, Laravel initie une nouvelle façon de concevoir un framework en utilisant ce qui existe de mieux pour chaque fonctionnalité ainsi que de nouveaux composants exclusifs. Parmi eux:

- un système de routage perfectionné.
- un créateur de requêtes SQL et un ORM performants.
- un système d'authentification pour les connexions.
- un système de migration pour les bases de données.

3.4 Outils utilisés

PhpStorm [5] C'est un environnement de développement conçu pour PHP, développé par la société JetBrains. Écrit en Java, il permet d'éditer du code en PHP. Il est compatible aussi bien sur Windows que sur Mac et sur Linux, et il est essentiellement dédié aux développeur travaillant en PHP. La plupart des licences sont payantes, mais l'université nous permet d'avoir une licence étudiant gratuitement.

Postman C'est un outils permettant de construire et de tester rapidement des requêtes HTTP. cet outil est un client REST proposé par Google. Il est disponible sous la forme d'une extension Chrome ou bien d'une application stand-alone. L'outil permet de créer des règles sous la forme d'url vers le service REST avec lesquelles ont définit des paramètres d'entête, de chaîne de requête, d'authentification. On choisit également un verbe HTTP comme GET.

3.5 Difficultés techniques

L'une des principales difficultés rencontrées durant la réalisation du projet était la manière de faire le lien entre notre application et la base de données de l'université. C'est-à-dire pouvoir au moins authentifier les utilisateurs de l'application en utilisant la base de l'université et éventuellement récupérer des informations qui pourraient nous être utiles comme la liste des UEs auxquelles un étudiant est inscrit, l'enseignant responsable d'une UE ou encore la liste des étudiants d'une UE.

Nous avons abandonné l'idée d'utiliser la base de données de l'université sauf pour le cas de l'authentification à cause de tous ce que ça nous impose comme contrainte. Par exemple le fait que l'on ait un accès restreint (à cause des données sensibles qu'elle contient (mot de passe ...)). Mais aussi la dépendance qu'on aurait eue avec l'administrateur de la base (le fait d'attendre l'accès à des données nous aurait sans doute retardés dans notre travail). Nous avons donc fais le choix de tous gérer en interne à l'exception de l'authentification.

D'autre part, comme c'est notre première expérience avec l'architecture REST, le système d'authentification par token, ainsi que le système d'annuaire LDAP (Lightweight Directory Access Protocol) utilisé par l'université pour authentifier les utilisateurs. Il y a donc eu une période de découverte, d'apprentissage et de tâtonnement afin de maîtriser les concepts et les technologies avant d'avoir un premier prototype, que l'on a ensuite amélioré pour avoir un résultat satisfaisant.

4 Structure de l'application

4.1 Organisation générale

Notre implémentation s'est basée sur le pattern MVC ,qui signifie Modèle-Vue-Contrôleur. Le modèle MVC décrit une manière d'architecturer une application informatique en la décomposant en trois sous-parties:

- **La partie Modèle :** encapsule la logique métier (« business logic ») ainsi que l'accès aux données. Il peut s'agir d'un ensemble de fonctions (Modèle procédural) ou de classes (Modèle orienté objet).
- **La partie Vue:** s'occupe des interactions avec l'utilisateur : présentation, saisie et validation des données.
- **La partie Contrôleur:** gère la dynamique de l'application. Elle fait le lien entre l'utilisateur et le reste de l'application.

voici un schéma décrivant les différentes interactions entre les trois composantes :

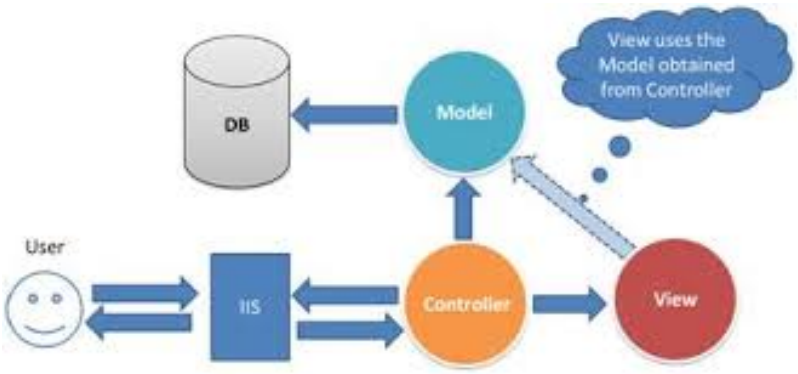

Figure 2: modèle MVC

- La demande de l'utilisateur (exemple: requête HTTP) est reçue et interprétée par le Contrôleur.
- Celui-ci utilise les services du Modèle à fin de préparer les données à afficher.
- Ensuite, le Contrôleur fournit ces données à la Vue, qui les présente à l'utilisateur (par exemple sous la forme d'une page HTML).

Les modèles Laravel propose un ORM (acronyme de object-relational mapping) c'est-à-dire que les éléments de la base de données ont une représentation sous forme d'objets manipulables. L'intérêt d'un tel outil est que il simplifie en grandement l'accès à la base de données. Avec l'ORM une table est représentée par une classe qui étend la classe Model. Pour notre table `ues` on a une classe model nommée `Ue.php`. La classe `Ue` dispose de plusieurs méthodes utiles:

- `students()`: permet de récupérer tous les étudiants qui sont inscrit à cette Ue.

- `teacher()`: récupère l'unique enseignant ayant crée l'Ue.
- `sessions()`: récupère toutes les séances lui appartenant.
- `questions()`: permet de récupérer toutes les questions de chaque séance relative à l'ue concernée.

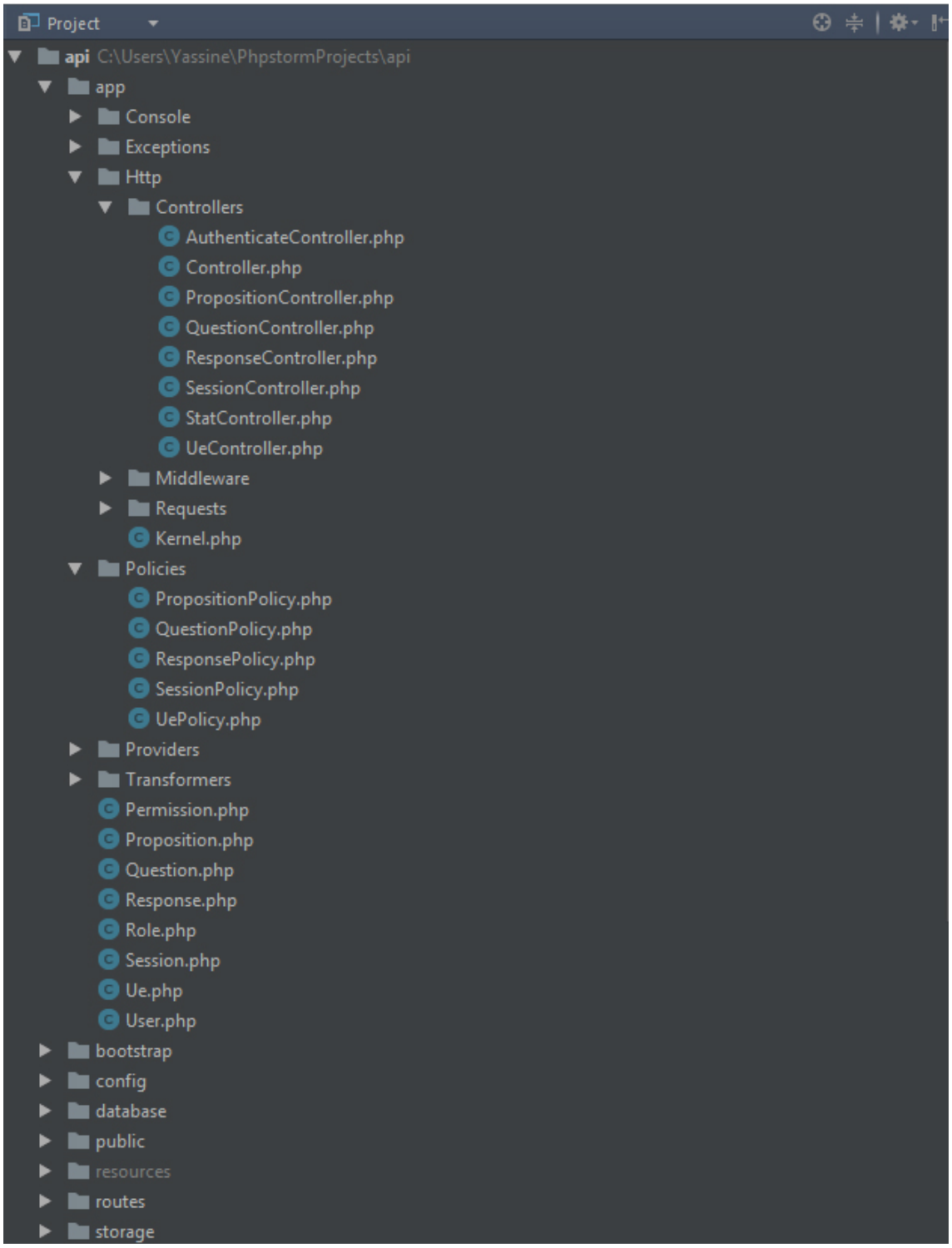

Figure 3: Organisation des classes

Les contrôleurs La tâche d'un contrôleur est de réceptionner une requête qui a déjà été sélectionnée par une route, et de définir la réponse appropriée en exécutant la méthode associée à une route. Voyons maintenant un exemple pratique de mise en œuvre d'un contrôleur. On va conserver notre exemple avec les Ues mais maintenant traité avec un contrôleur `UeController.php`. Ce contrôleur dispose de plusieurs méthodes permettant de traiter les requêtes d'affichage, de création, de mise à jour, de suppression et d'inscription aux Ues.

Les autorisations Laravel nous offre un système complet d'autorisations. Le plus simple pour voir comment cela fonctionne est de prendre un exemple. Dans notre application on a le dossier `app/Policies` dans lequel on trouve le fichier `UePolicy.php` résponsable de gérer les autorisations relatives aux Ues, voici la méthode `create` qui permet de déterminer si l'utilisateur courant (authentifié) à le droit de créer une Ue, dans autre cas on autorise uniquement utilisateurs ayant le rôle d'enseignant.

```
1     public function create(User $user){
2         return $user->hasRole('teacher');
3     }
```

Middleware Les middlewares sont chargés de filtrer les requêtes HTTP qui arrivent dans l'application, ainsi que celles qui en partent (beaucoup moins utilisé). Le cas le plus classique est celui qui concerne la vérification de l'authentification d'un utilisateur(dans notre cas étudiant ou biens enseignant) (pour qu'il puisse accéder à certaines ressources. On peut aussi utiliser un middleware par exemple pour démarrer la gestion des sessions. Donc dès qu'il y a un traitement à faire à l'arrivée des requêtes (ou à leur départ) un middleware est tout indiqué. Voici un schéma pour illustrer cela :

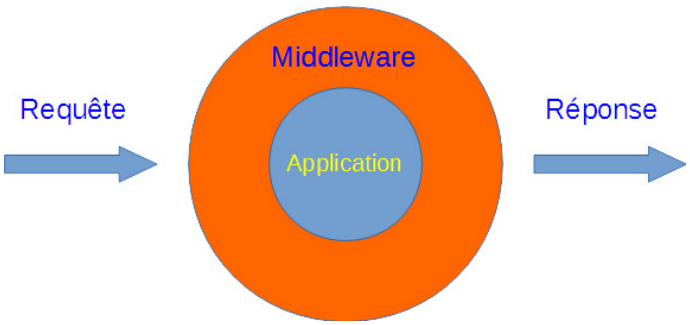

Figure 4: fonctionnement du middleware

routes toutes nos routes sont définies dans le fichier `routes/api.php` ,son rôle principal est de filtrer puis rediriger les requêtes bien formées vers le bon contrôler. Prenant l'exemple d'une requête pour créer une nouvelle session pour l'Ue ayant l'id "5". La route suivante permet de rediriger la requête `POST /ues/5/sessions'` sur la méthode `store()` du contrôleur `SessionController`.

```
1 $api->post(
2     'ues/{ue_id}/sessions',
3     'App\Http\Controllers\SessionController@store');
```

4.2 Le cycle de vie d'une requête [6]

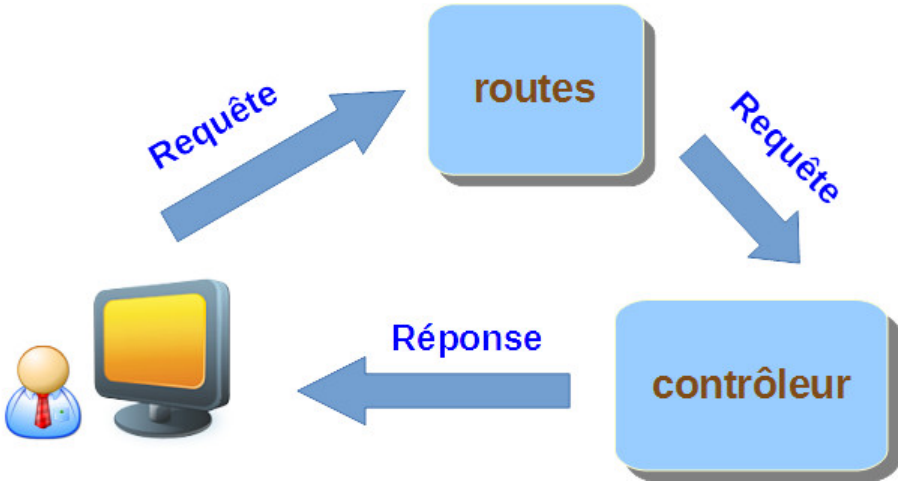

Figure 5: cycle d'une requête

Lorsque l'application reçoit une requête, si elle est bien formée (elle match avec une route) et si elle passe l'étape d'autorisation alors la méthode du contrôleur associé à la route est exécutée. Pour mieux comprendre le cycle de vie d'une requête, prenons l'exemple d'un utilisateur voulant créer une Ue "APS" ayant le code "4I503", et regardant toutes les étapes de son traitement:

1. la requête `POST /api/ues` match avec la route qui redirige vers la méthode `store()` de `UeController`.

2. avant d'être rediriger, la requête est filtrée par middleware `GetUserFromToken` qui s'assure que le token fournit par l'utilisateur est valide et l'utilise pour l'authentifier. Si ce n'est pas le cas on renvoie une erreur contenant le motif de l'échec de la requête.

3. La requête passe également par la couche responsable des autorisations, dans notre cas il s'agit de `UePolicy` qui vérifie que l'utilisateur authentifié à l'étape précédente possède les droits suffisants pour effectuer l'opération de création d'Ue, autrement dit l'utilisateur doit être enseignant.

4. la requête passe ensuite à l'étape de validation, dans notre cas cette étape permet de vérifier la présence des deux paramètres obligatoires (le nom et le code de l'Ue) et que le code de l'ue respecte l'expression régulière `[1-5](I|i)[0-9]{3}`.

5. Si la requête passe par les filtres des étapes précédentes, alors la méthode `store()` de `UeController` est exécutée. Cette méthode s'occupe de créer une nouvelle entrée dans la table `Ue` en utilisant les paramètres de la requête (`nom ="APS", code_ue = "4I503"`).

6. le contrôler `UeController` renvoie la réponse à l'utilisateur concerné sous la forme d'un `JSON` comme suit:

```
{
  "ue": {
    "code_ue": "4i503",
    "name": "aps",
    "id": 16
  }
}
```

5 Conclusion

Au cours de la réalisation de ce projet nous avons appris beaucoup de choses. C'est la première fois que nous nous sommes mise dans la situation d'une équipe qui doit réaliser une application destinée à être utilisé à l'échelle d'une université. Ce qui contraste avec ce que nous avions l'habitude de faire dans d'autres projets, C'est-à-dire des projets plus petits qui restent souvent au stade de prototype et qui sont réalisés uniquement dans un but pédagogique. Le fait de travailler sur un projet de plus grande envergure, avec une équipe de six personnes et un délai d'un semestre pour le réaliser nous a motivés pour faire un travail que nous espérant être de qualité.

Au début du projet nous nous sommes accordé un délai de quelques semaines pour analyser les problèmes qui nous étaient posés et se mettre d'accord avec les deux autres groupes sur les points les plus importants, définir une feuille de route et fixer les responsabilités de chaque groupe pour éviter d'éventuel conflit futures. Les principaux points que nous avons discutés avant de passer à l'implémentation sont: l'architecture globale du projet, définir les contrats d'interface entre les deux clients et le serveur et l'API du serveur et établir un premier plan de la base de données. Après cette phase d'analyse nous nous sommes mis d'accord sur la plupart des points citer plus haut, ce qui a permis à chaque binôme de travailler d'une manière relativement indépendante et autonome.

Durant la phase de conception la communication a occupé une partie de notre temps. Au final ça s'est avéré payant puisque nous avions anticipé la plupart des problèmes qui ce sont présenter à nous par la suite. Nous avons appris à ne pas nous précipiter sur notre éditeur préférer pour écrire du code et nous avons également appris l'importance d'une bonne communication et l'utilité d'une bonne conception. Cela-dit une bonne conception ne fait pas tout. Il nous est arrivée de prendre à la légère certains problèmes à la légère et de sous-estimer le temps qu'il aurait fallu pour les résoudre. ce qui nous a valu quelque nuit blanche. Malgré quelques aléas de temps en temps, on peut dire que globalement la conception et la réalisation du projet c'est bien passé et que nous avions énormément appris

References

[1] UML 2 par la pratique,Pascal Roques, Editions Eyrolles, 7 juil. 2011 396 pages.

[2] thèse Roy Fielding,Architectural Styles and the Design of Network-based Software Architectures ,2000.
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.html

[3] https://www.ekino.com/introduction-aux-json-web-tokens/
https://tools.ietf.org/html/rfc7519

[4] https://tools.ietf.org/html/rfc7519

[5] https://www.video2brain.com/fr/phpstorm

[6] https://laravel.com/docs/5.4