

École des Ponts

ParisTech

École des Ponts ParisTech
Novembre 2017 - Janvier 2018

Cours de Techniques de Développement Logiciel
Projet Click2Evaluate : Documentation

Marc-Antoine Augé, Hervé Andrès, Bastien Dechamps et Michaël Karpe,
En collaboration avec Barbara Gérard, assistante au S2iP
Encadrés par Xavier Clerc, professeur du module

Table des matières

1	Présentation du projet	3
1.1	Architecture du projet	3
2	Manuel d'utilisation	3
2.1	Déroulé d'une campagne d'évaluation	3
2.2	Application	4
2.3	Le panneau de contrôle	4
2.4	La gestion des questionnaires	4
2.5	Mise à jour des questionnaires pendant la campagne	4
3	Application	4
3.1	Préambule à Ionic	4
3.2	Architecture de l'application	5
3.3	Providers	5
3.4	Pages	5
3.4.1	Le login	5
3.4.2	Le menu	6
3.4.3	Le questionnaire	8
3.5	Les notifications	9
3.5.1	Plugin Local Notifications	9
3.5.2	Notifications pour Click2Evaluate	9
3.5.3	Fonctionnement du système de notifications	9
3.5.4	Problèmes et limites	9
3.6	Comportements	10
3.6.1	Questions obligatoires	10
3.6.2	Pas d'accès internet	10
3.6.3	Rester connecté	10
4	Serveur	11
4.1	Préambule à Python	11
4.2	Modèles	11
4.3	Remplir la base de données	12
4.3.1	Format des données d'entrées	12
4.3.2	Importation	13
4.4	API	14
4.4.1	Obtenir la liste des cours d'un élève donné	14
4.4.2	Obtenir un questionnaire donné	15
4.4.3	Envoyer des réponses	15
4.5	Panneau de contrôle	16
4.5.1	Principe général du code	16
4.5.2	Visualisation des taux de réponses par cours ou par département	16
4.5.3	Visualisation des réponses aux questions	16
4.5.4	Exportation des réponses	17
4.5.5	Importation des données	17
5	Mécanisme d'authentification	17
5.1	Schéma général d'authentification	18
5.2	Mise en place d'utilisateurs	18
5.3	Mise en place de l'authentification par token pour l'API	19
5.4	Gestion de l'authentification au niveau de l'app	19
5.5	Gestion des permissions	19

19section*.55	
5.6 Utilisation d'un annuaire ldap	20
5.7 Note sur la différence entre SSO et ldap	20
6 Déploiement	20
6.1 Serveur	20
6.2 Webapp	21
6.3 Applications	21
6.4 Base de données	21
7 Pour aller plus loin	21
8 Remerciements	22

1 Présentation du projet

Ce projet a été réalisé dans le cadre du cours de TDLOG pendant l'année scolaire 2017/2018 par Marc-Antoine AUGÉ, Hervé ANDRÈS, Bastien DESCHAMPS et Michaël KARPE en collaboration avec Mme GÉRARD du S2iP, en charge de l'évaluation des cours.

Il s'agissait de faire une application mobile pour que les élèves puissent évaluer directement leurs cours sur application, pour encourager davantage de réponses.

Les codes sources sont accessibles sur GitHub aux liens suivants :

- L'application : <https://github.com/HerrVey/Click2Evaluate>
- Le serveur : https://github.com/Marc-AntoineA/Click2Evaluate_server

1.1 Architecture du projet

Le projet repose sur une architecture client-serveur :

- Le **client** est ici l'application mobile, programmée avec Ionic [1] : cela nous a permis de développer une seule application pour iOS et Android tout en ayant accès aux notifications, fonctionnalité native.
- Le **serveur** permet à l'application de toujours disposer de données à jour, sans avoir besoin de la mettre à jour régulièrement. Le serveur permet également de réceptionner les réponses aux questionnaires des élèves. Par ailleurs, nous avons développé un panneau de contrôle accessible au S2iP pour lui permettre de mettre à jour facilement les cours disponibles, les questionnaires, ... mais également pour exporter les réponses des élèves.

2 Manuel d'utilisation

Il ne s'agit pas de réaliser un manuel complet mais de préciser les grandes lignes pour utiliser pleinement l'outil développé dans le cadre de ce projet.

2.1 Déroulé d'une campagne d'évaluation

Une campagne d'évaluation des cours a été conçue comme suit :

1. En amont, création du fichier élève (exportation de Gede) et mise à jour manuelle du fichier `.csv` des cours selon les formats appropriés (*cf.* Section ??). Le fichier de cours doit contenir uniquement les cours à évaluer : l'ensemble des cours indiqués dans le fichier élève n'a donc pas besoin d'être écrit ici.

2. Importation des fichiers en ligne (construction automatique de la base de données)
3. Mises à jour éventuelles / création de nouveaux questionnaires ou questions
4. Lancement automatique de la campagne de questionnaires dès la date de disponibilité des premiers questionnaires
5. Suivi en temps réel des réponses et des taux de réponses sur le panneau de contrôle
6. Exportation en fin de campagne de l'ensemble des fichiers de réponses dans un seul fichier .zip

Remarque : Un fichier dit `.csv` est un fichier excel particulier. À partir de excel, faire enregistrer sous / fichier `.csv` avec point-virgule (;) comme séparateur. Pour manipuler des fichiers `.csv`, le tableur de Libre Office est plus confortable. Pour lire un fichier `.csv` sur Excel, une petite manipulation est à effectuer¹

2.2 Application

Pour pouvoir répondre aux questionnaires, il faut être connecté à internet au moment du chargement des données et au moment de l'envoi des réponses. Dans le cas où l'élève n'a pas internet au moment de l'envoi, ces réponses sont sauvegardées en local et il doit lui même faire la démarche de les renvoyer dès que possible.

2.3 Le panneau de contrôle

L'accès au panneau de contrôle est sécurisé par mot de passe.

De nombreuses fonctionnalités ont déjà été développées et sont accessibles à l'URL `/s2ip/accueil` : il s'agit principalement de fonctionnalités de lecture.

Pour avoir accès à la base de données et modifier les cours ou des questions en temps réel sans renouveler entièrement la base de données et risquer de perdre des données, il faut se rendre sur l'URL `/admin/`.

2.4 La gestion des questionnaires

Il s'agit du principal point noir de l'application à ce jour. Pour rajouter des questions, il faut réaliser le questionnaire à la main sur une feuille de papier au préalable en numérotant les questions. Pour chaque question, il faut indiquer : un *titre* (= `title`), un *intitulé* (= `label`), un *type* parmi :

- `select` pour une questions à choix multiples. Il faut alors indiquer dans le champs `typeData` les différentes réponses possibles sous la forme suivante : `Choix 0;Choix 1; Choix 2` (les choix sont séparés par des « ; »).
- `selectOne` pour une question à choix unique. La syntaxe de `typeData` est la même que pour `select`.
- `text` ou `inline` pour une réponse texte (`inline` pour des réponses (courtes) sur une ligne).
- `number` pour une réponse en nombre. Il faut alors indiquer dans `typeData` les bornes inférieures et supérieures sous la forme `debut;fin`²

une *position* ainsi que pour les questions conditionnelles, dire s'il s'agit de *sous-questions* (= cocher `isSub`) puis indiquer quelle est la question *mère* (= le numéro de la question mère dans *questions-ParentPosition*) et enfin pour quelle valeur de la question mère on affiche la question fille. Cela ne fonctionne que pour les questions à choix multiples ou unique, on attend dans ce champ la position du choix, le premier choix étant à 0.

1. <http://www.cours-excel.fr/comment-mettre-en-forme-un-fichier-csv/>

2. Les bornes ne sont pas encore implémentées dans l'application

Une fois le questionnaire réalisé sur le papier, il faut rajouter manuellement chaque question sur le site `/admin/` : il y a la possibilité de modifier les questions ultérieurement. Une fois les questions créées, il y a possibilité de visualiser la structure du questionnaire dans l'onglet *Gestion des questionnaires* du panneau de contrôle.

Nous conseillons de tester les questionnaires sur mobile en créant un élève fictif (par exemple `barbara.gerard@enpc.fr` qui suit tous les cours).

2.5 Mise à jour des questionnaires pendant la campagne

Les questionnaires peuvent techniquement être modifiés pendant une campagne et alors les élèves qui répondraient après ces modifications auront le questionnaire modifié. Toutefois ce comportement conduit certainement à des bugs si jamais certaines personnes ont répondu à des questions différentes. Il est donc conseillé de ne plus les modifier dès le début de la campagne.

3 Application

3.1 Préambule à Ionic

Quelques commands et références :

- `ionic serve` pour lancer l'outil.
- `ionic cordova build browser` pour construire la version web de l'application, utilisable directement sur navigateur.

Ionic utilise de la programmation asynchrone avec des Promesses et des observables qu'il est important de bien comprendre. Nous nous sommes référés à <https://basarat.gitbooks.io/typescript/content/docs/promise.html>

3.2 Architecture de l'application

L'application repose sur l'architecture classique utilisée dans Ionic et se base sur une architecture modèle-vue-contrôleur. Précisons un peu plus :

- Dans le dossier `app`, nous avons juste indiqué les pages ainsi que les différents modules utilisés.
- Dans le dossier `assets/data` nous avons stocké l'ensemble des données qui permettent de tester le programme en local lorsque l'on précise dans `providers/api/api.ts` `noServer = True`. Ces données ne sont pas au même format que les requêtes serveur, en effet, nous ne pensions pas avoir le temps de mettre en place le serveur initialement.
- Dans le dossier `directives`, nous avons rajouté une directive `autosize` qui permet de créer des zones de texte dont la taille augmente dynamiquement lorsque l'utilisateur tape du texte.
- Dans le dossier `pages` on trouvera un dossier pour chaque page du programme et dans chaque sous-dossier trois fichiers :
 - Le fichier `.html` qui est un template angular qui définit les composants à afficher
 - Le fichier `.scss` pour la mise en forme du fichier `.html`
 - Le fichier TypeScript `.ts` dans lequel on définit quelques fonctions d'affichage. Aucune fonction de traitement des données n'est précisée ici.
- Les modèles et contrôleurs sont définis dans le dossier `providers` où l'on définit l'ensemble des classes de données utilisées ainsi toutes les fonctions de traitement : requêtes, stockage.
- Les autres dossiers n'ont pas été utilisés.

3.3 Providers

Les données dans quatre classes différentes :

- Des **cours** (`course-data.ts`) qui contient toutes les informations relatives à un cours spécifique (la date de la commission, le numéro du groupe, le type de questionnaire...). Cela correspondra à un **Survey** côté serveur.
- Des **questions** (`question.ts`) qui contient les informations relatives à une question ainsi que la réponse associée. Un traitement particulier pour les questions de type `select` est à prévoir puis-ce que la réponse est ici un tableau de booléen indiquant si un élève a répondu ou non au questionnaire.
- Des **élèves** (`students-data.ts`) qui constituent le cœur de l'application. On définit ici les fonctions pour connecter un élève.
- Des **Survey** (`survey-data.ts`) qui constituent les instances des questionnaires. Il y a un survey par cours. Ces derniers sont constitués d'un ensemble de questions et sont téléchargés depuis le serveur à chaque fois que l'utilisateur essaie de répondre à un questionnaire. On a également défini dans ce fichier la fonction d'upload du questionnaire, question par question.

3.4 Pages

3.4.1 Le login

La page de login est la page sur laquelle l'élève arrive lorsqu'il lance l'application. Cette page contient deux champs permettant d'identifier l'élève :

- l'**adresse LDAP** qui a la forme *prenom.nom@enpc.fr*
- le **mot de passe associé**

La question de l'authentification est gérée dans la section 5.

Nous avons initialement mis une case à cocher permettant de rester connecté mais sommes revenus sur cette fonctionnalité (voir Section 3.6.3). L'identification de l'élève est importante dans la mesure où elle permet de récupérer uniquement les modules dans lesquels il est inscrit.

3.4.2 Le menu

La page de menu est la page affichant la liste des cours suivis par l'élève connecté. Les cours sont répartis dans trois sections. La première contient les cours pour lesquels le questionnaire a été ouvert mais n'a pas encore été rempli par l'élève. La seconde contient tous les cours déjà évalués du semestre. Les cours de cette section sont par ailleurs marqués par une coche. La dernière contient les cours pour lesquels le questionnaire n'est pas encore ouvert ou alors est déjà fermé (mais non répondu). Notons également que les cours sont affichés dans chaque section par ordre alphabétique.

En haut de la page de menu se trouve quatre icônes cliquables. La première permet de rafraîchir la page. La seconde permet d'afficher une petite bulle d'aide. La troisième permet de se déconnecter de son compte élève et donc de revenir à la page de login. Enfin, la quatrième affiche une petite bulle contenant les noms des développeurs de l'application.

À partir de cet écran de menu, il est possible de cliquer sur un cours de la liste. Cela ouvre un *modal*, c'est-à-dire une page qui s'ouvre par-dessus la page courante (ie la page de menu), qui affiche les informations importantes sur le cours :

- si le questionnaire a déjà été complété ou non
- le numéro de groupe
- l'adresse LDAP du délégué du groupe
- la date d'ouverture du questionnaire
- la date de fermeture du questionnaire

Si le questionnaire est ouvert, le *modal* contient également un bouton "Evaluer" qui redirige vers le questionnaire du cours.

Click2Evaluate

Bienvenue !

Bienvenue sur Click2Evaluate, l'application
d'évaluation des cours de l'École des Ponts
ParisTech !
Veuillez vous identifier.

Ldap

prenom.nom@enpc.fr





MdP

Mot de passe

☐ Rester connecté

SE CONNECTER

FIGURE 1 – Page de login

Click2Evaluate    

Bienvenue herve.andres@enpc.fr

Voici la liste des modules à évaluer.
Cliquez sur un module pour avoir plus
d'informations sur le questionnaire associé.
Faites glisser le module que vous souhaitez
évaluer.

Cours disponibles à évaluer

Allemand - Debattieren

Analyse de Fourier

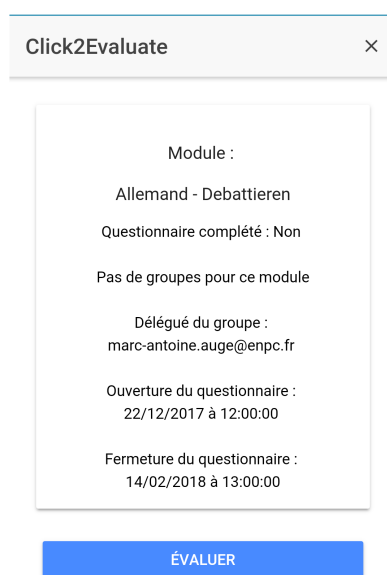
Analyse des équations aux dérivées partielles

Anglais - News and Views

Enjeux et méthodes de l'interculturel

Processus stochastics et applications

FIGURE 2 – Page de menu

FIGURE 3 – *modal*

Il est également possible d'accéder au questionnaire d'un cours directement à partir de la page de menu (donc sans passer par le *modal*) en faisant glisser un cours de la droite vers la gauche.

3.4.3 Le questionnaire

Un questionnaire prend la forme d'un ensemble de *slides*, chaque *slide* correspondant à une question. On peut naviguer d'une slide à l'autre en faisant glisser son doigt de la droite vers la gauche ou de la gauche vers la droite selon que l'on veuille aller à la question suivante ou précédente.

À chaque cours, est associé un type de questionnaire (voir 4.4.1) et pour chaque type de questionnaire, on dispose d'un fichier `.json` contenant la liste des questions et leur type (voir 4.4.2). Ce fichier est lu dans `surveyData` et le questionnaire est alors généré à partir des données stockées dans `surveyData` de la manière suivante : on parcourt l'ensemble des questions et on veille à ne pas afficher les sous-questions qui ne doivent être posées que si certaines réponses ont été données (par exemple on ne pose la question "Indiquer les changements mineurs souhaités" uniquement si l'élève a répondu "Oui avec des changements mineurs" à la question "Pensez-vous que ce module doit être maintenu?"). Ce tri des questions à afficher est effectué par la fonction `displayable`.

Chaque question attend un type de réponse particulier ce qui permet de répartir les questions en 5 catégories donnant lieu à un affichage spécifique. Les 5 catégories sont les suivantes

- **inline** : il s'agit d'une question qui attend comme réponse un petit texte d'une ligne.
- **text** : il s'agit d'une question qui attend comme réponse un texte d'une certaine taille. C'est pour ce type de question que nous avons ajouté la directive `autosize` qui permet de redimensionner dynamiquement la zone de texte en fonction de la quantité de texte tapée.
- **selectOne** : il s'agit d'une question pour laquelle il existe un nombre donné d'alternative mais l'élève ne peut faire qu'un seul choix.
- **select** : même type de question que le précédent sauf que l'élève peut faire plusieurs choix

← Questionnaire

Question obligatoire

Maintien du cours

Pensez-vous que ce cours doit être maintenu ?

Oui, sans changements ☐

Oui, avec des changements mineurs ☐

Oui, avec des changements majeurs ☐

Non ☐

FIGURE 4 – Exemple de question d’un questionnaire

— **number** : il s’agit d’une question pour laquelle la réponse attendue est un nombre.

La dernière **slide** du questionnaire contient toujours un bouton permettant d’envoyer les réponses au serveur.

3.5 Les notifications

3.5.1 Plugin Local Notifications

Il est nécessaire d’installer le plugin Ionic *Local Notifications* afin que l’application puisse programmer des notifications. Cela se fait en entrant les lignes de commandes suivantes :

- `ionic cordova plugin add cordova-plugin-local-notification`
- `npm install -save @ionic-native/local-notifications`

Ensuite, si ce n’est pas déjà fait, il faut ajouter ce plugin dans les providers du fichier `app.module.ts`.

Le site <https://github.com/katzer/cordova-plugin-local-notifications> offre plus de détails sur le fonctionnement de ce plugin, ainsi que des exemples d’utilisations.

Avec Local Notifications, une application configure localement les détails des notifications (contenu, date d’apparition, ...) et les communique au système du téléphone, qui ensuite les gère sans que l’application n’ait besoin de tourner en fond. Il ne faut pas les confondre avec les *Push Notifications* qui requièrent l’interaction avec des serveurs pour envoyer des notifications à l’appareil de l’utilisateur.

3.5.2 Notifications pour Click2Evaluate

Click2Evaluate ne compte qu’un seul type de notification : des notifications de rappel programmées quand des évaluations de cours sont disponibles. Elles se déclenchent tous les 3 jours à 18h00. Toutefois, dans la mesure où il ne s’agit pas de Push Notifications, l’utilisateur doit se connecter une première fois à l’application et rester connecté afin de voir les notifications apparaître (puisqu’elles sont gérées

localement). Ainsi, si l'utilisateur se déconnecte (et donc efface les cours et les réponses enregistrés dans le Local Storage), les notifications n'apparaîtront pas.

3.5.3 Fonctionnement du système de notifications

La planification des notifications se fait par la méthode `scheduleRemindNotification()` présente dans le fichier `students-data.ts`. Elle suit la logique suivante :

- S'il y a des évaluations disponibles (i.e. des cours dont le questionnaire est disponible et n'est pas encore envoyé) :
 - Si les notifications de rappel sont déjà programmées, ne rien faire.
 - Sinon, les programmer tous les 3 jours à 18h00, à l'aide de l'option `every` des Local Notifications
- Sinon, on annule les notifications (s'il y en a de programmées)

Cette fonction est appelée dans la fonction `getCourses()` de `students-data.ts` à chaque fois que la fonction `getCoursesOnline()` est appelée. Ainsi, une fois que les cours de l'élève sont chargés, les notifications sont programmées.

Une autre méthode `rescheduleRemindNotif()` de la classe `SurveyPage` permet d'annuler les notifications quand l'élève évalue le dernier cours disponible. Elle est donc appelée dans la méthode `send_survey()`.

Enfin, quand l'élève clique sur une notification de rappel, celle-ci l'amène sur la page menu.

3.5.4 Problèmes et limites

Plusieurs problèmes minimes perdurent :

- Un premier problème est directement lié au plugin Local Notifications : dans la mesure où il n'existe pas de propriété `"endAt"` pour spécifier une date d'arrêt des notifications, celles-ci continueront de se déclencher tous les 3 jours après la date butoire si l'utilisateur ne se déconnecte pas et ne retourne pas sur l'application alors qu'il restait des cours à évaluer avant la date butoire. Il suffit alors d'ouvrir l'application pour annuler automatiquement ces notifications.
- De même, si l'utilisateur force l'arrêt de l'application (dans Paramètres > Applications), alors les notifications seront annulées.

3.6 Comportements

3.6.1 Questions obligatoires

Certaines questions d'un questionnaire peuvent être paramétrées comme obligatoire. Les questions marquées comme obligatoires ne peuvent pas être ignorées lorsqu'un élève évalue un cours. En effet, si l'élève ne répond pas à une question obligatoire et cherche à passer à la suivante, un message apparaît pour lui indiquer que la question est obligatoire et, par ailleurs, il ne pourra plus changer de question tant qu'il n'aura pas répondu.

Pour cela, un événement `ionSlideNextStart` est déclenché à chaque fois que l'élève cherche à passer à la question suivante. Celui-ci appelle la méthode `check_possible` qui vérifie si la question sur laquelle il se trouve au moment du changement est obligatoire et auquel cas on vérifie s'il y a répondu. La difficulté est qu'il n'est pas possible de savoir quelle est la question sur laquelle l'élève se trouve lorsque que l'événement est déclenché. En revanche, nous avons accès à sa position dans le questionnaire via la méthode `getActiveIndex` de `Slides`. Nous avons donc créé, dans le contrôleur `survey.ts`, un tableau `base_questions` dont l'élément stocké à l'indice i est la i -ème question. Le problème est que certaines questions sont conditionnelles, c'est-à-dire que certaines réponses à certaines questions vont appeler une autre question généralement obligatoire (par exemple lorsque l'on répond "Oui avec des changements mineurs" à la question "Ce module doit-il être maintenu"). Ainsi la position d'une question

dans un questionnaire n'est pas fixe. Nous avons donc créé un dictionnaire `additional_questions` dont les clés sont les identifiants des questions de bases et la valeur associée à une clé est le tableau des questions additionnelles rattachées à la question dont l'identifiant est la clé. Ce dictionnaire permet d'insérer et de supprimer dynamiquement les questions additionnelles dans le tableau `base_questions` (avec `insert_question` et `delete_question`) de sorte à avoir à chaque instant un tableau faisant la correspondance entre position d'une question et la question elle-même, la mise à jour étant contrôlée via `displayable`. On réussit ainsi à vérifier de manière dynamique le caractère obligatoire d'une question lors d'un changement de question.

3.6.2 Pas d'accès internet

Si un individu essaie de répondre à un questionnaire ou de se connecter alors qu'il n'a pas accès à internet ou si le serveur est inaccessible, il recevra un message d'erreur et ne pourra pas exécuter l'action.

Les réponses sont toutefois sauvegardées.

3.6.3 Rester connecté

Note : Ce comportement est celui attendu après la mise en place de l'authentification ldap à venir.

Lorsqu'un élève essaie de se connecter il rentre son identifiant ldap³ et son mot de passe habituel. L'application exécute alors une requête serveur pour essayer de se connecter⁴ et lui renvoie un token perpétuel pour lui permettre de télécharger ses données. Il peut se déconnecter à tout moment sur la page menu.

Tant qu'un élève est connecté, les données sont sauvegardées dans le `localStorage` ce qui lui permet de compléter plus tard son questionnaire.

À chaque déconnexion, toutes les données sont supprimées.

4 Serveur

La partie serveur a été conçue avec Django et est constituée de deux applications qui ont des rôles très différents :

- L'API : Elle contient la partie modèle ainsi que la partie API avec notamment les requêtes et les `serializer`. Elle est accessible via l'url `/api/`
- Le panneau de contrôle : Cette application Django ne fait que de l'affichage de résultat. Aucun traitement des données n'y est fait. Cette application est accessible via l'url `s2ip`.

Une troisième application pourrait être une application élève pour lui permettre de répondre au questionnaire directement en ligne.

4.1 Préambule à Python

Pour exécuter Python, on utilise un environnement virtuel (`virtualenv`) sous Python 3. On utilise pour cela les commandes suivantes (aucune notion d'ordre dans cette liste) :

- `pip install virtualenv` : Attention à bien vérifier que pip est lié à Python3 et non à Python2. Sinon utiliser la commande équivalente : `python3 -m pip install virtualenv`
- `virtualenv -p python3 python_env` : Créer l'environnement Python sous le nom de `python_env` (un dossier est créé à cette position). Pour ma part ce dossier est dans le dossier du projet et renseigné dans le `.gitignore`

3. `prenom.nom@enpc.fr`

4. méthode `StudentsData.connect`

- `pip install -r requirements.txt` pour installer tous les packages spécifiés dans le fichier `requirements.txt`.
- `source python_env/bin/activate` : À exécuter dans chaque terminal qui va utiliser Python pour lui indiquer qu'on se place dans l'environnement virtuel `python_env`.
- Installer toutes les dépendances (Attention à avoir les bonnes versions, Django a eu de nombreuses modifications par exemple) :
 - `pip install Django (2.0.1)` : pour le serveur en lui même
 - `pip install django-rest-framework` : pour l'API (générer les `.json` à partir des requêtes et vice-versa).
 - `pip install django-cors-headers` : pour autoriser l'application à lancer des requêtes.
- `python` : Lancer le shell dans un terminal. `Ctrl + Z` pour le fermer sans fermer le terminal.
- `python manage.py runserver` : Lance le site, accessible au `127.0.0.1:8000` si on est dans le dossier du projet (ici `Click2Evaluate_server`).
- `python manage.py makemigrations main` et `python manage.py migrate` pour construire la Bdd à partir des modèles.

4.2 Modèles

Les données sont stockées selon la représentation de la figure 5 où chaque flèche représente une *clé étrangère*. Ainsi dans la classe `Course`, l'attribut `typeForm` est une clé étrangère qui pointe vers une instance de `TypeForm`.

Dans l'idée il y a trois groupes distincts :

- Le groupe **élève** avec les classes `Student` et `Département` : chaque élève appartient à un département.
- Le groupe **Questionnaire** qui contient un ensemble de questions (`Question`), ces dernières étant regroupées au sein de types de questionnaires (`TypeForm`).
- Le groupe **cours** avec les classes `Course` et `Group`. Par exemple : TDLOG est un cours qui est notamment lié à un questionnaire (par ex. *Classique*) et à une plage de disponibilité des questionnaires. À chaque cours correspond au minimum un groupe qui contient trois informations : son numéro, le délégué (= un élève) ainsi que le cours auquel il est lié.

Viennent ensuite la classe `Survey`, qui traduit la phrase *M. Dupont doit évaluer le cours de TDLOG dans le groupe 0*, et la classe `QuestionWithAnswer` qui regroupe l'ensemble des questions (et des réponses) posées à un élève particulier, relativement à un cours.

4.3 Remplir la base de données

En première approche, nous avons décidé de construire la base de données à partir de fichiers excels que le S2iP peut remplir en amont sur son ordinateur puis uploader sur le serveur.

4.3.1 Format des données d'entrées

Nous avons considéré les deux fichiers `.csv` (avec « ; » comme séparateur) suivants :

- Le tableau avec sur chaque ligne un élève et un cours ($n \times p$ lignes où n est le nombre d'élèves et p le nombre moyen de cours par élève) : on en tire les informations sur les élèves (mail, ldap, département) ainsi que les informations sur les `Survey`. On se référera à ce tableau sous le nom de **fichier élèves** (format Tableau 7).
- Le tableau où chaque ligne représente un cours : identifiant, label, dates d'évaluation... ainsi que les délégués (écrits avec les ldap séparés par des « , »), il y a autant de groupes que de

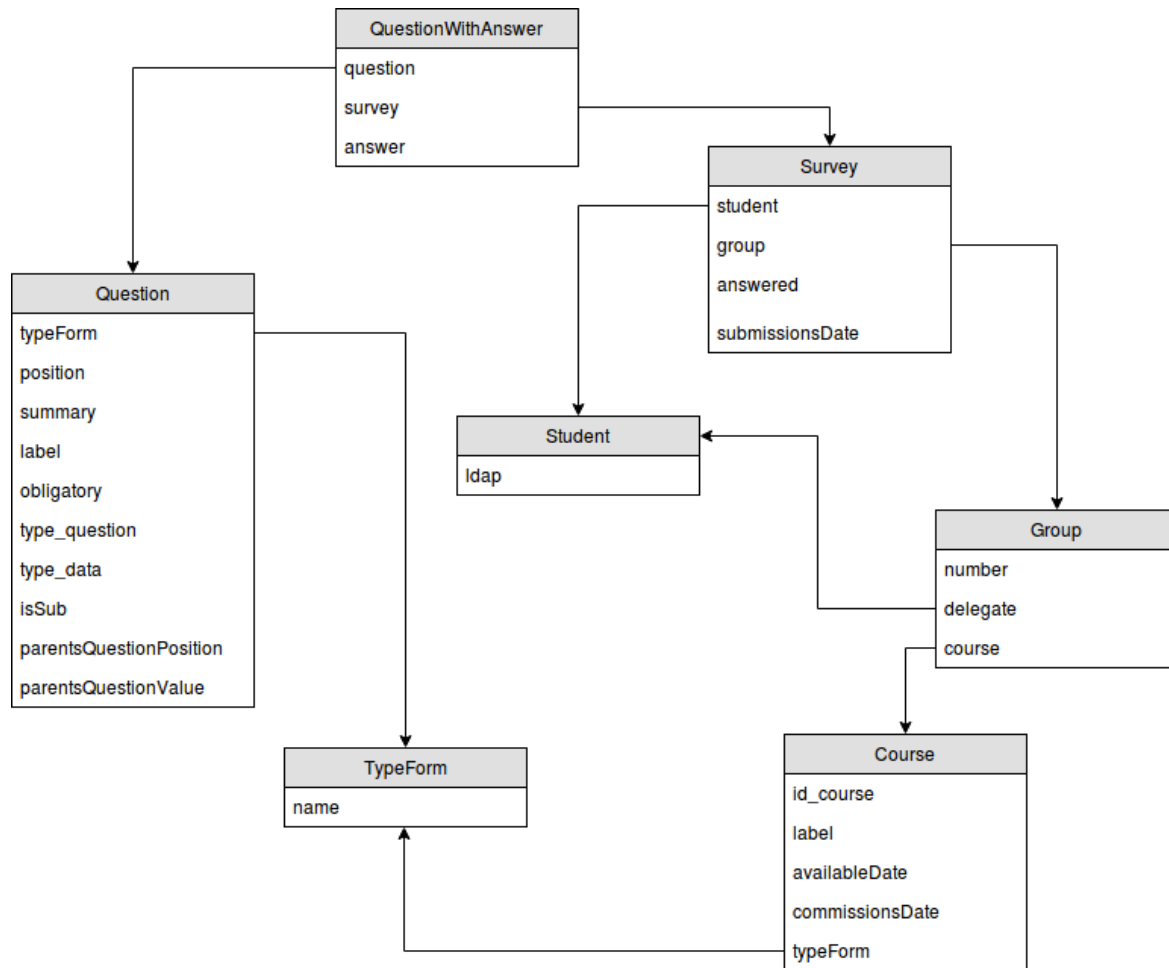


FIGURE 5 – Représentation UML des différents modèles utilisés

Intitulé des colonnes	Exemple
id_cours	MMC1
label	Mécanique débutant
commissionsDate	1836-08-21T00 :00 :00Z
availableDate	1785-02-10T00 :00 :00Z
typeForm	Classique
delegates	henri.navier@enpc.fr,augustin.cauchy@enpc.fr

FIGURE 6 – Format des données modules (.csv avec « ; » comme séparateur)

Intitulé des colonnes	Exemple
NOM	NAVIER
PRENOM	Henri
LOGIN_LDAP	henri.navier@enpc.fr
MAIL	henri.navier@eleves.enpc.fr
DEPARTEMENT	GMM
LIBELLE_MODULE	Mécanique débutant
CODE_MODULE	MMC1
SC_GROUPE	0
PERIODE	Automne

FIGURE 7 – Format des données élèves (.csv avec « ; » comme séparateur)

délégués : le premier délégué est dans le premier groupe et ainsi de suite. On se référera à ce tableau sous le nom de **fichier cours** (format Tableau 6).

Le fichier élève correspond directement à une extraction de la base de données **Gede** tandis que le fichier cours doit être rempli manuellement.

Après discussion avec Mme Gérard, il a été acté que **département** d'un élève correspond au premier mot de la chaîne de caractère indiquée dans la colonne **DEPARTEMENT** (avant le premier espace). Par exemple le département de « *IMI Finance* » est uniquement « *IMI* ».

Remarques :

- Le fichier .csv doit impérativement avoir un ; comme séparateur (utiliser LibreOffice au besoin).
- Tous les champs doivent être exactement ceux là.
- les **typeForm** doivent être définis préalablement sur le site internet,
- les dates doivent respecter strictement ce format là : AAAA-MM-JJTHH:MM:SSZ
- les ldap des délégués doivent être séparés par des « , » sans espaces
- Dans le fichier de cours, ne préciser que les cours qui doivent être évalués
- Dans le fichier élève, le numéro de groupe vaut 0 s'il n'y a pas de groupes ou un numéro ≥ 1 s'il y en a plusieurs.

4.3.2 Importation

Afin de construire la base de données nous faisons l'hypothèse que les données sont exactes *i.e.* il est possible de construire une base de données intègre et complète avec les données fournies. Si ce n'est pas le cas, une erreur apparaîtra sur le serveur et il faudra recommencer après avoir modifié les fichiers d'entrées.

La gestion des questionnaires n'est pas considérée ici, l'ajout de nouvelles questions se fait via le panneau d'administration de Django : /admin.

Le code de l'importation est géré dans `api.models` tandis que l'upload est réalisé via l'url `/s2ip/import` (cf 4.5.5).

Cette fonction en pseudo-code :

1. On ouvre les données dans `media/course_file.csv` et `media/student_file.csv` et on les convertit en `.json` *via* `pandas` (voir la fonction `convert` de `api.csvToJson`)
2. On efface la base de données précédente afin d'éviter de la faire grossir inutilement de semestre en semestre, il faut donc être particulièrement vigilant à avoir sauvegardé toutes les données utiles.
3. On construit les élèves à partir de la lecture du fichier élève. On rajout les départements qui n'existent pas déjà au fil de l'eau.
4. On construit les cours à partir de la lecture du fichier cours et pour chaque cours les groupes correspondants, en fonction du nombre de délégués renseignés.
5. On construit ensuite les **Survey** à partir du fichier élève : un **Survey** par cours suivi par élève.

Remarques :

- Cette requête peut être très très longue (de l'ordre de 10 à 15'), à voir suivant la BdD et le serveur utilisés.
- Les **QuestionWithAnswer** sont construites à l'envoi des réponses par les élèves. Cela autorise donc à modifier les questionnaires après avoir uploadé les fichiers élèves et cours.
- Des modifications marginales peuvent être effectuées directement sur le panneau d'administration de Django `/admin` afin de par exemple modifier une date limite pour un cours.

4.4 API

L'API a été codée avec le **Django-Rest-Framework** est des **serializers** qui permettent automatiquement de transformer un modèle en `.json`. Les requêtes s'écrivent alors très facilement dans la Vue : on fait une requête sur la base de données pour obtenir les données qui nous intéressent, on utilise un **serializer** dessus et on les renvoie.

L'ensemble de ces requêtes sont sécurisées (cf ??) et sont toutes en lecture seule, sauf l'envoi d'une réponse : seul l'utilisateur concerné par cette requête peut y accéder. Cela est permis par l'utilisation d'un token ajouté à chaque requête (dans le header). Les requêtes complètes avec la gestion de l'authentification sont décrites dans la Section 5.5. Ci-dessous l'ensemble des requêtes gérées par l'API :

4.4.1 Obtenir la liste des cours d'un élève donné

Requête : `/courses/prenom.nom@enpc.fr/`

Cette requête renvoie l'ensemble des cours⁵ actuellement suivis (= semestre en cours) par l'élève et pour chacun les informations suivantes :

- L'identifiant du cours (clé primaire fournie par Django), utilisé pour uploader les réponses.
- Le nom raccourci du cours (ex. TDLOG) (`id_course`)
- La plage de réponse (dates de début et de fin)
- Le type de questionnaire (le nom de ce dernier) (ex. « Classique »)
- Le groupe de l'élève (0 si un seul groupe)
- Si l'élève a déjà évalué ou non ce cours
- La date de la dernière modification de la réponse (aucun sens si l'élève n'a pas déjà évalué le cours).

5. En réalité il renvoie la liste de **Survey** Django correspondant à un élève

Listing 1 – Résultat de la requête `courses/prenom.nom@enpc.fr/`

```

1  [
2      {
3          "id": 1
4          "id_course": "TDLOG",
5          "label": "Techniques de développement logiciel",
6          "commissionsDate": "2018-01-27T00:00:00Z",
7          "availableDate": "2018-01-01T16:10:41Z",
8          "typeForm": "Classique",
9          "delegate": "herve.andres@enpc.fr",
10         "group": 0,
11         "answered": false,
12         "submissionDate": "2018-01-02T16:21:04Z"
13     },
14     ... (d'autres cours)
15 ]

```

4.4.2 Obtenir un questionnaire donné

Requête : `/typeForm/<nom>` (ex. `/typeForm/Classique`).

Cette requête en lecture seule renvoie la structure d'un questionnaire correspondant à un nom donné (ex. "Classique"). En réalité, renvoie la liste des questions, chaque question possède un id (clé primaire de Django), utilisé avec l'id de Survey afin de modifier les réponses.

Listing 2 – Résultat de la requête `/typeForm/Classique`

```

1  [
2      {
3          "id": 1,
4          "position": 0,
5          "summary": "Maintien",
6          "label": "Pensez-vous que ce cours doit être maintenu ?",
7          "obligatory": true,
8          "type_question": "selectOne",
9          "type_data": "Oui, sans changements;...;Non",
10         "isSub": false,
11         "parentsQuestionPosition": 0,
12         "parentsquestionsValue": 0
13     },
14     ... (Autres questions)
15 ]

```

4.4.3 Envoyer des réponses

Requête pour lire une réponse : `/answer/<id survey>/<id question>/`

Requête pour écrire une réponse : `POP /answer/<id survey>/<id question>/ answer="<answer>"`

Lorsqu'une réponse a déjà été envoyée, on mets à jour l'ancienne réponse. À réception d'une réponse on porte à True l'attribut correspondant de Survey et on met à jour la date.

Dans le cas où l'individu ne renvoie qu'une seule réponse, on considère tout de même qu'il a répondu à l'ensemble du questionnaire (cette fonctionnalité est gérée au niveau de l'application).

4.5 Panneau de contrôle

Étant donné que nous étions obligé de mettre en place un moyen d'uploader les fichiers de données et d'exporter les réponses aux questionnaires, nous avons décidé de coder un panneau de contrôle pour rendre ces fonctionnalités ergonomiques et afficher quelques statistiques en temps réel. Ce panneau de contrôle est accessible *via* authentification sur l'url `/s2ip/`.

En collaboration le S2iP, il a été décidé de mettre en place les fonctionnalités présentées dans les sections suivantes.

4.5.1 Principe général du code

Le fonctionnement est plutôt élémentaire : à chaque fonctionnalité correspond une page web (une url) qui est associée à une vue et à un template.

Les templates en jinja2 utilisent Bootstrap pour avoir le visuel et par soucis de simplicité n'utilisent pas d'héritage (ce qui est pénible à chaque modification du menu du haut).

Les vues ne réalisent **aucune** modification sur les données directement et font à chaque fois appel à `api.models.py`. Dans le cas de fonctionnalités manquantes (fonction pour compter le nombre d'élèves qui suivent un module par exemple), les implémenter dans `api.models.py`

4.5.2 Visualisation des taux de réponses par cours ou par département

Cahier des charges : Mme Gérard souhaitait pouvoir visualiser facilement pour chaque module et département le taux de réponses.

Réalisation : Pour cela, nous avons décidé de construire un template générique `specific.html` qui affiche ces résultats en trois colonnes. Afin d'obtenir un joli visuel et de donner la possibilité à l'utilisateur de trier les colonnes ou d'effectuer des recherches, nous avons utilisé *Bootstrap-table*⁶. Les différentes quantités affichées sont calculés dans `api.models` à travers les fonctions `nb_XXX` (ex. `Departement.nb_students`).

Amélioration : Utiliser le même affichage pour réaliser un suivi élève par élève.

4.5.3 Visualisation des réponses aux questions

Cahier des charges : Mme Gérard souhaitait connaître rapidement le taux de satisfaction pour chaque cours.

Réalisation : Même si la question de la satisfaction est présente dans l'ensemble des questionnaires, elle peut être présente sous des formes différentes (pas le même intitulé). Nous avons donc décidé de proposer une visualisation des résultats pour une question donnée (d'un type de questionnaire donné). Cette visualisation correspond à un histogramme de distribution des réponses pour les questions de type `select` et `selectOne`. Le code CSS pour cet histogramme est défini dans `/static`.

Problème : Mme Gérard souhaiterait qu'on puisse rassembler les questions de satisfaction. Cela pourrait être envisageable en supposant qu'elles ont exactement le même intitulé.

6. <http://bootstrap-table.wenzhixin.net.cn/>

Pistes d'amélioration : Nous pourrions proposer des affichages différents pour chaque type de question comme par exemple un nuage de mots pour les questions de type `text` ou `inline`.

Nous pourrions également proposer un tri pour ne regarder que les réponses à cette question pour un cours donné et non pas pour l'ensemble des cours où cette question a été posée.

4.5.4 Exportation des réponses

Cahier des charges : Nous souhaitons exporter au niveau d'un cours ou d'un groupe, au sein d'un (ou plusieurs dans un `.zip`) fichier excel (ou `.csv`) comportant les informations suivantes :

- Le nom du cours (dans le cas d'une synthèse avec plusieurs cours)
- Le numéro du groupe (dans le cas où il y a plusieurs groupes dans le fichier)
- Le nom de la personne ayant répondu à ce questionnaire (le S2iP rendra anonyme cette colonne)
- La date de la dernière réponse enregistrée
- Une colonne par question de type : `inline`, `text` ou `selectOne` contenant la réponse (pas le numéro de la réponse dans le cadre d'un `selectOne`)
- n colonnes pour les questions de type `select` et pour chaque colonne *Vrai*, ou rien suivant la réponse de l'élève.
- Les questions apparaissent dans l'ordre de leur position dans le questionnaire.

De plus nous souhaitons pouvoir générer des fichiers dits *anonymés* qui ne font pas apparaître ni le nom des élèves ni la date de réponse aux questionnaires.

Réalisation : Nous avons utilisé le module `bootstrap-table` pour afficher la liste de tous les modules et proposer d'en sélectionner uniquement certains. Au clic sur un des boutons *Télécharger*, la liste des identifiants (ex. [`'TDLOG'`, `'O2IMI'`]) est communiquée à la vue directement par l'url *via* une requête GET.

Ces informations sont ensuite traitées dans la fonction `export_zip_file` qui construit les réponses aux questionnaires pour des modules donnés : un fichier `.csv` par module est construit dans le dossier `/media/` puis ils sont regroupés dans un fichier `.zip` disponible au téléchargement.

Nous avons utilisé les modules `tarfile` et `csv`.

Génération des fichiers de réponses : Nous avons défini pour les classes `Question`, `TypeForm`, `QuestionWithAnswer` et `Survey` des fonctions `export_head` qui permettent de construire l'entête du fichier `.csv` (la première ligne) et des fonctions `export` pour exporter une réponse. On arrive ainsi à reconstruire aisément les réponses.

4.5.5 Importation des données

L'importation des données est déléguée au modèle de l'API, le mécanisme est décrit dans 4.3.2. Sur cette page, un formulaire permet d'uploader les fichiers `.csv` qui sont sauvegardés aux emplacements respectifs : `/media/course_file.csv` et `/media/student_file.csv` : la fonction d'importation `create_database` ouvre ces deux fichiers.

5 Mécanisme d'authentification

Deux mécanismes d'authentification sont considérés, pour l'application mobile et pour le panneau de contrôle :

- L'authentification au niveau de **l'application** permet de restreindre l'accès aux seuls élèves et de garantir que la personne qui essaie d'accéder à des questionnaires est bien la personne qu'il prétend être. Les identifiants doivent impérativement être identiques aux identifiants centralisés

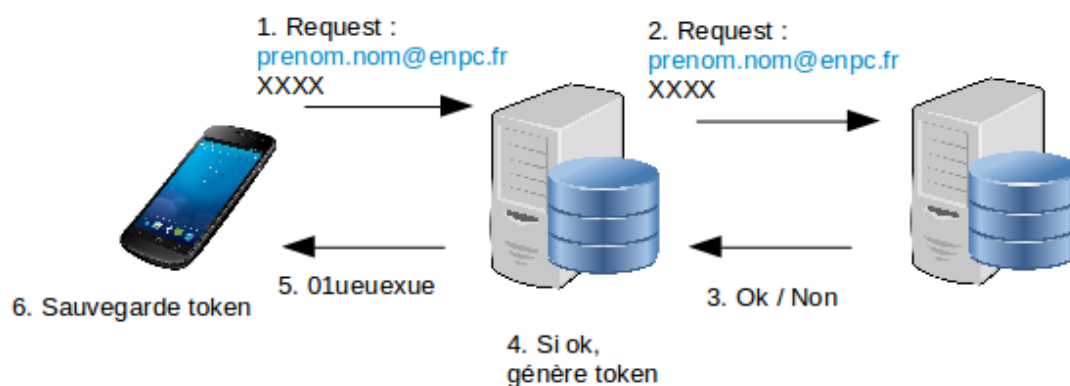


FIGURE 8 – Schéma du mécanisme d'authentification. Le serveur à droite correspond à l'annuaire ldap.

élèves afin de ne pas avoir des mots de passe différents. Pour cela, nous utilisons le service d'authentification centralisée ldap⁷

- L'authentification au niveau du **panneau de contrôle** permettra de n'autoriser l'accès à l'outil de gestion de campagne d'évaluation uniquement aux personnes autorisées (le S2iP).

5.1 Schéma général d'authentification

L'authentification avec l'application se réalise en six étapes, présentées sur le schéma 8 :

1. Envoi au serveur d'un couple identifiant / mot de passe
2. Transmission du serveur à l'annuaire ldap de ce couple
3. Réponse du ldap, soit positive, soit négative
4. Si la réponse est positive, le serveur génère un token associé à cet utilisateur
5. Le serveur renvoie alors ce token à l'application
6. L'application sauvegarde ce token dans le `local-storage` pour, notamment :
 - En cas de requête, le communiquer. En cas d'erreur (si le token n'est plus valide), l'utilisateur est déconnecté et un nouveau token doit être généré.
 - Au lancement de l'application, on regarde dans le `local-storage` si un token est en stock et dans ce cas on ouvre directement la page du menu
 - En cas de déconnexion, le token est supprimé du `local-storage`.

Les sous-sections suivantes présentent les solutions pour répondre à chacune de ces étapes.

5.2 Mise en place d'utilisateurs

Afin de mettre en place un système d'authentification, il y a besoin de mettre en place des `User`⁸. Dans notre cas, ils correspondent globalement aux étudiants. Nous avons donc rajouté un champ `User` en `OneToOne` dans la classe `Student`.

⁷. Pas encore mis en place en date du 4 février. La DSI nous a accordé un compte en mode lecture et nous mettrons cela en place avec eux dans les semaines à venir

⁸. <https://docs.djangoproject.com/en/2.0/topics/auth/>

Cela nous a conduit à mettre à jour la fonction `create_database` pour rajouter ces utilisateurs et les supprimer initialement ⁹

Afin de donner un accès particulier aux administrateurs, nous avons créé un document `controlPanel/admins.txt` qui doit contenir les identifiants des administrateurs (un seul par ligne, sous la forme `prenom.nom@enpc.fr`). Ces utilisateurs particuliers sont les seuls à pouvoir se connecter au panneau d'administration. Ces super-utilisateurs sont créés à chaque fois que la page Home est chargée. Ils se connectent avec leurs mots de passe DSI.

5.3 Mise en place de l'authentification par token pour l'API

Nous avons décidé de travailler directement avec Django Rest Framework pour l'authentification par Token ¹⁰. Pour cela, il y a plusieurs étapes :

- Paramétrer `settings.py` (dans `INSTALLED_APP` et `REST_FRAMEWORK` et penser à migrer la base de données (`python manage.py migrate`))
- Mettre en place un processus de génération de token : on utilise pour cela la vue prédéfinie `views.obtain_auth_token`.
- Mettre en place l'authentification par Token au niveau des requêtes de l'API et des permissions associées (en cours)

5.4 Gestion de l'authentification au niveau de l'app

Pour construire le token, nous utilisons la vue `obtain-auth` déjà définie par DRF. La requête, de la forme `curl -data "username=s2ip&password=password" http://127.0.0.1:8000/api/connect/` renvoie un token si l'identification est correcte.

L'idée est d'exécuter cette requête sur la page login et d'enregistrer le token dans le local-storage ainsi que dans le champ `token` de `StudentsData`.

Pour rajouter un token dans une requête, nous utilisons la syntaxe suivante ¹¹ :

Listing 3 – Syntaxe pour une requête avec authentification par token

```
this.http.get(url, {
  headers: {
    "Authorization": 'Token ' + this.token
  }
})
.subscribe(...
```

5.5 Gestion des permissions

Au niveau de l'API ¹² : On souhaite mettre en place les permissions suivantes sur les requêtes :

- Seul l'élève concerné peut accéder à la liste de ses cours
- Seul l'élève concerné peut envoyer ou lire ses réponses
- N'importe quel individu authentifié peut télécharger un questionnaire (questions)

Pour cela, nous avons défini les permissions `IsAnswerFromStudent` et `AreCoursesFromStudent` dans le fichier `api.permissions.py`. Nous nous sommes référés à la documentation correspondante ¹³//

9. Remarquons qu'il ne faut pas supprimer le *super utilisateur* `s2ip` qui permet d'accéder au panneau de contrôle.

10. <http://www.django-rest-framework.org/api-guide/authentication/>

11. Remarque : Il a fallu modifier le `Http` utilisé (<https://angular.io/guide/http>)

12. Avec Django Rest Framework

13. <http://www.django-rest-framework.org/api-guide/permissions/>

Concrètement, il faut bien veiller à définir cette permission dans les vues correspondantes et à appeler la fonction `self.check_object_permissions(request, obj)` où dans notre cas précis `obj` correspond à `courses` ou à `qwa`.

Pour résumer, on dispose des cinq requêtes suivantes, toutes sécurisées.

- `curl -X POST -data "answer=<answer>" http://127.0.0.1:8000/api/answer/<id_survey/<id_question>/ -H 'Authorization: Token 0671f'` pour **ajouter ou modifier une réponse**.
- `curl -X GET http://127.0.0.1:8000/api/answer/<id_survey/<id_question>/ -H 'Authorization: Token 0671f'` pour **lire une réponse**
- `curl -X GET http://127.0.0.1:8000/api/typeForm/Classique/ -H 'Authorization: Token 0671f'` pour **lire les réponses à un questionnaire**
- `curl -data "username=s2ippassword=password" http://127.0.0.1:8000/api/connect/` pour **se connecter** (et obtenir un token le cas échéant).
- `curl -X GET http://127.0.0.1:8000/api/courses/fabien.chaillard@enpc.fr/ -H 'Authorization: Token 06712ec'` pour **obtenir la liste des cours d'un élève**.

Au niveau du panneau de contrôle :

5.6 Utilisation d'un annuaire ldap

Nous n'avons pas encore trop travaillé sur ce point là, l'adresse du serveur ldap semble inaccessible en dehors du réseau de l'École. Nous avons donc décidé, dans un premiers temps de simuler cette étape à travers la création d'un *authentication backend* personnalisé¹⁴.

Des premières recherches nous ont montré l'existence des packages suivant pour se connecter en ldap :

- Installer ldap : `pip install ldap3` (se placer dans le virtualenv défini au préalable)
- Installer : `pip install django-python3-ldap`¹⁵
- Plusieurs packages Django disponibles dont `Django-auth-ldap`¹⁶

5.7 Note sur la différence entre SSO et ldap

La DSI nous propose deux technologies pour se connecter à l'authentification centralisée :

- La connexion avec **Cas** qui permet une connexion en SSO, cela est particulièrement adapté pour les connexions sur un site web mais pas spécialement pour une application.¹⁷ Notamment parce que la page d'authentification SSO n'est pas adaptée pour mobile.
- La connexion en interrogeant directement l'annuaire **ldap** : à travers l'utilisation de packages Python, on peut directement utiliser l'authentification ldap¹⁸

6 Déploiement

6.1 Serveur

Nous utilisons `mod_wsgi` pour gérer le serveur simplement. Les instructions sont décrites directement sur le repo git du développeur¹⁹.

14. <https://docs.djangoproject.com/en/2.0/topics/auth/customizing/#authentication-backends>

15. <https://github.com/etianen/django-python3-ldap>

16. <https://django-auth-ldap.readthedocs.io/en/latest/>

17. https://fr.wikipedia.org/wiki/Central_Authentication_Service

18. https://fr.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

19. https://github.com/GrahamDumpleton/mod_wsgi

Une fois connecté en ssh, il faut :

1. Ouvrir l'environnement python `source python_env/activate/bin`
2. `python manage.py collectstatic`
3. `python manage.py runmodwsgi` pour lancer le serveur en tant que tel

6.2 Webapp

Nous avons souhaité dès le départ que les élèves puissent ou non télécharger l'application et puissent continuer à répondre aux questionnaires sur ordinateur. La solution la plus simple que nous avons décidé de mettre en place est de rendre accessible l'application directement en ligne. Si l'utilisateur y accède depuis un ordinateur, l'interface n'est pas des plus adaptée (les mouvements ne sont pas très cohérents) mais tout est fonctionnel. Par contre s'il y accède depuis le navigateur de son smartphone, les fonctionnalités sont identiques, si ce ne sont les notifications non présentes.

Techniquement, on build le projet pour navigateur *via* la commande `ionic cordova build browser` puis on copie le dossier `platforms/browser/www` sur le serveur. Nous nous sommes référés au tutoriel d'OpenClassRoom²⁰ pour déployer le site ainsi copié.

6.3 Applications

Une fois les applications totalement achevées (authentification réellement mise en place et résolution d'éventuels bugs non encore détectés), il faudra les déployer.

M. Clerc a évoqué une piste avec des abonnements annuels pour éviter de passer par les store, aucune décision n'a été prise avec le S2iP.

6.4 Base de données

Dans une optique de tests, l'application utilise une base de données sqlite qui est très lente. Dans l'optique de gagner en efficacité, nous installerons un autre type de base de données au moment du déploiement sur les serveurs de la DSI.

7 Pour aller plus loin

De nombreuses fonctionnalités pourraient être rajoutées à l'outil et notamment :

- L'ajout de la fonctionnalité **emploi du temps** qui permettrait à l'élève de recevoir par notifications la salle où il a cours de manière plus ergonomique et personnalisée que l'actuel `emploidutemps.enpc.fr`. Pour cela, il faudrait concrètement fusionner avec l'application d'emploi du temps développée par un autre groupe dans le cadre du cours de TDLOG et développer un outil pour permettre au serveur de connaître en temps réel où ont lieu chaque cours et pour quel groupe. Cette fonctionnalité permettrait d'encourager les élèves à installer l'application.
- Permettre aux **délégués** de **se connecter** sur le site internet pour **télécharger les réponses** aux cours qu'ils suivent. Ils pourraient alors disposer du guide pour les aider à faire leur synthèse et pourraient la remettre directement sur ce site internet.
- L'envoi automatique des réponses aux questionnaires aux différents délégués par mail après les avoir anonymées²¹.
- Ajout d'informations concernant le délégué sur l'application lui rappelant quels cours il doit évaluer.
- Ajout d'informations supplémentaires sur le cours comme les objectifs du cours par exemple (sur l'application).

20. <https://openclassrooms.com/courses/mise-en-place-des-serveurs-apache-et-dns>

21. <https://docs.djangoproject.com/en/2.0/topics/email/>

- Adapter l’ergonomie de l’application pour ordinateurs
- Permettre un suivi individuel des réponses par élève : quels élèves n’ont pas répondu aux questionnaires ?

8 Remerciements

Nous remercions Barbara Gérard pour avoir initialement proposé ce sujet puis pour nous avoir challengé notre réalisation tout au long du projet. Ces rencontres régulières nous ont permis de comprendre le besoin et d’identifier les fonctionnalités clés.

Nous remercions Xavier Clerc pour l’encadrement technique et son expertise, notamment pour les technologies à privilégier.

Nous remercions Catherine Moraillon, de la DSI, pour sa réactivité : pour nous communiquer l’extraction des données élèves depuis Gede et discuter du déploiement de l’outil et de la partie authentification. Nous remercions également Philippe Ferreira de Sousa, élève IMI de deuxième année pour nous avoir aidé dans le déploiement du serveur.

Références

- [1] Documentation ionic 3. <https://ionicframework.com>, 2017.