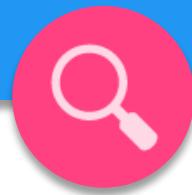




Projet de troisième année

Mai – juin 2015



APRAHAMIAN Kaïl

DUPONT Léo

KUHN Johann

HOUACINE Mehdi

INTRODUCTION.....	3
SUPPORTS DU PRODUIT.....	3
BASE DE DONNEES	4
STRUCTURE DE LA BDD	4
RECUPERATION DES DONNEES ET REMPLISSAGE DE LA BDD.....	4
SYNCHRONISATION DE LA BASE DE DONNEES AVEC ANDROID	5
<i>Mise à jour des données</i>	5
<i>Synchronisation des données</i>	6
<i>Version de la base de données</i>	6
AURION.....	7
ADE	11
LA WEB API D'ADE.....	11
NOTRE API PERSONNALISEE.....	11
<i>Utilisation de la fonction rechSalle</i>	12
<i>Utilisation de la fonction dispoSalle</i>	13
<i>Utilisation de la fonction dispoProf</i>	13
CONCLUSION	13
LA VERSION WEB.....	14
LE DESIGN	14
<i>Implémentation du design du site web :</i>	14
<i>Les media queries.....</i>	16
LES SCRIPTS, PAGE PAR PAGE.....	17
<i>La page index.php.....</i>	17
<i>La page recherche_avancee.php.....</i>	19
<i>La page fiche.php.....</i>	19
<i>La page connexion.php.....</i>	20
<i>La page aurion.php</i>	20
<i>La page recherche_professeur.php.....</i>	21
LA VERSION ANDROID.....	23
MOYENS TECHNIQUES	23
COMPATIBILITE.....	23
DESIGN.....	24
FONCTIONNALITES	24
<i>Navigation Drawer</i>	24
<i>Recherche de salles.....</i>	24
<i>Fiche salle</i>	26
<i>Recherche de professeur</i>	27
<i>Notes, Absences, Appréciations.....</i>	28
<i>À propos</i>	29
<i>Contribuer.....</i>	30
BILAN D'EXPERIENCE	30
CONTRIBUTION	32
CONCLUSION.....	34

INTRODUCTION

Tous les jours, des étudiants cherchent des salles libres pour pouvoir travailler ou simplement se réunir. Même les enseignants ont parfois besoin de trouver une nouvelle salle de classe.

Ce fut notre cas, un jour. Cependant, un problème s'est posé devant nous, toutes les salles que nous voulions investir étaient occupées. D'ici nous est venu une idée, celle de **MyESIEE** : une application qui nous aiderait à chercher une salle libre à n'importe quel moment de la journée.

Ce fût aussi l'occasion de créer un *hub* pour les étudiants, réunissant des informations intéressantes pour les élèves de l'ESIEE. Nous avons tout de suite pensé à Aurion qui réunit toutes nos notes, absences et appréciations mais qui n'est pas du tout adapté aux mobiles. Par la suite, après une première réunion avec notre tuteur, M. Hilaire, nous avons décidé de rajouter un annuaire des professeurs pour trouver toutes les informations utiles les concernant (numéro de bureau et email) et leur emploi du temps.

Supports du produit

Notre projet avait pour but de produire une application avant tout mobile, utilisable par les étudiants et les professeurs de l'ESIEE depuis leurs smartphones.

Les membres de notre équipe sont équipés de trois appareils Android et d'un appareil iOS, cependant, aucun ne possède d'ordinateur tournant sur Mac OS, ce qui rend la programmation pour iOS difficile, voire impossible. C'est pour cela que seule une application Android a été envisagée.

Afin de rendre notre produit utilisable sur les smartphones tournant sur d'autres systèmes d'exploitation, nous avons décidé de produire parallèlement une version Web de notre application, fonctionnant aussi bien sur PC que sur mobile.

BASE DE DONNEES

Structure de la BDD

Cette partie a été réalisée par Johann KUHN

Pour certaines fonctionnalités du site, nous avons à utiliser une base de données. Cette table, nommée **eroom** est constituée de quatre tables :

- Une table **prof** qui contient les informations des professeurs de l'ESIEE comme le nom, le resourceID (pour l'API ADE), le bureau et l'email.
- Une table **salle** qui contient les caractéristiques d'une salle comme le nom, le resourceID, le type de salle, la taille, la présence ou non d'un projecteur, d'un tableau et d'une imprimante.
- Une table **infos** qui contient la date de la dernière mise à jour de la base de données (db_last_update).
- Une table **contribution** qui contient l'id de chaque contribution, le type de contribution, la date de la contribution, le navigateur utilisé par l'utilisateur lors de la contribution, la version_android si utilisation de l'application mobile, l'email, la localisation (la page ou l'écran où l'utilisateur a fait sa contribution), le statut de la contribution et, enfin, le contenu.

Récupération des données et remplissage de la BDD

Cette partie a été réalisée par Mehdi HOUACINE

Les données des tables **salle** et **prof** proviennent en très grande partie de documents récupérés auprès du service de la planification de l'école. Il s'agit de documents Excel regroupant des informations sur toutes les salles de l'école (épi, étage, numéro de bureau et service rattaché entre autre, mais surtout le type de salle, la surface occupée et le nom de l'enseignant à qui appartient ce bureau s'il s'agit d'un bureau).

Cependant, ces documents n'étaient pas suffisamment complets pour notre usage (présence d'un projecteur, d'une imprimante, type de tableau notamment y étaient absents) et ces documents étaient assez anciens : ils n'avaient pas été mis à jours depuis novembre 2014.

Nous avons choisi de tenir à jour notre propre document Excel avec ces informations manquantes sur les salles que nous sommes allé vérifier par nous-mêmes (notamment Johann) dans presque toutes les salles de l'école sur une durée approximative d'une semaine.

Suite à cela, j'ai réalisé en Python une extraction de ces données (sur les salles et sur les enseignants) en les convertissant sous un format de données standard exploitable,

Rapport de projet de E3

le .csv. Une fois toutes ces données stockées dans une structure Python, je les ai stockés dans notre base de données grâce à une bibliothèque Python permettant des interactions avec une base de données MySQL : `mysql-connector`. Voici des exemples de code assez simples provenant de la documentation de cette librairie montrant comment créer une table en Python dans une base de données par le biais d'instruction SQL standard : <http://dev.mysql.com/doc/connector-python/en/connector-python-example-ddl.html>.

Pour les enseignants, la table `prof` est constituée d'un numéro de bureau provenant des fichiers Excel cités plus tôt, leur nom provient d'une liste de noms extraits de ces documents Excel et d'ADE par le biais de sa Web API, et leur adresse mail est générée par un simple script Python selon l'expression suivante : les sept premières lettres du nom de famille suivies de la première lettre du prénom ou bien "nom.prénom" si la taille du nom est inférieure à 7 lettres puis "@esiee.fr".

Nous sommes conscients au vu de l'ancienneté de ces données qu'une minorité d'entre elles est obsolète, pour cela nous avons donc prévu un module de participation sur l'application et le site Web afin que les étudiants qui l'utilisent puisse nous signaler les données erronées ou manquantes. Ces appréciations sont consignées dans une table de la base de données jusqu'à ce que nous les corrigeons.

Synchronisation de la base de données avec Android

Cette partie a été réalisée par Léo DUPONT

L'application Android utilise une copie de notre base de données principale afin de limiter la taille des requêtes HTTP et de proposer des prédictions pour les noms des salles et des profs (voir la partie Android).

Mise à jour des données

Afin de garder synchrones la BDD SQLite Android et la BDD MySQL du serveur, j'ai suivi ce tutoriel dans les grandes lignes : <http://programmerguru.com/android-tutorial/how-to-sync-remote-mysql-db-to-sqlite-on-android/> bien qu'il ait nécessité beaucoup de changements afin de l'adapter au projet. De plus, nous avons ici deux tables à synchroniser et non une seule.

De plus, contrairement au tutoriel, nous ne pouvons pas stocker dans la BDD du serveur une information indiquant que telle ligne a été synchronisée sur Android puisque notre application est destinée à être utilisée par plusieurs appareils (voir la partie suivante concernant la résolution de ce problème).

J'ai donc créé des scripts PHP permettant d'obtenir toutes les données des tables `salle` et `prof` au format JSON, en particulier le script `getData.php` qui est utilisé par l'application : https://mvx2.esiee.fr/mysql_sync/getdata.php?table=salle (pour récupérer la table `salle`).

Synchronisation des données

Afin de savoir si le contenu de la BDD d'un appareil Android correspond au contenu de la BDD du serveur, j'ai écrit un script PHP permettant d'obtenir un hash en SHA-256 de tout le contenu des tables `salle` et `prof`. De cette façon, il est facile de comparer le contenu de deux bases de données sans faire transiter un grand nombre de données par internet. Ce script est stocké dans le fichier `bdd.php` et accessible par cette URL :

<https://mvx2.esiee.fr/api/bdd.php?func=getHashVersion>

Le hash obtenu est stocké dans la mémoire du mobile à chaque fois qu'une mise à jour est effectuée. Il suffit alors de comparer le hash de version stocké dans l'appareil avec le hash de version obtenu par cette requête.

Version de la base de données

Afin de pouvoir afficher dans l'écran “À propos” de l'application la date de dernière mise à jour de la base de données principale, j'ai ajouté la fonction `getLastUpdate` au script `bdd.php`, accessible par cette URL :

<https://mvx2.esiee.fr/api/bdd.php?func=getLastUpdate>

Ce script se contente de récupérer la valeur de la clé `db_last_update` stockée dans la table `infos` et de la renvoyer.

AURION

Cette partie a été réalisée par Mehdi HOUACINE

Une part de notre application consiste à fournir aux étudiants d'ESIEE Paris la possibilité de consulter leurs notes, absences, appréciations et archives de tout ceci depuis l'application ou le site web. Ces données existent dans une base de données de l'école, mais son accès nous a été refusé pour des raisons de sécurité. J'ai donc choisi de construire une API permettant de récupérer ces données en fonction de certains paramètres puisqu'au lancement de ce projet, une telle API n'existe pas et n'était pas sujet d'actualité pour les développeurs de la plateforme Aurion.

Les principales difficultés de la mise en place d'une telle API reposent essentiellement pour nous sur la structure même d'Aurion qui fonctionne selon un affichage dynamique en Ajax : on ne peut donc pas accéder aux notes, par exemple, d'un étudiant à partir d'une url clairement définie telle que <https://aurionprd.esiee.fr?q=mes-notes&id=ABCD>.

C'est à partir de la même page <https://aurionprd.esiee.fr/faces/ChoixDonnee.xhtml>, à laquelle on peut accéder une fois connecté avec nos identifiants étudiants, que l'affichage évolue : va-et-vient dans les menus, affichage des notes, absences, etc...

À partir de ce constat, nous avons choisi d'utiliser les modules suivants :

- La bibliothèque Python `Selenium`, qui sert normalement à effectuer des tests d'automatisation pour s'assurer que les fonctionnalités d'un site web sont opérationnelles lors du développement. Elle permet d'instancier le navigateur de notre choix et de réaliser divers traitements sur la page en interagissant avec le DOM : en identifiant de façon certaine un élément d'une page (par sa classe HTML, son id HTML, son XPATH ou autre (voir la documentation ici : <http://selenium-python.readthedocs.org/en/latest/locating-elements.html>, au point 4)), on peut cliquer sur cet élément, le survoler, remplir et soumettre un formulaire entre autres.
- La bibliothèque Python `BeautifulSoup`, qui permet de parser du HTML et d'en extraire certaines informations très intuitivement grâce à une approche orientée objet (aperçu disponible ici : <http://www.crummy.com/software/BeautifulSoup/bs4/doc/#kinds-of-objects>).
- La bibliothèque JavaScript `PhantomJS`, qui permet de réaliser des traitements automatisés comme `Selenium`, mais surtout qui embarque un "headless browser" (un navigateur sans GUI) : le `GhostDriver`.

Tous les traitements de cette API sont donc réalisés en langage Python 3 pour des raisons de commodité : cette API me paraissait difficilement réalisable et maintenable, j'ai donc fait le choix d'utiliser un langage de programmation que j'utilise assez souvent pour réaliser des extractions Web et avec lequel je me sentais à l'aise. Ces traitements consistent donc à réaliser à la place de l'étudiant le petit

Rapport de projet de E3

parcours du combattant qu'il devrait faire sur son téléphone pour arriver à l'affichage des données puis à les extraire, comme suit :

- Utiliser la bibliothèque Selenium pour se rendre sur le site Aurion, s'y connecter avec le login et le mot de passe ESIEE d'un étudiant (remplir le formulaire de connexion et le soumettre), et réaliser le bon parcours vers les données désirées (cliquer sur les bons boutons dans le bon ordre, en essayant de les identifier de manière unique sans passer par des classes ou des identifiants HTML qui peuvent changer à chaque mise à jour, réaliser des survols de boutons (hover events) et naviguer dans la pagination d'affichage des données quand nécessaire). Selenium permet aussi d'extraire le code HTML de la page (ou d'un élément de celle-ci) une fois que les données voulues (notes, absences ou appréciations) sont affichées en Ajax sur Aurion.

On comprend par exemple assez intuitivement cet extrait du script api/config_selenium.py pour soumettre le formulaire de connexion d'Aurion :

```
# Identification des éléments de la page à remplir
form = driver.find_element_by_class_name('form')
loginBar = driver.find_element_by_id('j_username')
passwordBar = driver.find_element_by_id('j_password')

# Remplissage du formulaire et submit
loginBar.send_keys(l)
passwordBar.send_keys(pwd)
form.submit()
```

- Avant cela, configurer Selenium pour choisir le bon navigateur : le GhostDriver de Selenium, dans la bonne langue (Aurion propose un affichage en français et en anglais, mais l'affichage anglais est souvent mal, voire pas, traduit donc pas toujours exploitable), et avec les bons paramètres en terme de certificat pour pouvoir se rendre sur Aurion et ne pas se heurter à une page blanche (URL de type about:blank en général rencontrée par le GhostDriver sur certains sites chiffrés HTTPS). L'avantage d'utiliser le GhostDriver à la place d'un navigateur usuel est de ne pas avoir à charger inutilement une interface graphique dont le chargement peut allonger le temps d'exécution du script (en effet, aucun visuel n'est nécessaire puisque le parcours est automatisé côté serveur).

La configuration utilisée est la suivante, consultable dans api/mypkg/__init__.py :

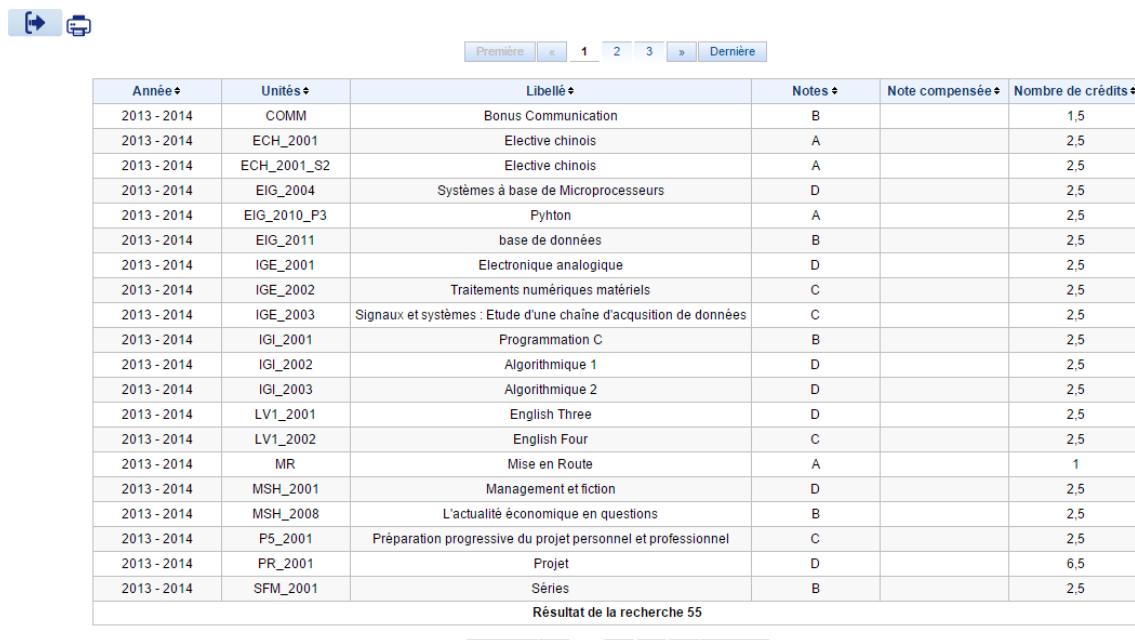
Rapport de projet de E3

```
# Configuration du webdriver en français pour accéder au Aurion fr.
webdriver.DesiredCapabilities.PHANTOMJS['phantomjs.page.customHeaders.Accept-Language'] = 'fr-FR'

# Instanciation du webdriver avec les paramètres nécessaires pour
avoir la permission d'accéder à Aurion.
driver = driver or webdriver.PhantomJS(service_args=['--ignore-
ssl-errors=true', '--ssl-protocol=tlsv1'])
```

- Une fois les données affichées sur le site et le code source HTML récupéré avec Selenium, analyser ce dernier et extraire uniquement les données nécessaires dans une structure en Python (ici un dictionnaire, une collection de type tableau associatif) avec la bibliothèque BeautifulSoup. Ici on extrait une <div> HTML contenant le système de pagination quand il est présent ainsi que le tableau de données. Le système de pagination est parsé afin de déterminer combien de pages de données il y a à extraire, et ce nombre est ensuite communiqué à une boucle de parcours de Selenium. On réalise alors une boucle mêlant les bibliothèques Selenium et BeautifulSoup qui permet d'appuyer sur le bouton "suivant" et d'extraire le tableau de données autant de fois qu'il y a de pages dans le système de pagination.

Un visuel du tableau de données et de la pagination au-dessus et en-dessous de celui-ci :



The screenshot shows a table of course data from the Aurion database. The table has columns for Année, Unités, Libellé, Notes, Note compensée, and Nombre de crédits. The data spans from 2013-2014 to 2014-2015. At the top of the table, there are two small icons: a blue square with a white arrow pointing right and a blue square with a white printer icon. Below the table, there is a navigation bar with buttons for 'Première', '<', '1', '2', '3', '>', and 'Dernière'. The number '1' is highlighted in blue, indicating the current page. At the bottom of the table, it says 'Résultat de la recherche 55'.

Année	Unités	Libellé	Notes	Note compensée	Nombre de crédits
2013 - 2014	COMM	Bonus Communication	B		1,5
2013 - 2014	ECH_2001	Elective chinois	A		2,5
2013 - 2014	ECH_2001_S2	Elective chinois	A		2,5
2013 - 2014	EIG_2004	Systèmes à base de Microprocesseurs	D		2,5
2013 - 2014	EIG_2010_P3	Python	A		2,5
2013 - 2014	EIG_2011	base de données	B		2,5
2013 - 2014	IGE_2001	Électronique analogique	D		2,5
2013 - 2014	IGE_2002	Traitements numériques matériels	C		2,5
2013 - 2014	IGE_2003	Signaux et systèmes : Etude d'une chaîne d'acquisition de données	C		2,5
2013 - 2014	IGI_2001	Programmation C	B		2,5
2013 - 2014	IGI_2002	Algorithmique 1	D		2,5
2013 - 2014	IGI_2003	Algorithmique 2	D		2,5
2013 - 2014	LV1_2001	English Three	D		2,5
2013 - 2014	LV1_2002	English Four	C		2,5
2013 - 2014	MR	Mise en Route	A		1
2013 - 2014	MSH_2001	Management et fiction	D		2,5
2013 - 2014	MSH_2008	L'actualité économique en questions	B		2,5
2013 - 2014	P5_2001	Préparation progressive du projet personnel et professionnel	C		2,5
2013 - 2014	PR_2001	Projet	D		6,5
2013 - 2014	SFM_2001	Séries	B		2,5

- Réorganiser les données avec des modules présents nativement en Python, comme la bibliothèque json de Python, pour les présenter et les afficher dans un format exploitable en PHP sur <https://mvx2.esiee.fr/aurion.php>. Ce traitement est réalisé dans la fonction data_to_json() dans api/parsing_bs4.py.

Rapport de projet de E3

- Finalement, il faut afficher ces données en PHP sur la page ci-dessus. Il faut donc paramétriser la page pour qu'elle appelle le script Python avec la fonction `shell_exec()` de PHP, qu'elle affiche des données au format JSON (avec la fonction `header("Content-type: application/json; charset=utf-8")`) de PHP) et pour qu'elle accepte des paramètres en POST qui sont nécessaires au déroulement du script (login ESIEE, mot de passe ESIEE, choix des données désirées (notes ou absences ou appréciations ou archives)) et pour éviter que ces deux premiers paramètres sensibles n'apparaissent en clair dans le navigateur ou sur le serveur. La construction de la requête et son exécution sont effectuées avec `cURL` en PHP et sont consultables dans le fichier `scripts/php/curl-post-request.php`.

Afin de faire des tests, mais aussi pour rendre cette API accessible pour d'autres utilisations, cette page accepte aussi des paramètres GET en se rendant sur <https://mvx2.esiee.fr/api/aurion.php?login=login&pwd=pwd&func=func>.

ADE

Cette partie a été réalisée par Léo DUPONT

Pour déterminer si une salle est libre ou non, nous avons choisi de nous baser sur l'emploi du temps de l'ESIEE, lequel est géré par le logiciel ADE.

La Web API d'ADE

Contrairement à Aurion, ADE propose une API Web étoffée, permettant d'obtenir n'importe quelle information disponible sur cette plateforme relativement facilement. Nous avons obtenu la documentation de cette API grâce à M. Bruno ROUGIER.

Cette API fonctionne par requêtes HTTP de type GET et fournit des réponses au format XML. Les requêtes sont de cette forme : <https://planif.esiee.fr/jsp/webapi?function=xxx>. Il s'agit à chaque fois de préciser la bonne fonction et les paramètres qui l'accompagnent. En général, il est nécessaire d'effectuer plusieurs requêtes d'affilée pour obtenir l'information désirée. Voici, par exemple, comment obtenir la liste des cours utilisant la salle 5201V le 19/06/2015 :

1. Connexion à une session. Afin de récupérer un sessionId :
?function=connect&login=lecteur1&password=
2. Choix du projet ADE. L'ESIEE crée un nouveau projet par année, le projet de l'année 2014-2015 est le numéro 4 :
?sessionId=14e0dae2bb6&function=setProject&projectId=4
3. Récupération du resourceId lié à la salle 5201V :
?sessionId=14e0dae2bb6&function=getResources&name=5201V&detail=0
4. Récupération des événements planifiés le 19/06/2015 utilisant la ressource 659 :
?sessionId=14e0dae2bb6&function=getEvents&resources=659&detail=0
&date=06/19/2015

Ceci nous retourne une liste d'éléments XML représentant chacun un cours planifié et ayant, entre autres, deux attributs intéressants de la forme :
endHour="10:00" startHour="08:30".

5. Déconnexion de la session :
?sessionId=14e0dae2bb6&function=disconnect

Notre API personnalisée

Comme nous le voyons, cette API permet de récupérer les informations souhaitées mais nécessite beaucoup de requêtes et d'interprétations des réponses XML. De plus, il n'est pas possible de rechercher les salles par des caractéristiques telles que la présence d'un projecteur ou d'une imprimante. Il nous a donc fallu créer notre propre

Rapport de projet de E3

API capable d'accepter les paramètres dont nous avons besoin (comme les caractéristiques d'une salle), de coupler les informations de notre base de données avec celles d'ADE, d'effectuer un algorithme pour déterminer les disponibilités d'une salle donnée, et de nous renvoyer la réponse dans un format clair et léger.

Pour cela j'ai créé une classe PHP ADE munie de toutes les fonctions nécessaires pour effectuer des requêtes vers notre base de données et vers ADE, ainsi que de choisir les salles concernées par nos critères de recherche. Cette classe permet d'obtenir une liste de salles avec leurs disponibilités au format JSON, mais également de générer une image de l'emploi du temps d'une salle ou d'un professeur à une date donnée (cette image est générée par la Web API d'ADE grâce à la fonction `imageET`). Voici le contenu du fichier `readme.md` que j'ai écrit pour accompagner cette API :

Utilisation de la fonction rechSalle

Cette fonction permet d'obtenir une liste de salles répondant à certains critères optionnels ainsi que leur disponibilité.

La réponse au format JSON est de cette forme : `[{"5004": "45"}, ...]`, où "5004" est le nom de la salle et "45" la disponibilité.

Format de l'URL et critères de recherche

Une requête peut être de la forme :

<https://mvx2.esiee.fr/api/ade.php?func=rechSalle&nom=5004&type=it&taille=m&projecteur=0&tableau=1&imprimante=0>

Voici la liste des paramètres possibles :

- `func=rechSalle` : pour utiliser la fonction de recherche de salles (seul paramètre obligatoire).
- `nom` : le nom complet de la salle en BDD. Si au moins un des paramètres `epi` ou `etage` est spécifié, le paramètre `nom` ne sera pas pris en compte.
- `type` : le type de salle recherchée (`it`, `elec` ou `banal`).
- `taille` : Peut prendre les valeurs `S`, `M` ou `L` (majuscule ou minuscule). Correspond à la taille de la salle, respectivement petite, moyenne et grande.
- `projecteur` : la présence d'un projecteur (`0` : non, `1` : oui).
- `tableau` : la présence de tableau(x) (`0` : aucun, `1` : blanc, `2` : noir, `3` : les deux).
- `imprimante` : la présence d'une imprimante (`0` : non, `1` : oui).
- `epi` : l'épi de la salle (correspond au premier chiffre des noms des salles).
- `etage` : l'étage de la salle (correspond au deuxième chiffre des noms des salles).

Format de la disponibilité d'une salle

La disponibilité d'une salle peut prendre ces valeurs :

- `--1` si la salle n'est pas disponible actuellement.
- `0` si la salle est disponible jusqu'à la fin de la journée.
- un autre entier correspondant au nombre de minutes durant lesquelles la salle est encore libre. Par exemple, si à 14h15, une salle a une disponibilité de 45, cela signifie qu'elle est actuellement libre mais qu'elle sera occupée à 15h00.

Utilisation de la fonction dispoSalle

Cette fonction permet d'obtenir une image au format GIF de l'emploi du temps d'une salle à un jour donné.

Format de l'URL et paramètres

Une requête peut être de la forme :

<https://mvx2.esiee.fr/api/ade.php?func=dispoSalle&nom=5004&date=06/18/2015>

Les paramètres `func=dispoSalle` et `nom` sont obligatoires. Le format du nom est le même que pour la fonction `rechSalle`.

Le paramètre `date` correspond à la date du jour souhaité au format américain "mm/jj/aaaa" (exemple : 06/18/2015 pour le 18 juin 2015). S'il est omis, la date d'aujourd'hui sera utilisée.

Les paramètres `largeur` et `hauteur` correspondent aux dimensions en pixels de l'image à générer.

Utilisation de la fonction dispoProf

Cette fonction permet d'obtenir une image au format GIF de l'emploi du temps d'un professeur à un jour donné. Cette fonction s'utilise comme la fonction `dispoSalle`, avec `func=dispoProf` et le paramètre `nom` faisant référence au nom d'un professeur enregistré en base de données.

Conclusion

Grâce à cette API personnalisée, basée elle-même sur la Web API d'ADE, nous pouvons obtenir facilement les informations nécessaires à notre projet à partir de la version Web comme de l'application Android avec une simple URL. De plus, cela garantit que les informations affichées sur le site Web et sur l'application seront les mêmes et qu'une modification du script PHP influera sur les deux versions du produit.

LA VERSION WEB

Le design

Cette partie a été réalisée par Kail APRAHAMIAN

Après quelques recherches sur les designs possibles de notre site Web, nous nous sommes penchés sur un style "Material Design" qui est une charte graphique réalisée par Google.

Pour nous aider à réaliser le design du site, nous nous sommes aidés d'un framework basé sur le Material Design nommé "Materializecss" (<http://materializecss.com>) qui propose de nombreux composants très adaptables et très épurés.

Il était compliqué de trouver des couleurs qui proposaient une belle harmonie. Après avoir essayé plusieurs tons de couleur, nous nous sommes penchés sur Material Palette (<http://www.materialpalette.com>) qui permet de montrer un aperçu d'un site internet et de fournir un fichier CSS avec les couleurs choisies.

Une fois que les éléments à utiliser étaient choisis et que les couleurs étaient définies, il ne restait plus qu'à dessiner des esquisses de nos pages et à les implémenter.

Implémentation du design du site web :

Nous avons donc choisi de réaliser le design du site Web en HTML 5 et CSS 3 qui sont les versions les plus récentes d'HTML et CSS. De plus, CSS 3 permet de définir des media queries afin de rendre un site Web responsive (voir sous-partie Media Queries).

Il nous fallait représenter toutes les fonctionnalités sur notre site internet, soit :

- Une page de **recherche de salle** avec des critères pour une recherche avancée qui comprend :
 - Quatre boutons permettant de choisir un type de salle ou la recherche avancée,
 - Une aire de texte qui va nous permettre de renseigner notre salle,
 - Un bouton permettant de lancer la recherche,
 - Une collection qui contiendra le numéro de salle, les icônes représentant les caractéristiques de la salle et le temps libre restant pour la salle.

Rapport de projet de E3

The screenshot shows a list of free rooms. Each room entry includes its number, a small icon representing the room type (e.g., a person icon for a classroom), and the status 'Libre'.

0112	<input type="checkbox"/>	Libre
0113	<input type="checkbox"/>	Libre
0114	<input type="checkbox"/>	Libre
0115	<input type="checkbox"/>	Libre
0162V	<input type="checkbox"/>	Libre
0163V	<input type="checkbox"/>	Libre

- Une fiche regroupant les informations de la salle choisie :
 - Une liste à puce qui présentera les informations de la salle,
 - Un sélecteur de date qui va permettre de choisir sur quel jour nous souhaitons les disponibilités de la salle,
 - Une image tirée d'ADE pour le jour et la salle choisie.

This screenshot shows the details for room 5309V++. It includes icons for electronic lab, projector, and printer, and descriptions of the room as a medium-sized room and a whiteboard room. It also indicates that the room is free for the day.

A modal window titled 'Choisir une date' (Select a date) displays a weekly calendar for June 17, 2015. A specific slot from 08h00 to 12h00 is highlighted in green, labeled 'IN3121:TPs Structures de données 1 5309V++ GEORGES JC. 08h00 - 12h00'.

- Une page permettant de rechercher un processeur qui comprend simplement une aire de texte et une fiche professeur très ressemblante à la fiche salle.
- La vie étudiante comportant trois parties : les notes, les absences et les appréciations. Les trois sont de design identique :
 - Un bouton pour chaque partie,
 - Un switch permettant d'accéder aux données archivées,
 - Une liste à puce sous forme de collection. Chaque puce contiendra une icône représentative de la donnée associée. (dans notre cas, la note ou si l'absence est excusée ou non).

The screenshot shows a list of courses with their respective grades, codes, and credit values:

C	Chinois CH_3201 Crédits : 2
Fx	Programmation d'applications graphiques IG_3604 Crédits : 3
B	Contrôle des systèmes bouclés IGE_3003 Crédits : 3
B	Conception VHDL de systèmes numériques IGE_3004 Crédits : 3
Fx	Eléments de traitements du signal IGE_3005 Crédits : 3
Fx	Systèmes d'exploitation IGL_3004 Crédits : 3

- La partie Contribuer de Johann KUHN (voir la partie Contribution).

Les media queries

Notre site Web devait absolument pouvoir s'adapter à n'importe quel type de support mobile car il devait remplacer notre application Apple. Pour cela nous utilisions "Responsive Test" (<http://responsivetest.net/#u=https://mvx2.esiee.fr/wip>) pour pouvoir avoir une vue de la page que nous souhaitions sur n'importe quel support. Il est donc possible d'avoir accès à notre site Web sur vos smartphones ou tablettes sans problème d'affichage.

Les scripts, page par page

Cette partie a été réalisée par Mehdi HOUACINE

La page index.php

C'est la page principale du site, qui dispose d'un filtre de recherche de salle réalisé en tant qu'application AngularJS, un framework d'applications JavaScript qui respecte l'organisation MVC : le fichier `index.php` constitue la vue, le contrôleur est géré dans un fichier `js/app.js` et est configuré dans un fichier `js/controllers/MainController.js`, les données du modèle sont rapatriées depuis l'API ADE que nous avons conçue pour le projet afin de connaître le statut d'occupation d'une salle (libre/occupée) mais aussi depuis notre base de données pour connaître le matériel dont ces salles disposent (tableau, imprimante, projecteur, etc....).

Le contrôleur

Le contrôleur (`js/controllers/MainController.js`) dans notre cas est très simple et est détaillé ci-après :

```
myApp.controller('MainController', ['$scope', '$http',
    function($scope, $http) {
        ...; // voir détails ci-dessous
}]);
```

Dans notre application `myApp`, nous créons un contrôleur `MainController` qui embarque deux services AngularJS :

- `$scope`, qui permet l'échange de données du modèle, entre le contrôleur et la vue via un moteur de template au travers d'expressions du type `{{ expression }}`.
- `$http`, qui permet de récupérer des données au format JSON grâce à la méthode `$http.get('path')` où 'path' est un chemin relatif ou absolu. Dans notre contrôleur, ce service nous sert à récupérer les données relatives aux salles de notre base de données au format JSON à cette URL : https://mvx2.esiee.fr/mysql_sync/getdata.php?table=salle, données que nous stockons dans la propriété `salles` de `$scope`. Donc `$scope.salles` contiendra un tableau JSON composé d'objets `salle` du type :

```
[  
 {  
     "nom": "0112",  
     "resourceID": "737",  
     "type": "banal",  
     "taille": "57",  
     "projecteur": "1",  
     "tableau": "1",  
 }
```

```

    "imprimante": "0"
},
...
]

```

tandis que `$scope.libres` contiendra un tableau JSON avec des informations sur les salles libres, récupérés en consultant notre API ADE à cette adresse : <https://mvx2.esiee.fr/api/ade.php?func=rechSalle>, composé d'objets du type :

```

[
  {'1205V': 0},
  {'3405': -1},
  ...
]

```

La vue

Dans la vue, c'est-à-dire dans le fichier `index.php`, nous constituons notre application par le biais de directives AngularJS :

- `<input name="salle" id="numero_salle" type="text" ng-model="room.nom" autocomplete="off" required>` : la directive `ng-model` va nous servir à trier dynamiquement les salles, et plus particulièrement les trier par leur nom que nous taperons dans cet input.
- `` : cette directive agit comme un eventListener JavaScript, car en cliquant sur cet élément HTML (il s'agit d'une icône dans ce contexte), le script triera les salles par type et n'affichera que les salles d'informatique.
- `<div class="col s12 offset-s3 stop_offset" ng-repeat="salle in values = (salles | filter: room | filter:roomFilter)">` : la directive `ng-repeat` agit à la manière d'une boucle `foreach` en JavaScript ou en PHP : elle permet de répéter le contenu de cette balise HTML autant de fois que nécessaire. Pour chaque salle nous appliquons deux filtres simultanément : la variable `roomFilter` pour filtrer les salles par type (informatique, électronique, banale) et la variable `room.nom` qui correspond au nom d'une salle que nous commencerons à taper dans l'input de recherche.

Le code qui est répété grâce à la vue `ng-repeat` est le code qui correspond aux éléments HTML permettant d'afficher une salle, mais il faut simplement retenir que lors de ce parcours des salles :

- `{{ salle.nom }}` correspond à la chaîne de caractères contenue dans l'attribut `nom` de l'objet de salle courant (ex: '0112'). De même `{{ salle.imprimante }}` correspond à un entier défini dans notre base de données (ex: 0 s'il n'y a pas d'imprimante dans cette salle, 1 s'il y en a une).

Rapport de projet de E3

- `{} salle.type | type {}` est une expression AngularJS caractérisée par le caractère 'pipe' ('|') qui permet d'utiliser des filtres. Cela signifie que nous formatons la donnée `salle.type` selon le filtre `type` que j'ai créé. Pour faire simple, le filtre `type` agit à la manière d'une fonction qui affichera 'Salle informatique' au lieu de simplement 'it'.
- Les filtres créés pour cette application AngularJS se trouvent dans `js/filters.js`, une documentation sur les filtres est accessible ici <https://docs.angularjs.org/guide/filter>.
- ``: la directive `ng-if` permet d'afficher le contenu de la balise en question si et seulement si la condition en valeur de `ng-if` est vraie au sens booléen.
- Dans la directive `ng-repeat="(key, val) in libres[$index]"`, `(key, val)` est une syntaxe JavaScript qui permet d'accéder aux clefs et aux valeurs du `$index`-ième objet du tableau `$scope.libres`. `$index` est une variable par défaut, un entier `int`, créé automatiquement à chaque `ng-repeat` afin d'accéder au numéro de l'itération en cours. Ici on accède donc au numéro de la salle (la clef) et à son statut d'occupation (la valeur).

La page recherche_avancee.php

Cette page permet de filtrer les salles en PHP par numéro d'épi, numéro d'étage, taille, présence d'une imprimante, d'un projecteur, type de tableau (craie, feutre, les deux en cochant ces deux choix).

Une fois les choix faits, un clic sur le bouton "Rechercher" déclenche un script AJAX qui permet, sans recharger la page, de soumettre notre sélection via une requête POST vers la même page, qui va ensuite :

- construire automatiquement l'URL `$url_api` permettant d'interroger notre API pour ADE avec uniquement les critères choisis,
- récupérer les données JSON issues de cette consultation de l'API avec un `json_decode(file_get_contents($url_api))`,
- puis afficher grâce à une animation jQuery la liste des salles que nous venons de recevoir sous forme de lien vers une fiche détaillée de cette salle.

La page fiche.php

Depuis la page `index.php` ou `recherche_avancee.php`, en cliquant sur une salle issue du filtre, ou bien depuis la page `index.php` en soumettant l'input de recherche avec le nom exact de la salle, nous pouvons nous rendre sur cette page. Dans ce dernier cas, si le nom de salle tapé ne correspond pas exactement à un nom de salle

Rapport de projet de E3

de notre base de données, un message d'erreur apparaît dans une fenêtre modale et l'utilisateur est redirigé vers la page `index.php` pour recommencer sa recherche.

Cette page prend un paramètre GET qui correspond à un nom de salle de notre BDD. Elle la consulte donc à la recherche de cette salle pour en extraire toutes les informations à son sujet (type, taille, projecteur, imprimante,...). En fonction du matériel disponible dans cette salle, les icônes correspondantes s'affichent.

La page propose aussi une image dont la source est une capture d'écran de l'emploi du temps de cette salle pour le jour même, récupérée grâce à notre API pour ADE. Cette image est surplombée d'un input qui, lorsqu'il est cliqué, affiche un calendrier. En choisissant une date, celle-ci est extraite dynamiquement en JavaScript, et l'attribut `src` de la balise `` de cet emploi du temps est lui aussi modifié avec cette nouvelle date. En effet, notre API permet d'afficher l'emploi du temps d'une salle pour un jour donné, si on précise la date en paramètre : <https://mvx2.esiee.fr/api/ade.php?func=dispoSalle&nom=0112&date=06/18/2015> par exemple.

La page `connexion.php`

Cette page permet de se connecter avec ses identifiants étudiants ESIEE Paris pour pouvoir se rendre sur les pages relatives à la vie étudiante (les notes, absences et appréciations d'Aurion). Elle consiste en un simple formulaire qui stocke les informations tapées (login et mot de passe sous forme d'un objet JSON) dans un cookie en PHP, suivi d'une redirection vers la page `aurion.php`.

À ce sujet, il est possible de se déconnecter (ce qui revient ici à supprimer le cookie de connexion dynamiquement, en jQuery donc, grâce à la librairie `$.JQuery.cookie`) grâce à un bouton de déconnexion dans la barre de navigation.

La page `aurion.php`

Cette page permet d'afficher les notes, les notes archivées, les absences, les absences archivées, les appréciations et les appréciations archivées d'un étudiant (ouf!). Elle prend un paramètre GET : `q` (pour `query`) pouvant valoir `notes`, `absences` ou `appreciations` pour afficher l'une ou l'autre de ces données. Elle peut aussi ne pas prendre de paramètres afin de suivre le comportement décrit ci-après.

Si le cookie de connexion est vide, une fenêtre modale comportant le formulaire de connexion s'ouvre afin d'inviter l'utilisateur à se connecter.

Elle propose initialement ces choix par le biais de boutons et d'un switch pour activer le mode archives. Le clic d'un bouton déclenche un script AJAX qui récupère un paramètre associé à ce bouton, qui le précède de la chaîne de caractères '`old_`' si le switch d'archives est activé, et qui effectue une requête auprès de l'API pour Aurion que j'ai conçu : <https://mvx2.esiee.fr/api/aurion.php>, avec le paramètre `func` correspondant au bouton cliqué, et les paramètres `login` et `pwd` récupérés depuis le cookie rempli à la connexion. Une animation jQuery permet de faire disparaître

Rapport de projet de E3

temporairement ces boutons et affiche un loader à la place afin de faire patienter le temps de la requête qui est assez longue.

Le script Python de l'API s'exécute donc avec ces paramètres. Si la connexion à Aurion échoue à cause d'une erreur de login ou de mot de passe, les boutons réapparaissent et une fenêtre modale comportant un message d'erreur et le formulaire de connexion s'ouvre afin d'inviter l'utilisateur à se reconnecter avec de nouveaux identifiants.

Si le script Python parvient à son terme, une balise volontairement laissée vide se charge alors avec les données à afficher grâce à la fonction `$.JQuery.load()`, qui fonctionne grâce au comportement de la page `aurion.php` si le paramètre GET `q` est renseigné (c'est-à-dire qu'en cliquant sur 'Notes', on va charger les résultats obtenus sur la page `aurion.php?q=notes`, par exemple). En cliquant ensuite à nouveau sur un bouton, la balise qui vient de se charger redevient vide grâce à la fonction `$.JQuery.empty()`, et le comportement décrit en AJAX se reproduit (boutons de choix cachés, loader pour patienter, utilisation de `$.JQuery.load()`, réapparition des boutons de choix).

Les modalités d'affichages variant d'une donnée à l'autre (les notes inférieures à E sont rouges, les absences excusées sont vertes, etc...), celles-ci sont stockées dans le dossier `includes`, dans les fichiers `notes.php`, `absences.php`, `appreciations.php`, et sont incluses dynamiquement dans le script AJAX selon le bouton cliqué.

Cela permet donc de consulter toutes ces informations sur la même page, et les paramètres GET permettent aux utilisateurs de ne pas subir la même contrainte qu'Aurion où l'on ne peut pas accéder aux notes (ou aux absences, ou autre) directement depuis une URL donnée.

La page `recherche_professeur.php`

La page `recherche_professeur.php` consiste tout d'abord en un `<input>` qui permet de taper le nom d'un enseignant. Les résultats sont affichés en temps réel en AJAX en-dessous de cet input au fur et à mesure qu'un utilisateur tape des caractères grâce à la gestion d'événements `onkeyup` en JavaScript. Ces résultats sont sélectionnés grâce à un algorithme de pertinence basé sur la distance mathématique de Levenshtein.

La distance de Levenshtein est définie par le manuel PHP (<http://fr.php.net/manual/en/function.levenshtein.php>) comme étant le nombre minimal de caractères qu'il faut remplacer, insérer ou supprimer dans la chaîne `$str1` pour la transformer en la chaîne `$str2`.

Le script appelé se trouve dans le fichier `scripts/php/levenshtein.php` et se charge d'interroger notre base de données d'enseignants avec la requête SQL '`SELECT nom FROM prof`'. Les noms de famille des enseignants sont stockés dans un tableau associatif PHP où les clefs sont les noms de famille et les valeurs sont des entiers.

Rapport de projet de E3

Ces entiers correspondent à la distance mathématique au sens de Levenshtein séparant le nom de famille en clef et la chaîne de caractères tapée dans la balise <input> du fichier recherche_professeur.php. Cette distance mathématique est calculée très simplement pour moi car elle existe déjà en PHP par le biais de la fonction Levenshtein(\$candidat,\$modele,\$cout1,\$cout2,\$cout3) où :

- \$candidat correspond à une chaîne de caractères (le nom de famille en base de données) que l'on souhaite comparer à la chaîne \$modele (chaîne tapée dans l'input),
- les poids sont des paramètres optionnels permettant d'ajuster l'algorithme de Levenshtein en affectant des coûts qui impacteront la distance de Levenshtein sur insertion, remplacement ou suppression de caractères.

Finalement, le tableau associatif est trié par ordre croissant des distances (une distance de 0 signifie que \$candidat et \$modele sont la même chaîne de caractères) et seuls les cinq premiers résultats sont conservés et communiqués à la page recherche_professeur.php pour y être affichés.

Lorsque qu'un nom de cette liste est cliqué sur cette page, la liste disparaît dans une animation JavaScript pour être remplacé en AJAX, sans recharge de page, par une fiche enseignant qui récapitule tout ce qu'il y a à savoir sur cette personne grâce aux informations de notre base de données : son nom, son numéro de bureau, son adresse email, une capture d'écran de son emploi du temps pour le jour même, et un calendrier intégré pour consulter son emploi du temps pour un jour donné (à la manière du calendrier de la fiche salle).

A noter enfin qu'on peut accéder directement à la fiche d'un enseignant à l'aide du paramètre en GET q valant le nom de cet enseignant tel qu'il est dans la base de données. On peut donc garder la fiche d'un enseignant en favoris dans le navigateur (ex: [https://mvx2.esiee.fr/recherche_professeur.php?q=HILAIRE Xavier](https://mvx2.esiee.fr/recherche_professeur.php?q=HILAIRE%20Xavier)).

LA VERSION ANDROID

Cette partie a été réalisée par Léo DUPONT

Nous allons maintenant décrire la partie Android, les technologies utilisées et les fonctionnalités implémentées dans cette version de notre produit.

Moyens techniques

L'application a été développée en Java et en XML avec l'IDE Android Studio. Pour tester l'application, j'ai utilisé trois appareils tout au long du développement :

- Un émulateur faisant tourner Android Lollipop 5.0 (version d'API 21 d'Android) qui était la dernière version du système d'exploitation disponible au public lors du lancement du projet.
- Un émulateur faisant tourner Android KitKat 4.4 (version d'API 19), équipant à lui seul 39.2% des Androphones en fonctionnement aujourd'hui.
- Mon téléphone personnel, tournant également sous Android KitKat afin de m'assurer du bon fonctionnement sur un appareil réel.

Johann KUHN et Mehdi HOUACINE ont également testé l'application sur leurs téléphones tournant chacun sous Lollipop.

Compatibilité

Nous avons choisi de développer une application compatible à partir de l'API 15 d'Android (qui correspond à la version Ice Cream Sandwich 4.0.3) jusqu'à l'API 21 (Lollipop 5.0). D'après les statistiques d'Android (disponibles ici : <https://developer.android.com/about/dashboards/index.html>), ce choix rend notre application compatible avec plus de 93% des appareils Android en fonctionnement aujourd'hui.

Cette décision n'a pas rendu le développement facile. En effet, tous les développeurs Android rencontrent des problèmes liés au support d'anciennes versions et nous n'avons pas échappé à la règle.

Concernant la langue, nous avons entièrement traduit l'application en anglais afin de la rendre accessible aux étudiants étrangers en séjour à ESIEE Paris.

La traduction a été très simple grâce à l'organisation d'Android en fichiers de ressources. En effet, le fichier `res/values/strings.xml` est chargé par défaut et contient toutes les chaînes de caractères utilisées dans l'application en anglais. Nous avons ajouté le fichier `res/values-fr/strings.xml` avec le même contenu traduit en français. Ainsi, Android choisira automatiquement quel fichier de ressources utiliser en fonction des paramètres de langue de l'appareil. L'anglais était la langue par défaut et le français réservé aux appareils en français.

Design

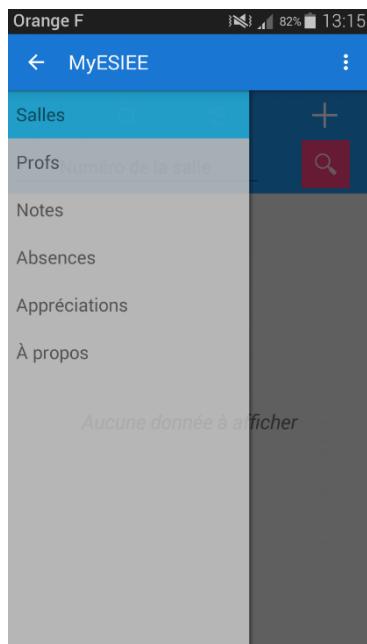
Concernant le design de l'application (et du site Web), nous avons choisi de suivre la charte graphique de Google nommée Material Design, très à la mode au sein des applications Android. Ceci afin de proposer un design moderne et dans lequel l'utilisateur est déjà familier.

Avec la dernière version d'Android Lollipop, des thèmes Material Design sont fournis dans le SDK. Cependant, ces thèmes ne sont pas disponibles pour les versions précédentes et il a fallu suivre des tutoriels sur internet pour créer notre propre thème Material Design, tel que celui-ci :

<http://d-codepages.com/index.php/android/android-beginners/30-create-material-design-for-older-versions-of-android>

Fonctionnalités

Navigation Drawer



Le Navigation Drawer est un menu coulissant à gauche de l'écran, présentant le menu. Celui-ci permet de naviguer entre les différentes sections de l'application : La recherche de salle, la recherche de professeur, les notes, les absences, les appréciations et l'à-propos.

Recherche de salles

L'écran principal de l'application est l'écran de recherche de salles car il s'agit du sujet initial de notre projet. Cet écran est généré par l'activité RechSalle dans le code.

Base de données locale

L'application dispose d'une BDD SQLite interne dans laquelle j'ai créé les tables `salle` et `prof` de notre BDD principale. Ainsi, lors d'une recherche de salle ou de professeur, les informations qui ne dépendent pas d'ADE sont lues directement dans la mémoire du téléphone, ce qui allège les requêtes HTTP et diminue la data utilisée, en particulier lorsque l'utilisateur est connecté en 3G par exemple.

Rapport de projet de E3

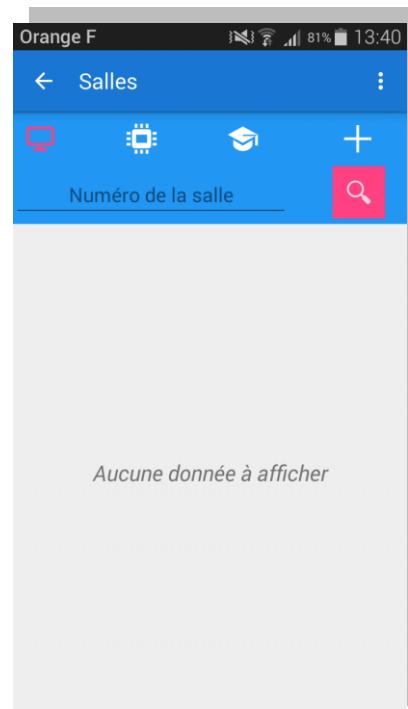
Lors du démarrage de l'activité RechSalle, l'application vérifie que le mobile est connecté à internet. Si c'est le cas, la version de la BDD locale est comparée à celle de la BDD du serveur grâce au script PHP bdd.php (voir la partie BASE DE DONNEES du rapport) ; la BDD locale est mise à jour grâce au script getData.php le cas échéant. Si le mobile n'est pas connecté à internet, un message Toast averti l'utilisateur et indique que la base de données n'est peut-être plus à jour.

Recherche

Il est possible de lancer une recherche sans spécifier de critères, il suffit alors de presser le bouton en forme de loupe. La requête HTTP envoyée au serveur est alors <https://mvx2.esiee.fr/api/ade.php?func=rechSalle> (voir la partie ADE).

Il existe ensuite 2 façons d'effectuer une recherche plus poussée :

Par type de salle. En effet, les 3 premiers boutons blancs en haut de l'écran correspondent respectivement aux salles informatiques, aux labos d'électronique et aux salles banalisées que nous avons appelées "salles de cours". L'appui sur un de ces boutons rend l'icône rose (la couleur d'accent de notre charte graphique) et sélectionne ce type de salle pour la prochaine recherche. Enfin, l'appui sur une autre icône remplace le type de salle sélectionné tandis que l'appui sur la même icône rose la désactive. La requête HTTP correspondant à ce cas de figure est celle-ci, dans le cas du choix "Salle informatique" : <https://mvx2.esiee.fr/api/ade.php?func=rechSalle&type=it>.



Par critères (peuvent être combinés au type de salle ou utilisés seuls). L'appui sur le bouton "+" ouvre une fenêtre de dialogue personnalisée nommée "Recherche avancée" et permet de personnaliser très finement sa recherche. Les critères pouvant être précisés sont :

- L'épi : salles hors épi, salles des épis 1 à 6.
- L'étage : salles des étages 0 à 4.
- Tableau blanc
- Tableau noir
- Projecteur
- Imprimante
- La taille : S, M ou L.

On peut également choisir d'afficher les salles occupées, celles-ci n'étant pas affichées dans les résultats par défaut.



Rapport de projet de E3

Un exemple de requête HTTP utilisant certains de ces critères : <https://mvx2.esiee.fr/api/ade.php?func=rechSalle&etage=3&tableau=2&projeteur=1&taille=m>.

Enfin, il est possible d'écrire le **nom** d'une salle en particulier dans le champ de texte "Numéro de la salle". Une liste des salles est proposée lorsque l'on commence à taper un numéro. Et le fait de lancer une recherche sur un nom de salle en particulier redirige directement vers la fiche de cette salle (cf. la partie suivante), à condition que cette salle existe.



Résultats

Concernant l'affichage des résultats, un composant `ListView` est rempli grâce à un `Adapter` et un `Layout` personnalisés. Chaque salle occupant une ligne de cette `ListView`, accompagnée d'icônes indiquant ses caractéristiques ainsi que d'une indication sur sa disponibilité.

La mention **Libre**, en vert, signifie que la salle est libre jusqu'à la fin de la journée. La mention **Occupée**, en rouge, signifie qu'elle actuellement occupée. Et la mention **XX min**, où XX est un entier, détermine le nombre de minutes pendant laquelle la salle est encore libre. Cette mention est verte si elle est supérieure à 30 minutes, orange sinon.

Le fait d'appuyer sur une salle dans les résultats démarre une nouvelle activité `FicheSalle` (cf. ci-dessous) avec un `Intent` (élément Android permettant aux activités et aux applications de communiquer entre elles) possédant un `extra` (ou un paramètre) contenant le numéro de la salle à afficher.

Salles			
	Numéro de la salle		
3207+	L	Libre	
3301	L	Libre	
3305	M	Occupée	
3307+	L	Libre	
3401V+	L	29 min	
3407	L	Libre	
4003	M	Libre	
4005	M	Libre	
4007	L	Libre	
4105V	L	Libre	
4109	M	Libre	
4201+	L	Libre	
4307+	L	Libre	

Fiche salle

L'activité `FicheSalle` est démarrée par un `Intent` avec un `extra` contenant le nom de la salle à afficher. Cette fiche rassemble les caractéristiques connues d'une salle (informations récupérées dans la BDD locale), un bouton ouvrant un `Date Picker` permettant de choisir une date, et une image de l'emploi du temps de cette salle issue d'ADE correspondant à la date choisie.

Rapport de projet de E3



La fiche de la salle 5309V++



Le Date Picker sous KitKat

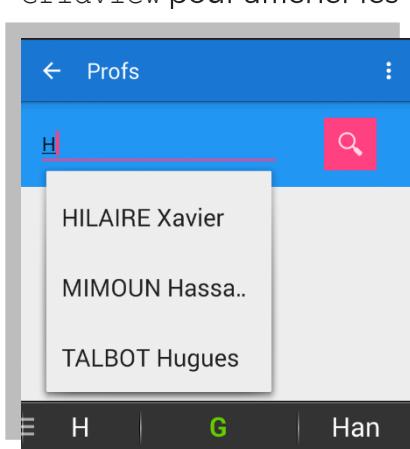


Le Date Picker sous Lollipop

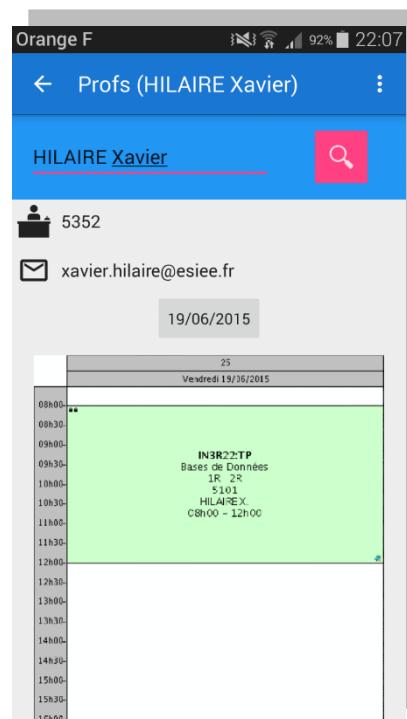
Recherche de professeur

L'activité RechProf ressemble à l'activité FicheSalle, elle contient la même GridView pour afficher les "caractéristiques" d'un professeur (le bureau et l'adresse e-mail), le même Date Picker et affiche aussi une image de l'emploi du temps du professeur sélectionné à la date choisie.

L'activité est également pourvue d'une barre de recherche avec prédition afin de trouver un professeur par son nom et/ou son prénom.



Lors d'un appui sur l'adresse e-mail d'un professeur, l'application génère un Intent destiné à l'application mail par défaut de l'appareil, avec comme adresse de destinataire celle du professeur (spécifiée par un extra attaché à l'Intent). De par le fonctionnement d'Android, si aucune application n'est enregistrée par défaut sur l'appareil, le système propose à l'utilisateur de choisir son application.



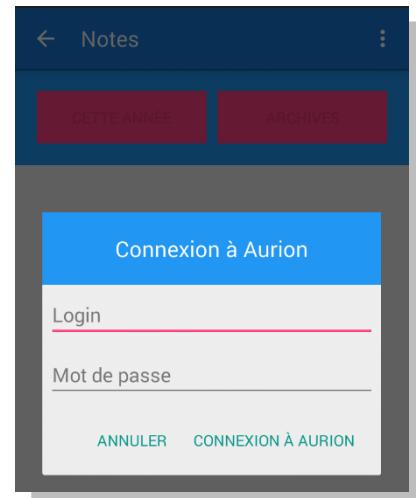
Notes, Absences, Appréciations

Ces trois activités, réunissant les informations récupérées sur Aurion grâce au script aurion.php, sont basées sur exactement le même modèle, la seule différence étant le Layout et l'Adapter liés à la ListView des résultats.

Connexion à Aurion

Lors du démarrage d'une de ces activités, si les identifiants Aurion de l'utilisateur ne sont pas enregistrés, une fenêtre de dialogue personnalisée apparaît afin qu'il puisse saisir ses identifiants. Ce n'est bien sûr pas obligatoire et les fonctionnalités non liées à Aurion peuvent fonctionner sans ces identifiants. Il est possible de modifier ou de supprimer ces informations grâce à un bouton dans les options des activités.

La validité de ces identifiants n'est pas vérifiée immédiatement car chaque requête vers Aurion est relativement lente compte tenu des moyens que nous avons dû utiliser pour concevoir notre API Aurion. Si une requête Aurion échoue à cause de mauvais identifiants, l'utilisateur est averti par un message au centre de l'écran.



Requêtes

Sur chaque activité, il existe un bouton "Cette année" et un bouton "Archives", permettant d'obtenir les informations de l'année en cours ou des précédentes. Les requêtes HTTP utilisées sont de cette forme (voir la partie Aurion) :

- URL : <https://mvx2.esiee.fr/api/aurion.php>
- Paramètres en POST :
 - o func => old_absences (pour les absences des années précédentes)
 - o login et pwd

Résultats

Chaque activité affiche les informations reçue d'une façon différente.

Pour les **notes**, on affiche le grade obtenu (vert de A à E et rouge pour F et Fx), l'intitulé de l'unité, son code en gras pour le discerner rapidement, et le nombre de crédits associés.

Pour les **absences**, on affiche le code et l'intitulé de l'unité en gras, le type de séance et l'intervenant, puis la date et l'heure de l'absence, accompagné du motif (rouge si non excusé, vert sinon).

Pour les **appréciations**, on affiche simplement le commentaire en gros, suivi de la période et de l'année en discret, par ordre chronologique inversé.

Rapport de projet de E3

Le cas où aucune donnée n'est à afficher a été pris en compte et un message avertira l'utilisateur.

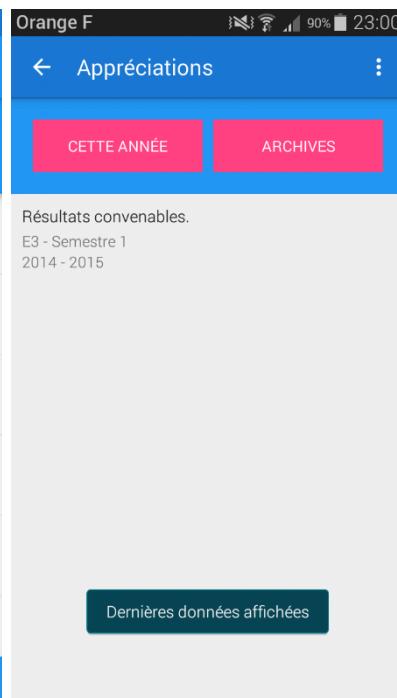
Les dernières données affichées sont enregistrées dans la mémoire du téléphone, ce qui permet de les réafficher immédiatement lorsque l'activité est relancée sans avoir à les recharger depuis internet.



Les notes (sous KitKat)



Les absences (sous Lollipop)



Les dernières appréciations affichées

Dernières données affichées

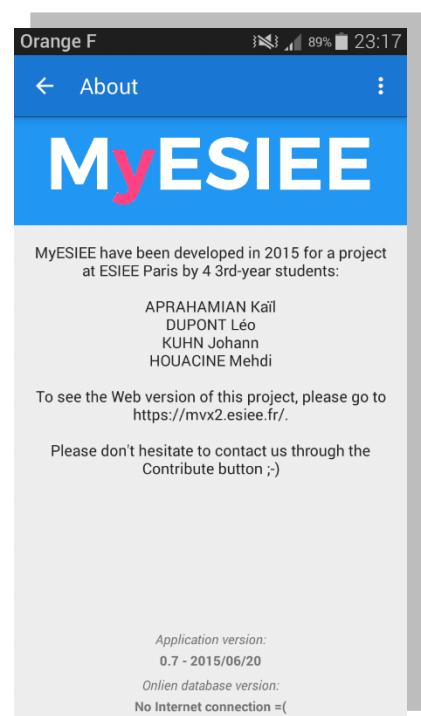
À propos



Enfin, j'ai ajouté une activité About, indiquant le contexte dans lequel cette application a été développée, qui a participé au projet et l'adresse de la version Web.

On peut également apercevoir la version de l'application, avec la date de dernière mise à jour de l'application et celle de la BDD principale (récupérée grâce au script bdd.php).

La présence d'une connexion internet a été prise en compte afin de ne pas afficher d'erreur ou de "vide". Un message signale que le mobile n'est pas connecté, et la version de la BDD est automatiquement récupérée quand la connexion revient.

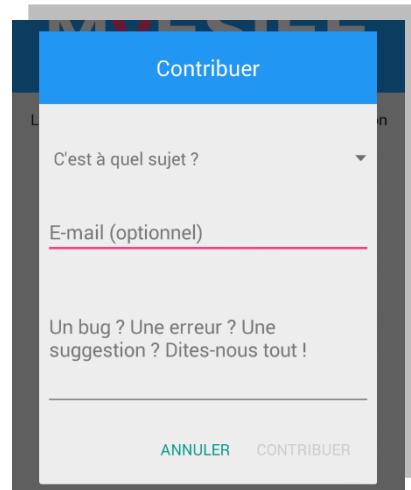


Contribuer

Dans toutes les activités de l'application, il est possible d'appuyer sur le bouton Contribuer dans les options (voir partie Maintenance). Ceci ouvre une fenêtre de dialogue personnalisée contenant trois champs :

- Le type de contribution (liste déroulante) : Bug, Erreur, Suggestion, Autre
- L'adresse e-mail (optionnelle)
- Le message de la contribution

Afin de ne pas perdre malencontreusement ce que l'utilisateur a écrit, le fait d'appuyer hors de la fenêtre de dialogue ne la fait pas disparaître. De même, le bouton d'envoi est inactif tant que les champs obligatoires (type et message) ne sont pas remplis et surtout tant que la connexion internet n'est pas active. L'utilisateur est averti par un message Toast qu'il doit être connecté pour contribuer, et dès que la connexion revient, le bouton s'active.



En plus de ce que l'utilisateur saisit, l'application récupère des informations pouvant être utiles en cas de debug :

- la présence d'un login Aurion enregistré,
- la version Android de l'appareil,
- le nom de l'activité depuis laquelle la contribution a été envoyée,
- le paramètre de langue de l'appareil.

La contribution est ensuite envoyée au script `contribution.php` qui se charge d'ajouter la date et de la stocker dans la base de données. Et un message Toast remercie l'utilisateur pour sa contribution.

Bilan d'expérience

N'ayant eu que très peu d'expériences en développement Android avant de commencer ce projet, j'ai eu beaucoup de mal en milieu de développement car j'ai pris quelques mauvaises décisions dès le début qui ne pouvaient pas être corrigées facilement. Par exemple je ne savais pas qu'il fallait utiliser des Fragments plutôt que des Activités lorsque l'on utilise un Navigation Drawer.

Le fait d'être confronté à beaucoup de problèmes et incompréhensions lors du développement m'a fait "boycotter" Android pendant environ 2 semaines. C'est-à-dire que je me suis beaucoup penché sur les scripts PHP d'ADE tout en évitant au maximum de travailler sur Android.

Lorsque le temps a commencé à manquer, je me suis forcé à retourner sur le développement de l'application "à temps plein" et je pense que cette pause m'a permis de prendre du recul et de repartir sur de bonnes bases car j'avais plus de facilités et de motivation.

Rapport de projet de E3

Je pense également que j'aurais dû me former davantage à Android avant de me lancer dans le développement d'une application aussi complète afin de l'aborder plus sereinement et d'éviter beaucoup d'erreurs. Aussi, si je devais de nouveau développer une application Android, je me renseignerais sur les bonnes pratiques avant d'utiliser un composant que je ne connais pas (comme un Navigation Drawer par exemple).

Concernant la documentation, les guides de Google assez complets (<http://developer.android.com/guide/index.html>) ont été d'une grande aide et le forum Stackoverflow.com, déjà très rempli sur le thème d'Android, d'un grand secours tout au long du développement.

CONTRIBUTION

Cette partie a été réalisée par Johann KUHN

Le site possède un système de contribution. Celui-ci permettra aux utilisateurs de signaler des bugs ou des erreurs, mais aussi de suggérer des améliorations pour le site et l'application mobile. Pour ce faire, on passe par un formulaire qui est inséré dans une fenêtre modale de MaterializeCss.

The screenshot shows a dark-themed web application. At the top, there's a navigation bar with links for Salles, Professeurs, Vie étudiante, Contribuer, and Connexion. The main content area has a dark blue header with the text "Contribuer". Below this, there's a form with fields for "Choisissez un type de contribution:", "Votre email", and a large text area labeled "Ecrivez !". At the bottom right of the form is a green "SOUMETTRE >" button. Below the form, there's a grid of three items, each with a small image, a number (1001, 1005, 1007+), some text, and the word "Libre".

Le formulaire contient un champ pour rentrer un **type de contribution** (obligatoire) : Amélioration, Erreur, Bug ou Autre, un champ pour rentrer son **email** (facultatif) et un champ pour écrire la **description** de la contribution (obligatoire).

Le formulaire contient aussi des champs cachés :

- la page dans laquelle il se trouve lorsqu'il écrit sa contribution,
- le user agent qui permet de connaître le navigateur utilisé et sa version.

Tous ces champs seront ensuite stockés dans la table `contribution` de la base de données grâce à un script PHP. J'ai aussi pensé aux éventuels problèmes de sécurité liés à un formulaire PHP (comme des injections SQL) mais il s'avère que les requêtes préparées (ce que j'utilise) rendent ces injections impossibles.

Récupérer le `user agent` n'a pas été simple. Normalement, il faut utiliser la fonction PHP `get_browser()` pour le récupérer mais celle-ci nécessite un fichier appelé `browsercap.ini` qu'il faut mettre à jour régulièrement sur le serveur. Pour pallier ce problème, j'ai trouvé un script sur Stackoverflow.com (<http://stackoverflow.com/a/8754134/2372933>) qui permet de récupérer le navigateur et sa version. Si le formulaire a bien été envoyé, un message `Toast` (de MaterializeCSS)

Rapport de projet de E3

est affiché sur le site pour confirmer que la contribution a bien été envoyée et enregistrée.

Les contributions, une fois envoyées, sont stockées dans la base de données au niveau de la table contribution. Pour traiter ces contributions, j'ai créé une page qui contient un tableau dynamique (grâce à un plugin jQuery appelé WaTable : <http://wootapa-watable.appspot.com/>) qui permet de mettre en place des filtres et des tris afin d'aiguiser, dans notre cas, la recherche d'une contribution particulière, comme par exemple, afficher toutes les contributions liée à Android, un navigateur spécifique, une certaine date, etc... Ce tableau est rempli grâce à un script JavaScript et un script PHP (pour récupérer les infos de la table) suivi d'un encodage en JSON. Une fois les contributions traitées, il suffira de modifier le statut de celles-ci dans le tableau.

idcontribution	login	type_contribution	date_contribution	navigateur	version_android	email	location	statut	contenu
1	dupontl	Suggestion	2015-06-19 00:32:00		API 19 (Android 4.4.2)		Profs (DOUAY Valérie) (LANG : français)	Nouveau	test first time
2	kuhnj	Bug	2015-06-19 00:32:07	Chrome (v. 43)			/wip/recherche_professeur.php	Nouveau	test tout marche V1
3	kuhnj	Bug	2015-06-19 00:33:52	Chrome (v. 43)			/wip/recherche_professeur.php	Nouveau	test tout marche v2
4		Autre	2015-06-19 00:35:49	Firefox (v. 38)			/wip/recherche_professeur.php	Nouveau	tout marche v3
5	kuhnj	Bug	2015-06-19 13:23:23					Nouveau	test header
6	#none	Autre	2015-06-19 14:40:03		API 19 (Android 4.4.2)	nn@nn.nn	5309V++ (LANG : français)	Nouveau	Louuuuuurd l'appli
7	houacinn	Amélioration	2015-06-20 11:12:39	Chrome (v. 43)		mehdi.houacine@edu.esiee.fr	/wip/aurion.php	Nouveau	Test Test Test
8	houacinn	Autre	2015-06-20 11:18:16	Chrome (v. 43)		mehdi.houacine@edu.esiee.fr	/aurion.php	Nouveau	Test Test Test
9		Bug	2015-06-20 15:37:58	Chrome (v. 43)			/fiche.php	Nouveau	C'est pas très beau la
10	dupontl	Autre	2015-06-20 21:42:28		API 19 (Android 4.4.2)	leo.dupont@edu.esiee.fr	Salles (LANG : français)	Nouveau	salut

Rows 1-10 of 24

« 1 2 3 4 5 » Rows ▲

CONCLUSION

Ces huit semaines de projet nous ont permis de réaliser presque tout ce que nous avions prévu d'intégrer à notre application.

	Prévu	Nouvelles idées
Fait	<ul style="list-style-type: none"> - Données Aurion - Disponibilité des salles en temps réel - Disponibilités d'un professeur - Filtrage des salles selon des critères spécifiques - Application Android - Site Web responsive 	<ul style="list-style-type: none"> - Système de contribution - Filtrage des salles par leur taille - Distance de Levenshtein pour tolérer les fautes d'orthographe en recherchant les enseignants par nom.
Pas fait	<ul style="list-style-type: none"> - Carte pour situer chaque salle - Notifications Aurion - Toilettes ouvertes - Menu cantine 	<ul style="list-style-type: none"> - Extraction des données des professeurs depuis la base LDAP - Filtrage sur l'affichage des notes

Avec un délai supplémentaire, nous aurions voulu réaliser une carte de l'école permettant de situer les salles. Cette carte aurait été accessible sur la page servant de fiche salle. Pour cela, nous aurions prévu, en base de données, un champ dans la table `salle` avec les coordonnées de chaque salle sur notre carte et nous aurions pu ainsi pointer la salle en question sur notre carte avec du CSS sur la page `fiche.php`. Il nous a toutefois manqué du temps pour construire cette carte sous un logiciel graphique et pour remplir notre base de données manuellement avec ces coordonnées pour lesquelles il n'y a, à priori, pas moyen de les générer automatiquement.

Nous aurions souhaité développer cette application pour iOS mais nous n'avions pas le matériel nécessaire pour cela. Nous avions prévu de nous initier au langage de programmation Swift d'Apple pour concevoir une telle application, mais il n'est utilisable que sous machine Apple type Mac.

Une fonctionnalité phare que nous n'avons pas menée à terme est un système permettant de notifier un étudiant lors de l'ajout d'une donnée sur Aurion (une nouvelle note, une nouvelle absence,...). Pour cela, notre script actuel d'extraction de données dynamiques aurait pu être exécuté régulièrement par le serveur grâce à une tâche CronJob mais cela aurait saturé l'utilisation du serveur qui a d'autres utilités pour l'école que notre projet. Un autre problème aurait été la question du stockage des identifiants des étudiants sur le serveur car nous ne pouvons pas les y entreposer en clair ou même chiffrés avec une clef connue alors qu'ils nous sont nécessaires pour extraire des données d'Aurion.

Une fonctionnalité accessoire que nous envisagions aussi en début de projet était une partie réservée au quotidien de l'école : une vue qui aurait affiché des informations sur des événements en cours à l'école (des événements sportifs, des événements organisés régulièrement par le service carrière ou l'infirmérie pour la

Rapport de projet de E3

prévention, ...) ou des informations utiles telles que le menu du jour à la cantine de l'école ou encore une liste des toilettes condamnées. Nous n'avons pas travaillé sur cette fonctionnalité car ces informations sont, en général, soit communiquées par mail par les services concernés de l'école.

Enfin, nous avons réussi à nous procurer des informations sur les enseignants, mais pas autant que nous le souhaitions. Nous disposons de leur(s) bureau(x) respectif(s) et de leur emploi du temps mais leur adresse email est générée par un script, ce qui rend cette information erronée pour les quelques enseignants qui n'utilisent jamais leur mail ESIEE. Donc nous nous retrouvons avec une table `prof`, au final, assez creuse. Nous avons découvert un peu tard, toutefois, que nous aurions pu avoir une base de données plus complète à l'aide de la base LDAP ESIEE Paris.

Aussi, même si cela n'est pas développé dans le rapport car nous l'avons développé en toute fin de projet, nous avons amélioré l'affichage des données sur la page `aurion.php`. Il est désormais possible d'appliquer des filtres de tri dynamique sur les données extraites. On peut ainsi trier les notes par résultats obtenus à l'unité et par nom d'unité dans l'ordre alphabétique à l'aide de boutons radio HTML, et aussi en tapant le nom d'unité dans une barre de recherche. De même pour les absences qui sont triables par temps d'absence et par unité. Ces deux vues affichent aussi le nombre total de crédits obtenus (en tenant compte des notes F et Fx qui n'apportent pas de crédits) et le nombre total d'heures d'absences.

Enfin, même si ce projet E3 est désormais fini, il s'agit d'une application que nous utiliserons à l'avenir et, si cela nous est possible, nous aimerais pouvoir continuer à la développer selon ces axes d'amélioration. C'est en ce sens d'ailleurs que nous avons conçu le module de contribution qui permet aux étudiants de nous signaler des erreurs à corriger sur notre application ou de proposer leurs idées.