

Aprahamian Kaïl

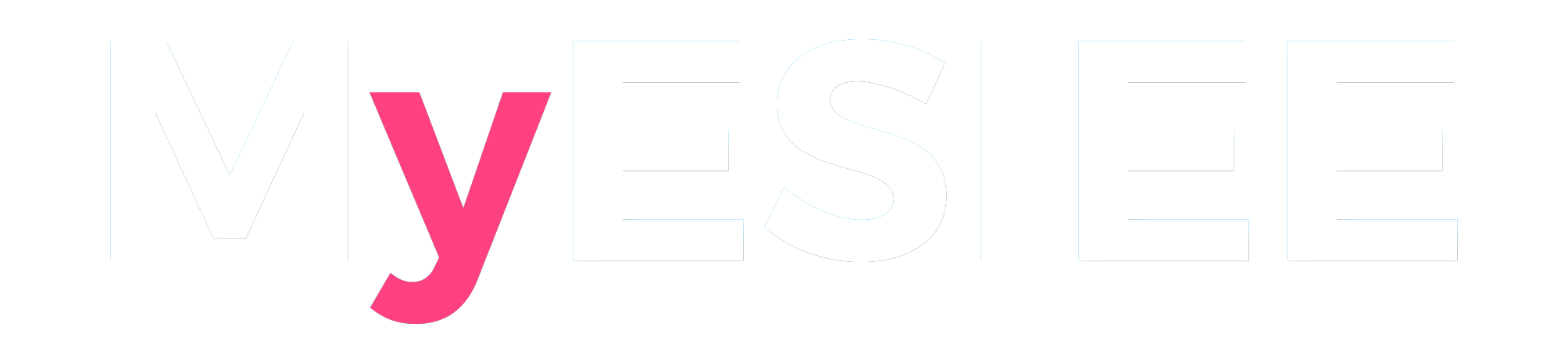
Dupont Léo

Kuhn Johann

Houacine Mehdi

Projet de troisième année

Mai – juin 2015



[Introduction 3](#_Toc422567837)

[Base de données 4](#_Toc422567838)

[Structure de la BDD 4](#_Toc422567839)

[Récupération des données et remplissage de la BDD 4](#_Toc422567840)

[Synchronisation de la base de données avec Android 5](#_Toc422567841)

[Mise à jour des données 5](#_Toc422567842)

[Synchronisation des données 5](#_Toc422567843)

[Version de la base de données 6](#_Toc422567844)

[Aurion 7](#_Toc422567845)

[ADE 8](#_Toc422567846)

[La Web API d’ADE 8](#_Toc422567847)

[Notre API personnalisée 8](#_Toc422567848)

[Utilisation de la fonction rechSalle 9](#_Toc422567849)

[Utilisation de la fonction dispoSalle 10](#_Toc422567850)

[Utilisation de la fonction dispoProf 10](#_Toc422567851)

[Conclusion 10](#_Toc422567852)

# Introduction

# Base de données

## Structure de la BDD

*Cette partie a été réalisée par Johann Kuhn*

La base de données est composée de quatre tables :

## Récupération des données et remplissage de la BDD

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.

Cette partie a été réalisée par Mehdi Houacine

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, 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 libraire 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 Excels cités plus tôt, leur nom provient d’une liste de noms extraites 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 corrigions.

## 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

# 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=   1. 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   1. Récupération du resourceID lié à la salle 5201V :   ?sessionId=14e0dae2bb6&function=getResources&name=5201V&detail=0   1. 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".   1. 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 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 "43" 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

Ci cool

# 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.

## 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

|  |  |
| --- | --- |
| C:\Users\Byakko\AppData\Local\Microsoft\Windows\INetCache\Content.Word\NavigationDrawer.png | 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.

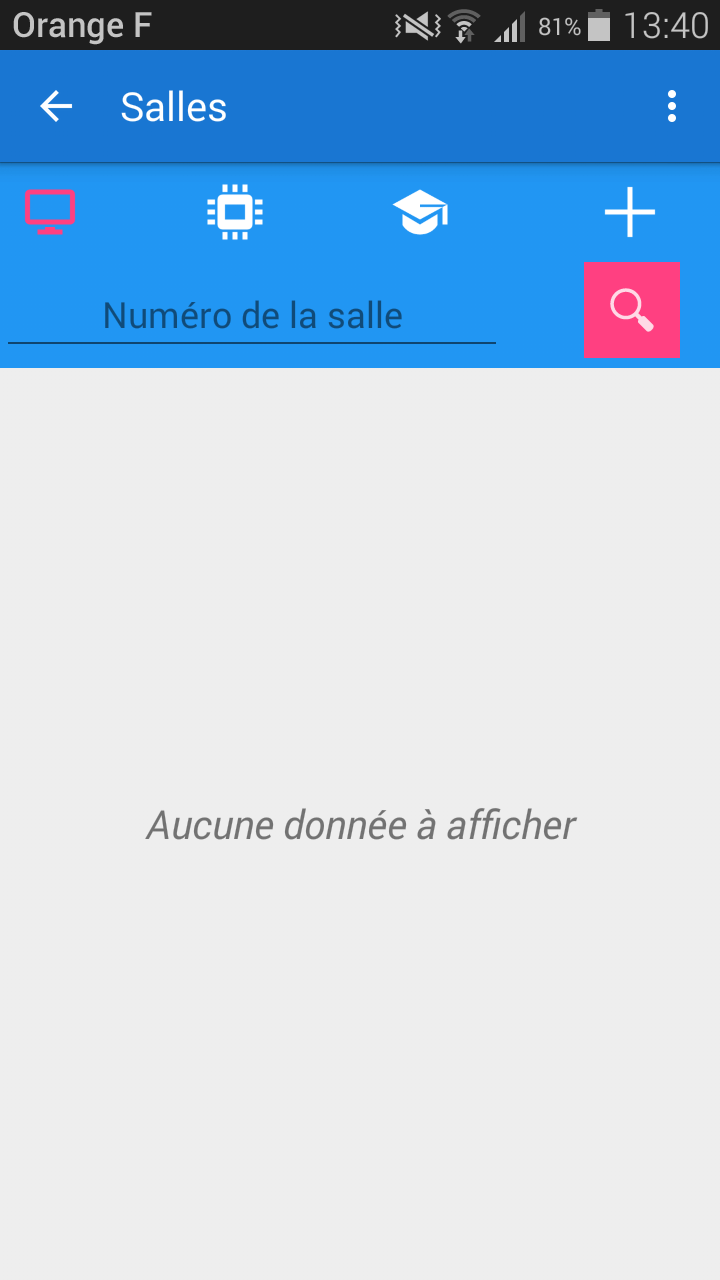
#### Base de données locale

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.

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 données 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

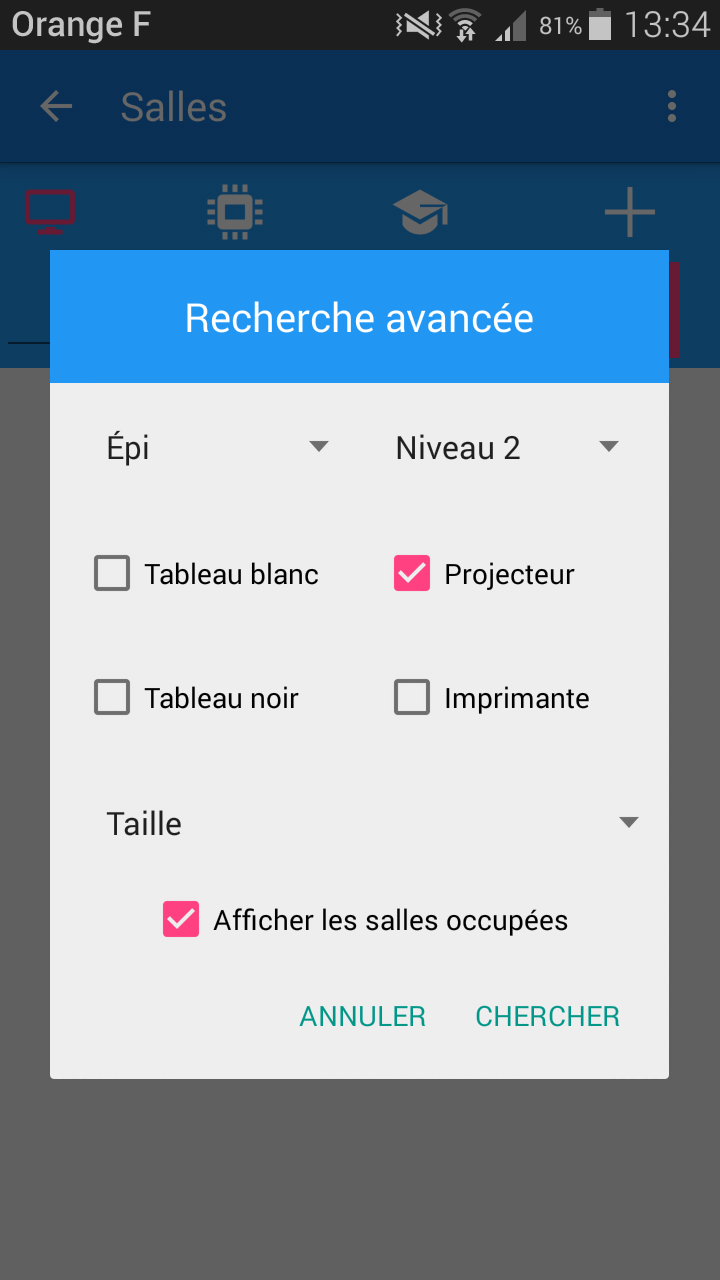
Il est possible de lancer une recherche **sans spécifier de critères**, il suffit alors de presser le bouton en forme de loupe. La requête HTTP envoyée au serveur est alors <https://mvx2.esiee.fr/api/ade.php?func=rechSalle> (voir la partie ADE).



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>.

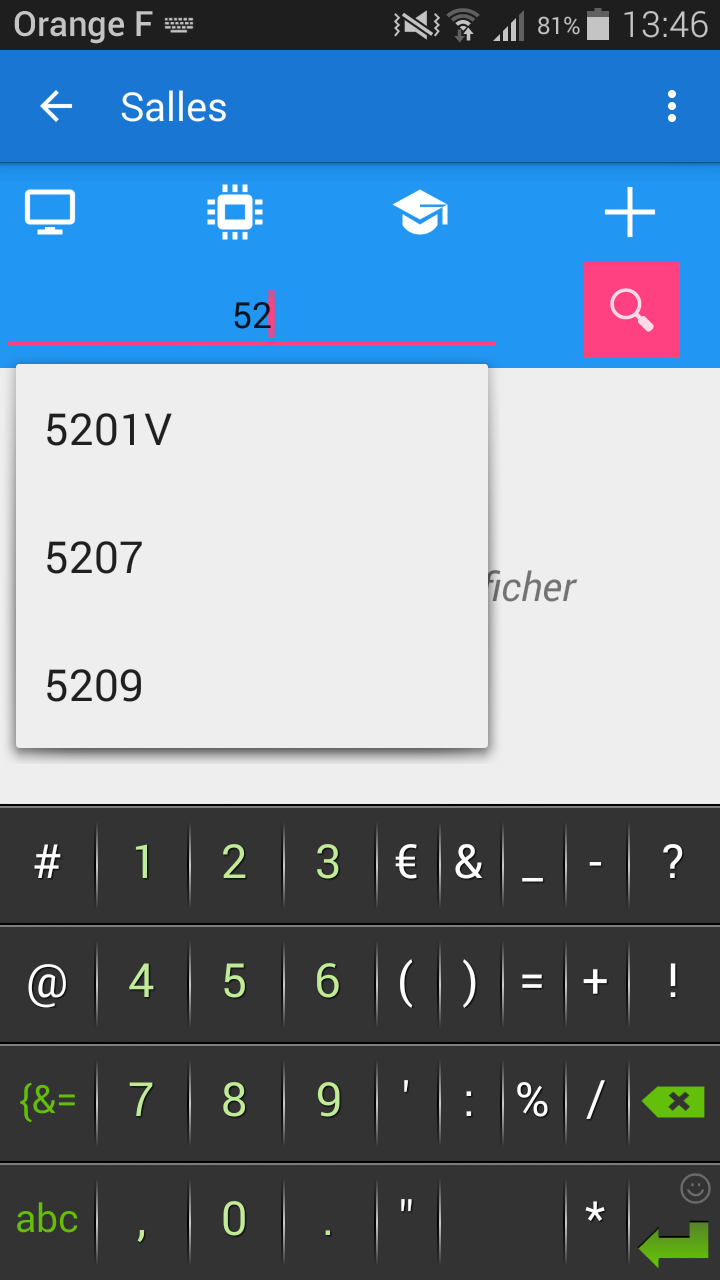
**Par critères** (peuvent être combinés au type de salle ou utilisés seuls). L’appui sur le bouton "+" ouvre une fenêtre de dialogue personnalisée nommée “Recherche avancée” et permet de personnaliser très finement sa recherche. Les critères pouvant être précisés sont :



* 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.

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