Projet Easy Course
HMIN205 - Rapport de projet Android

Joël BEYA NTUMBA Christophe QUENETTE

25 juillet 2019

Année universitaire 2018/2019

Sommaire

1 Présentation du projet

Le but de ce projet de développement d'application Android est de créer de façon concrète une application fonctionnant sur des appareil ayant comme système d'exploitation Android (i.e. des smartphones donc mais aussi des tablettes, ...).

Nous avions le choix entre trois sujets de projet. Nous avons décider d'opter pour le projet dit "MOOC" : la création d'une application Android proposant des cours/formations en ligne. Nous avons nommé notre projet `Easy Course`. C'est, dans l'idée, une application spécialisée qui permet de prendre des cours sur l'informatique, et cela à n'importe quel niveau d'étude, du collège au Bac + 5.

Le cahier des charges pour ce projet nous indiquait d'utiliser certaines technologies dont `Material Design`, les `Fragment`s et les préférences entre autres.

1.1 Technologies utilisées

Nous avons développé notre application en utilisant l'IDE Android Studio et nous avons stocké le code du projet sur gitlab (`https://gitlab.info-ufr.univ-montp2.fr/e20140034671/easy-course`, nous vous y avons donné l'accès).

De plus pour rendre notre application disponible sur internet et pas seulement en local, nous avons utilisé le service serveo, qui nous permet d'associer à l'adresse locale d'un ordinateur (par exemple `http://127.0.0.1:8080`) une URL publique, accessible depuis n'importe où (`https://easycourse.serveo.net`).

FIGURE 1 – Serveo (`http://serveo.net`)

FIGURE 2 – Node.js (`https://nodejs.org`)

FIGURE 3 – MongoDB (`https://www.mongodb.com/`)

Après avoir longuement hésité, nous avons décidé de créer un serveur avec Node.js et une base de données NoSQL avec MongoDB. Nous avons comparé ces deux technologies avec PHP/MySQL et nous les avons choisies pour trois raisons :

— Nous avons pu apprendre, au cours du premier semestre, à utiliser de façon basique Node.js et MongoDB pour créer une application Web

— Il est possible de créer, de façon simple, une API REST avec `Node.js` en utilisant le framework `Express`. Cela nous permet de réaliser, avec d'autres technologies que celle que nous avons utilisées lors du cours `HMIN210`, un service Web REST.

— L'API REST est facilement exploitable si l'on souhaite par la suite créer une application Web associée à l'application Android, les deux sortes de client ayant accès au mêmes informations, l'utilisateur pourra facilement passer de l'application sur téléphone à un périphérique utilisant un navigateur.

2 Fonctionnalités implémentées

Dans le cahier des charges, il était indiqué d'utiliser entre Matérial Design, Fragments, les services, les capteurs,...

De ce cahier des charges nous avons implémenté les fonctionnalités que nous détaillons ci-dessous :

— **Lancement de l'application :** Nous avons créé un **Wizard**, c'est à dire une présentation des fonctionnalités mises à disposition de l'utilisateur dans l'application. Nous y avons expliqué en utilisant un `ViewPager` le but de l'application et nous avons ajouté une interface de connexion.

— **Apparence générale de l'application :** Nous avons utilisé **Matérial Design** qui est un ensemble de règles de design proposées par Google et qui s'appliquent à l'interface graphique des logiciels et applications. Ceci nous a conduit à basculer la structure du projet sur **Android X** pour pallier la compatibilité. Nous avons obtenu grâce à cette technologie un design clair et moderne.

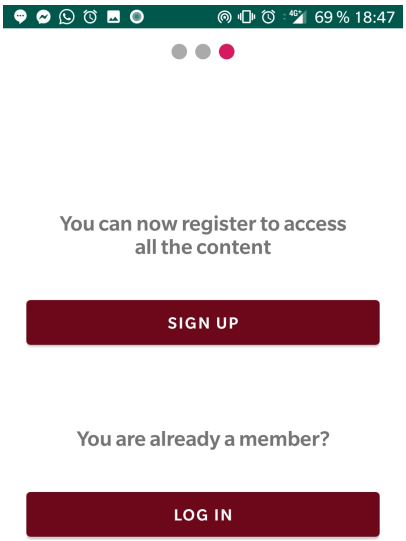

FIGURE 4 – Connexion ou inscription dans le Wizard

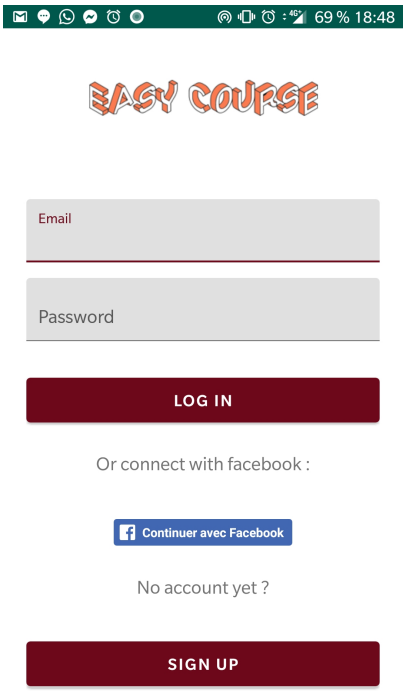

FIGURE 5 – Interface de connexion

— **Utilisation des fragments :** Nous avons défini les deux interfaces d'inscription pour Parent et Elève en `Fragments` de telle sorte que selon que l'on dispose d'un écran large ou pas, on puisse avoir les deux interfaces dynamiquement adaptables. On les a aussi utilisés pour les cours, selon qu'on ait selectionné un cours X ou un cours Y, on a la possibilité de voir les contenus des cours directement adaptés aux dimensions de l'écran.

— **Utilisation des Recycler, Card, Grid, Search... View :** Nous avons trouvé pour chacun de ces gabarits un cas d'utilisation ergonomique mettant en symbiose les différentes formes d'utilisation de ceux ci pour l'aboutissement du projet.

— **Connexion via Facebook :** Nous avons aussi ajouté comme fonctionnalité la connexion avec Facebook. Cela nous permet d'accéder au profil public de l'utilisateur et de pouvoir utiliser des données telles que son nom, son prénom, son email et sa photo de profil entre autres. Ainsi, l'utilisateur peut soit renseigner ses coordonnées à la main, soit utiliser son compte facebook directement.

— **Vérification de compte par envoi d'email :** Nous avons intégré un système de vérification de compte avant connexion en envoyant un email de confirmation à l'adresse avec laquelle l'utilisateur souhaite s'inscrire. Nous avons pour cela créé une adresse de messagerie GMail et l'envoi des emails se fait en `SMTP` avec le module `nodemailer` de `Node.js`. Tant que l'utilisateur n'a pas vérifié son email, ne peut pas accéder au contenu de l'application .

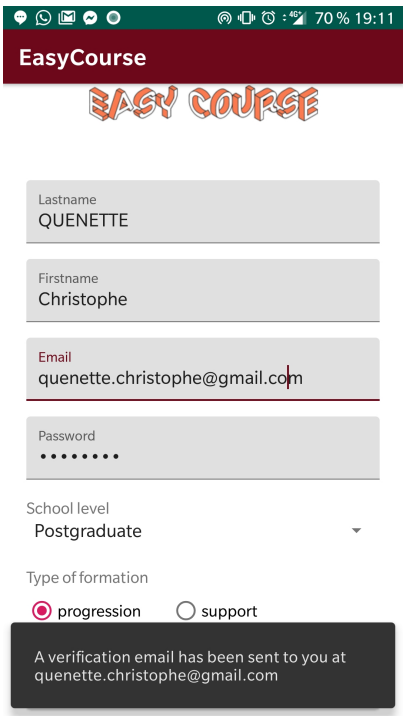

FIGURE 6 – Inscription d'un utilisateur...

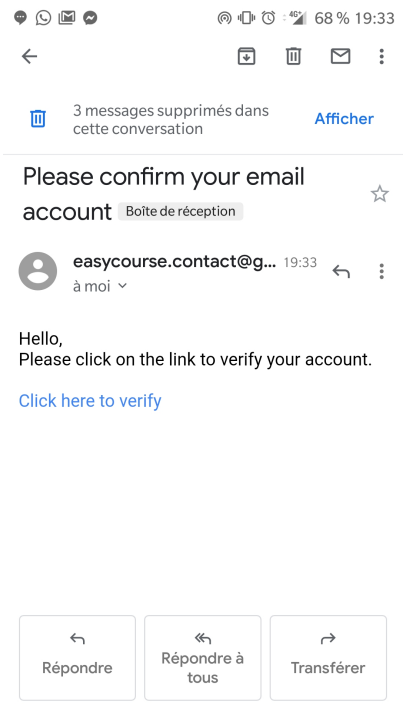

FIGURE 7 – ... reception de l'email de vérification ...

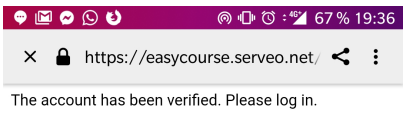

FIGURE 8 – ... et confirmation.

— **Internationalisation :** Nous avons internationalisant les ressources string notre application. Nous avons pris en compte deux langues : l'anglais et le français. Selon que l'utilisateur aura son smartphone en anglais ou en français l'application s'adaptera automatiquement à sa langue.

De plus nous avons crée tous les composants du seveur (routes, controlleurs, modèles) associés à l'application.

3 Difficultés rencontrées

Nous avons eu un certain nombre de contraintes à gérer et nous pouvons les classer en tant que difficultés liées au serveur ou liées à Android.

3.1 Difficultés liées au serveur

3.1.1 Stockage de données avec MongoDB

L'utilisation de `MongoDB` à la place de SQLite a amené le fait de découvir comment implémenter certaines fonctionnalités avec avec ce SGBD. En effet étant donnée qu'il s'agit d'une base de données NoSQL, les relations entre les élements, comme par exemple le lien d'héritage entre un utilisateur général et les élèves et parents qui en sont des objets fils, doivent être implémentées de facon différentes que si l'on utilisait un SGBD SQL "classique" comme MySQL.

Nous avons pu y trouver une solution en utlisant `Mongoose` (`https://mongoosejs.com/`), un *Object Data Manager*, qui nous permet donc de créer des entités objets pour représenter les cours, les parents, etc. avec `MongoDB` ; et plus particulièrement les *discriminators* de `Mongoose` qui permettent de définir des relations d'inclusion entre les différents objets.

3.1.2 Accès à notre back-end avec une URL publique

Nous avons commencé par développer notre application sur l'émulateur et nous nous sommes rendus comptes du manque de praticité de ce dernier lors de l'exécution de tâches simples par exemple lors que l'envoi de l'email (cela demandait d'avoir un compte email associé sur l'émulateur) ou encore lors de la connexion avec Facebook (là encore il fallait que l'utilisateur théorique s'identifie sur Facebook alors qu'en pratique facebook récolte directement les informations de l'utlisateur à partir de l'application Facebook).

Ainsi nous avons aussi lancé notre application sur un téléphone portable physique. Il fallait donc avoir un serveur externe sur lequel ont pouvait accéder aux données. Nous avons cherché plusieurs solutions en local qui n'ont pas réellement fonctionné, le `localhost` ne pas semblant fonctionnant sur les périphériques physiques.

Nous nous sommes donc tournés vers d'autres services. Nous avons trouvé premièrement `ngrok` (`https://ngrok.com/`). Ngrok possède les mêmes fonctionnalités que `serveo` (notre solution finale dont nous avons expliqué le fonctionnement précedemment) à la différence que ce service ne permet pas d'obtenir d'URL fixe (du moins pas dans sa version gratuite). Il était donc pratique pour

effectuer des tests mais pas durable pour la suite (il nous fallait constamment changer l'URL du serveur qui était stockée dans la configuration de l'application Android).

3.2 Difficultés liées à Android

Côté Java, il était aussi nécessaire de communiquer avec le serveur, ce dernier envoyant des objets en JSON. Nous avons donc dû utliser Retrofit. Il s'agit d'un client HTTP pour Android et Java et qui permet de transformer une API HTTP en une interface Java. La mise en concordnace des types de la base de données et des modèles créés dans l'application Android a parfois été fastidieuse et source de nombreuses erreurs.

4 Conclusion et perspectives

La mise en place du serveur a été l'étape qui au final nous a pris le plus de temps lors de la réalisation de ce projet. En effet, les vues xmls et la logique Java a été longue a implémenter mais pas particulièrement compliquée. Nous avons réussi terminer sa conception, ce qui nous a laissé un peu moins de temps pour le reste de la partie Android.

Aussi nous avons eu à gérer notre temps de façon critique avec nos autres travaux universitaires (dépot de projets pendant les cours, TER, examens...), ainsi il est fort problable qu'avec une meilleure organisation de notre part nous aurions pu implémenter plus de fonctionnalités.

Nous pourrions par la suite améliorer les points suivants :

— Effectuer une confirmation de mot de passe et un renvoi d'un email de confirmation si le token d'inscription a expiré
— Mettre en places les différentes vues manquantes en implémenter un service de gestion de l'avancée des utilisateurs
— Faire un Wizard plus complet avec version de test de l'application
— Créer un système de reconnaissance de l'utilisateur pour que ses données soit sauvegardées de facon durable entre différentes connexions.
— ...

Lien d'une démo de l'application via Google Drive :

`https://drive.google.com/open?id=1-7K0i1Vuw0sw`$_N yMWla - w - sds7UJJ9a1$