Université Assane SECK de Ziguinchor

Unité de Formation et de Recherche des Sciences et Technologies

Département d'Informatique

Développement d'un cas d'application

L3 – 2I

Mars 2023

©Papa Alioune CISSE

Papa-alioune.cisse@univ-zig.sn

Résumé :

1 - Enoncé du cas d'application (un mini-projet)

Ce mini-projet consiste à développer un embryon d'application mobile qui va permettre aux étudiants de l'UASZ de visualiser leurs emplois du temps. En effet, à partir de la L3, les emplois du temps sont parfois sujets à de nombreux changements au cours de l'année. L'idée est d'avoir une application Android qui donne l'emploi du temps et notifie l'utilisateur si des changements sont survenus.

Nous supposons pour cela que l'UFR ST dispose déjà d'une **application de gestion des emplois du temps (AGET)** qui est constituée d'une interface graphique utilisateur permettant de gérer (créer, modifier, supprimer) les emplois du temps ; d'une base de données pour stocker ces informations et d'un service web permettant à des applications tierces (comme celle que nous proposons de développer) de cueillir ces informations.

2 - Fonctionnalités de l'application Android

2.1 - Connexion à l'application

A son lancement, l'application présente une page d'accueil pour permettre à un utilisateur inscrit de se connecter sur la plateforme. Cette page doit comporter un logo (celui de l'UASZ) ; un message d'accueil ; deux champs de login (email) et de mot de passe et un bouton pour se connecter ; et un lien pour se rediriger vers la page de création de compte.

2.2 - Création de compte

Il s'agit d'un écran qui permet de créer un compte utilisateur dans notre application. Il comporte entre autres, les champs suivants : Prénom, Nom, Référence (numéro d'identification de l'étudiant UASZ), Email, Mot de passe, classe (L3 Info par exemple).

2.3 - Visualiser l'emploi du temps d'une classe

Quand un utilisateur se connecte sur l'écran d'accueil, le système doit vérifier et récupérer auprès de l'application AGET, l'emploi du temps de l'utilisateur. Celui-ci est ensuite redirigé vers l'écran d'affichage de son emploi du temps. Cet écran affiche l'emploi du temps sous forme de tableau. Chaque cellule de ce tableau correspond à une heure de cours. S'il y'a cours à cette heure, la cellule est coloriée en jaune et rempli avec le code du cours. Au clic sur la cellule, les détails du cours sont affichés en bas du tableau de l'emploi du temps.

2.4 - Cueillir les emplois du temps

Notre application doit pouvoir interroger le module **AGET** pour recueillir les dernières informations sur les emplois du temps. Pour cela, nous proposons de développer un web service qui, à chaque connexion de l'application à internet, vérifie s'il y'a des changements sur les emplois et les récupérer au cas échéant.

Figure 1. Pages de connexion / de création de comptes / de visualiser d'emplois du temps

3 - PREMIERS PAS DANS LA RÉSOLUTION DU PROBLÈME

3.1 - Modélisation du problème

Un **Cours** est défini par : une *matière* (définie par un code – Dev Mob par exemple – et une intitulée – Développement Mobile par exemple), une *classe* (Ex : L3 Info) de type String, d'un *enseignant* (Ex : Monsieur CISSE) de type String, d'une *salle de classe* (Ex : Salle C003) de type String, d'un *jour* (Ex : Lundi) de type String, d'une *heure de début* (Ex : 8 pour dire 8H) de type int et d'une *heure de fin* (Ex : 10 pour dire 10H).

Un **Emploi du temps** n'est ici rien d'autre qu'un ensemble de **Cours** donnant une répartition des différents enseignements dans le temps et dans l'espace (les salles de classes). Attention : il est important de signaler qu'un emploi du temps ici n'est pas le tableau graphique qui donne une représentation visuelle

Un **Tableau d'Emploi du Temps** est un tableau constitué d'un ensemble de colonnes (les différents jours de cours : Lundi à Samedi) ; d'un ensemble de lignes (les différentes heures de cours : 8h à 13h et 15h à 20h) et d'un ensemble de **Cellules** ou cases correspondant chacune à un **Cours** d'une classe donnée.

Une **Cellule** d'un **Tableau d'Emploi du Temps** est donc un objet qui se réfère (qui porte la référence) à un cours donné.

3.2 - Aperçu sur l'architecture MVC

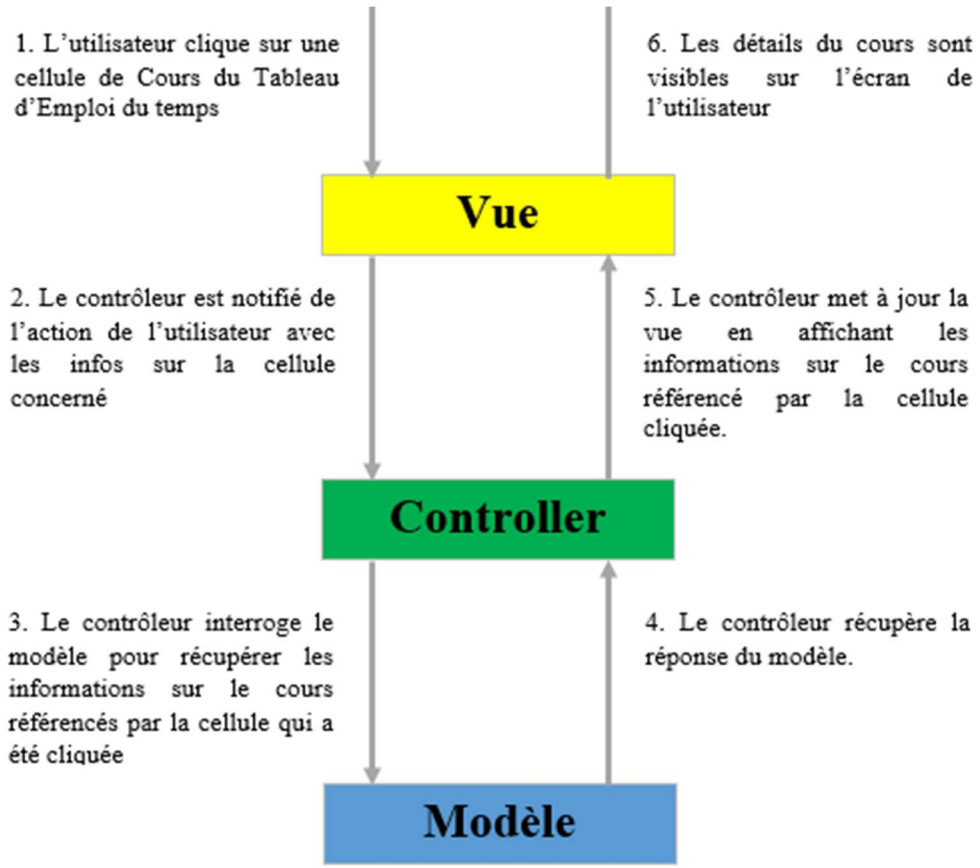
Dans le souci de faciliter le développement logiciel et la compréhension des codes, il existe plusieurs conventions ou des méthodes de bonne pratique. Parmi ces conventions, il y'a l'architecture MVC pour Model-View-Controller (ou Modèle-Vue-Controller en français).

L'architecture MVC consiste à découper son code pour qu'il appartienne à l'une des trois composantes du MVC. Lorsque vous créez une nouvelle classe ou un nouveau fichier, vous devez donc savoir à quelle composante il appartient :

- **Modèle** : contient les données de l'application et la logique métier. Par exemple, les **Cours** et les **Emplois du temps** constituent, pour le problème que nous posons ici, le « métier » de l'application. La composante modèle n'a aucune connaissance de l'interface graphique. Dans notre application, elle regroupera l'ensemble des données sur les cours et les emplois du temps. Nous créons un package nommé « model » pour cela et y mettre tout le modèle (les différentes classes).
- **Vue** : contient tout ce qui est visible à l'écran et qui propose une interaction avec l'utilisateur. Par exemple, les boutons, les images, les zones de saisie, etc. Dans notre application, cette composante est définie par le dossier ***layout*** (qui contiendra les fichiers xml correspondant aux interfaces des activités).
- **Contrôleur** : c'est la "colle" entre la vue et le modèle, qui gère également la logique de l'application. Le contrôleur permet de réagir aux interactions de l'utilisateur et de lui présenter les données qu'il demande. Et ces données, où les récupère-t-il ? Dans le modèle bien entendu ! Dans notre application, cela correspond aux différentes activités que nous aurons à développer). Nous pourrions ajouter un package nommé « ***controller*** » et y mettre toutes les activités de l'application.

3.3 - Exemple d'application du modèle MVC

Les écrans des téléphones étant petits, il nous sera difficile de mettre toutes les informations de tous les **Cours** dans les **Cellules** du **Tableau d'emploi du temps**. Nous allons donc procéder ainsi : une **Cellule** de **Cours** sera coloriée en jaune avec seulement le code de la matière concernée. On permet ainsi à l'utilisateur de cliquer sur une **Cellule** et de visualiser les détails du **Cours** en bas de du **Tableau d'emploi du temps**. Avec MVC, ce processus est matérialisé dans l'image suivant :



4 - DÉVELOPPEMENT DU CAS D'APPLICATION

4.1 - Création du projet et mise en place de l'architecture MVC

Nous allons créer un « projet à Empty Activity » appelé « ***Cours UASZ*** » avec comme **Package name** « ***com.uasz.pacisse.coursuasz »*** (remplacer ***pacisse*** par votre nom par exemple).

Pour mettre en place l'architecture MVC dans notre projet, nous allons juste ajouter les packages suivants : ***model*** (pour contenir le modèle de l'application, c'est-à-dire les différentes classes de notre application) ; ***controler*** (pour contenir les différents contrôleurs, c'est-à-dire

les différentes activités) et normalement le package ***view*** (sauf que nous n'avons pas besoin de l'ajouter, puisque le répertoire ***res/layout*** joue déjà le ce rôle).

Dans AS, l'ajout d'un package se réalise en faisant clic droit sur le répertoire parent (ici le package com.uasz.pacisse.coursuasz), puis **New -> Package.** Ajouter le mot « model » à la fin de la chaine dans la petite fenêtre qui s'ouvre et tapez la touche « enter » pour valider. Répéter le même procédé pour ajouter le sous package « ***controler*** » au même niveau que « ***model*** ».

La création du projet s'est accompagnée de la création d'une activité principale (**MainActivity**) et de son interface associée (**activity_main**). Nous allons considérer **MainActivity** comme étant l'écran d'accueil qui permet à un utilisateur de se connecter à l'application.

Pour respecter l'architecture MVC, nous devons déplacer **MainActivity** dans le package ***controler***. Pour cela, il faut faire un ***clic-maintient-déposer***, puis ***Refractor***.

Le modèle de l'application (c'est-à-dire les différentes classes métier de l'application, du répertoire **model**) est constitué des classes suivantes :

- **Cours** pour représenter un cours selon la définition donnée plus haut
- **EmploiDuTemps** pour représenter un emploi du temps en tant qu'un ensemble de **Cours**
- **CelluleTableauEmploiDuTemps** pour représenter une cellule d'un tableau qui représente un emploi du temps
- **TableauEmploiDuTemps** pour représenter un tableau constitué d'un ensemble de **Cellule** pointant chacune sur un **Cours** de l'**EmploiDuTemps**
- **Etudiant** pour représenter un étudiant selon la définition donné plus haut
- **Classe** pour représenter la classe d'un **Etudiant**.
- Etc.

Les implémentations de ces différentes classes seront vues en TP.

4.2 - Développement de la page de connexion

Nous supposons que la page de connexion correspond à l'activité « MainActivity ».

Dessin de l'interface utilisateur de l'activité « MainActivity »

Il faut se rappeler que dans Android, chaque page (écran) d'une application est un couple Activité/Layout. Le Layout est un fichier XML (fichiers stockés dans le répertoire « res/layout » d'un projet) que l'activité va charger après avoir été instanciée et qui permet de présenter des éléments graphiques et des éléments de contrôle ou widgets à l'utilisateur afin de lui permettre

d'interagir avec l'activité. Ces widgets peuvent être des boutons, des zones de saisie ou des menus déroulants, par exemple. Par convention, le nom du layout associé à l'activité « MainActivity » est « activity_main.xml » (tout en minuscule, avec le mot « Activity » qui vient avant le reste qui le précédait dans le nom de l'activité, séparés cette fois ci par « _ »).

~~Pour l'édition du layout « activity_main.xml », voire d'abord le chapitre XXX et codage en TP.~~

Actions préalables à faire : personnalisation du thème de l'application (fichier /values/themes) ; ajout d'images comme le logo de l'uasz directement dans le répertoire res/drawable ; ajout du fichier « dimensions.xml » pour les dimensions personnalisées (clic droit sur res, puis New → XML → Values XML File) ; ajout des shape files (clic droit sur drawable, puis New → Drawable Ressource File) ; *partager* le contenu du fichier « strings.xml).

A voir sur internet : Material Design (https://m2.material.io/components?platform=android) pour les composants à utiliser ; enlever ActionBar dans ActivityMain : ajouter d'abord l'option dans le thème (fait dans themes.xml), l'appliquer ensuite dans le manifeste (partie concernant ActivityMain et les autres activity concernés : android:theme="@style/Theme.CoursUASZ.NoActionBar") ; personnaliser l'icone de lancement (https://developer.android.com/studio/write/image-asset-studio).

Référencer les éléments graphiques de l'interface dans l'activité ActivityMain

Vous remarquerez que la classe ***ActivityMain*** possède pour le moment une seule méthode « ***onCreate*** » qui est appelée lorsque l'activité est créée. Vous verrez aussi que dans cette méthode, la dernière instruction (***setContentView***) permet de déterminer la vue (layout) à afficher.

Nous avons dans activity_main.xml quatre éléments graphiques avec lesquels l'utilisateur peut interagir et qui possède chacun un identifiant : les champs de mail et de mot de passe, le bouton de validation et le lien d'inscription. Il faut donc les référencer dans ActivityMain. Pour cela, il faut d'abord déclarer les variables qui les représentent dans le code :

```
private EditText mEmailConnexionInput;
private EditText mMotDePassConnexionInput;
private Button mConnectionButton;
private TextView mInscriptionLink;
```

Et ensuite les « brancher » avec les widgets du layout en utilisant la méthode « ***findViewById()*** » dans la méthode « onCreate) :

```
mEmailConnexionInput = (EditText)
findViewById(R.id.activity_main_emailInput);
mMotDePassConnexionInput = (EditText)
findViewById(R.id.activity_main_motDePassInput);
mConnectionButton = (Button)
findViewById(R.id.activity_main_connectionButton);
mInscriptionLink = (TextView)
findViewById(R.id.activity_main_inscriptionLien);
```

Gérer les actions de l'utilisateur

Dans cette page de connexion, l'activité doit interagir avec l'utilisateur et répondre à ses différentes actions. Les deux actions sur lesquelles l'activité doit répondre sont : les clics sur le bouton et sur le lien d'inscription. Donc, le seul évènement qui nous intéresse ici est « l'évènement clic ». Notre classe ActivityMain doit donc implémenter l'interface « View.OnClickListener » et donc implémenter la méthode « onClick() » de cette interface. C'est dans cette méthode qu'il faut faire le nécessaire pour réagir aux actions de l'utilisateur. L'implémentation intégrale de cette méthode se fera en salle, mais il faut comprendre comment s'effectue le démarrage d'une nouvelle activité à partir de la méthode : exemple, quand l'utilisateur saisit son email et son mot de passe, il doit être redirigé vers la page de présentation de son emploi du temps, après validation de son compte.

Ces deux lignes de code permettent de démarrer l'activité qui gère la présentation d'un emploi du temps :

```
Intent afficherEmploiActivity = new Intent(MainActivity.this,
AfficherEmploiDuTempsActivity.class);
startActivity(afficherEmploiActivity);
```

La méthode « startActivity() » permet de communiquer avec le système d'exploitation Android en lui demandant de démarrer une activité donnée. Pour préciser quelle activité lancer, un objet spécifique est utilisé : « Intent ». Lorsque la méthode « startActivity() » est appelée, l'objet interne Android « ActivityManager » inspecte le contenu de l'objet « Intent » et démarre l'activité correspondante.

Pour l'objet « Intent », il existe plusieurs constructeurs. Celui utilisé ici prend un premier paramètre qui correspond au contexte de l'application. Pour faire simple, cela correspond à l'activité appelante (car la classe Activity hérite de la classe Context). Le second paramètre correspond à la classe de l'activité à démarrer.

Valider le compte (adresse mail et mot de passe) de l'utilisateur

En développement, il est conseillé de toujours valider un formulaire côté client avant d'envoyer les données au serveur. Par exemple, une adresse mail doit normalement respecter un format ; un mot de passe aussi peut avoir un minimum et un maximum de caractères. Ces vérifications peuvent être effectué coté client avant que l'on vérifie au serveur si le compte existe réellement.

Pour faire ce type de vérifications, il existe plusieurs outils que vous pouvez voir sur Google en tapant par exemple : « Android form validation ». Awesome (rechercher sur Google « Android form validation using Awesome ») est l'un de ces outils que nous allons utiliser : nous le verrons en salle.

Gestion du bouton « Back » à la page d'accueil

Sur la page d'accueil, on voudrait que si l'utilisateur clique sur le bouton « Retour » du téléphone que l'application lui demande de confirmer s'il veut quitter. L'implémentation se verra en salle.

Utilisation d'un Snackbar

Vous pouvez utiliser un Snackbar pour afficher un bref message à l'utilisateur. Contrairement aux notifications, le message disparaît automatiquement après une courte période. Un Snackbar est idéal pour les messages brefs auxquels l'utilisateur n'a pas nécessairement besoin d'agir. Par exemple, une application de messagerie peut utiliser une Snackbar pour indiquer à l'utilisateur que l'application a réussi à envoyer un e-mail. Lien : https://developer.android.com/develop/ui/views/notifications/snackbar/showing.

Passer des données d'une activité à une autre

On voudra par exemple, quand un étudiant se connecte depuis la page de connexion, il est redirigé vers la page de présentation de son emploi du temps. Cette page doit donc récupérer et exploiter l'étudiant qui s'est connecté et cette donnée lui est passée par la page de connexion. Pour passer la donnée « étudiant » (qui est ici un objet), il faut utiliser la méthode « putExtra() » de l'Intent avant l'appel de la méthode « startActivity() ».

```
Intent afficherEmploiActivity = new Intent(MainActivity.this,
AfficherEmploiDuTempsActivity.class);
afficherEmploiActivity.putExtra("etudiant", etudiant);
startActivity(afficherEmploiActivity);
```

Pour passer un objet à « putExtra() », comme c'est le cas ici, sa classe doit être « sérialisable » (implémenter Serializable), ainsi que ses sous classes éventuelles.

Pour récupérer l'objet passé dans l'Intent, il faut faire comme suit :

```
Etudiant etudiant = (Etudiant)intent.getSerializableExtra("etudiant");
```

Fin.