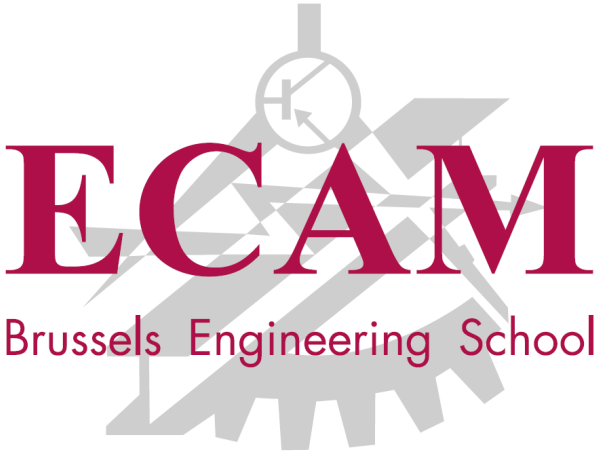

ECAM, 4EI (section génie électrique)

Projet d'application mobile

Application de récupération de scores sportifs

Auteurs :
M. AmauryLekens
M. BenoitWery

Encadrant :
Mr. Lurkin

Q2 2018

Table des matières

Chapitre 1

Présentation du projet

1.1 Que fait l'application ?

Le projet suivant est une application mobile qui a pour but principal de rechercher et d'afficher des scores de football issus de différentes compétitions.
L'utilisateur peut effectuer une recherche sur base d'une compétition, d'une équipe et d'un intervalle de temps entre deux dates. Tous les matchs correspondants à ces trois critères sont ensuite affichés.

L'application offre également la possibilité d'enregistrer des favoris, pour éviter de devoir sélectionner à chaque recherche une compétition et une équipe qu'il consulte fréquemment. De cette façon, il enregistre les critères de la recherche et celle-ci sera disponible à tout moment dans les favoris. L'accent a été mis sur la simplicité d'utilisation en ne surchargeant pas celle-ci de fonctionnalités.

1.2 Origine des données

Pour fonctionner, l'application doit évidemment avoir accès à des données. Dans l'optique d'avoir un large panel de choix dans les différents critères de recherches, il est impossible de développer une base de données personnelle. En effet, cela demanderait un travail conséquent et inutile étant donné la présence de solutions existantes et faciles à mettre en oeuvre. Notre choix s'est donc porté vers l'utilisation d'une API, que nous avons sélectionnée sur base de 3 critères : la facilité d'utilisation, les requêtes nécessaires à l'application, la gratuité.
L'API retenue est la suivante : `https://apifootball.com/`. Il s'agit d'une API REST (requêtes http) et elle dispose de filtres de recherches correspondants à notre application (voir

documentation).De plus, elle possède une version gratuite et une version de type premium avec 15 jours d'essais (que nous utiliserons).

1.3 Cahier des charges

1.3.1 Activité d'accueil

Cette page doit permettre d'accéder à :
— La recherche d'un score
— La gestion des favoris
— La gestion des préférences

1.3.2 Recherche d'un score

La recherche d'un score est divisée en plusieurs activités :
— Sélection d'une compétition : toutes les compétitions disponibles sur l'API sont récupérées et présentées à l'utilisateur pour qu'il puisse effectuer son choix.
— Sélection d'une équipe : toutes les équipes qui participent à la compétition sont récupérées et affichées à l'utilisateur pour qu'il puisse effectuer un choix.
— Sélection de dates : l'utilisateur doit pouvoir sélectionner un intervalle de temps.
— Affichage des résultats : tous les résultats correspondants aux critères de recherches sont affichés à l'utilisateur. Ces résultats correspondent donc à l'ensemble des rencontres de l'équipe (avec les scores), dans le cadre de la compétition choisie et dans le temps imparti.

1.3.3 Gestion des favoris

L'activité de gestion des favoris permet deux choses :
— Ajouter des favoris via un bouton. L'utilisateur est alors redirigé vers l'activité de sélection des compétitions et d'équipes.
— Sélectionner un des favoris et effectuer une recherche dessus. L'utilisateur est alors redirigé vers l'activité du choix des dates.

1.3.4 Gestion des préférences

Cette activité permet de gérer les préférences de l'application. Elle doit permettre en premier lieu de changer l'apparence de l'application (sa couleur).

1.3.5 Fonctionnement hors ligne

L'application doit être capable de fonctionner sans connexion Internet. Pour cela, nous enregistrerons une copie locale des données de l'API et celle-ci sera ensuite utilisée lorsque la connexion n'est pas disponible.

1.3.6 Ergonomie

L'application doit permettre un bon fonctionnement aussi bien en mode portrait qu'en mode paysage. De plus, la gestion de la navigation doit se faire de telle sorte qu'une activité garde connaissance de son état et n'effectue pas à chaque fois de nouvelles requêtes vers l'API.

Chapitre 2

Implémentation

2.1 Recherche d'un score

2.1.1 Vérification de la connectivité

Avant que l'utilisateur puisse effectuer une recherche, le système vérifie si une **connexion Internet valide** est disponible.

— Si tel est le cas, alors la recherche pourra se faire en ligne et donc en utilisant les données les plus à jour de l'API.
— Dans le cas contraire, un message l'averti qu'il est en mode *offline* et donc les recherches se feront via les fichiers json existants. Ceux-ci auront été mis à jour la dernière fois que l'application a pu se connecter à l'API.

La vérification de la connexion est effectuée par le code ci-dessous : on vérifie qu'un réseau est disponible et que l'appareil est connecté à celui-ci.

```
ConnectivityManager connectivityManager = (ConnectivityManager)
    getSystemService(CONNECTIVITY_SERVICE);
NetworkInfo networkInfo =
    connectivityManager.getActiveNetworkInfo();

if(networkInfo != null && networkInfo.isAvailable() &&
    networkInfo.isConnected()) {
     // go to competition choice
}
```

Dans les sections suivantes, nous considérons le cas où la connexion existe et où les requêtes vers l'API se font donc via le protocole HTTP.

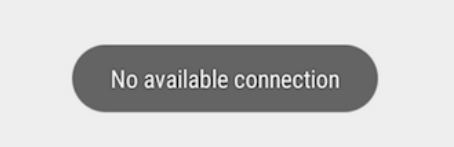

FIGURE 2.1 – Toast d'avertissement

2.1.2 Choix du championnat

En lançant cette activité, l'utilisateur provoque une requête http qui va récupérer les données des championnats via l'API. Pour ne pas bloquer l'interface graphique, la récupération des données est effectuée de manière **asynchrone**. Concrètement, on va créer dans notre activité une classe interne qui implémente la classe abstraite *AsynchTask* fournie par *Android*.

```
public class QueryTask extends AsyncTask<String, Void, String> { }
```

Cette classe va implémenter 3 méthodes :
- *protected void onPreExecute()* : effectue une tâche avant l'exécution asynchrone.
- *doInBackground(String... params)* : effectue l'exécution asynchrone. C'est dans cette méthode que se trouvera la requête http.
- *protected void onPostExecute(String queryResults)* : effectue une tâche après l'exécution asynchrone. Cette méthode reçoit en paramètre la valeur retournée par *doInBackground*.

La requête http en tant que telle s'effectue via un *inputStream* qui est récupéré sur une connexion http. Le code suivant effectue ces deux tâches.

```
HttpURLConnection urlConnection = (HttpURLConnection)
    urlObject.openConnection();
InputStream in = urlConnection.getInputStream();
```

Une fois les données récupérées, il faut les afficher. Pour cela on va utiliser un *Adaptater* et un *AdaptaterView*.

1. On crée une liste avec les données réceptionnées.
2. On passe cette liste à un *Adaptater* ainsi qu'un *Layout* à respecter pour chaque élément de la liste.
3. On passe l'*Adaptater* à un *AdaptaterView* qui se charge du *Layout* de l'ensemble.

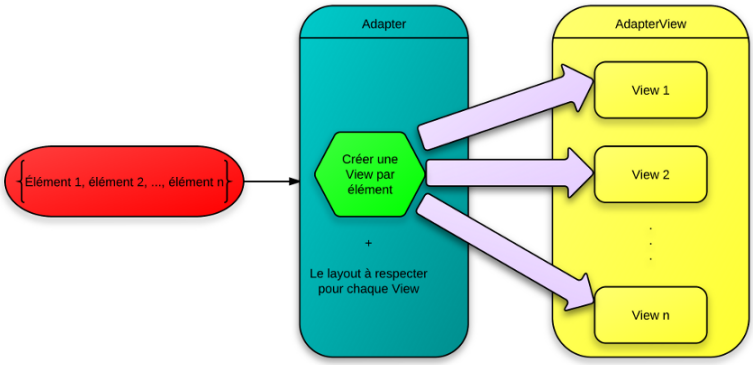

FIGURE 2.2 – Schéma de l'adaptateur (source : openclassroom)

Au niveau du code, ces trois étapes sont effectuées comme ceci :

```
listView = (ListView) findViewById(R.id.listView);
ArrayAdapter<String> adapter = new
    ArrayAdapter<String>(ChampChoiceActivity.this,
    android.R.layout.simple_list_item_single_choice,
    competitionsList);
listView.setAdapter(adapter);
```

Une fois les données affichées, l'utilisateur peut sélectionner une compétition et passer à l'étape suivante via le bouton *Submit*.
En interne, l'activité en lance une autre, via l'*Intent*, et lui fournit les données suivantes :

- l'id de la compétition,
- le mode : qui permet de différencier le cas où l'utilisateur effectue une recherche de score (*search*) et le cas où il veut ajouter un favori (*addFavorite*).

2.1.3 Choix de l'équipe

L'organisation de cette activité est très semblable à l'activité précédente (requête http asynchrone, affichage des données, choix utilisateur) mais varie en quelque points :

- La requête http et le traitement des données réceptionnées sont différents.
- La gestion de l'événement *Submit* : la méthode qui gère l'événement (*onClick()*) va vérifier le mode (SEARCH ou ADDFAVORITE). Dans le cas du mode "*search*", il lance l'activité du choix de la date en lui passant l'identifiant de la compétition et le nom de l'équipe. Dans le cas du mode "*addFavorite*", il écrit le nom de l'équipe et l'identifiant du championnat en dur dans un fichier et lance l'activité de gestion des favoris.

2.1.4 Choix de la date

Dans cette activité, il n'y pas de requête http ou de gestion d'affichage de liste. Il y a simplement deux widgets de type *DatePicker* qui permettent à l'utilisateur de sélectionner deux dates, pour déterminer un intervalle de temps.
Lors de l'événement *Submit*, l'activité de l'affichage du résultat est lancée en lui passant l'identifiant du championnat, le nom de l'équipe et les deux dates.

2.1.5 Affichage du résultat

Cette activité va effectuer une requête http avec tous les paramètres reçus de l'activité précédente et gérer l'affichage des données réceptionnées.

2.2 Gestion des favoris

2.2.1 Recherche sur un favori

A son lancement l'activité de gestion des favoris va lire dans un fichier les équipes précédemment enregistrées. Cette opération est effectuée comme une simple lecture dans un fichier en Java.

```
FileInputStream input = openFileInput(originFile);
StringBuffer readed = new StringBuffer();

while((value = input.read()) != -1) {
    readed.append((char)value);
}
```

Une fois les données récupérées, elles sont affichées comme précédemment avec un *Adaptater* et un *AdaptaterView*. L'utilisateur peut alors sélectionner une équipe et effectuer une recherche : il sera directement renvoyé vers l'activité du choix de la date.

2.2.2 Ajout d'un favori

Pour ajouter un favori l'utilisateur clique sur le bouton *Add favorite* qui va lancer successivement l'activité de choix du championnat et ensuite celle du choix de l'équipe mais cette fois avec le mode "*addFavorite*". Une fois le choix de l'équipe validé avec le bouton *Submit*, la méthode *onCreate()* reconnait le mode "*addfavorite*". L'identifiant de la

compétition et le nom de l'équipe sélectionnée sont alors enregistrés en dur sur un fichier. Cette opération est réalisée par une simple écriture en Java.

```
FileOutputStream output = openFileOutput(destinationFile,
    MODE_APPEND);
String toWrite = "toWrite";
output.write(toWrite.getBytes());
```

2.3 Préférences

Cette fonctionnalité permet à l'utilisateur de configurer l'application. Pour l'instant elle ne permet que la configuration de la couleur. Android permet une gestion aisée de la configuration avec ses composants : *PreferenceScreen*, *PreferenceFragment* et *SharedPreferences*

PreferenceScreen C'est un composant XML à l'intérieur duquel on va définir nos différentes options de configuration. A chaque option, on va attribuer une clé afin de pouvoir récupérer les valeurs dans les différentes activités :

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
<ListPreference
    android:defaultValue="red"
    android:entries="@array/color_labels"
    android:entryValues="@array/color_values"
    android:key="color"
    android:title="Color" />
</PreferenceScreen>
```

PreferenceFragment : il permet d'intégrer le composant *PreferenceScreen* dans le layout de configuration de l'application.

SharedPreferences : il permet de récupérer les valeurs des préférences dans la logique de l'application. Dans notre cas, on récupère la valeur de la couleur sélectionnée pour appliquer le thème adéquat.

```
SharedPreferences sharedPreferences =
    PreferenceManager.getDefaultSharedPreferences(this);
String theme = sharedPreferences.getString("color", "");
```

2.4 Ergonomie

2.4.1 Orientation de l'écran

Un changement d'orientation de l'écran fait appel à la méthode *onDestroy()* d'une activité, il est nécessaire d'implémenter une sauvegarde de son état courant. Dans le code de l'annexe .2, on peut voir que les états des éléments *View* de l'activité sont sauvés dans un Bundle via la méthode *onSaveInstanceState(Bundle outState)*. A la création de cette même activité, elle vérifie si ce Bundle est vide, dans quel cas elle se crée par défaut, ou s'il contient des valeurs, dans quel cas elle peut recréer son état de sauvegarde.

Dans certains cas, un *Layout* aproprié a été redéfini pour le mode paysage, tel que ci-dessous. Le choix du bon *Layout* est fait automatiquement par l'application selon l'orientation de l'écran.

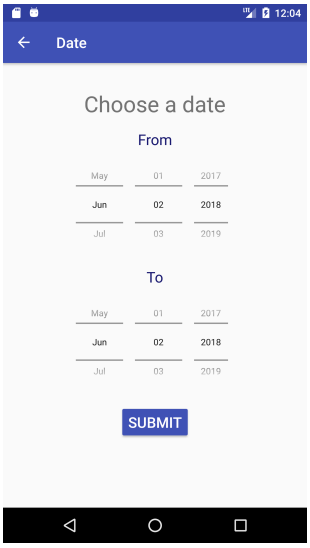

FIGURE 2.3 – Orientation portrait

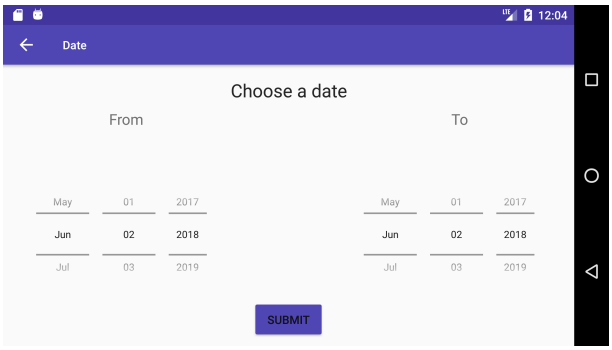

FIGURE 2.4 – Orientation paysage

2.4.2 Navigation

La navigation a été assurée de telle sorte que la flèche de retour (*Up Navigation*) d'une activité renvoie directement à son parent, à l'état dans lequel il se trouvait avant de passer en *background*. Pour cela, on utilise également le Bundle *saveInstanceState* dans le parent et la fonction *onOptionsItemSelected(MenuItem item)* dans l'enfant, pour faire le retour en arrière (exemple voir annexe .2). De cette façon, l'activité du parent n'effectue pas de nouvelle requête http car le contenu de la première requête a été enregistré.

Chapitre 3

Améliorations

3.1 Refractoring

Le projet avait pour but principal de découvrir l'environnement de développement d'Android. Le code est loin d'être optimal : il semble possible de l'améliorer tout en conservant ses fonctionnalités.

3.2 Visuel

Il s'agit ici d'un critère que nous avons volontairement simplifié au maximum dans la mesure où il s'agit de détails d'affichages et de configurations des Layouts,...

3.3 Configurations supplémentaires

L'application est pour l'instant pauvre en configurations (seulement la couleur). Toutefois, l'intérêt étant ici de voir comment des préférences peuvent être gérées au moyen des fragments existants, nous n'avons pas jugé utile d'en ajouter d'autres. Des configurations supplémentaires, comme par exemple l'autorisation d'utiliser les réseaux 3G/4G ou la sélection de la langue, peuvent être facilement ajoutées une fois ces concepts intégrés.

.1 Screenshots des activités

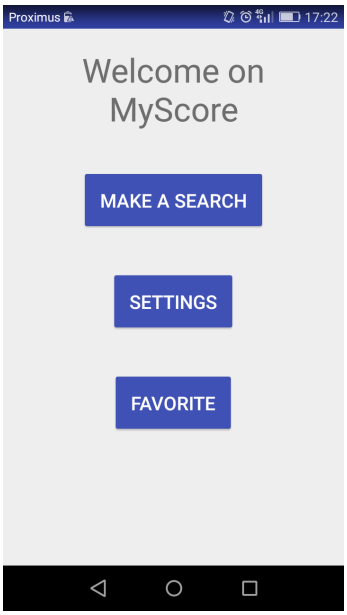

FIGURE 1 – Page d'accueil

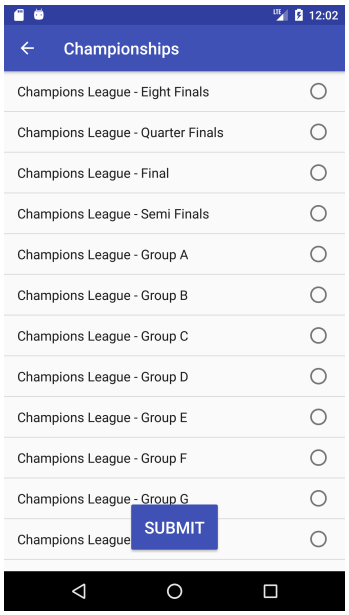

FIGURE 2 – Sélection championnat

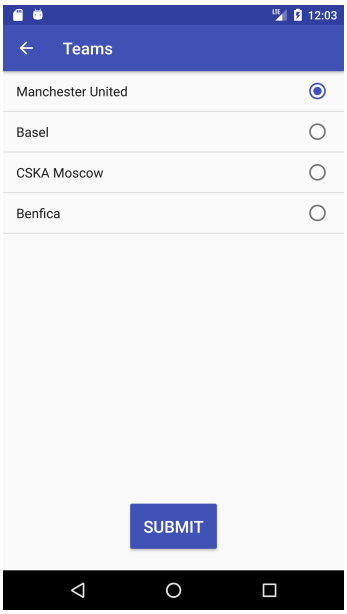

FIGURE 3 – Sélection équipe

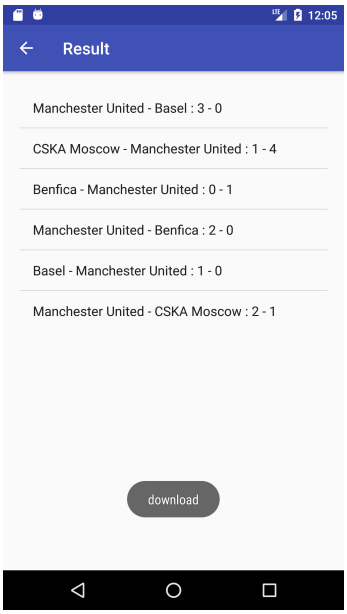

FIGURE 4 – Résultat de la recherche

.2 Orientation écran

```
public class TeamChoiceActivity extends AppCompatActivity implements
    View.OnClickListener{

    static final String STATE_COMPETITION_ID = "competition_id";
    static final String STATE_MODE = "mode";
    static final String STATE_TEAMS = "teams_names";

    [...]

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        [...]

        /* Check if the activity has a previous state saved */
        if(savedInstanceState != null) {
            if(savedInstanceState.containsKey(STATE_COMPETITION_ID)) {
                competition_id=
                    savedInstanceState.getString(STATE_COMPETITION_ID);
                mode = savedInstanceState.getString(STATE_MODE);
                names =
                    savedInstanceState.getStringArrayList(STATE_TEAMS);
            }
        }else{
            // Get data from the intent
            i = getIntent();
            competition_id = i.getStringExtra("COMPETITION");
            mode = i.getStringExtra("MODE");
            String output =
                String.format("https://apifootball.com/api/....");
            new QueryTask().execute(output);
        }

        [...]

    }

    [...]

    // Enable Up navigation to make a proper return on parent activity
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if(id == android.R.id.home) {
        NavUtils.navigateUpFromSameTask(this);
    }
    return super.onOptionsItemSelected(item);
}

/* This onSaveInstanceState method saves the current state
   (before onStop() is called)
 * in a Bundle to save information about each View object in your
    activity layout.
 * This bundle is then used for the re-creation of the activity
    (method onCreate() )
 * with its previous state
 */
@Override
protected void onSaveInstanceState(Bundle outState) {
    outState.putString(STATE_COMPETITION_ID, competition_id);
    outState.putString(STATE_MODE, mode);
    outState.putStringArrayList(STATE_TEAMS, (ArrayList)names);

    super.onSaveInstanceState(outState);
}
```