Suivi de traffic aérien

Vous êtes dans une jeune entreprise créant des applications comme services.
Un client vient à vous pour vous confier le projet suivant.
Le but ici est de créer une application montrant des informations à propos du traffic aérien.

OBJECTIF: Créer une application Android en utilisant le SDK Android et répondant aux spécifications qui suivent.

L'API sur laquelle se baser est https://opensky-network.org/ dont la documentation est https://opensky-network.org/apidoc/rest.html

Liste de spécifications:
- Ecran 1: En entrant dans l'application, nous devons permettre à l'utilisateur de sélectionner un aéroport de départ ou d'arrivée, ainsi qu'un intervalle de temps sur lequel il veut voir les vols.
- Ecran 2: Pour la liste des vols, afficher pour chacun des éléments:
    - L'aéroport de départ
    - L'aéroport d'arrivée
    - Date de départ (heure locale)
    - Date d'arrivée (heure locale)
    - Le temps de vol
    - Le nom du vol
- Ecran 3: Le clic sur un élément de la liste permet de voir une carte avec le trajet effectué par l'avion sur ce vol. Un bouton « Voir détails de l'avion » y est également proposé.
- Ecran 4: Un clic sur le bouton précédent permet de voir la position actuelle de cet avion sur une carte ainsi que les caractéristiques actuelles de son vol (si disponibles): Vitesse, altitude, montée ou descente, ainsi que toutes les informations sur son vol actuel.
- Ecran 4bis: Toujours dans ce dernier écran, afficher la liste des vols de cet avion sur les trois derniers jours. Afficher pour chacun des éléments:
    - L'aéroport de départ
    - L'aéroport d'arrivée
    - Date de départ (heure locale)
    - Date d'arrivée (heure locale)
    - Le temps de vol
    - Le nom du vol

Ressources:
Pour retrouver la liste des aéroports:
https://gist.github.com/tdreyno/4278655

Autres considérations:
- Le design proposé devra respecter le material design.
- L'application devra fonctionner sur tablette, avec un design adapté.
- L'application ne pourra fonctionner qu'avec une connexion. Proposer des écrans le cas échéant.
- Ecrire un code clair, compréhensible pour une autre équipe

EXAMPLES DE L'API

https://opensky-network.org/api/tracks/all?icao24=3964e1&time=1572210401
https://opensky-network.org/api/flights/arrival?
airport=LFPO&begin=1572172110&end=1572258510
https://opensky-network.org/api/flights/aircraft?
icao24=3964e1&begin=1572172110&end=1572258510
https://opensky-network.org/api/states/all?icao24=3964e1&time=0

TP M2 : Flight Archives

Ce document est pour vous une feuille de route, permettant de construire le projet pas à pas. Vous pouvez ne pas le suivre à la lettre et sauter les étapes voire les faire dans le désordre. Vous organiserez le travail tel que vous le voulez. Le barème à la fin de chaque partie indique le poids de chacune d'entre elles (Le barème complet est accessible à la fin de ce document.
Ce projet sera noté et comptera pour 2/3 de la moyenne. L'autre tier sera un examen en classe, lors du dernier jour de cours le 13 décembre 2022.

Il est également demandé que le projet soit disponible dans un repo git ce qui permettra de voir l'évolution du projet via l'historique de commits. **Ce projet sera noté sur son état dans le repo au 18 décembre 2022 à 23h59.** Les commits venant après cette date ne seront pas pris en compte. L'emplacement du repo devra être fourni par email dès que possible à miranda@goodbarber.com.

Ce document décrit un projet construit sur Kotlin, les live coding seront également en Kotlin. Vous pouvez choisir de faire votre projet en JAVA, mais sachez que les dépendances devront être ajustées et que le les Coroutines n'existent pas…

Dernière chose, il y aura un peu de live coding pour débloquer certains aspects ou illustrer des concepts. Il est donc possible de consulter l'état du project en classe ici: https://github.com/sergio2a/flightarchives
Il faut avoir en tête que ce projet sera à la fin très sommaire et ne saurait pas constituer un projet assez élaboré pour avoir la moyenne à ce TP (nous irons jusqu'au point V en live coding, pas au delà). Néanmoins, vous pouvez vous en servir comme base si vous le souhaitez.

Il est possible de faire ce TP par 2 maxi. Les groupes devront être faits dès le départ du TP et resteront inchangés.

I. Creation du projet Android

Points: 0

A. Création du projet

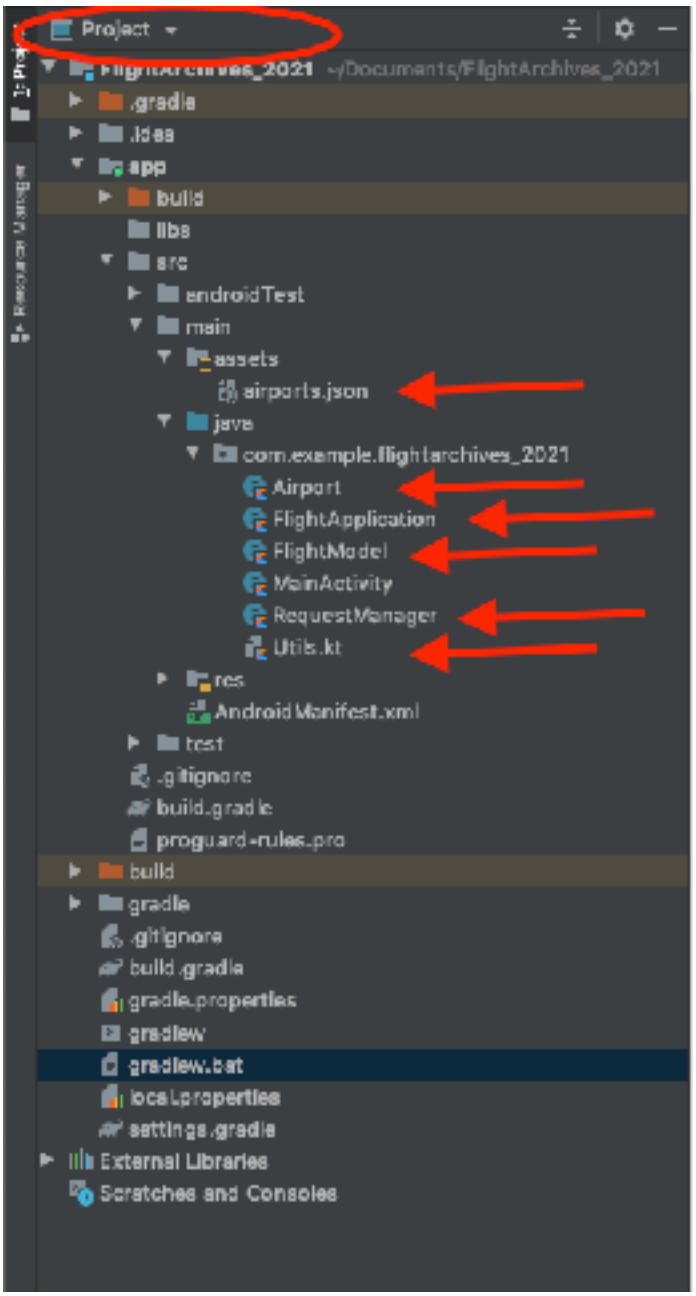
Note: Veillez à ce qu'Android Studio soit en version 4.0 ou ultérieure. Si ce n'est pas le cas et que vous êtes sur votre propre machine, il est conseillé de faire une mise à jour. Sinon bonne chance :D

Ouvrez Android Studio et créez un nouveau projet. Sélectionnez les options suivantes:
- *Project Template:* Empty Activity
- *Language*: Kotlin
- *Minimum SDK*: API 21

Lancer votre application sur un simulateur.
Si votre application ne se lance pas. N'allez pas plus loin et faites appel à l'intervenant

B. Ajout des fichiers utils

Afin d'aller plus vite et aider au démarrage de l'application, certains fichiers ont été créés et sont disponibles sur https://drive.google.com/file/d/1MNoDb0diB_Aa6SjQcG4v2f6IZolEAsu3/view?usp=sharing. Récupérer les et ajoutez les dans le projet tels qu'il est décrit dans la capture d'écran suivante (veillez à être en vue « Project » au lieu de « Android » par défaut):



C. Ajustement des dépendances

Dans le fichier *AndroidManifest.xml*, ajoutez le paramètre suivant dans le tag *Application:*

```
android:name=".FlightApplication"
```

Dans le fichier *app/build.gradle*, ajoutez les dépendances suivantes:

```
implementation 'commons-io:commons-io:2.4'
implementation 'com.google.code.gson:gson:2.8.6'
```

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
```

Enfin, Afin de tester que cette étape est correcte, ajouter la ligne suivante dans votre activité puis lancez l'application sur votre appareil Android:

```
Utils.generateAirportList()
```

Si l'application n'a aucun crash au lancement, on est tout bon.

D. Création d'un repo git

Il est temps de mettre votre projet sur Github ou autre service git permettant de récupérer votre projet. Accordez l'accès en lecture à miranda@goodbarber.com
Jetez un oeil sur https://kbroman.org/github_tutorial/pages/first_time.html pour comment le faire sur Github.

II. Création de la maquette

Points: 15

Lorsqu'un projet nous est donné, il vaut mieux savoir ou on va. Pour cela, on va établir un prototype visuel de ce à quoi l'application va ressembler et nous baser dessus pendant le développement.

Le prototype peut être fait à main levé ou sur un outil dédié. C'est le contenu et la réflexion qui en découle qui sont essentiels pour la notation.

Doit apparaitre sur la maquette:
- Tout les écrans avec leurs éléments complet
- Version mobile et tablette de l'application
- La navigation entre les écran doit être explicitée (en mobile et tablette)

N'hésitez pas à expliciter le concept de votre application pare des notes, des flèches, …

Quelques éléments pour l'écran 1:
Pour avoir la liste des vols à afficher sur l'écran 2, il faut faire la requête suivante: https://opensky-network.org/api/flights/arrival?airport=LFPO&begin=1572172110&end=1572258510
Nous avons donc besoin de 4 infos que doit nous fournir l'utilisateur:
- L'aéroport
- Veut-on les vols au départ ou à l'arrivée de l'aéroport?
- Le début de l'intervalle de recherche
- La fin de l'intervalle de recherche

III. La première page

Points: 10

Pénalités: -5 si non utilisation de LiveData et de ViewModel
-2 si non utilisation de Coroutines

A. Implémenter les éléments graphiques

Pas de contraintes ici sur comment disposer les éléments. Il faut vérifier que l'interface reste correcte sur tablette.

B. *Affichages des aéroports*

La fonction *Utils.generateAirportList()* nous permet de récupérer la liste des aéroports présents dans le fichier airports.json.
Cette liste doit être affichée quelque part. Si vous avez choisi d'utiliser un Dropdown (ou Spinner su Android), il vous faudra utiliser un ArrayAdapter pour permettre l'affichage et la sélection des aéroports dans le Spinner

Après vérification de l'affichage et de la sélection de l'aéroport, il faut être capable de récupérer la valeur sélectionnée. Vous pouvez par exemple la stocker dans un liveData contenu dans le ViewModel

C. *Gestion des dates*

La gestion de la date peut être très longue car il y a de nombreuses choses à faire pour que le tout fonctionne et ait un sens pour l'utilisateur.
La solution UX dans votre maquette vous a peut-être mené au fait que nous avons besoin. d'afficher deux DatePicker pour définir l'intervalle de la recherche des vols sur un aéroport.
Il faudra être capable de sélectionner une date (classe *Calendar*) pour chacun des pickers et stoker la valeur quelque part (ViewModel + LiveData ? :)).

Il faut également formatter la date stockée pour qu'elle soit lisible par l'utilisateur. Cela se fait en utilisant l'API SimpleDateFormatter.

Enfin, les limites de l'API d'Opensky Network ainsi que le bon sens va vous mener à la restriction des dates sélectionnable par l'utilisateur.

D. *Construction de l'URL*

Pour avoir la liste des vols à afficher sur l'écran, il nous faut plusieurs infos. Ces infos vont nous servir à exécuter la requête de type https://opensky-network.org/api/flights/arrival?airport=LFPO&begin=1572172110&end=1572258510

Dans cette étape, nous devons être capable de récupérer les informations entrées par l'utilisateur et générer l'URL à appeler

E. *Exécution de la requête*

Il est à présent temps d'exécuter la requête sur l'URL construite précédemment. Nous le ferons à l'aide d'une coroutine dans le ViewModel et de la fonction *RequestManager.getSuspended()*.
Pour valider cette étape il faut être sûr que nous sommes capables de récupérer la liste des vols (*List<FlightModel>*) via log ou en affichant le résultat brut dans une nouvelle activité par exemple.

Le succès de la requête doit entrainer l'ouverture d'une nouvelle activité. Une erreur doit entrainer l'affichage d'un *Toast*.

IV. Utilisation des fragments

Points: 0 mais le design sur tablette ne pourra être fait sans cela

Après explications de l'intervenant sur le concept des fragments, créer un fragment et y transférer l'UI du formulaire. Le rôle de l'activité est uniquement alors d'afficher ce fragment.

Dorénavant, chaque nouvelle interface devra être construite sur un fragment pour obtenir le maximum de points

V. Ecran n°2: la liste

Points: 10
Pénalités: -5 si non utilisation de LiveData et de ViewModel
-5 si la donnée affichée ne provient pas de a donnée réelle.

A. Création de la cellule

Le premier objectif pour afficher une liste est de créer une cellule type. Une fois le design voulu, nous pouvons y affecter la donnée

B. Affichage de la liste complète

Une fois la cellule faite, nous pouvons créer un RecyclerView avec son Adapter. Celui-ci aura pour travail d'affecter la donnée au bonnes cellules et engendre leur affichage

VI. Ecran n°3: Le trajet de l'avion

Points: 20
Pénalités: -5 si non utilisation de LiveData et de ViewModel
-2 si non utilisation de coroutines

A. Affichage de du bon vol

Cette première étape consiste à ouvrir un nouveau fragment au clic sur une des cellules de l'écran 2 en y passant l'information sur le vol sélectionné

B. Requête sur le vol

Comme précédemment, il faut être méthodique lorsqu'on fait une requête. Tout d'abord il faut savoir comment la construire. Une fois que nous savons générer une URL correcte, il faut exécuter la requête dans une coroutine.
Enfin il faut pour exploiter le résultat de la requête et le manipuler sous forme d'objet il faut penser à créer une *data class* qui est le modèle de la ressource demandée (comme *FlightModel*), puis populate ce modèle.
Comme précédemment et selon le besoin, veiller au bon fonctionnement via des Logs par exemple.

C. Affichage de la map

Pour afficher une map sur Android, il faudra créer des clés d'API à embarquer dans le projet. Il faut chercher comment faire dans la documentation Android.

D. Ajout du trajet

Une fois que la map est affichée correctement, nous pouvons dessiner dessus. Le résultat de la requête nous donne une liste de points qu'il faut afficher sur la carte.

VII. Ecran n°2 et 3 sur tablette

Points: 10

Dupliquer le layout de l'activité pour prendre en compte une interface tablette différente du mobile. Sur tablette, nous devons être capable d'afficher les écrans 2 et 3 en même temps.

L'activité doit être également capable de savoir quand afficher le fragment avec la carte (cas du mobile) ou quand la recharger uniquement (tablette).

VIII. Ecran n°4: Avion en temps réel

Points: 20

A. Création de l'interface

L'activité doit ici être capable de gérer un affichage mobile et un affichage pour tablette

B. Requête sur l'état de l'avion

Comme précédemment, il faut être méthodique lorsqu'on fait une requête. Tout d'abord il faut savoir comment la construire. Une fois que nous savons générer une URL correcte, il faut exécuter la requête dans une coroutine.
https://opensky-network.org/api/states/all?icao24=3964e1&time=0
Ne pas oublier la création d'un modèle pour manipuler le résultat de la requête

C. Affichage de la position courante

Comme précédemment, afficher la carte et un point montrant la position de l'avion

D. Affichage des autres paramètres en direct de l'avion

Après avoir lu la documentation sur l'API, afficher si disponible au minimum Vitesse, Altitude, Montée ou Descente

IX. Ecran n°4 bis: la liste des vols de l'avion sélectionné

Points: 10

L'URL pour pouvoir obtenir cette information est de type https://opensky-network.org/api/flights/aircraft?icao24=3964e1&begin=1572172110&end=1572258510

X. Evolutions / Optimisations possibles

Points: 5

Une fois que vous êtes allé au bout, que vous ayez fini ou non ce TP, j'aimerais avoir votre regard sur le travail que vous avez fourni. Les points à évoquer sont:

- Votre ressenti sur ce que vous avez produit
- Les améliorations possible pour ce projet en termes de fonctionnalités ou d'optimisations de l'actuel.

Annexe: Notation

I. 0 point
II. 15 points
III. 10 points
IV. 0 point
V. 10 points
VI. 20 points
VII. 10 points
VIII. 20 points
IX. 10 points
X. 5 points

Total: 100

En plus des fonctionnalités sur chacune des partie, les éléments généraux suivant seront notés

- **10** points bonus pour le soin accordé au **design** (au moins 3 écran fonctionnels)

Cela porte le nombre de point maximum à 110 points sur 100

Le correcteur se donne également le droit d'accorder des points supplémentaires pour tout originalité bienvenue sur l'application

Pénalités

- **Crash** trouvé: -**20** points
- Projet **non compilable:** -**30** points
- Suspicion de **copie** de projet sur un autre étudiant / groupe: -**60** points
- Projet non disponible sur un repository git: -**10** points