WEATHER STATION

Rendu projet TAC

Réalisation d'une application mobile Android

The Weather Station°

Alexy LEDAIN

Version 1.00 du 24/11/2020

1 Introduction

Dans ce document technique sera expliqué tous les choix techniques choisis pour la réalisation de l'application Android dans le cadre du projet de fin de semestre en TAC.

Il faut savoir que me concernant et très brièvement. Cela fait maintenant 3 ans que je suis en alternance à UbiSolutions. Une entreprise travaillant dans le domaine de la traçabilité informatique à l'aide d'outils tel que la RFID. Nous réalisons un grand nombre d'application mobile permettant à nos clients de pouvoir effectuer cette traçabilité. J'ai donc 3 années d'expérience dans le domaine de l'Android et tous mes choix techniques fait dans cette application mobile sont des choix techniques que j'utilise également en entreprise.

2 Librairies utilisées

Dans cette application, j'ai utilisé 2 librairies.

La première, est une librairie permettant de réaliser des appels à des webservices REST sur Android de la façon la plus simple qui soit. Son nom ? **Retrofit**. Elle a été développée par le groupe square.

Pour l'utiliser, il suffit d'ajouter les 2 lignes suivante à vos dépendances :

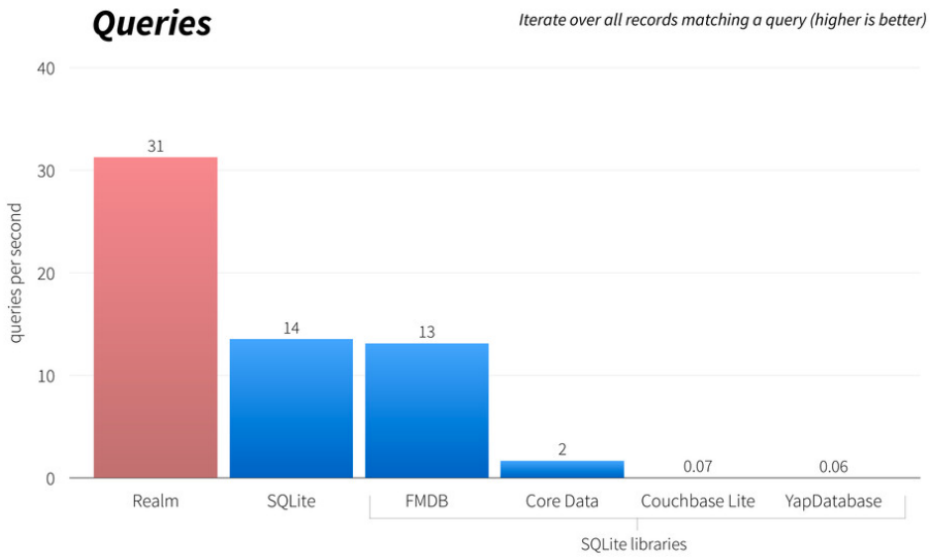
```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

La deuxième servant à convertir les données reçues au format JSON en objet très simplement.

La seconde, est une base de données créée dans le but de s'exécuter sur des plates-formes mobiles. Ses performances combinées avec sa facilité d'utilisation en font un outil très fonctionnel pour les développeurs mobiles que ce soit sur un projet Android ou IOS. Son nom ? **Realm**.

Pour l'utiliser, je vous invite à suivre le tutorial mise à disposition de l'équipe Realm à l'adresse suivante : https://realm.io/docs/java/latest/

Afin de vous donner une idée des performances de realm, ce diagramme ci-dessous montre les performances de celui-ci comparé à tous ces concurrents directs sur mobile.



3 Concepts utilisés

Dans cette application et afin de me démarquer des autres. J'ai décidé de réaliser celle-ci en Kotlin. Kotlin qui est un langage de programmation orienté objet et fonctionnel tous comme java, avec un typage static qui permet de compiler pour la machine virtuel Java, Javascript, et vers plusieurs plateformes en natif. Kotlin depuis quelques années penchent à devenir le langage de prédilection pour l'Android. Personnellement, pour l'avoir pratiqué pendant 2 ans. Je trouve le langage extrêmement utile afin de réaliser de petite application mobile. Cependant, ce n'est pas un langage à mettre entre les mains de jeunes développeurs, c'est un langage pratique est lisible lorsqu'il est bien utilisé. Un mauvais développeur utilisant du kotlin va créer une application mobile avec un code illisible. Je parle en connaissance de cause, j'ai eu le cas dans mon entreprise.

Dans cette application, j'ai également décidé d'utiliser le viewbinding ainsi que le databinding. C'est un concept que j'ai découvert très récemment en voyant qu'une des librairies que j'utilisais, Butterknife, a été remplacé par le viewbinding et le databinding. C'est un concept très simple qui permet de faire le lien entre le fichier XML et le fichier java ou kt juste en y mettant le nom du fichier XML dans le fichier java ou kt.

Prenons un exemple très simple :

- Je nomme un fichier XML → login_fragment.xml (majuscule non autorisé dans les fichiers xml)
- Je nomme un fichier JAVA ou KT → LoginFragment.kt/.java
- Dans mon fichier java/kt, je vais avoir un attribut de type LoginFragmentBinding, le lien entre mon fichier java et xml est implicite.

Le viewbinding permet principalement de supprimer les « findViewById » et de faire appel aux views de notre layout directement avec leurs ids.

Le dataBinding lui, permet de mettre des événements de click sur un bouton par exemple, directement dans le fichier XML. Il permet également grâce à la balise <data><data/> de rajouter un objet directement dans le xml afin de rempli les textViews de l'écrans par exemple.

4 Fonctionnement général de l'application

Mon application est bâtie de la manière suivante :

- MainActivity.java
  - Class qui extends **AppCompatActivity**.
  - **C'est la seule est unique activity de l'application.**
  - Cette activity gère le fonctionnement global de l'application.
  - C'est depuis celle-ci que nous allons lancer chaque fragment.
  - Lorsque l'application démarre, un fragment se lance directement.
  - Dans cette MainActicity, nous allons trouver toutes les méthodes nécessaires au lancement d'un fragment, c'est elle qui va gérer la pile de fragment.
- StartFragment.java
  - Synchronise les données depuis l'API et les stocks dans REALM.
- HomePageFragment.java
  - Affiche une liste avec la météo pour chaque ville, dans la toolbar il est possible de passer du mode liste au mode grille.
- DetailVilleFragment.java
  - Affiche le détail d'une ville au clic sur une cellule.

5 Parc matériel ciblé

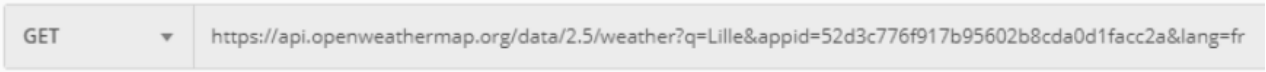
Mon application fonctionne sous Android 10. Soit une grande partie des mobiles Android de nos jours. Plus de 400 millions de smartphones sont sous Android à récemment annoncé google.

Mon application fonctionne aussi bien sur mobile que sur tablette.

L'API minimum autorisé est 24.

6 Contrat d'interface

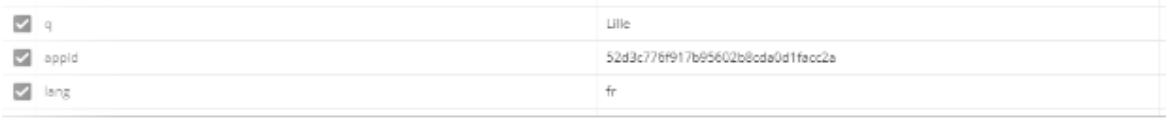
Mon application communique via le biais d'une seule requête de type GET. Cette requête est la suivante :



Dans cette requête, je passe en query param 3 choses :

- La ville dont je souhaite avoir la météo
- La clé d'accès à l'API (constante donnée par le créateur de l'API lors de la création du compte sur son site)
- La langue dont je souhaite recevoir la réponse

Voici un exemple :



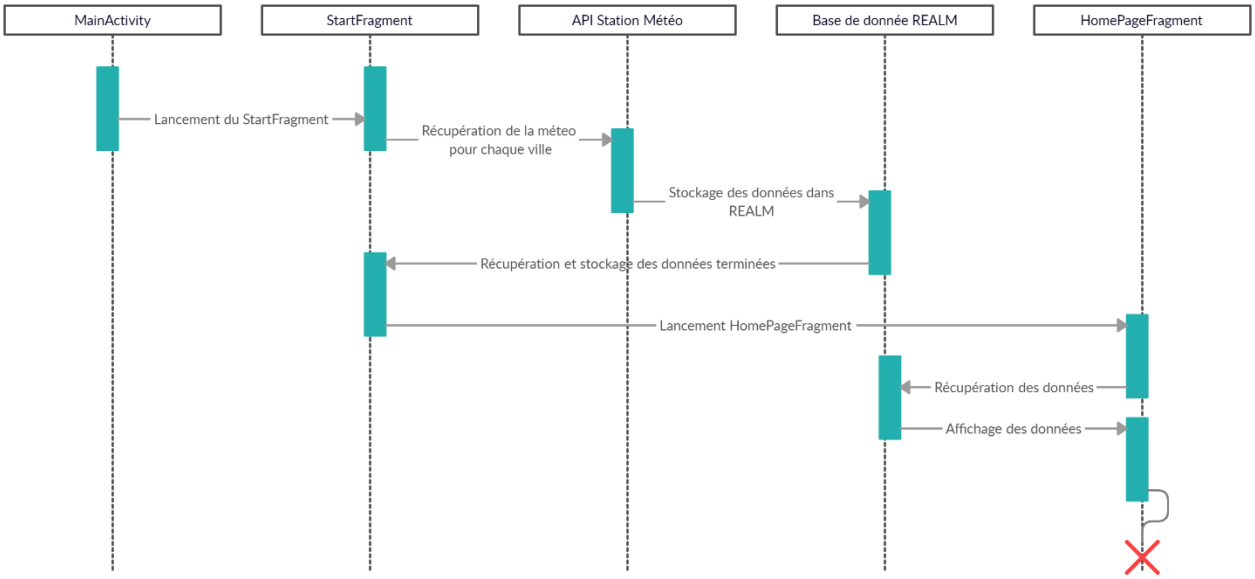
La réponse que me retourne ensuite l'API, est au format JSON. Elle me renvoie une liste d'information concernant la ville passée en paramètre sur la météo qu'il fait dans celle-ci.

Voici la liste des informations que cette API me renvoi :

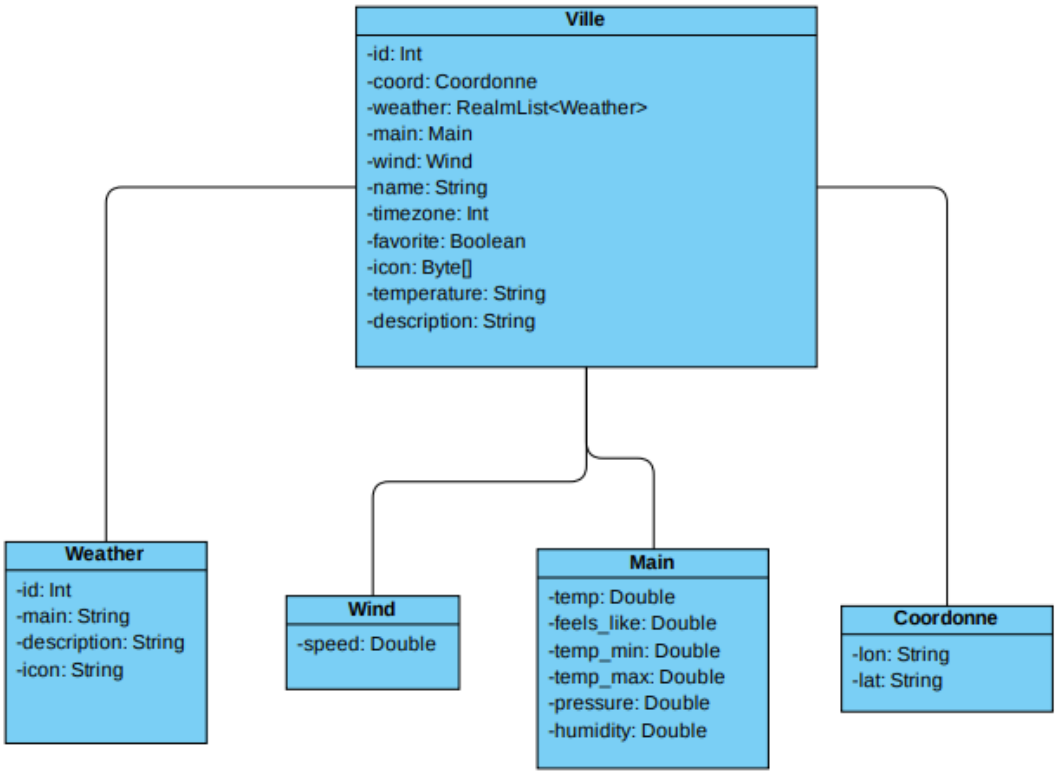

```
{
    "coord": {
        "lon": 3.06,
        "lat": 50.63
    },
    "weather": [
        {
            "id": 804,
            "main": "Clouds",
            "description": "couvert",
            "icon": "04n"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 282.74,
        "feels_like": 279.06,
        "temp_min": 282.04,
        "temp_max": 283.15,
        "pressure": 1016,
        "humidity": 81
    },
    "visibility": 10000,
    "wind": {
        "speed": 4.1,
        "deg": 160
    },
    "clouds": {
        "all": 90
    },
    "dt": 1606326166,
    "sys": {
        "type": 1,
        "id": 6559,
        "country": "FR",
        "sunrise": 1606288757,
        "sunset": 1606319449
    },
    "timezone": 3600,
    "id": 2998324,
    "name": "Lille",
    "cod": 200
}
```

Il est évident que toute ces informations ne me sont pas nécessaires. Les informations que je stock et avec lesquels j'interagis dans mon application sont les suivantes :

7 Diagramme de séquences



8 Diagramme de classe/package



9 Amélioration possible de l'application

Une amélioration possible de l'application serait de la rendre plus jolie esthétiquement parlant. Je ne me suis pas consacré à 100% dans la réalisation d'un design parfait. La situation actuelle ne m'a pas donnée d'inspiration concerné le design de cette application.

Une autre amélioration serait de récupérer la position GPS de l'utilisateur afin de lui permettre d'obtenir la météo dans le lieu où il se trouve actuellement.

On pourrait encore aller chercher dans une autre API externe la liste des plus grandes villes du monde afin de les afficher dans la liste. Actuellement seule quelques villes sont présentes, si l'utilisateur en veux plus, il doit les rajouter manuellement.