


Documentation technique – Libeery

Nicolas Fernandes – M1 E-Services - TAC

Sommaire :

Présentation et Fonctionnement de l'application :	2
Libraires utilisées	3
Le parc matériel ciblé	3
Choix et concept utilisés	4
Diagramme de séquences.....	5
Diagramme de classe	6
Contrat d'interface.....	7

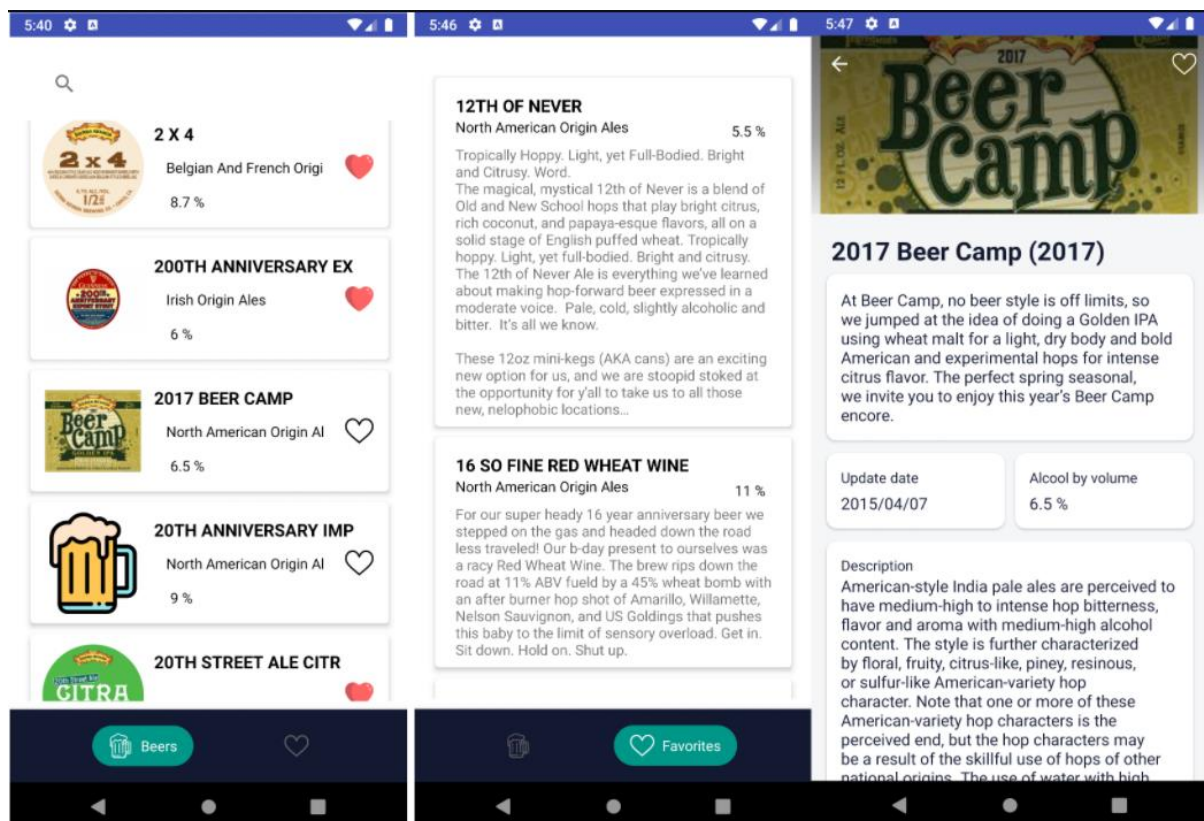
Présentation et Fonctionnement de l'application :

Libeery est une application mobile développée en Java qui permet de répertorier différents types de bières. Grâce à cette app vous pourrez plus facilement retrouver vos coups de cœur et même les sauvegarder en favoris. En effet l'app intègre un système de favoris, il suffit d'ajouter votre bière à l'aide de l'icône  Qui apparaît sur plusieurs écrans. Vous pouvez retrouver vos favoris dans le fragment « Favorites ». Il vous suffit de swiper vers la gauche et décocher le cœur pour supprimer la bière de vos favoris.

À l'aide de la barre au-dessus de la liste vous pouvez facilement rechercher une bière, en cliquant dessus vous verrez apparaître les détails la concernant (description plus complète, statuts des infos...).

Voici quelques captures de l'application :

Cliquez pour voir [le dépôt Github](#)



Libraires utilisées

Dans cette partie je vais vous présenter les différentes librairies que nous avons utilisées pour le développement de cette application.

Pour communiquer avec la partie serveur, API, nous avons utilisées [Retrofit2](#). Cette Librairie nous a facilitée la communication et intègre des outils pour parser directement les données reçues (grâce aux annotations) dans un POJO.

Nous avons aussi utilisé [Room](#) pour sauvegarder les données en local. Room permet d'intégrer une base de données en local. La persistance des données est très importante surtout pour les favoris.

Ensuite nous avons utilisé d'autres librairies qui ont permis d'intégrer un beau contenu sans réinventer la roue !

On a implémenté [Chip-navigation-bar](#) pour la barre de navigation en bas.

Grâce à [Picasso](#), nous avons pu insérer les images très facilement avec seulement une URL du type <http://notreimage>.

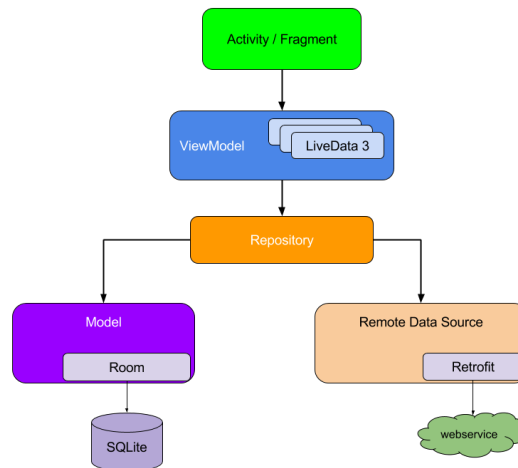
Le parc matériel ciblé

Nous avons choisi de rendre notre application disponible pour tous les appareils Android ayant une version de logiciel supérieure ou égale à Android 7.0 ([API level 24](#)), le SDK cible est le 30.

Cela représente 79 % du parc matériel Android. Nous ne pouvons pas aller en-dessous car la librairie chip-navigation-bar ne nous le permet pas.

Choix et concept utilisés

Pour le développement de cette application mobile, le choix conceptuel c'est porté sur une structure MVVM avec un pattern [repository](#). La communication entre chaque composant (Model, ViewModel, Repository) s'effectue avec des LiveData. Comme ceci :



Grâce à cette architecture de logiciel, la vue est « branchée » au LiveData et observe les données.

La classe Repository a accès aux deux sources de données, c'est elle qui fait office d'interface entre les données exploitées dans l'app et les données sauvegarder dans Room ou situé dans l'API.

L'activité principale de l'app se compose de 2 fragments, un pour la liste de bière et un pour les favoris. Les deux fragments sont composés de recyclerview, et donc d'adapter qui permettent d'interpréter les données du ViewModel dans les listes.

La récupération des données s'effectue grâce à l'interface BeerApi et BeerClient qui implémente Retrofit2 dans le but de requêter l'API.

Nous avons également une interface qui fait office de DAO pour room et une classe qui initialise/récupère l'instance de la base de données.

Nous avons plusieurs modèles, ce sont des POJO. C'est différents modèles de l'objet Beer permettent de séparer la forme des infos reçu depuis l'API et la forme sauvegarder dans notre bdd room. En effet nous avons un model pour l'API et un autre pour Room.

Durant la conception de cette app, nous avons fait face à quelques problématiques, notamment celle de la création du view model dans les différentes vues. Pour une cohérence de données dans notre application nous voulions la même instance de repository dans nos différentes vues. Nous avons alors implémenté factory dans la classe BeersViewModelFactory qui extends ViewModelProvider.Factory, cette classe est aussi un singleton.

Cette Classe nous permet d'instancier nos ViewModel dans nos différentes vues. A l'initialisation nous créons un repository qui sera repris pour toutes les autres instances de viewModelBeer créées avec cette factory.

Cette architecture permet une clarté du code et une bonne maintenabilité de l'application.

Diagramme de séquences

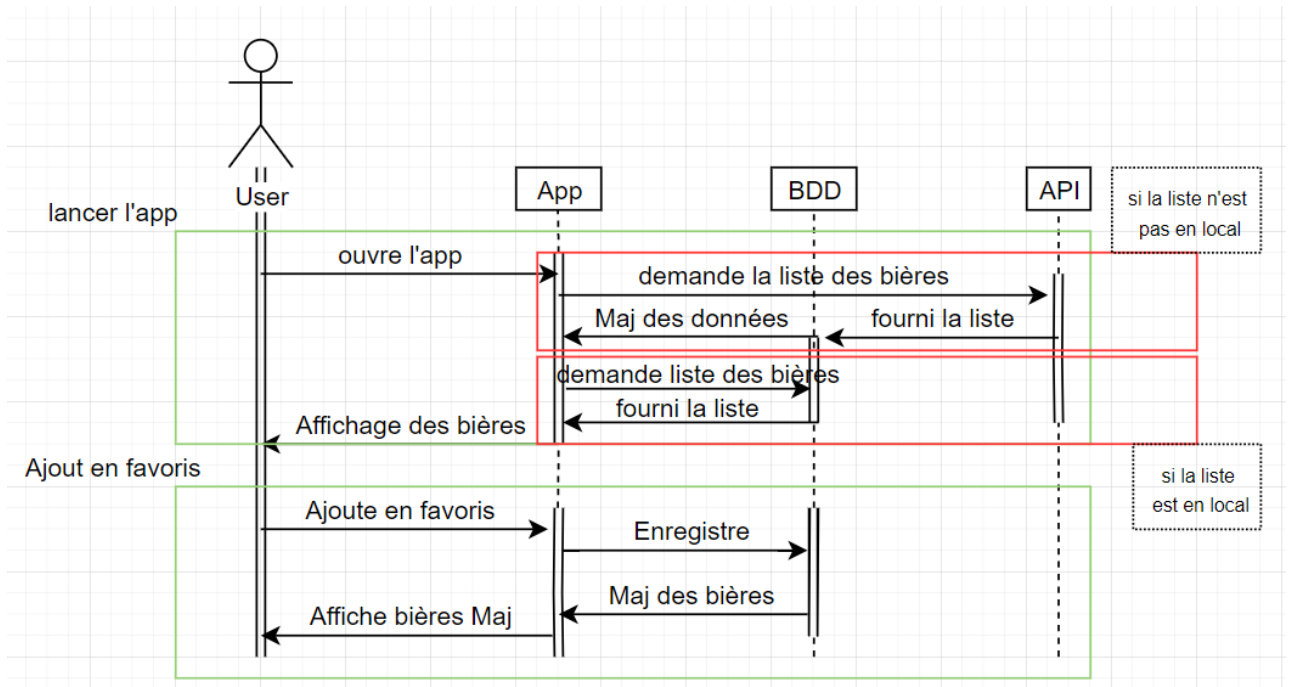
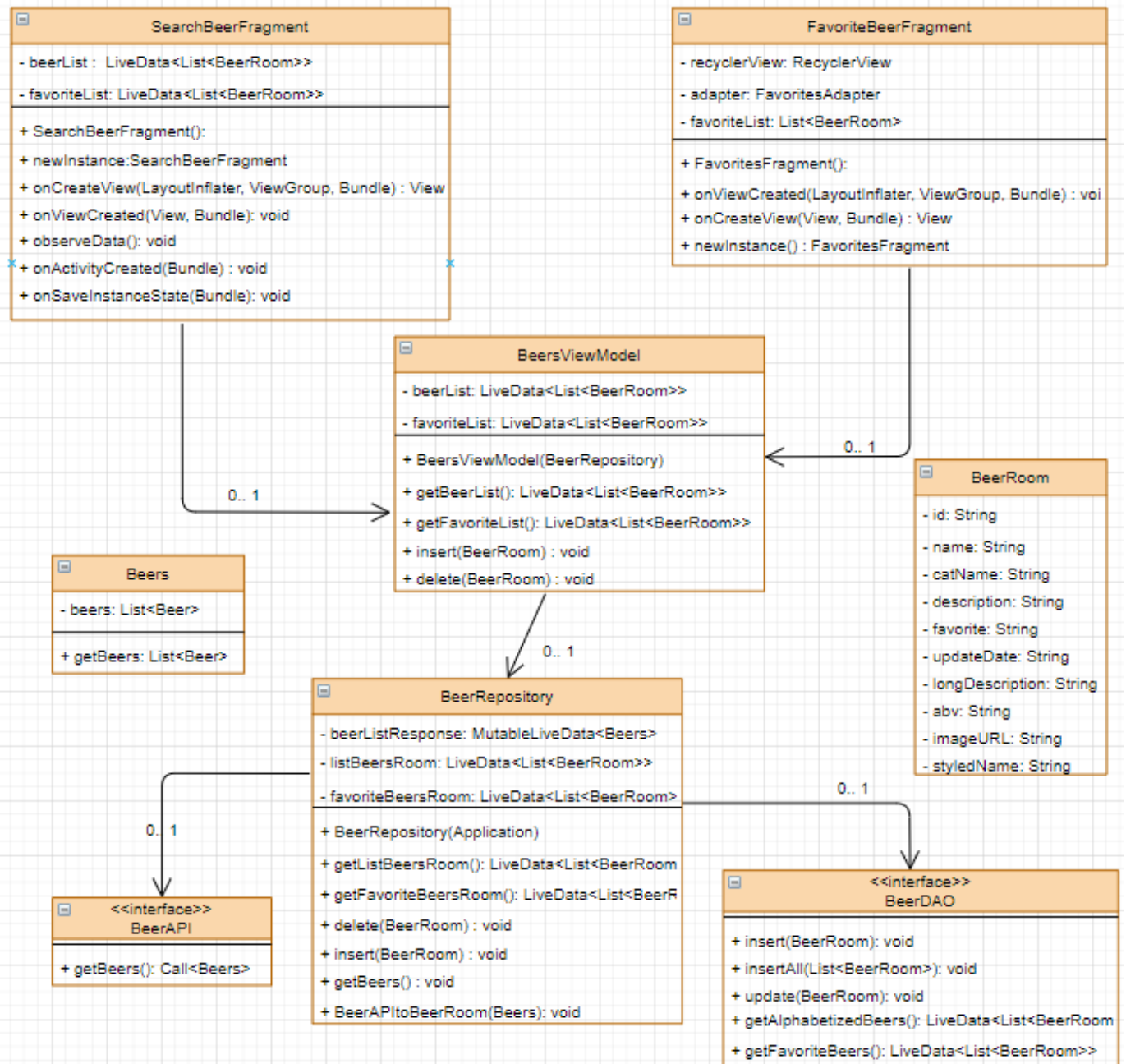


Diagramme de classe



Contrat d'interface

L'application permet de récupérer une liste de bières. On obtient cette liste grâce à l'API [BreweryDB](#).

Cette api contient beaucoup de données sur les bières et brasseries. Nous avons décidé d'exploiter les bières mais il est possible qu'on utilise les brasseries dans une prochaine version de l'app...

Voici l'url qu'on utilise : <https://api.brewerydb.com/v2/beers?key=yourkey>

Yourkey est la clé que nous avons obtenu par l'API, il faut en effet s'inscrire sur le site de breweryDB pour obtenir une clé. Cette clé nous donne accès à un jeu de données limité, il faut payer une licence pour avoir toutes les données.

Ici nous récupérerons la liste des bières sous un format JSON en requêtant l'url ci-dessus en GET :

```
"data": [
  {
    "id": "c4f2KE",
    "name": "Murican Pilsner",
    "nameDisplay": "Murican Pilsner",
    "abv": "5.5",
    "glasswareId": 4,
    "styleId": 98,
    "isOrganic": "N",
    "isRetired": "N",
    "labels": {
      "icon": "https://brewerydb-images.s3.amazonaws.com/beer/c4f2KE/upload_jjKJ7g-icon.png",
      "medium": "https://brewerydb-images.s3.amazonaws.com/beer/c4f2KE/upload_jjKJ7g-medium.png",
      "large": "https://brewerydb-images.s3.amazonaws.com/beer/c4f2KE/upload_jjKJ7g-large.png",
      "contentAwareIcon": "https://brewerydb-images.s3.amazonaws.com/beer/c4f2KE/upload_jjKJ7g-contentAwareIcon.png",
      "contentAwareMedium": "https://brewerydb-images.s3.amazonaws.com/beer/c4f2KE/upload_jjKJ7g-contentAwareMedium.png",
      "contentAwareLarge": "https://brewerydb-images.s3.amazonaws.com/beer/c4f2KE/upload_jjKJ7g-contentAwareLarge.png"
    },
    "status": "verified",
    "statusDisplay": "Verified",
    "createDate": "2013-08-19 11:58:12",
    "updateDate": "2018-11-02 02:15:14",
    "glass": {
      "id": 4,
      "name": "Pilsner",
      "createDate": "2012-01-03 02:41:33"
    },
    "style": {
      "id": 98,
      "categoryId": 8,
      "category": {
        "id": 8,
        "name": "North American Lager",
        "createDate": "2012-03-21 20:06:46"
      },
      "name": "American-Style Pilsener",
      "shortName": "American Pilsener",
      "description": "This classic and unique pre-Prohibition American-style Pilsener is straw to deep gold in color. Hop bitterness should be used. Malt flavor and aroma are medium. This is a light-medium to medium-bodied beer. Sweet corn-like dimethyl sulfide should be no chill haze. Competition organizers may wish to subcategorize this style into rice and corn subcategories.",
      "ibuMin": "25",
      "ibuMax": "40",
      "abvMin": "5",
      "abvMax": "6",
      "srmMin": "3",
      "srmMax": "6",
      "ogMin": "1.045",
      "fgMin": "1.012",
      "fgMax": "1.018",
```

Nous n'utilisons pas toutes les données fournies par l'API. Voici les champs qu'on utilise dans l'application :

Le champ « nom » représente le nom principal de la bière.

Le champ id est l'id de la bière.

cat_name représente le label de la bière c'est lui qui est affiché dans la liste.

Le champ update_date nous donne l'info dans les détails de la bière quand est ce que les infos de la bière ont été mises à jour pour la dernière fois.

Abv représente le pourcentage d'alcool dans la bière.

Le champ label représente les urls pour toutes les images de la bière.

Le champ name dans style représente la catégorie à laquelle la bière appartient.

Il existe d'autre route pour obtenir d'autres info, par exemple si on veut le détail d'une bière en particulier, on peut appeler l'url en remplaçant beers par beer et en rajoutant l'id de la bière :

<https://api.brewerydb.com/v2/beer/12345?key=yourkey>

Une fois le JSON récupéré nous utilisons Retrofit2 pour parser l'objet Json en objet Java.