RES508 - Qualité de développement Android - TP noté

Jocelyn CARAMAN

17 janvier 2025

1 Objectifs

Créer une application Android *Ma collection de cartes Pokémon*. Elle permettra d'afficher sa collection de cartes Pokémon sous forme de liste, de pouvoir les afficher individuellement en grand et d'*acheter* des boosters. Une collection de carte *de base* pour débuter sa collection sera disponible à l'achat.

2 Modalités

- Travail à réaliser en **binôme**.
- Le projet sera abordé durant le cours du vendredi 17 janvier et sera à rendre au plus tard le **dimanche 16 février 2024** à **23h59**.
- Le code du projet sera à rendre par l'intermédiaire de GitHub Classroom sur la classe nommée *IUT ACY 3 INFO - 2024-2025 - S508 - Qualité de développement (Android)*. Il vous faudra un compte GitHub. Un lien sera fourni et permettra de créer un groupe ou en rejoindre un existant. Merci de nommer votre groupe avec vos deux noms de famille séparer par un espace (exemple DUPONT-MARTIN EL HADI).

3 Besoins

3.1 Version minimale (requis)

- Dans cette version minimale, l'application ne contiendra que des cartes de type **Pokémon**. Les cartes de type **Énergie** et **Dresseur** ne sont pas considérées.
- Le code doit respecter les **recommandations d'architecture** vues en cours (architectures en couches).
- Le thème est **personnalisé**, l'interface est **soignée** et vous intégrerez **Material Design** dans le choix des composants.

3.1.1 Écran collection

- Implémentez un **écran** pour afficher la **collection de cartes Pokémon** possédée par l'utilisateur. L'affichage peut être sous forme de liste verticale ou en grille, dans les deux cas, on peut la faire défiler. Les informations à afficher au minimum pour chaque carte sont : son **nom** et son **illustration**. Vous pouvez par exemple afficher une version miniature de la carte (par rapport à l'affichage lors de la consultation de la carte) en faisant attention que le nom soit bien visible.

- À la **première ouverture** de l'application, la collection de cartes est **vide**. Un bouton *Acheter la collection de base* est présent au milieu de l'écran pour obtenir une "collection de base". Au clic, vous utiliserez la bibliothèque Retrofit pour télécharger un JSON représentant la **liste d'ids** des cartes à ajouter à la collection.

L'affichage est mis à jour avec les cartes ainsi obtenues.
⚠️ Attention de bien avoir des cartes dans votre base de données ou dans la liste définie dans votre code avec les mêmes ids.

— Au clic sur un **élément de la liste**, on bascule sur la vue **détaillée** de la carte. Plus de détails dans la partie *Écran consultation carte*.

— Un **bouton** *Acheter un booster* est également présent. Au clic dessus, **3 cartes aléatoires non-possédées** jusqu'à présent sont ajoutées à notre collection. La liste est **directement rafraîchi** avec les nouvelles cartes. Il n'y a pas de gestion de doublons. Ainsi, il faudra s'assurer de ne pas ajouter de cartes déjà possédées. De même, s'il reste moins de 3 cartes non-possédées par l'utilisateur, on lui donnera le maximum de cartes possible à l'achat. Le bouton est **désactivé** quand l'achat n'est plus possible (c'est-à-dire, plus de cartes non possédés).

3.1.2 Écran consultation d'une carte

— Implémentez un **écran** pour afficher la **carte en détails** sur toute la **largeur** de l'écran du téléphone. La carte ne doit **pas** être une image mais un **composable**. Ici on affiche toutes les données liées à la carte définis dans la table 1.

— Un **bouton** permettant de retourner à l'écran précédent (écran de collection) est visible. Utiliser une icône de flèche vers la gauche (icône standard pour ce comportement).

— Le **type du Pokémon** est affiché sous forme d'**icône** (pas de texte). Le **coût d'une attaque** et le **coût de retraite** sont également affichés sous forme d'**icône** (exemple : pour un coût de 3 énergies, j'affiche trois icônes "énergie").

Nom de la propriété	**Description**
id	identifiant de la carte, doit être unique entre toutes les cartes (un entier)
name	le nom de la carte
hp	le nombre de point de vie (un entier)
type	le type du Pokémon (feu, eau, poison...). Deux valeurs max. Utiliser une *enum class*
illustration	un lien URL d'une image illustrant cette carte
moves	une liste d'attaques du Pokémon. Au maximum 2. Détaillé dans la table 2
weakness	faiblesses : les types super effectifs sur le Pokémon
strenghts	forces : les types dont le Pokémon est résistant
retreatCost	coût en énergie de la retraite (sortir du plateau de jeu)

TABLE 1 – Modèle d'une carte de type Pokémon

3.2 Version avancée (fonctionnalité(s) à ajouter au choix)

— Ajoutez des **tests unitaires** qui font sens sur un ViewModel et/ou un Repository, **4 tests** au maximum.

— Utiliser la bibliothèque **Jetpack Navigation with Compose** pour gérer la navigation entre les écrans dans l'application.

Nom de la propriété	Description
id	identifiant de l'attaque (un entier)
name	le nom de l'attaque
description	description de l'attaque
cost	le coût en énergie. Pas de gestion du type d'énergie, on considère qu'il y a un seul type.
damages	un entier représentant les dégâts infligés par cette attaque

Table 2 – Modèle d'une attaque de Pokémon

— Réaliser un **écran** d'**ouverture de booster** quand on clic sur le bouton *Acheter un booster*. Cette écran affiche les 3 cartes au plus qui viennent d'être obtenues. Les cartes sont affichées **en grand** comme sur l'écran de consultation de la carte. Elles sont affichés dans l'ordre que vous voulez mais toujours **une par une**. En appuyant sur une carte, on affiche la carte suivante. Quand on appuie sur la dernière carte, on ferme l'écran d'ouverture de booster.

— Ajouter les **deux autres types** de cartes Pokémon : **Dresseur** et **Énergie**. L'affichage sera différent pour ces types étant donné que les champs disponibles sont également différents.

Nom de la propriété	Description
id	l'identifiant de la carte, doit être unique entre toutes les cartes (un entier)
name	le nom de la carte
category	la catégorie de la carte qui est soit un objet (OBJECT), soit un support (SUPPORTER). Utiliser une *enum class*
illustration	un lien URL d'une image illustrant accompagnant cette carte
action	ce que fait cette carte quand elle est joué
condition	les conditions d'utilisation de la carte

Table 3 – Modèle d'une carte de type Dresseur

nom propriété	Description
id	l'identifiant de la carte, doit être unique entre toutes les cartes (un entier)
name	le nom de la carte
illustration	un lien URL d'une image illustrant cette carte

Table 4 – Modèle d'une carte de type Energie

3.3 Version très avancée (fonctionnalité(s) à ajouter au choix)

— Implémentez la **persistance de données** pour sauvegarder votre collection de cartes. Vous utiliserez la bibliothèque **Room** vue en annexe du cours. Cela vous permettra de garder en mémoire les cartes que vous détenez déjà après avoir fermer l'application. Pensez bien à sauvegarder les cartes au moment de l'achat.
Note : dans ce cas, pour repartir de zéro et supprimer votre collection actuelle, vous pouvez supprimer les données de l'application dans

Paramètres > *Applications* > *Le_nom_de_votre_app* > *Données* ou *Stockage* > *Supprimer les données.*

— Un bouton pour **changer de tri** est présent sur l'écran de collection. Comme son nom l'indique, il permet de changer l'ordre d'affichage de la liste de cartes. 3 tris possible : par *nom*, par *type* puis par *nom* ou par *PV*.

4 Moyens

Le dépôt crée pour chaque groupe contiendra un code de départ pour éviter les erreurs de configuration.

Le JSON pour obtenir la liste d'ids de cartes Pokémons est disponible à cette adresse : https://raw.githubusercontent.com/Josstoh/res508-qualite-dev-android/refs/heads/main/rest/base-collection-ids.json.

Libre à vous d'héberger vous-même par d'autres moyens (comme votre propre dépôt GitHub) une autre liste d'ids.

clé JSON	Description
base_collection_ids	un tableau d'entier représenant les ids des cartes Pokémon correspondant à la colle

TABLE 5 – Spécification de l'objet JSON renvoyer par l'API

5 Livrable

Le code est rendu par l'intermédiaire de GitHub. Il ne sera pas possible de pousser des modifications après la date limite de rendu.

Il est attendu que le code :

— Soit clair et lisible ;
— Respecte au maximum les guides de style Kotlin (abordés dans la première partie du cours) ;
— Soit commenté quand nécessaire ;
— Ne crash pas (penser à la gestion des erreurs avec les blocks try/catch).