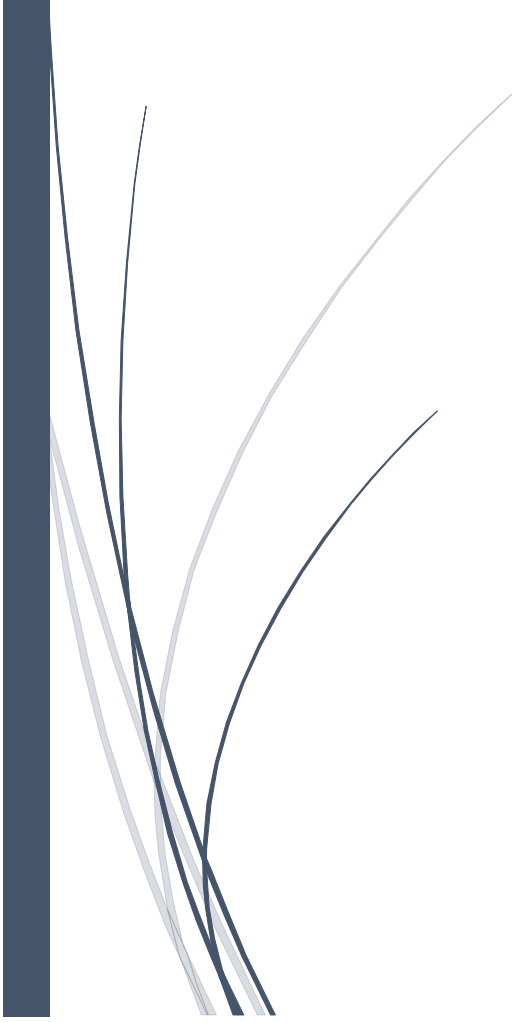
Android

Projet info306

PokeSearch

RAPPORT



BAILLY Lucas

SOMMAIRE

CONTENU

Internationaliser son application

Dans le dossier « res » se trouve un dossier « values » dans lequel se trouve un fichier strings.xlm dans lequel nous pouvons stocker des chaines de caractères liées à un ID. Afin que le texte entre autres, présent dans l'application soit disponible en plusieurs langue, il suffit d'ajouter un dossier « values-*préfixe_du_langage** » dans le répertoire « res », puis de créer un fichier « strings.xml » dans ce répertoire. Chaque chaine de caractère répertoriée dans le fichier strings.xml du dossier « values », devra aussi figurée dans le fichier strings.xml du dossier « values-*préfixe_du_langage* » et on veillera que la traduction soit bien effectuée. Par défaut, on ajoutera tous les textes anglais dans le dossier « values ». Pour récupérer ces textes, on utilise : "@string/*id*". Enfin, le système affichera automatiquement le texte traduit suivant les paramètres de l'appareil, lors du lancement de l'application.

*« values-fr » pour y ranger des chaines de caractères en français.

Les Ressources

Les strings ne sont pas la seule ressource que j'utilise. J'utilise aussi des couleurs, et créé un style propre à mon application. De plus j'utilise les icon créé dans le dossier drawable afin d'enjoliver mon menu de navigation de ma mainActivity par exemple.

Les Toasts

J'utilise des Toast à divers endroits notamment pour prévenir de la capture de tous les Pokémons disponibles dans la PokéMaps (activity3). Leur utilisation est simple :

Toast.makeText(context, "text ", durée).show()

Les Menus

Tout d'abord, il est possible d'associer une icone à un menu. Pour ce faire, il faut créer un nouveau Vector Asset dans le répertoire « *res/drawable/ ».* Un sous-écran assez intuitif s'ouvre permettant de créer choisir l'icone souhaité et son nom facilement. Ensuite pour créer des menus, il faut d'abord créer un nouveau répertoire android que l'on nomme menu, dans le répertoire « *res/* ». Ensuite on créer nos menus « menu.xml » par exemple, dans ce dossier « menu ». Pour cela on utilise différentes balises item/menu en prenant soins de hiérarchiser les éléments. Un item peut contenir un menu de plusieurs item par exemple. On récupère l'icone souhaité avec : « android:icon='@drawable/*nom_Icone'* ». Enfin pour ajouter ce menu dans la barre des statuts d'une activité, on récupère notre menu via la fonction inflate(R.menu.*nom_menu*), sur une var ManuInflater, le tout dans la fonction onCreateOptionMenu(Menu menu) de notre activité. Si l'on veut associer des évenements en fonction des items sélectionnés de notre menu, il suffit d'implémenter la fonction onOptionItemSelected(MenuItem item).

Les Fragments

Afin de manipuler des fragments de manière dynamique, j'ai choisi de créer un menu afin de naviguer entre plusieurs fragments à l'intérieur d'une même activité. De la même manière que le menu situé dans la barre des statuts, j'ai d'abord créé mes icones dans le dossier « res/drawable/ ». J'ai ensuite créé un menu « res/menu/bottom_navigation.xml » et dans ce menu j'ai ajouté 3 items associés aux 3 icones que j'ai créés. Ensuite j'ai ajouté un conteneur FrameLayout de le fichier.xml de mon activité afin d'un accueillir mes futurs fragments. Puis, j'ai ajouté mon menu en bas de mon activité en utilisant la balise « <com.google.android.material.bottomnavigation.BottomNavigationView> » et en récupérant mon menu avc « app:menu="@menu/bottom_navigation" ». J'ai ensuite créé 3 fragments (ex : « res/layout/fragment_home.xml » avec « HomeFragment.java » situé dans le même dossier que mes activités.java). Je fais le liens entre le .java et le .xml dans le onCreateView(…) de mes fragment.java. Dans mon activité.java, je récupère mon menu via un findViewById. Ensuite je crée un évènement (« navListener ») pour afficher tel ou tel fragment selon l'item sélectionné. Enfin, il faut que je fasse en sorte qu'un fragment s'affiche lors du lancement de l'application. Pour cela j'ajoute : « getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new HomeFragment()).commit(); » également dans mon onCreate() .

Shared Preferences

Dans mon HomeFragment, j'ajoute la possibilité de choisir entre un mode clair et un mode sombre via 2 boutons. Pour changer le thème de l'application j'utilise « AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO/YES) ». J'ai créé un style dans le dossier values qui utilise des couleurs définies à la fois dans le dossier values mais aussi dans le dossier values-night. De ce fait, si le mode nuit est définit (YES) alors les couleurs du dossier values-night seront utilisées, dans le cas contraire, se sont les couleurs du dossier values qui sont utilisés. Pour sauvegarder les préférences de l'utilisateur, j'utilise le concept de Shared Preferences. En effet, j'instancie un objet de type SharedPreferences.Editor sur lequel j'appelle la fonction putBoolean. Pour mon cas, j'utilise la clé « night-mode » et la valeur true si le mode sombre est choisi. Puis j'appelle la méthode commit() sur cet editor. Enfin dans le onCreate je récupère la valeur du sharedPreferences ayant l'id « night-mode » via la méthode getBoolean. Si la valeur vaux true, alors j'active le mode sombre dès la construction.

Bundle

Afin de rester dans le même fragment de l'activité 1 lors d'une rotation d'écran, j'utilise la méthode onSaveInstanceState(Bundle ouState). Dans cette méthode, je récupère l'id de l'item sélectionné de ma barre de navigation avant la rotation avec outState.putInt("x", bottomNav.getSelectedItemId()) (bottomNav étant mon menu me servant de barre nav). Une fois la rotation effectuée, je récupère l'id précédemment enregistré via savedInstanceState.getInt("x") (avec x la clé me de l'élément enregistré). En fonction de l'id récupéré, j'affiche tel ou tel fragment de la même manière que le fait le onItemSelected() de mon bottomNav.

Broadcast Receiver

Dans le fragment « About » ou « À Propos » (SearchFragment) j'affiche le pourcentage de batterie de l'appareil. Pour cela je crée d'abord un textView dans le fichier xml dont je récupère l'id dans le fichier java. J'utilise ensuite un BatteryReceiver, class que j'ai moi-même implémenté afin d'utiliser un Broadcast Receiver utilisant la batterie de l'appareil. Le constructeur prends en paramètre un objet de type textView, dont je set le text dans le onReceive (je récupère me pourcentage via intent.getInExtra(« level », 0)). Ensuite, je créé mon objet BatteryReceiver dans le SearchFragment qui set par la même occasion le textView. Enfin « j'inscris » le batteryReceiver dans le onStart et le « désinscris » dans le onStop().

Base de données SQLite

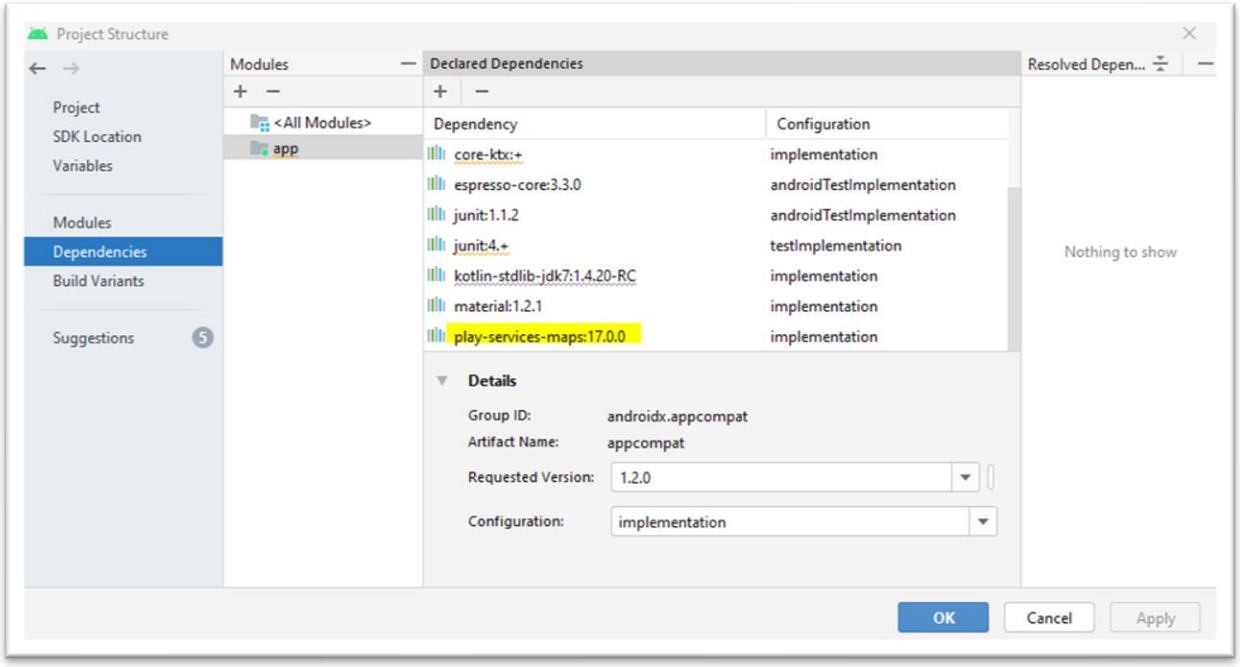
Pour créer cette base de données, Je crée une classe *DatabaseManager* dans laquelle on trouvera toutes les requêtes SQL nécessaire. Dans le *OnCreate* de cette classe, je créé et remplie cette BDD avec comme version initiale 1. Pour modifier cette BDD à l'avenir, il me suffit de changer le n° de version et d'exécuter mes requêtes SQL dans le *onUpgrade.* Le procédé est toujours le même, on stock notre requête SQL dans une variable String puis on appelle la fonction *execSQL* sur l'objet *SQLiteDatabase*, en plaçant notre variables String en paramètre de cette dernière. Pour remplir cette BDD, on utilise dans le fichier.java de l'activité souhaité, la fonction *insertPokemon (name, type, estCapturé)*, sur la variable *DatabaseManager* instanciée dans cette dernière activité. Enfin pour récupérer chaque Pokémon de la BDD afin de pouvoir l'afficher, je crée une Classe PokemonData qui me permet de construire un Pokemon à de la même façon que dans la BDD. Ensuite, je crée une fonction readPokemon dans ma classe DatabaseManager qui me permet élément par élément, grâce à une variable de type Cursor, récupérer les attributs de chaque Pokemon présent dans la BDD pour ensuite en créer autant via le constructeur de PokemonData. Ensuite, cette fonction permet de ranger tous les PokemonData créés dans une List<PokemonData> et de return cette liste. Enfin il faut récupérer cette liste en appelant la fonction readPokemon() sur notre variable DatabaseManager dans le fichier.java dans lequel on veut afficher notre BDD. On peut récupérer ensuite les diverses données de chaque Pokémons via les getters présent dans la classe PokemonData. Enfin il suffit plus que de manipuler plusieurs variables de type LinearLayout ou encore TextView dans une boucle pour les afficher les données récupérées de la manière que l'on veut.

Géolocalisation

Etape 1 : demander les droits d'utiliser la position

Etape 2 : Afficher la carte

- Télécharger la librairie google maps et activer les droits d'accès dans l'API Google



- Vue :

Création du fragment map dans la vue liée à ma classe

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</fragment>
```

Nom de la classe de fragment que l'on veut insérer

- Récupération du fragment créé dans le OnCreate() :

```
FragmentManager fragmentManager = getFragmentManager();
mapFragment = (MapFragment)fragmentManager.findFragmentById(R.id.map);
```

- On load la map

Via la méthode loadMap(), qui est appelée dans checkPermission() uniquement si l'utilisateur a donné son accord. De ce fait, on ne revérifie pas les permissions de géolocalisations et on supprime manuellement les Warnings.

- Markers

Dans la méthode onMapReady, je créé dans une boucle, un nombre limité de markers avec googleMap.addMarker(), googleMap étant l'Objet googleMap de mon activité. Je place les configurations de chaque marker en paramètres de la méthode addMarker. Une liste pokemonCapturable récupère tous les pokemons non capturés via la méthode readPokemonsAttrapable() du databaseManager. Ensuite dans le .title j'attribu le nom Marker en choisissant aléatoirement un pokemon capturable de ma liste, puis j'appelle le getName() de la classe PokemonData sur ce Pokemon, afin d'obtenir son nom (String). En fonction de son nom, j'affecte tel ou tel BitmapDescriptor dans le .icon. Sachant que mon BitmapDescriptor est construit via la méthode bitmapDescriptorFromVector qui prend en paramètres le context et une icon du dossier drawable. J'ai donc créé au préalable différentes icons dans ce dossier que je récupère via R.drawable.***ic***_*pokemon_torterra* par exemple***.*** Enfin leurs coordonnées sont choisis une à une via une List de LatLng que l'on injecte dans le .position. La liste des pokemons capturable est actualisé chaque fois que l'on lance la map. Ainsi, on aura accès uniquement aux pokemons non Capturés dans la Maps. Si jamais plus aucun Pokemon n'est disponible (si la liste est vide) alors émet un Toast nous avertissant qu'aucun Pokemon n'est disponible, et aucun Marker n'est créé. Afin de pouvoir attraper un Pokemon (toucher un Marker) j'appelle la méthode setOnMarkerClickListener qui prend en paramètre un GoogleMap.OnMarkerClickListener sur ma googleMap. La méthode onMarkerClick de la méthode OnMarkerClickListener calcule d'abord la distance entre l'utilisateur et le Marker touché via la méthode getDistance. Ensuite si le Marker est asser près, on modifie la base de données pour capturer le pokemon via la méthode setEtatOn(marker.getTitle()) du databaseManager, on rend le Marker invisible, et on ajoute un toast nous confirmant de la capture. Dans le cas contraire, un Toast nous indiquant que le pokemon est trop loin intervient.

Notification

J'ai décidé d'ajouter une notification qui nous prévient lorsque l'on attrape un Pokémon. Sachant que l'on attrape un Pokémon en touchant un Marker, j'ai simplement ajouter le code de la notification dans le onMarkerClick que j'attribue à chaque Marker que je créé. Afin de créer la notification, je crée un objet de type NotificationCompat.Builder sur lequel j'appelle différente méthode me permettant de personnaliser la notification (ex : j'affiche le nom du Pokemon Capturé dans le ContentText via builder.setContentText(marker.getTitle())). Enfin je crée la variable notificationManagerCompat de l'activité en question (Activity3) sur laquelle j'appelle la fonction notify, prenant en parametre la méthode build() appelé sur notre builder précédemment créé.

.

CONCLUSION

Ce Projet d'info306 m'a permis de faire mes premiers pas dans la programmation Android. L'apprentissage du java en L1 m'a permis d'avoir quelques base pour ce projet, et le projet HTML m'a lui aussi donné certains repères vis-à-vis des fichiers XML.

Néanmoins j'ai aussi rencontré certaines difficultés notamment lors de l'utilisation du « Shared Preferences ». En effet, une fois le thème sombre sélectionné, si nous quittons l'application avant même d'avoir lancé une autre activité, alors les préférences de l'utilisateur ne seront pas prises en compte lors de la prochaine utilisation.

De plus, mon application est loin d'être parfaite. En effet, il est possible de l'améliorer en ajoutant une fonction générant des coordonnées aléatoires pour chaque Marker afin de ne pas choisir aléatoirement des coordonnées définies statiquement dans une Liste de coordonnée. Enfin, il est surement possible d'intégrer aussi des fonctionnalités réseaux permettant d'échanger des Pokémons.