WARFRAME TRACKER

Benjamin VANDENBUSSCHE

13152

Rapport Applications Mobiles

Objectifs de l'app

Warframe est un jeu qui contient des alertes et des invasions. Ceci sont des missions qui ne sont disponibles que quelques heures et qui peuvent fournir des récompenses qui sont très rares, voir, ne peuvent être obtenus que dans des alertes de ce types (‘Nitain extract’ -exclusif ‘Forma’ -disponible une fois par jour seulement). Mais même si les récompenses peuvent être rare cela ne veut pas dire que les alertes ou les invasions le sont. En général il n’y a que quelque mission par jour qui sont intéressantes et il faudrait régulièrement vérifier les alertes et invasions disponibles pour être sûr de ne rien rater d’intéressant.

Le but de Warframe tracker était de créer une app qui pourrait avertit l’utilisateur lorsqu’une alerte ou une invasion spécifique à eu lieu. L’utilisateur pourrait rajouter des filtres selon le niveau de la mission et/ou la récompense de la mission. Une fois fait cela se rajouterais à la liste de filtres et à chaque fois qu’une alerte ou une invasion correspond au filtre l’utilisateur serait prévenu.

Au départ l’idée était d’avertir l’utilisateur grâce à une notification hors app mais sous conseils de l’enseignant il a été décidé de garder les notifications à l’intérieur de l’app car il serait difficile d’implémenter les notifications hors app et en plus de cela, ça imposerait un cout sur la batterie (et personnellement je ne suis pas intéressé par une telle app)

Figure 1. Example d'alertes et de récompenses potentiellement intéressantes

L'information donné ci-dessus décrit le but principal de l'application mais lors de la phase recherche de du projet, j'avais trouvé une multitude d'API intéressante qui aurais pu améliorer le projet. Une des idées était de créer une librairie de toutes les récompenses possibles ou des schémas de construction qu'on pourrait vendre sur le marché en ligne. Avec cela il aurait aussi été intéressant de rajouter le prix de vente de ce qu'on pourrait vendre.

Une autre idée possible était de rajouter un écran pour la manipulation d'extracteurs. Ceci était plus quelque chose à considérer plutôt qu'une idée sérieuse. Je ne vais pas rentrer trop en détail mais il serait possible en théorie de manipuler les extracteurs du jeu en utilisant des bouts de codes de l'application officielle de warframe sur github. Ceci cependant, n'a pas été approuvé par le studio de jeu (Digital Extremes), et tous compte utilisant ce genre de technique prend le risque d'être bannis. J'avais cependant laissé l'idée sur la liste par curiosité, j'aurai pu tester l'idée sur un personnage se trouvant sur un compte secondaire. D'autres idées potentielles peuvent être trouvées sur ReadMe.md du repository github[1]

[1] https://github.com/benjvdb9/Warframe-Tracker

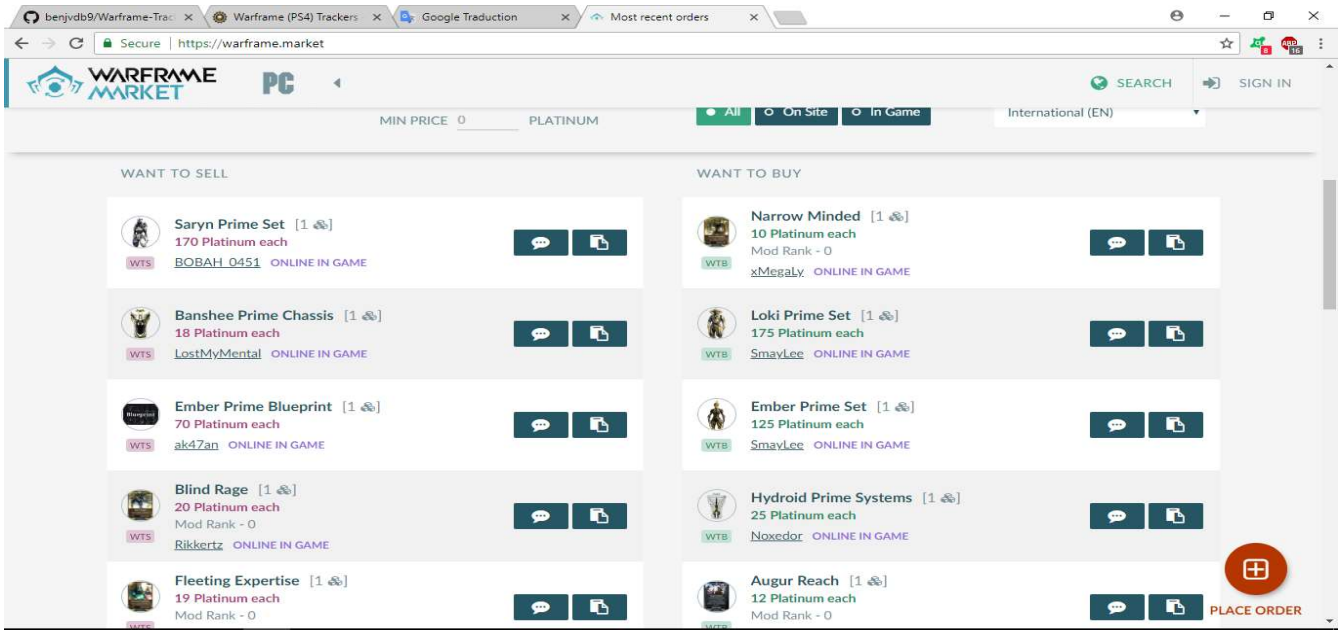


Figure 2. L'économie de Warframe: On farm ce qui est en haute demande et on le revend à haut prix sur le marché

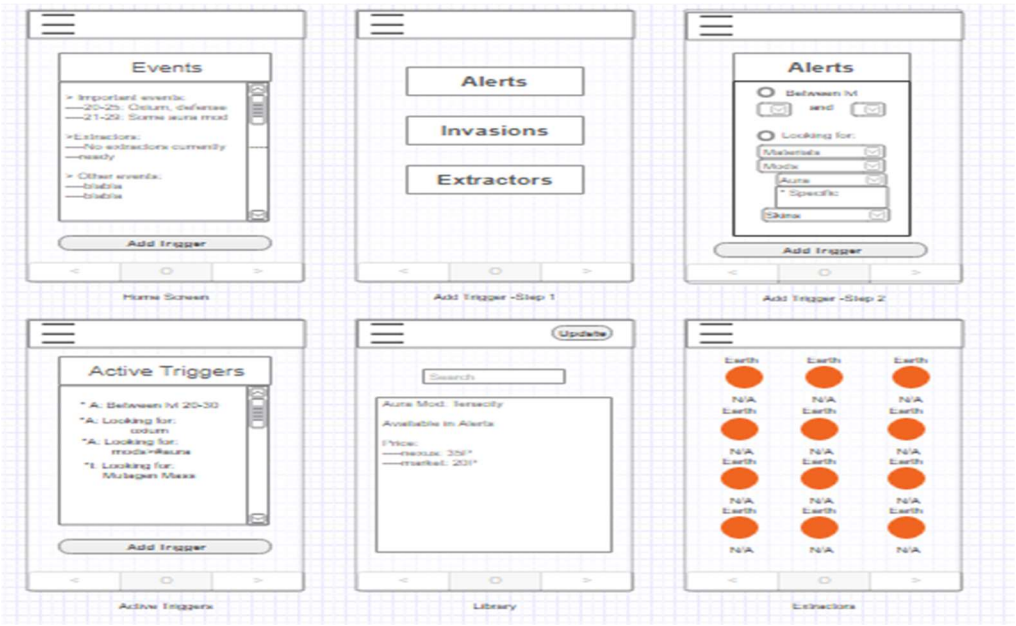


Figure 3. Le mockup initial

Le développement

J’avais choisi de coder le projet en Kotlin pour apprendre le langage. Ceci à emmener ses propres problèmes ici et là mais je pense qu’en fin de compte c’était plus bénéfique que désavantageux. Le problème principal était de trouver de bonnes références pour commencer et de trouver de sources d’informations pour Kotlin spécifiquement, mais après quelques jours cela n’était plus un problème et Kotlin étant assez populaire il était relativement facile de trouver de l’aide en ligne.

Tout d’abord j’avais commencé par la partie XML, en créant les 4 premières vues de l’app. Il était un peu compliqué au début de comprendre comment les ConstraintLayout fonctionnaient et comment placer des composants à l’endroit exact ou je voulais sans qu’ils soient déplacés automatiquement. Pour les écrans de notifications et de filtres j’ai utilisé des ScrollView pour m’assurer que toutes les notifications soient toujours visibles.

Et pour l’écran qui sert à créer une nouvelle alerte j’ai dû emboiter plusieurs Views les unes dans les autres pour m’assurer que le résultat ressemblait à ce à quoi je m’attendais. Dans les images ci-dessous on voit que le ConstrainLayout qui représente l’écran et contenu dans un DrawerLayout. Ceci et dut au fait que l’application utiliser une NavigationView, un menu de navigation qui nous permet de passer d’une page à l’autre facilement. La NavigationView se comporte un peu comme un deuxième écran hors de porté du premier écran et donc les deux écrans sont placés dans un DrawerLayout.

Si on continue à analyser cet écran-ci on peut noter que sous le ConstraintLayout il y a une toolbar. Ceci est du au fait que comme pour ouvrir le menu de navigation il nous faut un bouton, il faut placer un bouton sur le toolbar et pour cela il faut donc remplacer le toolbar de base et un placer un manuellement. Après ça il y a une ScrollView qui contient un LinearLayout pour ordonner les composantes. Dans le LinearLayout j’ai aussi utilisé un ConstraintLayout lorsque je voulais mettre plusieurs TextView sur une ligne. Si la TextView est une variable je laisse une valeur par défaut et dans le cas opposé, le champ réfère une ressource dans res/values/string.xml, là où se trouve toutes les chaines de charactères statique. Et finalement j’ai créé un style spécial que j’ai rajouter au bouton tout en bas pour arrondir les bords.

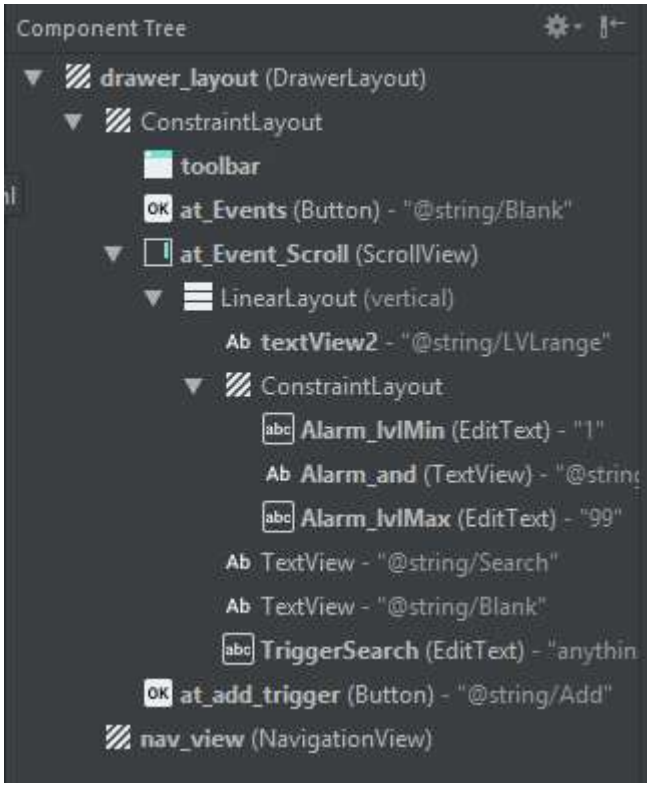


Figure 4 & 5. La structure de add_trigger_2

Il est aussi intéressant de noter que le block at_Events ne contient rien. Ceci est dû au fait qu'on utilise cette vue aussi bien pour les alarmes que les invasions.

Une fois toutes les vues crées j'ai aussi créé un groupe dans le dossier res/menu. Ce groupe contient le contenu et les icones utilisées dans le menu de navigation et il est utilisé dans le paramètre app:menu du widget NavigationView de chaque vue.

Après les vue XML j'ai commencé l'écriture des activités et tout ceci était écrit en Kotlin. Il y a 6 classes en tout 5 classes utilisant les 4 vues et une class utilité qui en fin de compte n'a pas vraiment été utilisé. Pour chaque activité, il faut définir les actions effectuées lors d'un clic sur une des options du menu de navigation. Au début, comme cela devait être partout la même chose, je comptais définir la fonction dans la classe utilité et la réutiliser pour toutes les autres classes qui utilisait le menu de navigation. Cependant, comme cette fonction dépendait du contexte il n'était pas possible de faire ça, même lorsque j'avais rajouté le contexte en tant que paramètre.

En fin de compte, comme j'avais perdu pas mal de temps sur ça j'ai décidé d'abandonner cette partie et de tout simplement copier-coller la fonction pour chaque activité. Par contre si je devais créer un nouveau projet avec un menu de navigation ou si je devais refactorer mon code je créerais probablement une classe héritant des classes que toutes les activités utilisaient (AppCompatActivity, NavigationView.OnNavigationItemSelectedListener) et je définirais la fonction qui est répété partout. Ensuite je ferais hériter toutes les activités de cette classe plutôt que des deux classes mentionnées ci-dessus. Bien sûr, il est probable que cela vienne avec ses propres problèmes comme la nécessité de définir certaines fonctions (onCreate par exemple) mais normalement ceci ne devrait pas poser de gros problèmes est rendre le code beaucoup plus lisible en générale et éliminerais pas mal de copier-coller.

Ceci est quelque chose que j'avais utilisé lors de l'écriture de add_trigger_2b. La classe add_trigger_2 est la classe qui utilise la vue qu'on avait utilisé plus haut. Cette page permet de créer un filtre pour une alarme mais il faut aussi pouvoir créer un filtre pour les invasions. Ceci est pour add_trigger_2b. Mais comme la vue de la face et même son comportement est extrêmement similaire il aurait été dommage de ne pas réutiliser du code existant. J'ai donc fait hériter add_trigger_2b de add_trigger_2 et fait un override des fonctions ou cela était nécessaire, comme la fonction qui définissait le texte du titre de la page.

Pour ce qui est de l'application elle commencerait sur un page d'accueil (MainScreen.kt) affichant les notifications. Le menu de navigation aurait emmené au 3 autres pages et il y aurait eu un bouton pour ajouter un filtre. Ceci nous emmènerait vers un autre écran (add_trigger_1) qui nous permet de choisir le type de filtre à rajouter : alarme ou invasions. Appuyer sur l'un d'entre eux nous amènerait soit à add_trigger_2 ou add_trigger_2b et ici on pourrait choisir les options du filtre.

Le niveau peut min au max peut être définit ici et / ou le type de récompense peut être spécifié. Ceci ce fait grâce à un TextView. Un des problèmes avec le code telle qu'il est, est que lorsqu'on essaye d'écrire dans le TextView le clavier virtuel cache le contenu du TextView et ont doit donc écrire à l'aveugle. Après avoir entré les données at avoir rajouter le filtre le filtre et sauvegardé dans un fichier texte et on est renvoyé vers la page contenant tous les filtres. Ici le ScrollView affiche le contenu de ce fichier et il y a un bouton qui permet d'effacer le contenu du fichier.

Le développement de l’application cependant ne s’est pas déroulé comme prévu. Tout d’abord il à fallut beaucoup de temps. Beaucoup de temps à été perdu par exemple en essayant de travailler avec la librairie okhttp. EN sois l’erreur étais que j’avais entre autres importé la librairie en tant que fichier jar mais que j’avais aussi rajouté la ligne ‘compile ‘…’’ Dans le fichier build.gradle et cela générai des conflits mais le rapport d’erreur ne disait pas où l’erreur étais généré ou ce qui la générai.

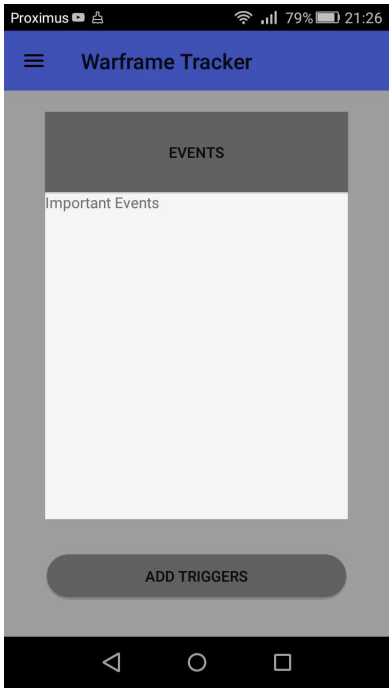
En général tout ce qui étais debugging a été un gros problème au cours du projet. Comme je ne savais pas comment correctement debugger je me suis débrouillé en utilisant LogCat et en laissant des messages ici et la mais ceci pouvait dire qu’il fallait des fois beaucoup de temps avant que je puisse trouver la source d’une erreur. Il y a aussi le fait que lors de certaines erreurs le programme ignorait tout simplement l’erreur et quelle ne ce voyait même pas dans le menu debugging.

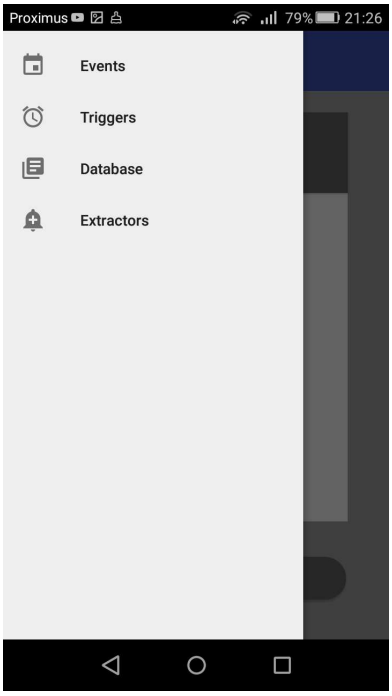
Ce n’est que vers la fin du projet que j’ai découvert les breakpoint et watches dans android studio qui m’ont aidé à comprendre ce qui n’allait pas avec mes JSONObjects.

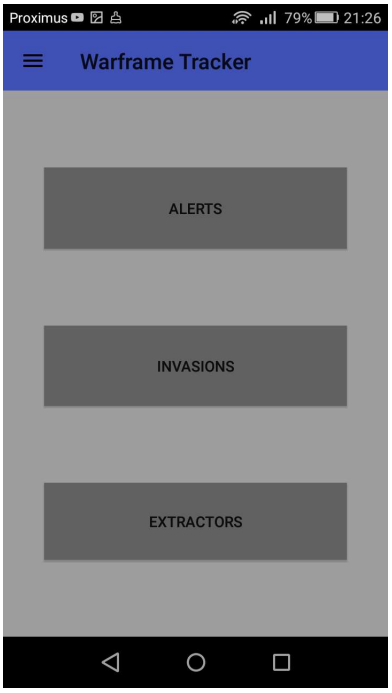
L’erreur avec les JSONObjects par exemple était une erreur lorsque je voulais convertir un String vers un JSONObject utilisant la librairie klaxxon. Cela ne marchait pas du tout et comme je n’avais aucun feedback je ne comprenais pas le problème. En fin de compte j’ai réalisé que le problème était que je ne devais pas le convertir vers un objet JSONObject, mais bien vers un JsonObject : une classe propre à la librairie klaxxon.

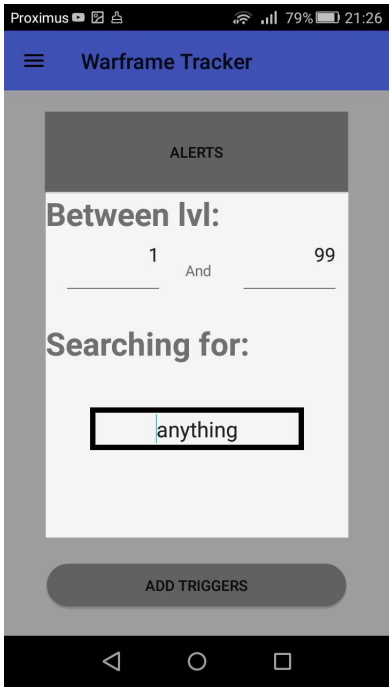
Dû à des erreurs de ce genre et une mauvaise estimation du temps de complétion du projet, l’application n’est pas capable de filtrer les données comme prévus, à la place elle devrait afficher des infos sur les alertes disponibles.

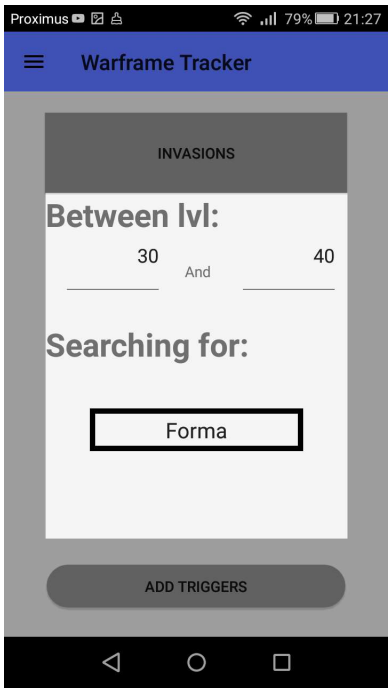
Voici quelque screenshots de l’app. Le code, lui, est disponible sur ‘https://github.com/benjvdb9/Warframe-Tracker’

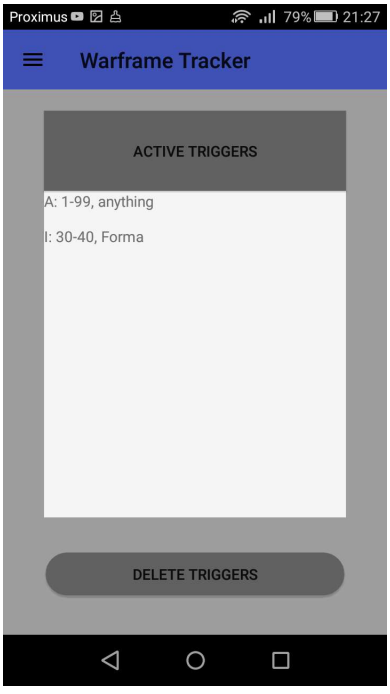

Proximus 79% 21:26
Warframe Tracker
EVENTS
Important Events
ADD TRIGGERS


Proximus 79% 21:26
Events
Triggers
Database
Extractors


Proximus 79% 21:26
Warframe Tracker
ALERTS
INVASIONS
EXTRACTORS


Proximus 79% 21:26
Warframe Tracker
ALERTS
Between lvl:
1
And
99
Searching for:
anything
ADD TRIGGERS


Proximus 79% 21:27
Warframe Tracker
INVASIONS
Between lvl:
30
And
40
Searching for:
Forma
ADD TRIGGERS


Proximus 79% 21:27
Warframe Tracker
ACTIVE TRIGGERS
A: 1-99, anything
I: 30-40, Forma
DELETE TRIGGERS