UNIVERSITÉ DE MONTRÉAL

Rapport IFT 3150

PAR

ÉMILE LABBÉ

TRAVAIL PRÉSENTÉ À GUY LAPALME

ET SYLVIE HAMEL

DANS LE CADRE DU COURS IFT 3150

AVRIL 2018

Introduction

Le projet MONA est une application mobile devant permettre à l'utilisateur de découvrir l'art public sous forme de jeu. Le développement du projet a débuté dans le cadre du cours IFT3105 par Daniel Jimenez et Lena Krause en 2015. Le projet consiste à développer une application mobile qui sera partagée via l'App Store (iOS) et Google Play(Android) où les utilisateurs seront encouragés à découvrir les différentes œuvres d'art publiques de Montréal sous forme de chasse aux trésors. L'application aura pour principale fonctionnalité d'identifier les différentes œuvres à travers Montréal et de permettre aux utilisateurs de prendre en photo les œuvres trouvées ainsi que de partager celles-ci sur les réseaux sociaux s'ils le souhaitent. L'application devra utiliser la localisation GPS afin d'aider l'utilisateur à s'orienter et récolter des données sur l'utilisateur avec son consentement à des fins de recherche. L'objectif académique de l'application est de mesurer l'intérêt de la population pour l'art public. Trois étudiants en informatique ont travaillé sur ce projet dans le cadre du cours 3150. Vincent Beauregard a travaillé sur le côté serveur, Paul-Philippe Chaffanet sur le côté iOS et moi-même sur le côté Android.

Résumé de mon travail

Ma partie de travail était de développer la version Android de l'application en implémentant les fonctionnalités demandées tout en respectant les normes de développement Android. Mon travail se divise en six parties. Chaque partie du travail sera expliquée avec davantage de détails.

1. Choisir comment réaliser l'application
2. Se familiariser avec le code source
3. Adapter la gestion de données au serveur et au développement des nouvelles fonctionnalités.
4. Simplifier le code source
5. Implémentation des nouvelles fonctionnalités
6. Correction de bugs

Choisir comment réaliser l'application

La première partie du travail a été de choisir comment je souhaitais réaliser l'application. Cette partie du travail a été effectuée en équipe avec Paul et Vincent. Nous devions principalement décider dans quel langage nous allions développer notre application. J'ai essayé brièvement le framework Kivy. J'ai fait une petite application mais Paul et moi nous sommes aperçus qu'il y avait très peu de documentation lorsque nous rencontrions des problèmes. J'ai rencontré plusieurs problèmes en essayant d'implémenter la localisation GPS. Nous avons par la suite essayé React Native qui semblait intéressant puisqu'il permet le développement Android et iOS de manière simultanée. Cependant, je n'ai pas été capable de faire rouler une application sur un émulateur et Paul n'a pas réussi à faire résoudre un bug avec la version Android. Nous avons donc décider d'utiliser les SDK natifs.

Une fois le choix d'utiliser les SDK natifs pour chaque côté, j'avais le choix d'utiliser le code de Lena qu'elle avait réalisé en 2015 ou de commencer une nouvelle version. Puisque Lena semblait assez satisfaite du travail qu'elle avait réalisé, j'ai choisi de travailler à partir de sa version. Cette première partie du travail a été réalisée du 4 janvier au 18 janvier.

Se familiariser avec le code source

Puisque j'ai choisi de baser mon travail sur la version de l'application développée par Lena et Daniel en 2015, la deuxième partie de mon travail a été de me familiariser avec le code de la version précédente de l'application. Dans le contexte du développement Android, une activité est une instance qui représente un écran de l'application. Un fragment est un élément contenu par une activité qui peut prendre une taille variable sur l'écran. Le code source était très complexe en puisque plusieurs fonctionnalités étaient implémentées en double. La plupart des fragments était codés de manière séparé en activité et pour utiliser

une fonctionnalité, parfois le fragment était utilisé alors que d'autre fois c'était l'activité. J'ai eu plusieurs difficultés à comprendre la gestion de données. Le code source utilisait une base de données SQL qui était utilisée et modifiée par pratiquement toutes les classes ce qui rendait difficile à comprendre les responsabilités de chaque classe. Le code source avait également plusieurs bugs tel qu'il fallait une activité de lancement pour l'application au lieu de pouvoir lancer l'application directement. Un autre problème du code original était que la navigation de la carte était problématique en raison de l'implémentation des swipes entre les différentes fonctionnalités. Cette partie du travail a débuté le 19 janvier.

Organisation des données

Dans le code source, les données des œuvres d'art étaient téléchargées à chaque fois que l'application était lancée avec une connexion internet et les données étaient conservées dans une base de données SQLite. À chaque activité ou fragment, les données étaient récupérées à l'aide d'un curseur. Les développeurs du code source avaient eu beaucoup de problème avec la gestion de ces curseurs et avec la modification de cette base de données. En raison de ces problèmes, ils n'avaient pas réussi à implémenter le tri des œuvres en fonction de divers paramètres dont la distance. Le code fourni obtenait les données directement du fichier du Bureau d'Art public de Montréal.

Comme il était souhaité d'intégrer les murales à l'application, j'ai dû changer la source des données. Au lieu d'aller chercher le fichier sur le site du Bureau d'Art Publique de Montréal, l'application prend ses données d'un fichier JSON situé sur le serveur du DIRO qui contient les données de l'ancienne source en plus des données des murales. J'en ai profité pour implémenter les différents scénarios possibles au lancement de l'application. Les quatre scénarios possibles sont :

1. L'appareil possède une connexion internet avec des données des œuvres d'art à jour

2. L’appareil possède une connexion internet sans données ou avec des données dont la date de modification est plus ancienne que la date affichée sur le serveur
3. L’appareil n’est pas connecté à internet mais possède des données
4. L’appareil n’est pas connecté à internet et ne possède pas de données. L’implémentation de ces quatre scénarios permet de limiter le nombre de téléchargements et d’éviter plusieurs bugs.

Le deuxième scénario a été compliqué à programmer car il faut actualiser les données du téléphone sans perdre les informations des œuvres que l’utilisateur a consultées. La solution a donc été de faire en sorte que, si une œuvre existait déjà dans la base de données, la version de la base de données est conservée sinon celle du serveur est rajoutée. Le travail qu’il me reste sur ce point est de faire en sorte que si une œuvre change de titre ou tout autre attribut, que la base de données de l’application puisse changer les renseignements de l’œuvre sans perdre les informations de l’utilisateur.

J’ai également changé les gestions des données pour simplifier l’application. Au lieu d’utiliser une base de données SQLite, j’ai implémenté une base de données ROOM. La particularité de ce type de base de données est qu’il a un niveau d’abstraction supplémentaire par rapport à SQLite et que je peux obtenir une liste d’objet à la suite d’une requête au lieu d’avoir un curseur. Les données sont conservées sous forme d’objets ce qui facilite la création de la base de données.

Avec la base de données ROOM, j’ai pu créer les requêtes SQL sans avoir à générer le code associé car le code est généré automatiquement et la gestion de la réponse est beaucoup plus facile. Par exemple, pour les tris, puisque j’obtiens une liste au lieu d’un curseur, je peux trier la liste directement avec des comparateurs appropriés. Une autre particularité de la base de données ROOM est que la modification des objets est plus simple. Pour modifier un objet, j’ai simplement à envoyer une ligne de code en passant l’objet modifié en paramètre pour que la base de données se mette à jour.

Un des inconvénients des bases de données ROOM par rapport à SQLite est que les requêtes ne doivent pas être lancées sur le fil d’exécution principal. J’ai donc dû créer

plusieurs classes asynchrones pour pouvoir exécuter les requêtes. Malgré tout, l'implémentation s'est bien déroulée.

Cette partie du travail a été principalement réalisée entre le 22 janvier et le 26 février. J'ai perdu beaucoup de temps à essayer d'implémenter une version de l'application qui n'utilisait pas de base de données mais qui enregistrait les données dans des fichiers et qui se passerait des données en paramètre. J'avais une version fonctionnelle sans utiliser de base de données et j'avais réussi à compresser suffisamment les données pour respecter la limite de taille de la classe 'Parcel' mais lors de changements rapides d'activités/fragments les données ne se transféraient pas avant le prochain changement de activités/fragments. J'ai donc été implémenté des variables statiques pour récupérer les données s'ils étaient perdus dans une transition. Puisque je ne pouvais pas éviter le recours aux variables statiques (qui prenaient beaucoup de la mémoire), j'ai décidé de réimplémenter une base de données pour limiter la mémoire requise. Par suite, j'ai appris le fonctionnement des bases de données ROOM.

Structure de l'application

Un des problèmes majeurs du code source était qu'il n'était pas structuré de manière efficace. L'application avait 2 activités différentes qui étaient également des fragments avec la même fonction: FicheActivity, MapActivity. Pour rendre la structure de l'application plus efficace, j'ai réduit le nombre d'activités à quatre.

1. FirstActivity est l'activité de lancement qui s'occupe de gérer les permissions et de gérer les données des œuvres d'art. Cette activité sera également responsable de l'authentification de l'utilisateur
2. MainActivity est l'activité principale dans laquelle l'utilisateur peut naviguer entre les différentes fonctions de l'application soit consulter la fiche du jour, la carte, la liste des œuvres d'art et voir ses photos.

3. PhotoActivity est l'activité qui permet de consulter les photos prises en plein écran. Cette activité était fournie dans le code source et à l'exception des modifications pour ajuster l'activité aux changements de la gestion des données de l'application, je n'ai pas beaucoup travaillé sur cette partie du code. Cette activité deviendra probablement un DialogFragment dans les semaines à venir puisque cette activité complique la navigation avec les boutons de retour arrière. L'avantage du DialogFragment est que l'utilisateur a davantage l'impression de rester dans l'application contrairement au lancement d'une activité en plein écran.
4. PreferencesActivity est l'activité qui permet de changer les paramètres de l'application. Cette activité sera travaillée pour rajouter des paramètres sur la gestion de données lorsque les comptes utilisateurs seront créés. Je n'ai pas travaillé sur cette activité qui provient majoritairement de l'auto-génération de code d'Android Studio.

La structure de l'activité MainActivity a également changé. J'ai choisi de conserver le fonctionnement avec les quatre onglets. Le troisième onglet était originellement pour consulter les favoris. La fonctionnalité de liste était dans le code source une activité qui était lancée à partir de la carte. Je l'ai donc combinée avec les favoris. Je l'ai modifié pour afficher une liste de toutes les œuvres et j'ai implémenté un filtre optionnel pour pouvoir voir seulement les favoris. L'objectif était de simplifier les fonctionnalités de l'application sans en enlever. Dans le code source, les visualisations d'une fiche autre qu'une fiche du jour était une activité alors que visualiser la fiche du jour était un fragment. J'ai donc modifié cela pour que les deux cas soient des fragments ce qui permettra de modifier le code plus facilement à l'avenir. La seule différence est que la fiche de l'œuvre qui n'est pas l'œuvre du jour est un DialogFragment ce qui veut dire qu'elle apparait dans une nouvelle fenêtre. J'ai fait ce choix car il permet de garder le nombre d'onglets de navigation à quatre et que l'utilisateur reste dans l'application donc qu'il voit toujours en background ce qu'il faisait avec l'application. Par exemple, si l'utilisateur sélectionne une œuvre sur la carte, la carte sera en arrière-plan de la fiche et ce sera clair pour lui que s'il

ferme la fiche il sera de retour à la carte. Ce choix permet également à l'utilisateur de naviguer sans toujours utiliser le bouton de navigation arrière des appareils Android ce qui était le cas avec l'activité FicheActivity du code original. Le DialogFragment que j'ai développé possède un bouton pour fermer celui-ci pour rendre la navigation agréable et intuitive.

Cette partie du travail a été effectuée majoritairement entre le 27 février et le 19 mars.

Développement de nouvelles fonctionnalités

Une autre partie de mon travail a été d'implémenter certaines idées de Lena ou certains éléments que je jugeais pertinents. Voici certains des éléments que j'ai rajoutés à l'application dans le cadre du projet :

1. Option d'affichage de la liste des œuvres dont le tri par distance. Dans la liste des œuvres, j'ai implémenté un tri par distance. Les distances sont affichées en mètres ou en kilomètres. Cette option est importante car elle permet à l'utilisateur de trouver rapidement quelles sont les œuvres les plus proches. Le principal problème pour moi était d'aller partager les données de localisation utilisé par le fragment de carte pour éviter de dupliquer les requêtes de localisation. J'ai également dû calculer la distance à partir de longitude et latitude ainsi que de créer une classe 'comparateur' pour effectuer le tri. Une autre option rajoutée est d'afficher seulement les œuvres classées favorites. Cette option a été assez facile à implémenter même s'il reste un bug mineur à résoudre. Cette fonctionnalité a été implantée entre le 20 et le 27 février.
2. Affichage d'une mini-carte pour les fiches sans photos. Lorsqu'une œuvre n'a pas été prise en photo, dans la fiche de celle-ci se trouvera une carte indiquant la location de cette œuvre au lieu d'une photo vide. Cette modification a été assez complexe puisqu'il a fallu recréer un fichier de xml pour une fiche avec carte et un autre pour une fiche avec photo et puisque je voulais que les deux scénarios soient gérés par le même fragment (pour limiter le nombre de classes de l'application), j'ai dû trouver une façon de passer d'un scénario à l'autre comme par exemple

lorsque l'œuvre a été prise en photo. La raison pour laquelle le changement est problématique est qu'on peut seulement déclarer le fichier xml une fois par fragment. La solution trouvée a donc été de détruire le fragment et d'en recréer un nouveau lorsqu'une photo est prise. Cette tâche a été faite entre le 19 mars et le 26 mars.

3. Ajout des favoris sur la carte. Lorsque l'utilisateur indique une œuvre comme favorite, celle-ci apparaitra en rouge sur la carte au lieu d'en vert. Bien que cet ajout semble anodin, j'ai passé beaucoup de temps à comprendre le fonctionnement de MapView et de OverlayItems sur lesquels la documentation est limitée. Dans le code source, une fois les œuvres placées sur la carte, cette carte était gardée en mémoire donc les objets étaient placés une seule fois. J'ai donc fait en sorte que chaque fois que le fragment de la carte est appelé, les objets sont replacés sur la carte pour s'assurer que toutes les œuvres sont dans la bonne catégorie. Cette tâche a été faite entre le 26 et le 30 mars
4. La gestion des permissions (Storage, Localisation, Camera). Le code source a été fait visant les versions d'Android égales ou inférieures à 5. Ces versions ne requièrent pas une gestion des permissions à l'extérieur du manifest. À partir de la version 6, en plus de demander la permission dans le manifest, le code doit demander l'accord de l'utilisateur au moins une fois avant l'utilisation des fonctionnalités nécessitant ces permissions en plus de vérifier que l'utilisateur n'a pas enlevé la permission à chaque fois qu'il utilise une de ses fonctionnalités. J'ai développé une classe qui s'occupe de regrouper la gestion des permissions pour l'application pour s'assurer que le code soit clair et simple. Cette classe est appelée entre autres au lancement de l'application pour demander toutes les permissions nécessaires à l'utilisateur. Cette partie du travail a été faite entre le 19 mars et le 26 mars.

Corrections de bugs

Bien entendu, en travaillant sur ce projet, j'ai rencontré de nombreux bugs et une quantité considérable de mon temps a été passé à tenter de les résoudre. Certains provenaient du code source tandis que d'autres ont été découvert après l'ajout de certaines fonctionnalités. Voici les problèmes les plus importants rencontrés :

1. Le "swipe" qui empêchait la navigation de côté sur la carte. Ce problème était dans le code source. Lorsque l'utilisateur était sur la carte et qu'il essayait de déplacer la carte de côté, l'application changeait d'onglet à la place. Puisque le fait de pouvoir changer de fragment en glissant l'écran de coté était pas un élément qui était apprécié, je crée un nouveau ViewPager qui ne supportait pas le "swipe".
2. L'accès à la base donnée ROOM ne doit pas être faite dans le fil d'exécution principal comme mentionné plus haut. Cependant plusieurs éléments du fil d'exécution principal ont besoin d'information de la base de données pour fonctionner. Un exemple est le fragment de fiche. Dépendant de si l'œuvre a une photo ou non, le fragment sera assigné de manière permanente un fichier XML. La solution que j'ai implémentée est de prévoir ses situations particulières et d'aller chercher les réponses en avance. Dans l'exemple de la fiche, avant de lancer le fragment de la fiche, je vais chercher la fiche de l'œuvre et je place en variable statique si elle est photographiée ou non. Lors du lancement du fragment, je vais consulter la variable statique.
3. L'actualisation des fragments lors des changements d'onglet. Ce problème n'est pas encore résolu. Lorsque l'utilisateur change d'onglet après avoir fait une action dans l'onglet où il était, il y de bonne chance que le nouvel onglet ne sera pas à jour. Par exemple, si l'utilisateur prend une photo à partir de la liste, il se peut que la photo n'apparaisse pas dans la galerie immédiatement. Ce problème arrive puisque le ViewPager garde toujours trois fragments en mémoire et que si le fragment est resté en mémoire il ne sera pas actualisé. Pour que le fragment soit actualisé, il doit être sorti de la mémoire du ViewPager. J'ai réussi diminuer la fréquence du problème mais pour une raison que j'ignore, je n'ai pas réussi à l'éliminer.

Conclusion

Durant la session, l'application a énormément progressé, si bien que l'application a été soumise à un test usager. Il reste cependant plusieurs choses à faire avant de publier l'application sur Google Play. Puisqu'un des objectifs du projet MONA est de récolter des données à des fins de recherches, il faudra créer des comptes utilisateurs et envoyer leurs données avec leurs consentements sur un serveur. Il y a plusieurs autres éléments que j'aimerais améliorer dont rendre l'application bilingue, modifier l'activité PhotoActivity pour que la photo soit affichée en meilleure qualité et rajouter des options pour réduire l'utilisation de la batterie de l'application. Je suis optimiste que l'application sera disponible sur Google Play dans un futur rapproché surtout que j'ai obtenu un contrat pour pouvoir continuer à travailler sur ce projet à l'extérieur du cours IFT3150.