IFT3150: Projet supervisé

Projet d'application mobile iOS: MONArt

Nom: Paul Chaffanet

Matricule: 1009543

MONArt est un projet d'application mobile utilisant les données d'oeuvres d'art public de la ville de Montréal. Cette application mobile doit permettre de parcourir les oeuvres d'art sur une carte ou sur une liste. Pour chaque oeuvre d'art, sa fiche permet d'y ajouter une photo afin de la collectionner, de noter cette oeuvre et de la commenter.

Ce projet d'application mobile est porté par Lena Krause, étudiante en maîtrise en histoire de l'art, qui a développée une partie de l'application sur Android en Java. Elle a fait appel à trois étudiants (dont moi) en baccalauréat informatique afin de poursuivre le développement de cette application mobile. J'ai donc notamment travaillé avec Vincent Beauregard, responsable de la partie serveur (gestion des données, requêtes HTTP) et Émile Labbé, responsable du développement de l'application sur Android. Pour ma part, j'étais chargé du développement de l'application sur la plateforme iOS.

1. Description des besoins de l'application

Afin de réaliser l'application mobile MONArt, plusieurs besoins devaient être comblés. En premier lieu, l'application devait afficher les oeuvres d'art sous forme de listes triables. Afin de rendre la navigation efficace au sein de l'application, il était également nécessaire de pouvoir afficher toutes les oeuvres d'art présentes dans un arrondissement, ou correspondantes à un artiste. Enfin, les oeuvres d'art devaient pouvoir être triées de manière alphabétique, par date de réalisation ou encore par distance depuis la localisation de l'usager.

Pour que l'utilisateur puisse effectuer des recherches plus ciblées, une fonction de recherche devait être implémentée. À l'aide des termes de la recherche, il est possible de retrouver les artistes ou les oeuvres d'art correspondants, et ceci plus rapidement que par le défilement d'une liste.

L'application devait être également capable d'afficher les oeuvres d'art de Montréal sur une carte. Sur celle-ci, chaque oeuvre montréalaise est indiquée par une épingle précisant sa position géographique. En cliquant sur cette épingle, l'usager devait pouvoir accéder à la fiche technique correspondante à l'oeuvre. De plus, cette carte devait permettre à l'usager d'indiquer sa position géographique dans le but de découvrir les oeuvres à proximité de sa position.

Concernant les fiches d'oeuvres évoquées plus haut, plusieurs fonctionnalités devaient être implémentées. Tout d'abord, chaque fiche a pour but de renseigner l'usager sur le titre, le ou les artistes, les dimensions de l'oeuvre, sa date de réalisation, les matériaux ou encore les techniques utilisées pour sa réalisation. De plus, des renseignements concernant l'arrondissement devait être affiché en complément d'une carte où uniquement l'oeuvre en question devait être indiquée.

En arrivant sur l'une de ces fiches, l'usager devait pouvoir prendre une photo dans le but de compléter sa collection personnelle d'oeuvres. Après avoir validé l'une des photos prises, l'usager devait être en mesure de noter l'oeuvre sur une échelle de 1 à 5 étoiles. De plus, l'usager devait avoir la possibilité de rédiger un commentaire sur l'oeuvre. Il etait impératif que la fiche de l'oeuvre puisse également permettre à l'usager d'ajouter l'oeuvre en question à sa liste de souhaits. La liste de souhaits devait ainsi contenir tous les oeuvres d'art ciblés par l'usager afin de les retrouver plus facilement plus tard.

L'un des besoins importants pour l'application mobile MONArt est la récolte de données. En effet, les données doivent pouvoir servir à des fins de recherche universitaire. Dans ce but, je devais rendre possible l'envoi des données (note, commentaire, etc.) de l'usager à propos d'une oeuvre à la base de données côté serveur. Par la suite, des données pourraient être récoltées via une fonctionnalité de partage de la photo sur les réseaux sociaux (Facebook, Twitter, Instagram, etc.).

Afin de fidéliser l'usager, et rendre l'utilisation de l'application plus ludique, des badges et des trophées devaient être implémentés dans l'application. Un décompte des oeuvres visitées et collectionnées (photographiées donc) devraient être fait afin de récompenser les utilisateurs les plus assidus. Ces récompenses peuvent provenir du nombre d'oeuvres visités, du nombre de commentaires faits ou encore des références à l'oeuvre faites par l'usager sur les réseaux sociaux.

Toujours dans le but de rendre l'application plus interactive avec les utilisateurs, l'application devait pouvoir proposer quotidiennement une oeuvre aléatoire afin d'inciter l'usager à partir à la découverte de "*l'oeuvre du jour*".

Encore dans l'optique de fidéliser l'usager, l'application devait être en mesure d'envoyer des notifications à l'usager afin de l'informer de la présence d'une oeuvre d'art à proximité de sa position.

Enfin, l'application étant destinée à la population montréalaise, j'avais comme impératif de créer une application bilingue anglais/français afin qu'elle soit disponible pour le plus grand nombre.

2. Conception de l'application

Le code est contenu en plusieurs dossiers qui représentent les différentes parties importantes de l'application.

Un dossier contient le modèle de données de l'application mobile tel que décrit dans la figure 1[1]. Ce modèle m'a été très pratique tout au long du développement de l'application. Les

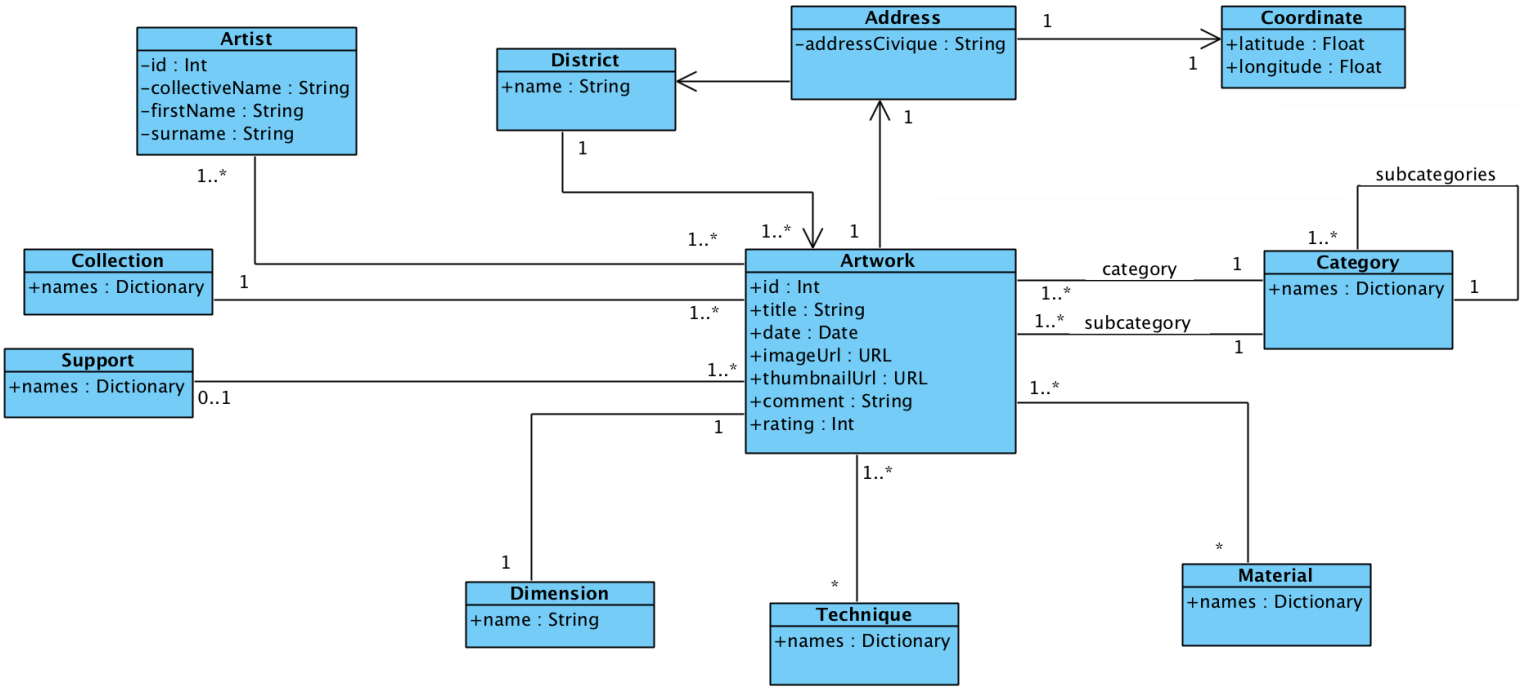

Figure 1: Modèle

[1] Les propriétés données pour chaque classe dans la figure 1 ne sont pas exhaustives. En revanche, les associations données dans le diagramme sont exhaustives pour chaque classe.

objets Category, Material, Technique, District, etc. permettent d'accéder facilement à la liste d'oeuvres qu'ils contiennent. Également, à partir d'une oeuvre d'art (une instance de Artwork), l'ensemble de ses propriétés tels que ses matériaux, son arrondissement, sa catégorie et sa sous-catégorie sont facilement accessibles.

L'application requiert qu'elle soit disponible en anglais et en français. Par exemple, les catégories, les techniques, les matériaux ont des traductions en anglais et en français dans notre jeu de données du bureau d'art public. Pour cela, chaque objet disposant d'une traduction a un attribut names qui contient un dictionnaire avec pour clé la langue (un type enum) et pour valeur une string correspondant au nom de l'objet dans cette langue. Par exemple, pour une technique, son attribut names pourrait être un dictionnaire [.en : "bolted", .fr : "boulonné"] où .en et .fr sont des valeurs d'un type enum Language. En fonction de la langue choisie par l'usager, nous pourrons ainsi retourner la bonne string à afficher dans l'application. Les messages d'erreur, le nom des listes, ou le nom des tris (comme le tri par "Titre") sont également autant de string qui doivent disposer d'une traduction en anglais et en français. Pour cela, j'utilise deux fichier de strings pour le français et l'anglais qui contiennent les traductions pour chaque langue. Dans le code, je fais appel à NSLocalizedString pour retrouver la string correspondante à la langue de l'usager. La langue de l'usager est paramètrable dans les paramètres de l'application.

J'ai un dossier appelé Data qui à ce jour contient une unique classe DataManager. DataManager est un singleton dans l'application. Cet objet me permet d'organiser la persistence des données au sein de mon application par des procédures de sauvegarde ou de chargement des données du modèle. Ce singleton me permet également de charger l'image d'une oeuvre d'art (à partir d'un objet Artwork, grâce à son attribut imageUrl), ou de sauvegarder une image d'une oeuvre d'art à un url donné. Enfin, en raison dernièrement des problèmes de performances auxquels j'ai été confrontés (notamment la gestion de la ram, pour une raison détaillé plus bas dans la partie 3), j'ai également donné la responsabilité à ce singleton de contenir une liste de tous les objets Artwork, Category, District, etc. afin d'avoir accès aux données essentielles au fonctionnement de l'application à un seul endroit.

Une opération importante à effectuer lors du lancement de l'application mobile est la récupération des données des oeuvres d'art. Quatre cas sont traités:

1. Si l'usager n'a pas Internet, et qu'il n'y a pas de données encore sauvegardées dans son téléphone (c'est le cas notamment lorsqu'un usager ouvre l'application pour la toute première fois sur son téléphone sans qu'il n'y ait de connexion internet), alors parser le fichier d'oeuvres d'art public JSON de base incorporé dans l'application. Pour cette étape, un package appelé SwiftyJSON est utilisé pour le parsing. L'étape de parsing nous permet alors de construire une liste d'objets Artwork qui seront sauvegardés immédiatement dans le téléphone grâce au DataManager.

2. Si l'usager a Internet, mais n'a pas de données encore sauvegardées, alors télécharger le dernier fichier JSON sur le serveur, et sauvegarder ces données dans le téléphone. Semblable à l'étape 1 en ce qui concerne le parsing du fichier JSON et la sauvegarde des données.

3. Si l'usager n'a pas Internet, mais des données déjà sauvegardées, alors l'application peut se lancer car on dispose des données nécessaires à son affichage. En effet, il suffit alors de demander au DataManager de charger les données en mémoire.

4. Si l'usager a Internet, et des données sauvegardées, alors télécharge le fichier JSON sur le serveur, et met à jour les données. Ainsi, si une oeuvre d'art dans le fichier JSON téléchargé contient de nouvelles informations (un nouveau titre, un nouvel auteur), ces informations se mettront à jour automatiquement grâce à l'attribut id.

J'ai un dossier appelé Network qui contient un fichier par opération (requête) réseau. Pour effectuer facilement des requêtes HTTP avec Swift, et pouvoir parser la réponse JSON, et permettre une gestion plus simple des erreurs, j'ai utilisé un petit package appelé

HermesNetwork[2] installé via CocoaPods[3]. Comme dit un peu plus haut, il m'est nécessaire d'affectuer une requête HTTP afin d'obtenir le dernier fichier JSON du serveur: j'ai donc un objet GetArtworksOperation, dont la méthode d'exécution effectue une requête HTTP et retourne une liste d'objets Artwork en parsant la réponse JSON de la requête HTTP.

D'autres opérations réseaux sont également utilisées dans l'application. Par exemple, j'ai codé deux opérations AddCommentOperation et AddRatingOperation. AddCommentOperation permet d'effectuer une requête pour déposer (uploader) un commentaire d'une oeuvre d'art donné par un utilisateur (que l'on identifie actuellement et provisoirement par son username) sur le serveur. Pour qu'un utilisateur dépose une note, je fais appel à l'opération AddRatingOperation. Également, je tiens à mentionner l'implémentation partielle des opérations LogInUserOperation et CreateUserOperation. Partielle, car nous devons encore mettre en place un protocole plus sécuritaire pour l'authentification d'un utilisateur. Ces opérations réseaux ne sont donc pas encore totalement implémentées.

En ce qui concerne la partie plus visible de l'application mobile, la partie qui m'a donné le plus de travail est la liste de la section "Oeuvres". Cette liste est inspirée de la liste de l'application Music d'iOS9 que je trouvais très efficace notamment par la présence d'un index.

La liste dans la section "Oeuvres" est triable avec un objet TapSegmentedControl permettant de sélectionner le tri voulu. TapSegmentedControl hérite de UISegmentedControl. En effet, UISegmentedControl est une classe qui appartient au framework UIKit et qui n'envoie pas d'évènements lorsque qu'un segment est tapé plusieurs fois. Il ne réagit qu'au changement de segment (par exemple, passer d'un tri par titre à un tri par distance). Il était ainsi nécessaire que je redéfinisse les méthodes de UISegmentedControl afin d'envoyer un évènement lorsqu'un usager tape sur le même segment déjà sélectionné et ainsi permettre un tri ascendant ou descendant. Pour les oeuvres d'arts, celle-ci sont triables par leur titre, par leur date de réalisation ou par distance entre la position de l'usager et la position de l'oeuvre. Pour les catégories, les

---

[2] https://github.com/malcommac/HermesNetwork

[3] Gestionnaire de paquets pour Swift utilisé pour l'application

artistes ou encore les arrondissements, ceux-ci sont triables par leur titre uniquement. Afin de rendre ces listes rapidement parcourables, j'ai installé un petit package appelé MYTableViewIndex[4] que j'ai légèrement modifié afin que lorsque que l'on glisse son doigt vers la partie haute de l'index, ce geste nous ramène tout en haut de la liste au niveau des contrôles de tri (objet TapSegmentedControl). Cet index s'adapte également selon que les sections de la liste soient des lettres, des dates ou des distances. J'ai également travaillé l'animation afin que l'index apparaîsse au moment où l'usager parcoure la liste, et disparaîsse une fois que l'usager est tout en haut de la liste au niveau du TapSegmentedControl.

J'ai créé pour le moment deux prototypes de cellules regroupé dans un dossier appelé Custom TableViewCells. Le premier prototype de cellule, un objet appelé ArtworkTableViewCell, permet l'affichage d'un thumbnail, d'un titre et d'un sous-titre. Ce premier modèle de cellule est utilisé dans l'affichage d'une oeuvre d'art dans une liste. Le thumbnail étant une image coupée (cropped) carrée et plus légère de la photo de cette oeuvre d'art. Un deuxième modèle de cellule, appelé GeneralTableViewCell, se contente d'afficher seulement un titre et un sous-titre. Une GeneralTableViewCell est typiquement utilisée pour afficher par exemple les artistes, les catégories ou les arrondissement dans une liste, avec comme sous-titre le nombre d'oeuvres d'art contenues. Ces deux prototypes de cellule pour une liste sont largement réutilisés dans l'application, que ce soit pour l'affichage de résultats de recherche, ou par la navigation à travers les oeuvres d'art, les artistes, etc.

J'ai également configuré quatre sources de données (ArtworksTableViewDataSource, CategoryTableViewDataSource, etc.) pour les oeuvres, les artistes, les arrondissements et les catégories qui spécifient quelles sont les données à affcher: nom des sections, contenu de l'index, type de cellule à utilisé, etc. Ainsi j'utilise des cellules du type ArtworkTableViewCell pour afficher des oeuvre, et des GeneralTableViewCell pour afficher les données des artistes, des arrondissements et des catégories dans une listes[5].

---

[4] https://github.com/mindz-eye/MYTableViewIndex

[5] Chaque cellule contient une référence de l'oeuvre d'art (une instance de Artwork) qu'il représente. Les GeneralTableViewCell contient une référence de la catégorie, de l'artiste, ou de l'arrondissement qu'il représente.

Lorsque que l'on clique sur une GeneralTableViewCell, celle-ci donne accès à une sous-liste qui contient les oeuvres d'arts contenues par l'objet représenté: une liste avec des cellules de type ArtworkTableViewCell.

Lorsque que l'on clique sur une ArtworkTableViewCell, celle-ci ouvre une vue de type ArtworkDetailsViewController. Cet objet représente la fiche technique d'une oeuvre d'art et est également largement réutilisé au sein de l'application, notamment dans la section "Carte".

Un objet ArtworkDetailsViewController est responsable du comportement des fonctionnalités de prise de photo, de notation et de commentaire d'une oeuvre. La vue renseigne également sur la description de l'oeuvre. Elle implémente aussi la possibilité d'ajouter une oeuvre à la liste de souhaits, et de partager la photo d'une oeuvre sur les réseaux sociaux. Lors de la prise d'une photo, la photo est sauvegardée au sein de l'application. On demande immédiatement après à l'usager de noter cette oeuvre d'art. L'usager peut ensuite laisser un commentaire pour cette oeuvre s'il le souhaite. Plus tard, le commentaire et la note sont rééditables. Après chaque édition de note ou de commentaire, ceux-ci sont envoyés au serveur grâce aux opérations AddCommentOperation et AddRatingOperation. La vue ArtworksDetailsViewController affiche également une mini-carte qui permet à l'usager d'obtenir un aperçu de la localisation de l'oeuvre. Lorsque l'usager clique dessus, une carte contenant uniquement l'oeuvre s'ouvre. L'usager peut se localiser sur cette carte et peut ainsi facilement déterminer le chemin à effectuer pour parvenir à l'oeuvre.

La section "Carte" affiche toutes les oeuvres d'art de l'application. Un dossier "Map" contient les différents fichiers de code gérant cet aspect de l'application. J'utilise dans cette partie le framework CoreLocation. La carte affiche toutes les oeuvres d'art issus de nos données. Lorsque que l'on clique sur une épingle, une annotation s'ouvre. Lorsque que l'on clique sur l'annotation, un ArtworkDetailsViewController s'ouvre proposant les mêmes fonctionnalités précisées plus haut. Par ailleurs, il est possible de trier les épingles sur cette carte selon que les oeuvres d'art soient présentes dans la collection ou dans la liste de souhaits.

Les fichiers de code qui implémentent la fonctionnalité de recherche de l'application sont regroupés dans un dossier "Search". Un objet CustomSearchController (qui hérite de UISearchController du framework UIKit) est responsable d'une barre de recherche qui contient les termes de la rcherche. Un objet SearchResultsController s'occupe d'afficher les résultats de la recherche. Le SearchResultsController affiche les résultats dans une liste dont les données sont fournis par un objet SearchDataSource. La liste classe les résultats par sections: "Oeuvres d'art", "Artistes", "Arrondissements", "Catégorie". À chaque caractère entré ou enlevé dans les termes de recherche, l'affichage des résultats se met à jour immédiatement (en instanciant un nouvel objet SearchDataSource) en vérifant si les termes de la recherche correspondent au titre d'une oeuvre, au nom d'un artiste ou encore au nom d'un arrondissement. Afin de ne pas avoir une liste trop longue, chaque section ne peut pas contenir plus de 3 éléments. Il suffit de cliquer sur le nom d'une section pour afficher l'ensemble des résultats de la recherche contenue dans la section. Ainsi SearchResultsController n'affiche pas plus de 3 éléments par section, soit 12 éléments maximum, en prenant les 3 premiers résultats correspondants par ordre alphabétique. Par ailleurs, afin de permettre à l'usager de retrouver ses recherches récentes, un historique des recherches de l'usager est accessible dans la barre de recherche. Lorsque que l'on clique sur élément de l'historique de recherche, cela rentre cette recherche de nouveau dans le CustomSearchController, ce qui provoque la mise à jour du SearchResultsController car celui-ci instancie un nouveau SearchDataSource. Comme pour les listes, le fonctionnement de la fonctionnalité de recherche est inspirée de celle présente dans l'application Music d'iOS9.

J'ai également plusieurs vues qui gèrent les différentes partie du programme, telles que WishlistViewController, responsable d'afficher les oeuvres d'arts ciblées par l'usager. Lorsque que cette vue est chargée, je vais chercher les oeuvres d'art ciblés par l'usager (grâce à la propriété isTargeted d'une instance de Artwork) dans le DataManager qui contient une référence aux oeuvres d'art de l'application. J'affiche ensuite ces oeuvres dans une liste grâce à mon objet ArtworkTableViewCell. La vue CollectionViewController fonctionne de manière similaire à WishlistViewController afin d'afficher les oeuvres collectionnées par l'usager grâce à la propriété isCollectionned d'une instance de Artwork.

J'ai également une classe UserDefaultsManager qui me permet d'effectuer une persistence des paramètres de l'utilisateur, comme par exemple les paramètres de la langue pour l'application. UserDefaultsManager sera principalement utilisé par la vue SettingsViewController afin de permettre à l'usager de modifier les paramètres de l'application.

Enfin, j'ai dernièrement mis en place une procédure d'inscription et de connexion pour un usager, avec plusieurs vues tels que LoginViewController, qui affiche un formulaire permettant à un usager de se connecter à l'application, et SignUpViewController qui permet d'enregistrer les informations nécessaires à l'inscription d'un nouvel usager à notre base de données côté serveur. Il reste cependant à effectuer cette procédure de manière sécuritaire.

3. Problèmes rencontrés

J'ai rencontré de nombreux problèmes tout au long du développement de cette application mobile.

Une des premières questions que nous avons abordées lors du commencement de nos réunions de groupe fut le choix d'utiliser un framework mobile cross-platform ou non. Un avantage dans l'utilisation d'un framework mobile cross-platform est que nous pouvions obtenir une version 1.0 de l'application mobile plus rapidement grâce à la présence de deux développeurs pour travailler sur une même version de code côté client. D'autre part, nous aurions eu besoin que d'un seul développeur maîtrisant le framework afin de mettre à jour l'application mobile sur les deux plateformes par la suite.

Pour répondre à cette question fondamentale pour le commencement du projet, il a été décidé que nous allions tester l'utilisation d'un framework cross-platform durant une semaine. Kivy a été le premier framework proposé en test. Kivy est un framework open-source écrit en Python permettant le développement d'interface usager sur Android, iOS, macOS, Linux et Windows. J'ai donc testé le framework Kivy en créant une application mobile très simple consistant à prendre une photo avec l'APN (Appareil Photo Numérique) d'un téléphone Android et iOS et à l'enregistrer dans l'album photo de l'appareil. Cette application a bien fonctionné sur

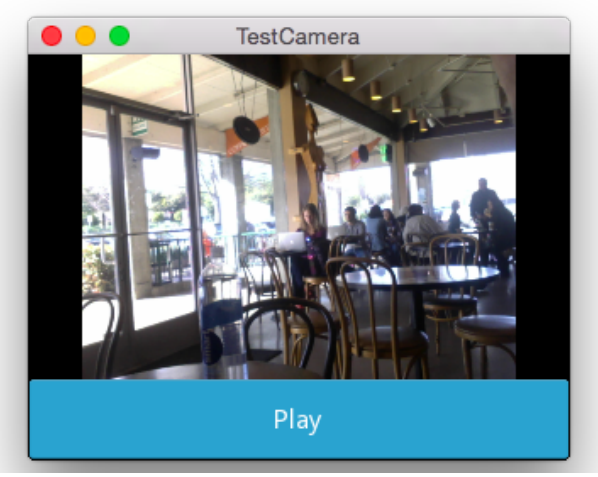

Figure 2: Capture d'écran à titre illustratif d'une application avec le framework Kivy

les deux plateformes, mais il m'est apparu que le design de cette application était assez inhabituel pour une application iOS ou Android, ce qui m'a dérangé. En effet, un principe important en conception d'interface usager est de suivre les standards propres à chaque plateforme afin de proposer une interface usager cohérente pour l'usager. MONArt est une application qui souhaite rendre facilement accessible les oeuvres d'art publique de la ville de Montréal, et je croyais donc que le respect de ce principe était important.

En effectuant de plus amples recherches sur Kivy[6], je suis tombé sur une réponse très intéressante d'une personne, dont je cite une un extrait:

"

Kivy.org offers one cross platform solution that uses Python. But even then it's not ideal. It creates controls in OpenGL like a game would typically do, which works fine for a game, but looks not at all like native controls on any platform. If you want to create a 100% themed app, this isn't so bad. But you're also locking yourself into the ecosystem, so that if Kivy doesn't have a control that you need, you will need to make it yourself.

"

[6] https://www.quora.com/Can-I-build-iPhone-apps-using-Python

Ainsi, si nous voulions respecter une interface assez classique d'une application mobile propre à chaque plateforme, Kivy ne semblait pas adapter dans notre situation. Ce framework nous donnait une bien trop grande flexibilité dans la conception de notre interface usager, et nous voulions avant tout faire en sorte que les usagers naviguent facilement au sein de l'application, et soient dans des repères qu'ils connaissent.

Dans ce même lien, ce développeur conseillait donc l'utilisation du framework React Native plus adapté selon lui pour une interface usager plus conventionnelle et qui repose sur des outils de développement plus mature. J'ai donc dans les jours qui ont suivi exploré React Native en suivant la même méthode que pour Kivy: développer une application simple de prise de photo et l'enregistrer dans l'appareil.

Comme lors de mon essai avec Kivy, j'ai voulu codé la prise et l'enregistrement de photo sur Android et iOS. Bien que mon code fonctionnait sous iOS, j'ai eu beaucoup de mal à le faire fonctionner pour Android. Après quelques recherches, je crois que c'est le module photo[7] qui présentait quelques bugs avec la nouvelle version d'Android.

Bien que séduit par React Native qui correspondait parfaitement à nos besoins de développement pour "MONArt", j'ai "pataugé" pendant un jour ou deux à essayer de trouver une solution pour résoudre le bug qui concernait Android, sans succès. Ceci m'a convaincu que les SDK natifs semblaient être plus adaptés afin de proposer une application fonctionnelle plus rapidement.

En effet, les framework que nous avons explorés disposaient d'une documentation relativement pauvre. React Native est un framework qui gagne en popularité, mais j'ai pu constaté que la communauté est quand même relativement réduite. Le bug que je n'ai pu résoudre concernant la prise de photo dû à une version d'Android nouvelle qui interagissait mal avec le module photo de React Native démontrait bien que l'utilisation d'un framework nécessite également une gestion des dépendances très rigoureuse et régulière afin d'assurer une plus grande longévité à l'application mobile. Enfin un framework ne résout pas tous les problèmes. Par exemple, React Native ne permet pas un fonctionnement cross-platform pour absolument toutes les fonctionnalités du téléphone: par exemple certaines fonctionnalités requièrent d'être

[7] https://github.com/react-native-community/react-native-camera

écrites malgré tout en code natif, tels que l'accès aux capteurs du téléphone, ou la gestion des notification.

Dans notre contexte, nous étions deux développeurs côté client. Nous pouvions ainsi disposer d'un développeur par plateforme. De plus, nous disposions également d'une base de code déjà écrite par Lena pour la plateforme Android. En ce qui concerne la plateforme iOS, il existait de très nombreux tutoriels et une documentation très riche, ce qui m'a beaucoup aidé pour résoudre les nombreux problèmes que j'ai rencontrés durant le développement de l'application.

L'utilisation de React Native m'a paru pleines de possibilités, et c'est donc pour cela que je le garde sous le coude pour de futur projets mais il me paraissaitt également important d'avoir de bonnes notions en iOS et en Android avant de se lancer dans l'utilisation d'un framework mobile cross platform afin de maîtriser plus en profondeur les concepts sous jacents induits par ce framework.

Figure 3: Application utilisant le framework cross-platform React Native

Après que la question importante d'utiliser un framework cross-platform ou non fut tranchée, j'ai commencé à développer l'application MONArt en Swift 4.0 avec l'IDE Xcode en version 9.3. Swift est langage qui se rapproche fortement de Java ou de Python du point de vue de la syntaxe, ce qui m'a donné beaucoup de facilités pour le comprendre. Afin d'apprendre Swift, je me suis assez vite lancé dans le développement de l'application iOS. Tout au long du développement de l'application MONArt, j'ai ainsi amélioré mon niveau de compétence dans ce langage en consultant très régulièrement des sites internet tels que Stackoverfow ou la documentation officielle de Swift. Cela m'a pris quelques semaines afin de m'approprier les spécificités du langage, tels que les types optionnels et non-optionnels, les variables mutables et non-mutables, les protocoles[8], ou encore les extensions[9]. Cet apprentissage d'un nouveau langage de programmation m'a amené à commettre de nombreuses erreurs de conception. Par exemple, je n'ai commencé à m'intéresser que récemment à la gestion de la mémoire dans Swift. Il s'avère que la gestion de la mémoire s'effectue avec un compteur de référence. Dans mon diagramme de la figure 1, il existe de nombreux associations cycliques entre les instances. À la mesure de l'utilisation de la mémoire vive dans l'application, j'ai observé une accumulation d'objets morts, non désalloués, ce qui entraînait une mauvaise performance de l'application. Alors que je n'utilisais que des liens forts entre les objets, j'ai appris l'importance de l'utilisation du mot clé **weak** en Swift qui permet de créer une référence à un objet qui ajoute 0 au compteur de référence de cet objet (contrairement à une référence avec un lien fort qui ajoute 1 au compteur de référence de l'objet pointé). Ce type d'erreur provoqué par mon inexpérience dans Swift m'ont amené à ré-écrire plusieurs fois des parties de mon code.

Un problème que j'ai rencontré également fut la persistence des données. Il faut savoir que deux solutions standards existent pour la persistence des données dans iOS: utiliser NSCoding, du framework Foundation et NSKeyedArchiver pour sauvegarder les données sur le téléphone, ou s'appropier le framework CoreData. J'ai fait le choix de rendre encodable et décodable les données via le protocole NSCoding, et de sauvegarder les données grâce à

---

[8] Un protocole est pllus ou moins un équivalent des interfaces en Java, excepté qu'un protocole peut spécifier l'implémentation de propriétés (attributs).

[9] Une extension permet d'ajouter de nouvelles méthodes n'importe où dans le programme à une classe déjà implémentée dans le programme.

NSKeyedArchiver car c'est la solution préconisée de manière générale pour un faible nombre d'objets (moins de 10000). Mais je manque encore d'expérience pour savoir si ma décision est la bonne à terme, ne sachant pas ce que CoreData me permet vraiment de faire.

J'ai également perdu du temps à comprendre l'utilisation de CocoaPods (gestionnaires de packages) et l'installation de packages pour une application iOS. Mais cela m'a permis ensuite d'installer des packages importants comme SwiftyBeaver[10]. SwiftyBeaver est un package qui améliore la création de fichiers de logs utiles notamment pour la débuggage et la localisation des erreurs dans le code. Cela m'a permis d'améliorer la gestion des erreurs par la suite dans l'application.

Les framework UIKit et CoreLocation m'ont également demandé un certain temps d'apprentissage afin d'en saisir les subtilités et spécificités. J'ai également eu quelques problèmes dans l'apprentissage des animations car je souhaitais rendre la navigation au sein de l'application MONArt agréable pour l'usager.

Ainsi, tout cet apprentissage d'un nouveau langage de programmation, et de nouveaux frameworks m'a amené à faire de nombreuses erreurs, et m'a ralenti dans l'implémentation des fonctionnalités de l'application.

À cet égard, les fonctionnalités de partage sur les réseaux sociaux n'ont pas été implémentées. Apple a facilité le partage sur les réseaux sociaux avec le framework Social, mais cette solution ne m'a pas vraiment convaincue après l'avoir implémentée. En effet, lorsque que je partageais un tweet, il n'était pas possible avec le framework Social de récupérer le lien du tweet partagé. Or cela pose problème dans la mesure où l'on souhaiterait récupéré le contenu d'un tweet. L'API de Twitter semble donner cette possibilité, encore faut il que j'apprenne à l'utiliser. Je dois donc encore apprendre à utiliser les API de Twitter, Facebook et Instagram ce qui va prendre un certain temps apprentissage.

Les notifications n'ont pas été implémentées par manque de temps lié à cet apprentissage de l'environnement iOS. C'est également un nouveau framework à apprendre et une gestion des permissions efficace à effectuer. La section "Oeuvres du jour" n'a pas encore de design, et les

[10] https://github.com/SwiftyBeaver/SwiftyBeaver

modalités d'apparition de l'oeuvre du jour ne sont pas encore fixés, ce besoin n'est ainsi pas encore comblé.

Enfin, je regrette l'absence des renseignements sur les techniques et les matériaux pour les oeuvres d'art, mais je ne dispose pas encore de données fiables pour permettre leur affichage. Notamment, les traductions manquent de fiabilité ou de cohérence dans les données, et nous devons encore améliorer cet aspect afin de pouvoir permettre la recherche d'oeuvres d'art selon la matériau ou la technique utilisée.

4. Pistes de développement

Il reste encore beaucoup à faire afin de permettre la publication de l'application MONArt sur l'App Store.

J'ai comme priorité absolue de corriger les bugs et problèmes de performance d'ici la fin de la session afin d'avoir une application parfaitement utilisable avec les fonctionnalités déjà implémentées quelque soit les conditions d'utilisation: sauvegarde des données quand la batterie descend à 0 par exemple, ou stockage des requêtes afin d'envoyer les données lorsque que l'on recouvre internet en cas de perte du réseau à l'utilisation. Un premier test usager de l'application fin mars à permis de remonter de nombreux bugs à l'utilisation, ainsi que des problèmes de performance que je dois résoudre.

Une priorité importante à régler d'ici la fin de la session est l'implantation d'un système d'authentification (inscription, connexion) sécuritaire de l'usager. Les requêtes HTTP échangées entre le client et le serveur sont pour le moment extrêment rudimentaires. Nous devrions nous assurer de l'utilisation de requêtes HTTPS afin de crypter le contenu des échanges entre le serveur et l'application. Nous réfléchissons également à l'utilisation de OAuth 2.0 afin de permettre la gestion de l'authentification de l'usager et permettre un maintien de la connexion usager à la réouverture de l'application (afin que l'usager n'ait pas à entrer ses identifiants à chaque ouverture de l'application) de façon sécuritaire. Nous devons aussi mettre en place un système de récupération de mot de passe. Dans l'idéal, nous devrions aussi permettre une inscription immédiate avec un bouton "S'incrire par Facebook" afin d'accéler l'inscription de

l’usager à l’application. Cela fait partie des priorités importantes à implémenter d’ici la fin du mois d’avril.

J’attends également les design de la fiche d’une oeuvre et le design de la section "Oeuvre du jour". En effet, des étudiants en design sont censés nous fournir un design de cette section durant le mois de mai ou de juin. La fiche d’une oeuvre doit également être redessinée afin qu’il soit plus agréable pour l’utilisateur d’interagir avec elle.

Il faut également implémenter les fonctionnalités de partage sur les réseaux sociaux: je n’ai aucune connaissance sur l’utilisation des API propres à chaque réseau social, je dois donc encore déterminer quelles sont les possibilités offertes par chaque API, et savoir comment nous pouvons organiser la récolte de données des réseaux sociaux à partir de l’application. En connaissant les limites propres à chaque API, nous pourrons alors décider en réunion quelles seront les modalités exactes de partage sur les réseaux sociaux.

En priorités un peu moins importantes, mais qui seront implémentés durant l’été, il y a le système de badges et de récompenses: nous devrons dans les prochaines réunions déterminer le design des badges et des récompenses, leurs conditions d’obtention et leur emplacement général au sein de la navigation dans l’application. Nous devons également aborder le besoin des notifications plus en détail.

Je veux également poursuivre mon travail pour rendre l’application encore plus facilement navigable en rendant les renseignements dans la fiche d’une oeuvre (tel qu’un artiste, une catégorie, un arrondissement) cliquables afin d’afficher les oeuvres d’arts associées.

Une fois l’application prête, et que nous aurons procédé à une deuxième vague de tests usagers avec toutes les fonctionnalités implémentées, nous tenterons de la soumettre à l’App Store pour publication à la fin du mois d’août. Ça sera une étape importante dans laquelle nous n’avons aucune expérience, mais qui sera dans tous les cas très instructive.

5. Conclusion

Après quatre mois de développement de cette application, la plupart des besoins énoncés au début du projet ont été comblés. Ceci permettent une navigation fluide et efficace parmi les oeuvres d'art de la ville de Montréal. En outre, les fonctionnalités de prise de photo, de note d'une oeuvre et de commentaire d'une oeuvre sont implémentées, ainsi que la localisation des oeuvres sur une carte, parmi une collection et parmi une liste de souhaits. L'application est également bilingue anglais/français.

Durant cet exercice, j'ai rencontré divers obstacles, certains plus importants que d'autres. Cette première expérience d'application iOS m'a poussé à apprendre un nouveau langage de programmation et de nouveaux frameworks, mais ce nécessaire apprentissage à également eu un impact sur ma capacité à implémenter toutes les fonctionnalités souhaitées à l'origine dans un intervalle de temps de 3 mois. L'authentification de l'usager doit encore être sécurisée et plus aboutie. Il manque également les fonctions de partage sur les réseaux sociaux qui pourront servir à la récolte de données, ou encore les fonctions de récompenses et de notification afin de fidéliser l'usager. De plus, l'arrivée d'étudiants en design dans l'équipe engage quelques modifications du design de la fiche technique d'une oeuvre et de la section "Oeuvre du jour".

J'aspire donc à poursuivre le développement de cette application afin d'implémenter les besoins nécessaires à son aboutissement. Le but sera ensuite d'arriver à publier l'application sur l'App Store après s'être assuré de son parfait fonctionnement.