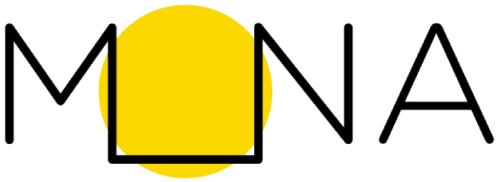
Gaspard Damoiseau-Malraux

Matricule 20255583

Rapport de projet

IFT-3150

Session hiver 2023



Université de Montréal

gaspard.dama@gmail.com

Découverte du projet

La Maison MONA est un organisme à but non lucratif qui travaille à la démocratisation de l'art public au Québec, par la création d'un espace commun centré autour d'activités, d'évènements et surtout d'une application mobile. Cette application permet aux utilisateurs de prendre des photos des œuvres d'art public et de les ajouter à leur collection virtuelle, créant ainsi une expérience de chasse au trésor dans l'espace québécois. L'objectif est de favoriser l'appropriation de l'art public par le plus grand nombre.

Mon rôle dans l'organisme était en développement Android. J'ai pu télécharger et tester l'application MONA dès décembre 2022, et j'ai eu accès au code source après le premier rendez-vous. J'ai pu y rencontrer d'autres membres du projet : Lena Krause, Prof. Guy Lapalme et les deux autres étudiants mandatés pour le projet (Yan Zhuang en développement iOS et Natacha Rivière en développement serveur. Nous y avons appris les enjeux du projet, son état actuel et les attendus globaux de la session. L'application est séparée en cinq sections :

- L'onglet Découverte du jour : propose une œuvre, un lieu ou un élément de patrimoine tiré aléatoirement ;
- L'onglet Liste : affiche la liste de toutes les découvertes existantes dans l'application ;
- L'onglet Carte : montre toutes les découvertes à faire et déjà trouvées sur une carte interactive ;
- L'onglet Collection : répertorie les découvertes collectionnées par l'utilisateur ainsi que la photo associée à chaque élément ;
- L'onglet Autres : paramètres, métadonnées.

Initialement, j'étais supposé travailler sur les points suivants :

- Poursuivre l'implémentation d'une nouvelle architecture de l'application ;
- Poursuivre l'ajout d'un nouveau type aux œuvres de l'application (type « patrimoine ») ;
- Maintenance globale de l'application ;
- Introduction de nouvelles fonctionnalités si la restructuration avance bien.

J'ai donc commencé, dès la première semaine, à apprendre le Kotlin et la programmation Android pour me mettre au travail sur l'application. Parallèlement, je me suis renseigné sur les besoins de l'application et les bugs existants. J'ai remarqué trois bugs majeurs :

- La découverte du jour ne fonctionne pas (rien ne s'affiche) ;
- Il est impossible de prendre une photo avec les téléphones utilisant *Android stock* (Google Pixel, Fairphone, etc.) ;
- Sérieux problème de performance à l'ouverture des détails d'une œuvre (menant régulièrement au crash de l'application).

Travail effectué

- Notre organisation

Nous avions un rendez-vous toutes les semaines le mercredi, pour faire un point sur l'avancement de chacun, les problèmes rencontrés et les éventuelles solutions, et pour décider de la direction des efforts fournis.

C'était à mes yeux une manière de procéder très utile, parce que nous avions souvent besoin d'un interlocuteur. C'était en quelque sorte une mise en application de la méthode AGILE, où Lena nous donnait un retour chaque mercredi sur notre travail, et l'on décidait ensemble sur quoi travailler la semaine suivante. Nous alternions un rendez-vous sur deux à distance, l'autre au bureau C-8130. Certaines rencontres étaient un peu différentes (rendez-vous avec Pr. Lapalme, séances avec des testeurs, rendez-vous à des évènements culturels, etc.) et toutes nous ont été utiles et intéressantes.

Nous communiquions principalement via Element.

- Travail avec Kotlin

Face à la complexité du code existant et en reconstruction, j'ai eu beaucoup de difficultés à comprendre la manière dont l'application fonctionnait, d'autant plus que je n'avais aucune expérience en développement Android.

Heureusement, j'ai pu assez rapidement corriger les deux premiers bugs (œuvre du jour non fonctionnelle, et prise de photo impossible sur Android stock) et nous avons pu déployer cette version corrigée de l'application assez vite sur le Google Play Store. En revanche, si j'ai pu améliorer les performances dans certaines situations, l'application souffrait toujours de crashes irréductibles. Nous avons essayé de contacter d'anciens étudiants, dans l'espoir que leur expérience puisse débloquer la situation, sans succès.

Mi-février, nous avons organisé un rendez-vous avec Pr. Lapalme et Simon Janssen pour discuter d'une refonte complète de l'application. Une grande quantité de travail devait être injectée dans le projet, et la migration vers une nouvelle version de l'API serveur était nécessaire. Grâce à mes recherches préalables, nous avons étudié plusieurs possibilités si l'application devait effectivement être réécrite :

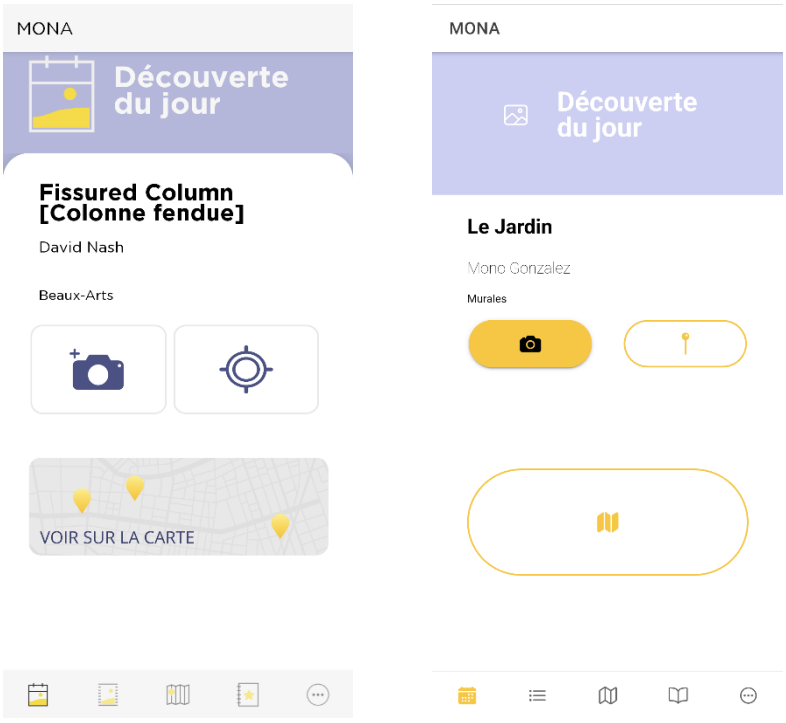
- **Kotlin (reprendre de quasiment zéro et réécrire en natif)**
    - Avantages : application native, donc intégration et performances maximales ;
    - Inconvénients : développement long et fastidieux, surtout vu le peu d'expérience que j'avais ;
- **Utiliser le framework React Native**
    - Avantages : intégration/performances similaires au Natif, sans passer par Kotlin ;

    - Inconvénient : courbe d'apprentissage difficile ;
- **Framework Flutter**
    - Avantage : identique à React mais facile d'utilisation et agréable ;
    - Inconvénient : nécessite l'apprentissage du langage Dart ;
- **Framework Ionic (utilisation d'un navigateur intégré à l'application pour l'interface)**
    - Avantages : développement facile et rapide, utilisation d'un framework web (Vue.js, Angular, React) pour l'interface ;
    - Inconvénient : potentiels problèmes de performances, limitations du web ;

Après réflexion, nous avons voté pour essayer le framework Ionic en raison sa facilité. Étant donnée la relative simplicité de l'application, les questions de performances posaient un problème limité. L'attention se portait surtout sur l'accès à la caméra, à la localisation, la possibilité d'écrire sur la mémoire locale et celle d'afficher une carte dynamique. Je me suis mis au travail le soir-même.

- Travail avec Ionic

À partir de ce moment, nous avons décidé de ne plus travailler sur l'ancienne version de l'application Android pour ne pas perdre de temps. Le défi pour moi était de proposer une version complètement fonctionnelle de l'application. J'ai choisi le framework Vue.js en collaboration avec Ionic pour construire l'interface utilisateur. J'ai commencé par reproduire la section des découvertes du jour, pour évaluer le temps nécessaire et la difficulté pour produire une interface statique. Les résultats (primitifs, certes) étaient assez concluants et, comme prévu, avec un minimum d'expérience en CSS il est facile de construire quelque chose.

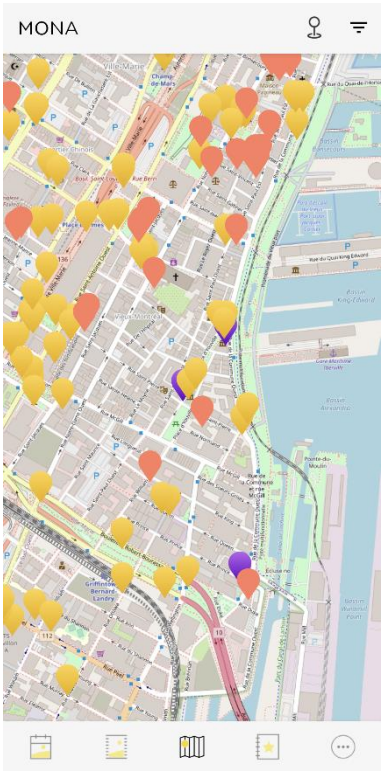


À gauche, la version alors actuelle de l'application. À droite, la preuve de concept avec Ionic.

L'étape suivante était l'intégration de la carte interactive, pour vérifier la faisabilité du projet. La référence dans le monde du web est Leaflet, il m'a donc semblé naturel de tenter cette solution. Cependant, après de nombreuses heures d'effort infructueux, j'ai décidé de passer à OpenLayers (autre grande bibliothèque permettant d'afficher une carte interactive), qui a permis une intégration beaucoup plus facile et adaptée à mon environnement de développement.

Au fur et à mesure du développement, j'ai recréé chaque page de l'application aussi proche possible de la disponible sur le Figma de la graphiste, Barbara Marche. Je me suis aussi servi des nouvelles ressources disponibles sur Figma. Les pins sur la carte ont été adoucis, par exemple, en comparaison avec la version précédente.

Assez vite, j'ai rencontré un problème que je n'arrivais pas à résoudre : il était impossible d'envoyer des photos au serveur depuis l'application. Au bout d'une semaine de blocage, j'ai demandé de l'aide sur la messagerie Élément et Abdelhakim (ancien étudiant MONA en 2019) a eu la gentillesse de m'aider, et nous avons pu trouver une solution.



Haut : carte de l'ancienne app ; bas : carte de la nouvelle app (rendu final)

Un problème que j'ai rencontré était qu'il est impossible d'initialiser un Random Number Generator (RNG) avec une *seed* en JavaScript/TypeScript. J'ai donc dû implémenter un RNG manuellement, dont la seule utilité était l'introduction d'aléatoire contrôlé dans l'application (par exemple : pour la découverte du jour qui ne doit pas changer dans une même journée, en cas de réouverture de l'application, j'ai utilisé comme *seed* l'entier représenté par `JOURMOISANNEE`, comme 01012023 pour le 1er janvier 2023).

Les bases de données ont été implémentée en TypeScript, et fonctionnent de manière indépendante : lors du lancement de l'application pour la première fois, elles se connectent au serveur et téléchargent les données des œuvres/lieux/patrimoines. Jusqu'alors, sur l'ancienne version, les données locales étaient incluses dans le package de l'application et il fallait faire une nouvelle mise à jour à chaque fois qu'on voulait ajouter des données. Maintenant, à chaque lancement, l'application fait des requêtes en arrière-plan pour vérifier s'il y a de nouvelles données

et les inclut en temps réel si elle en trouve. C'est beaucoup plus efficace et ça permet une autonomie des applications.

De même, j'ai mis en place un système qui télécharge automatiquement les données des utilisateurs après leur connexion : auparavant, les photos étaient seulement locales, et un utilisateur qui désinstallait l'application n'avait pas de moyen de récupérer sa collection (bien que le système de compte existait déjà). J'ai dû beaucoup travailler avec Natacha (développeuse serveur cette session) pour développer les fonctionnalités, notamment la création de nouvelles routes de l'API. Par exemple, les restrictions CORS en vigueur sur les navigateurs empêchent l'accès à certaines ressources situées sur des domaines différents pour des raisons de sécurité. En raison de cela, il m'était impossible de télécharger les images (statiques) sur le serveur et nous avons dû mettre en place une route API dédiée au transfert des images.

De même, nous avons réalisé à la fin de la session que le serveur n'acceptait pas de recevoir des fichiers plus grands que 2 mégaoctets (Mo). Avec l'augmentation de la qualité photo des appareils ainsi que l'agrandissement du stockage interne, une photo lambda pèse facilement plus de 5 Mo. De plus, étant donné qu'on a ajouté la possibilité d'envoyer une photo choisie depuis la galerie, il faut s'attendre à ce que les photos puissent atteindre les 10 Mo (un panorama par exemple a une très grande résolution). Nous avons contacté Raouf Bencheraiet pour augmenter cette restriction. Côté client (applicatif), j'ai aussi choisi d'augmenter la compression des photos pour réduire leur taille en même temps.

Conclusion

Ce projet était extrêmement instructif pour moi et je suis très heureux de l'avoir choisi. J'ai pu apprendre les bases du développement Android natif et du langage Kotlin, qui en est le langage standard, et le choix de restructuration avec Ionic en web intégré m'a permis d'apprendre l'utilisation d'un framework web, et d'améliorer mon travail en équipe. Au final, les réalisations de la session sont très différentes des objectifs fixés initialement.

L'application est finie à 90% ou 95%. Il reste surtout l'implémentation des badges (qui récompensent la collection d'un certain nombre d'œuvres), des filtres sur la carte/la liste des œuvres, et la prise en charge de l'anglais pour l'interface. J'ai beaucoup appris et je suis globalement content de mon travail, surtout étant donné le peu de temps imparti. Malgré tout, je n'ai pas pu résoudre tous les problèmes que j'ai rencontrés :

- Si l'onglet Carte a déjà été visité, et qu'on essaie de l'ouvrir aux coordonnées de la découverte du jour, la vue ne se met pas à jour et reste telle qu'on la laissée ;
- De même, si l'on a déjà ouvert l'onglet Collection et qu'on y ajoute une nouvelle photo, la collection ne se met pas à jour. Pour contrer cela, j'ai mis une solution temporaire qui permet de forcer le rafraîchissement de la page, mais idéalement ce devrait être automatique.

De manière générale, j'ai eu du mal à faire communiquer les pages entre elles et j'ai souvent trouvé des moyens de contourner ce problème, par exemple en générant dynamiquement le chemin vers les détails d'une œuvre avec `/discovery-details/TYPE/ID/` où `TYPE` est le type de la découverte (œuvre, lieu, patrimoine) et `ID` l'identifiant unique de la découverte. De cette manière, il n'y a pas de caching car l'URL est unique. Avec du recul, je pense que c'était la manière la plus élégante de résoudre ce problème, mais malheureusement ce n'est pas une solution applicable aux problèmes de la carte et de la collection.

Je recommande vivement aux futurs étudiants de lire la documentation Ionic et Vue.js, et notamment de retenir le schéma du cycle de vie d'une page Vue.js. On ne lit jamais trop de documentation ! Si un étudiant lit ces lignes, qu'il n'hésite pas à me contacter pour une question ou une explication. Certains choix ont été faits dans le design de l'application à un moment donné, ça ne signifie pas que c'était le meilleur choix possible. Parce que je me formais parallèlement au développement de l'application, certaines bonnes pratiques n'ont pas été appliquées partout alors il ne faut pas hésiter à corriger.

Voici le rendu final des cinq pages principales :