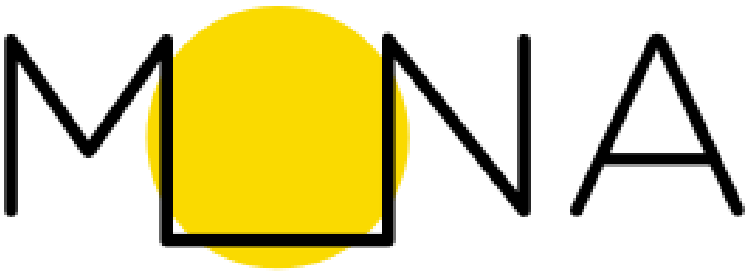
RAPPORT FINAL IFT3150

Natacha Rivière

Matricule 20255581



SESSION HIVER 2023

MAISON MONA

RAPPORT FINAL PROJET

Natacha Rivière 20255581

INTRODUCTION : LE PROJET MONA

La maison MONA est un organisme sans but lucratif qui a pour mission d'«inviter à des rencontres avec l'art, de créer un espace commun pour les échanges et de faire résonner les sens et les sensibilités de chacun ». La maison MONA a plusieurs projets dont l'organisation d'activités de médiation culturelle et la création d'une application mobile gratuite pour inviter à la découverte de l'art et des lieux culturels au Québec, et en particulier à Montréal. Cette application recense les œuvres d'art public dans la région de Montréal et propose à l'utilisateur de les découvrir, à travers une carte interactive, ou une liste.

Cette application a été créée en 2016 par Lena Krause dans le cadre d'un cours d'informatique de l'Université de Montréal. Depuis lors, plusieurs générations d'étudiants se sont succédé, sous la direction du professeur Guy Lapalme et de Lena Krause, pour poursuivre le développement de cette application. Au cours du semestre d'hiver 2023, j'ai eu le plaisir de prendre part à ce projet dans le rôle de développeuse serveur.

J'ai choisi ce projet car je suis une étudiante française en échange à Montréal pour la session d'hiver et que grâce à ce projet, je pouvais découvrir la ville de Montréal par son art public.

MON MANDAT

J'ai travaillé sur la partie serveur de l'application Mona. Mon mandat s'est scindé en trois parties. Tout d'abord, j'ai eu une phase de familiarisation avec le serveur et l'application, puis j'ai beaucoup travaillé sur l'API. Enfin, je me suis penchée sur l'interface administrateur.

I. FAMILIARISATION

Lors de mon arrivée, je n'avais jamais travaillé sur un serveur. J'étais un petit peu familière avec l'outil *ssh* mais rien de plus. J'ai donc passé quelque temps à me familiariser avec le serveur MONA et avec l'application, pour mieux en comprendre le fonctionnement. Le serveur utilise le Framework Laravel, ainsi que Vue.js pour l'interface administrateur. J'avais déjà codé en PHP, le langage utilisé par Laravel mais mes connaissances s'arrêtaient là. J'ai donc commencé par me renseigner sur les bases de Laravel, mais j'avais beaucoup de mal à lire la documentation sans avoir une idée de comment elle s'appliquait dans mon cas. J'ai donc exploré le serveur et la documentation en parallèle.

Ma toute première mission était de mettre sur le serveur MONA les sites web de compte-rendus des étudiants de l'an passé et de cette session. Cela a fait une bonne introduction à la manière dont le serveur sert les données sur internet car j'ai dû créer une nouvelle route afin de permettre d'accéder au site web de Manping, qui était situé dans un sous-dossier.

Ensuite, Simon, un ancien étudiant de MONA, m'a fait une présentation du serveur et m'a expliqué rapidement que je devais créer une instance locale du serveur MONA afin de pouvoir tester mes modifications avant de les mettre en production. Cette étape n'est pas évidente mais j'y suis parvenue grâce à un article de blog recommandé par Simon. J'ai tout de même rencontré plusieurs problèmes, par exemple l'installation requiert des versions très spécifiques de logiciels comme PHP et MySQL et plusieurs des scripts à exécuter contiennent du code trop vieux, qu'il m'a fallu mettre à jour afin de réussir à faire fonctionner le système.

Une fois accoutumée au fonctionnement général du serveur, je me suis intéressée à l'API.

II. API

Pendant nos discussions lors des réunions hebdomadaires, nous avons décidé d'essayer de développer une nouvelle application Android car l'ancienne était trop pleine de bugs et lente. Ceci nous a mené à réfléchir à une solution pour faciliter le passage d'une application à l'autre pour les utilisateurs. En effet, à ce moment-là, il était impossible de récupérer les données personnelles de son compte. Par exemple, si un utilisateur envoyait une photo au serveur pour une œuvre à partir d'un téléphone, il ne pouvait pas la récupérer en se connectant sur un autre téléphone. Cela posait donc des problèmes pour le passage à une nouvelle application, qui ne pouvait pas récupérer les données. De plus, beaucoup d'utilisateurs ont probablement oublié leur mot de passe et leur nom d'utilisateur, alors même si la récupération de données mentionnée plus haut fonctionnait, ils ne pourraient pas se connecter à leur compte. J'ai donc créé plusieurs choses sur l'API.

Premièrement, il est désormais possible de changer son mot de passe, son nom d'utilisateur et son courriel. J'ai implémenté ces routes mais leur application dans les applications n'a pas encore été faite. Ces opérations ne sont d'ailleurs réalisables que par un utilisateur connecté. Pour améliorer cette fonctionnalité, il faudrait ajouter la possibilité de changer ses informations en renseignant son courriel, mais comme aucun utilisateur de la base de données n'a renseigné d'email, cette solution n'est pas encore applicable. Afin de permettre de mettre en œuvre cette solution dans le futur, l'inscription sur les applications a été modifiée pour collecter l'adresse courriel de l'utilisateur.

Par ailleurs, il est très important de trouver un moyen de forcer les utilisateurs iOS à changer leur nom d'utilisateur et mot de passe car à cause d'un bug de l'application, au lieu d'envoyer le nom d'utilisateur et le mot de passe définis par l'utilisateur au serveur, l'application envoyait des valeurs par défaut, donc inconnues des utilisateurs. Par conséquent, les utilisateurs ne peuvent en aucun cas récupérer leurs données. J'ai pu vérifier dans la base de données que ce malheureux bug concerne 1772 comptes.

Ensuite, afin de rendre possible la récupération des données utilisateurs à la connexion, j'ai parcouru plus en détail les routes présentes dans l'API. Je me suis rendu compte que cette option existait déjà en partie, mais que personne ne semblait le savoir. En effet, en étant connecté, un utilisateur peut faire une requête au serveur, qui lui envoie la liste des œuvres collectionnées ainsi que les notes, commentaires et le nom de la photo associée. J'ai donc documenté cette possibilité sur le GitHub du serveur afin de permettre aux développeurs application de s'en servir.

Je pensais que cette route déterrée permettrait aux développeurs d'implémenter sans mal la récupération des données mais un dernier obstacle se dressait sur cette route. En effet, les photos utilisateurs sont toutes stockées dans un dossier public du serveur. C'est-à-dire qu'à condition de connaître le nom de l'image, on peut y accéder avec l'url https://picasso.iro.umontreal.ca/~mona/storage/photos/NomDeLaPhoto.jpg. Malheureusement, lorsqu'on accède à cette url, le serveur renvoie directement le fichier, sans aucun en-tête http. Ceci est problématique car sans les en-têtes d'autorisation à tous les domaines, l'application Ionic, qui est donc un navigateur, n'a pas le droit d'accéder à ces routes. Cela signifie que ces images sont accessibles depuis un navigateur, mais pas depuis l'application. Il m'a fallu trouver une solution à ce problème.

Cette solution est la suivante : créer une route qui sert les photos d'utilisateurs. Malheureusement, comme dit précédemment, on ne peut pas simplement envoyer un fichier, car alors les autorisations nécessaires ne sont pas présentes. Lors de mes premiers tests, j'obtenais tout simplement une erreur « header method does not exist ». J'ai fini par comprendre que pour toutes les réponses de l'api, le serveur ajoutait par défaut les en-têtes nécessaires. Lorsque j'essayais d'envoyer un fichier, il voulait y ajouter des en-têtes, ce qui n'est pas possible, créant ainsi une erreur. Lorsque j'ai voulu créer une exception pour les *BinaryFileResponse*, qui correspondait au type de réponse que j'envoyais, cela à fait dysfonctionner l'identification par API. Comme cette fonctionnalité est gérée par défaut par Laravel et que je n'ai pas trouvé comment, je n'ai pas insisté sur cette voie.

Il fallait donc trouver un autre moyen d'envoyer un fichier, en y mettant des en-têtes. La solution que j'ai donc implémentée est d'envoyer les données binaires des photos directement. Ce n'est donc pas un fichier, et l'ajout d'en-tête fonctionne. Cette manière de fonctionner impose aux développeurs application de convertir en photo les données qu'ils reçoivent. Il y a peut-être une meilleure manière de faire, mais je n'ai pas trouvé comment, et Gaspard, le développeur Android, a réussi à utiliser cette route sans difficulté donc je suis plutôt satisfaite du résultat.

La dernière modification significative apportée à l'API a découlé des tests utilisateurs réalisés en fin de session. Nous avons rassemblé une petite dizaine de personnes pour venir tester nos applications, et Lena nous a mené le long d'un petit parcours autour de l'université pour découvrir les œuvres qui nous entourent. J'ai remarqué en parcourant l'interface admin et la base de données que seules quelques photos envoyées par l'application Android avaient été effectivement reçues par le serveur. Gaspard et moi avons réalisé de nombreux tests pour finalement comprendre que les photos ne s'envoyaient pas car elles étaient trop grosses. En effet, par défaut, le serveur n'accepte pas les images de plus de 2Mo, ce qui est très peu. C'est pourquoi seules les photos zoomées ou de moins bonne qualité étaient effectivement postées. En regardant les codes Swift et Kotlin, nous avons réalisé que la solution adoptée auparavant était de compresser à 70% les photos avant de les envoyer, ce qui nous a semblé un pari hasardeux. En effet, il n'y a aucune vérification de la taille du fichier envoyé, et compresser autant est très décevant. J'ai donc contacté Raouf Bencheraiet, technicien du DIRO, afin de modifier cette limite. Il serait alors pertinent pour les prochains étudiants de se renseigner sur la taille allouée au serveur MONA et sur l'occupation actuelle de la place pour faire une estimation de la taille qui devrait être autorisée.

Enfin, j'ai commencé à effectuer une mise à jour des badges de l'application, il fallait en ajouter ainsi que modifier les descriptions de chaque badge. Je n'ai pas encore terminé ces modifications mais j'en aurai le loisir après la fin de session.

III. INTERFACE ADMINISTRATEUR

La troisième partie de mon mandat a porté sur l'interface administrateur de MONA. Avant tout, cette application est à but de recherche en histoire des arts. En effet, l'objectif est de collecter des données sur la perception de l'art public à Montréal. Ceci justifie l'existence de cette interface web. Cette interface recense toutes les photos, toutes les notes et tous les commentaires faits par les utilisateurs de l'application et les rassemble de façon lisible. Cette page n'est accessible qu'après identification afin de privatiser les données de nos utilisateurs.

Voici ce à quoi elle ressemble (figure 1) :

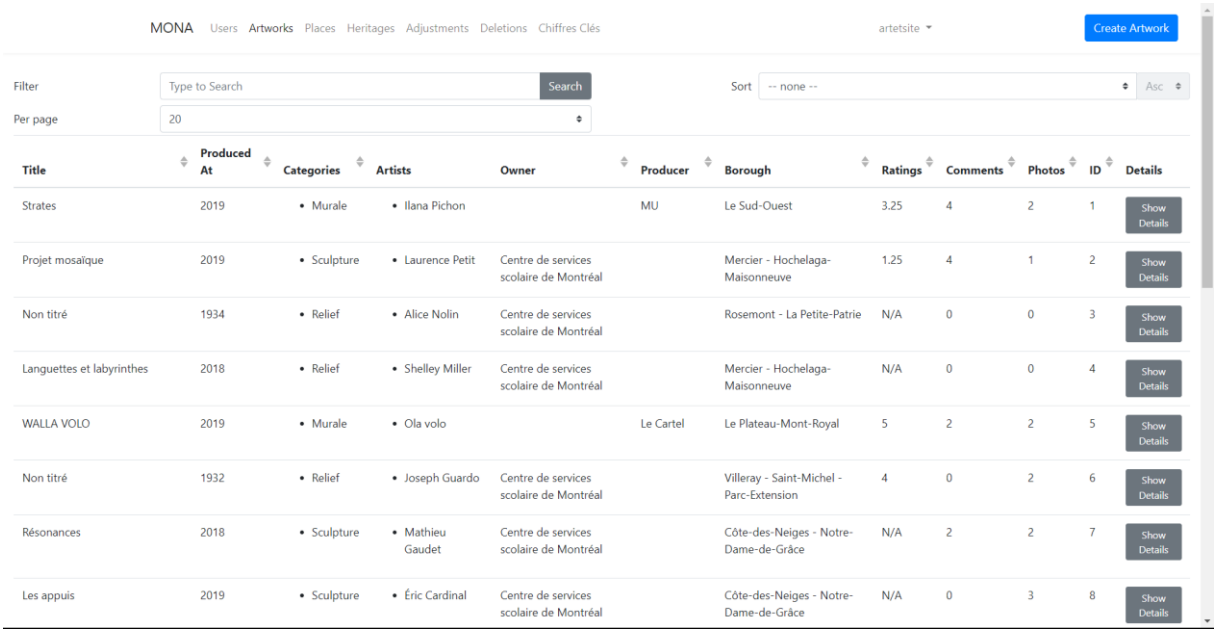


FIGURE 1 : PAGE PRINCIPALE DE L'INTERFACE ADMINISTRATEUR

Elle est en partie construite avec Vue.js et en html/CSS.

Mon objectif était de rendre cette interface plus lisible pour des fins de recherche. J'ai donc commencé par résoudre des problèmes qui existaient avant d'implémenter de nouvelles fonctionnalités.

Premièrement, il est possible de trier les œuvres selon toute une liste de critères : le titre, la date de production, la catégorie à laquelle elle appartient (par exemple sculpture, mural, etc.), l'artiste, et cætera. Un des tris importants de cette interface est le tri par nombre de photos prises. Ce tri-là ne fonctionnait pas, ou pas complètement. Lorsqu'il était sélectionné, il rangeait les œuvres par « booléen ». C'est-à-dire qu'il mettait à la fin (ou au début en fonction de l'ordre demandé) les œuvres qui n'avaient pas de photo, et au début, celles qui en avaient.

Ces options étaient gérées par Vue.js, n'étant pas du tout familière avec cet outil, je me suis replongée dans de la documentation avant de pouvoir régler ce problème. Il s'est donc avéré que le tri était réalisé par une fonction par défaut, et qu'elle essayait de comparer des listes. La solution choisie par le langage a donc été de mettre les types nuls après les listes non vides, ce qui équivalait à un test vrai ou faux, d'où le tri par « booléen ». Il fallait donc le forcer à compter le nombre d'item dans chaque liste avant de comparer ces valeurs. Désormais, le tri fonctionne parfaitement.

La deuxième chose que j'ai corrigé était la barre de recherche. Elle était terrible à utiliser car au lieu d'attendre que l'utilisateur aie terminé d'écrire sa recherche, elle faisait une requête. C'est-à-dire qu'elle effectuait une recherche dans la base de données à chaque caractère entré et mettait à jour la page à chaque fois. C'était terriblement long et on ne pouvait pas entrer un mot complet fluidement. De plus le bouton à droite de la barre

de recherche n'était pas « Search » mais « Clear » ce qui est très incohérent avec les standards, même si je comprends qu'au vu du fonctionnement de la barre, on n'ait pas voulu effacer caractère par caractère une recherche. Après de nombreuses recherches dans la documentation, j'ai fini par rendre la barre de recherche *lazy* c'est-à-dire qu'elle ne fait rien tant qu'elle n'a pas perdu le focus, donc en appuyant sur entrée ou en cliquant ailleurs sur la page (d'où l'intérêt du bouton « Search » qui ne sert à rien d'autre que d'inciter à cliquer ailleurs que sur la barre). A ce moment-là seulement, la recherche se fait.

Un troisième problème était que les pages des œuvres étaient bien plus complètes que les pages des lieux ou des patrimoines. Ceci est lié au fait que ces deux dernières ont été ajoutées à MONA bien après les œuvres alors elles n'ont pas été autant complétées. J'ai donc uniformisé les trois types de pages. C'était une opération facile, car le code existait déjà, il suffisait de le réadapter.

En annexe, une capture d'écran de l'apparence des pages d'œuvre désormais (figure 4).

Auparavant il y avait un autre tableau sur cette page, qui n'avait pas de titre. Après avoir lu le code de ce tableau, j'ai compris que c'était un tableau qui affichait les œuvres les plus proches. Malheureusement, à cause des calculs effectués pour récupérer ces œuvres, la page était extrêmement longue à charger. En effet, le système parcourait toutes les œuvres, calculait la distance avec celle sélectionnée et ne gardait que celle à moins de 500 mètres. Etant donné que le serveur compte 3746 œuvres, lieux ou patrimoine, c'était bien trop conséquent comme calcul. Ce tableau ne m'a pas semblé suffisamment utile pour développer une solution alternative et j'ai décidé de le supprimer. En revanche, depuis lors, certaines pages d'œuvres sont inaccessibles. Par exemple, la page de l'œuvre 1. Le serveur renvoie une erreur 500 et je n'ai jamais réussi à diagnostiquer pourquoi.

Après avoir travaillé sur ce qui existait déjà, j'ai implémenté de nouvelles fonctionnalités, les tags. Le principe est de pouvoir ajouter des mots-clés, les tags, aux œuvres ou aux photos afin de les catégoriser selon les critères souhaités par les chercheurs. Pour cela, il fallait ajouter le type tags à la base de données ainsi que les liens avec les autres tables existantes. J'ai alors affronté les migrations. Une migration est contenue dans un fichier, elle recense toutes les commandes SQL effectuées pour modifier la base de données. Ainsi, les modifications peuvent être effectuées de la même manière sur deux bases de données identiques. Et elles laissent une trace de ce qui existe. J'ai eu beaucoup de mal à écrire ma première migration car comme SQL est une base de données relationnelle et que je créais des tables avec des contraintes de clés étrangères, il fallait qu'il y ait une cohérence interne des types. Ainsi, les migrations fonctionnaient seulement en partie et cela créait encore plus de bugs. Heureusement que j'avais mon propre serveur local pour mes tests car autrement j'aurai peut-être fait une bêtise grave sur le serveur. J'ai dû recréer ma base de données de zéro car je ne comprenais plus ce qui se passait. Une fois diagnostiqué le problème de type, ma migration fonctionnait très bien et j'ai pu mettre le tout en production.

En revanche, à la suite de cela, je suis restée un long moment bloquée sur un problème, j'avais oublié d'utiliser la fonction *url()* qui préfixe l'url donné par la racine du serveur, alors mes modifications fonctionnaient en local, car je n'ai pas de préfixe, mais pas sur le serveur, sur lequel il y a *~mona/*. Je pensais alors que les routes PUT de l'interface web (car ce sont quasiment les seules routes PUT du serveur) ne fonctionnaient pas. De plus, l'unique autre route PUT est buguée (celle qui permet de modifier les œuvres depuis l'interface) et je n'ai pas regardé pourquoi. Je cherchais donc une solution de ce côté avant de réaliser l'erreur que j'avais faite.

Désormais, on peut donc très facilement accéder à cette page (figure 2) pour ajouter un tag à une photo. L'équivalent existe pour ajouter un tag à une œuvre, la seule différence est l'absence d'image.

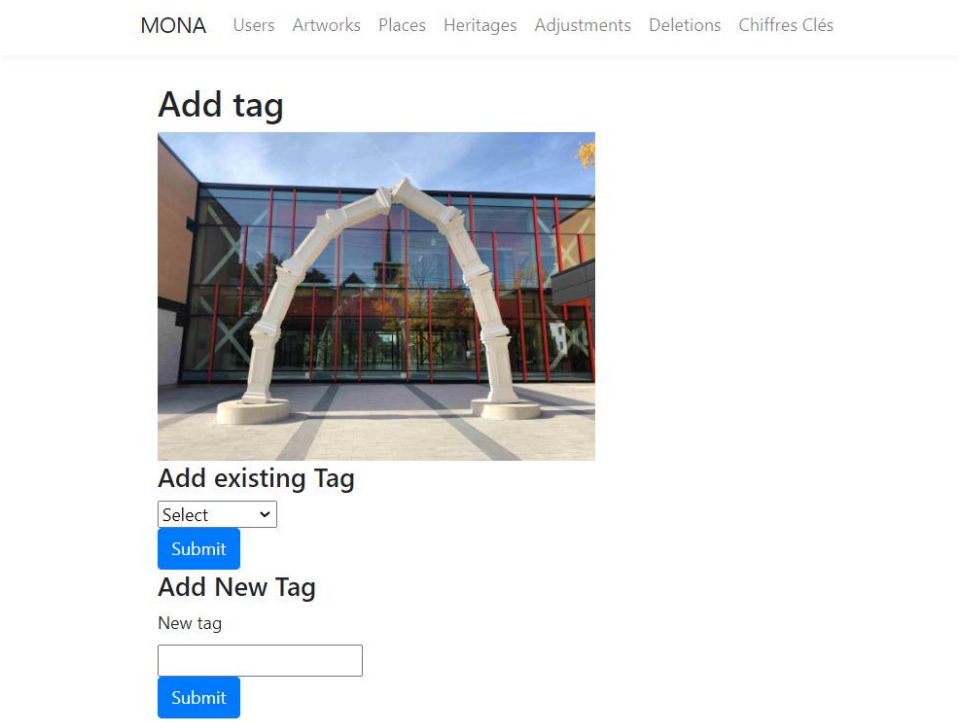
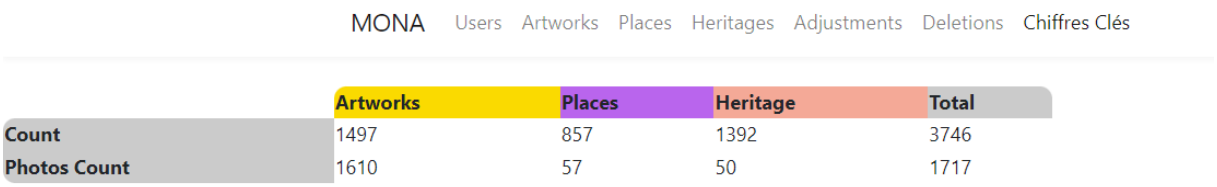


FIGURE 2 : PAGE D'AJOUT DE TAG

La prochaine étape qu'il faudra réaliser, probablement par un prochain étudiant, sera d'ajouter la possibilité de voir les tags des œuvres directement sur la page de liste et de pouvoir chercher et trier par tag.

Toujours dans l'objectif d'améliorer la lisibilité de l'interface admin, j'ai implémenté une page Chiffres Clés (figure 3), qu'il faut encore compléter. L'objectif est de recenser les chiffres importants de l'application en un seul endroit. Cette page est assez longue à charger, je ne sais pas comment l'accélérer car parfois le chargement est trop long et renvoie une erreur 500.



	Artworks	Places	Heritage	Total
Count	1497	857	1392	3746
Photos Count	1610	57	50	1717

FIGURE 3 : PAGE CHIFFRES CLES

Un problème récurrent auquel j'ai fait face, qui a provoqué souvent beaucoup de frustration était que mes changements ne se reflétaient pas toujours sur l'interface. Quasiment chaque fois, j'avais oublié de vider le cache de l'application ou de lancer la recompilation des fichiers vue. Ce problème est apparu dès le début de la session et m'a poursuivie tout du long, même une fois diagnostiqué. C'était souvent une bonne piste pour résoudre mes problèmes.

CONCLUSION

J'ai beaucoup apprécié prendre part à ce projet, dans une équipe très sympathique et dont le but est intéressant. J'ai beaucoup appris grâce à ce projet, je pense que ce que j'y ai acquis me servira grandement à l'avenir. Je suis très contente d'avoir l'occasion de poursuivre et peut-être terminer les missions qui m'ont été confiée grâce à un court contrat à la fin de la session.

Mes plus grandes difficultés venaient toujours de mon manque d'expérience avec les outils utilisés et du fait que je suis arrivée sur un projet déjà commencé. Il était compliqué de comprendre le fonctionnement du serveur sans avoir eu l'occasion de le configurer moi-même. Il fallait toujours que je cherche assez longtemps dans la documentation et dans le code pour comprendre le fonctionnement avant de pouvoir y faire des améliorations et des ajouts. Je suis consciente que j'aurai souvent à travailler sur des projets déjà commencés à l'avenir et je ne sous-estimerai plus l'importance de la documentation.

QUELQUES IDEES POUR LA SUITE

Bien que le serveur fonctionne généralement très bien, il y a quelques tâches qu'il serait bien de réaliser :

- Ajouter les pages create / edit des patrimoines et des lieux
- Débuguer la page edit (je ne me suis pas penchée sur la question car la session approchait de sa fin)
- Trouver la source de l'erreur 500 sur quelques œuvres
- Compléter la page chiffres clés (et l'accélérer si possible)
- Ajouter une liste des artistes et une page *show* des artistes avec les liens vers leurs œuvres (je compte m'y pencher après la fin de session)

ANNEXE

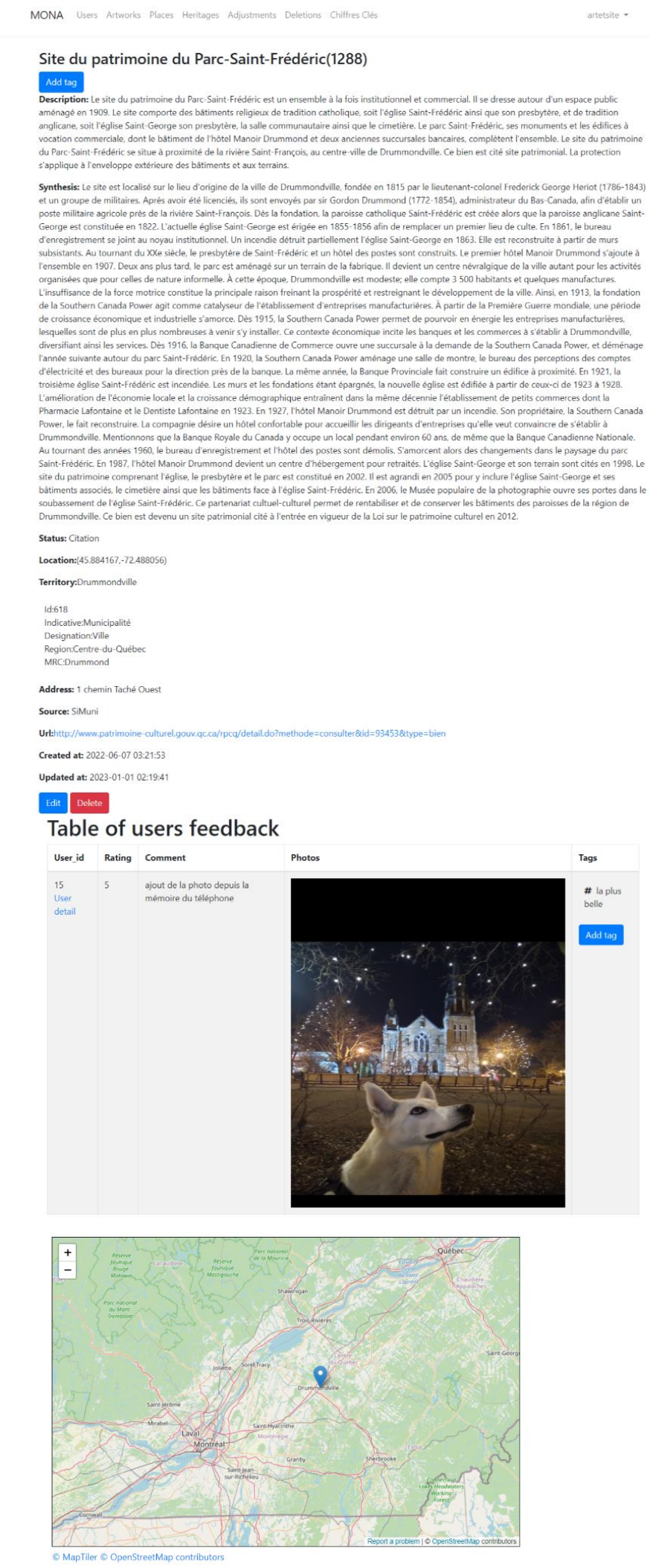


FIGURE 4 : PAGE D'OEUVRE