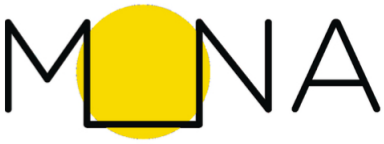
Rapport Final IFT3150

Simon Janssen

27 avril 2021

1 Introduction



Le projet d'informatique auquel j'ai contribué pour mon projet en IFT3150 était MONA. C'est une application mobile qui encourage les gens à visiter leur environnement afin d'y découvrir les œuvres d'art public. Un peu comme le grand succès, Pokemon GO, les utilisateurs sont invités à se promener et photographier les œuvres qu'ils rencontrent. De plus, il existe une interface web avec laquelle des chercheurs en histoire de l'art pourront voir les rétroactions anonymisées des utilisateurs pour leur recherche. MONA a été créé par Lena Krause dans un cours et étoffé par la suite lors d'un Hackathon. Elle en demeure la chef de développement.

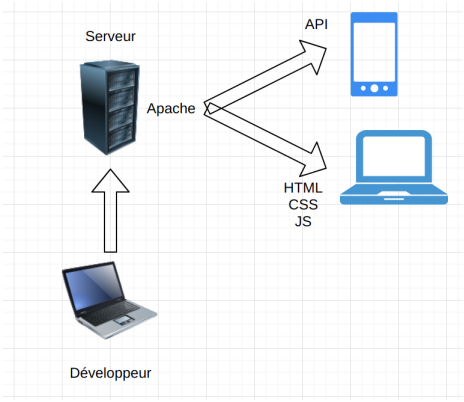
Le projet m'a attiré parce que je commençais à m'intéresser à l'histoire de l'art au moment où j'ai vu l'annonce. Mon frère étudie en histoire de l'art et il me faisait découvrir des aspects de l'art que je n'avais jamais remarqués. MONA me semblait un projet idéal pour développer ma culture de l'art ainsi que pour appliquer mes connaissances en informatique.

En arrivant sur le projet, j'avais hâte de travailler en équipe car ça serait ma première fois à le faire. J'avais des grandes attentes par rapport à ce que j'allais pouvoir contribuer. À ce moment, mon expérience se limitait à des connaissances en GNU/Linux et l'entretien d'un petit serveur pour mon site personnel.

Mes apports au projet peuvent se diviser en deux grands groupes: réviser le déploiement et implémenter l'API des badges.

2 Le déploiement est mort, vive le déploiement!

À mon arrivée, la méthode de déploiement en production consistait à copier manuellement par SSH tous les changements du code. Ce n'était évidemment pas optimal parce que c'était long, les erreurs se faisaient régulièrement et les bases de code de développement et de production accumulaient des différences avec chaque mise à jour.



La première tâche qui m'a été attribuée était celle de créer un mode de déploiement plus simple. Préférablement, ce déploiement se ferait avec GIT étant donné que c'est un logiciel libre que tous les développeurs auraient installé.

Une tâche qui est, semble-t-il, plutôt simple. J'ai suivi un guide sur github qui expliquait comment le serveur, grâce à un dépôt GIT et un HOOK spécial, permettrait le déploiement via une commande PUSH du bord du développeur. Par cette commande, les différences entre l'état du code déployé et l'état du code de développement seraient copiées au serveur. J'ai choisi cette méthode par ce qu'elle me semblait être une solution non bouleversante qui minimiserait l'utilisation des ressources du serveur.

```
/home/mona
├── MONA-LAST
│   └── ...
└── mona-prod.git
    └── hooks
        └── post-receive
```

Ci-dessus, on voit le répertoire de fichiers comme il est sur le serveur. Le dossier MONA-LAST contient le code du projet et le dépôt GIT s'appelle mona-prod.git. Lorsque le dépôt détecte un changement grâce au hook, il copie les nouveaux fichiers dans MONA-LAST.

Cependant, parce que le dépôt GIT sur le serveur est initialement vide, la première fois que la commande de déploiement est lancée, le code du serveur

sera estompé par le code de développement. Normalement, ce ne serait pas un problème car les bases de code du serveur et de développement seraient identiques à quelques différences près. Dans notre cas, parce que les deux versions accumulaient des différences avec chaque mise à jour manuelle de SSH, la première utilisation de la commande a tué le serveur. En fait, le serveur n'était pas tué, tout fonctionnait sur le serveur, mais toutes tentatives de s'y connecter par l'API ou par HTTP n'aboutissaient qu'en erreur 403.

J'ai contacté Mathieu Perron, un ancien du MONA, pour me donner un coup de main pour diagnostiquer les problèmes. Nous avons aussi contacté Raouf Bencheraiet de l'équipe de TI qui nous a grandement aidé. Le plus gros problème était que APACHE, le serveur web que nous utilisons pour desservir les documents, n'avait pas les permissions nécessaires pour compiler les fichiers PHP de notre projet. Comme le backend est presque entièrement écrit en PHP, tout était inaccessible par les utilisateurs.

Jusqu'au matin, nous étions en train de réparer l'erreur. Nous avons changé la configuration d'APACHE pour qu'elle utilise FPM, ce qui lui a permis de compiler et d'envoyer les fichiers PHP, résolvant le plus gros des problèmes.

Vous vous demandez peut-être comment il se fait que les fichiers PHP fonctionnaient correctement avant. En fait, avant cette erreur, tous les fichiers du serveur avaient des permissions élevées qui leur permettait d'accéder aux fichiers PHP sans problème. Ceci est par contre une grande faille de sécurité car ses fichiers ont un accès total à la machine contenant notre serveur. Quand la commande de déploiement a été lancée, elle a remplacé tous ces fichiers à permissions totales, d'où l'erreur. La résolution de cette erreur n'était pas la fin de cette tâche, il restait qu'une partie du site du MONA, l'interface administrateur, demeurait inaccessible. Après plusieurs diagnostics, j'ai découvert que composer, le gestionnaire de paquets qu'on utilise pour notre projet, n'était pas installé sur le serveur. Les paquets du serveur n'étaient pas à jour.

En installant une version locale du gestionnaire et en faisant une mise-à-jour, le dernier des problèmes à été réglé (on utilise encore une ancienne version de composer, mais le serveur et le développement sont en parité et c'est ce qui est important).

Suite à cette expérience, j'ai incorporé une commande additionnelle dans le HOOK qui permet le déploiement au serveur; maintenant, lorsqu'on envoie une mise-à-jour du code, une copie de sauvegarde du code du serveur se fera automatiquement. Ce genre de problème ne devrait donc plus se passer dorénavant.

Pour lancer la commande, il suffit de se rendre dans la branche MASTER de son répertoire GIT local, puis de lancer GIT PUSH PRODUCTION MASTER.

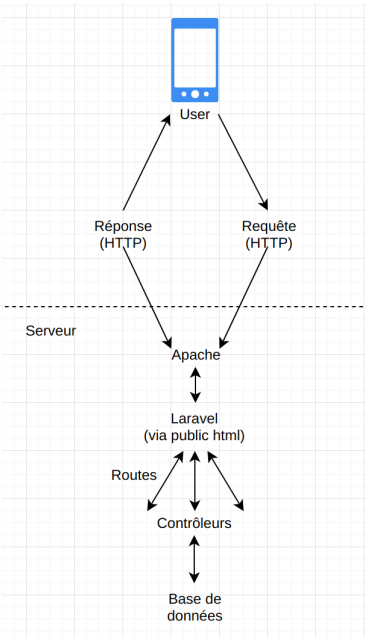
Yuning a eu quelques difficultés inconstantes avec le nouveau système mais les problèmes ne durent pas. Il resterait à estomper ces problèmes mineurs.

3 Félicitations! Voici un badge!

Ma deuxième grosse tâche était la modification de l'API pour qu'il puisse retourner des données de badges. Ces badges seront affichés dans l'application pour encourager les utilisateurs à collectionner des œuvres, et pour les féliciter suite à la collection d'œuvres.

Avec l'équipe de développement d'application, je suis tombé sur une idée de badges qui seraient composés d'une fonction générale et où la fonctionnalité plus spécifique serait définie par des arguments supplémentaires. Certains de ses arguments seraient obligatoires comme le nombre d'œuvres à collectionner pour un badge dont la fonction est la collection d'œuvres. D'autres arguments seraient optionnels et serviraient à filtrer les actions des utilisateurs qui contribueraient à la collection d'un badge. Un exemple serait l'argument du quartier qui limite les collections d'œuvres importantes à celles faites dans un quartier spécifique.

Suite à une première rencontre pour concevoir la structure des badges afin qu'ils soient extensibles, j'ai commencé l'implémentation concrète.



D'abord, j'ai créé une route dans le framework de Laravel qui permettra aux requêtes faites avec l'URL .../api/badges de se faire traiter. Ceci s'est passé sans trop de problèmes.

Ensuite, j'ai ajouté dans les ressources de Laravel l'objet badge, ce qui m'a permis d'ajouter un contrôleur qui traite les requêtes pour l'API de badges.

Finalement j'ai définie dans la base de données une table pour les badges avec des migrations de Laravel. Les migrations servent à suivre les changements faits à la base de données.

Ces trois étapes se sont faites à répétition pour itérer le fonctionnement des badges.

Une fois que tout a été mis en place, il restait à peupler la table de badges à partir d'un fichier qui serait facile à modifier pour ajouter des nouveaux badges. Avant que je commence à travailler sur les badges, il existait un google sheet contenant plusieurs types de badges qui devraient se retrouver dans le résultat final. J'ai tenté en vain d'en tirer un fichier .CSV qui pourrait être lu par le job Laravel qui mettra les rangées dans la base de données. Finalement, j'ai fait un tout nouveau fichier .CSV et je l'ai rempli avec les données que nous avions.

Afin de transmettre les badges du .CSV dans la base de données afin qu'ils puissent se faire servir par l'API, il faut une job Laravel. Un job est un script dont la tâche, dans ce cas-ci, est de copier ligne par ligne les données de badges du fichier à la base de données. Nous nous en servons dans MONA pour importer les œuvres des données ouvertes d'art du Québec. Dans le futur, nous pourrions modifier les données des badges de la base de données par l'interface admin.

J'ai un peu aidé Yuning avec la mise-à-jour des jobs d'œuvres auparavant, ce qui m'a donné les bases pour faire de même avec les badges.

En ce moment le job Laravel comporte encore des bogues, il reste à les corriger. De plus, l'équipe de développement d'application a voulu que je fasse des modifications mineures à la structure des données envoyées par l'API.

4 Conclusion et vision du futur

J'ai adoré travailler sur le projet MONA! Je pense que c'est un projet super qui ne fera qu'améliorer et j'ai bien hâte de voir ce que ça deviendra!

Quant à ma contribution, j'ai été surpris par des imprévus bouleversants qui m'ont certes ralenti. C'est une réalité du développement, semble-t-il. J'ai appris à persévérer malgré ces obstacles, et j'en ai appris énormément sur le développement en backend d'un API. De plus, travailler en groupe était un grand plaisir.

Il reste à corriger les quelques petits bogues qui restent dans les badges, et sinon enrichir la base de données avec des métadonnées qui permettront des requêtes plus complexes.