Université de Montréal

Département d'informatique et de recherche opérationnelle

IFT3150 – Projet informatique

Rapport final

Développement de la recherche avancée de l'interface d'administration de MONA

Mariama Amadou Amadou, 20289908

Superviseur : Guy Lapalme

Encadrante : Léna Krause

Été 2026

Résumé

Ce projet porte sur le développement de la recherche avancée de l'interface d'administration de MONA, une plateforme consacrée à la découverte, à la documentation et à la valorisation de l'art public, du patrimoine et des lieux culturels. L'objectif principal était d'améliorer les outils mis à la disposition des administrateurs afin de faciliter la consultation et le filtrage des données contenues dans le système.

Le travail réalisé s'est appuyé sur une application existante développée principalement avec Laravel, Vue.js et MariaDB. Une analyse du code, de la structure de la base de données et des besoins exprimés par l'équipe a d'abord été effectuée. Cette analyse a permis d'identifier les différents types d'entités à rechercher, notamment les œuvres d'art, les artistes, les utilisateurs, les lieux patrimoniaux et les lieux culturels, ainsi que les principaux critères de filtrage nécessaires.

La solution proposée repose sur une nouvelle interface de recherche avancée intégrée à l'espace d'administration. Elle comprend un contrôleur Laravel chargé du traitement des recherches ainsi que plusieurs composants Vue.js responsables de l'affichage des filtres, des résultats et des outils de navigation. Une attention particulière a été accordée à la réutilisation du code existant, à la séparation des responsabilités et à la compatibilité avec l'architecture actuelle de MONA.

Table des matières

1 Introduction **3**
1.1 Contexte . . . 3
1.2 Problématique et motivation . . . 3
1.3 Objectifs du projet . . . 4

2 Analyse des besoins et étude préliminaire **5**
2.1 Exigences fonctionnelles et non fonctionnelles . . . 5
2.2 Démarche d'analyse et de conception . . . 6
2.3 Étude de solutions existantes ou similaires . . . 6
2.4 Analyse du système préexistant . . . 7

3 Conception **8**
3.1 Architecture du système . . . 8
3.2 Modèle de données . . . 9
3.3 Choix technologiques . . . 9
3.4 Maquette de l'interface . . . 9

4 Implémentation et réalisation technique **11**
4.1 Présentation synthétique de l'implémentation . . . 11
4.2 Particularités vis-à-vis de la conception . . . 11
4.3 Description des composants principaux . . . 13

5 Évaluation **16**
5.1 Tests unitaires et fonctionnels . . . 16
5.2 Évaluation de performance . . . 16
5.3 Évaluation d'utilisabilité . . . 17
5.4 Comparaison avec les objectifs . . . 17

6 Discussion critique **18**

7 Conclusion **19**

1. Introduction

1.1 Contexte

MONA est un projet initié par Léna Krause visant à faciliter la découverte de l'art public à Montréal. Le projet se compose de deux volets : une application mobile destinée au grand public, permettant de localiser et de découvrir des œuvres d'art, des lieux patrimoniaux et des lieux culturels ; et une interface d'administration web permettant à l'équipe de gérer le contenu de la base de données (œuvres, artistes, utilisateurs, contributions, etc.).

La base de données de MONA regroupe près de 8255 enregistrements répartis entre différents types de contenu (œuvres d'art, lieux patrimoniaux, lieux culturels, artistes et utilisateurs), chacun avec ses propres attributs et relations.

1.2 Problématique et motivation

L'interface d'administration permet d'accéder à un volume important de données, mais ne disposait pas d'un outil permettant d'effectuer des recherches avancées en combinant plusieurs critères. Cette limitation rend plus difficile l'exploration ciblée des données, notamment lorsqu'une recherche doit porter simultanément sur plusieurs caractéristiques.

Le problème est d'autant plus complexe que les données manipulées sont hétérogènes. Une recherche peut concerner une œuvre d'art, un lieu patrimonial, un lieu culturel, un artiste ou un utilisateur, et les critères pertinents diffèrent selon le type de contenu.

La motivation principale du projet est donc de fournir aux administrateurs et aux chercheurs un moyen plus flexible et efficace d'interroger ces données. Cette fonctionnalité doit permettre de filtrer les différents types de contenus à partir de critères adaptés à leurs caractéristiques tout en offrant une expérience de recherche cohérente.

1.3 Objectifs du projet

L'objectif principal du projet est de concevoir et d'intégrer une fonctionnalité de recherche avancée à l'interface d'administration existante de MONA.

Plus précisément, le projet vise à :

- un système de filtres couvrant l'ensemble des types de contenu et de leurs attributs ;
- une présentation des résultats adaptée au type d'éléments affichés, avec pagination et personnalisation des colonnes ;
- une vue cartographique permettant de visualiser géographiquement les résultats de recherche.

2. Analyse des besoins et étude préliminaire

2.1 Exigences fonctionnelles et non fonctionnelles

Les exigences fonctionnelles identifiées avec l'équipe couvrent l'ensemble des types de contenu gérés par MONA : œuvres d'art, lieux patrimoniaux, lieux culturels, artistes et utilisateurs. Pour chacun, la recherche avancée devait permettre de filtrer sur les informations générales (type, identifiant, dates), sur les attributs spécifiques (support, technique, matériaux, catégorie, fonction ou usage selon le type), sur la localisation (région, territoire, municipalité, adresse) ainsi que sur des statistiques de contribution (notes, commentaires, photos). Une recherche géographique, ainsi qu'une visualisation cartographique des résultats, faisaient également partie des besoins exprimés.

Besoin	**Description**
Recherche multi-types	Rechercher parmi les œuvres d'art, les lieux patrimoniaux, les lieux culturels, les artistes et les utilisateurs.
Filtres avancés	Combiner des critères généraux et des critères propres aux différents contenus.
Recherche géographique	Rechercher selon la localisation ou dans un rayon autour de coordonnées géographiques.
Exploitation des résultats	Consulter, paginer, personnaliser et exporter les résultats obtenus.
Visualisation cartographique	Représenter géographiquement les résultats disposant de coordonnées.

TABLE 2.1 – Principaux besoins de la recherche avancée

Sur le plan non fonctionnel, la solution devait rester performante face à un volume de plusieurs milliers d'enregistrements, s'intégrer à l'architecture existante (Laravel, Vue.js 2, MariaDB) sans rupture de compatibilité, et offrir une interface suffisamment flexible pour être adaptée à mesure que de nouveaux besoins de filtrage apparaîtraient.

2.2 Démarche d'analyse et de conception

La première étape a consisté à comprendre le fonctionnement de MONA et son environnement technique. L'interface d'administration, le code existant et la base de données ont été étudiés afin d'identifier les fonctionnalités réutilisables et de vérifier quels critères pouvaient réellement être implémentés.

Un inventaire des critères a ensuite été réalisé et les données ont été explorées avec DBeaver (Annexe A.4). Cette étape a montré l'importance de travailler à partir des valeurs réellement présentes dans la base plutôt que de supposer leur contenu. Elle a notamment servi à déterminer les valeurs disponibles pour plusieurs caractéristiques des œuvres, du patrimoine et des artistes.

Une première maquette statique a finalement été construite afin d'organiser les nombreux critères avant leur implémentation.

2.3 Étude de solutions existantes ou similaires

Des benchmarks réalisés à partir de plateformes de collections culturelles, notamment le Rijksmuseum, la Réunion des musées nationaux et Gallica, ont servi de référence pour la conception de l'interface. Ces benchmarks ont été réalisés par Corélie (développeuse serveur) et Camila (direction de la recherche), et j'en ai exploité les constats pour orienter la conception de la recherche avancée.

Plusieurs principes ont été retenus : conserver les filtres accessibles après une recherche, afficher clairement les critères actifs, privilégier des valeurs prédéfinies lorsqu'elles sont connues et adapter les filtres affichés au type de contenu sélectionné. Ces principes ont ensuite été adaptés aux contraintes propres aux données de MONA.

2.4 Analyse du système préexistant

MONA utilise Laravel côté serveur et Vue.js 2 avec BootstrapVue pour son interface d'administration. La nouvelle fonctionnalité devait donc être intégrée à cette architecture.

L'analyse du système a également montré que les différents types de contenus ne possèdent pas une structure uniforme. Des informations conceptuellement similaires, notamment la localisation, peuvent être stockées différemment selon le modèle : pour `Artwork` et `Place`, l'adresse est un attribut directement porté par le modèle, tandis que pour `Heritage`, elle est représentée par une relation vers un modèle `Adresse` distinct. La recherche devait par conséquent fournir une interface commune tout en adaptant le traitement aux données interrogées.

Cette analyse a par ailleurs permis d'identifier plusieurs éléments existants réutilisables pour la nouvelle fonctionnalité : la fiche détaillée des découvertes, le mécanisme d'exportation des données en JSON, ainsi que les tableaux existants de l'interface d'administration, qui ont servi de base d'inspiration pour la conception de la grille de résultats.

3. Conception

3.1 Architecture du système

La recherche avancée a été conçue comme une extension de l'interface d'administration existante. Son architecture repose sur trois niveaux principaux : l'interface Vue.js, le traitement de la recherche côté Laravel et les modèles donnant accès à la base de données.

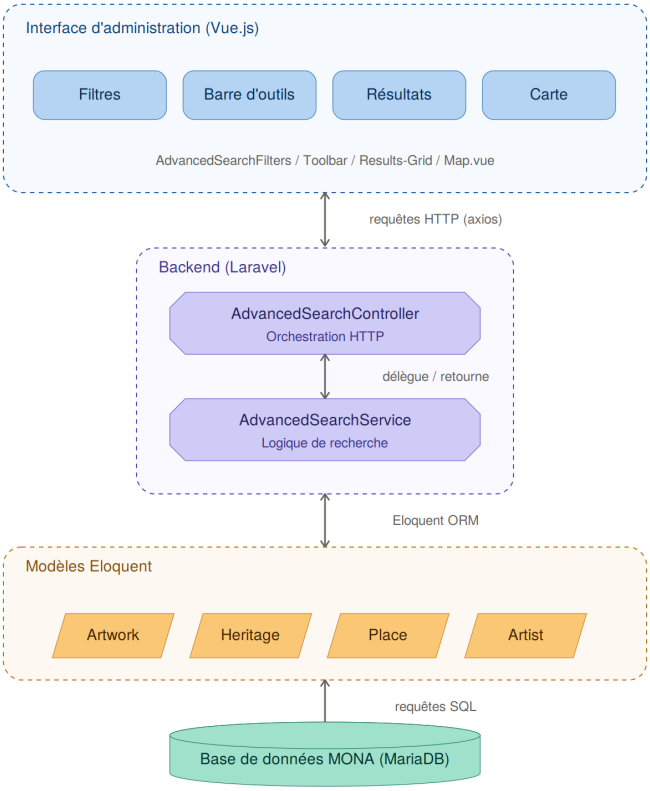

FIGURE 3.1 – Architecture générale du système : interface Vue.js, backend Laravel (contrôleur, service) et modèles Eloquent, connectés à la base de données MariaDB.

3.2 Modèle de données

La recherche repose sur le modèle de données existant de MONA. Les principaux contenus sont représentés par les modèles `Artwork`, `Heritage` et `Place`. À ceux-ci s'ajoutent les artistes et les utilisateurs ainsi que les données relatives à leurs contributions.

Certaines propriétés peuvent être filtrées directement à partir des attributs des modèles, tandis que d'autres doivent être calculées à partir de relations. C'est le cas des statistiques associées aux artistes et aux utilisateurs qui nécessitent, par exemple, de prendre en compte les œuvres et les contributions qui leur sont associées.

Les données géographiques présentent également une particularité : leur structure varie selon le type de contenu. La solution masque cette différence à l'utilisateur en proposant des critères communs, tandis que les requêtes sont adaptées côté serveur.

3.3 Choix technologiques

Laravel, Vue.js 2 et BootstrapVue étant déjà utilisés par MONA, la fonctionnalité a été développée avec ces technologies afin de conserver la compatibilité avec le système existant et de réutiliser ses composants et modèles.

Leaflet a été utilisé pour la représentation cartographique des résultats. Pour la recherche par proximité, le calcul de distance est réalisé directement dans la base de données avec `ST_Distance_Sphere`. Cette solution évite l'introduction d'un service externe de géocodage lorsque les coordonnées du point de recherche sont connues.

3.4 Maquette de l'interface

La première maquette organisait les critères en différentes sections afin d'éviter d'afficher simultanément l'ensemble des champs. Elle a ensuite évolué à partir des benchmarks, des données réelles et des retours obtenus pendant le développement.

L'interface finale distingue notamment l'état initial, centré sur les filtres, de l'état suivant une recherche, où les filtres restent accessibles à côté des résultats. Les critères sélectionnés sont également affichés sous forme de badges pouvant être retirés individuellement.

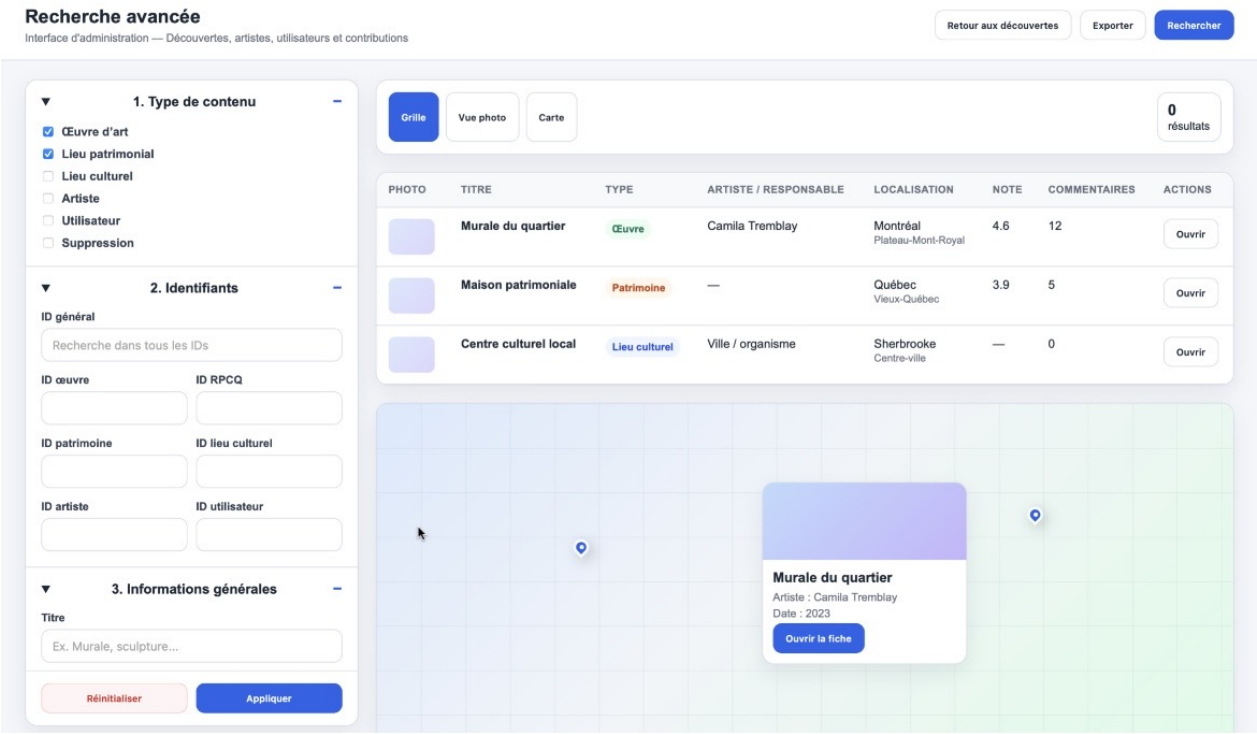

FIGURE 3.2 – Maquette de l'interface : filtres à gauche, résultats en mode Grille (tableau) ou Carte selon le mode sélectionné. Le rectangle bleu-vert représente le mode Carte, avec la fiche sommaire d'un marqueur sélectionné.

4. Implémentation et réalisation technique

4.1 Présentation synthétique de l'implémentation

L'implémentation s'est déroulée de façon incrémentale, en commençant par les filtres backend (informations générales, dates, œuvres d'art, patrimoine, lieux culturels, localisation, recherche géographique, artistes, utilisateurs, notes/commentaires/photos, suppression), puis par la grille de résultats et enfin par la vue cartographique.

La logique de filtrage a d'abord été implémentée directement dans le contrôleur, avant d'être extraite dans `AdvancedSearchService` afin d'améliorer la lisibilité et la testabilité du code.

La pagination a été conçue en deux temps : une première requête légère récupère uniquement les identifiants et une clé de tri pour l'ensemble des types confondus, afin d'obtenir un tri et un compte global cohérents ; une seconde requête charge ensuite uniquement les éléments de la page courante avec leurs relations complètes.

4.2 Particularités vis-à-vis de la conception

La figure 4.1 présente l'interface finale de la recherche avancée. Les critères sont regroupés dans le panneau de gauche, tandis que la partie droite permet de consulter les résultats et de choisir leur mode d'affichage.

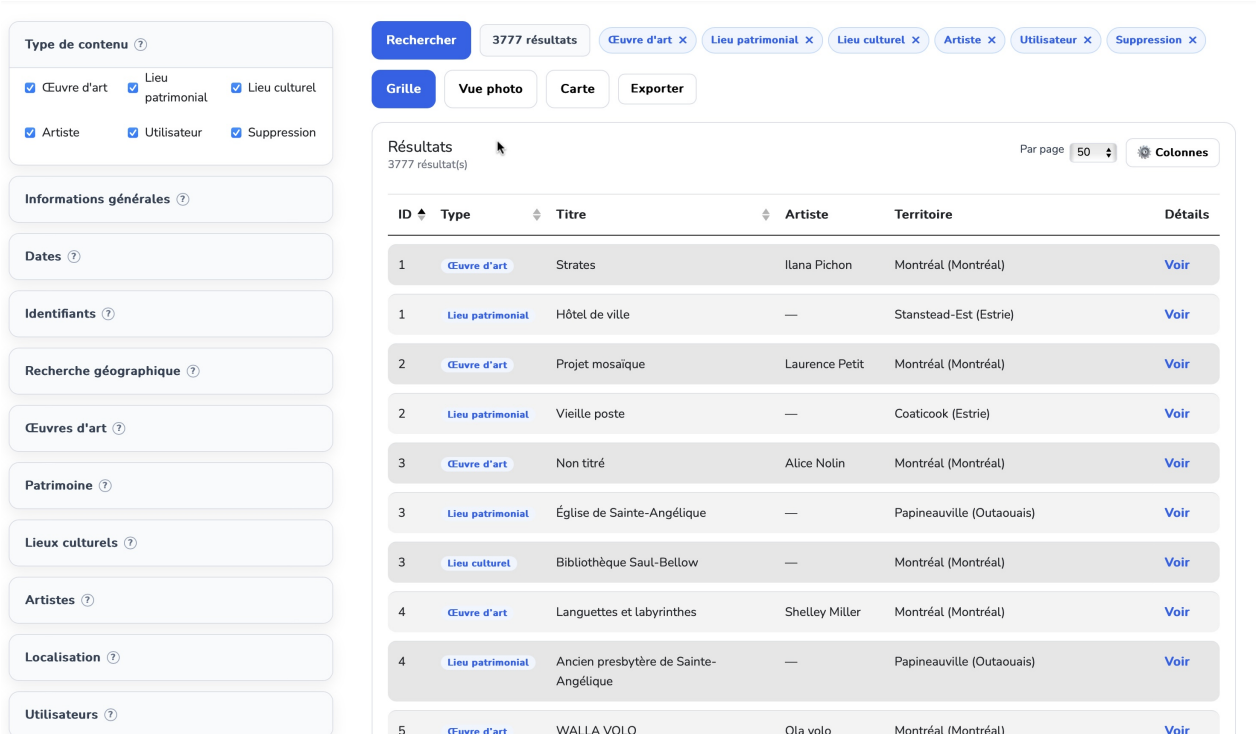

FIGURE 4.1 – Interface finale de la recherche avancée : panneau de filtres et grille de résultats.

Certains ajustements ont été nécessaires par rapport à la conception initiale, notamment dans l'organisation des filtres. Par exemple, le champ « Mention », initialement considéré comme une information générale, a finalement été déplacé vers la section « Œuvres d'art », puisqu'il ne concerne que ce type de contenu. Ce changement a également nécessité une adaptation de la logique serveur afin que l'utilisation de ce filtre limite correctement les types de résultats pouvant être retournés.

La grille de résultats a également dû être adaptée selon le type de contenu retourné. La table générale, utilisée pour les œuvres, les lieux patrimoniaux et les lieux culturels, conserve la présentation historique de l'interface, mais bascule automatiquement vers une table dédiée lorsque les résultats affichés sont exclusivement des artistes ou exclusivement des utilisateurs.

4.3 Description des composants principaux

AdvancedSearchController.php et AdvancedSearchService.php – Reçoivent les critères de recherche envoyés par le frontend, construisent dynamiquement les requêtes selon les filtres actifs et retournent les résultats paginés.

AdvancedSearchFilters.vue – Formulaire de saisie des critères, organisé en sections correspondant aux types de contenu et à leurs attributs propres.

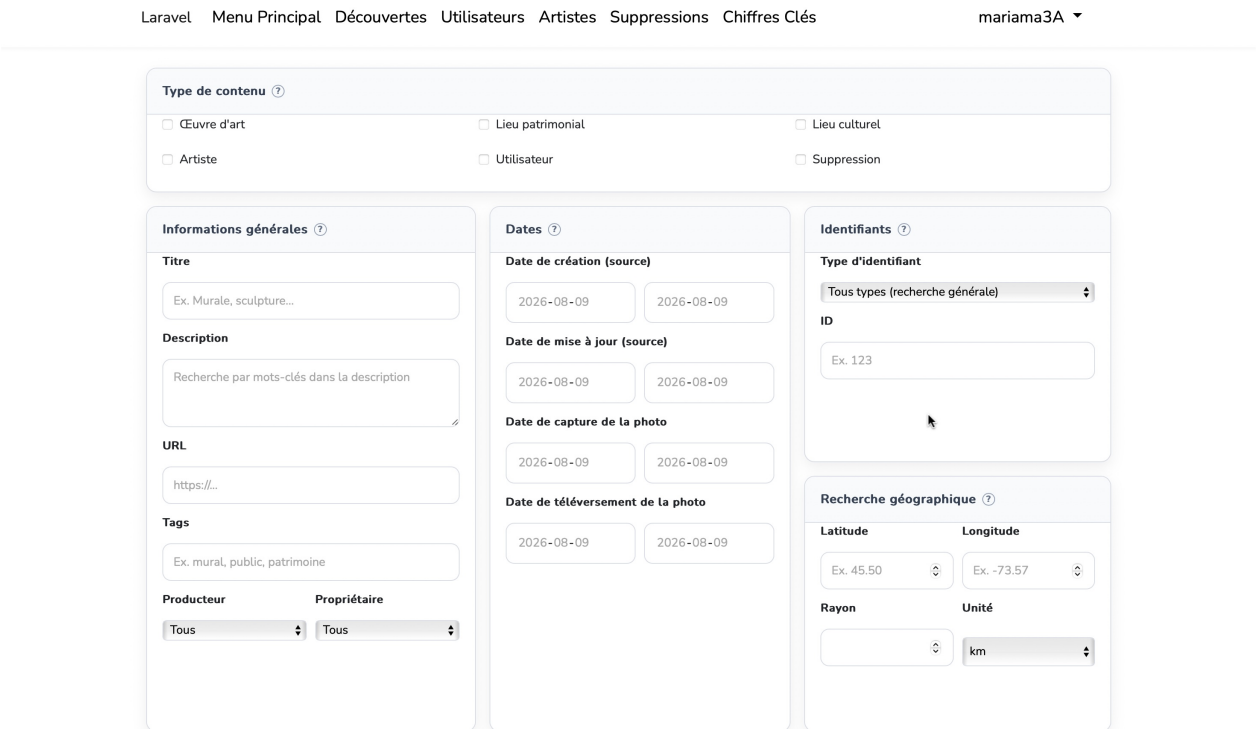

FIGURE 4.2 – Panneau de filtres de la recherche avancée, organisé en sections repliables par type de contenu.

AdvancedSearchToolbar.vue – Barre d'actions permettant de lancer la recherche et d'ajuster le nombre de résultats par page (20/50/100).

AdvancedSearchResults.vue et AdvancedSearchGrid.vue – Affichage des résultats sous forme de tableau, avec des colonnes adaptées au type de contenu et personnalisables par l'utilisateur.

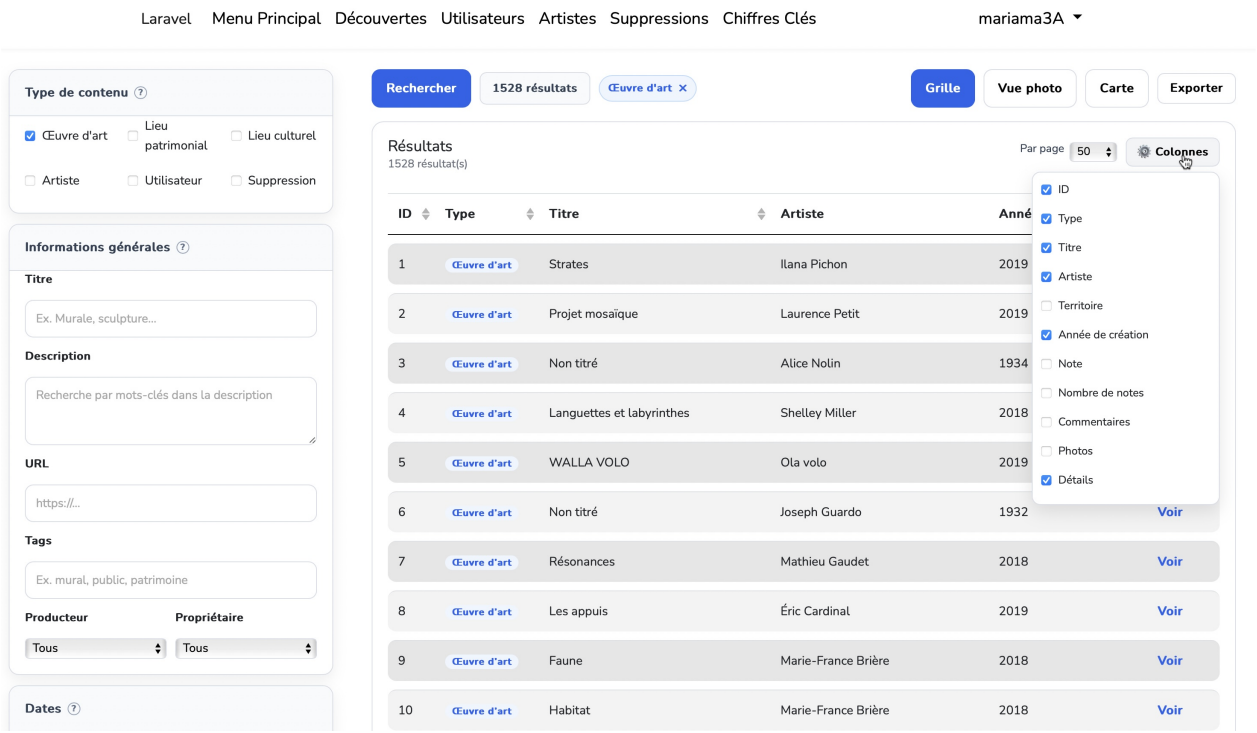

FIGURE 4.3 – Grille de résultats, avec colonnes personnalisables selon le type de contenu affiché.

AdvancedSearchMap.vue – Vue cartographique basée sur Leaflet et `leaflet.markercluster`, avec des marqueurs colorés selon le type d'élément. Elle s'appuie sur deux routes dédiées : une première retournant l'ensemble des marqueurs légers (identifiant, type, titre, coordonnées) sans pagination pour l'ensemble du jeu de données, et une seconde retournant le détail d'un élément pour l'affichage de son popup au clic.

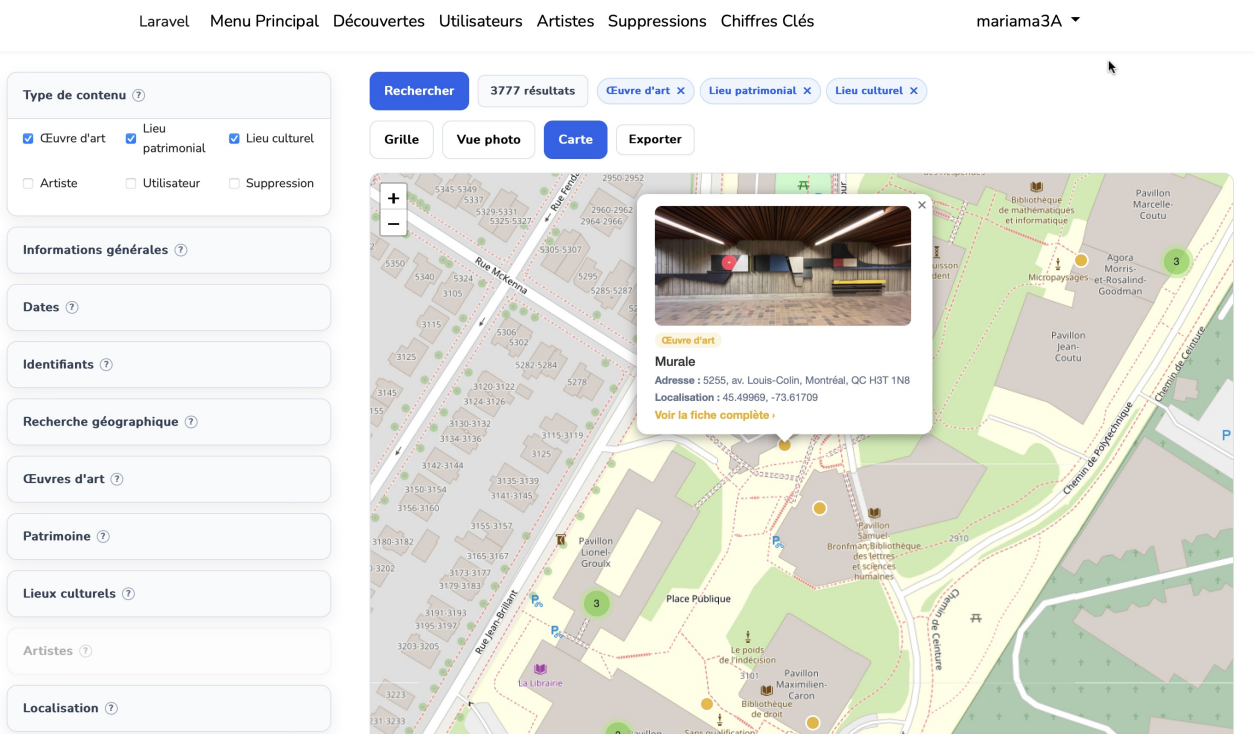

FIGURE 4.4 – Vue cartographique des résultats avec regroupement des marqueurs par proximité et affichage de la fiche sommaire d'un élément sélectionné.

5. Évaluation

5.1 Tests unitaires et fonctionnels

Une suite de 18 tests unitaires PHPUnit a été écrite pour la méthode `computeEligibleTypes()` du service `AdvancedSearchService`. Cette méthode a été ciblée en priorité car elle détermine, à partir des filtres actifs, quels types de contenu (œuvre, patrimoine, lieu culturel, artiste, utilisateur, suppression) restent éligibles à la recherche – un comportement central et particulièrement sensible aux régressions, puisqu'une erreur y fait silencieusement disparaître ou apparaître des catégories entières de résultats.

Les méthodes de construction de requêtes (`applyXxxFilters()`, `buildTypeBuilders()`) n'ont pas fait l'objet de tests automatisés à ce stade : leur validation nécessiterait des tests d'intégration avec une base de données réelle, un chantier plus lourd qui n'a pas encore été entamé. La validation fonctionnelle de ces parties du système s'est donc faite manuellement, en particulier lors des sessions de validation hebdomadaires avec l'équipe.

5.2 Évaluation de performance

Le temps de réponse du backend a été mesuré directement en production locale (environnement Docker Sail), en chronométrant l'exécution des deux endpoints les plus sollicités sur l'ensemble du jeu de données (environ 8255 enregistrements, tous types confondus). Pour chaque endpoint, trois mesures ont été prises successivement.

- **Recherche sans filtre** (`search()`, avec hydratation complète et pagination) : une moyenne d'environ **336 ms**.
- **Chargement des marqueurs de la carte** (`mapPins()`, retour de champs légers sans pagination) : une moyenne d'environ **277 ms**.

Ces résultats montrent que `mapPins()` est légèrement plus rapide que `search()` malgré l'absence de pagination, ce qui s'explique par le fait qu'il ne retourne que des champs minimaux

(identifiant, type, titre, coordonnées). Dans les deux cas, le temps de réponse reste largement sous la seconde et n'introduit pas de latence perceptible pour l'utilisateur dans le contexte d'une interface d'administration.

Ces mesures restent indicatives : elles ont été prises manuellement via des horodatages (`microtime()`) insérés temporairement dans le code. Une évaluation plus rigoureuse (test de charge avec plusieurs utilisateurs concurrents, profilage des requêtes SQL) constitue une piste d'amélioration future.

5.3 Évaluation d'utilisabilité

L'interface a fait l'objet d'une validation hebdomadaire avec l'ensemble de l'équipe tout au long du projet, plutôt que d'un test d'utilisabilité formel ponctuel. Camila, responsable de la direction de la recherche, a joué un rôle particulièrement actif dans ces validations. Cette démarche itérative a permis d'ajuster progressivement l'interface – par exemple la réorganisation de certains filtres entre sections, ou la révision des colonnes affichées par défaut dans les tableaux de résultats – en fonction des retours recueillis semaine après semaine.

5.4 Comparaison avec les objectifs

Objectif	**État**
Recherche sur plusieurs types de contenus	Réalisé
Combinaison de filtres avancés	Réalisé
Affinement d'une recherche existante	Réalisé
Exportation des résultats	Réalisé
Visualisation cartographique	Réalisé
Tests automatisés complets	Partiel
Évaluation quantitative des performances	Partiel

TABLE 5.1 – Comparaison avec les principaux objectifs du projet

Les tests automatisés couvrent la méthode la plus critique du service (`computeEligibleTypes()`) mais pas encore les méthodes de construction de requêtes, qui demanderaient des tests d'intégration ; et l'évaluation de performance repose sur des mesures manuelles ponctuelles plutôt que sur un test de charge formel.

6. Discussion critique

Le principal défi du projet a été l'hétérogénéité des données. Des critères qui paraissent similaires du point de vue de l'utilisateur ne correspondent pas nécessairement aux mêmes structures dans les différents modèles. La qualité des données a également eu un impact sur le développement : certaines listes contenaient des doublons ou des valeurs manifestement incorrectes. Le choix a été de nettoyer les suggestions proposées par l'interface sans modifier les données originales.

La recherche géographique a également nécessité plusieurs ajustements, notamment en raison des conventions de représentation des coordonnées. Le choix d'une recherche à partir d'une latitude et d'une longitude plutôt que d'une adresse évite une dépendance à un service de géocodage externe, mais constitue un compromis en matière de facilité d'utilisation.

Une autre difficulté est apparue avec l'augmentation progressive du nombre de filtres. Le contrôleur, initialement suffisant, était devenu trop volumineux et mélangeait l'orchestration HTTP avec la logique de recherche. La création d'`AdvancedSearchService` a permis de mieux séparer ces responsabilités et de faciliter l'ajout de tests. Cette évolution montre qu'une architecture adaptée au début d'un développement peut devoir être réévaluée lorsque la complexité réelle de la fonctionnalité apparaît.

Certaines décisions restent ouvertes. L'ordre d'affichage des résultats mixtes en constitue un exemple : les identifiants appartenant à différents types de contenus n'ont pas nécessairement de signification commune. Plutôt que d'introduire une règle de classement arbitraire, le comportement actuel a été conservé en attendant une décision avec l'équipe.

Sur le plan personnel, ce projet m'a surtout permis d'apprendre à intervenir dans une application existante. Au-delà de l'approfondissement de Laravel et de Vue.js, j'ai dû comprendre un modèle de données réel, réutiliser du code existant, adapter la conception aux contraintes découvertes pendant le développement et refactoriser une solution devenue progressivement plus complexe. Cette expérience m'a montré l'importance d'une démarche itérative dans laquelle l'analyse, l'implémentation et la validation se poursuivent tout au long du développement.

7. Conclusion

Ce projet a permis de concevoir et de développer une fonctionnalité de recherche avancée intégrée à l'interface d'administration de MONA. La solution permet désormais de combiner de nombreux critères sur plusieurs types de contenus et d'exploiter les résultats sous forme de tableau, d'export ou de représentation cartographique.

Le projet m'a permis d'approfondir Laravel et Vue.js tout en travaillant sur des enjeux plus généraux liés à l'intégration dans un système existant, à l'hétérogénéité des données, à la conception de requêtes complexes et à la maintenabilité du code.

La fonctionnalité pourrait être poursuivie par l'ajout de tests automatisés couvrant davantage de filtres, une évaluation quantitative des performances et des tests d'utilisabilité auprès des utilisateurs de l'interface. Ces travaux permettraient de consolider la solution avant ses futures évolutions dans MONA.