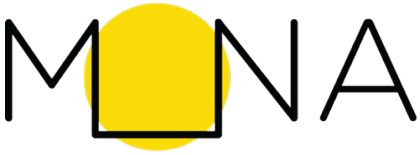
Université de Montréal

Département d'informatique et de recherche opérationnelle

IFT3150 — Projet informatique



Génération automatique d'un fichier de correctifs SQL

ANISSA OULD FERROUKH, 20311207

Superviseur : **GUY LAPALME**

Session Hiver 2026

Résumé

Le projet MONA (Musée d'art numérique et ouvert Accessible) est une application mobile et un serveur backend permettant la découverte d'œuvres d'art public à Montréal. Les données de l'application sont régulièrement réimportées depuis des sources externes, ce qui écrase les corrections manuelles apportées par les expertes du domaine. Ce projet vise à mettre en place un mécanisme automatique permettant de préserver ces corrections lors de chaque réimportation.

La solution développée repose sur l'exploitation du système d'ajustements (trait Adjustable et modèle Adjustment) déjà présent dans le backend Laravel, mais non activé. Ce mécanisme a été implémenté, corrigé et étendu afin d'intercepter les opérations de création (INSERT), de mise à jour (UPDATE) et, ultérieurement, de suppression (DELETE) effectuées sur les principales entités de l'application (Artwork, Artist, Heritage, Place, Badge). Un service AdjustmentSqlExporter a été développé pour extraire ces enregistrements et les écrire dans un fichier de correctifs SQL, mis à jour en temps réel à chaque modification.

Les fonctionnalités clés livrées incluent : la génération automatique de requêtes SQL de type UPDATE et INSERT, la compatibilité avec l'interface d'administration et l'API (guards web et api), la gestion correcte des valeurs NULL, et l'organisation du fichier de patchs par date et par utilisateur. La gestion du DELETE reste à finaliser suite aux discussions en cours avec l'équipe mais ne sera pas dans le cadre de ce projet.

Table des matières

1. Introduction

1.1 Contexte

La Maison MONA est un organisme à but non lucratif inclusif et dynamique, dont la mission est de:

- Inviter à des rencontres avec l'art.
- Créer un espace commun pour les échanges.
- Faire résonner les sens et les sensibilités de chacun·e

MONA est un projet né en 2016 sous l'initiative de Lena Krause, chercheuse en histoire de l'art spécialisée dans le numérique. Il se concrétise alors sous la forme d'une application mobile qui permet aux utilisateurs de découvrir les œuvres d'art public de la ville de Montréal. L'application s'appuie maintenant sur un serveur backend développé avec le framework Laravel (PHP), exposant une API REST consommée par les clients mobiles iOS et Android.

Les données alimentant l'application proviennent de plusieurs sources hétérogènes (villes, musées, registres patrimoniaux) et sont donc importées dans la base de données via un pipeline ETL. Ce pipeline est exécuté périodiquement pour mettre à jour le catalogue d'œuvres d'arts, d'artistes, de lieux patrimoniaux et de lieux culturels.

L'équipe de développement est composée de développeurs et d'expertes du domaine de l'art (historiennes de l'art). Le projet s'appuie sur des outils tels que GitHub, Element (messagerie), pCloud, Figma et une infrastructure de déploiement sur serveur.

1.2 Problématique et motivation

Les données importées depuis les sources externes ne correspondent pas toujours à la réalité du terrain ou aux standards de qualité attendus. Les expertes du domaine et l'équipe de contenu procèdent donc régulièrement à des corrections manuelles directement dans l'application, via l'interface d'administration ou l'API à partir de requêtres cURL ou Postman.

Or, chaque nouvelle réimportation de données écrase ces corrections manuelles. Les données corrigées sont alors perdues à chaque mise à jour du pipeline ETL. Cette situation engendre un travail de requalification répétitif et source d'erreurs, et nuit par consequent à la qualité et à la cohérence des données présentées aux utilisateurs.

La problématique centrale de ce projet est donc la suivante : comment capturer automatiquement les corrections manuelles apportées aux données, afin de pouvoir les réappliquer de manière fiable lors de chaque réimportation ?

1.3 Objectifs du projet

- Analyser le mécanisme d'ajustements existant dans le backend MONA et comprendre son état de développement.
- Implémenter et corriger ce mécanisme afin qu'il intercepte fidèlement les modifications apportées aux entités principales.
- Développer un service capable de transformer ces enregistrements en requêtes SQL valides (UPDATE, INSERT) et de les écrire dans un fichier de correctifs.
- Assurer la compatibilité du système avec l'interface d'administration et l'API REST.
- Étendre le système à toutes les entités concernées : Artwork, Artist, Heritage, Place, Badge.

- Documenter la démarche et les décisions techniques pour faciliter la reprise par l'équipe.

2. Analyse des besoins et étude préliminaire

2.1 Exigences fonctionnelles

Les exigences fonctionnelles ont été identifiées progressivement au cours des premières semaines, en collaboration avec les membres de l'équipe de développement :

- Le système doit intercepter toute modification (UPDATE) effectuée sur une entité métier depuis l'interface admin ou via l'API.
- Le système doit intercepter toute création (INSERT) d'une nouvelle entité.
- Les corrections doivent être enregistrées dans un fichier SQL, mis à jour en temps réel.
- Le fichier de patch doit être organisé par date et par nom d'utilisateur pour faciliter le diagnostic.
- Le système doit correctement distinguer une valeur NULL volontaire d'un champ absent de la requête.
- Les champs obligatoires (ex: titre, latitude, longitude, source) doivent retourner une erreur de validation s'ils sont mis à NULL.
- Le système doit être généralisable à toutes les entités : Artwork, Artist, Heritage, Place, Badge.

2.2 Exigences non fonctionnelles

- Le mécanisme doit être transparent: il ne doit pas modifier le comportement visible de l'API pour les utilisateurs.
- Le code doit s'intégrer proprement dans l'architecture Laravel existante, sans créer de dépendances inutiles.
- Le fichier de patch doit rester lisible et maintenable par un humain.
- Le système doit être testable via des requêtes cURL ou Postman sans outillage supplémentaire.
- Le code doit être cohérent avec les conventions du projet et faciliter la revue par l'équipe.

2.3 Démarche d'analyse et de conception

La démarche a débuté par une phase d'exploration approfondie du code existant. Les premières semaines (semaines 1 à 4) ont été consacrées à la prise de connaissance du projet MONA, de son architecture, de ses outils de collaboration et de son pipeline ETL. Les rencontres avec Simon qui est responsable de l'infrastructure m'ont permis de comprendre le fonctionnement du serveur et le processus de déploiement. Les rencontres avec Corélie qui est développeuse backend m'ont permis de comprendre l'API v3 et son architecture.

Les semaines 5 à 7 ont été dédiées à une étude plus approfondie du pipeline ETL et à la compréhension du rôle du fichier de correctifs SQL dans ce processus. Cette analyse a révélé que le fichier de corrections est central dans l'intégration des données et repose sur des transformations SQL manuelles.

À partir de la semaine 8, l'exploration du code a permis d'identifier le mécanisme d'ajustements déjà présent mais non activé (trait Adjustable, modèle Adjustment, AdjustmentController non implémenté). La semaine 9 a été consacrée à l'évaluation de deux approches alternatives: l'utilisation du système d'ajustements vs l'extraction directe dans les contrôleurs. La décision, prise en semaine 10, a été de conserver la méthode basée sur les ajustements, pour ses avantages en termes de traçabilité et d'historique.

2.4 Étude de solutions existantes

Deux approches ont été évaluées expérimentalement (semaine 9) :

- **Approche 1** : Système d'ajustements (table adjustments) : les modifications sont d'abord enregistrées dans la table adjustments par le trait Adjustable, puis converties en SQL. Cette approche offre une traçabilité complète (horodatage, utilisateur, valeurs avant/après) mais présentait initialement plusieurs problèmes (valeurs parasites, double déclenchement …etc).

- **Approche 2** : Extraction directe dans les contrôleurs : les modifications sont comparées dans le contrôleur avant/après mise à jour, et les commandes SQL sont générées immédiatement. Plus simple à implémenter, mais sans historique et sans traçabilité complète.

La conclusion de cette analyse comparative a conduit à retenir l'approche 1, complétée par les corrections nécessaires pour éliminer les artefacts observés.

Par ailleurs, les outils ETL Apache Airflow et OpenRefine ont été étudiés en début de projet (semaines 2-3). Apache Airflow a été écarté en raison de sa complexité. OpenRefine a été considéré comme un outil d'exploration mais n'a finalement pas été intégré directement dans la solution retenue.

2.5 Analyse du code préexistant

L'analyse du code existant a révélé les éléments suivants :

- Le trait Adjustable était présent mais présentait des problèmes : écrasement des champs lors d'un update en deux temps (update() + save()), déclenchement intempestif de l'événement updating, enregistrement de valeurs parasites (NULL, dimensions, location).
- Le modèle Adjustment existait et utilisait une relation polymorphique (morphTo), mais n'était pas activement utilisé.
- L'AdjustmentController était vide (méthodes non implémentées).
- La méthode *Auth::check()* utilisée dans le trait ne supportait que le guard web, excluant les requêtes API authentifiées par token Bearer.
- Les méthodes *store()* de plusieurs contrôleurs effectuaient un *create()* suivi d'un *save()* séparé, ce qui générait deux événements et deux lignes dans le patch.

3. Conception

3.1 Architecture du système

L'architecture de la solution s'inscrit entièrement dans le backend Laravel (mona-server). Elle repose sur les composantes suivantes, organisées selon le patron Trait/Modèle/Service :

- **Trait Adjustable :** ajouté aux modèles Artwork, Artist, Heritage, Place, Badge. Il écoute les événements Eloquent (creating, updating, deleting) et crée un enregistrement Adjustment avant/après chaque opération.

- **Modèle Adjustment :** entité polymorphique stockant les modifications (champs adjustable_id, adjustable_type, before, after au format JSON, created_at, utilisateur).

- **Service AdjustmentSqlExporter :** lit la table adjustments, génère les requêtes SQL correspondantes (UPDATE ou INSERT), et écrit dans le fichier adjustments_patch.sql.

- **Contrôleurs** (ArtworkController, ArtistController, HeritageController, PlaceController, BadgeController) : modifiés pour appeler le service d'export et pour corriger le comportement de create/update.

Les interactions entre composantes suivent le flux suivant : requête HTTP (admin ou API) → Contrôleur → Modèle Eloquent → Trait Adjustable → Table adjustments + Service AdjustmentSqlExporter → Fichier adjustments_patch.sql.

3.2 Modèle de données

- La table adjustments contient les colonnes suivantes :

```
MariaDB [mona_app]> DESCRIBE adjustments;
+----------------+---------------------+------+-----+---------+----------------+
| Field          | Type                | Null | Key | Default | Extra          |
+----------------+---------------------+------+-----+---------+----------------+
| id             | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| user_id        | bigint(20) unsigned | NO   | MUL | NULL    |                |
| adjustable_id  | int(10) unsigned    | NO   |     | NULL    |                |
| adjustable_type| varchar(191)        | NO   | MUL | NULL    |                |
| before         | longtext            | NO   |     | NULL    |                |
| after          | longtext            | NO   |     | NULL    |                |
| created_at     | timestamp           | YES  |     | NULL    |                |
| updated_at     | timestamp           | YES  |     | NULL    |                |
+----------------+---------------------+------+-----+---------+----------------+
8 rows in set (0.002 sec)

MariaDB [mona_app]>
```

  - Les autres tables impliquées sont les tables métier : artworks, artists, heritage_sites, places, badges, ainsi que les tables de relations (geo_addresses, subcategories, territories, etc.).

- Le fichier adjustments_patch.sql utilise un format SQL standard, avec des commentaires d'organisation par date et par utilisateur.

```
-- ====================================
-- 2026-04-14
-- ====================================

-- Modifié par : TESTanissa
UPDATE `artworks` SET `title` = 'Titre test API 1' WHERE `id` = 11;
-- Modifié par : TESTanissa
UPDATE `artworks` SET `title` = 'L''horizon' WHERE `id` = 11;
-- Modifié par : TESTanissa
UPDATE `artworks` SET `title` = 'Strates test' WHERE `id` = 1;
-- Modifié par : TESTanissa
UPDATE `artworks` SET `title` = 'Strates test API' WHERE `id` = 1;
```

3.3 Choix technologiques

Le projet s'inscrit entièrement dans le stack technologique existant de MONA :

- **Laravel (PHP)** : framework backend utilisé pour l'ensemble de la logique métier, les contrôleurs, les modèles Eloquent et les traits.
- **MariaDB :** base de données relationnelle utilisée pour stocker les entités et la table adjustments
- **Système d'événements Eloquent (Laravel)** : les événements creating, updating, deleting ont été retenus pour déclencher la capture des modifications, car ils sont natifs à Laravel et ne nécessitent pas d'infrastructure supplémentaire.
- **Fichier SQL plat (adjustments_patch.sql) :** format retenu pour le fichier de correctifs, car il est lisible, portable et directement réutilisable dans le pipeline ETL sans dépendance supplémentaire.

Les alternatives envisagées incluaient l'utilisation d'un job queue Laravel pour l'export asynchrone (écarté pour simplifier la solution initiale) et l'utilisation d'un outil ETL externe comme Apache Airflow (écarté pour sa complexité).

3.4 Gestion des cas particuliers

Plusieurs cas particuliers ont été identifiés lors de la conception et ont nécessité un traitement spécifique :

- **Valeurs NULL** : distinction entre un champ absent de la requête (non modifié) et un champ explicitement mis à NULL. Utilisation de *$request->has()* pour détecter la présence explicite.
- **Champs obligatoires vs nullable** : les champs title, latitude, longitude, source ne peuvent pas être mis à NULL (validation integer/required). Les champs nullable acceptent NULL.
- **source_id** : champ de type int NOT NULL pouvant recevoir des valeurs non entières converties silencieusement en 0 par MariaDB. Validation integer ajoutée.
- **required_args et optional_args (Badge)** : colonnes NOT NULL en base mais optionnelles dans les requêtes. Valeurs par défaut '' et '{}' assignées.
- **Incohérences de nommage** : les noms de champs diffèrent entre store et update pour certains modèles (ex. geo_addresses vs geo_addressesLists pour Heritage).

4. Implémentation / Réalisation technique

4.1 Vue d'ensemble de l'implémentation

L'implémentation s'est déroulée de la semaine 8 à la semaine 13, en suivant une progression incrémentale : d'abord la correction du trait Adjustable et sa validation sur le modèle Artwork, puis l'extension au modèle Artist, puis à Heritage, Place et Badge, et enfin l'ajout du support des opérations INSERT.

La branche de développement a été maintenue séparée de la branche principale tout au long du projet. Des tests via requêtes cURL ont été effectués à chaque étape pour valider le comportement du système.

4.2 Trait Adjustable

Le trait Adjustable a été modifié pour :

- Écouter les événements creating (INSERT), updating (UPDATE) et deleting (DELETE) du cycle de vie Eloquent.
- Supporter les deux guards d'authentification Laravel : web (interface admin) et api (requêtes avec token Bearer). La vérification *Auth::check()* a été remplacée par une vérification sur les deux guards.
- Filtrer les valeurs parasites : la méthode *getDiff()* a été améliorée pour normaliser la comparaison des champs complexes (notamment location) avant enregistrement.
- Éviter les doubles enregistrements : le trait ne crée qu'un seul Adjustment par opération, indépendamment du nombre d'appels à *save()* dans le contrôleur.

4.3 Correction des contrôleurs

Les méthodes *store()* et *update()* des contrôleurs ArtworkController, ArtistController, HeritageController, PlaceController et BadgeController ont été corrigées :

- **store()** : regroupement de toutes les données (champs directs et clés étrangères) dans un seul create(), éliminant le pattern create() + save() qui générait deux événements Eloquent et deux lignes dans le patch.
- **update()** : utilisation de $nullableFields / $requiredFields avec *$request->has()* pour distinguer les champs absents des champs volontairement mis à NULL. Remplacement du pattern *Model::find($id)->update([...])* par une approche plus fine.
- Ajout du modificateur *sometimes* sur les champs obligatoires dans les règles de validation des requêtes UPDATE, pour permettre des mises à jour partielles.

4.4 Service AdjustmentSqlExporter

Le service AdjustmentSqlExporter est responsable de :

- Lire les enregistrements de la table adjustments (filtrés par type d'entité si nécessaire).
- Identifier le type d'opération (INSERT ou UPDATE) à partir des champs before et after.
- Générer la requête SQL correspondante, en ciblant la bonne table et la bonne entité.
- Écrire dans le patch de corrections, en ajoutant des commentaires d'organisation (date, utilisateur).

- La méthode resolveTableName() permet de mapper chaque type d'entité (adjustable_type) vers son nom de table en base de données. L'extension à de nouveaux modèles ne nécessite que l'ajout d'une entrée dans cette méthode.

4.5 Composants principaux et interactions

Les interactions entre les composants lors d'une modification depuis l'interface admin se déroulent comme suit : l'utilisateur modifie une entité dans l'interface admin et soumet le formulaire. Le contrôleur (ex. ArtworkController@update) reçoit la requête, valide les données, puis appelle save() sur le modèle Eloquent. Le trait Adjustable intercepte l'événement updating, compare les valeurs avant et après, et crée un enregistrement dans la table adjustments. Le service AdjustmentSqlExporter est ensuite appelé pour lire cet enregistrement et mettre à jour le fichier adjustments_patch.sql.

Le flux est identique pour les requêtes API, à la différence que l'authentification passe par le guard api et le token Bearer.

5. Évaluation

5.1 Tests unitaires et fonctionnels

Les tests ont été réalisés principalement via des requêtes curl directes sur l'API. La démarche suivante a été adoptée pour chaque modèle testé :

- Création d'un compte utilisateur via l'API, puis attribution du rôle admin via requête SQL directe.
- Exécution de requêtes curl de type POST (update) et (store) avec différentes combinaisons de champs.
- Vérification de la table adjustments via DBeaver après chaque requête.
- Vérification du contenu du patch de corrections.

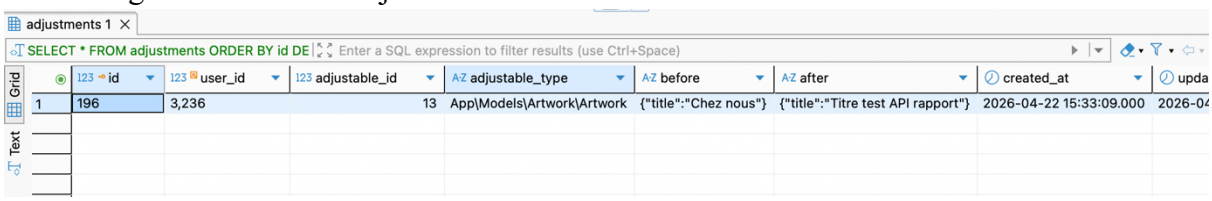
Exemples :

1- **UPDATE pour une œuvre d'art:**

- La requête + réponse :

```
ouldferroukh@Nissas-MacBook mona-server %
curl -X POST http://localhost:8080/api/v4/artworks/13 \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer p6CIZiXOOGcVGqf65ylsuIx2l2E8TSI897prU320j0GpB9IvrjQvpTPKMB4I" \
  -d '{"title": "Titre test API rapport"}'
{"message":"Artwork modified successfully","artwork_id":13}%
ouldferroukh@Nissas-MacBook mona-server %
```

- Ligne dans la table adjustments:



- Ligne dans le patch de corrections:

```
411  -- ====================================
412  -- 2026-04-22
413  -- ====================================
414
415  -- Modifié par : TESTanissa
416  UPDATE `artworks` SET `title` = 'Titre test API rapport' WHERE `id` = 13;
417
```

2- INSERT pour une œuvre d'art:

- La requête + réponse:

```
ouldferroukh@Nissas-MacBook mona-server %
curl -X POST "http://localhost:8080/api/v4/artworks/store" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -H "Authorization: Bearer p6CIZiXOOGcVGqf65ylsuIx2l2E8TSI897prU320j0GpB9IvrjQvpTPKMB4I" \
  -d '{
    "title": "Oeuvre Test rapport",
    "latitude": 45.5017,
    "longitude": -73.5673,
    "produced_at": "2020-01-01",
    "description_fr": "Description en français",
    "description_en": "Description in english",
    "geo_addresses_fr": " Montréal, Qc",
    "geo_territory": 813,
    "geo_borough": 23
  }'
{"message":"Artwork created successfully","artwork_id":2005}%
ouldferroukh@Nissas-MacBook mona-server %
```

- Ligne dans la table adjustments:

adjustments 1

SELECT * FROM adjustments ORDER BY id DE | Enter a SQL expression to filter results (use Ctrl+Space)

	id	user_id	adjustable_id	adjustable_type	before	after	created_at
1	198	3,236	2,005	App\Models\Artwork\Artwork	[]	{"title":"Oeuvre Test rapport","title_en":null,"location":{},"	2026-04-22 15:4

- Ligne dans le patch de corrections:

```
419  -- Modifié par : TESTanissa
420  INSERT INTO `artworks` (`id`, `title`, `title_en`, `location`, `produced_at`, `acquisition_fr`, `acquisition_en`, `mention_fr`, `mention_en`
421      VALUES (2005, 'Oeuvre Test rapport', NULL, '[]', '2020-01-01', NULL, NULL, NULL, NULL, 'Montréal, Qc', NULL, NULL, NULL, NULL, NULL, 'D
422
```

Les cas de test couverts incluent :

- Mise à jour d'un seul champ (ex. titre d'une œuvre).
- Mise à jour de plusieurs champs simultanément.
- Mise à jour avec un champ nullable explicitement mis à NULL.
- Tentative de mise à NULL d'un champ obligatoire (vérification de l'erreur de validation retournée).
- Création d'une entité complète (INSERT) avec vérification d'une seule ligne dans le patch.

- Requêtes avec des valeurs non entières pour source_id (vérification du rejet par validation).
- Modifications depuis l'interface admin et via l'API (vérification de la compatibilité des deux guards).

5.2 Résultats des tests

Les tests ont confirmé que :

- Les adjustments sont correctement générés pour tous les modèles testés (Artwork, Artist, Heritage, Place, Badge).
- Une seule ligne est générée dans le patch par opération (correction du double save() effective).
- Les champs obligatoires retournent une erreur de validation lorsqu'ils sont mis à NULL.
- Les champs nullable acceptent correctement NULL.
- Les modifications via l'interface admin et via l'API produisent des adjustments identiques.

5.3 Évaluation de performance

L'impact sur les performances de l'API est minimal. L'écriture dans la table *adjustments* est une opération d'insertion simple qui s'exécute dans la même transaction que la modification principale, sans surcharge notable. L'écriture dans le patch de corrections est synchrone, c'est à dire qu'elle s'effectue dans le même thread que la requête HTTP, mais reste rapide car elle consiste en une simple opération d'ajout en fin de fichier (append).

Dans le contexte actuel du projet MONA, où les corrections manuelles sont effectuées par un nombre limité d'utilisateurs (équipe de contenu, historiennes de l'art), ce comportement synchrone est tout à fait acceptable. Aucune dégradation notable des temps de réponse n'a été observée lors des tests effectués.

Cependant, dans un contexte à plus grande échelle ou en cas d'utilisation intensive de l'API, deux points de vigilance méritent d'être mentionnés. Premièrement, l'écriture concurrente dans le fichier de patch par plusieurs processus simultanés pourrait provoquer des conditions de course (*race conditions*) et corrompre le fichier. Une solution serait d'utiliser un verrou fichier (flock) ou de passer par une écriture centralisée via un job Laravel Queue. Deuxièmement, si le volume d'ajustements devient très important, la table adjustments pourrait croître rapidement et ralentir les requêtes de lecture. L'ajout d'index sur adjustable_type, adjustable_id et created_at permettrait de maintenir des performances acceptables.

5.4 Comparaison avec les objectifs initiaux

L'objectif principal du projet qui est de capturer automatiquement les corrections manuelles apportées aux données et les consigner dans un fichier de correctifs SQL réapplicable lors des réimportations, a été atteint pour les opérations de type UPDATE et INSERT, sur l'ensemble des modèles concernés (Artwork, Artist, Heritage, Place, Badge).

Plus précisément, les sous-objectifs suivants ont été accomplis :

Interception des modifications: le trait Adjustable, initialement présent mais non fonctionnel, a été implémenté et corrigé pour intercepter fidèlement les événements creating et updating du cycle de vie Eloquent. Les problèmes de double déclenchement, de valeurs parasites et d'incompatibilité des guards d'authentification ont tous été résolus.
Génération du fichier de patch: le service AdjustmentSqlExporter produit un fichier d'ajustements structuré, organisé par date et par utilisateur, contenant des requêtes SQL valides prêtes à être réappliquées après une réimportation. Ce fichier est mis à jour en temps réel à chaque modification.

Compatibilité interface admin et API: le système fonctionne indifféremment depuis l'interface d'administration (guard web) et depuis l'API REST avec token Bearer (guard api), ce qui couvre l'ensemble des points d'entrée du système.

Généralisation: l'architecture conçue (trait générique, relation polymorphique, méthode resolveTableName()) permet d'étendre le système à tout nouveau modèle sans modification structurelle majeure.

En revanche, les problèmes résiduels qui ont été identifiés lors des tests : l'incohérence de nommage entre store et update pour Heritage (geo_addresses vs geo_addressesLists) ainsi que la gestion des opérations de type DELETE n'ont pas été finalisés dans le cadre de ce projet. Ce point fait l'objet de discussions au sein de l'équipe, notamment sur le choix entre un soft delete et un hard delete, chacun ayant des implications différentes sur la forme des requêtes SQL à générer dans le patch. La maison MONA m'ayant proposé un contrat à la suite de ce projet, cette fonctionnalité pourra être finalisée dans ce cadre, mais ne fait pas partie des livrables du présent projet de session.

6. Discussion critique

6.1 Analyse critique de la solution

La solution développée présente plusieurs points forts. En exploitant le système d'ajustements déjà présent dans le projet, elle s'intègre naturellement dans l'architecture existante sans introduire de nouvelles dépendances. La relation polymorphique du modèle Adjustment permet d'étendre le système à n'importe quelle entité sans modification structurelle. Le fichier de patch SQL produit est lisible, portable et directement réutilisable dans le pipeline ETL.

Cependant, la solution présente aussi des limites. La détection des modifications repose sur les événements Eloquent, ce qui implique que toute opération effectuée directement en base de données (sans passer par les modèles Laravel) ne sera pas interceptée. De plus, la solution actuelle écrit dans le fichier de patch de manière synchrone, ce qui pourrait poser des problèmes de concurrence en environnement multi-utilisateurs intensif. Enfin, la gestion des champs complexes (JSON imbriqués, relations multiples) peut générer des requêtes SQL difficiles à appliquer manuellement.

6.2 Problèmes majeurs rencontrés

Plusieurs problèmes techniques significatifs ont été rencontrés et résolus au cours du projet :

- Double déclenchement du trait Adjustable: les contrôleurs effectuaient un *update()* suivi d'un *save()* séparé, ce qui créait deux Adjustments distincts. Résolu en regroupant toutes les modifications dans un seul *save()*.
- Incompatibilité des guards d'authentification: *Auth::check()* ne supportait que le guard web, rendant le système inopérant pour les requêtes API. Résolu en vérifiant les deux guards dans le trait.
- Valeurs parasites dans la table adjustments: owner_id, producer_id, dimensions et location étaient parfois enregistrés comme modifiés alors qu'ils ne l'étaient pas. Résolu par une normalisation de la comparaison dans *getDiff()* et par le nettoyage des contrôleurs.
- Valeurs non entières pour source_id: MariaDB convertissait silencieusement des chaînes en 0. Résolu par l'ajout d'une validation integer explicite.
- Ecrasement des champs non inclus dans une requête POST: le contrôleur remplaçait tous les champs, y compris ceux non fournis, par NULL. Résolu par l'utilisation de *$request->has()* pour ne traiter que les champs présents.

6.3 Compromis réalisés

Le principal compromis concerne la décision de conserver le système d'ajustements existant plutôt que de développer une approche d'extraction directe dans les contrôleurs. Bien que l'extraction directe soit plus simple à implémenter et produise des résultats plus propres, elle ne conserve pas d'historique des modifications. Le choix du système d'ajustements offre une meilleure traçabilité, au prix d'une complexité accrue pour gérer les artefacts du mécanisme.

Un autre compromis concerne la synchronicité de l'export: l'écriture dans le fichier de patch est effectuée de manière synchrone dans le flux de traitement de la requête, ce qui simplifie la logique mais peut introduire une légère latence. Une implémentation asynchrone serait plus robuste en production.

6.4 Retour sur l'expérience de projet

Bien que j'aie déjà occupé un poste de développeuse full stack avant ce projet, avec une expérience de lecture de code existant, de collaboration en équipe et de gestion de contraintes de compatibilité; je n'étais pas pleinement à l'aise avec le développement backend. Cette expérience au sein du projet MONA m'a permis de combler cette lacune et d'acquérir une réelle aisance dans ce domaine.

D'un point de vue technique, j'ai approfondi ma compréhension du framework Laravel, notamment en ce qui concerne le cycle de vie des modèles Eloquent, la gestion des événements (creating, updating, deleting), l'utilisation des traits pour encapsuler des comportements transversaux, et la gestion de l'authentification via plusieurs guards simultanément. Ces concepts, que je connaissais superficiellement, sont devenus des outils que je maîtrise désormais avec confiance.

D'un point de vue méthodologique, travailler sur une API en conditions réelles m'a permis de développer des réflexes de débogage backend : utiliser cURL pour isoler un problème, inspecter directement les tables en base de données avec DBeaver, lire les logs du serveur et raisonner sur le comportement d'un système distribué entre une interface admin, une API et une base de données.

Enfin, cette expérience m'a appris à travailler dans un contexte où les contributions de chaque membre de l'équipe sont étroitement imbriquées. Mon travail était particulièrement lié à celui de ma collègue Corélie : son code constituait la base sur laquelle je devais intégrer ma

contribution, et j'ai dû à plusieurs reprises effectuer des modifications dans ses contrôleurs et ses méthodes pour que mon système fonctionne correctement. Cela a nécessité une communication constante; présenter mes changements lors des rencontres hebdomadaires, valider avec elle que mes modifications n'introduisaient pas d'erreurs, et s'assurer que nos deux contributions restaient cohérentes au fil des semaines. Cette forme de collaboration technique étroite, où l'on touche au code d'une autre personne avec discernement et transparence, est une compétence que je n'avais pas eu l'occasion de développer aussi concrètement dans mes expériences précédentes.

7. Conclusion

Ce projet a permis de développer et d'intégrer un mécanisme de préservation automatique des corrections manuelles dans le pipeline de données du projet MONA. Le système mis en place intercepte les opérations de création et de mise à jour effectuées sur les entités principales de l'application, et génère automatiquement un fichier de correctifs SQL prêt à être réintégré lors des réimportations de données.

Les objectifs initiaux ont été atteints dans leur grande majorité. Les fonctionnalités livrées (génération de requêtes UPDATE et INSERT, compatibilité admin/API, validation des données, extension à tous les modèles concernés) constituent une base solide pour la pérennisation des corrections dans le projet MONA.

Les pistes d'amélioration et de développement futur les plus immédiates concernent la finalisation de la gestion du DELETE, qui sera prise en charge dans le cadre du contrat qui m'a été proposé par la maison MONA, ainsi que la résolution des incohérences de nommage entre les méthodes store et update pour certains modèles.

Au-delà des aspects techniques, ce projet a représenté une expérience d'apprentissage particulièrement enrichissante. Il m'a permis de grandir autant sur le plan technique que sur le plan professionnel, en me confrontant aux réalités d'un projet actif, avec de vraies contraintes, une vraie équipe et de vrais utilisateurs. Je repars de cette expérience avec une plus grande confiance en mes capacités backend, une meilleure compréhension de ce que signifie contribuer à un projet collectif, et l'envie de continuer à évoluer dans cet environnement, ce que le contrat proposé par MONA me donnera l'opportunité de faire.

9. Annexes

Voici quelques exemples de requêtes et de leurs réponses, ainsi qu'un aperçu du fichier de corrections et de la table *adjustments*:

```
ouldferroukh@Nissas-MacBook mona-server % curl -X POST "http://localhost:8080/api/v4/artworks/7" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json" \
    -H "Authorization: Bearer p6CIZiXOOGcVGqf65ylsuIx2l2E8TSI897prU320j0GpB9IvrjQvpTPKMB4I" \
    -d '{"title": null}'
  {"message":"The given data was invalid.","errors":{"title":["The title field is required."]}}%
ouldferroukh@Nissas-MacBook mona-server %
```

```
ouldferroukh@Nissas-MacBook mona-server % curl -X POST http://localhost:8080/api/v4/heritages/1 \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <token>" \
    -d '{"title": null, "latitude": null, "longitude": null, "source": null}'
  {"message":"The given data was invalid.","errors":{"title":["The title field is required."],"latitude":["The latitude field is required."]
  ,"longitude":["The longitude field is required."],"source":["The source field is required."]}}%
ouldferroukh@Nissas-MacBook mona-server %
```

```
ouldferroukh@Nissas-MacBook mona-server % curl -X POST http://localhost:8080/api/v4/badges/store \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <token>" \
    -d '{
      "title_fr": "Badge Test",
      "description_fr": "Description test badge",
      "secret": false,
      "visibility": "public",
      "action": "General_collection",
      "required_args": {"count": 5},
      "optional_args": {"category": "art", "type": "sculpture"}
    }'
  {"message":"Badge created successfully","badge_id":32}%
ouldferroukh@Nissas-MacBook mona-server %
```

```
ouldferroukh@Nissas-MacBook mona-server % curl -X POST http://localhost:8080/api/v4/heritages/store \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <token>" \
    -d '{
      "title": "Heritage Test",
      "latitude": 45.5017,
      "longitude": -73.5673,
      "source": "ImMuni",
      "produced_at": "1900",
      "produced_end": "1950",
      "description": "Description test heritage",
      "url": "https://example.com",
      "synthesis": "Synthèse test",
      "mgmt_status": "Citation",
      "mgmt_id_rpcq": 99999,
      "geo_territory": 817,
      "geo_borough": 23
    }'
  {"message":"Heritage created successfully","heritage_id":1423}%
ouldferroukh@Nissas-MacBook mona-server %
```

```
ouldferroukh@Nissas-MacBook mona-server % curl -X POST http://localhost:8080/api/v4/artists/store \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <token>" \
    -d '{
      "name": "Artiste Test",
      "alias": "artiste-test",
      "birth": "1980-01-01",
      "bio_fr": "Biographie en français",
      "bio_en": "Biography in English",
      "url_fr": "https://example.com/fr",
      "url_en": "https://example.com/en",
      "isCollective": true
    }'
  {"message":"Artist created successfully","artist_id":982}%
ouldferroukh@Nissas-MacBook mona-server %
```

```
393  -- ========================================
394  -- 2026-04-14
395  -- ========================================
396
397  -- Modifié par : TESTanissa
398  UPDATE `artworks` SET `title` = 'Titre test API 1' WHERE `id` = 11;
399  -- Modifié par : TESTanissa
400  UPDATE `artworks` SET `title` = 'L''horizon' WHERE `id` = 11;
401  -- Modifié par : TESTanissa
402  UPDATE `artworks` SET `title` = 'Strates test' WHERE `id` = 1;
403  -- Modifié par : TESTanissa
404  UPDATE `artworks` SET `title` = 'Strates test API' WHERE `id` = 1;
405  -- Modifié par : TESTanissa
406  UPDATE `artworks` SET `title` = 'Strates' WHERE `id` = 1;
407  -- Modifié par : TESTanissa
408  UPDATE `artworks` SET `title` = 'Strates TEST VIA ADMIN' WHERE `id` = 1;
409  -- Modifié par : TESTanissa
410  UPDATE `artworks` SET `title` = 'STRATES VIA TERMINAL' WHERE `id` = 1;
411  -- ========================================
412  -- 2026-04-22
413  -- ========================================
414
415  -- Modifié par : TESTanissa
416  UPDATE `artworks` SET `title` = 'Titre test API rapport' WHERE `id` = 13;
417  -- Modifié par : TESTanissa
418  UPDATE `artworks` SET `title` = 'Chez nous' WHERE `id` = 13;
419  -- Modifié par : TESTanissa
420  INSERT INTO `artworks` (`id`, `title`, `title_en`, `location`, `produced_at`, `acquisition_fr`, `acquisition_en`, `mention_fr`, `mentio
421  -- ========================================
422  -- 2026-04-25
423  -- ========================================
424
425  -- Modifié par : TESTanissa
426  UPDATE `artworks` SET `title` = 'Titre test API sur Lhorizon' WHERE `id` = 11;
427
```