Rapport technique de stage

Refactorisation d'un outil de l'entreprise

Du 20 janvier au 02 Mai 2025

Conrad BADOU

Tuteur université :
MOURAD HAKEM,
mourad.hakem@univ-fcomte.fr

Tuteur entreprise :
Volodymyr Savchenko,
vladimir.savchenko@ten.com

Sommaire

Introduction

Dans le cadre de mon stage de fin d'études au sein de Technip Energies, j'ai eu l'opportunité de travailler sur un projet de refactorisation et d'amélioration d'un outil interne appelé AEDATA.

Mon objectif était de finaliser le travail de migration de l'application existante en PHP pur vers le framework Laravel tout en conservant les fonctionnalités principales et en réajustant le travail fait afin d'obtenir l'expérience utilisateur très proche de celle de l'application de base. Ce rapport technique présente en détail les réalisations effectuées, les problèmes rencontrés ainsi que les solutions mises en œuvre durant ce stage.

1 Contexte du Projet

1.1 Présentation de Technip Energies

Technip Energies est une entreprise internationale spécialisée dans l'ingénierie et la technologie pour le secteur énergétique. Elle met l'accent sur des solutions innovantes et durables pour accompagner la transition énergétique.

1.2 Présentation de l'outil AEDATA

AEDATA est un outil développé par Technip Energies pour extraire, manipuler et exporter les données issues de bases de données complexes via des DBViews (vues de base de données). Il offre une interface intuitive permettant aux ingénieurs d'accéder facilement aux informations techniques et de projet, issues d'outils comme AVEVA Engineering ou d'autres sources. AEDATA facilite l'analyse, le traitement et l'intégration des données dans des processus métiers spécifiques, tout en garantissant leur cohérence et leur traçabilité.

Cet outil joue un rôle stratégique pour Technip Energies, leader mondial de l'ingénierie pour la transition énergétique, en optimisant les flux de travail et en centralisant la gestion des données critiques. AEDATA contribue à réduire les erreurs humaines, à améliorer la collaboration entre les équipes et à accélérer les délais d'exécution, tout en soutenant l'engagement de l'entreprise à fournir des solutions fiables, durables et compétitives.

1.3 Technologies et Outils Utilisés

L'application AEDATA a évolué d'une solution développée en PHP pur vers une architecture basée sur le framework Laravel. Ce choix a permis de bénéficier des avantages d'une structure MVC (Modèle-Vue-Contrôleur), favorisant la modularité, la maintenabilité et l'évolutivité du code.

1.3.1 Structure du projet

/app : Cœur de l'application, contient la majorité du code.
- **Console** : Commandes Artisan personnalisées.
- **Exceptions** : Gestionnaires d'exceptions de l'application.
- **Http** : Contrôleurs, middleware et requêtes HTTP.
- **Models** : Modèles Eloquent pour interagir avec la base de données.
- **Providers** : Fournisseurs de services pour le framework.

/config : Fichiers de configuration de l'application.
/database : Migrations, factories et seeders pour la base de données.
/public : Point d'entrée de l'application (index.php) et assets publics.
/resources : Vues, fichiers LESS/SASS, JavaScript et autres ressources.
- **views** : Templates Blade.
- **lang** : Fichiers de traduction.
- **js/css** : Assets frontend.

/routes : Définitions des routes de l'application.
- **web.php** : Routes web avec session/cookies. (Voir Figure 2)
- **api.php** : Routes API sans état.
- **console.php** : Routes pour console.
- **channels.php** : Routes de diffusion.

/storage : Fichiers générés par l'application (logs, caches, uploads).

/tests : Tests unitaires et d'intégration.

/vendor : Dépendances installées via Composer.

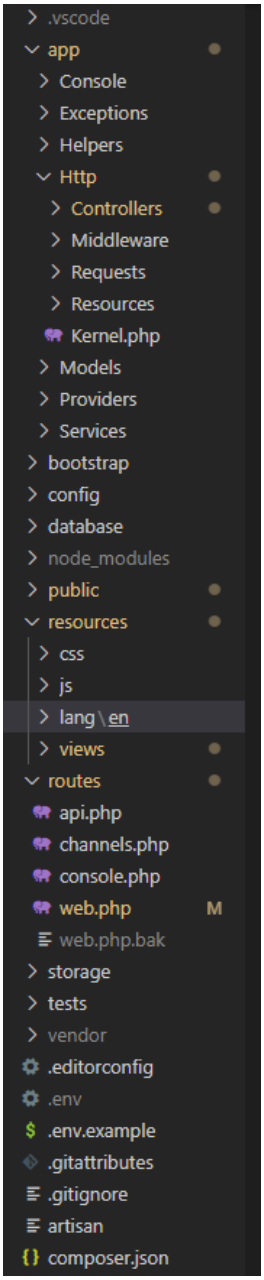


FIGURE 1 – Structure Laravel de l'application

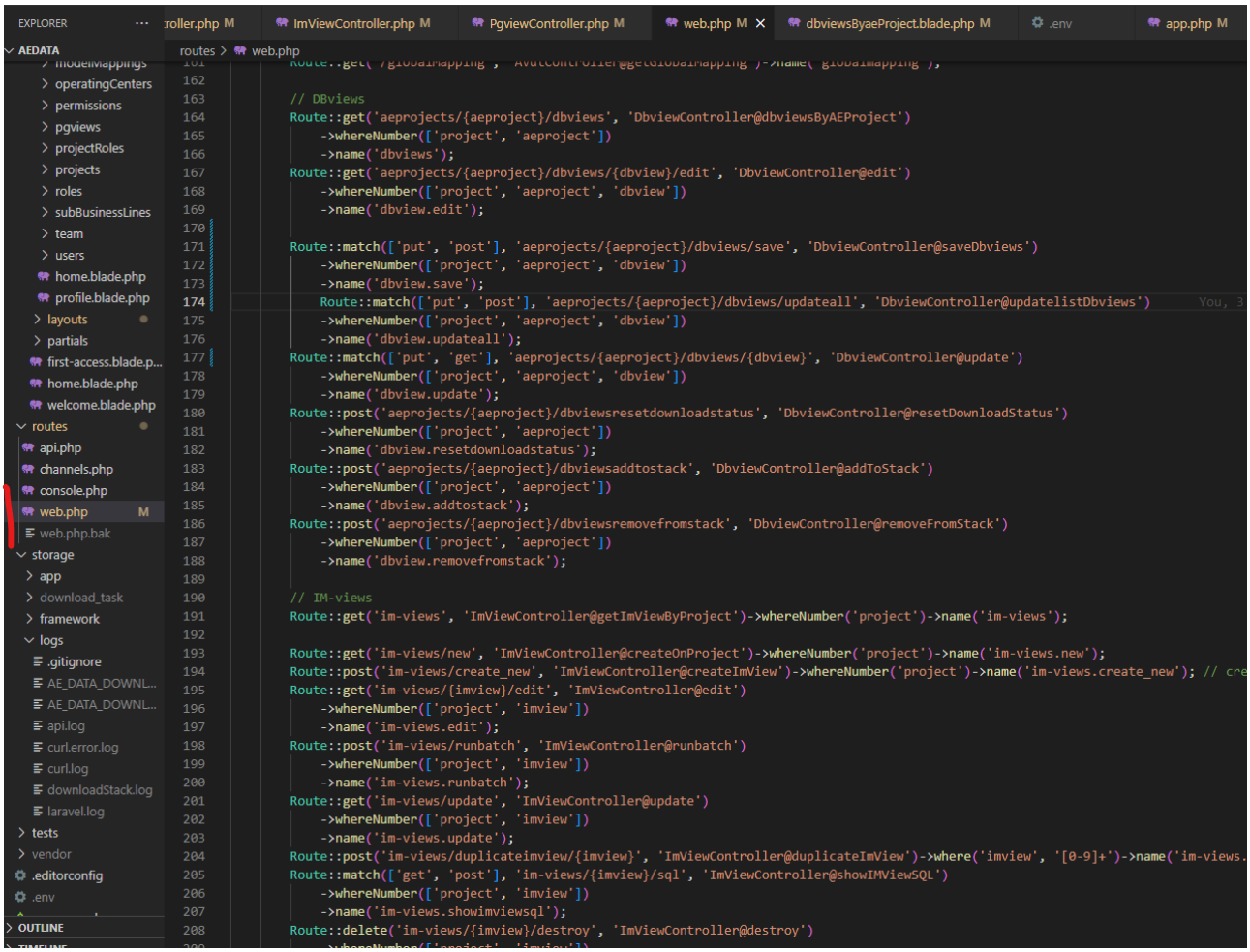


FIGURE 2 – Exemples de quelque routes

1.3.2 Technologies principales

— Technologie initiale : PHP pur, avec une structure de code monolithique et peu organisée.
— Technologie migrée : Laravel, un framework PHP moderne qui offre des fonctionnalités avancées comme les commandes Artisan, les migrations de base de données, et l'ORM Eloquent.

1.3.3 Outils complémentaires

— League.Csv : Une bibliothèque PHP permettant une manipulation aisée des fichiers CSV, notamment pour l'importation et l'exportation de données.
— PhpOffice.PhpSpreadsheet : Un outil puissant pour la génération et la lecture de fichiers XLSX, utilisé pour fournir des exports avancés aux utilisateurs.
— PhpOffice.PhpSpreadsheet : Un outil puissant pour la génération et la lecture de fichiers XLSX, utilisé pour fournir des exports avancés aux utilisateurs.
— DataTables : Une bibliothèque JavaScript interactive pour le rendu des tableaux HTML, offrant des fonctionnalités comme le tri, la recherche et la pagination.
— Bootstrap et notifications Toast : Utilisés pour améliorer l'interface utilisateur, en ajoutant des modales pour les interactions utilisateur et des notifications visuelles pour le retour d'informations.

Ces technologies et outils ont permis de moderniser l'application tout en garantissant une expérience utilisateur fluide et intuitive.

1.4 Objectifs du Stage

Les objectifs principaux de mon stage se sont concentrés sur trois axes majeurs :

1. Améliorer l'interface utilisateur :

   Je devais améliorer l'interface utilisateur de l'application laravel afin qu'elle se rapproche le plus possible de l'application PHP pur.

2. Implémenter et corriger les fonctionnalités essentielles :
   - Ajouter ou corriger les fonctionnalités liées aux DBViews (mise à jour, sélection, réinitialisation, etc.).
   - Intégrer des outils pour la gestion des fichiers CSV/XLSX (génération, téléchargement).
   - Implémenter fonctionnalités pour les ImViews, PgViews, ExViews, EdmsViews, comme la duplication et la création dynamique.

3. Optimiser les performances :
   - Gérer efficacement de gros volumes de données grâce à des optimisations SQL et des traitements par lots.
   - Automatiser les tâches répétitives pour réduire les délais et minimiser les erreurs humaines.

Ces objectifs m'ont permis de structurer mon travail tout au long du stage, en priorisant les besoins des utilisateurs finaux et en assurant la stabilité et la performance de l'application.

2 Méthodologie et Réalisation

2.1 Étapes principales de refactorisation

La refactorisation de l'application AEDATA a suivi une approche méthodique pour garantir une transition fluide et efficace :

1. Analyse du code existant :

   Une étude approfondie du code source de l'application PHP pur et laravel a été réalisée pour identifier les fonctionnalités clés, les dépendances et les zones nécessitant des corrections ou des améliorations.

2. Correction des fonctionnalités existantes et des bugs critiques :

   Avant d'ajouter les nouvelles fonctionnalités, il était essentiel de résoudre les problèmes existants, notamment corriger et mettre à jour les fonctionnalités existantes qui ne marchaient pas bien, il fallait également apporter les corrections nécessaire à l'interface pour atteindre l'objectif.

3. Implémentation de nouvelles fonctionnalités :

   Une fois la base stabilisée, les fonctionnalités manquantes de l'ancienne application ont été intégrées, en suivant une approche modulaire.

4. Tests :

   Des tests ont été réalisés pour valider le bon fonctionnement des nouvelles fonctionnalités et garantir la compatibilité avec les systèmes existants.

2.2 Développement de fonctionnalités principales

2.2.1 Gestion des dbviews

— **fonction Update du bouton Update List :**

Cette fonction est responsable de la mise à jour des DBViews. Elle commence par récupérer tous les DBViews liés à un projet donné (AvdtAeProject) et filtre ceux qui ont une propriété selected non nulle. Si aucun DBView sélectionné n'est trouvé, la fonction retourne une réponse JSON avec une erreur. Pour chaque DBView sélectionné, la fonction appelle une méthode interne (updateDBView) pour effectuer la mise à jour.

La fonction gère les erreurs globales via un bloc try-catch et utilise la méthode updateDBView pour effectuer la mise à jour réelle des DBViews. Elle accepte une requête HTTP et prend également en compte si la mise à jour doit être effectuée à partir d'un fichier CSV (fromcsv). En cas de succès, une réponse JSON de réussite est retournée, sinon une réponse JSON contenant l'erreur est renvoyée. (Voir Figure 3)

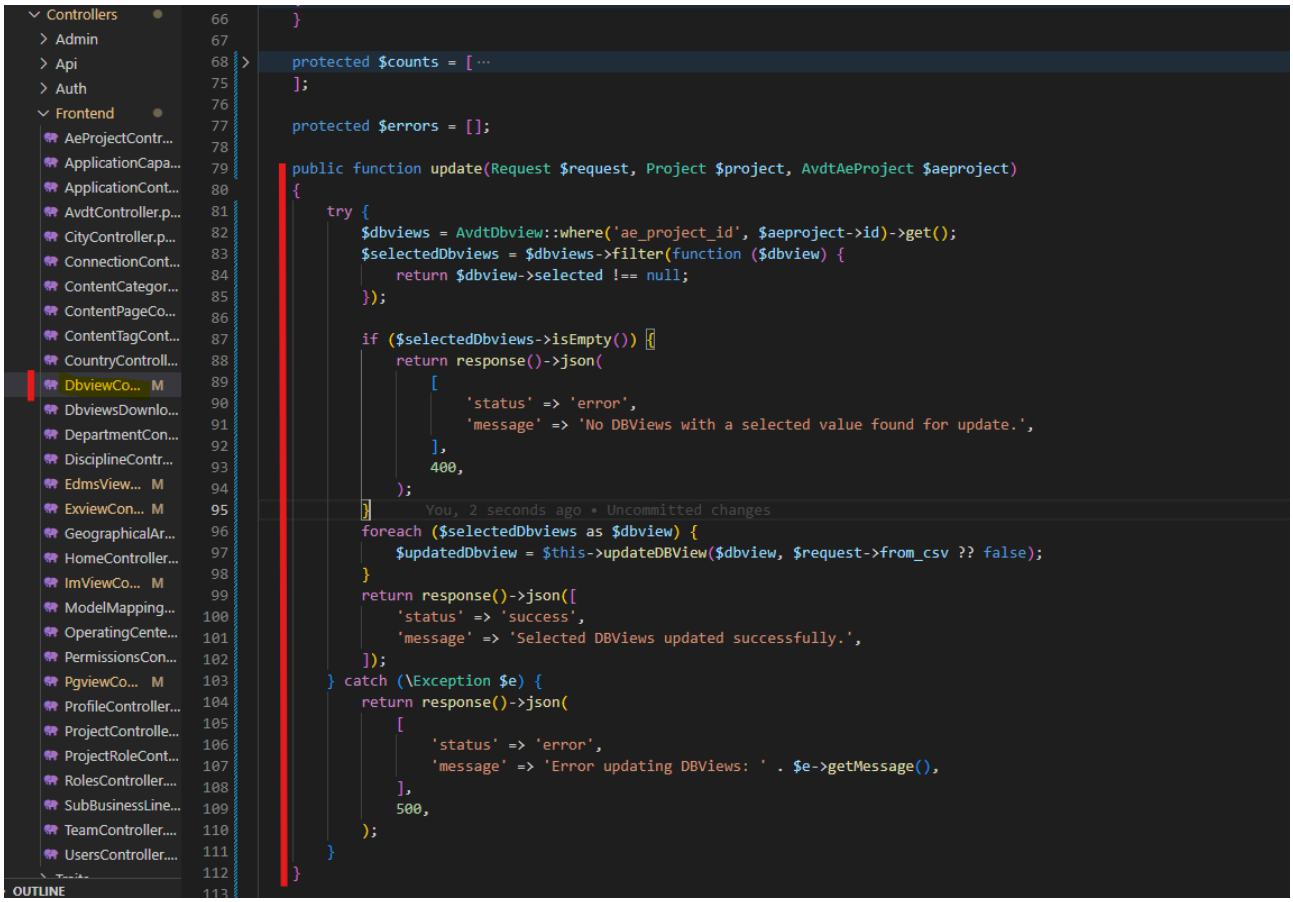


FIGURE 3 – Fonction Update de toute les dbviews

— Fonction updateDBView

Cette fonction est appelée par la méthode update pour effectuer la mise à jour spécifique d'un DBView. Elle commence par enregistrer le temps de début pour mesurer la durée de la mise à jour. Ensuite, elle récupère les colonnes du DBView, soit depuis une source depuis RESTAPI avec la fonction getDBViewColumns (Voir Figure 15 en annexe), soit depuis un fichier CSV avec getDBViewColumns-FromCSV. Si la récupération des colonnes est réussie, elle met à jour les données des colonnes, le statut de modification et les métadonnées associées. Si des colonnes sont déjà présentes, elle compare les colonnes existantes avec les nouvelles pour détecter les modifications. En cas d'échec de la récupération, elle enregistre l'erreur.

La fonction continue en enregistrant les informations dans le modèle DBView et gère les erreurs via un mécanisme de suivi. En cas d'exception, l'erreur est ajoutée à une liste d'erreurs, et l'exception est levée. Cette fonction constitue la logique principale pour la mise à jour d'un DBView, prenant en compte les différences entre les colonnes existantes et les nouvelles pour déterminer si des modifications ont été apportées. Elle utilise un mécanisme de suivi (counts) pour compter les modifications selon leur statut.(Voir Figure 4)

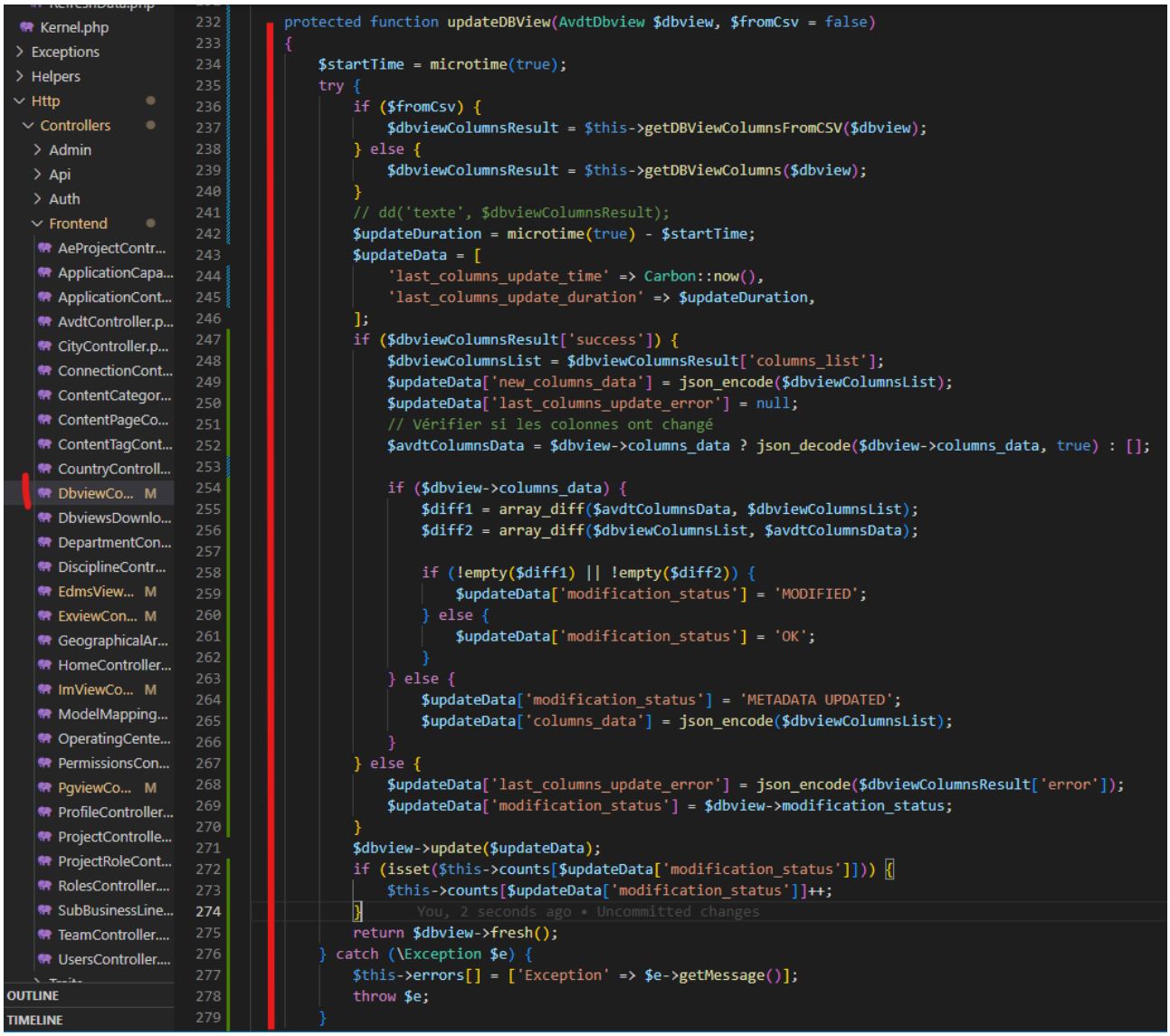


FIGURE 4 – Fonction Update des dbviews

— **fonction Save dbviews du bouton Save Selection :**

Cette fonction permet de sauvegarder les modifications apportées aux DBViews. Elle suit ces étapes principales : d'abord, elle récupère tous les DBViews liés à un projet spécifique (AvdtAeProject). Ensuite, elle parcourt chaque DBView et met à jour les champs data_update et selected en fonction des entrées utilisateur reçues via la requête HTTP. Enfin, elle redirige l'utilisateur vers une route spécifique avec un message de succès.(Voir Figure 5)

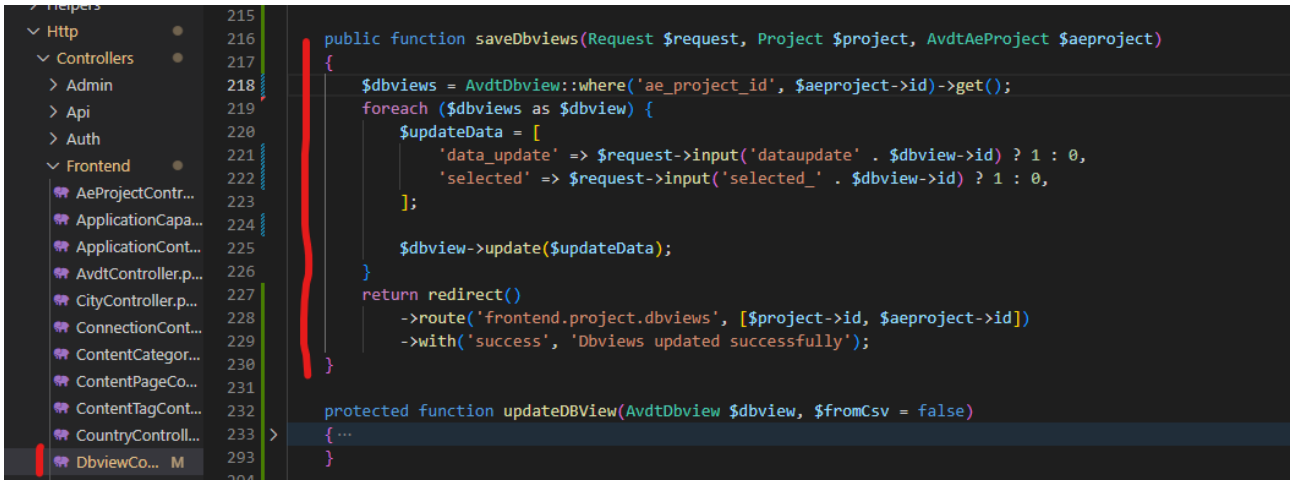


FIGURE 5 – Save Selection

2.2.2 Commandes Laravel

Création des commandes downloadDbviews, processJsonFiles.

— **Commande DownloadDbviews**

Cette classe downloadDBviews est une commande Artisan Laravel qui gère le téléchargement et l'importation de vues de base de données (DBViews) sous forme de fichiers CSV. La méthode handle est le point d'entrée principal, où elle commence par récupérer les paramètres du projet et de la vue (DBView) passés en arguments. Elle lit ensuite un fichier JSON contenant des informations sur l'URL de téléchargement et d'autres métadonnées. Si le dossier de destination pour les fichiers téléchargés n'existe pas, il est créé dynamiquement. Une requête HTTP est alors effectuée pour télécharger le fichier CSV en utilisant des options telles que l'authentification NTLM et un certificat SSL. Si le téléchargement réussit, le fichier est stocké, et si le fichier JSON indique que les données doivent être mises à jour, la méthode importCsvToDatabase est appelée pour importer le contenu du fichier CSV dans une table de base de données. En cas d'échec, des logs d'erreur sont enregistrés. (Voir Figure 6)

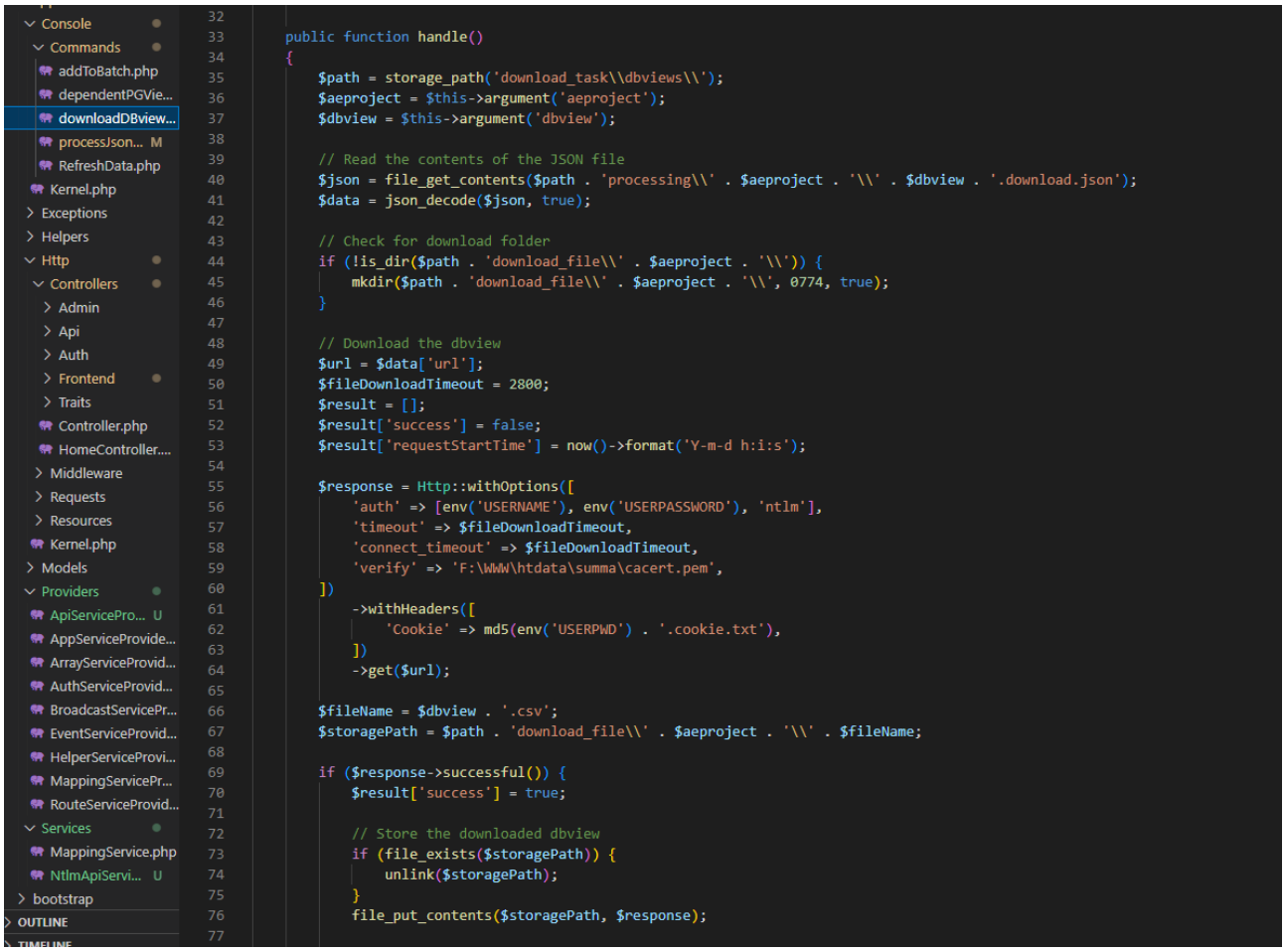


FIGURE 6 – Command CommandDbviewsdowload

La méthode importCsvToDatabase gère l'importation des données CSV dans une base de données. Elle commence par lire le fichier JSON pour extraire des métadonnées, comme les noms du projet, du sous-projet et de la vue, qui sont ensuite utilisés pour construire dynamiquement le nom de la table. Elle nettoie également les noms des colonnes pour s'assurer qu'ils respectent les conventions de nommage des bases de données. Si une table portant le même nom existe déjà, elle est supprimée avant d'en créer une nouvelle avec les colonnes correspondantes. Les données CSV sont ensuite insérées dans la table en utilisant un traitement par lots pour optimiser les performances. Tout le processus est entouré de gestion des erreurs et de journaux pour assurer un suivi clair, avec des logs détaillés sur les étapes importantes comme la création de la table, l'insertion des données, et la gestion des exceptions.(Voir Figure 7)

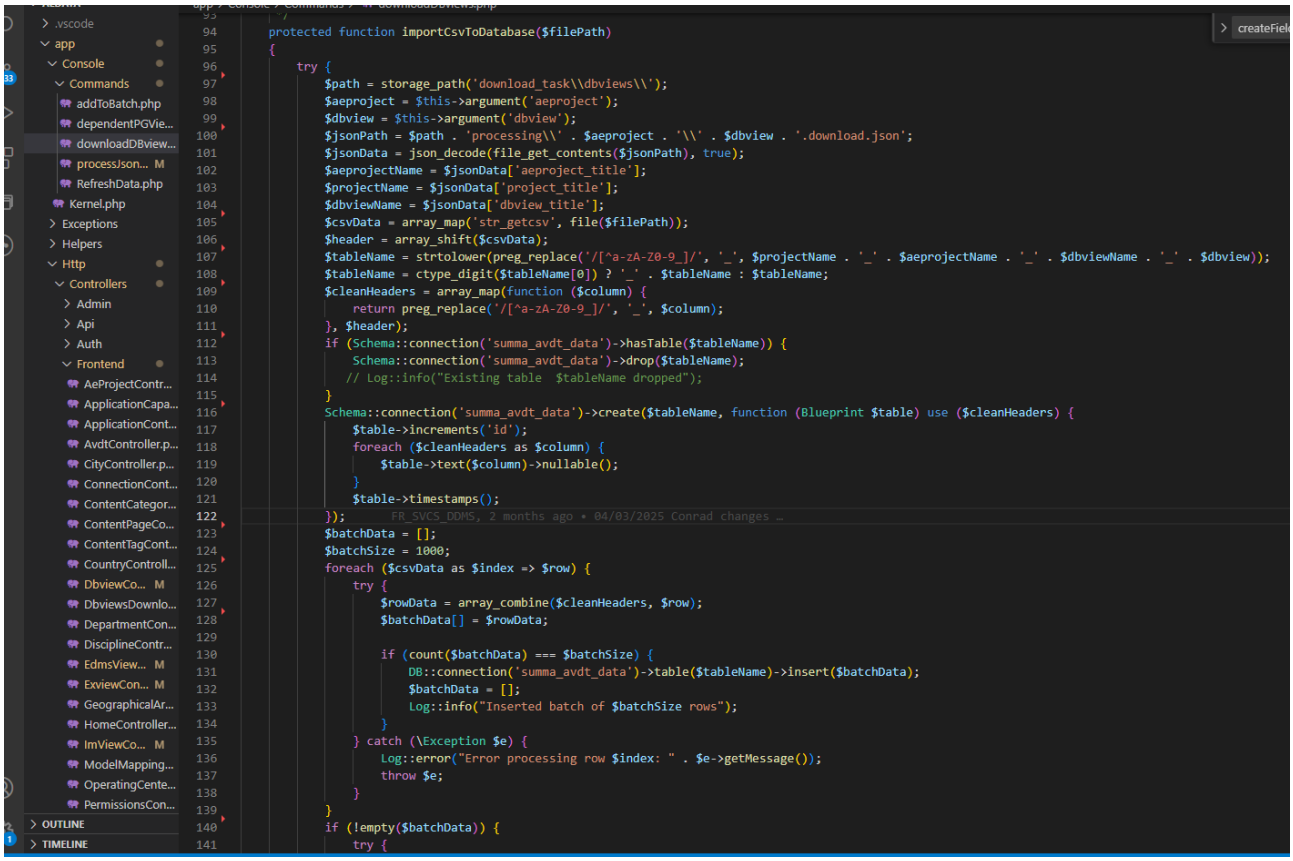


FIGURE 7 – Méthode ImportCsvToDataBase

— **Commande ProcessJsonFiles**

Il s'agit d'une commande artisan Laravel qui gère le téléchargement d'une vue de base de données (DBView) à partir d'une URL spécifiée dans un fichier JSON et, si nécessaire, importe les données dans une base de données. Elle commence par lire le contenu d'un fichier JSON situé dans le dossier "processing", lequel contient des informations sur l'URL du fichier à télécharger et d'autres paramètres. Si le dossier de destination pour les fichiers téléchargés n'existe pas, il est créé dynamiquement. Ensuite, la fonction effectue une requête HTTP pour télécharger le fichier CSV correspondant au DBView, en utilisant des options spécifiques comme l'authentification NTLM, un certificat de validation SSL, et un timeout configurable.

Si la requête HTTP est réussie, le fichier est sauvegardé dans le dossier de téléchargement. Si un fichier existant porte le même nom, il est supprimé avant d'écrire le nouveau fichier(Voir la structure de ces dossier à la Figure 19). Si le paramètre data_update dans le fichier JSON est défini sur true, le fichier CSV est ensuite importé dans la base de données via une méthode appelée importCsvToDatabase. En cas de succès, un log est enregistré pour indiquer que le fichier a été téléchargé et sauvegardé avec succès. En revanche, si la requête échoue, un message d'erreur est enregistré dans les logs, et la fonction retourne un code d'échec. Cette fonction assure donc le téléchargement sécurisé et conditionnel d'un DBView, tout en permettant une intégration des données dans une base de données si nécessaire. (Voir Figure 8)

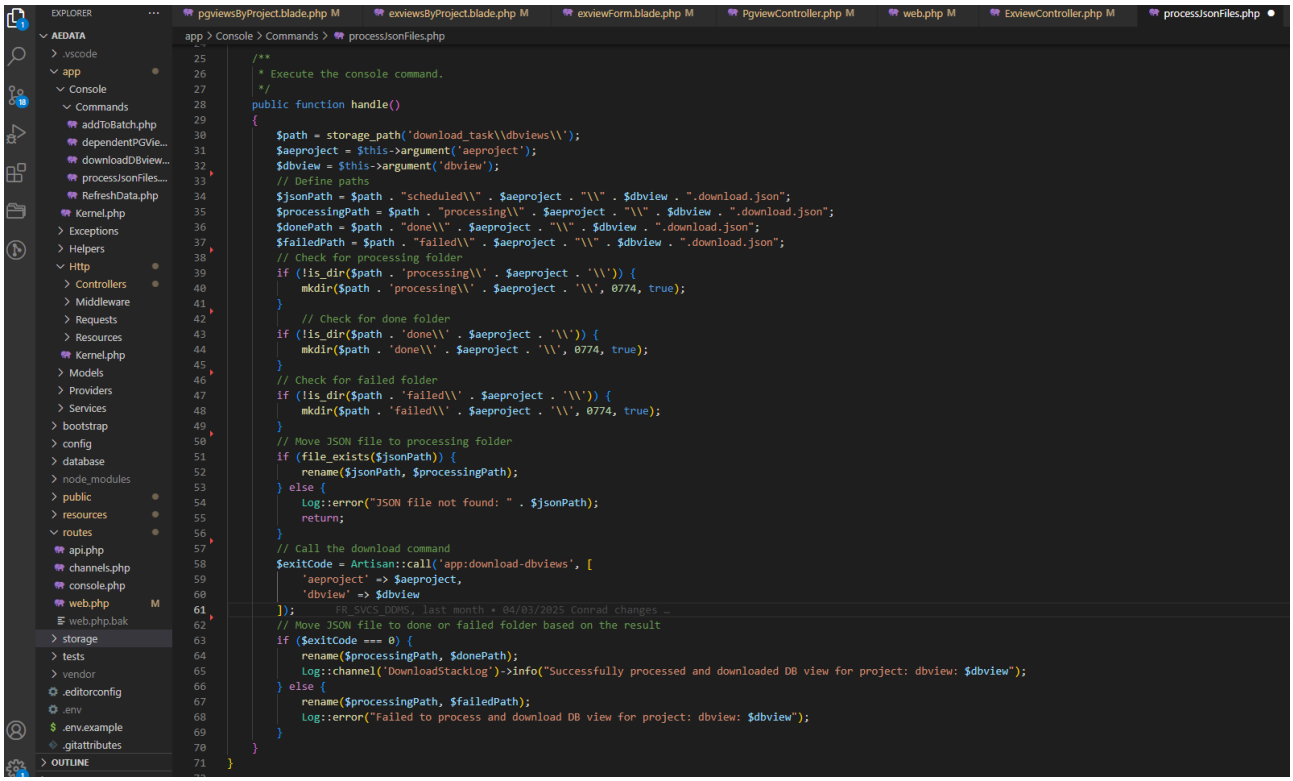


FIGURE 8 – Command ProcessJsonfiles

2.2.3 Gestion des IMViews

— **Duplicate IMviews**

La fonction duplicateImView (Voir Figure 9) permet de dupliquer une vue existante (imview) dans un projet donné. Elle commence par récupérer l'objet imview à partir de son identifiant en utilisant le modèle AvdtImView. Si l'identifiant fourni ne correspond à aucune vue existante, une erreur est automatiquement levée grâce à la méthode findOrFail. Ensuite, la fonction récupère l'identifiant le plus élevé (maxId) parmi toutes les vues existantes, afin de garantir que la vue dupliquée ait un identifiant unique.

La duplication est effectuée en utilisant la méthode replicate, qui crée une copie de l'objet imview sans l'enregistrer immédiatement dans la base de données. Un nouvel identifiant est explicitement assigné à la copie en incrémentant la valeur de maxId. Le titre de la nouvelle vue est modifié pour inclure l'indication " - Copy" afin de distinguer la copie de l'original. Une fois les modifications effectuées, la nouvelle vue est sauvegardée dans la base de données. Enfin, la fonction retourne une réponse JSON indiquant le succès de l'opération, ainsi que les détails de la vue dupliquée. Cette fonction est utile pour permettre aux utilisateurs de cloner rapidement une vue tout en s'assurant de la cohérence des données dans la base.

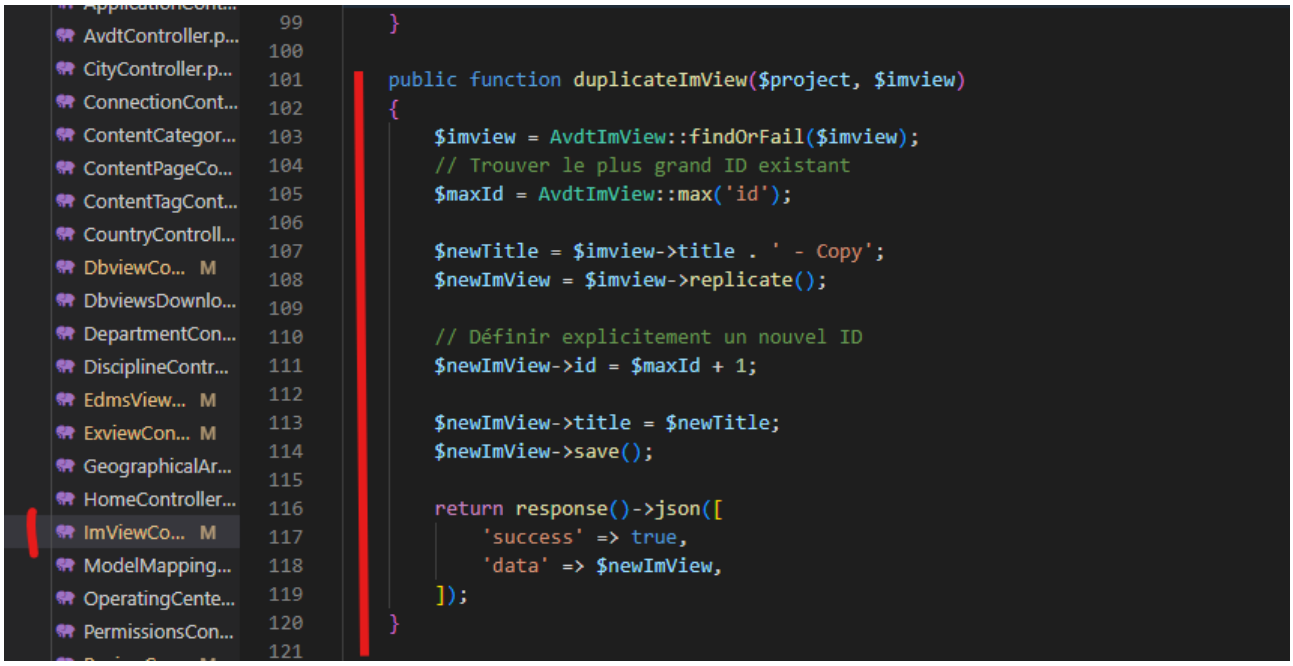


FIGURE 9 – Fonction DuplicateImview

— **Génération de fichiers CSV/XLSX**

La fonction generateFiles (Voir Figure 10) est utilisée pour générer des fichiers CSV et XLSX à partir de données extraites d'une base de données, en fonction des colonnes et des mappages définis dans une vue spécifique (imview). Elle commence par récupérer les informations de la vue depuis la table avdtimviews. Si la vue n'existe pas, une exception est levée. Ensuite, les colonnes SQL sélectionnées et les données de mappage sont décodées à partir de leur format JSON. La fonction parcourt chaque colonne sélectionnée, vérifie si elle est présente dans le mappage, et extrait les données correspondantes de la table SQL concernée. Les noms des colonnes mappées sont ajoutés en tant qu'en-têtes, et les valeurs des colonnes sont collectées sous forme de lignes. Une fois les données collectées, elles sont organisées sous un format structuré avec des en-têtes et des lignes, en utilisant une méthode appelée transposeRows pour ajuster la disposition des données.

Une fois les données préparées, la fonction génère deux fichiers : un fichier CSV et un fichier XLSX, en appelant respectivement les méthodes generateCSV et generateXLSX (Voir Figure 20 en annexe). Ces fichiers sont sauvegardés avec des noms basés sur le titre de la vue (imview). La fonction met également à jour la table avdtimviews pour enregistrer la date de mise à jour et le nombre de lignes dans les données générées. Enfin, elle retourne un tableau contenant des informations sur le succès de l'opération, l'ID et le titre de la vue, les noms des fichiers générés, les liens de téléchargement des fichiers, et le nombre total de lignes dans les données. Cette fonction est donc essentielle pour transformer des données de base de données en fichiers téléchargeables tout en assurant la traçabilité et la mise à jour des métadonnées associées.

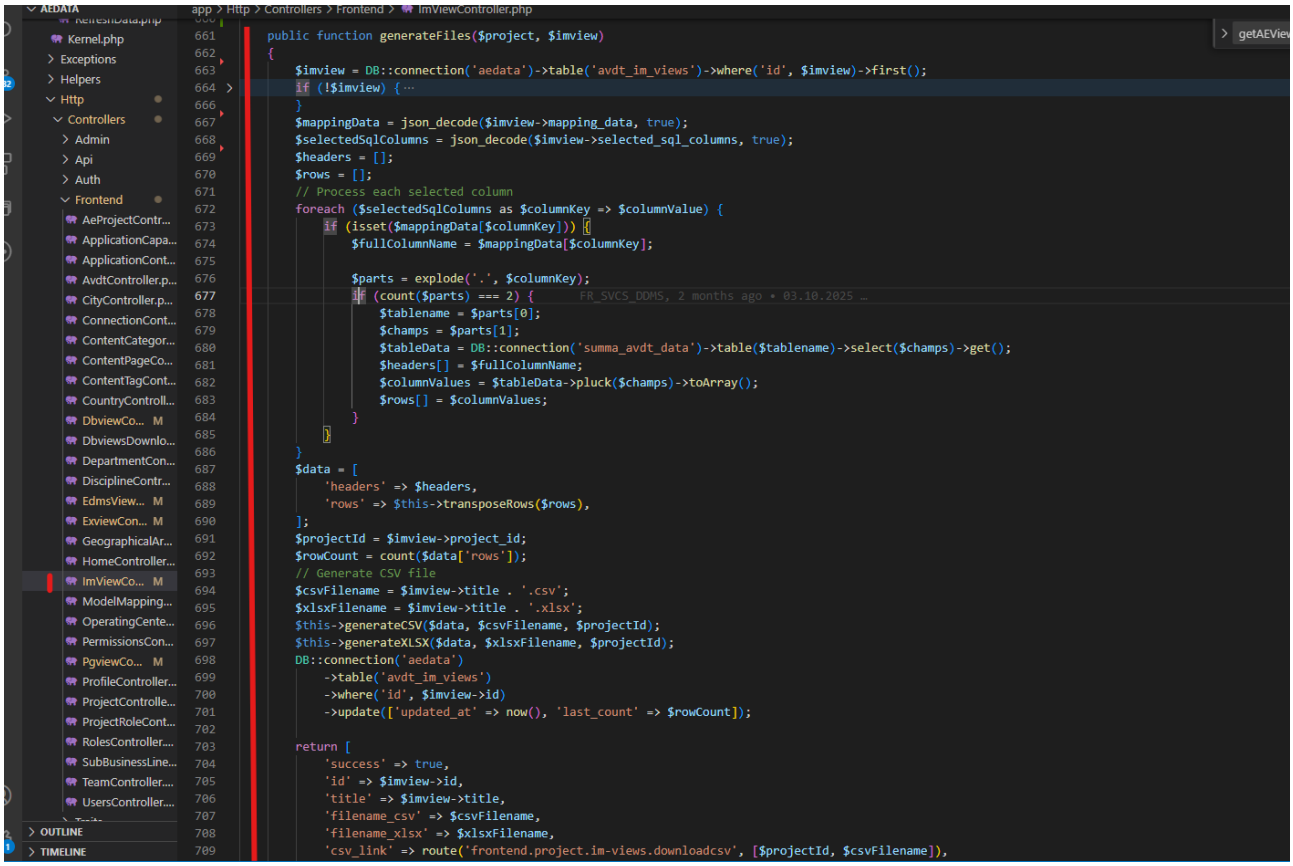


FIGURE 10 – Fonction generatefilesImview

2.2.4 Gestion des PgViews

— **Les fonctions pour les Pgviews**

Le contrôleur **PgviewController** implémente une série de fonctions pour gérer la création et la mise à jour des "PGViews", qui sont des vues personnalisées générées dynamiquement à partir de données stockées dans une ou plusieurs bases de données. La méthode principale, pgviewupdate, est le point d'entrée de ce processus. Elle récupère les données nécessaires pour un PGView spécifique en appelant la fonction getAvdtPGViewData, qui effectue une requête SQL sur une base de données, afin d'obtenir les informations détaillées sur la vue et son projet associé. Ces données sont ensuite transmises à la fonction createPGView, qui génère dynamiquement une vue ou une table temporaire dans une autre base de données en exécutant une série de requêtes SQL.

La fonction createPGView joue un rôle central dans la génération de la vue. Elle commence par formater le nom de la vue à partir des données reçues, puis génère une requête SQL personnalisée à l'aide de la méthode getCreatePGViewSQL. Cette dernière analyse les données des tables associées, applique des transformations comme des remplacements de valeurs ou des exclusions de colonnes, et construit une requête SQL en combinant les clauses 'SELECT', 'FROM', et 'WHERE'. Une fois la requête SQL générée, createPGView utilise la fonction executeSql pour exécuter les requêtes SQL nécessaires, y compris la suppression préalable d'une vue existante et l'application de transformations supplémentaires définies dans les données. Enfin, executeSql gère l'exécution des requêtes SQL tout en capturant les erreurs éventuelles.

Les différentes fonctions du contrôleur sont fortement interconnectées. pgviewupdate agit comme une orchestration, en s'appuyant sur getAvdtPGViewData pour récupérer les données et sur createPG-View pour générer ou mettre à jour la vue. À son tour, createPGView utilise getCreatePGViewSQL pour

construire dynamiquement les requêtes SQL, et executeSql pour exécuter ces requêtes. La fonction getAvdtTablesFields, bien qu'utilisée uniquement dans getCreatePGViewSQL, est essentielle pour récupérer les colonnes des tables nécessaires à la génération de la vue. Ce contrôleur met donc en œuvre un flux de travail structuré et modulaire pour la gestion des PGViews, en utilisant des bases de données multiples et des transformations dynamiques.

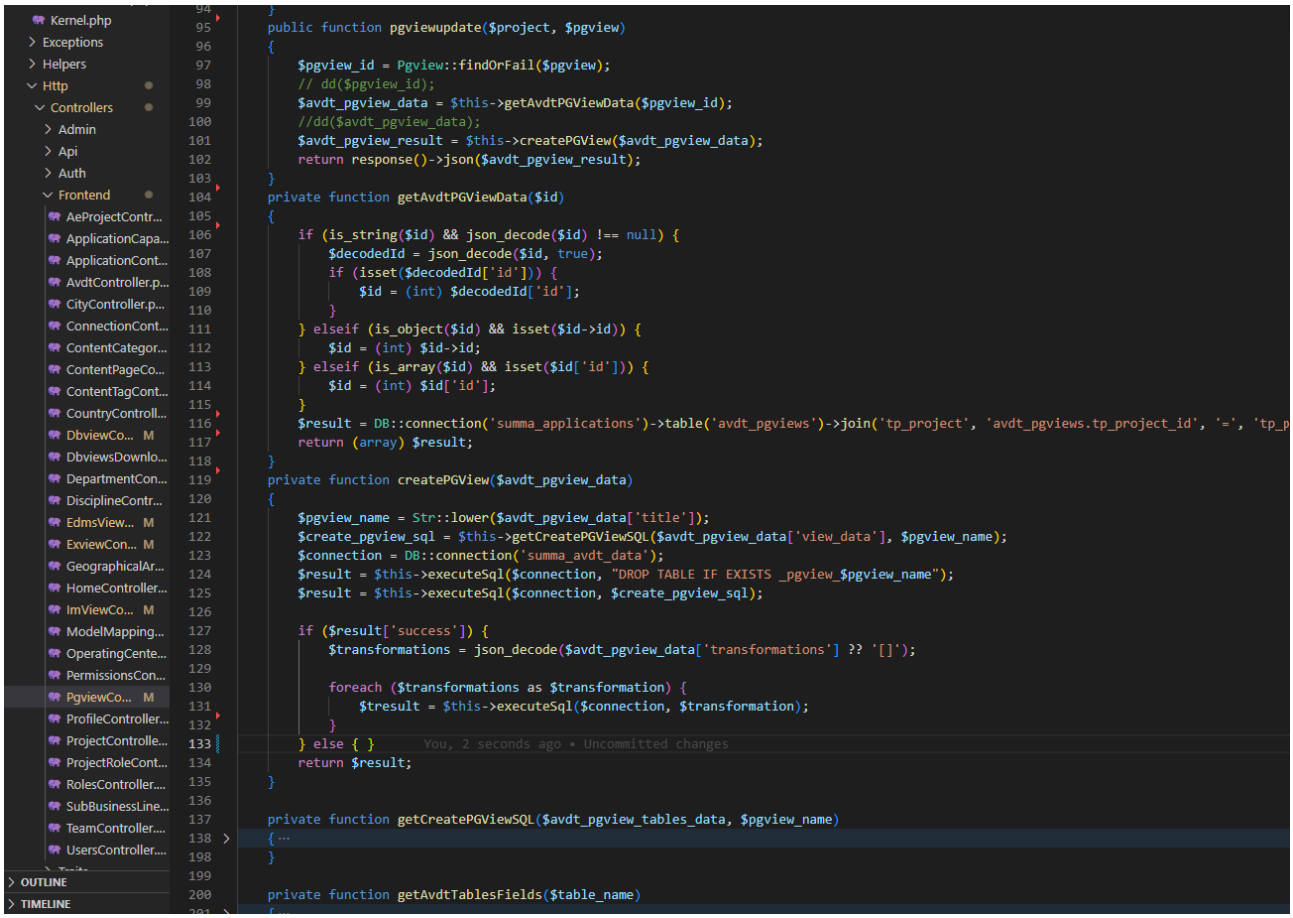


FIGURE 11 – Les fonctions du Pgviewscontroller

— **Commande Pgviews**

La commande Laravel dependentPGViews vérifie l'état d'une DBView spécifique en examinant l'existence d'un fichier JSON correspondant dans trois répertoires différents (done, failed, processing). Selon l'emplacement du fichier, elle met à jour le statut de la DBView dans la base de données et déclenche le traitement des PGViews dépendantes lorsque le téléchargement est réussi. Cette commande est planifiée pour s'exécuter automatiquement via le planificateur de tâches Laravel. La commande est configurée pour s'exécuter toutes les heures pour chaque DBView sélectionnée dans un projet spécifique, en parallèle avec une commande app :move-and-download-dbviews qui gère le téléchargement initial. La configuration est basée sur un fichier JSON aedata-db-views-schedule.json qui définit les projets et leurs horaires de traitement. Les résultats sont consignés dans un fichier de log dédié pour faciliter le suivi des opérations. Cela permet une gestion automatisée des dépendances entre les différentes vues et assurant l'intégrité des données dans l'ensemble du système. Son fonctionnement basé sur l'état des fichiers permetant un suivi précis du processus de traitement et une reprise efficace en cas d'échec.

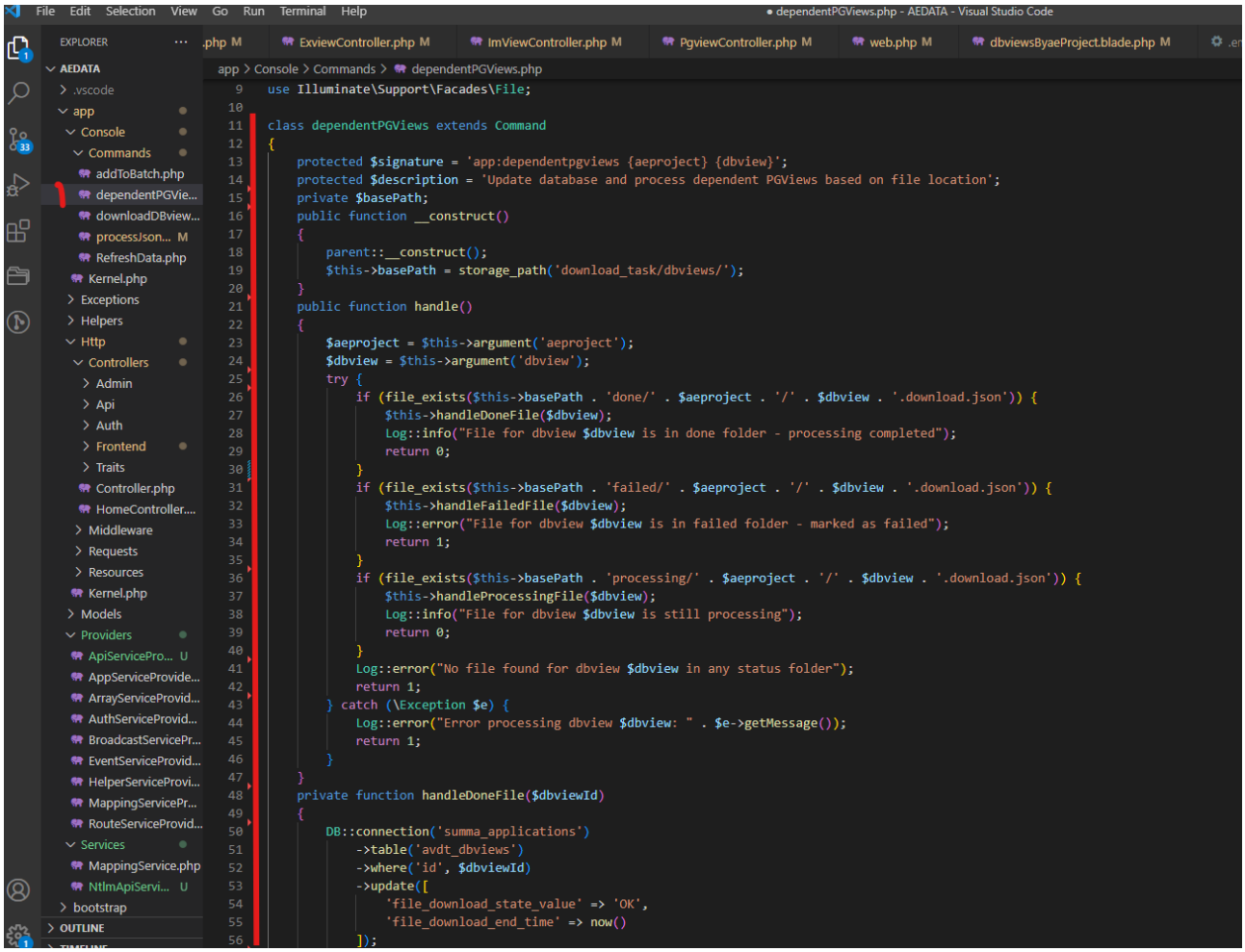


FIGURE 12 – Command Pgviews

2.2.5 Controller des ExViews

Ce controller gère les ExViews (vues externes) qui permettent d'importer des données depuis diverses sources externes vers le système. Voici une description des principales fonctions et leur fonctionnement :

- **getEXViewByProject :** Cette fonction récupère et affiche toutes les ExViews associées à un projet spécifique. Elle collecte les ExViews depuis la base de données, récupère les données des projets AE correspondants, et renvoie une vue frontend avec ces informations.
- **updateExview :** Cette fonction met à jour une ExView spécifique. Elle récupère d'abord les données de l'ExView depuis la base de données, puis lance le processus de création/mise à jour via la méthode createExView. En cas de succès, elle renvoie le résultat avec le titre de l'ExView ; en cas d'échec, elle renvoie un message d'erreur approprié.
- **getAvdtExViewData (privée) :** Cette méthode auxiliaire récupère les données complètes d'une ExView depuis la base de données. Elle gère intelligemment différents formats d'ID (chaîne JSON, objet, tableau) et effectue une jointure avec la table des projets pour obtenir des informations supplémentaires. (Voir Figure 25)
- **createExView (privée) :** Cette fonction principale effectue l'importation des données externes. Son fonctionnement comprend :
  - Gestion des fonctions personnalisées si l'ExView est de type "custom"
  - Connexion à différentes sources de données (SQL Server, MySQL, ODBC, Oracle) selon le type spécifié
  - Création d'une table temporaire dans la base pour stocker les données importées

— Importation des données ligne par ligne avec gestion de l'encodage UTF-8
— Mise à jour du nombre de lignes importées dans l'enregistrement de l'ExView

Le système supporte plusieurs types de connexions (sqlsrv, mysql, odbc, oracle) et gère les transactions de base de données pour assurer l'intégrité des données pendant l'importation. En cas d'erreur à n'importe quelle étape, la transaction est annulée et un message d'erreur détaillé est renvoyé

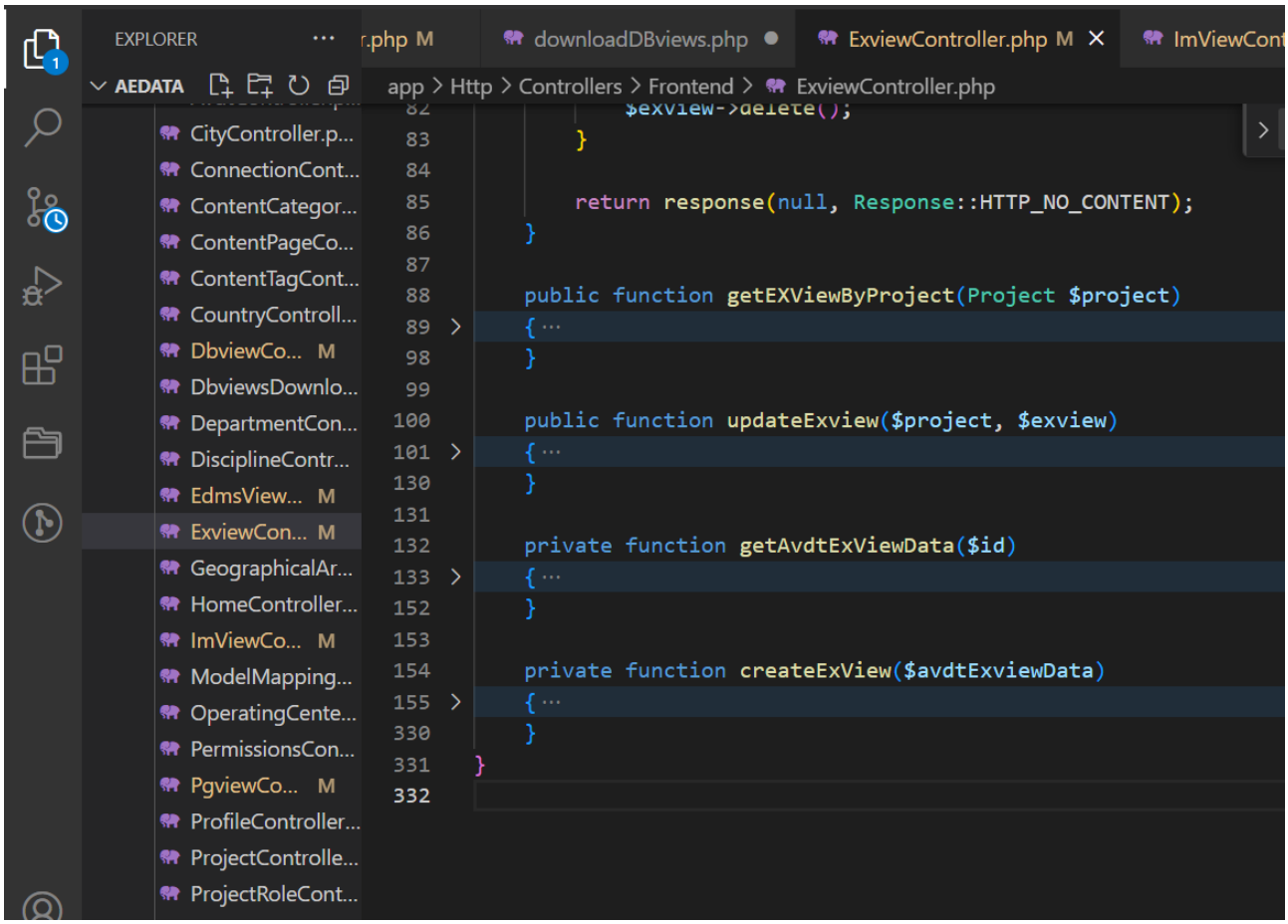


FIGURE 13 – ExviewController

2.2.6 Optimisation et mise en forme

Amélioration des DataTables avec ajout d'options "Select All", tri, recherche, et affichage dynamique. L'affichage dynamique a été optimisé pour que les tables puissent s'adapter automatiquement à la taille de la page, garantissant une meilleure lisibilité et une navigation plus fluide.

La table a été corrigée pour qu'elle puisse s'afficher entièrement sur la page, évitant ainsi les problèmes de défilement horizontal et améliorant l'expérience utilisateur. Les modifications ont également été apportées au code CSS pour s'assurer que l'apparence et le comportement des DataTables se rapprochent le plus possible de l'ancienne version en PHP, tout en intégrant les nouvelles fonctionnalités. Ces ajustements CSS incluent la mise en forme des icônes, l'optimisation de la disposition des éléments, et l'amélioration de la réactivité de l'interface. (Les deux versions ont été présenter, la nouvelle version laravel et la version php, voir Figure 21 vs Figure 23 en annexe)

3 Défis Techniques Rencontrés

Défi	**Solution Apportée**
Compréhension du projet sans documentation	Analyse manuelle du code, réunions fréquentes avec Adrien FERNANDEZ et M. Volodymyr
Migration PHP vers Laravel	Utilisation de la documentation Laravel et adaptation aux structures MVC
Gestion des fichiers volumineux	Implémentation de traitement par lots et optimisation SQL
Bugs liés aux fichiers CSV en production	Correction du formatage des caractères spéciaux
Problèmes d'UI/UX	Ajout de modales, notifications toast, optimisations d'affichage

TABLE 1 – Défis Techniques Rencontrés

4 Résultats Obtenus

— **Automatisation du traitement CSV/XLSX : commandes artisan programmées.**

Ce point fait référence à la mise en place de commandes Artisan (outils de ligne de commande dans Laravel) permettant d'automatiser la génération de fichiers CSV et XLSX à partir de données. Ces commandes sont planifiées pour être exécutées à des moments spécifiques, comme via un cron job, afin de réduire les interventions manuelles. Les avantages incluent un gain de temps, car les fichiers sont générés automatiquement sans nécessiter d'actions utilisateur, une fiabilité accrue grâce à l'exécution régulière des tâches planifiées, et une gestion simplifiée avec des données exportées toujours à jour, cruciales pour des rapports ou des analyses.

— **Correction de bugs (importation base de données, erreurs lors du reset des statuts, etc.)**

Cette section traite de la correction de bugs liés à l'importation de données dans la base de données et aux différentes erreurs. Les erreurs d'importation peuvent inclure des problèmes de formats de données incorrects, des conflits de clés primaires ou des erreurs d'encodage. Les bugs liés au reset des statuts, qui réinitialisent les statuts des enregistrements (comme actif/inactif, validé/en attente), ont été corrigés pour garantir un comportement attendu, en résolvant des erreurs de logique ou des requêtes SQL incorrectes.

— **Fonctionnalités implémentées :** Update dbviews, génération de fichiers multiples, duplication d'IM-Views.

— **Expérience utilisateur améliorée :** Ajout de modales de chargement, notifications de succès ou d'échec, modifications de la vue et ajout des icônes.

5 Perspectives d'Évolution

5.1 Améliorer l'intégration d'un éditeur de code (ACE ou alternatif)

Améliorer l'intégration d'un éditeur de code (ACE ou alternatif) pour l'édition de requêtes SQL L'intégration d'un éditeur de code comme ACE ou un autre éditeur alternatif permettrait aux utilisateurs de rédiger et de modifier des requêtes SQL directement dans l'application. Cela offrirait une interface plus intuitive et interactive pour les développeurs, facilitant la rédaction, la vérification et l'exécution des requêtes SQL. Un éditeur de code intégré pourrait également fournir des fonctionnalités avancées telles que la coloration syntaxique, l'auto-complétion, et la gestion des erreurs, améliorant ainsi la productivité et la précision des utilisateurs.

5.2 Refactorisation continue

Une refactorisation continue pour optimiser la base de code La refactorisation continue est essentielle pour maintenir une base de code propre, efficace et facile à maintenir. Cela implique de revoir régulièrement le code existant pour identifier et éliminer les redondances, simplifier les structures complexes, et améliorer la lisibilité. En optimisant la base de code, on peut réduire les bugs, améliorer les performances, et faciliter l'ajout de nouvelles fonctionnalités. La refactorisation permet également de suivre les meilleures pratiques de développement et de garantir que le code reste aligné avec les standards industriels.

5.3 Ajout de tests automatisés

Ajout de tests automatisés pour renforcer la fiabilité future du projet L'ajout de tests automatisés est crucial pour assurer la fiabilité et la stabilité du projet à long terme. Les tests automatisés permettent de vérifier que chaque fonctionnalité fonctionne comme prévu et de détecter rapidement les régressions lors des modifications du code. En couvrant les différentes parties du projet avec des tests unitaires, des tests d'intégration et des tests fonctionnels, on peut garantir que les nouvelles mises à jour n'introduisent pas de bugs et que le système reste robuste face aux changements.

5.4 Modernisation de l'interface utilisateur

Modernisation de l'interface utilisateur avec un framework front-end comme Vue.js ou React La modernisation de l'interface utilisateur avec un framework front-end comme Vue.js ou React permettrait de créer des interfaces plus dynamiques, réactives et conviviales. Ces frameworks offrent des outils puissants pour développer des composants réutilisables, gérer l'état de l'application, et améliorer l'interactivité. En adoptant Vue.js ou React, on peut également bénéficier d'une meilleure performance et d'une expérience utilisateur plus fluide, tout en facilitant le développement et la maintenance de l'interface.

Conclusion

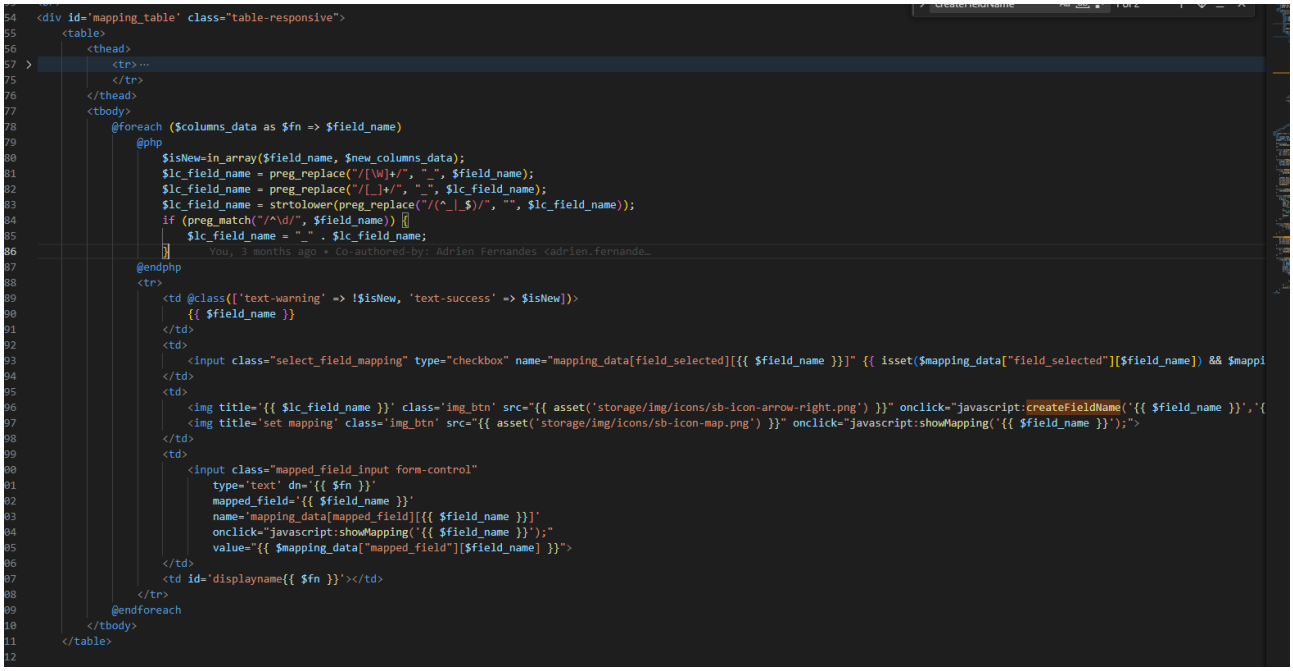
Ce projet m'a permis de développer mes compétences en Laravel, notamment en backend, Artisan, et ORM Eloquent. J'ai appris à utiliser efficacement ces outils pour créer des fonctionnalités robustes et performantes. La gestion de fichiers volumineux en base de données a été un aspect crucial du projet, me permettant de maîtriser les techniques nécessaires pour manipuler et optimiser de grandes quantités de données. J'ai également approfondi mes connaissances en optimisation d'interface utilisateur, en intégrant des éléments visuels et interactifs qui améliorent l'expérience utilisateur.

Travailler sur ce projet réel et complexe m'a offert l'opportunité de respecter les standards industriels, garantissant ainsi la qualité et la fiabilité des solutions développées. Je suis fier d'avoir contribué de manière significative à l'amélioration d'un outil stratégique pour Technip Energies, en apportant des innovations et des améliorations qui ont un impact direct sur l'efficacité et la productivité de l'entreprise. Cette expérience a été enrichissante et formatrice, consolidant mes compétences techniques et ma capacité à gérer des projets de grande envergure.

Table des figures

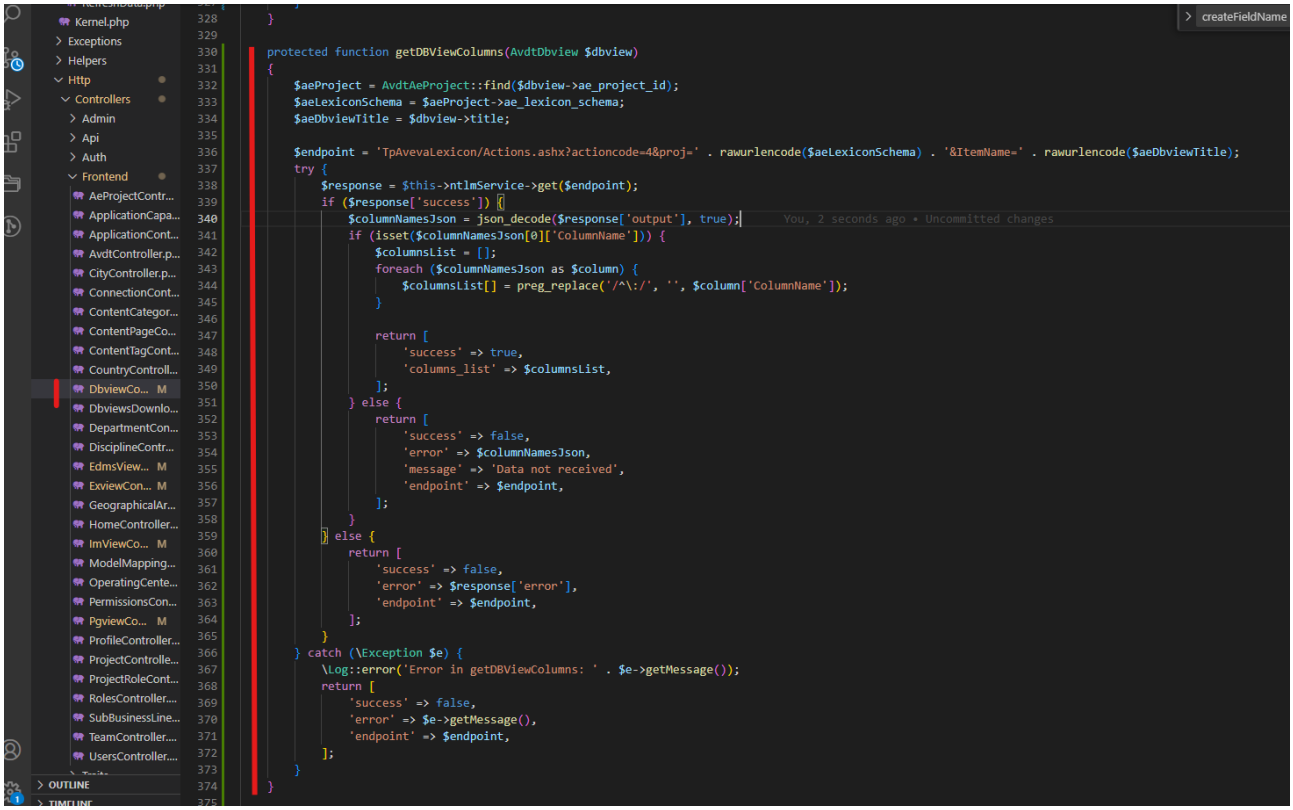
Liste des tableaux

Annexes



```
<div id='mapping_table' class="table-responsive">
    <table>
        <thead>
            <tr> ···
            </tr>
        </thead>
        <tbody>
            @foreach ($columns_data as $fn => $field_name)
                @php
                    $isNew=in_array($field_name, $new_columns_data);
                    $lc_field_name = preg_replace("/[\W]+/", "_", $field_name);
                    $lc_field_name = preg_replace("/[_]+/", "_", $lc_field_name);
                    $lc_field_name = strtolower(preg_replace("/(^_|_$)/", "", $lc_field_name));
                    if (preg_match("/^\d/", $field_name)) {
                        $lc_field_name = "_" . $lc_field_name;
                    }      You, 3 months ago • Co-authored-by: Adrien Fernandes <adrien.fernande…
                @endphp
                <tr>
                    <td @class(['text-warning' => !$isNew, 'text-success' => $isNew])>
                        {{ $field_name }}
                    </td>
                    <td>
                        <input class="select_field_mapping" type="checkbox" name="mapping_data[field_selected][{{ $field_name }}]" {{ isset($mapping_data["field_selected"][$field_name]) && $mappi
                    </td>
                    <td>
                        <img title='{{ $lc_field_name }}' class='img_btn' src="{{ asset('storage/img/icons/sb-icon-arrow-right.png') }}" onclick="javascript:createFieldName('{{ $field_name }}','{
                        <img title='set mapping' class='img_btn' src="{{ asset('storage/img/icons/sb-icon-map.png') }}" onclick="javascript:showMapping('{{ $field_name }}');">
                    </td>
                    <td>
                        <input class="mapped_field_input form-control"
                            type='text' dn='{{ $fn }}'
                            mapped_field='{{ $field_name }}'
                            name='mapping_data[mapped_field][{{ $field_name }}]'
                            onclick="javascript:showMapping('{{ $field_name }}');"
                            value="{{ $mapping_data["mapped_field"][$field_name] }}">
                    </td>
                    <td id='displayname{{ $fn }}'></td>
                </tr>
            @endforeach
        </tbody>
    </table>
```

FIGURE 14 – Correction du formatage des noms de champs



```
    }

    protected function getDBViewColumns(AvdtDbview $dbview)
    {
        $aeProject = AvdtAeProject::find($dbview->ae_project_id);
        $aeLexiconSchema = $aeProject->ae_lexicon_schema;
        $aeDbviewTitle = $dbview->title;

        $endpoint = 'TpAvevaLexicon/Actions.ashx?actioncode=4&proj=' . rawurlencode($aeLexiconSchema) . '&ItemName=' . rawurlencode($aeDbviewTitle);
        try {
            $response = $this->ntlmService->get($endpoint);
            if ($response['success']) {
                $columnNamesJson = json_decode($response['output'], true);      You, 2 seconds ago • Uncommitted changes
                if (isset($columnNamesJson[0]['ColumnName'])) {
                    $columnsList = [];
                    foreach ($columnNamesJson as $column) {
                        $columnsList[] = preg_replace('/^\:/', '', $column['ColumnName']);
                    }

                    return [
                        'success' => true,
                        'columns_list' => $columnsList,
                    ];
                } else {
                    return [
                        'success' => false,
                        'error' => $columnNamesJson,
                        'message' => 'Data not received',
                        'endpoint' => $endpoint,
                    ];
                }
            } else {
                return [
                    'success' => false,
                    'error' => $response['error'],
                    'endpoint' => $endpoint,
                ];
            }
        } catch (\Exception $e) {
            \Log::error('Error in getDBViewColumns: ' . $e->getMessage());
            return [
                'success' => false,
                'error' => $e->getMessage(),
                'endpoint' => $endpoint,
            ];
        }
    }
```

FIGURE 15 – Fonction getDBViewColumns

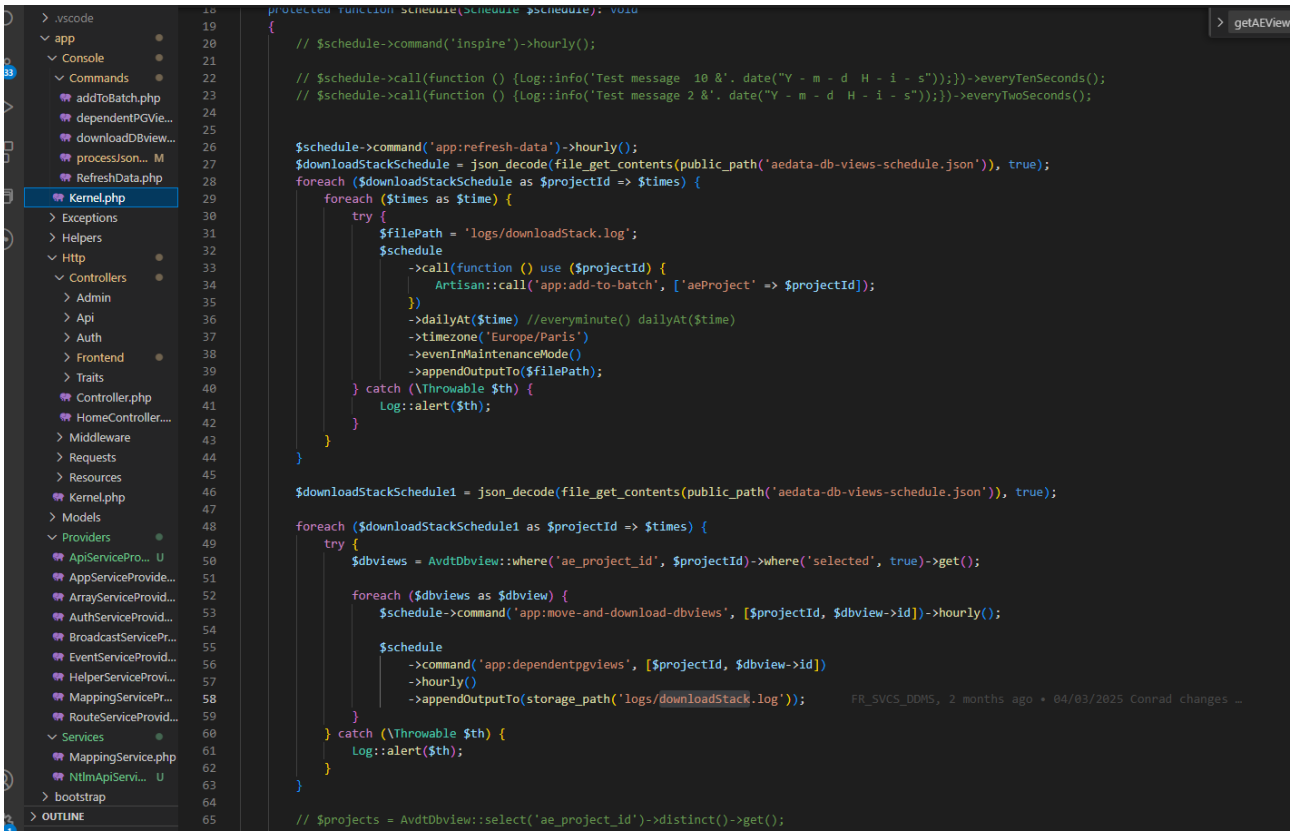


FIGURE 16 – Fonction Schedule

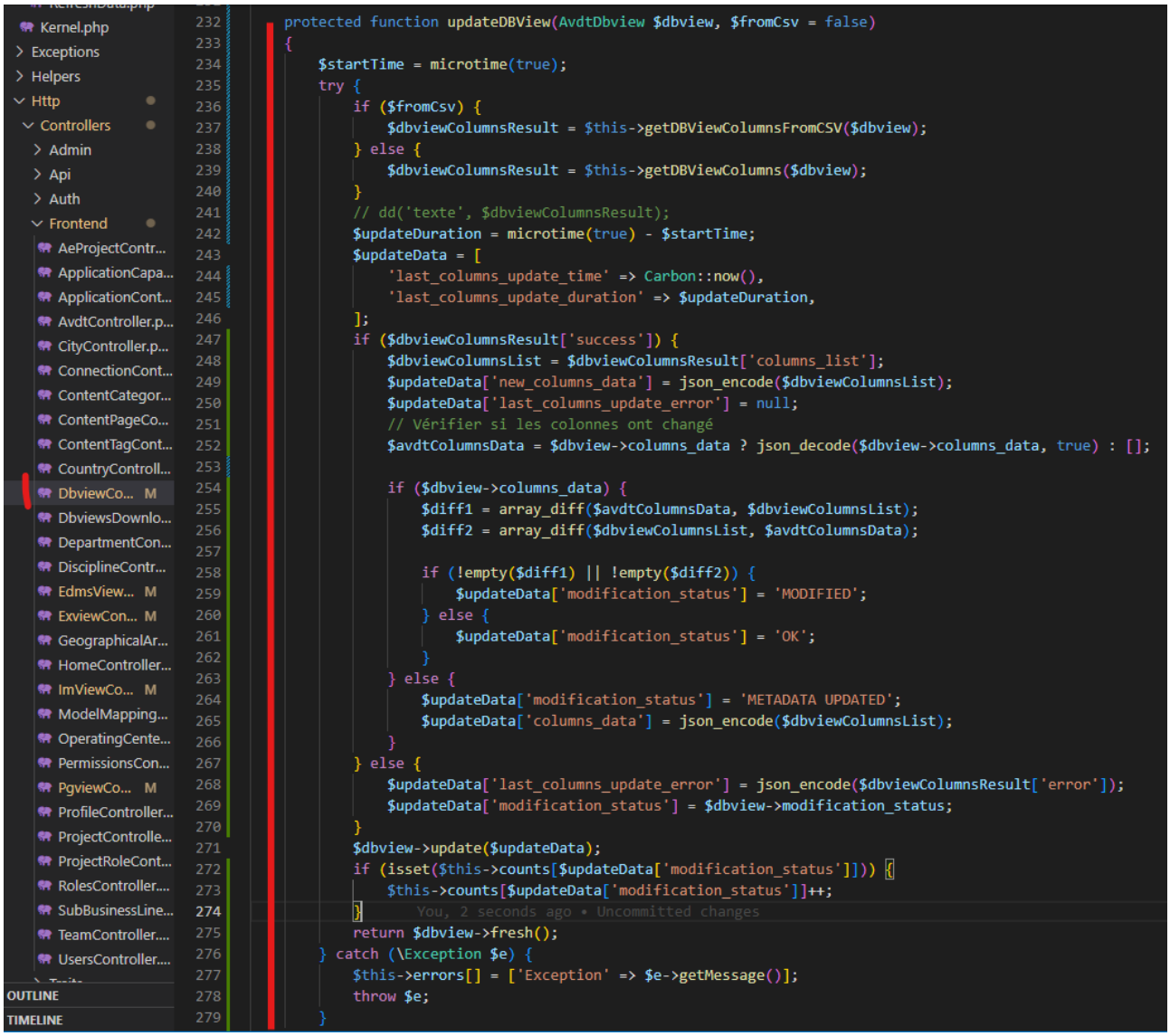
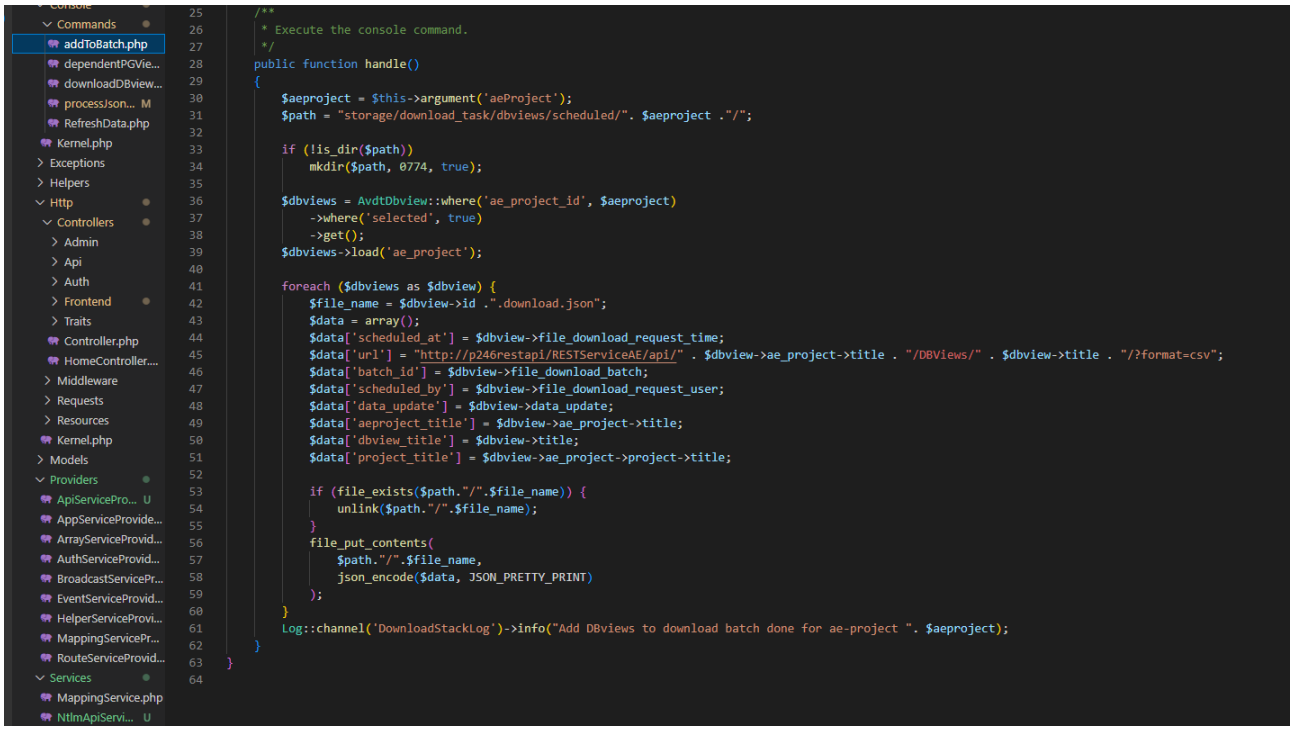


FIGURE 17 – Fonction Update appler dans Update List




```php
    /**
     * Execute the console command.
     */
    public function handle()
    {
        $aeproject = $this->argument('aeProject');
        $path = "storage/download_task/dbviews/scheduled/". $aeproject ."/";

        if (!is_dir($path))
            mkdir($path, 0774, true);

        $dbviews = AvdtDbview::where('ae_project_id', $aeproject)
            ->where('selected', true)
            ->get();
        $dbviews->load('ae_project');

        foreach ($dbviews as $dbview) {
            $file_name = $dbview->id .".download.json";
            $data = array();
            $data['scheduled_at'] = $dbview->file_download_request_time;
            $data['url'] = "http://p246restapi/RESTServiceAE/api/" . $dbview->ae_project->title . "/DBViews/" . $dbview->title . "/?format=csv";
            $data['batch_id'] = $dbview->file_download_batch;
            $data['scheduled_by'] = $dbview->file_download_request_user;
            $data['data_update'] = $dbview->data_update;
            $data['aeproject_title'] = $dbview->ae_project->title;
            $data['dbview_title'] = $dbview->title;
            $data['project_title'] = $dbview->ae_project->project->title;

            if (file_exists($path."/".$file_name)) {
                unlink($path."/".$file_name);
            }
            file_put_contents(
                $path."/".$file_name,
                json_encode($data, JSON_PRETTY_PRINT)
            );
        }
        Log::channel('DownloadStackLog')->info("Add DBviews to download batch done for ae-project ". $aeproject);
    }
}
```

FIGURE 18 – Command Addtobatch

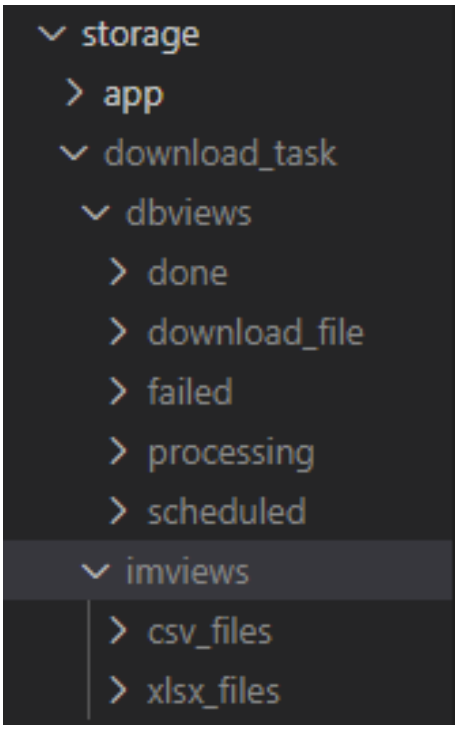


FIGURE 19 – Structure des dossiers de téléchargement

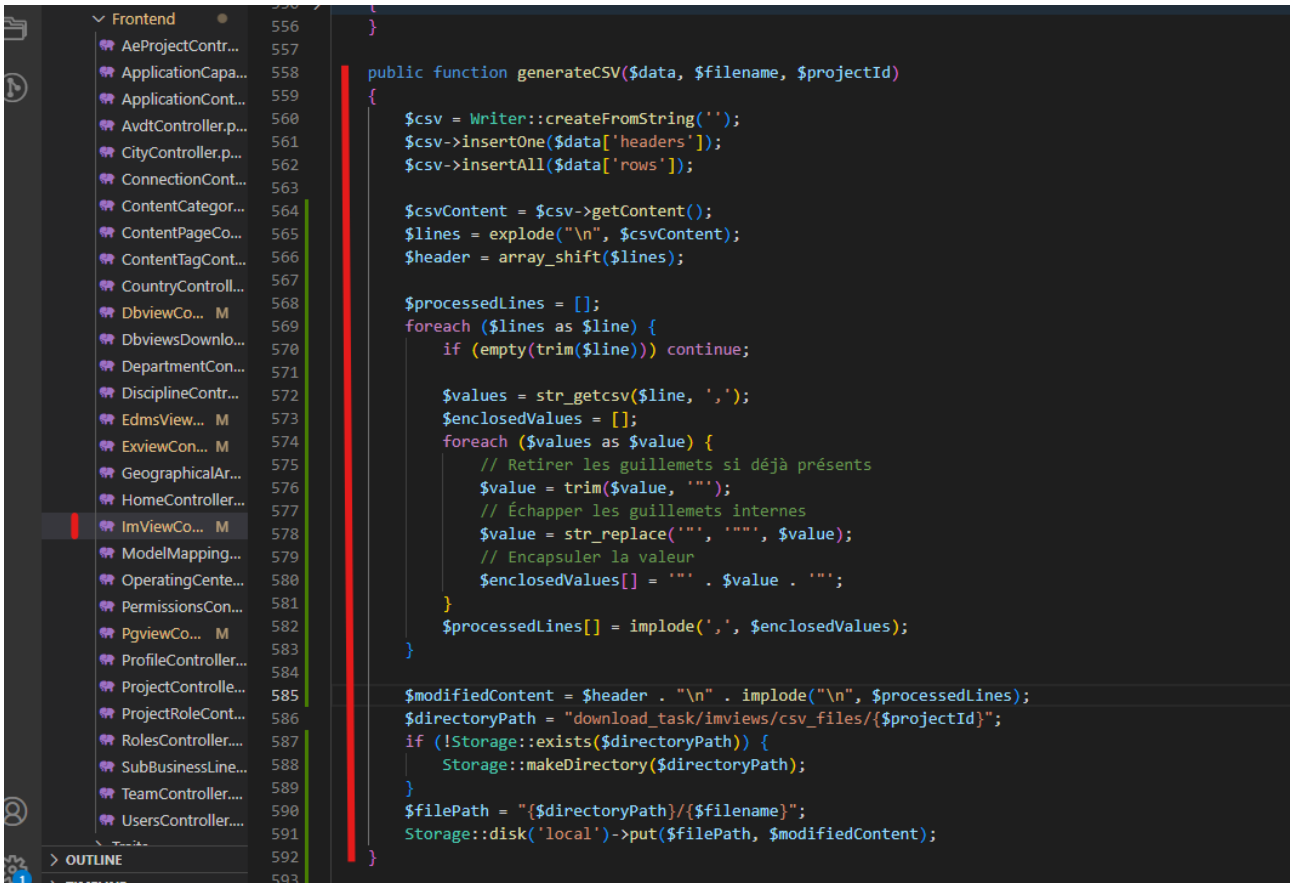


FIGURE 20 – Command generatecsvImview

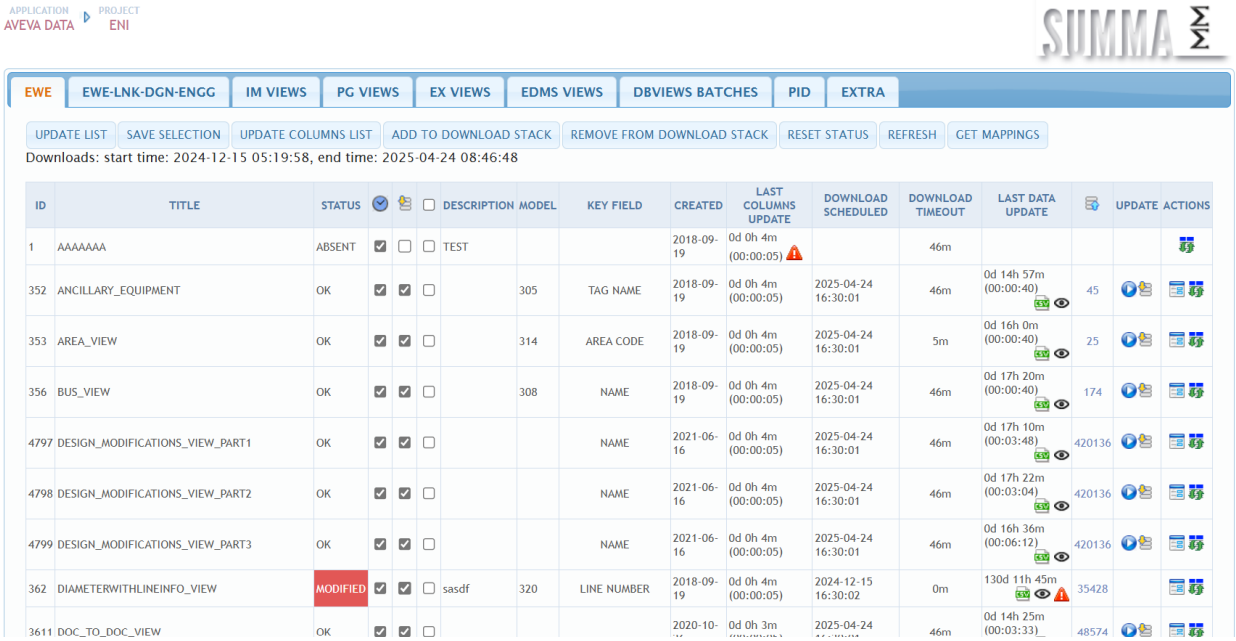


FIGURE 21 – Interface de la version PHP

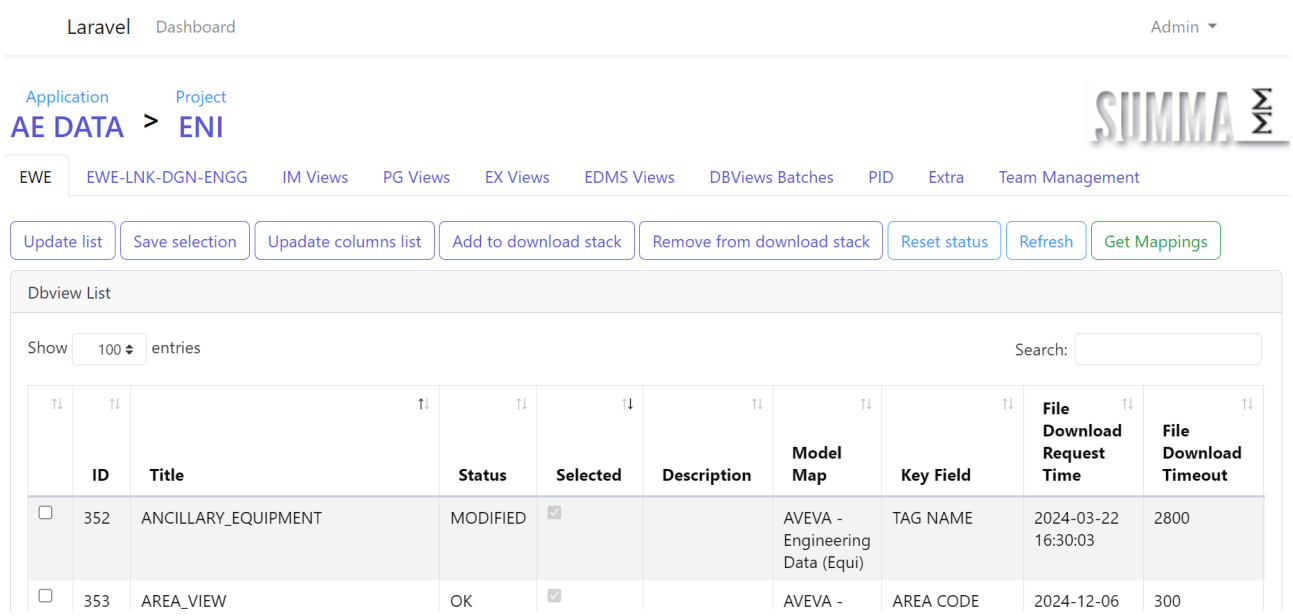


FIGURE 22 – Ancienne Interface Laravel

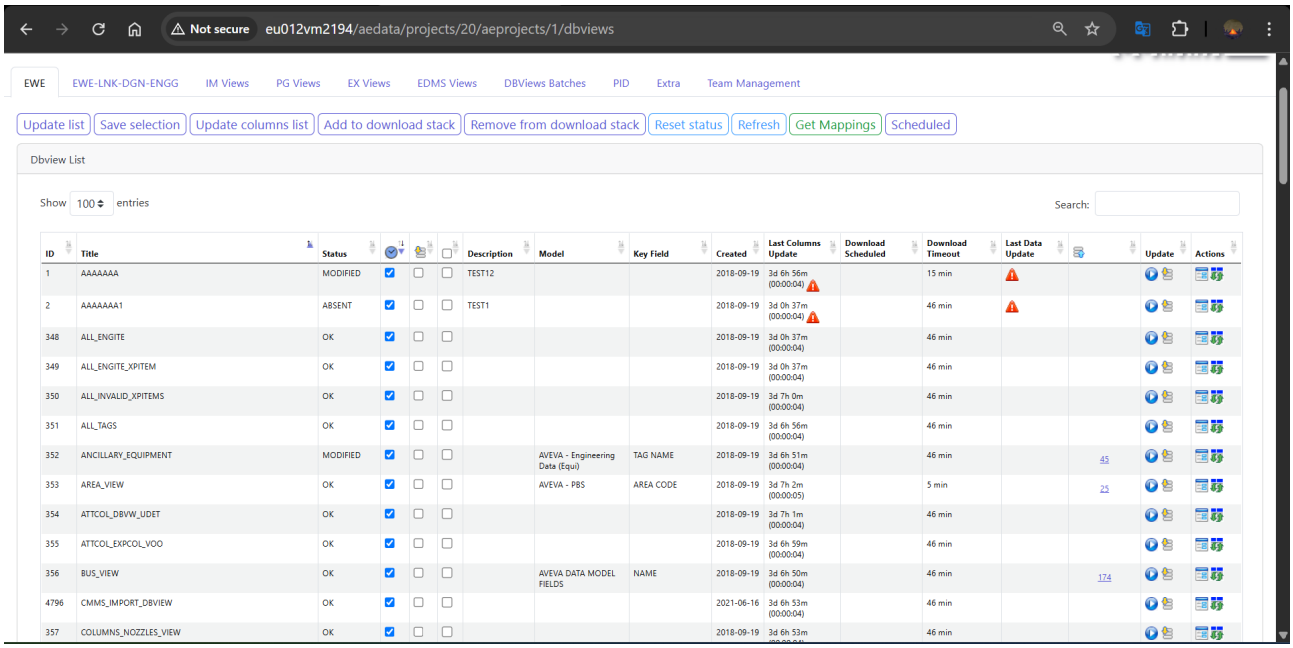
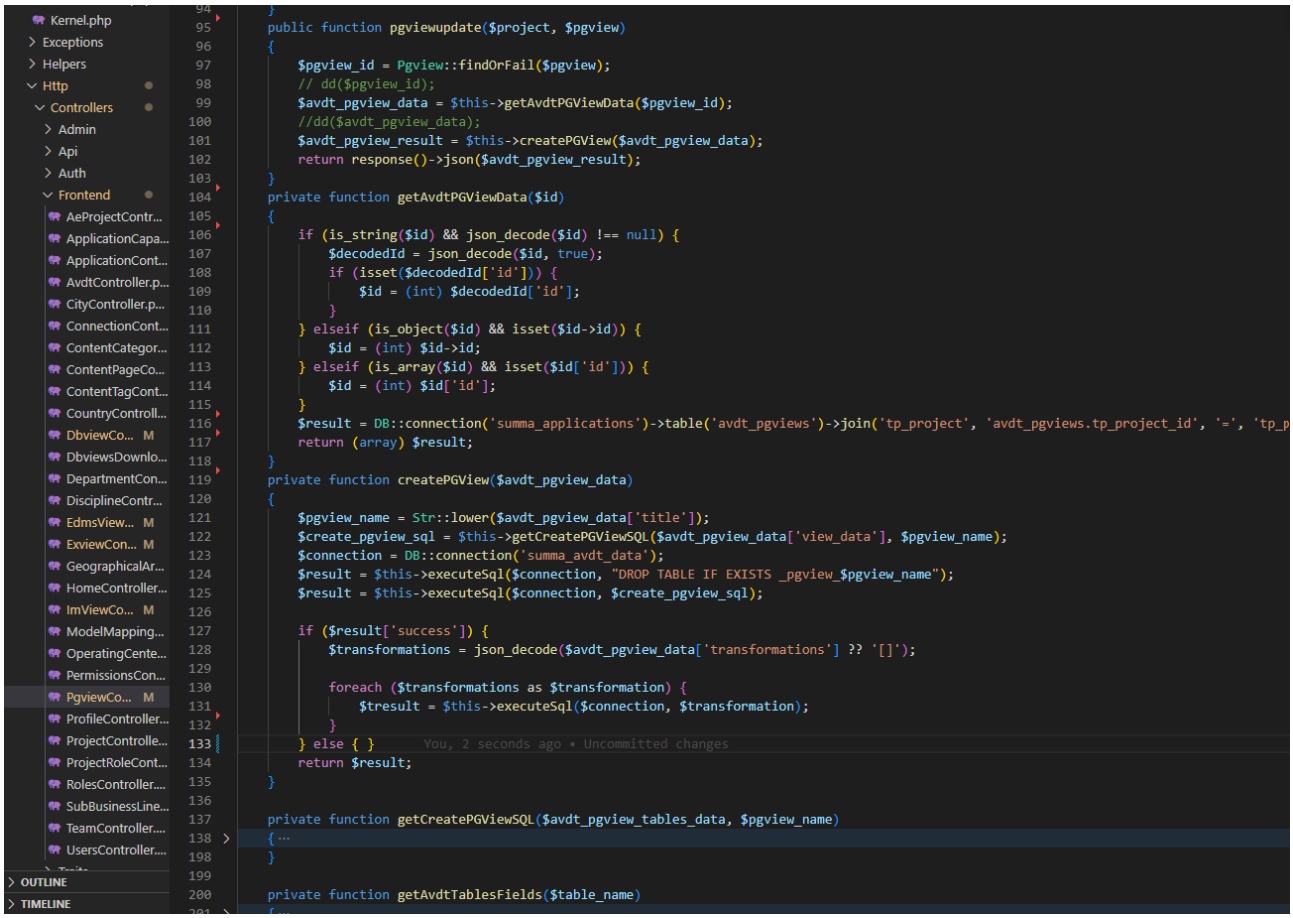


FIGURE 23 – Interface Actuel de la version Laravel



```php
    }
    public function pgviewupdate($project, $pgview)
    {
        $pgview_id = Pgview::findOrFail($pgview);
        // dd($pgview_id);
        $avdt_pgview_data = $this->getAvdtPGViewData($pgview_id);
        //dd($avdt_pgview_data);
        $avdt_pgview_result = $this->createPGView($avdt_pgview_data);
        return response()->json($avdt_pgview_result);
    }
    private function getAvdtPGViewData($id)
    {
        if (is_string($id) && json_decode($id) !== null) {
            $decodedId = json_decode($id, true);
            if (isset($decodedId['id'])) {
                $id = (int) $decodedId['id'];
            }
        } elseif (is_object($id) && isset($id->id)) {
            $id = (int) $id->id;
        } elseif (is_array($id) && isset($id['id'])) {
            $id = (int) $id['id'];
        }
        $result = DB::connection('summa_applications')->table('avdt_pgviews')->join('tp_project', 'avdt_pgviews.tp_project_id', '=', 'tp_p
        return (array) $result;
    }
    private function createPGView($avdt_pgview_data)
    {
        $pgview_name = Str::lower($avdt_pgview_data['title']);
        $create_pgview_sql = $this->getCreatePGViewSQL($avdt_pgview_data['view_data'], $pgview_name);
        $connection = DB::connection('summa_avdt_data');
        $result = $this->executeSql($connection, "DROP TABLE IF EXISTS _pgview_$pgview_name");
        $result = $this->executeSql($connection, $create_pgview_sql);

        if ($result['success']) {
            $transformations = json_decode($avdt_pgview_data['transformations'] ?? '[]');

            foreach ($transformations as $transformation) {
                $tresult = $this->executeSql($connection, $transformation);
            }
        } else { }      You, 2 seconds ago • Uncommitted changes
        return $result;
    }

    private function getCreatePGViewSQL($avdt_pgview_tables_data, $pgview_name)
    {...
    }

    private function getAvdtTablesFields($table_name)
```

FIGURE 24 – Les fonctions du Pgviewscontroller

```php
private function getAvdtExViewData($id)
{
    if (is_string($id) && json_decode($id) !== null) {
        $decodedId = json_decode($id, true);
        if (isset($decodedId['id'])) {
            $id = (int) $decodedId['id'];
        }
    } elseif (is_object($id) && isset($id->id)) {
        $id = (int) $id->id;
    } elseif (is_array($id) && isset($id['id'])) {
        $id = (int) $id['id'];
    }
    $avdtExviewData = DB::connection('summa_applications')
        ->table('avdt_exviews')
        ->join('tp_project', 'avdt_exviews.tp_project_id', '=', 'tp_project.id')
        ->where('avdt_exviews.id', $id)
        ->select(['avdt_exviews.*', 'tp_project.code as tp_project_code', 'tp_project.title as tp_project_title', 'tp_project.imviews_folder as imviews_folder', 'tp_project.regex_file_name'])
        ->first();

    return $avdtExviewData ? (array) $avdtExviewData : null;
}
```

FIGURE 25 – La fonctions du getAvdtExViewData