

Rapport de stage

Développement d'un convertisseur de fichiers géospatiaux

Entreprise : Poisson Soluble

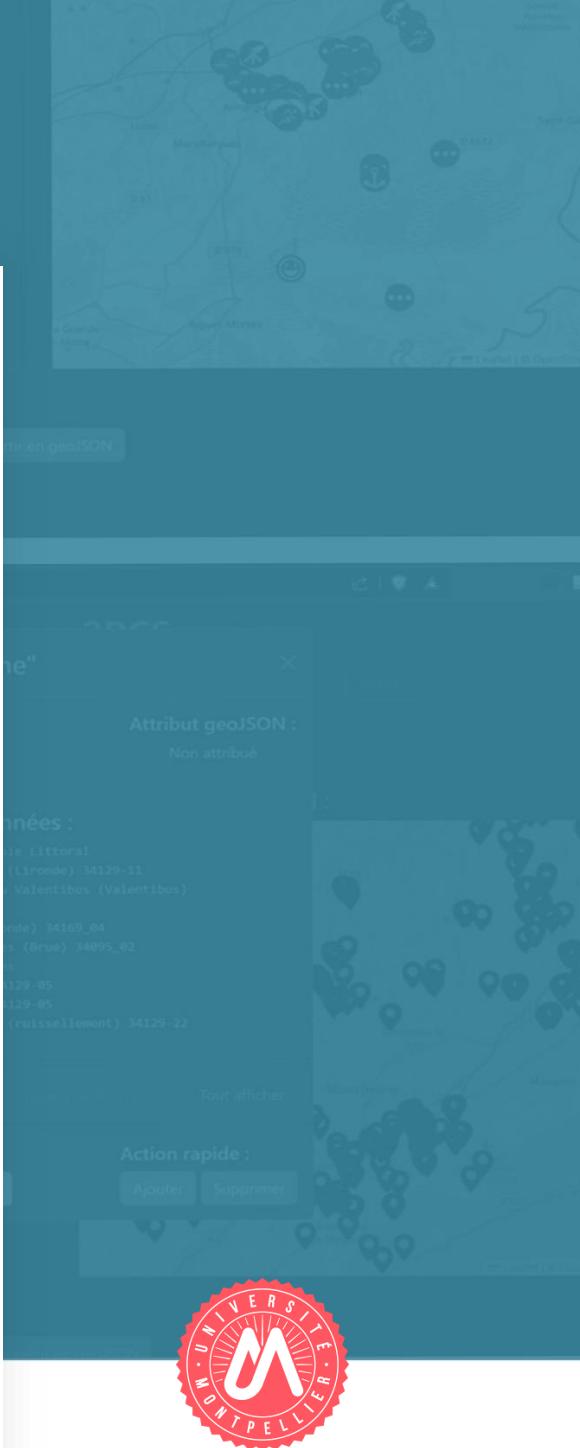
Créé par : Aurel GEORGES

Diplôme : BUT Informatique

Année universitaire : 2023 – 2024

Tuteur IUT : M. Antoine CHOLLET

Tuteur Poisson Soluble : M. Stéphane WOUTERS



Remerciements

Je tiens à exprimer ma gratitude à mon tuteur de stage chez Poisson Soluble, M. Stéphane WOUTERS, pour son encadrement, ses conseils précieux et son aide tout au long de ce projet. Je remercie également mon tuteur de l'IUT, M. Antoine CHOLLET, pour son accompagnement et son expertise qui ont grandement contribué à la réussite de ce stage. Enfin, je souhaite remercier toute l'équipe de Poisson Soluble : Patrice, Fred, Jonathan, Olivier, Faniry, David, Alex, Christophe, Théo, Léo et Vincent pour leur accueil chaleureux et leur sympathie.

Résumé & Abstract

Résumé :

Le projet consiste à développer une application web permettant de convertir automatiquement des fichiers shapefile en fichiers geoJSON, facilitant ainsi l'intégration des données géospatiales dans les systèmes de Predict Services, client principal de Poisson Soluble. L'application utilise le framework Symfony pour le backend, avec une architecture MVC, et intègre des technologies comme PostgreSQL/PostGIS pour le traitement des données géométriques. Les fichiers shapefile sont décompressés, analysés pour déterminer leur code EPSG, puis convertis et stockés dans une base de données. Les attributs peuvent être mappés et ajustés automatiquement ou manuellement pour correspondre aux conventions de nommage de Poisson Soluble. Le fichier geoJSON final est généré et renvoyé. L'application comprend une interface utilisateur intuitive et des fonctionnalités automatisées pour améliorer l'efficacité et la précision de la gestion des données géospatiales.

Abstract :

The project involves developing a web application to automatically convert shapefile data into geoJSON format, thus facilitating the integration of geospatial data into Predict Service's systems, the main client of Poisson Soluble. The application uses the Symfony framework for the backend, with an MVC architecture, and incorporates technologies such as PostgreSQL/PostGIS for processing geometric data. Shapefiles are decompressed, analyzed to determine their EPSG code, then converted and stored in a database. Attributes are automatically mapped and adjusted to match Poisson Soluble's naming conventions, and the final geoJSON is generated and returned. The application features an intuitive user interface and automated functions to enhance the efficiency and accuracy of geospatial data management.

Mots-clés/ Keys-Word

Mots-clés :

données géospatiales, conversion de shapefile, geoJSON, framework Symfony, architecture MVC, PostgreSQL, PostGIS, code EPSG, traitement automatisé des données, application web, intégration des données, mapping des attributs, Poisson Soluble, Predict, Predict Services, interface utilisateur, développement backend, gestion des données, fichier shapefile, fichiers JSON, développement logiciel.

Keys-Word :

geospatial data, shapefile conversion, geoJSON, Symfony framework, MVC architecture, PostgreSQL, PostGIS, EPSG code, automated data processing, MVC web application, data integration, attribute mapping, Poisson Soluble, Predict, Predict Service, user interface, backend development, data management, shapefile, JSON files, software development.

Table des matières

| | |
|--|-----|
| Remerciements..... | I |
| Résumé & Abstract..... | II |
| Mots-clés/ Keys-Word | III |
| Table des matières | IV |
| Table des figures..... | VI |
| Glossaire | VII |
| Introduction..... | 1 |
| Présentation de l'entreprise..... | 2 |
| Cahier des charges..... | 3 |
| Contexte du projet | 3 |
| Description du projet..... | 3 |
| Analyse de l'existant..... | 4 |
| Situation actuelle..... | 4 |
| Architecture de l'application web Prédikt..... | 4 |
| Spécifications Fonctionnelles | 5 |
| Spécifications techniques | 5 |
| Backend | 5 |
| Frontend | 5 |
| Base de données..... | 5 |
| Déploiement | 6 |
| Spécifications non-fonctionnelles | 6 |
| Interface utilisateur intuitive..... | 6 |
| Interface utilisateur claire | 6 |
| Interface utilisateur simple | 7 |
| Un README clair et documenté..... | 7 |
| Rapport Technique | 8 |
| Conception..... | 8 |
| Conception du backend..... | 10 |
| Conception de l'interface | 13 |

| | |
|---|-----|
| Réalisation | 21 |
| Réalisation du backend..... | 21 |
| Réalisation du frontend..... | 27 |
| Validation..... | 31 |
| Difficulté technique | 32 |
| Intégration des shapefiles contenant des lignes..... | 33 |
| Changement d'approche de développement | 33 |
| Ecouteurs d'évènement dédoublé | 33 |
| Les SRID et la commande Ogrinfo | 34 |
| Méthodologie et organisation du projet..... | 35 |
| L'organisation du travail dans l'entreprise..... | 35 |
| Méthodes de Développement et de Travail..... | 35 |
| Conclusion | 37 |
| Visa du tuteur de stage..... | 38 |
| Bibliographies | I |
| Annexes techniques..... | II |
| Annexe 1. Fonction getData du ShapeRepository..... | II |
| Annexe 2. Google document de suivi de projet | II |
| Annexe 3. Readme de l'application | III |
| Annexe 4. Exemple de l'historique des commit | V |
| Annexe 5. Fichier de questions à poser avant un point | V |
| Annexe 6. Notes Notion de Stage..... | VI |

Table des figures

| | |
|--|----|
| Figure 1 : Diagramme des cas d'usage du convertisseur | 8 |
| Figure 2 : Diagramme d'activité de la fonctionnalité convertir | 9 |
| Figure 3 : Diagramme de séquence de la fonctionnalité convertir..... | 10 |
| Figure 4 : Page principale du projet | 13 |
| Figure 5 : Page principale du projet avec une erreur | 14 |
| Figure 6 : Deuxième block de la page principale du projet..... | 14 |
| Figure 7 : Troisième block de la page principale | 15 |
| Figure 8 : Quatrième block de la page principale..... | 16 |
| Figure 9 : Modal d'un attribut | 16 |
| Figure 10 : Utilisation de l'attribution | 17 |
| Figure 11 : Exemple du code couleur des attributs..... | 18 |
| Figure 12 : Pop-ups de la map | 19 |
| Figure 13 : Modal des options sans erreurs | 20 |
| Figure 14 : Modal des options avec une erreur | 20 |
| Figure 15 : Page principale en mode clair | 21 |
| Figure 16 : MainController backend..... | 22 |
| Figure 17 : Fonction findShapeFile backend | 22 |
| Figure 18 : Fonction searchFile backend | 23 |
| Figure 19 : Fonction findEPSG backend..... | 23 |
| Figure 20 : Exemple de résumé de ogrinfo | 24 |
| Figure 21 : Fonction pushDataInDB | 25 |
| Figure 22 : Fonction geoJSON..... | 25 |
| Figure 23 : Fonction setAttribut | 26 |
| Figure 24 : Fonction findFile frontend..... | 28 |
| Figure 25 : Fonction findEPSG frontend | 29 |
| Figure 26 : Fonction pushDataInDb frontend..... | 30 |
| Figure 27 : Fonction appendDataExample frontend | 30 |
| Figure 28 : Fonction checkSyntaxe frontend | 31 |
| Figure 29 : Fonction removeAllEventListeners frontend | 34 |

Glossaire

| Terme | Définition |
|--|--|
| Angular | Angular est un framework open source TypeScript développé par Google, utilisé pour créer des applications web à interface utilisateur dynamique et interactive. |
| Backend | La partie d'une application informatique qui traite les données en arrière-plan et qui n'est pas directement visible par l'utilisateur. Il est responsable du traitement des requêtes, de l'accès aux bases de données, de la logique métier et de la gestion des opérations serveur. |
| Docker | Une plateforme logicielle qui permet de développer, déployer et exécuter des applications dans des conteneurs. Les conteneurs Docker encapsulent les logiciels avec toutes leurs dépendances dans une unité standardisée, assurant ainsi une portabilité et une efficacité accrues pour le déploiement d'applications sur différentes infrastructures informatiques. |
| Frontend | La partie visible et interactive d'une application informatique avec laquelle l'utilisateur interagit. |
| GeoJSON | Un format de fichier utilisé pour stocker des données géospatiales, basé sur la syntaxe JSON. Il permet de représenter des points, des lignes et des polygones, ainsi que des attributs associés. |
| PostGIS | Une extension open source pour la base de données PostgreSQL, permettant de gérer des données géographiques et spatiales. Elle comprend aussi une bibliothèque contenant des commandes pour manipuler des données géospatiales. |
| PostgreSQL | PostgreSQL est un système de gestion de base de données relationnelle open source. |
| Fichier de forme ou Shapefile ou fichier Shape | Un format de fichier géospatial utilisé pour stocker des données géographiques, telles que des points, des lignes et des polygones, ainsi que des attributs associés. Développé par Esri, il est largement utilisé dans les systèmes d'information géographique (SIG) en raison de sa simplicité et de sa compatibilité. |
| Symfony | Un Framework PHP open source utilisé pour le développement web. Il offre une structure modulaire et des composants réutilisables pour faciliter la création d'applications web robustes et évolutives. |

Introduction

Avec l'évolution rapide de la technologie et la transformation numérique des entreprises, il existe une grande diversité de moyens de stocker des données. Toutes ces manières de stocker ont donné lieu à une multitude de types de fichiers différents. Cela pose des défis en termes de compatibilité et d'interopérabilité entre les systèmes. La capacité de convertir les données d'un format à un autre est devenue essentielle pour optimiser leur utilisation, réduire les erreurs de traitement et améliorer l'efficacité des opérations. Que ce soit pour la gestion de données géospatiales, la communication entre différents logiciels ou l'intégration dans des applications complexes, la conversion de fichiers est un processus clé pour assurer une gestion fluide et efficace des informations.

Actuellement, l'intégration de données géospatiales nécessite souvent un processus manuel fastidieux ou le recours à des méthodes onéreuses. Ce processus manuel est non seulement sujet à des erreurs humaines, mais il est également chronophage, ce qui ralentit considérablement les flux de travail. De plus, les solutions coûteuses ne sont pas toujours accessibles pour toutes les entreprises, limitant ainsi leur capacité à gérer efficacement les données géospatiales. Cette situation met en évidence le besoin urgent de solutions automatisées et abordables pour convertir et intégrer ces données de manière fiable et rapide.

C'est dans ce contexte que s'inscrit mon stage chez Poisson Soluble, une entreprise spécialisée dans la communication multimédia. Poisson Soluble travaille principalement pour la société Predict, spécialisée dans la gestion des risques climatiques. Mon projet de stage consiste en la conception et le développement d'un convertisseur de fichiers de forme en fichiers de données géospatiales au format JSON. Ce projet vise à automatiser le processus de conversion pour faciliter l'intégration de différents fichiers dans les applications développées pour Predict.

Après une présentation détaillée de Poisson Soluble, en expliquant son positionnement dans le secteur de la communication multimédia et ses principaux objectifs, l'état du cahier des charges du projet sera effectué. L'existant sera analysé, puis les spécifications techniques, fonctionnelles et non-fonctionnelles du projet seront exposées. Ensuite, un rapport technique sera présenté, passant en revue la conception, la réalisation, la validation et les différentes difficultés rencontrées. Enfin, après avoir décrit l'organisation adoptée pour ce projet, une conclusion synthétisera les principaux résultats et les enseignements tirés de cette expérience.

Présentation de l'entreprise

Mon stage se déroule chez Poisson Soluble, une entreprise fondée en 2005, spécialisée dans la communication multimédia. L'objectif principal de Poisson Soluble est d'assister les particuliers et les entreprises dans la création d'une identité visuelle et de les accompagner dans l'élaboration de supports de communication ou de vente, principalement sous forme de sites web ou d'e-commerce, ainsi que d'applications mobiles ou web.

Poisson Soluble est composée de dix employés. Les locaux de l'entreprise sont basés au Mas Rouge, en périphérie de Montpellier. L'entreprise est dirigée par M. Patrice ROBERT, fondateur et chef de projet de l'entreprise. M. Frédéric VERNAY est responsable administratif et gère les ressources humaines. Mon tuteur de stage, M. Stéphane WOUTERS, occupe le poste de Lead Développeur. L'équipe inclut également un web designer, deux développeurs full-stack, trois développeurs backend*, un chef de projet et un développeur frontend*.

Le principal client de Poisson Soluble est Predict Services, une entreprise spécialisée dans la gestion des risques climatiques. Les clients de Predict Services, présents en France et à l'international, incluent des collectivités, des professionnels et des assureurs. Chacun de ces organismes peut accéder à un espace qui lui est propre au sein de l'application de Predict Services développée par Poisson Soluble. Ce projet occupe six développeurs à temps plein et représente la majorité du chiffre d'affaires de l'entreprise. Parallèlement, Poisson Soluble continue de créer des identités visuelles et de développer des sites et applications web ou mobiles, bien que le volume de ces projets soit limité du fait de l'engagement de ressources sur le projet Predict.

Les enjeux de mon stage chez Poisson Soluble gravitent autour de l'automatisation d'un processus, alignant ainsi la mission du stage avec les besoins stratégiques de l'entreprise. La création du convertisseur de fichiers shapefile* en geoJSON* est une réponse directe à la nécessité de moderniser les méthodes de travail et d'optimiser les interactions avec les données géographiques. En termes de compétences, ce stage offre l'opportunité de développer des aptitudes essentielles dans la gestion de projets informatiques et la collaboration au sein d'une équipe technologique. Le recours aux méthodes classiques pour la gestion du projet, en particulier, permet de s'aligner avec les pratiques modernes du secteur et de répondre efficacement aux besoins des utilisateurs finaux. En outre, ce stage me permet de me familiariser avec l'écosystème numérique et de comprendre les implications de la transformation numérique au sein d'une organisation, en mettant en pratique des concepts tels que la digitalisation des processus et la responsabilité sociétale des entreprises.

Cahier des charges

Contexte du projet

Au cours de cette année, Poisson Soluble a reçu à deux reprises des fichiers shapefile à intégrer dans son application. En effet, Predict compte parmi ses clients des communes françaises, qui sont tenues de disposer de Plans Communaux de Sauvegarde^[1] (PCS). Les PCS sont des dispositifs obligatoires pour les communes, destinés à planifier les mesures de protection et de gestion de crise en cas d'événements majeurs, tels que des inondations ou des incendies. Ces plans sont des outils de gestion de crise, et Predict propose un service pour les gérer via son application.

Certaines communes disposent déjà de leur PCS sous forme de shapefile et les transmettent à Predict, qui demande alors à Poisson Soluble de les intégrer dans son outil. C'est dans ce contexte qu'est apparu le besoin d'un convertisseur de fichiers shapefile en geoJSON. Le développement de ce convertisseur permet non seulement de réduire les erreurs humaines et le temps nécessaire à l'intégration des données, mais aussi de standardiser les processus et d'améliorer l'efficacité globale de la gestion des PCS.

Description du projet

Le projet de stage a pour objectif la conception d'une application web permettant de convertir un fichier shapefile en fichier geoJSON. Les fichiers shapefile, ou fichiers de forme, ou fichier Shape*, sont un format standard utilisé dans les systèmes d'information géographique (SIG). Ils peuvent contenir un seul type de formes géométriques telles que des points, des lignes ou des polygones, ainsi que des données associées^[2]. Le format geoJSON, quant à lui, est un format basé sur JSON, optimisé pour représenter des entités géométriques accompagnées de leurs données^{[3] [4]}.

L'application web permet aux utilisateurs de télécharger un fichier compressé contenant le shapefile. Une fois le fichier téléchargé, l'utilisateur peut sélectionner les attributs spécifiques à inclure dans le fichier geoJSON final et lancer la conversion de manière automatique. L'application garantit la possibilité de renommer des attributs du shapefile en champs standards (donnés par Poisson Soluble), pour une utilisation efficace dans les applications web de Predict.

De plus, l'application inclut une fonctionnalité de prévisualisation des données, permettant aux utilisateurs de visualiser les points géographiques sur une carte avant de finaliser la conversion. Cette fonctionnalité est cruciale pour vérifier l'exactitude des données et assurer une intégration sans faille dans les systèmes de Predict. L'interface utilisateur est conçue pour être intuitive, claire et simple, facilitant ainsi son utilisation par les développeurs internes de Poisson Soluble. Ce projet vise à améliorer l'efficacité et la précision de

l'intégration des données géospatiales, en automatisant et en simplifiant le processus de conversion des fichiers shapefile en geoJSON.

Analyse de l'existant

Préalablement au développement de l'application de conversion de fichiers shapefile en geoJSON, il est important d'examiner le système et les processus actuels utilisés chez Poisson Soluble. Cette analyse aide à identifier les limitations et les défis de l'intégration manuelle des fichiers géospatiaux. En comprenant ces éléments, nous pouvons mieux concevoir une solution adaptée aux besoins de Poisson Soluble.

Situation actuelle

Avant le développement du convertisseur automatique, Poisson Soluble gérait l'intégration des Plans Communaux de Sauvegarde (PCS) des nouveaux clients communaux de Predict de manière semi manuelle, un processus fastidieux et sujet à erreurs. Les communes, lors de leur intégration en tant que clients, fournissaient un shapefile contenant les données géographiques du PCS. Pour intégrer ces données dans l'outil de Predict, qui utilise le format geoJSON, un processus en plusieurs étapes était nécessaire. Initialement, les données du shapefile étaient transférées dans une base de données PostgreSQL* à l'aide d'une commande de PostGIS* (outil permettant la gestion de données géospatiales). Cette étape était suivie d'un processus manuel où les données étaient extraites de la base pour être reformatées en geoJSON. Certains attributs du shapefile devaient être renommés ou modifiés. Ce processus non seulement requérait une intervention manuelle significative, augmentant le risque d'erreurs et de perte de données, mais il était aussi chronophage. Mon projet vise à automatiser ces actions pour que quelques clics suffisent à convertir un shapefile en geoJSON, le tout en utilisant le même environnement technique que les développeurs qui travaillent sur l'application de Predict.

Architecture de l'application web Prédic

Les applications et services web destinés à Predict sont principalement développés en PHP, utilisant le framework Symfony* pour le backend. Pour les interfaces utilisateur, selon leur complexité, Poisson Soluble utilise Angular* pour les applications plus importantes et React pour les interfaces plus légères. Le déploiement de ces applications est géré via Docker*, permettant une mise en œuvre homogène et standardisée sur les serveurs de Poisson Soluble. La gestion de version est réalisée à travers Git, avec le code hébergé sur GitHub.

Spécifications Fonctionnelles

L'application doit inclure plusieurs fonctionnalités essentielles pour répondre aux besoins de Poisson Soluble.

Premièrement, l'application doit garantir l'intégrité des données géographiques du shapefile et les rediriger dans un fichier JSON suivant la norme RFC 7946, plus communément appelée geoJSON. Ce fichier doit être téléchargeable.

Deuxièmement, il doit être possible d'effectuer diverses actions sur les colonnes contenant des données associées à chaque point. Parmi ces actions, l'utilisateur doit pouvoir ajouter une colonne au geoJSON, ce qui implique que les données de cette colonne soient reportées dans le geoJSON avec la même association que dans le shapefile. De plus, l'utilisateur doit avoir la possibilité d'ajouter une colonne avec un nom correspondant aux besoins spécifiques de Poisson Soluble. Enfin, l'application doit permettre de supprimer une colonne précédemment ajoutée du geoJSON.

Spécifications techniques

Le développement du convertisseur de fichiers pour Poisson Soluble nécessite l'adoption de technologies et d'approches précises. Voici les choix qui ont été faits.

Backend

L'utilisation de Symfony comme cadre de développement principal s'explique par son adoption étendue par l'équipe de développement de Predict. Cela assure une meilleure cohérence et maintenabilité. Symfony facilite également le développement modulaire, permettant d'ajouter uniquement les modules nécessaires à l'application.

Frontend

Pour le frontend, l'utilisation de Bootstrap permet une conception facile avec un minimum de responsive design. JavaScript, en combinaison avec Stimulus et Webpack Encore, a été choisi pour minimiser les dépendances externes. Leaflet a été sélectionné pour permettre la visualisation des formes du shapefile sur la carte, car c'est l'outil utilisé par les équipes.

Base de données

PostgreSQL, avec son extension PostGIS^[5], est utilisé en raison de sa compatibilité native avec les opérations géospatiales. Cela est essentiel pour le traitement et la gestion des données issues des fichiers shapefile. Cette technologie a été spécifiquement choisie pour sa

capacité à exécuter la commande `shp2pgsql`, facilitant ainsi la conversion des données en un format directement utilisable.

Déploiement

Docker est utilisé pour le déploiement de l'application, permettant une mise en œuvre uniforme et efficace sur les serveurs. La création et la mise à jour régulières du Dockerfile garantissent que toutes les dépendances et configurations nécessaires sont correctement instaurées et maintenues à travers les différentes phases du déploiement.

Spécifications non-fonctionnelles

Les spécifications non-fonctionnelles de l'application de conversion de fichiers shapefile en geoJSON sont essentielles pour garantir une expérience utilisateur optimale pour les développeurs internes de Poisson Soluble. Ces spécifications se concentrent sur l'interface utilisateur et la documentation, assurant ainsi une utilisation efficace et intuitive de l'application.

Interface utilisateur intuitive

L'interface utilisateur doit être conçue de manière à être intuitive, permettant aux développeurs de naviguer facilement à travers les différentes fonctionnalités de l'application. Chaque étape du processus de conversion doit être clairement indiquée, avec des éléments interactifs bien définis. Les développeurs doivent pouvoir comprendre immédiatement comment télécharger des fichiers, sélectionner des attributs et lancer le processus de conversion sans avoir besoin d'explications. L'objectif est de minimiser le temps d'apprentissage et de maximiser l'efficacité des utilisateurs.

Interface utilisateur claire

L'interface utilisateur doit être claire, en utilisant un design propre et épuré. Les informations doivent être présentées de manière cohérente, avec des libellés explicites et des icônes compréhensibles. Les messages d'erreur et les confirmations de succès doivent être affichés de manière visible et compréhensible, permettant aux développeurs de savoir exactement ce qui se passe à chaque étape. La clarté de l'interface contribue à réduire les erreurs et les frustrations, améliorant ainsi l'expérience utilisateur globale.

Interface utilisateur simple

Étant destinée à des développeurs internes, l'interface utilisateur doit rester simple et fonctionnelle. Elle doit se concentrer sur les fonctionnalités essentielles sans inclure des éléments superflus qui pourraient compliquer l'utilisation. Les développeurs doivent pouvoir accomplir leurs tâches rapidement et efficacement, avec un minimum de clics et de manipulations. La simplicité de l'interface garantit que les développeurs peuvent se concentrer sur leur travail principal sans être distraits par des éléments d'interface inutiles.

Un README clair et documenté

Un aspect crucial de l'application est la fourniture d'une documentation complète et claire sous la forme d'un fichier README. Ce fichier README doit expliquer de manière détaillée le processus d'installation, les prérequis système, les étapes pour lancer l'application et une description des principales fonctionnalités. Une documentation bien structurée est essentielle pour assurer que les développeurs puissent utiliser l'application de manière autonome et résoudre rapidement les éventuels problèmes qu'ils pourraient rencontrer.

Rapport Technique

Ce rapport technique est destiné à un informaticien et vise à fournir une analyse détaillée du développement de l'application de conversion de fichiers shapefile en geoJSON. Tout d'abord, la section conception décrira les choix architecturaux et les décisions de design prises pour le projet. Ensuite, la section réalisation détaillera les étapes et les fonctions les plus importantes. Puis, la partie validation expliquera les différentes méthodes employées pour tester l'application. Enfin, la section difficulté technique abordera les principaux défis rencontrés et les solutions apportées pour les surmonter.

Conception

La partie conception de ce rapport technique se divise en deux aspects principaux, la conception de la logique interne et la conception de l'interface utilisateur. Pour m'aider dans la conception, j'ai dessiné un diagramme des cas d'usage du convertisseur (cf. Figure 1) :

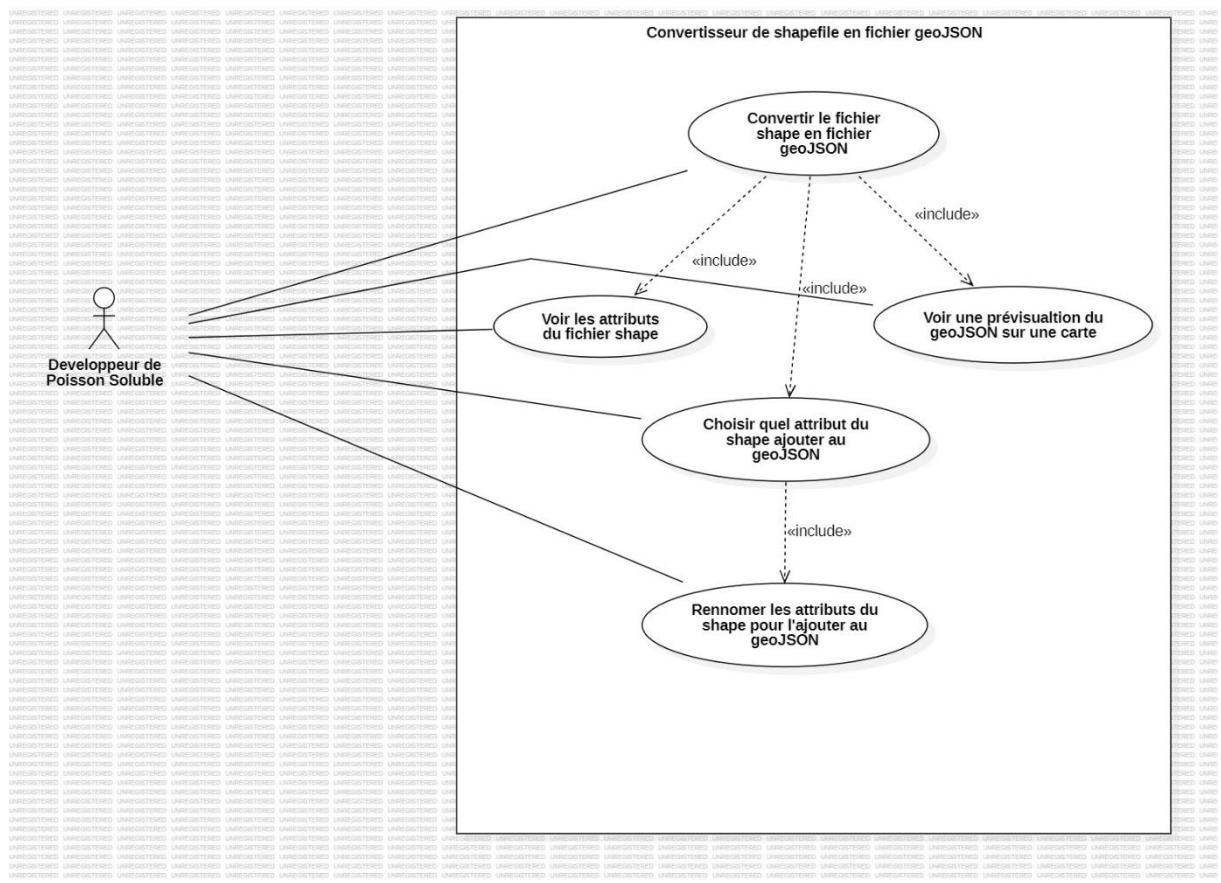


Figure 1 : Diagramme des cas d'usage du convertisseur

Afin de mieux comprendre comment fonctionne l'application, j'ai aussi élaboré un diagramme d'activité et de séquence (cf. Figure 2 et Figure 3) :

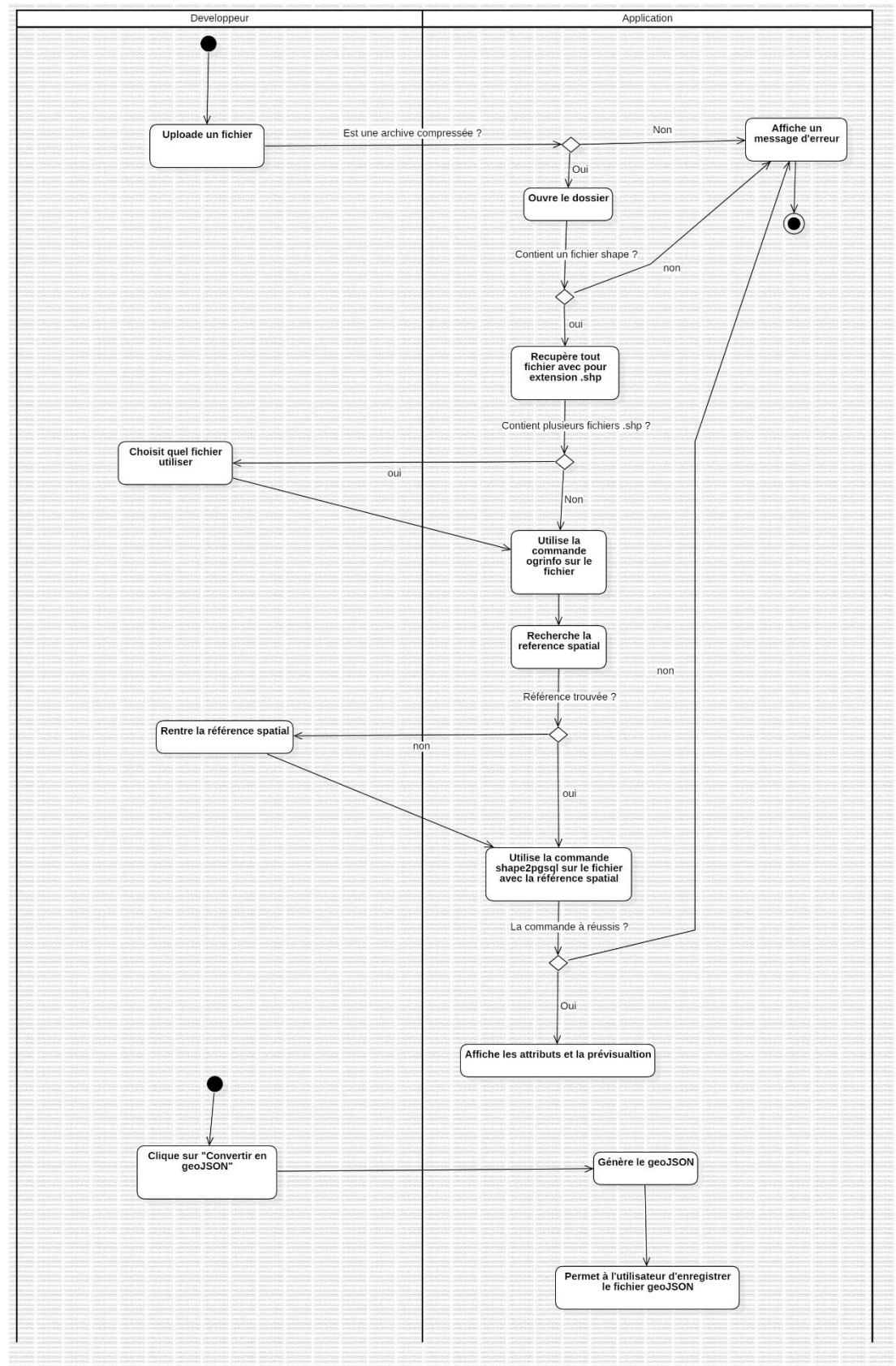


Figure 2 : Diagramme d'activité de la fonctionnalité convertir

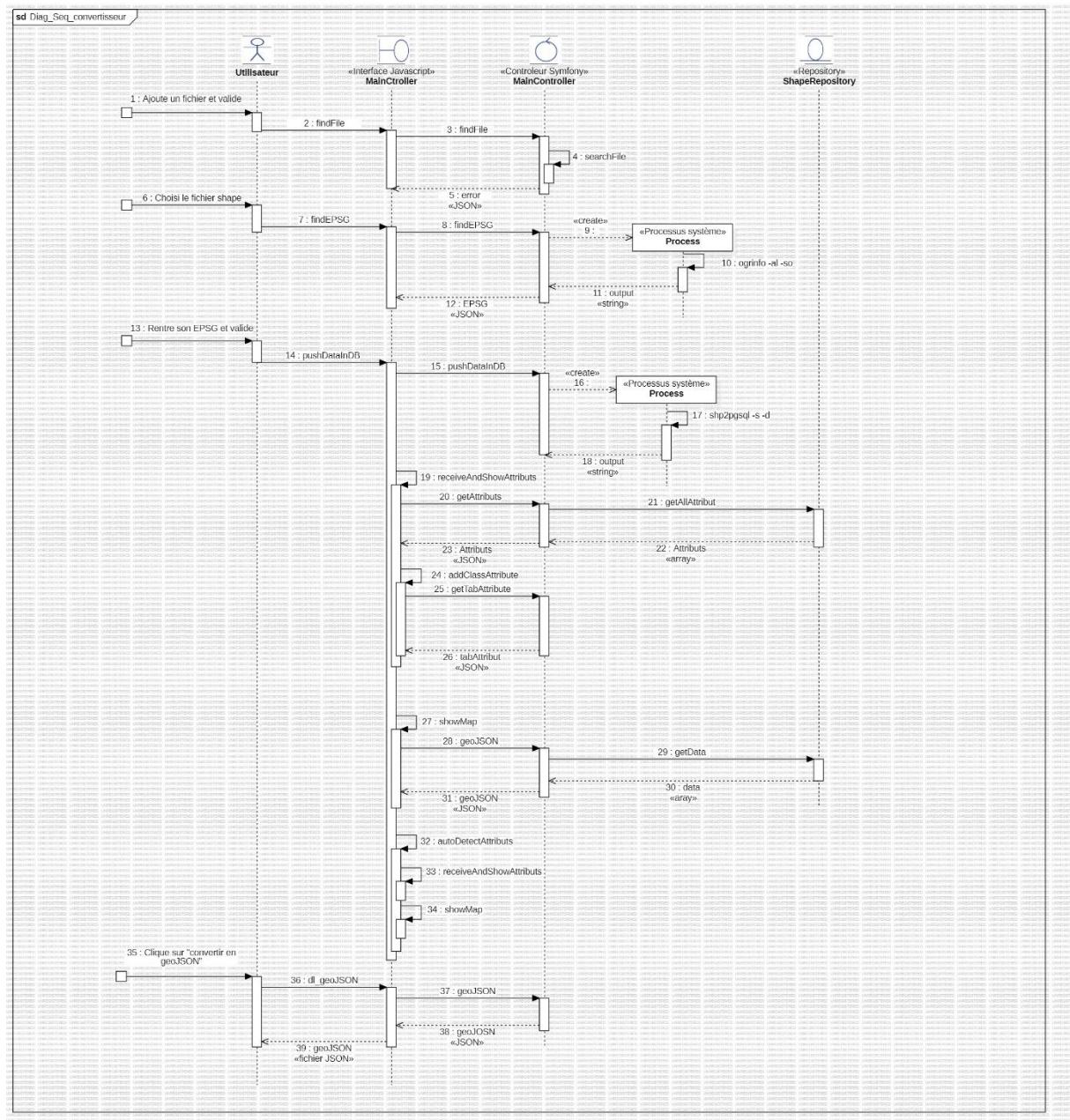


Figure 3 : Diagramme de séquence de la fonctionnalité convertir

Conception du backend

La logique interne, ou « backend », repose principalement sur Symfony, un framework PHP. J'ai appliqué le patron MVC (Modèle-Vue-Contrôleur) à l'architecture de l'application, encouragé par la structure de base de Symfony. À la racine du projet, on trouve un dossier « templates » et un dossier « src ». Le dossier « templates » contient des fichiers Twig, qui sont l'équivalent de fichiers HTML, mais avec des balises supplémentaires pour une modularité accrue. Ces fichiers constituent les vues de l'application. Le dossier « src » contient les contrôleurs et les repositories. Le contrôleur principal, nommé « MainController », regroupe toute la logique nécessaire au fonctionnement de l'application. Un autre contrôleur, «

`OptionController` », est dédié à la gestion des options. Pour que cette partie soit la plus claire possible, mes explications suivront le chemin d'un fichier dans le convertisseur.

La première étape pour convertir un shapefile en geoJSON consiste à utiliser la commande `shp2pgsql`^[6] de PostGIS. Les shapefiles sont des fichiers binaires, rendant la lecture des informations très complexe. Cette commande importe toutes les données d'un shapefile dans une base de données PostgreSQL. Pour fonctionner, elle nécessite également deux autres fichiers : un `.shx` et un `.dbf`. Le fichier `.shp` contient les entités géométriques, le `.shx` est un index binaire pour un accès rapide aux entités, et le `.dbf` contient les attributs des entités sous forme de table. Ces fichiers sont souvent envoyés sous forme d'archive compressée par les communes à Predict, puis à Poisson Soluble. Ainsi, l'utilisateur devra fournir un dossier compressé contenant ces trois fichiers pour démarrer la conversion.

Lorsque le développeur saisit le fichier et clique sur le bouton de validation, la fonction `findShapeFile` est appelée par le frontend à l'adresse « /`find_file` ». Cette fonction décomprime l'archive, recherche tous les fichiers se terminant par « `.shp` » et renvoie leurs noms. Elle s'appuie sur une autre fonction, `searchFile`, qui effectue une recherche récursive dans le dossier et ses sous-dossiers pour trouver un motif spécifique. La fonction `findShapeFile` retourne un tableau PHP structuré ainsi : `['error' => number, 'data' => string]`. En cas de succès, '`error`' est à zéro et '`data`' contient les chemins des shapefiles. Si une erreur survient, '`error`' passe à un et '`data`' contient le message d'erreur. J'ai réutilisé cette structure tout au long du projet, car elle permet de transmettre efficacement les messages d'erreur au frontend.

L'étape suivante consiste à rechercher le bon identifiant de référence spatiale, ou SRID^[7] (Spatial Reference Identification), du shapefile. Les données géométriques peuvent être encodées de multiples façons. L'EPSG (European Petroleum Survey Group)^[8] a défini des codes standards à quatre chiffres pour identifier les différents SRID. Leaflet, le module JavaScript recommandé pour afficher des cartes, utilise des coordonnées encodées en EPSG 4326. Heureusement, la commande `shp2pgsql` permet également de convertir les données géométriques d'un SRID à un autre. La fonction `find_EPSG` a donc pour objectif de déterminer le code EPSG avec lequel le shapefile a été encodé. Cette recherche est facilitée par l'utilisation de la commande « `ogrinfo` » de PostGIS. En utilisant les paramètres « `-al` » et « `-so` », cette commande résume les propriétés d'un fichier et permet d'en extraire son EPSG. Une fois le code EPSG identifié, il est renvoyé. En cas d'erreur lors de cette recherche, une chaîne vide est retournée. Cette méthode assure que les données géométriques sont correctement référencées, permettant ainsi leur conversion et leur affichage précis sur les cartes.

Une fois le shapefile à convertir sélectionné et son code EPSG identifié, la commande `shp2pgsql` de PostGIS peut être utilisée. Cette commande s'exécute en précisant l'EPSG du fichier shapefile ainsi que l'EPSG souhaité pour la base de données. Ensuite, la table dans laquelle les données géographiques et les attributs doivent être insérés est spécifiée. La fonction `pushDataInDB` lance cette commande et intègre le même système de gestion des erreurs que `findShapeFile`. Une fois la base de données remplie, le frontend affiche les

attributs grâce à la fonction `getAttributs`. Cette dernière appelle une fonction du repository qui récupère tous les noms des colonnes de la base de données, facilitant ainsi la manipulation et l'affichage des données.

Après que la base de données soit remplie et que les attributs du shapefile soient clairement identifiés, l'utilisateur doit pouvoir décider lesquels ajouter au geoJSON. Pour cela, la fonction `setAttribut` permet d'associer un nom spécifique dans le geoJSON à un attribut du shapefile. Par exemple, Poisson Soluble pourrait recevoir un shapefile avec les attributs « nom », « picto » et « description ». Toutefois, les noms utilisés par Poisson Soluble dans les geoJSON pour le nom, les pictogrammes et la description sont respectivement « `name` », « `alias` » et « `comment` ». La fonction `setAttribut` permet donc de créer une association entre le nom de l'attribut dans le shapefile et son équivalent dans le geoJSON. Cette association est ajoutée à un tableau PHP, stocké dans la session.

À ce stade, il est possible de construire le geoJSON. La fonction `geoJSON` se charge de cette tâche en parcourant les informations de la base de données. Elle ajoute au geoJSON les données de chaque attribut, en effectuant les associations définies dans le tableau stocké en session. En reprenant l'exemple précédent, les données associées à chaque point en tant que « `nom` » dans le shapefile, seront désormais nommées « `name` » dans le geoJSON. Une fois ce processus terminé, la fonction renvoie le geoJSON final.

Enfin, pour éviter au développeur d'ajouter manuellement les attributs dont les noms sont corrects, la fonction `autoDetectAttributs` s'en charge automatiquement. Elle recherche les correspondances entre les noms des attributs du shapefile et ceux des champs de la base de données de Poisson Soluble. Ces fonctionnalités principales sont assurées par les fonctions du contrôleur principal.

Lors du développement, mon premier réflexe, pour stocker la liste des noms des attributs nécessaires à Poisson Soluble a été de créer une fonction qui la renvoie. Cependant, il arrive que ces attributs changent lors de mises à jour majeures de l'application de Predict. C'est pourquoi j'ai décidé de stocker cette liste ainsi que les attributs détectés automatiquement dans des fichiers JSON, modifiables directement depuis l'interface de mon projet. À cet effet, le contrôleur « `OptionController` » contient des fonctions permettant de lire, modifier et vérifier le contenu de ces fichiers. Ces données auraient dû être persistantes, mais l'utilisation de docker fait que mon projet est souvent recharge et donc redéployé tel qu'il est sur le GitHub. Au moment où j'écris ce rapport, ce problème n'est pas encore réglé.

Pour conclure cette partie, la conception du backend a été pensée pour être appelée par le frontend et fournir toutes les informations nécessaires à l'affichage des données utiles à l'utilisateur. Le problème des données non-persistantes sera résolu d'ici la fin du stage.

Conception de l'interface

L'interface est un élément important du projet. Son but étant de simplifier un processus de l'entreprise, il faut que son interface soit facile et intuitive. C'est pourquoi j'ai opté pour un affichage crescendo des actions à effectuer, le tout sur une seule page. En effet, le convertisseur se présente sur une page internet découpée en quatre blocs. Chaque bloc est caché tant que le bloc précédent n'est pas complété. Si une erreur se produit et qu'elle est correctement gérée, elle peut être affichée par le biais d'alertes en rouge en haut de la page (cf. Figure 5).

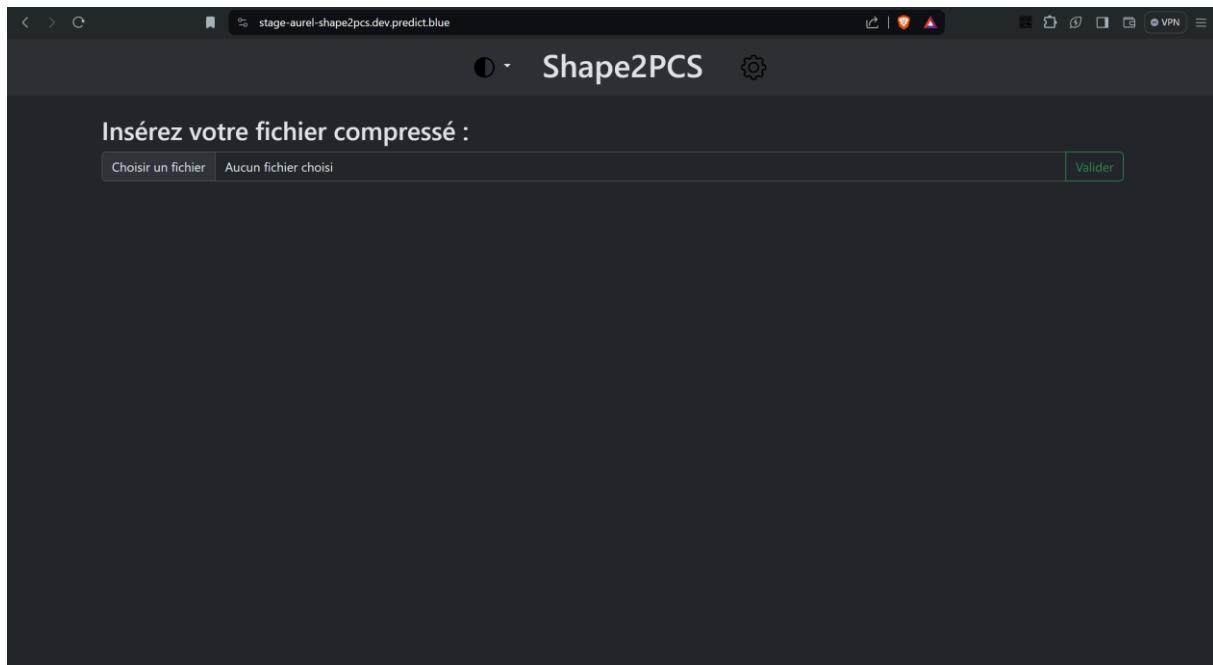


Figure 4 : Page principale du projet

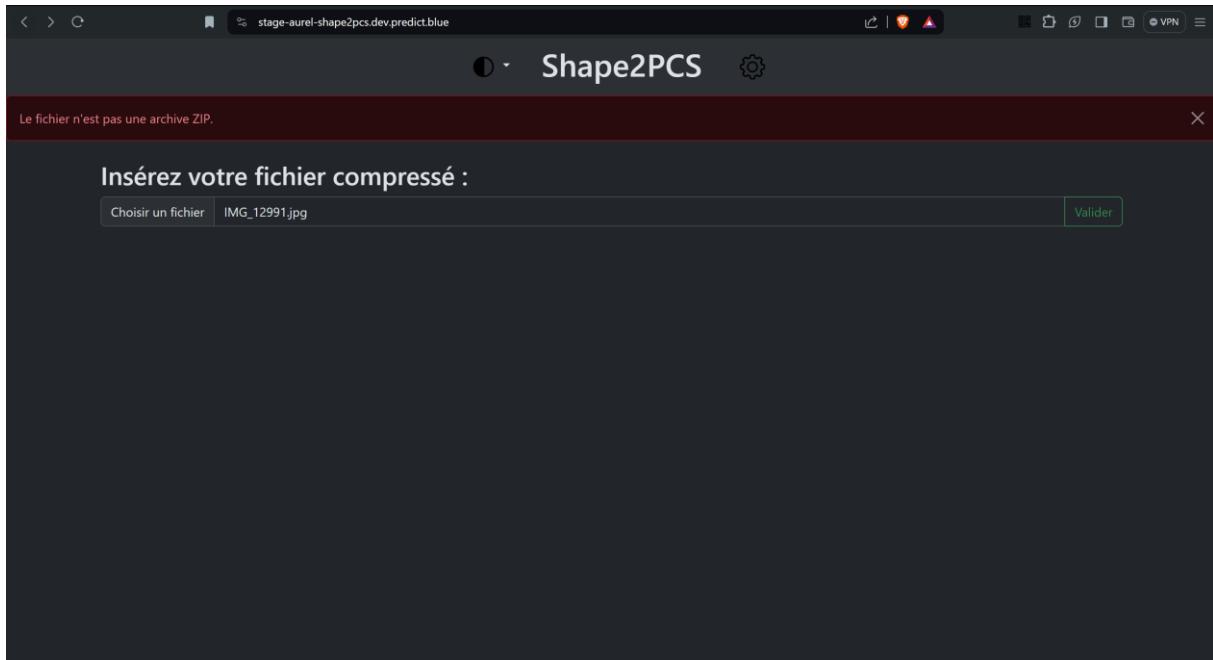


Figure 5 : Page principale du projet avec une erreur

Le premier bloc permet de télécharger un fichier compressé contenant le shapefile et ses fichiers associés (cf. Figure 4). En appuyant sur le bouton "valider", la fonction `findFile` de l'interface est déclenchée, et transmet le fichier à la fonction `findShapeFile` du backend. Si la réponse obtenue n'est pas une erreur, le deuxième bloc s'affiche. Ce deuxième bloc présente une liste des différents shapefiles contenus dans le fichier compressé (cf. Figure 6). Si un seul shapefile est présent, le bloc suivant s'affiche automatiquement. Sinon, l'utilisateur doit sélectionner un shapefile en cliquant dessus.

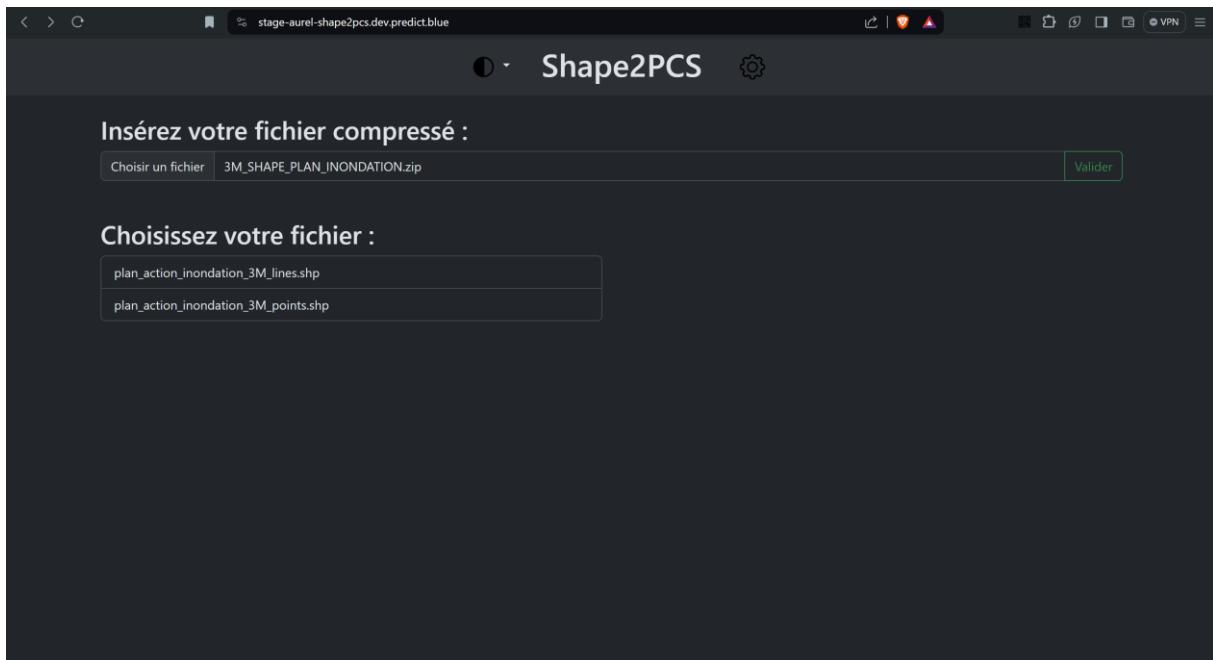


Figure 6 : Deuxième block de la page principale du projet

Le troisième bloc propose un champ de texte pour entrer le Spatial Reference Identifier (cf. Figure 7). La fonction `findEPSG` gère cette étape en envoyant le chemin du shapefile sélectionné à la fonction `findEPSG` du backend. La réponse obtenue est le code EPSG. Si la réponse est vide, une erreur s'affiche et l'utilisateur doit entrer le code manuellement. Sinon, la fonction `pushDataInDB` est appelée. Celle-ci communique avec la fonction homonyme du backend pour spécifier le fichier à utiliser et le code EPSG. Ensuite, elle déclenche les fonctions suivantes pour afficher le quatrième bloc : `receiveAndShowAttributs`, `showMap`, et `autoDetectAttribut`.

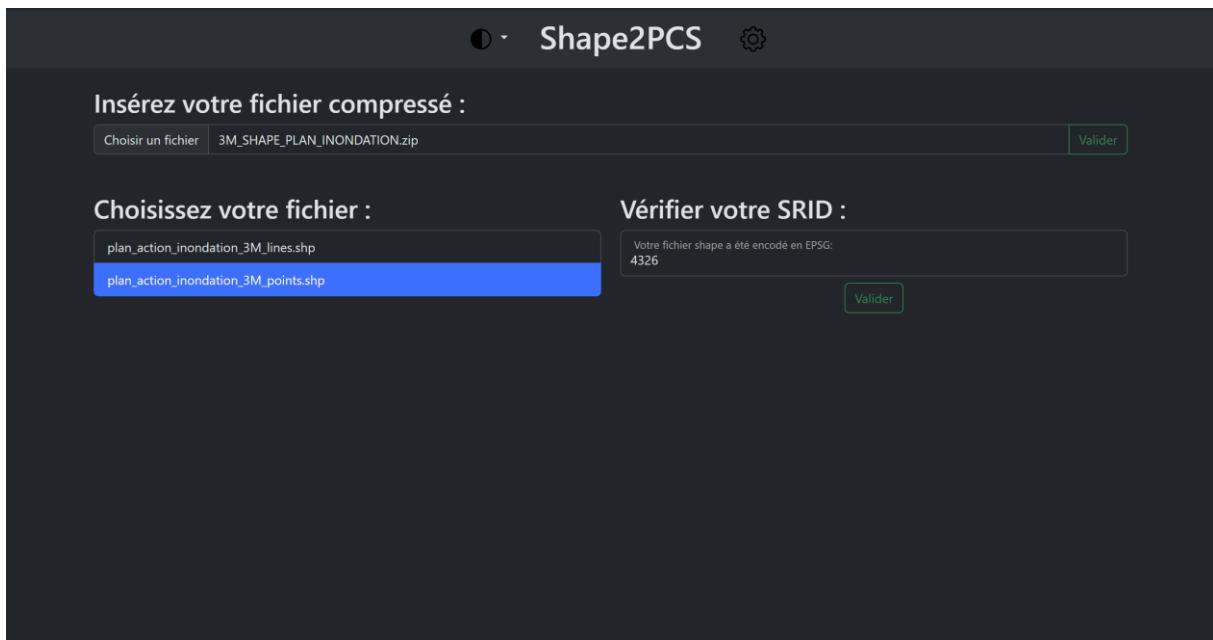


Figure 7 : Troisième block de la page principale

Le quatrième et dernier bloc est dédié à la prévisualisation (cf. Figure 8). Il se compose de deux parties distinctes : les attributs, affichés via la méthode `receiveAndShowAttributs`, et la carte, affichée par la fonction `showMap`.

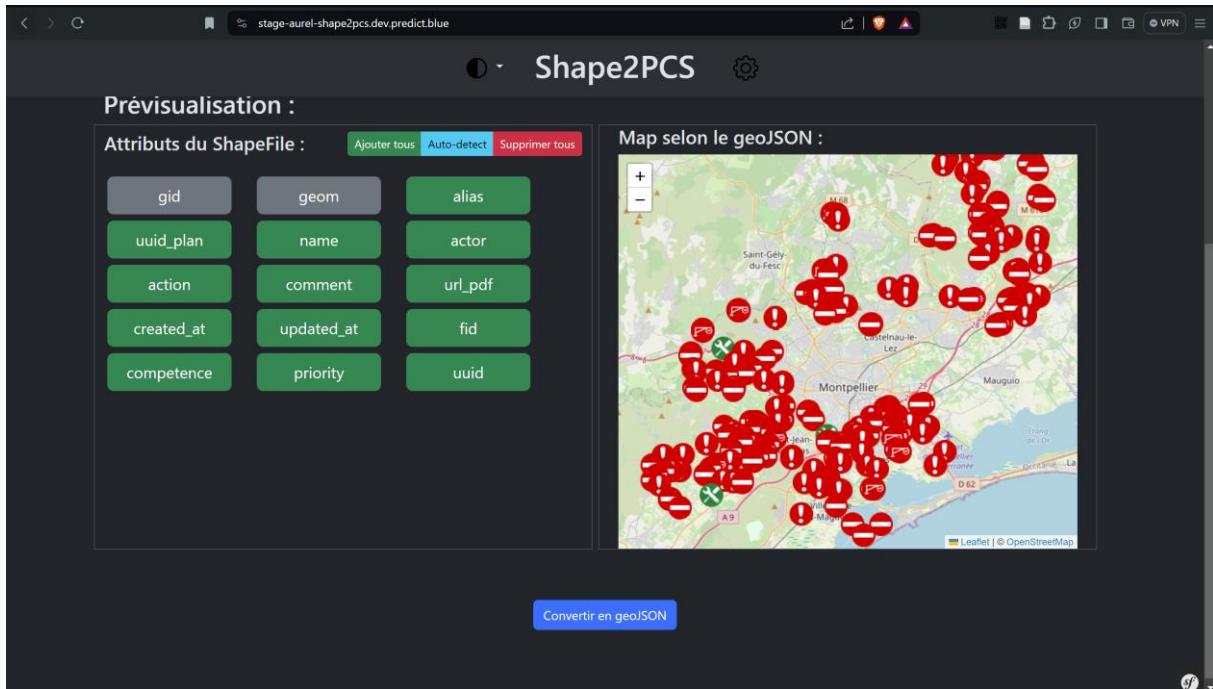


Figure 8 : Quatrième block de la page principale

Les attributs représentent les colonnes contenues dans le shapefile. Par exemple, pour un shapefile contenant des points avec les colonnes "description" et "actor", chaque point possédera une valeur pour "description" et "actor". Un clic sur un attribut ouvre une fenêtre modale remplie par la fonction `fillModal` (cf. Figure 9). Cette fenêtre affiche le nom de l'attribut avant et après l'affectation au geoJSON, un exemple des données contenues dans la colonne, et la quantité de lignes nulles.

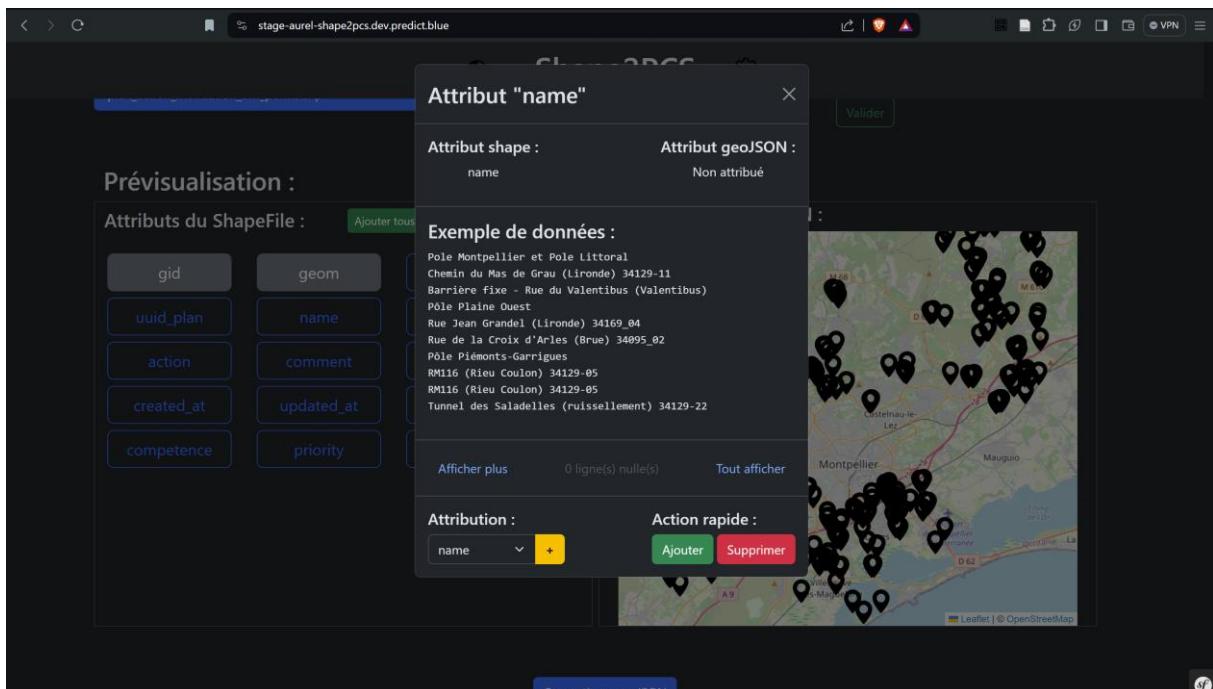


Figure 9 : Modal d'un attribut

L'objectif principal du convertisseur est de permettre la modification des noms de colonnes pour qu'ils correspondent à ceux utilisés par Poisson Soluble pour les PCS des communes. Ainsi, il est possible de renommer un attribut grâce au menu "Attribution" de la fenêtre modale (cf. Figure 10). Une liste déroulante permet de choisir parmi les différents champs de la base de données de Poisson Soluble, ces champs sont modifiables depuis les options que je détaillerai plus loin. Un clic sur le "+" à côté de la liste ajoute l'attribut du shapefile au geoJSON avec le nom sélectionné. On peut également ajouter l'attribut avec le même nom en cliquant sur le bouton "Ajouter" dans le menu "Action rapide". Enfin, il est possible de supprimer l'attribut du geoJSON en cliquant sur le bouton "Supprimer".

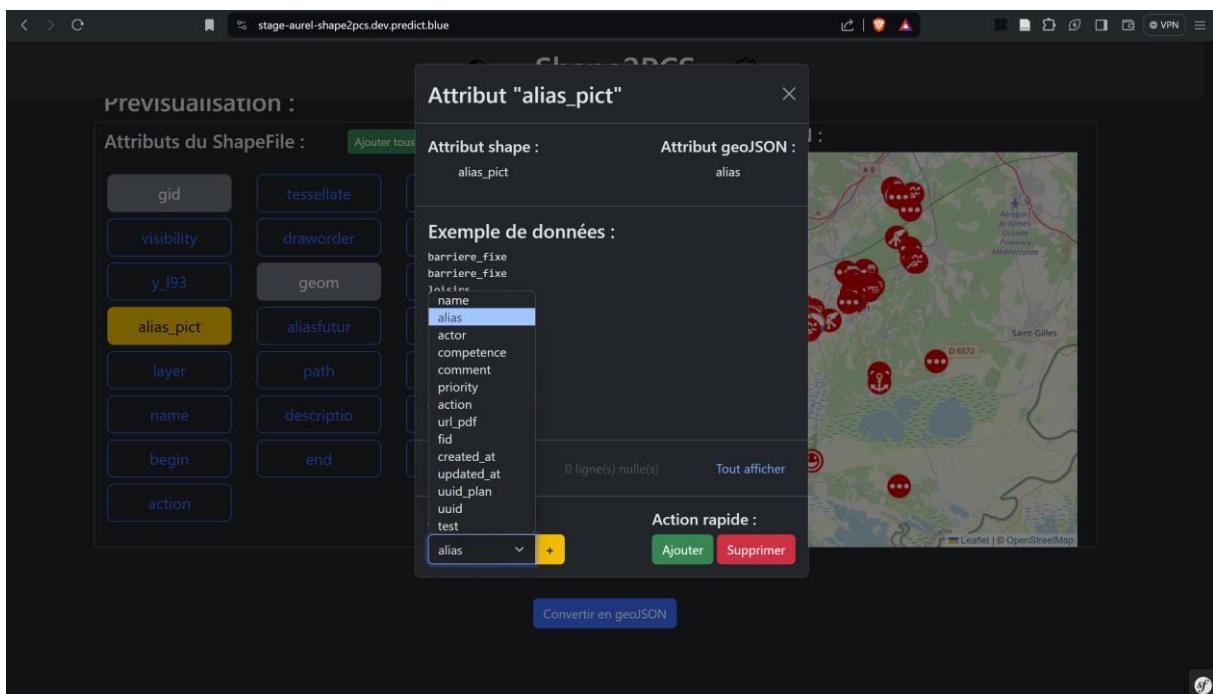


Figure 10 : Utilisation de l'attribution

Un code couleur est présent dans la liste des attributs du shapefile pour faciliter la compréhension de leur état. Le vert signifie que l'attribut est ajouté au geoJSON avec le même nom. L'orange indique que l'attribut est ajouté avec un nom différent. Le bleu avec un fond transparent signifie que l'attribut n'est pas ajouté au geoJSON. Enfin, la couleur grise indique que c'est un attribut ajouté pour la conversion du shapefile dans la base de données et donc absent du shapefile original (cf. Figure 11).

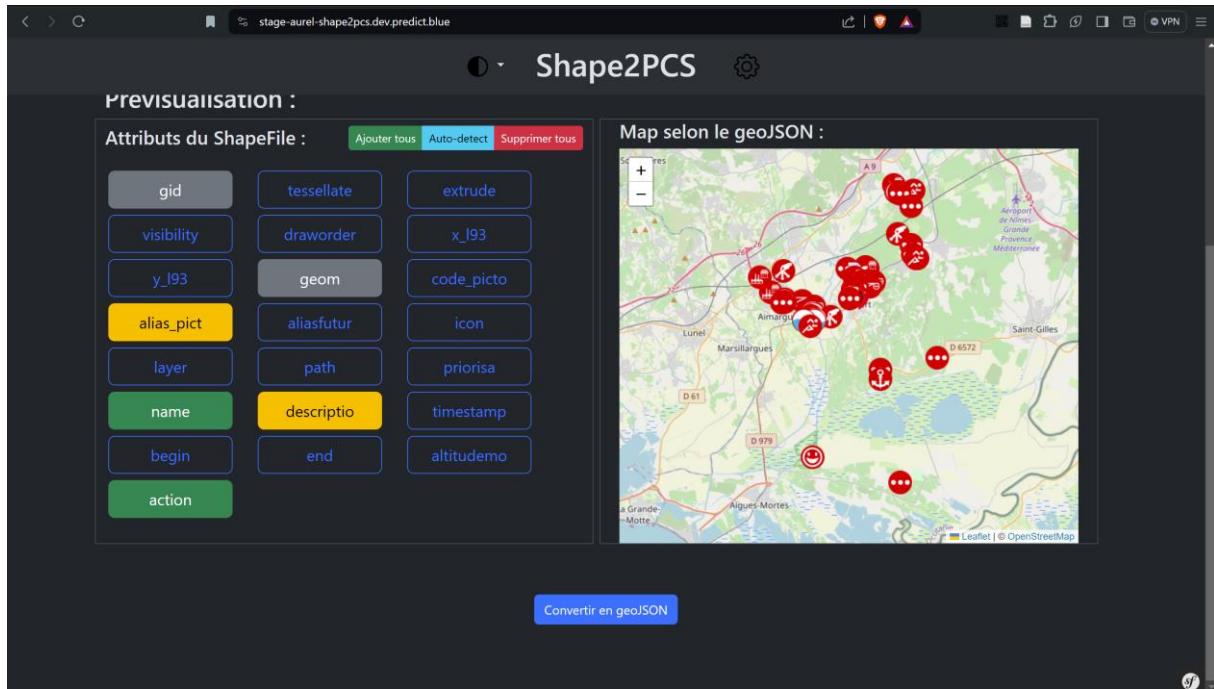


Figure 11 : Exemple du code couleur des attributs

La carte offre une prévisualisation des formes, fidèle à l'outil utilisé par les développeurs de Poisson Soluble pour le produit Predict. Elle s'adapte aux attributs ajoutés pour imiter l'interface qu'on les communes dans l'application Predict. Par exemple, pour un shapefile contenant des points, l'attribut nommé "alias" déterminera l'apparence des points sur la carte. De plus, le contenu des pop-ups dépend de plusieurs attributs, à savoir "name", "actor", "competence", "priority", "comment", et "action" (cf. Figure 12).

Map selon le geoJSON :

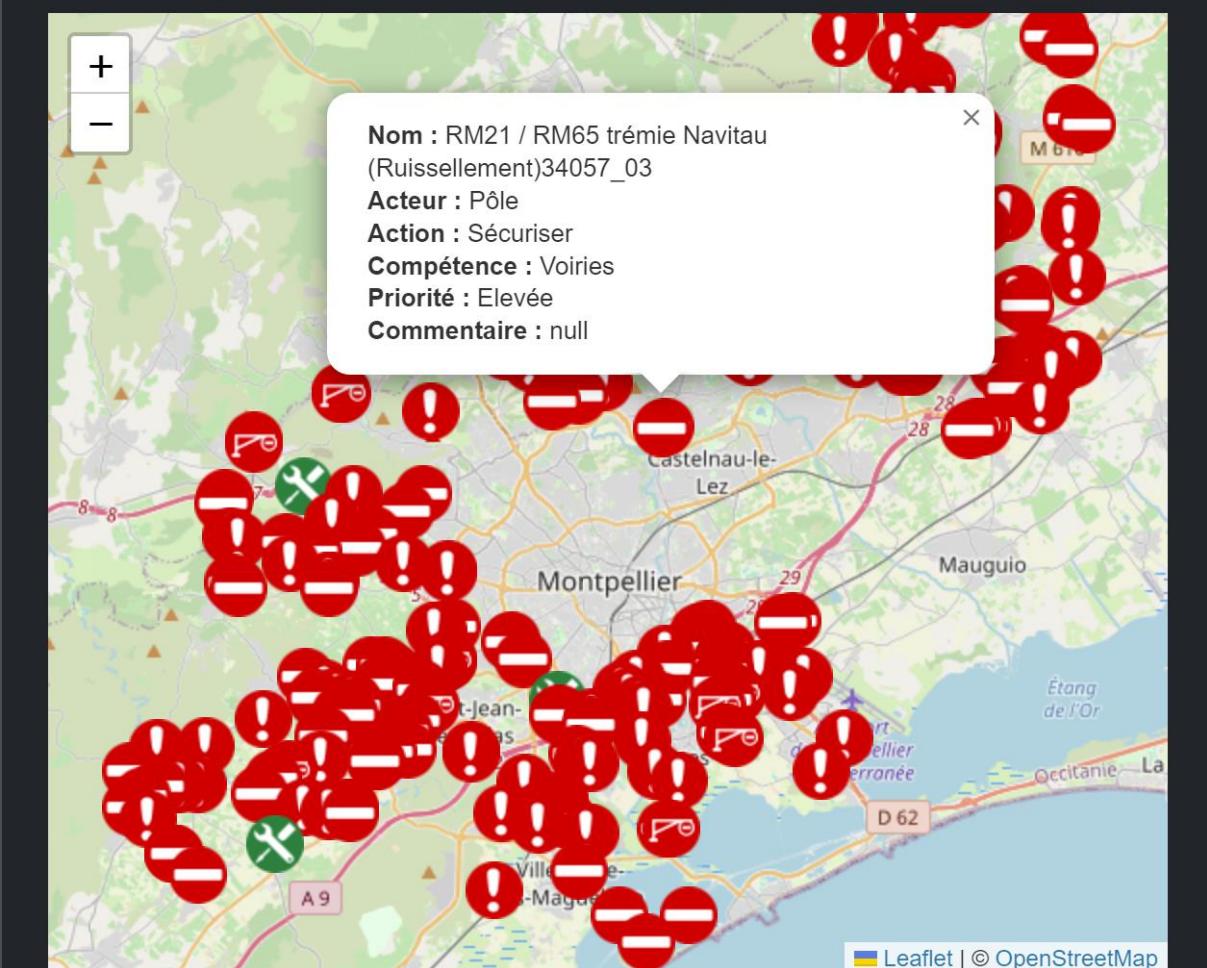


Figure 12 : Pop-ups de la map

La dernière partie de l'interface est une fenêtre modale qui s'ouvre lorsque l'on clique sur la roue crantée située à droite du titre. Ces options permettent de modifier les attributs recherchés par Poisson Soluble ainsi que les attributs ajoutés automatiquement. Des champs de texte modifiables sont disponibles pour effectuer ces modifications (cf. Figure 13). Des alertes situées au-dessus de ces champs informent l'utilisateur en temps réel de la validité de la syntaxe JSON (cf. Figure 14).

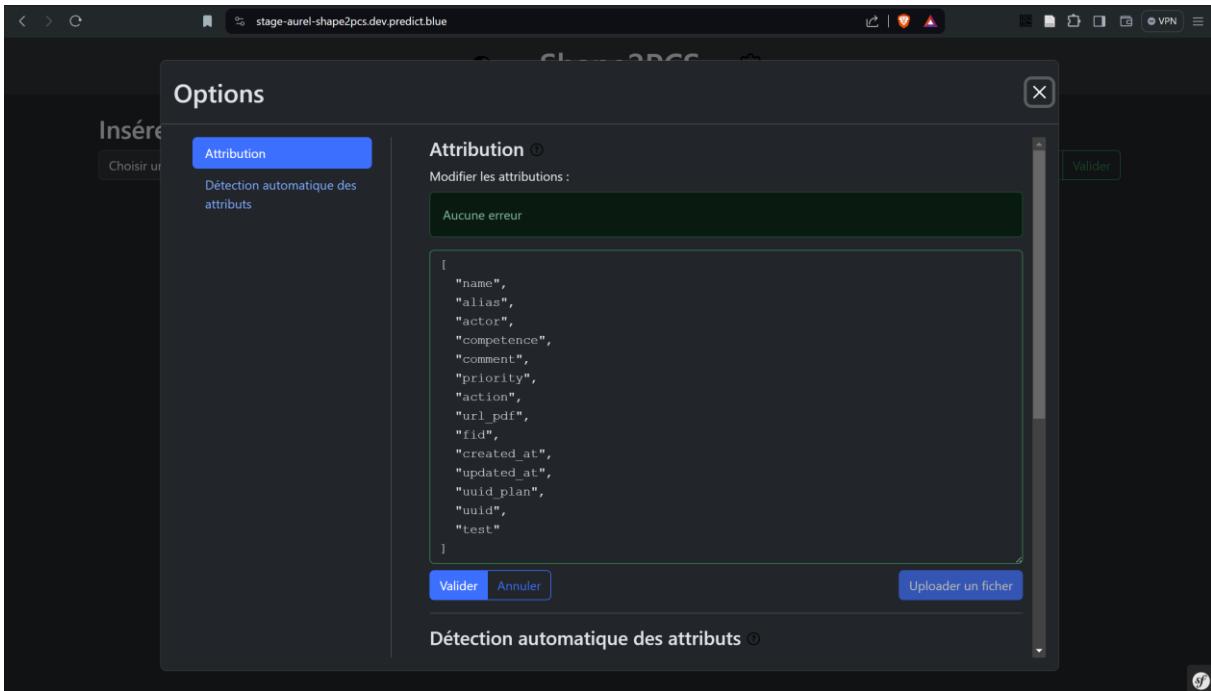


Figure 13 : Modal des options sans erreurs

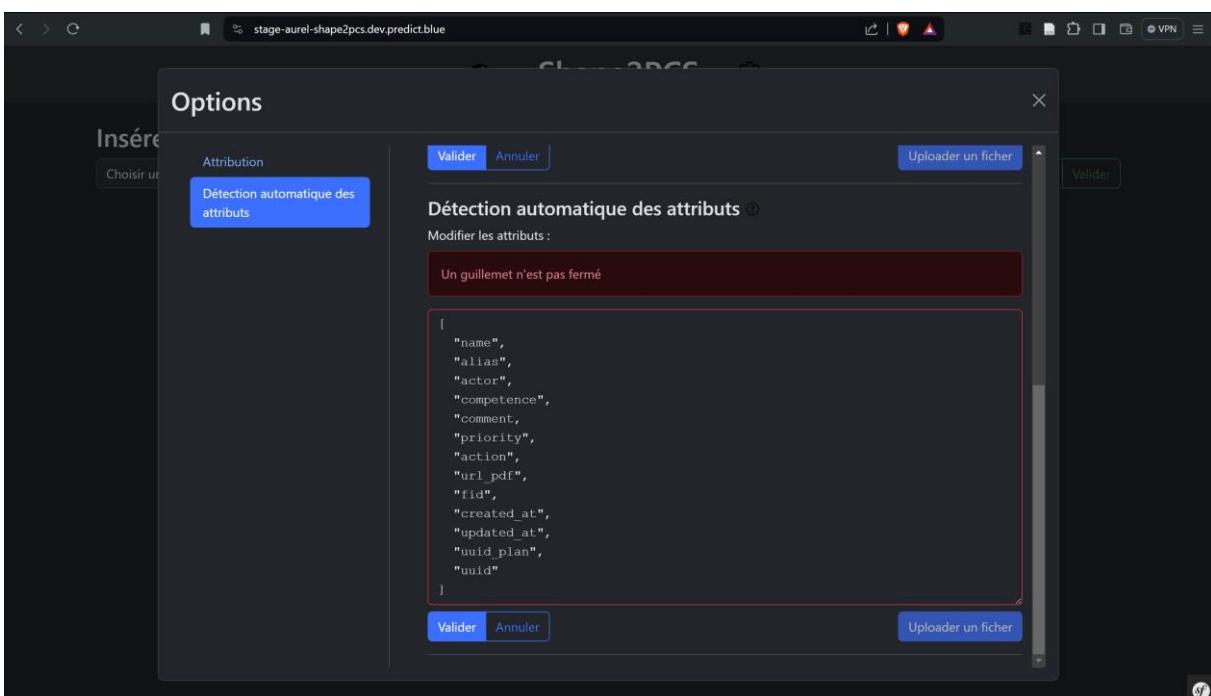


Figure 14 : Modal des options avec une erreur

Enfin, il est possible de choisir entre trois modes d'affichage : "Light", "Night" et "Automatique" (cf. Figure 15). Le mode "Light" correspond à un affichage clair, le mode

"Night" à un affichage sombre, et le mode "Automatique" s'adapte automatiquement en fonction du mode du système.

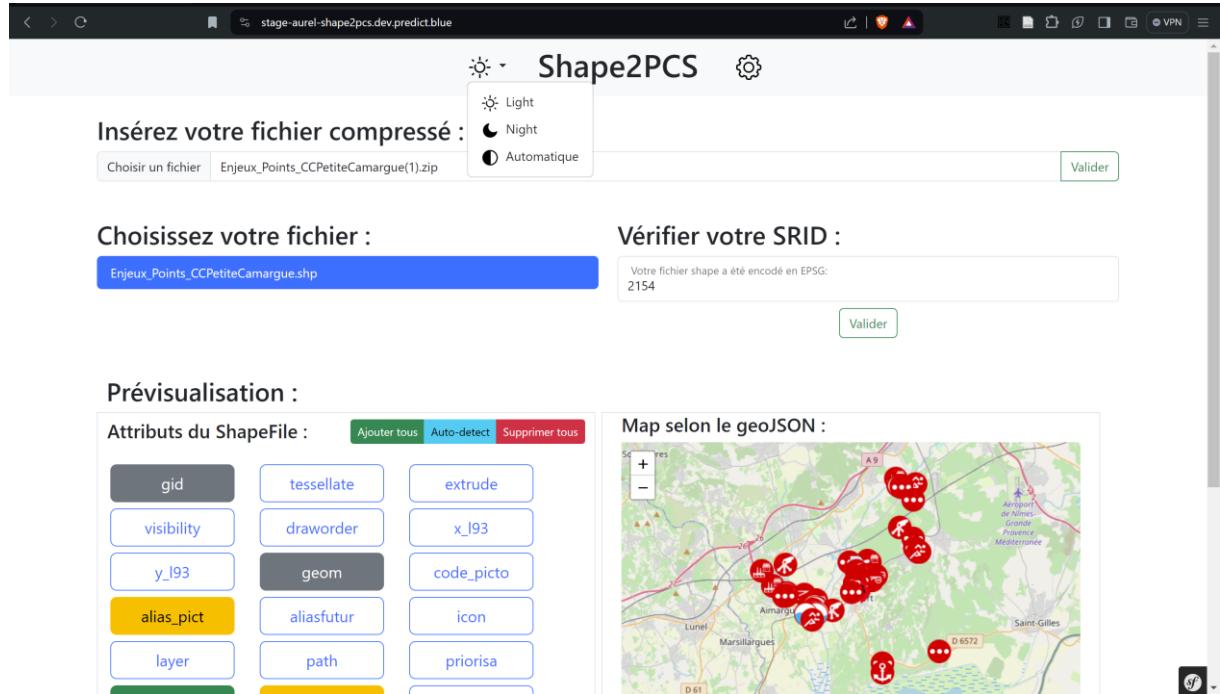


Figure 15 : Page principale en mode clair

En conclusion, cette interface permet une navigation rapide et simple au sein de l'application. Je n'ai pas abordé les nombreuses pop-ups présentes pour expliquer les éléments qui ne sont pas immédiatement intuitifs. Toutefois, la création d'une légende pour expliciter le code couleur des attributs est nécessaire et est prévue pour les dernières semaines de développement.

Réalisation

Dans cette partie, nous verrons en détail les fonctions les plus importantes, d'une part, du backend, et d'autre part, du frontend.

Réalisation du backend

Les fonctions absolument nécessaires au fonctionnement de l'application se trouvent dans le fichier « `MainController` », dont voici le « `construct` » et les attributs :

```

18 no usages ▲ Aurel +1*
19 class MainController extends AbstractController
{
20     6 usages
21     private string $path;
22     4 usages
23     private RequestStack $session;
24     7 usages
25     private ShapeRepository $shapeRepository;
26
27     no usages ▲ Aurel *
28     public function __construct(KernelInterface $kernel, RequestStack $requestStack, ShapeRepository $shapeRepository)
29     {
30         $this->path = $kernel->getProjectDir() . "/var/doc/";
31         $this->session = $requestStack;
32         $this->shapeRepository = $shapeRepository;
33     }

```

Figure 16 : MainController backend

L'attribut « path » est le chemin vers le dossier où l'application enregistre et décomprime les fichiers. « session » permet d'accéder à la session et « shapeRepository » est le modèle qui accède à la base de données.

Voici la fonction `findShapeFile` :

```

43 no usages ▲ Aurel
44 #[Route('/find_file', name: 'app_find_files')]
45 public function findShapeFile(Request $request): Response
46 {
47     exec( command: 'rm -rf ' . $this->path); //Clear all old files
48     $this->setTabAttribut([]);
49     $file = $request->files->get( key: 'shapefile');
50     if ($file) { //Test if there is a file
51         $file->move($this->path, $file->getClientOriginalName());
52         $zipFile = new ZipArchive();
53         $canOpen = $zipFile->open( filename: $this->path . $file->getClientOriginalName());
54         if ($canOpen !== true) {
55             $error = match ($canOpen) {
56                 ZipArchive::ER_NOZIP => "Le fichier n'est pas une archive ZIP. ",
57                 ZipArchive::ER_OPEN => "Impossible d'ouvrir le fichier.",
58                 ZipArchive::ER_READ => "L'archive ZIP est inconsistante.",
59                 default => "Une erreur inconnue empêche d'ouvrir le fichier",
60             };
61             return new JsonResponse([ "error" => 1, "data" => $error]);
62         } else {
63             $zipFile->extractTo($this->path);
64             $zipFile->close();
65             $tabShapeFile = $this->searchFile( pattern: $this->path . "*.*shp");
66             if (!$tabShapeFile) {
67                 return new JsonResponse([ "error" => 1, "data" => "Aucun fichier ne semble avoir l'extension .shp"]);
68             }
69             return new JsonResponse([ "error" => 0, "data" => $tabShapeFile]);
70         }
71     } else {
72         return new JsonResponse([ "error" => 1, "data" => "Pas de fichier"]);
73     }
}

```

Figure 17 : Fonction `findShapeFile` backend

Une des fonctionnalités pratiques de Symfony est la ligne « `#[Route('/find_file', name: 'app_find_files')]` » qui permet d'associer l'URL « `projet/find_file` » à l'exécution de cette fonction. Pour rappel cette fonction décomprime le fichier transmis par

le frontend et renvoie tous les liens des fichiers se terminant par `.shp`. Le plus intéressant est la manière dont l'application trouve ces fichiers, elle fait appel à la fonction `searchFile` :

```

276     /**
277      * @param $pattern string
278      * @return array
279      * Recherche récursivement dans tout les dossiers et sous dossiers qui respect le pattern $pattern
280      */
281      2 usages  ± Aurel
282      public function searchFile(string $pattern): array
283      {
284          $files = glob($pattern);
285
286          ⚡ foreach (glob(pattern: dirname($pattern) . '/*', flags: GLOB_ONLYDIR | GLOB_NOSORT) as $dir) {
287              $files = array_merge($files, $this->searchFile(pattern: $dir . '/' . basename($pattern)));
288          }
289
290      }

```

Figure 18 : Fonction `searchFile` backend

En PHP il n'existe que la fonction « `glob` » qui permet de chercher des fichiers dans dossiers, mais pas dans des sous-dossiers. C'est pourquoi j'ai utilisé une technique algorithmique adaptée pour ce problème complexe : la récursivité (AC22.02). Cette fonction utilise « `glob` » récursivement dans tous les sous-dossiers de l'archive décompressée.

La fonction `findEPSG` :

```

75      no usages  ± Aurel
76      #[Route('/find_EPSG', name: 'app_find_EPSG')]
77      public function findEPSG(Request $request): Response
78      {
79          $this->setTabAttribut([]);
80          $filePath = $request->request->get(key: 'filePath');
81          $p = Process::fromShellCommandline(command: 'ogrinfo -al -so "$shapefile"'; //Summarize shapefile properties
82          $p->run(callback: null, ['shapefile' => $filePath]);
83          if (!$p->isSuccessful()) {
84              return new JsonResponse(data: '');
85          } else {
86              $ogrinfo = $p->getOutput();
87              $code_EPSG = substr(strrchr($ogrinfo, needle: '"EPSG",'), offset: 2, length: 4);
88          }
89      }

```

Figure 19 : Fonction `findEPSG` backend

Cette fonction récupère le chemin du fichier sélectionné par l'utilisateur et recherche le code EPSG correspondant. La première ligne de la fonction réinitialise le tableau des associations entre les noms des attributs dans le shapefile et ceux dans le geoJSON. En effet, lorsque cette fonction est appelée, cela signifie que l'on cherche le code EPSG d'un nouveau fichier, et les anciennes associations ne sont donc plus nécessaires. Ce tableau est stocké dans la session et peut être accédé via la méthode `getTabAttribut` et modifié par `setTabAttribut`.

Ensuite, l'application récupère les informations du shapefile grâce à la commande `ogrinfo` avec les paramètres « `-al` » et « `-so` ». Ces paramètres permettent d'obtenir

uniquement un résumé des informations du fichier. Ogrinfo fait partie de la bibliothèque GDAL, qui contient des commandes pour le traitement de données géospatiales. J'exécute cette commande grâce au module Symfony Process et à sa méthode statique fromShellCommandLine. Cette méthode permet l'exécution de la commande en utilisant l'interpréteur du système d'exploitation sur lequel le programme s'exécute.

Grâce à Docker, l'application est encapsulée dans un environnement standard. En y important la bibliothèque GDAL et en vérifiant l'exécution correcte de la commande, je peux garantir son bon fonctionnement à chaque utilisation.

Voici un exemple de résumé que la commande ogrinfo renvoie d'un shapefile :

```
aurel@Aurel-Inspiron:~/Documents/tests/3M_SHAPE_PLAN_INONDATION (copie)/3M_SHAPE_PLAN_INONDATION$ ogrinfo -al -so plan_action_inondation_3M_points.shp
INFO: Open of 'plan_action_inondation_3M_points.shp'
      using driver 'ESRI Shapefile' successful.

Layer name: plan_action_inondation_3M_points
Metadata:
  DBF_DATE_LAST_UPDATE=2024-02-27
Geometry: Point
Feature Count: 257
Extent: (3.705560, 43.510100) - (4.046870, 43.756550)
Layer SRS WKT:
GEOGCRS["WGS 84",
  DATUM["World Geodetic System 1984",
    ELLIPSOID["WGS 84",6378137.298,257223563,
      LENGTHUNIT["metre",1]]],
  PRIMEM["Greenwich",0,
    ANGLEUNIT["degree",0.0174532925199433]],
  CS[ellipsoidal,2],
  AXIS["latitude",north,
    ORDER[1],
    ANGLEUNIT["degree",0.0174532925199433]],
  AXIS["longitude",east,
    ORDER[2],
    ANGLEUNIT["degree",0.0174532925199433]],
  ID["EPSG",4326]
Data axis to CRS axis mapping: 2,1
uuid: String (254,0)
alias: String (100,0)
uuid_plan: String (254,0)
name: String (100,0)
actor: String (100,0)
action: String (254,0)
comment: String (254,0)
url_pdf: String (254,0)
created_at: String (24,0)
updated_at: String (24,0)
fid: String (254,0)
competence: String (254,0)
priority: String (254,0)
```

Figure 20 : Exemple de résumé de ogrinfo

Le code EPSG se trouve encadré en rouge, c'est pourquoi je récupère la sortie de cette commande. J'utilise la fonction « strrchr » qui trouve la dernière occurrence d'un caractère dans une chaîne pour chercher la chaîne « "EPSG" ». Puis, findEPSG renvoie uniquement les quatre chiffres qui suivent l'espace et la virgule grâce à la fonction substr.

Voici la fonction pushDataInDB :

```

no usages ± Aurel *
94     #[Route('/push_data', name: 'app_push_data')]
95     public function pushDataInDB(Request $request, ParameterBagInterface $params): Response
96     {
97         $filePath = $request->request->get('key: filePath');
98         $codeEPSG = $request->request->get('key: codeEPSG');
99         $p = Process::fromShellCommandLine(command: 'shp2pgsql -s "$EPSG":4326 -d "$shapefile" SHAPE | psql "$database_url"');
100        $p->run(callback: null, ["EPSG" => $codeEPSG, "shapefile" => $filePath, "database_url" => $params->get("DATABASE_URL")]);
101        if (!$p->isSuccessful()) {
102            return new JsonResponse(['error' => 1, "data" => $p->getErrorOutput()]);
103        } else {
104            return new JsonResponse($p->getOutput());
105        }
106    }

```

Figure 21 : Fonction pushDataInDB

La commande `shp2pgsql` provient de la bibliothèque PostGIS, et le module `Process` est utilisé pour son exécution. Le paramètre « `-s` » permet de spécifier le SRID du fichier, ainsi que celui dans lequel on souhaite encoder les données. Nous voulons que les coordonnées géographiques de la base de données soient encodées en EPSG 4326, car le module JavaScript Leaflet, qui permet d'afficher des cartes, utilise ce SRID par défaut.

Le nom de la base de données dans laquelle les données seront envoyées est "Shape". Le paramètre « `-d` » indique de créer la base de données si elle n'existe pas. La sortie de `shp2pgsql` est une série d'instructions pour créer la table, que je redirige vers « `psql "$database_url"` », ce qui ouvre un interpréteur SQL directement dans la base de données.

Voici la fonction `geoJSON` :

```

121     #[Route('/geoJSON', name: 'app_geoJSON')]
122     public function geoJSON(): Response
123     {
124         $datas = $this->shapeRepository->getData();
125         $geoJSON = $this->getGeoJsonTemplate();
126         $features = $this->getFeatureTemplate();
127         foreach ($datas as $data) {
128             foreach ($this->getTabAttribut() as $shpAttrib => $geoJSONAttrib) {
129                 $features["properties"][$geoJSONAttrib] = $data[$shpAttrib];
130             }
131             $features["geometry"] = json_decode($data["geometry"]);
132             $geoJSON["features"][] = $features;
133         }
134         return new JsonResponse($geoJSON);
135     }

```

Figure 22 : Fonction geoJSON

Cette fonction génère le `geoJSON` en fonction des coordonnées des formes stockées dans la base de données et du tableau « `tabAttribut` ». Pour chaque forme de la base de données, tous les attributs répertoriés dans le tableau `tabAttribut` sont ajoutés au `geoJSON` avec les noms qui leur sont attribués dans le tableau. Par exemple, si `tabAttribut` contient

["alias_pict" => "alias", "description" => "comment"], alors les attributs alias_pict et description seront ajoutés au geoJSON sous les noms "alias" et "comment".

Enfin voici la fonction setAttribut :

```
1 usage  ± Aurel
137 #[Route('/setAttribut/{shapeAttribut}/{geoJSONAttribut}', name: 'app_setAttribut')]
138 public function setAttribut(string $shapeAttribut, string $geoJSONAttribut = ""): Response
139 {
140     $tab = $this->getTabAttribut();
141     //I have to unset all old key/value to not duplicate any of them
142     if (array_key_exists($shapeAttribut, $tab)) {
143         unset($tab[$shapeAttribut]);
144     }
145     if (in_array($geoJSONAttribut, $tab)) {
146         unset($tab[array_keys($tab, $geoJSONAttribut)[0]]); //I'm sure there is only one value
147     }
148     if ($geoJSONAttribut != "") { #if geoJSONAttribut = "" then we just delete the attribute
149         $tab[$shapeAttribut] = $geoJSONAttribut;
150     }
151     $this->setTabAttribut($tab);
152     return new JsonResponse($tab);
153 }
```

Figure 23 : Fonction setAttribut

Cette fonction permet d'ajouter ou de supprimer un attribut du tableau « tabAttribut », par conséquent, du geoJSON. La première ligne de code définit la route suivante : « /setAttribut/{shapeAttribut}/{geoJSONAttribut} ». Cela indique à Symfony d'injecter les attributs envoyés par la méthode GET dans les paramètres de la fonction.

Ensuite, il est nécessaire de vérifier si la clé (l'attribut du shapefile) existe déjà. Si c'est le cas, elle est supprimée. Le même test est effectué sur les valeurs. Selon la valeur de "geoJSONAttribut", l'association est ensuite ajoutée ou supprimée.

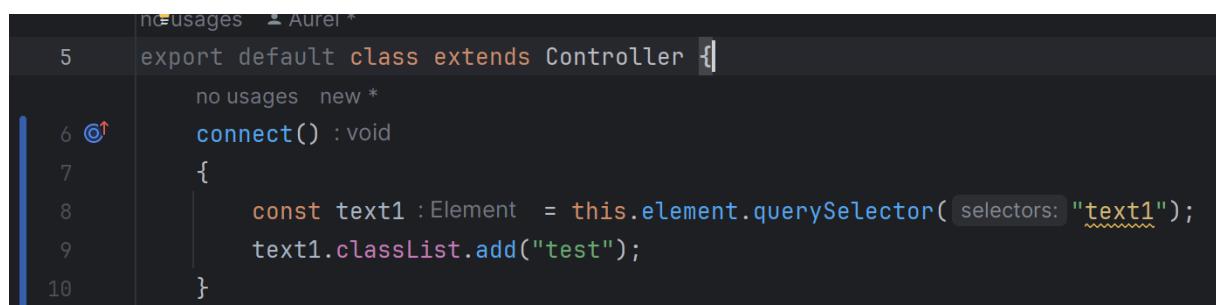
Ces fonctions sont les plus notables parmi celles du backend. Les autres sont soit similaires (ajouter tous les attributs ou supprimer tous les attributs) ou assez simples (détecter automatiquement les attributs ou renvois des Template geoJSON). Je n'ai pas non plus abordé les fonctions du repository, car elles se contentent de requête SQL simple. Les plus grosses difficultés seront abordées dans la partie « Difficulté technique ». En conclusion, le backend aura été une grosse partie de ce projet. Cette réalisation a permis de transformer les concepts théoriques en un système opérationnel efficace, prêt à répondre aux besoins spécifiques de Poisson Soluble.

Réalisation du frontend

Le frontend fonctionne en appelant diverses fonctions du backend via leurs routes définies. Il est développé avec Stimulus, un framework JavaScript doté de nombreuses fonctionnalités, et utilise Node.js pour la gestion des modules. Comme mentionné précédemment, Stimulus est employé dans ce projet pour simplifier l'intégration de JavaScript "vanilla" avec Symfony. Il permet de structurer le code en différents fichiers, chacun correspondant à une section spécifique de la page HTML. Par exemple :

```
<div data-controller="home" >  
    <p id="text1"></p>  
</div>  
<p id="text2"></p>
```

En définissant l'attribut « `data-controller` » à « `home` », Stimulus associera le fichier `home_controller.js` situé dans le dossier « `/assets/controller/` » à cette div. Cela permet d'exécuter ce type de code :



The screenshot shows a code editor with a dark theme. A sidebar on the left lists file usages for 'Aureel'. The main code area has line numbers from 5 to 10. Line 5 starts with 'export default class extends Controller {'. Line 6 contains a call to 'connect()' with a blue arrow pointing to it. Lines 7 through 10 define the 'connect' method, which uses 'this.element.querySelector' to find an element with the selector 'text1' and adds a 'test' class to it.

```
no usages ▾ Aureel *  
5  export default class extends Controller {  
6      no usages new *  
6  ⌂ connect(): void  
7  {  
8      const text1: Element = this.element.querySelector(selectors: "text1");  
9      text1.classList.add("test");  
10 }
```

La méthode « `connect` » est appelée lorsque le fichier est initialisé. La div peut être récupérée en utilisant « `this.element` ».

Dans ce projet, nous utilisons trois contrôleurs Stimulus : un pour la page principale, un pour le header et un autre pour les options. Celui que nous allons détailler dans cette section est le « `main_controller` », car il contient les fonctionnalités principales. Bootstrap est également utilisé principalement pour styliser l'application. Comme dans la section « Réalisation du backend », nous nous concentrerons sur les fonctions les plus intéressantes.

Voici la fonction `findFile` :

```

29 Show usages  ▾ Aurel
30     async findFile() : Promise<void>
31     {
32         const input : Element = this.element.querySelector( selectors: "#inputFile");
33         const formData : FormData = new FormData();
34         formData.append( name: 'shapefile', input.files[0]);
35
36         const data = await(await fetch( input: "/find_file", init: {
37             method: "POST",
38             body: formData
39         }).json());
40         if (data["error"] === 0) { //No errors
41             this.hideAllDiv();
42             const list_group : Element = this.element.querySelector( selectors: "#list_shp_file");
43             this.clearElements(list_group);
44             if (data["data"].length === 1) { //If there is only one shapefile we use it automatically
45                 let button : HTMLButtonElement = document.createElement( tagName: "button");
46                 button.classList.add("list-group-item", "list-group-item-action", "active");
47                 button.textContent = data["data"][0].substr(data["data"][0].lastIndexOf("/") + 1);
48                 list_group.appendChild(button);
49                 await this.findEPSG(data["data"][0]);
50             } else {
51                 for (let i : number = 0; i < data["data"].length; i++) { //Otherwise we ask the user to choose
52                     let button : HTMLButtonElement = document.createElement( tagName: "button");
53                     button.classList.add("list-group-item", "list-group-item-action");
54                     button.textContent = data["data"][i].substr(data["data"][i].lastIndexOf("/") + 1);
55                     button.addEventListener( type: "click", listener () : void => {
56                         this.findEPSG(data["data"][i]);
57                         this.toggleActiveButton(button, list_group);
58                         this.element.querySelector( selectors: "#previ").classList.add("invisible");
59                         this.element.querySelector( selectors: "#convert_button").classList.add("invisible");
60                     })
61                     list_group.appendChild(button);
62                 }
63             }
64             this.element.querySelector( selectors: "#choice_shp_file").classList.remove( tokens: "invisible");
65         } else {
66             this.addAlert(data["data"]);
67         }
68     }

```

Figure 24 : Fonction `findFile` frontend

La majorité des fonctions de l'application seront précédées du mot-clé « `async` », indiquant qu'elles peuvent être appelées de manière asynchrone. Cela signifie que la fonction contient des requêtes au backend, pour lesquelles la réponse est attendue avec le préfixe « `await` ». De plus, la fonction elle-même peut être mise en attente lors de son appel en utilisant le même préfixe.

La fonction `findFile` récupère le fichier de l'utilisateur et l'envoie au backend avec la méthode POST. Selon la réponse du backend, le deuxième bloc de l'interface, qui est la liste des fichiers shape, est affiché. Le « `if` » à la ligne 43 permet d'afficher ce deuxième bloc directement s'il n'y a qu'un seul fichier. La ligne 46 affiche uniquement le nom du fichier, plutôt que tout le chemin.

Il y a une légère duplication de code entre les lignes 44 à 48 et 51 à 60. Cela s'explique par le fait que, dans le premier cas, nous voulons créer un bouton pour représenter notre seul fichier et afficher directement le troisième bloc. De plus, ce bouton ne doit rien faire lors du clic afin d'empêcher l'utilisateur de recharger tous les autres blocs. Dans le deuxième cas, nous devons

créer plusieurs boutons qui chargent les blocs suivants en fonction du shapefile qu'ils représentent. Il est plus simple d'inclure le code du premier cas dans une boucle, ce qui entraîne cette légère duplication.

Voici la fonction `findEPSG` :

```
70 Show usages  ▲ Aurel
71
72     async findEPSG(filePath) :Promise<void>
73     {
74         this.clearAlert();
75         const data = await(await fetch( input: "/find_EPSG", init: {
76             method: "POST",
77             body: new URLSearchParams( init: {
78                 'filePath': filePath
79             })
80         }).json());
81         const inputEPSG :Element = this.element.querySelector( selectors: '#EPSG');
82         const div_EPSG :Element = this.element.querySelector( selectors: '#choice_EPSG');
83         let button_EPSG :Element = this.element.querySelector( selectors: "#button_EPSG");
84         button_EPSG.removeAllEventListeners(button_EPSG);
85         button_EPSG.addEventListener( type: "click", callback: () :void => {
86             this.pushDataInDb(filePath, inputEPSG.value)
87         });
88         inputEPSG.value = data;
89         if (data != "") {
90             await this.pushDataInDb(filePath, data);
91         } else {
92             this.addAlert("Aucun code EPSG détecté.Si vous ne le connaissez pas, il est visible sur des logiciels comme QGIS");
93         }
94         div_EPSG.classList.remove( tokens: "invisible");
95     }
96 }
```

Figure 25 : Fonction `findEPSG` frontend

La fonction « `async findEPSG(filePath)` » envoie une requête POST asynchrone à l'endpoint `"/find_EPSG"` avec un paramètre `filePath`, et attend la réponse JSON. Cette réponse contient le code EPSG qui est ensuite inséré dans un champ de saisie HTML. Si un code EPSG est détecté, on appelle la fonction `pushDataINDB`, sinon, un message d'alerte est affiché pour informer l'utilisateur de l'absence de code EPSG.

En plus, la fonction prépare un bouton pour permettre à l'utilisateur de rentrer manuellement le code EPSG au cas où l'application l'aurait mal détecté. Avant d'ajouter un événement click à ce bouton, tous les anciens écouteurs d'événements sont supprimés pour éviter les doublons.

Voici la fonction `pushDataInDB` :

```

96     Show usages ▾ Aurel
97     ▾ async pushDataInDb(file, EPSG) :Promise<void>
98     {
99         await fetch( input: "/push_data", init: {
100            method: "POST",
101            body: new URLSearchParams( init: {
102                'filePath': file,
103                'codeEPSG': EPSG
104            })
105        })
106        const div_previ :Element = this.element.querySelector( selectors: "#previ");
107        await this.receiveAndShowAttributs();
108        await this.showMap();
109        await this.autoDetectAttributs();
110        this.element.querySelector( selectors: "#convert_button").classList.remove( tokens: "invisible");
111        div_previ.classList.remove( tokens: "invisible");

```

Figure 26 : Fonction `pushDataInDb` frontend

Cette fonction est assez explicite, elle appelle toutes les autres fonctions nécessaires pour afficher le quatrième block.

Voici la fonction `appendDataExample` :

```

321     Show usages ▾ Aurel
322     async appendDataExample(divDataExample, attribut, limit) :Promise<void>
323     {
324         let offset :number = Number(divDataExample.dataset.offset);
325         const dataAttribut = await(await fetch( input: "/getDataAttribute/" + attribut + "/" + offset + "/" + limit)).json();
326         if (dataAttribut.length !== 0) {
327             dataAttribut.forEach(data => {
328                 divDataExample.textContent += data[attribut] + "\n";
329             })
330             divDataExample.dataset.offset = String( value: offset + limit);
331             if (dataAttribut.length < limit) {
332                 this.element.querySelector( selectors: "#btnShowMore").classList.add("invisible");
333                 this.element.querySelector( selectors: "#btnShowAll").classList.add("invisible");
334             }
335         } else {
336             this.element.querySelector( selectors: "#btnShowMore").classList.add("invisible");
337             this.element.querySelector( selectors: "#btnShowAll").classList.add("invisible");
338             divDataExample.textContent += "Aucune données";
339         }

```

Figure 27 : Fonction `appendDataExample` frontend

Cette fonction gère la fonctionnalité d'affichages des données exemples dans le modal des attributs (cf. Figure 9 : Modal d'un attribut). Elle ajoute dix lignes de données à chaque clic sur le bouton "Afficher plus". Elle utilise l'attribut « `offset` » stocké dans la div pour déterminer le nombre de lignes déjà affichées. Le paramètre « `limit` » spécifie le nombre de lignes supplémentaires à afficher. Enfin, lorsque l'utilisateur clique sur "Tout afficher", la fonction est appelée avec « `limit` » fixé à un million, ce qui devrait afficher toutes les lignes disponibles.

Enfin voici la dernière fonction intéressante :

```

119 | Show usages ▾ Aurel
120 | checkSyntax(string) :{error: number, message: string}
121 | {
122 |     let bracketClose : boolean = true;
123 |     let quoteClose : boolean = true;
124 |     let quoteType : string = "";
125 |     for (let i : number = 0; i < string.length; i++) {
126 |         let char = string[i];
127 |         if (char === '"' || char === "'") {
128 |             if (quoteType === "") {
129 |                 quoteType = char;
130 |             }
131 |             if (char !== quoteType) {
132 |                 return {"error": 1, "message": "N'utilisez que \" ou ' mais pas les deux"};
133 |             }
134 |             quoteClose = !quoteClose;
135 |
136 |             if (char === "[") {
137 |                 bracketClose = false;
138 |             }
139 |             if (char === "]") {
140 |                 if (!bracketClose) {
141 |                     bracketClose = true;
142 |                 } else {
143 |                     return {"error": 1, "message": "Vous fermez un crochet sans l'ouvrir avant"};
144 |                 }
145 |             }
146 |             if (!bracketClose) {
147 |                 return {"error": 1, "message": "Un crochet est fermé sans avoir été ouvert"};
148 |             }
149 |
150 |             if (!quoteClose) {
151 |                 return {"error": 1, "message": "Un guillemet n'est pas fermé"};
152 |             }
153 |
154 |         }
155 |     }
156 |

```

Figure 28 : Fonction checkSyntax frontend

Cette fonction vérifie que les zones de texte des options sont correctement formatées (cf. Figure 13 : Modal des options sans erreurs et Figure 14 : Modal des options avec une erreur). Elle aide l'utilisateur à détecter rapidement des erreurs simples, telles que l'utilisation incorrecte de guillemets ou l'oubli de les fermer, ainsi que l'ouverture ou la fermeture incorrecte de crochets. Pour ce faire, elle utilise des booléens pour suivre l'état d'ouverture et de fermeture.

Les autres fonctions du frontend suivent un schéma similaire : elles appellent le backend et modifient les classes Bootstrap en conséquence. La réalisation de cette interface m'a permis d'acquérir des compétences avancées en JavaScript, en Bootstrap et en gestion de la communication entre le backend et le frontend. En conclusion, je suis satisfait de cette interface. Elle répond au cahier des charges et a obtenu l'approbation de mon tuteur en entreprise.

Validation

La validation de l'application de conversion de fichiers shapefile en geoJSON a été une étape essentielle pour garantir son bon fonctionnement et sa fiabilité. Pour ce faire, j'ai eu à ma disposition deux fichiers shapefile que Poisson Soluble a reçu de communes. Ces fichiers m'ont permis de tester l'application dans des conditions réelles, en simulant le processus de conversion que les utilisateurs finaux effectueront.

Le premier test consistait à vérifier l'intégrité des fichiers shapefile après leur décompression et leur importation dans la base de données PostgreSQL. J'ai utilisé la commande `shape2pgsql` pour importer les données et vérifié que toutes les entités géométriques et leurs attributs étaient correctement transférés. Pour cela, j'ai comparé les données originales avec celles importées dans la base de données, en m'assurant que les attributs étaient bien mappés et que les géométries étaient précises.

Ensuite, j'ai testé la fonctionnalité de conversion des fichiers shapefile en geoJSON. J'ai utilisé les deux fichiers shapefile pour vérifier que l'application pouvait convertir correctement les données géométriques et les attributs en geoJSON. J'ai particulièrement porté attention à la fonctionnalité `autoDetectAttributs`, qui devait identifier et mapper les attributs automatiquement. J'ai également vérifié que la fonction `setAttribut` permettait de renommer les attributs conformément aux conventions de Poisson Soluble. Chaque geoJSON généré a été inspecté pour s'assurer que les données étaient correctes et utilisables dans les applications de Predict.

Enfin, bien que je n'aie pas créé de fichiers de test automatisés en raison de la taille modeste de l'application, j'ai effectué des tests manuels exhaustifs pour couvrir un large éventail de scénarios possibles. Ces tests comprenaient la gestion des erreurs, comme les fichiers shapefile manquants ou corrompus, ainsi que la vérification de la robustesse de l'application lors de l'importation de grandes quantités de données. J'ai également testé l'interface utilisateur pour m'assurer qu'elle était intuitive et réactive, et que toutes les fonctionnalités étaient accessibles et fonctionnelles.

En conclusion, les tests réalisés avec les fichiers shapefile fournis ont démontré que l'application est capable de gérer efficacement la conversion des données géospatiales en geoJSON, tout en assurant une intégration sans faille dans les systèmes existants de Predict. La validation manuelle approfondie a permis de confirmer la fiabilité et la précision de l'application, offrant ainsi une solution robuste et performante pour les besoins de Poisson Soluble.

Difficulté technique

Le développement de l'application de conversion de fichiers shapefile en geoJSON n'a pas été exempt de défis. Plusieurs difficultés techniques ont été rencontrées au cours du projet, nécessitant des ajustements et des solutions innovantes pour garantir le bon fonctionnement de l'application. Cette section détaille les principaux obstacles rencontrés et les approches adoptées pour les surmonter. Les sous-sections suivantes aborderont les défis spécifiques liés à l'intégration des shapefiles contenant des lignes, les changements d'approche de développement, le dédoublement des écouteurs d'événements, et les complications associées aux SRID et à la commande `ogrinfo`.

Intégration des shapefiles contenant des lignes

Les shapefiles ne peuvent contenir qu'un seul type de forme. Jusqu'à la moitié de mon stage, je travaillais uniquement avec des shapefiles contenant des points pour faciliter le développement initial. Cependant, lorsque j'ai voulu intégrer des lignes dans mon application, j'ai rencontré un problème. Pour récupérer des points avec PostGIS, on utilise les fonctions SQL ST_X() et ST_Y().

Les lignes étant des tableaux de points, cela ne fonctionnait pas. J'ai donc tenté de développer une fonction alternative pour identifier les lignes dans la table et reformater complètement les données afin qu'elles fonctionnent avec Leaflet, le module JavaScript qui affiche la carte. Malheureusement, cela n'a jamais fonctionné. J'ai dû trouver une autre méthode pour manipuler des données hétérogènes (AC24.04).

Après avoir parcouru une grande partie de la documentation de PostGIS, j'ai trouvé une fonction SQL qui convertit n'importe quel type de forme en coordonnées GeoJSON. Cela m'a permis de conserver une seule fonction pour récupérer les données de la base de données (cf. Annexe 1. Fonction getData du ShapeRepository)

Changement d'approche de développement

Dans la fonction `findShapeFile`, il est nécessaire de récupérer le fichier shape, de le déplacer dans un dossier et de le décompresser. Initialement, ces actions étaient réalisées à l'aide de commandes Linux exécutées de la même manière que `shp2pgsql` et `ogrinfo`. Cependant, cette approche ne permettait pas une bonne gestion des erreurs.

Ce constat m'a été révélé lorsque j'ai consulté la documentation de PHP et découvert qu'il existe des fonctions déjà intégrées à PHP pour décompresser et déplacer des fichiers. En utilisant ces fonctions natives, nous pouvons améliorer la gestion des erreurs et rendre le code plus robuste et maintenable.

Ecouteurs d'évènement dédoublé

Lors du développement de l'interface, j'ai dû ajouter des écouteurs d'événements à des boutons déjà présents sur la page initiale. Cependant, ces écouteurs étaient ajoutés lors de certaines actions effectuées sur la page. Lorsque l'utilisateur répétait ces actions plusieurs fois, les écouteurs s'ajoutaient en plus de ceux déjà existants. En conséquence, le code des écouteurs d'événements s'exécutait autant de fois qu'il y avait d'écouteurs, entraînant des pertes de performances notables. La meilleure solution que j'ai trouvée a été de coder la fonction suivante :

```

438 /**
439 * Clone un élément et le renvoie après avoir supprimé tous ses écouteurs d'événements.
440 * @param element
441 * @returns {Node | ActiveX.IXMLDOMNode}
442 */
443 Show usages ↗ Aurel
444 removeAllEventListeners(element) :Node | IXMLDOMNode
445 {
446     const clonedElement :Node | IXMLDOMNode = element.cloneNode( deep: true );
447     element.parentNode.replaceChild(clonedElement, element);
448     return clonedElement;
449 }

```

Figure 29 : Fonction `removeAllEventListeners` frontend

La fonction « `removeAllListeners` » récupère un élément, en fait une copie, remplace l'original dans le DOM par cette copie et revoit cette copie. En spécifiant « `true` » pour l'attribut « `deep` » de « `cloneNode` », on s'assure que tous les enfants de l'élément sont également clonés. La particularité de ce clone est qu'il ne conserve pas les écouteurs d'événements, ce qui permet d'obtenir un élément sans les anciens écouteurs. Cette approche garantit une performance optimale en évitant les multiples exécutions des écouteurs d'événements.

Les SRID et la commande Ogrinfo

L'un des défis majeurs de mon projet a été la compréhension et l'utilisation des SRID pour les données géospatiales. Ce domaine, nouveau pour moi, nécessitait une plongée dans la cartographie et les concepts liés aux données géographiques. Heureusement, j'ai rapidement découvert la commande `ogrinfo` qui m'a permis d'extraire les SRID des fichiers shape. En intégrant cette commande dans mon code backend, j'ai pu résoudre ce problème crucial.

D'une part, ce défi s'inscrit directement dans l'AC22.01 : "Choisir des structures de données complexes adaptées au problème". En effet, la maîtrise des SRID impliquait la compréhension de structures de données géospatiales spécifiques, telles que les systèmes de référence et les projections cartographiques.

D'autre part, l'utilisation de la commande `ogrinfo` relève de l'AC22.02 : "Utiliser des techniques algorithmiques adaptées pour des problèmes complexes". En effet, cette commande implémente des algorithmes de traitement de données géospatiales pour extraire des informations cruciales des fichiers shape.

Méthodologie et organisation du projet

L'organisation du travail dans l'entreprise

Chez Poisson Soluble, l'organisation du travail repose sur des méthodes classiques de gestion de projet. Predict, le client principal, dépose en continu des missions sur un tableau Trello. Ces missions peuvent inclure des bugs à corriger, des fonctionnalités à développer ou des améliorations à apporter. Le chef de projet de Predict évalue chaque mission, demandant des précisions supplémentaires si nécessaire, ou sondant les développeurs pour estimer le temps requis pour accomplir chaque tâche. Les missions sélectionnées sont ensuite attribuées aux développeurs en fonction de leurs compétences et de leur charge de travail actuelle.

Les développeurs travaillent dans un environnement de développement dédié où ils peuvent tester leurs modifications en toute sécurité. Une fois qu'une mission est terminée, le code est déplacé vers un environnement de recette, où le chef de projet effectue une validation. Si le travail est jugé satisfaisant, il est mis en production et devient accessible à Predict. Dans le cas contraire, le développeur doit retravailler la mission en fonction des retours du chef de projet et la soumettre à nouveau pour validation.

Chaque matin à 10h, toute l'équipe de Poisson Soluble se réunit sur Discord pour un point quotidien. Durant cette réunion, chaque développeur partage ce qu'il a accompli la veille et ce qu'il prévoit de faire dans la journée. Cette pratique permet de maintenir une communication fluide et une coordination efficace au sein de l'équipe. Elle assure également que chacun est au courant de l'état d'avancement des différents projets et missions, favorisant ainsi une collaboration harmonieuse et proactive. En tant que stagiaire, j'ai participé à ces réunions, ce qui m'a permis de rester aligné avec les objectifs de l'équipe et de m'intégrer pleinement dans le flux de travail quotidien.

Méthodes de Développement et de Travail

Au début de mon stage, mon tuteur m'a préparé un Google Document (cf. Annexe 2. Google document de suivi de projet) contenant des étapes détaillées pour prendre en main Symfony, puis pour débuter le projet. Cette ressource a structuré mon apprentissage et assuré une compréhension solide du Framework avant de commencer le développement de l'application. Les premiers jours, je faisais un point quotidien avec mon tuteur pour discuter de mes progrès, poser des questions et recevoir des conseils sur les prochaines étapes. Cette interaction régulière a été cruciale pour établir une base solide et clarifier toute ambiguïté initiale. Cela a permis de respecter les enjeux de cohérence et de qualité du projet (CE4.02, CE4.05).

Une fois familiarisé avec Symfony, j'ai commencé à développer l'application fonctionnalité par fonctionnalité. Au fur et à mesure de l'avancement du projet, les points de suivi avec mon tuteur sont devenus hebdomadaires (cf. Annexe 5. Fichier de questions à poser

avant un point). Ces réunions hebdomadaires permettaient de faire le point sur les fonctionnalités développées, d'identifier les éventuels problèmes rencontrés et de planifier les prochaines étapes. Travailleur de manière incrémentale m'a permis de rester concentré sur des objectifs spécifiques et de progresser de manière régulière et organisée. J'ai adopté une démarche proactive, créative et critique pour améliorer constamment le projet (CE5.04).

J'ai travaillé seul sur ce projet, ce qui m'a donné une grande autonomie. J'ai utilisé Git pour versionner mon code, en hébergeant le projet dans l'organisation GitHub "Predict Service". Étant le seul développeur sur ce projet, j'avais la liberté de pousser directement sur la branche principale sans avoir besoin de processus de Merge Request. Cette configuration a simplifié le workflow et m'a permis de travailler efficacement sans attendre des validations externes. J'ai documenté régulièrement mes progrès et maintenu un historique clair des modifications via les commits Git (cf. Annexe 3. Readme de l'application et Annexe 4. Exemple de l'historique des commit), assurant ainsi la sécurité des données et leur intégrité (AC24.02).

Bien que j'aie travaillé seul sur ce projet, j'ai veillé à m'intégrer pleinement dans l'équipe de Poisson Soluble. J'ai pris l'initiative d'aller voir un par un les membres de l'équipe pour mieux comprendre leur métier et les défis spécifiques qu'ils rencontrent au quotidien (cf. Annexe 6. Notes Notion de Stage). Ces échanges m'ont permis d'adapter certaines fonctionnalités de l'application à leurs besoins réels. De plus, j'ai discuté avec le graphiste pour améliorer l'ergonomie de mon application, en veillant à ce que l'interface utilisateur soit intuitive. Enfin, j'ai réalisé une présentation de mon application à toute l'équipe, en mettant l'accent sur les développeurs qui l'utiliseront. Cela m'a permis de recueillir leurs retours et de faire les ajustements nécessaires pour optimiser l'outil en fonction de leurs attentes et de leurs pratiques quotidiennes.

Tout au long de mon stage, je n'ai pas effectué de tâches en dehors de mon projet principal. Cette concentration exclusive m'a permis de rester focalisé et de garantir la qualité et la complétude du développement de l'application. Mon approche méthodique et organisée, combinée à une communication régulière avec mon tuteur, a été déterminante pour mener à bien ce projet de manière autonome et proactive. J'ai pris l'initiative de rechercher des informations complémentaires lorsque nécessaire (cf. Annexe 6. Notes Notion de Stage) et de proposer des solutions aux problèmes rencontrés, ce qui m'a permis de développer des compétences précieuses en résolution de problèmes et en auto-apprentissage (AC25.01, AC25.04).

Conclusion

Au terme de ce projet, j'ai acquis des compétences significatives et renforcé mes connaissances en développement informatique, en lien avec les référentiels de compétences du BUT Informatique.

Tout d'abord, j'ai appliqué des principes d'accessibilité et d'ergonomie (AC21.02). En concevant l'interface utilisateur de l'application, j'ai veillé à ce qu'elle soit intuitive et facile à utiliser pour les développeurs internes de Poisson Soluble. J'ai utilisé des outils comme Bootstrap et Leaflet pour garantir une expérience utilisateur optimale, tout en respectant les standards ergonomiques actuels.

De plus, j'ai adopté de bonnes pratiques de conception et de programmation (AC21.03). En utilisant Symfony pour le backend et Git pour versionner mon code, j'ai assuré la qualité et la maintenabilité du projet. La documentation régulière des progrès et l'utilisation d'un environnement de développement et de recette ont permis de maintenir une cohérence et une qualité élevées tout au long du projet.

Enfin, j'ai vérifié et validé la qualité de l'application par les tests (AC21.04). Même si je n'ai pas créé de fichiers de test automatisés, j'ai effectué des tests manuels exhaustifs avec les fichiers shapefile fournis par Poisson Soluble. Ces tests ont permis de garantir que l'application fonctionnait correctement et répondait aux spécifications établies.

En conclusion, ce projet m'a permis de mettre en pratique de nombreuses compétences acquises au cours de ma formation, tout en me familiarisant avec des outils et des méthodologies professionnelles. J'ai développé une application fonctionnelle, tout en respectant les exigences et en assurant une qualité optimale, ce qui constitue une expérience précieuse pour mon avenir professionnel.

Visa du tuteur de stage

Lu et approuvé le 6 juin 2024

A handwritten signature in black ink, appearing to read "SW".

Stéphane Wouters

Bibliographies

- [1] Préfecture de la Manche, « FAQ sur le PCS ». [En ligne]. Disponible sur: <https://www.manche.gouv.fr/contenu/telechargement/18913/123057/file/FAQ+sur+le+PCS.pdf>
- [2] Esri., « Fichier de formes—Aide ArcGIS Online | Documentation ». Consulté le: 4 juin 2024. [En ligne]. Disponible sur: <https://doc.arcgis.com/fr/arcgis-online/reference/shapefiles.htm>
- [3] Internet Engineering Task Force, « GeoJSON ». Consulté le: 31 mai 2024. [En ligne]. Disponible sur: <https://geojson.org/>
- [4] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, et S. Hagen, « The GeoJSON Format », Internet Engineering Task Force, Request for Comments RFC 7946, août 2016. doi: 10.17487/RFC7946.
- [5] P. P. & Osg. PostGIS PSC & OSGeo, « PostGIS », PostGIS. Consulté le: 31 mai 2024. [En ligne]. Disponible sur: <https://postgis.net/>
- [6] PostGIS PSC & OSGeo, « Chapter 4. Data Management ». Consulté le: 31 mai 2024. [En ligne]. Disponible sur: https://postgis.net/docs/using_postgis_dbmanagement.html#shp2pgsql_usage
- [7] Esri., « Qu'est-ce qu'un SRID ?—ArcMap | Documentation ». Consulté le: 4 juin 2024. [En ligne]. Disponible sur: <https://desktop.arcgis.com/fr/arcmap/latest/manage-data/using-sql-with-gdbs/what-is-an-srid.htm>
- [8] Geomatic Solution, « EPSG Geodetic Parameter Dataset ». Consulté le: 31 mai 2024. [En ligne]. Disponible sur: <https://epsg.org/home.html>

Annexes techniques

Annexe 1. Fonction getData du ShapeRepository

```
21     public function getData(): array
22     {
23         $sql = "SELECT *, ST_AsGeoJSON(geom) AS geometry FROM " . $this->tableName;
24         $query = $this->connection->executeQuery($sql);
25         return $query->fetchAllAssociative();
26     }
```

Annexe 2. Google document de suivi de projet

The screenshot shows a Google Document with the following structure:

- Sujet stage Aurel**
 - Chapitre 1 - Découverte des PCS
 - Étape 0 - Symfony
 - Étape 1 - Première carte avec Leaflet
 - Étape 2 - Affichage d'un geojson
 - Étape 3 - Branchement au temp...
 - Étape 4 - Liste déroulante
 - Étape 5 - Markers personnalités ...
 - Étape 6 - Hébergement en envir...
 - Chapitre 2 - Fichiers shapes
 - Observation avec QGIS
 - Importation d'un fichier shape a...
 - Création d'un webservice
 - Chapitre 3 - Le projet
 - POC : Importation d'un shape p...
 - Recherche ergonomique

Sujet stage Aurel

Avril 2024

L'objectif est de développer un outil qui permet d'importer un PCS (Plan communal de sauvegarde) au format shape dans une base de données pour l'afficher sur une carte.

Pour arriver à cet objectif, une succession d'étape est proposée pour prendre en main les technologies nécessaires.

Chapitre 1 - Découverte des PCS

Étape 0 - Symfony

Suivre les vidéos tutos suivantes pour découvrir Symfony:

https://www.youtube.com/watch?v=I51E68Lw6dc&ab_channel=Grafikart.fr

- Présentation du framework
- Installation du framework
- Nos premières pages
- Moteur de template twig
- Comprendre les services (Ne pas faire la partie ORM/Formulaires)

Initialiser un projet symfony 6.4 PHP 8.3 vierge nommé "shape2pcs"

Étape 1 - Première carte avec Leaflet

Avec Symfony, créer une page HTML avec Twig sur la route "/lab/predict".

[Liens vers le google doc](#)

Annexe 3. Readme de l'application

The screenshot shows a dark-themed README file for a project named "Shape 2 PCS". The file includes sections for prerequisites, installation instructions, environment variables, and PostgreSQL setup.

Shape 2 PCS

Ce projet a pour but de transformer un fichier shape (.shp) en fichier geoJSON correct.

Prérequis

- PHP 8.3
- Composer
- Symfony 6.4
- NPM
- PostgreSQL & PostGIS

Installation :

```
composer install
npm install
```

Variable du .env :

```
DATABASE_URL="postgresql://postgres@127.0.0.1:5432/shape2pcs" PICTO_URL="https://pictos.dev-wiki.predict.blue/picto/plan/"
PCS_URL="https://carto.dev-wiki.predict.blue/pcs/FRA/"
COMMUNES_URL="https://geoloc.dev-wiki.predict.blue/france/get-all"
```

Création de la base de donnée PostgreSQL :

Créer une base de donnée PostgreSQL nommée "shape2pcs"

Activer l'extension POSTGIS :

```
\c shape2pcs
CREATE EXTENSION postgis;
```

Lancement :

Lancer npm et le serveur php :

```
php -S localhost:8000 -t public/  
npm run watch
```



Manuel d'utilisation :

Résumé

- Sur la page d'accueil, uploadez un dossier compressé contenant le .shp et les fichiers qui vont avec (Attention : la configuration de PHP accepte les fichiers de maximum 2MO). Cliquez sur "Valider".
- Si votre dossier compressé contient plusieurs .shp, sélectionnez celui que vous voulez convertir, sinon passez à l'étape suivante.
- Normalement, l'encodage des coordonnées est détecté automatiquement. Sinon, il vous faut aller voir les propriétés de votre fichier .shp dans un logiciel de cartographie (QGIS par exemple). Cliquez sur "Valider".
- Vous pouvez visualiser la carte des points et la liste des attributs du .shp. Vous pouvez cliquer sur un attribut pour voir un exemple des données contenues.
- Dans ce modal, vous pourrez, d'une part, changer le nom que l'attribut aura dans le geoJSON grâce à la liste déroulante et en cliquant sur le +. D'autre part, ajouter l'attribut en gardant le même nom ou le supprimer.

Précisions :

- Le code couleur des attributs est le suivant :
 - Sans couleur : L'attribut n'est pas ajouté au geoJSON.
 - Vert : L'attribut est ajouté dans le geoJSON avec le même nom.
 - Jaune : L'attribut est ajouté avec un nom différent dans le geoJSON. Par exemple, si un attribut qui a pour nom "alias_pict" dans le shape a été renommé "alias" dans le geoJSON, alors il apparaîtra en jaune.
- Les attributs détectés automatiquement et ceux disponibles dans la liste déroulante de l'attribution peuvent être changés depuis le menu des options ⓘ en gardant la syntaxe suivante : ["alias", "name", "comment", ...]

Annexe 4. Exemple de l'historique des commit

The screenshot shows a GitHub commit history with the following details:

- Commits on May 29, 2024:**
 - ajout d'une erreur sur la page principale si les attributs rentrés ne sont pas bon (Unmoriel committed last week - 2/2) - Commit ID: b1f4219
 - hotfix (Unmoriel committed last week - 2/2) - Commit ID: 4cb877c
 - Correction de bug et meilleur affichage des erreurs dans les options (Unmoriel committed last week - 2/2) - Commit ID: da8aa9c
- Commits on May 28, 2024:**
 - readme.md mis à jour. Géctions des erreurs plus claire (Unmoriel committed last week - 2/2) - Commit ID: 0fde01c
- Commits on May 27, 2024:**
 - * Ménage dans les fichiers (controllers & templates inutiles supprimé) (Unmoriel committed last week - 2/2) - Commit ID: f4709ed
- Commits on May 21, 2024:**
 - Refonte de logique d'affichages des attributs. (Unmoriel committed 2 weeks ago - 2/2) - Commit ID: c97c42c
- Commits on May 17, 2024:**
 - Problème du rechargement de la map lors de l'ajout des prop résolue. (Unmoriel committed 3 weeks ago - 2/2) - Commit ID: 09f4ed3
- Commits on May 13, 2024:**
 - Merge remote-tracking branch 'origin/main' (Unmoriel committed 3 weeks ago - 2/2) - Commit ID: d275072
 - Petites améliorations graphiques (mode dark/light) + précision des erreurs accrues (Unmoriel committed 3 weeks ago) - Commit ID: 8888982

Annexe 5. Fichier de questions à poser avant un point

The screenshot shows a Notepad file titled "questions [Lecture seule]" containing the following text:

```
1 * Tailles max des fichiers ?
2 - Mettre le max dans le dockerFile
3
4 * Propriétés de base du geoJSON à garder (si oui -> préremplir ces catégories ?)
5 - Voir avec Vincent
6
7 * Sélectionner toute les propriétés directement ?
8 - Utile mais pas indispensable
9
10 * Possibilité de renommer une propriété ?
11 - Oui (voir avec Vincent)
12
13 * Plus d'une utilisation en même temps sur le serveur ?
14 - Commencer à y réfléchir
15
16 * Rennomage de la table postgres
17 - Voir dans le code lors de la création de la table
18
19 * Possibilité de réunir deux fichiers shapes ?
20 - Intéressant mais ça va prendre du temps
21
22 * Interfaces ok ?
23 - Continuer avec Olivier (partie basse du modal à corriger)
```

Annexe 6. Notes Notion de Stage

The screenshot shows a Notion database titled "Note de Stage". The interface includes a header with options to add an icon, cover image, or description, and a status bar indicating the last modification was just now. The main area is a table view with the following data:

| Aa Nom | Date | Étiquettes |
|-----------------|---------------|---|
| Jour 1 | 8 avril 2024 | Installation, Etape 1, Léo, David |
| Infos Pratiques | 8 avril 2024 | Informations, Horaires, Stéphane |
| Sujet du Stage | 8 avril 2024 | sujet, technologies, pcs |
| Jour 2 | 9 avril 2024 | Etape 2, Faniry, Vincent |
| Jour 3 | 10 avril 2024 | Théo, Etape 4, webapp |
| Jour 4 | 11 avril 2024 | git, Christophe, Installation, docker, SI |
| Jour 5 | 12 avril 2024 | Alexandrine, Nettoyage, Node.js |
| Jour 6 | 15 avril 2024 | docker |
| Jours 7 | 16 avril 2024 | PostGis, PostgreSQL, RSE, BDD, shp2pgsql |
| Jour 8 | 17 avril 2024 | shp2pgsql, Jonathan, Olivier, PostgreSQL |
| Jour 9 | 18 avril 2024 | Stimulus, Stéphane, .env |
| Jour 10 | 19 avril 2024 | Stimulus, .env |
| Jour 11 | 22 avril 2024 | shp2pgsql |
| Jour 12 | 23 avril 2024 | bug, EPSG |
| Jour 13 | 24 avril 2024 | bug, EPSG, Stéphane, javascript |
| Jour 14 | 25 avril 2024 | bug, ré attribution des attributs, docker |

[Lien vers les notes notions](#) (Il vous faut un compte notion)

The screenshot shows a web-based application for managing spatial data. On the left, there is a sidebar with a tree view of layers: 'layer', 'name', 'begin', 'end', and 'action'. Below this are several input fields: 'y_193', 'geom', 'alias_pict', 'aliasfutur', 'code_pict', 'icon', 'layer', 'path', 'priorita', 'timestamp', 'name', 'descriptio', and 'altitudemo'. A large button labeled 'Convertir en geoJSON' is at the bottom. The main area features a map of a coastal region with numerous location markers. A modal window titled 'Attribut "name"' is open, showing a table of data with columns 'Attribut shape:', 'Attribut geoJSON:', and 'Exemple de données:'. The 'Attribut shape:' column contains 'name'. The 'Attribut geoJSON:' column is marked as 'Non attribué'. The 'Exemple de données:' column lists several place names. At the bottom of the modal are buttons for 'Ajouter' and 'Supprimer'. The background shows a blurred version of the same interface.

