

RAPPORT DE PROJET SEMESTRIEL

LICENCE 3 INFORMATIQUE
ANNÉE 2023-2024

Carte et statistiques des bornes de recharge et des stations thermiques en France

HEQUET Erwann, BERNARD Thomas, BAUD Joris

Encadré par
FELEA Violeta



Février 2024

Table des matières

Introduction	2
1 Cadre et objectifs du projet	3
1.1 Contexte de la transition énergétique	3
1.2 Objectifs du projet	3
1.3 Planning et gestion du projet	4
2 Analyse	5
2.1 Choix des technologies	5
2.2 Collecte des données	6
2.3 Nettoyage des données	7
2.4 Définition de la base de données	8
3 Développement de l'application	10
3.1 Généralités	10
3.2 Visualisation spatiale des stations	11
3.3 Statistiques	17
3.4 Comptes utilisateurs	17
3.5 Pages administrateurs	19
4 Déploiement de l'application	23
4.1 Connexion et utilisation d'une machine virtuelle	23
4.2 Serveur web et installation	24
4.3 Configuration	24
4.4 bilan du déploiement	27
Conclusion	28
Annexe	29

Introduction

Dans le cadre de l'unité d'enseignement Projet de la troisième année de Licence informatique à l'Université de Franche-Comté, nous avons entrepris un projet d'analyse et de visualisation de l'infrastructure publique de recharge électrique et des stations de carburant en France. Ce projet est rendu possible grâce à l'utilisation de données publiques OpenSource et d'autres provenant du gouvernement français. L'objectif est d'analyser l'évolution et la distribution spatiale de ces infrastructures essentielles, à travers une carte et des statistiques.

Nous visons à créer une plateforme conviviale et évolutive, permettant aux utilisateurs de visualiser facilement l'emplacement des bornes de recharge électrique et des stations de carburant à travers la France, tout en offrant la possibilité d'ajouter et de modifier les données. Cette approche participative permet non seulement de maintenir les informations à jour, mais également d'enrichir la base de données avec des détails supplémentaires sur les infrastructures existantes.

En associant l'analyse des données à une interface utilisateur intuitive, ce projet vise à créer un outil pratique pour tous les conducteurs, qu'ils utilisent des véhicules électriques ou non, à préparer et optimiser leur trajet grâce à une visualisation claire de la localisation des bornes de recharge électrique et des stations de carburant.

Ayant la volonté de mettre en pratique les connaissances que l'on a pu accumuler ces dernières années et grâce à la supervision de Violeta FELEA, nous espérons que cette approche inclusive contribuera à sensibiliser un public plus large aux avantages de la mobilité électrique, et encouragera une transition progressive vers des modes de transport plus respectueux de l'environnement.

Ce rapport est donc réalisé comme un rapport technique, le but n'étant pas de vendre l'application, mais plutôt de justifier nos choix et de retranscrire comment se sont passées les différentes phases que nous avons pu traverser tout au long de ce projet.

Après avoir donné quelques éléments de contexte sur ce projet, ce rapport retranscrira les principales étapes d'analyse réalisées. Puis, il présentera la phase de conception de l'application, quels choix ont été faits et pourquoi, avant de détailler le développement en lui-même de l'application ainsi que son déploiement. Cette partie sera l'occasion de rentrer en profondeur sur différents aspects et de justifier certains choix. Pour conclure, ce rapport dressera un bilan technique, mais aussi personnel, quant à l'organisation au sein du groupe ainsi que les apports de ce projet. Ce sera l'occasion de revenir sur les objectifs accomplis et de proposer des axes d'améliorations futurs de l'application.

1 Cadre et objectifs du projet

1.1 Contexte de la transition énergétique

Il est crucial d'agir rapidement contre le réchauffement climatique. Les émissions de gaz à effet de serre, largement issues des industries fossiles, jouent un rôle majeur dans ce phénomène. Pour contrer cette tendance, la transition énergétique se concentre sur la réduction de ces émissions en décarbonant divers secteurs économiques, notamment l'industrie automobile. L'Union européenne s'est fixée des objectifs ambitieux en matière de réduction des émissions de gaz à effet de serre et de développement des énergies renouvelables. La vente de véhicules thermiques neufs sera interdite en Europe à partir de 2035. Pour accompagner cette transition, l'UE vise à installer des bornes de recharge rapide tous les 60 kilomètres sur les principales autoroutes d'ici 2026. Pour soutenir cette transition énergétique, il est essentiel de développer et innover dans de nouvelles technologies de visualisation des infrastructures, ainsi que des statistiques afin de sensibiliser et accompagner la population dans ce processus.

1.2 Objectifs du projet

1.2.1 Objectifs initiaux

À l'origine, le sujet du projet stipulait deux objectifs principaux : la visualisation spatiale des bornes de recharges électriques en France, ainsi des statistiques à propos de celles-ci. Ces statistiques permettront notamment de savoir si l'évolution du nombre, de la puissance et de la répartition de ces bornes partout en France est en adéquation avec le discours tenu par l'UE.

1.2.2 Évolutions et ajouts d'objectifs

Bien conscients qu'à l'heure actuelle, 97,7% des véhicules en circulation en France sont des véhicules thermiques, il nous semblait intéressant de traiter non pas uniquement des bornes de recharge, mais également de la visualisation des stations thermiques, et de pointer du doigt certaines statistiques intéressantes à ce sujet. En effet, l'application serait ainsi plus complète et permettrait de toucher plus de personnes sans s'éloigner du sujet initial.

Afin de proposer une meilleure expérience, la mise en place de comptes utilisateurs paraissait être une bonne option. Cela permettrait aux utilisateurs de définir leurs stations favorites, mais aussi de donner leur avis quant à une station, de par une note ou même un commentaire écrit. En alignment avec cette vision d'amélioration de l'expérience utilisateur, l'application prévoit également de mettre en place des filtres permettant aux utilisateurs d'afficher uniquement les informations qui les intéressent. De plus, ils auront la possibilité de créer de nouvelles stations ou de proposer des modifications aux stations existantes.

Comme dit précédemment, nous avons la volonté de créer une application qui se suffit à elle-même, c'est-à-dire où tout peut être géré depuis l'application. Il semblait donc

intéressant de proposer toute une partie gestion administrateur afin de gérer les stations, les comptes utilisateurs, ou interagir avec la base de données. Dans ce cadre, la création de comptes administrateurs s'avère également indispensable.

1.3 Planning et gestion du projet

Le projet nous a été attribué le 24 octobre 2023 à la suite d'une sélection rigoureuse. Tous les projets ont été publiés en ligne et évalués par les différents groupes. Chacun a ensuite classé les projets par ordre de préférence, et les attributions ont été réalisées dans le but de satisfaire autant de groupes que possible. Le sujet qui nous a été octroyé a été celui de la carte des bornes électriques en France, et dès lors, sur la période de novembre à décembre, nous avons immédiatement entamé une analyse et une définition de la structure du projet. Suite à cette période de réflexion et de réunions, nous avons commencé le développement principal du projet en janvier. La figure 1.1 présente les grandes lignes de cette répartition temporelle.

Notre emploi du temps prévoyait des semaines dédiées au projet, ce qui nous a permis de concentrer nos efforts sur son avancement. La répartition des tâches au sein du groupe s'est faite de manière équilibrée et cohérente avec les compétences et les envies de chacun. Par exemple, Thomas s'est concentré sur le développement du style et la gestion de compte, Joris sur la gestion des données et des statistiques, et Erwann sur le développement de la carte et de la partie administrateur du site.

Afin de travailler de manière collaborative et organisée, nous avons utilisé Git pour le partage de code source, ce qui nous a permis de travailler simultanément sur différentes fonctionnalités. Pour communiquer efficacement, nous avons utilisé Discord. Cela nous a permis de partager des idées, de poser des questions et de résoudre rapidement les problèmes rencontrés.

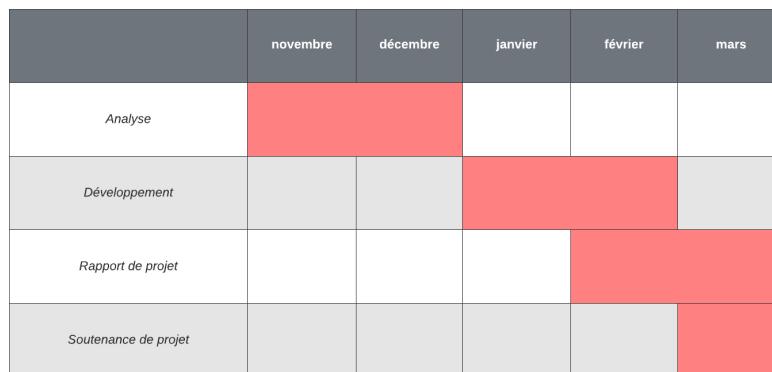


FIGURE 1.1 – Diagramme de gantt des principales étapes du projet

2 Analyse

2.1 Choix des technologies

En raison des contraintes et des objectifs du projet (accessibilité, interopérabilité, mises à jour régulières...), il semblait évident de s'orienter vers une application web.

L'application a été développée selon un modèle client-serveur, où le serveur est chargé de traiter les données et où le client sert d'interface utilisateur.

Parmi tous les langages de programmation disponibles, nous avons choisi JavaScript, qui permet de créer une application fullstack où le même langage est utilisé côté client et côté serveur. Pour ce dernier, nous avons opté pour l'environnement d'exécution Node.js, qui offre la possibilité de gérer un grand nombre de clients simultanément grâce son moteur JavaScript V8 et son architecture non bloquante. Architecture dont on peut apercevoir le mécanisme en figure 2.1. Effectivement, lorsque le serveur reçoit une requête du client, celui-ci ne se bloque pas le temps de répondre au client, mais se positionne immédiatement dans l'état prêt afin de gérer de nouvelles requêtes. Une fois que la tâche relative à une requête est terminée, le serveur en est averti grâce à un mécanisme de promesse. Il peut alors transmettre la réponse au client. Un autre avantage de Node.js réside dans la gestion des packages grâce à npm (Node Package Manager). En effet, cela permet d'installer facilement des bibliothèques externes et d'assurer un déploiement simple sur une autre machine. De plus, JavaScript donne accès à de nombreuses API qui seront très utiles, ce qui évite d'avoir à tout coder et permet d'utiliser des librairies performantes.

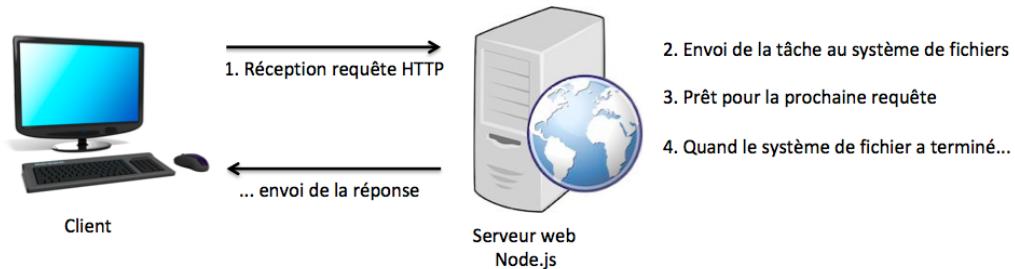


FIGURE 2.1 – Architecture non bloquante de Node.js

Afin de communiquer entre le client et le serveur, nous utilisons des websockets pour échanger des informations et des données. Leur nature non bloquante présente un avantage certain en termes de rapidité d'exécution.

Nous avons également utilisé un mélange d'HTML et de CSS pour structurer et styliser les pages. Tous les éléments pouvant varier sont créés via JavaScript et ajoutés au DOM¹, qui correspond au contenu de la page web. En revanche, les éléments qui sont toujours présents sont directement créés dans le fichier HTML.

Concernant la base de données, nous avons opté pour MariaDB, une solution à la fois robuste et facile à installer. Notre familiarité avec MariaDB, acquise lors de cours

1. Le Document Object Model est une interface de programmation permettant à des scripts d'examiner et de modifier le contenu du navigateur web.

précédents, a été déterminant dans ce choix. Cette familiarité a permis de mettre en place rapidement la base de données nécessaire au projet.

2.2 Collecte des données

2.2.1 Les données électriques

La qualité des données électriques revêt une importance cruciale dans le développement de l'application pour garantir la pertinence du projet. Lors de la phase de recherche, aucune source de données ne combinait à la fois une grande quantité et une haute qualité de données. Afin d'optimiser la pertinence des informations, nous avons pris la décision de croiser deux sources de données distinctes et d'en tirer le meilleur parti.

La première source de données électrique est OpenStreetMap. Cette base de données cartographiques est ce qui se rapproche le plus d'une source de données qualitative et quantitative. L'avantage, mais aussi l'inconvénient, réside dans le fait que la base de données soit collaborative. Les informations sont mises à jour fréquemment grâce à la contribution de la communauté Open Source, ce qui assure une actualisation régulière des données. Cependant, cette collaboration peut également entraîner des données erronées ou incomplètes, car la qualité et la précision peuvent varier en fonction des contributeurs, de leurs compétences et de leur expérience dans le domaine.

La seconde source de données venant compléter les données non référencées dans la base OpenStreetMap sont les données du site data.gouv.fr. Ce site constitue le portail open data du gouvernement français. Depuis sa création en 2011, il met à disposition un large éventail de jeux de données provenant de diverses administrations et organismes publics. Ces données couvrent de nombreux domaines, tels que l'environnement, le transport...

Dans le cadre du projet, nous utilisons le "Fichier consolidé des Bornes de Recharge pour Véhicules Électriques". Ce fichier est mis à jour régulièrement par les administrations publiques, les gestionnaires de réseaux de recharge, les opérateurs de services de recharge, et d'autres entités impliquées dans le déploiement et la gestion de l'infrastructure de recharge pour véhicules électriques en France. Ces données sont ensuite consolidées et rendues accessibles au public.

Afin d'ajouter quelques statistiques supplémentaires, nous avons intégré deux nouvelles sources de données :

La première est fournie par Enedis et documente l'évolution du nombre de stations électriques par trimestre, ainsi que leur répartition d'accès public, privé ou réservé aux entreprises.

La seconde source de données correspond à l'évolution du nombre de voitures par commune depuis le 1er trimestre 2020, en fonction de leur typologie (ici, le nombre de voitures électriques immatriculées par commune). Cette seconde source de données est stockée dans la table CommuneData.

2.2.2 Les données carburant

Pour accompagner les données électriques, l'application utilise le jeu de données "Prix des carburants en France - Flux instantané - v2 améliorée" mis à disposition par le ministère de l'Économie et des Finances via son site open data.

Ces données comprennent des informations générales sur le point de vente telles que l'adresse, les coordonnées géographiques, les horaires d'ouverture, les services proposés, ainsi que les prix, les ruptures de stock de carburants, et les fermetures définitives ou temporaires des points de vente. Le flux de données instantané est mis à jour toutes les 10 minutes, et les prix sont actualisés via une interface réservée exclusivement aux gérants de points de vente, garantissant ainsi une excellente qualité des données. À noter qu'un arrêté ministériel du 12 décembre 2006 rend obligatoire la déclaration des prix pratiqués pour tout gérant de point de vente de carburants ayant vendu au moins 500 mètres cubes des carburants SP95, SP95-E10, gazole, E85, GPLC et SP98.

Afin de valoriser les données, nous utilisons l'historique des prix des carburants mis à disposition par le ministère de l'Économie. Ces données nous permettent d'accéder à l'historique des prix pour chaque station depuis 2007, ainsi que de créer une moyenne.

2.3 Nettoyage des données

Comme mentionné précédemment, une seule source de données ne pouvant pas répondre aux attentes en termes de qualité et de quantité, nous avons pris la décision d'utiliser deux sources de données électriques distinctes et d'en tirer le meilleur de chacune.

Pour éviter les doublons dans la base de données, lors de l'insertion des données Open Source d'OpenStreetMap, le serveur utilise une fonction spécifique. Cette fonction vérifie si une station est déjà présente dans la base de données en examinant ses coordonnées. Elle procède en déterminant si une autre station se trouve dans un carré d'environ 55 mètres autour de la station actuelle.

```

1. /**
2. * Retourne si les coordonnées sont déjà dans la base de données modulo 0.0005
3. *
4. * @param {double} lat la latitude
5. * @param {double} lon la longitude
6. * @param {*} coordAlreadyInDB les coordonnées déjà dans la base de données
7. * @returns
8. */
9. function CheckCoordAlreadyInDB(lat, lon, coordAlreadyInDB) {
10.     for (let i = 0; i < coordAlreadyInDB.length; i++) {
11.         if (Math.abs(coordAlreadyInDB[i].lat - lat) <= 0.0005 && Math.abs(coordAlreadyInDB[i].lon - lon) <= 0.0005) {
12.             return true;
13.         }
14.     }
15.     return false;
16. }
```

FIGURE 2.2 – Fonction CheckCoordAlreadyInDB

L'utilisation de différentes sources de données entraîne d'autres problèmes, dus à l'hétérogénéité des données et à leurs formats différents (respectivement JSON et CSV), avec des attributs et une structuration différente, nécessitant ainsi deux traitements totalement distincts. En plus de leur structuration distincte, certaines données étaient erronées (coordonnées en dehors de la France) ou incomplètes (noms de ville ou de rue manquants...).

Afin de minimiser ces lacunes de données, nous utilisons une API du gouvernement²

2. <https://adresse.data.gouv.fr/api-doc/adresse>

qui, à partir de coordonnées géographiques, renvoie les informations dont il dispose pour ce lieu (adresse, région...). Cette méthode permet de réduire considérablement les données manquantes.

En ce qui concerne les données sur le carburant, le traitement est très simple : les données manquantes sont remplacées par une chaîne de caractères "Inconnu". Cela permet d'éviter la présence de valeurs NULL dans la base de données et simplifie l'implémentation de l'application, en considérant toutes les valeurs enregistrées comme étant correctes.

2.4 Définition de la base de données

La figure 4.9 présente donc un diagramme MCD³ représentant la structure de notre base de données.

De par ces données plus ou moins hétérogènes, il est important de bien séparer celles dites statiques et celles dynamiques. De plus, certains champs de la base de données n'auraient pas de sens pour certaines stations (par exemple le prix d'un carburant pour une station électrique). Nous avons donc pris la décision de créer plusieurs tables, notamment une pour les types de prises et leur puissance et une autre pour les prix des carburants. Cependant, pour privilégier la performance, nous ajouterons toutes les informations communes aux stations à la même table afin d'éviter l'utilisation d'union.

Cette approche permet aux requêtes de s'exécuter plus rapidement, mais nous impose de gérer correctement l'identifiant servant de clé primaire. Effectivement, s'il est d'usage de considérer ce champ en "auto-incrémentation" c'est-à-dire que le SGBD gère lui-même le numéro qu'il attribue à un nouveau t-uplet inséré, ici ce n'est pas possible. L'utilisation de la table traitant des historiques des prix des carburants nous impose de conserver le même identifiant pour les stations thermiques que celui présent dans les fichiers d'origines. Lors de l'ajout en base de données, les stations thermiques conserveront donc cet identifiant tandis que celles électriques se verront attribuer un nouvel identifiant laissé libre.

Concernant les autres tables, nous aurons l'occasion, au cours de ce rapport, de détailler et de justifier leur essence et la présence d'un tel ou tel champ.

En plus de la possibilité d'être mise à jour manuellement via une interface administrateur, la table HistoriqueCarburant est mise à jour quotidiennement à 5h30. Quant aux informations des stations électriques et carburants, celles-ci sont mises à jour toutes les heures.

3. Modèle Conceptuel des Données

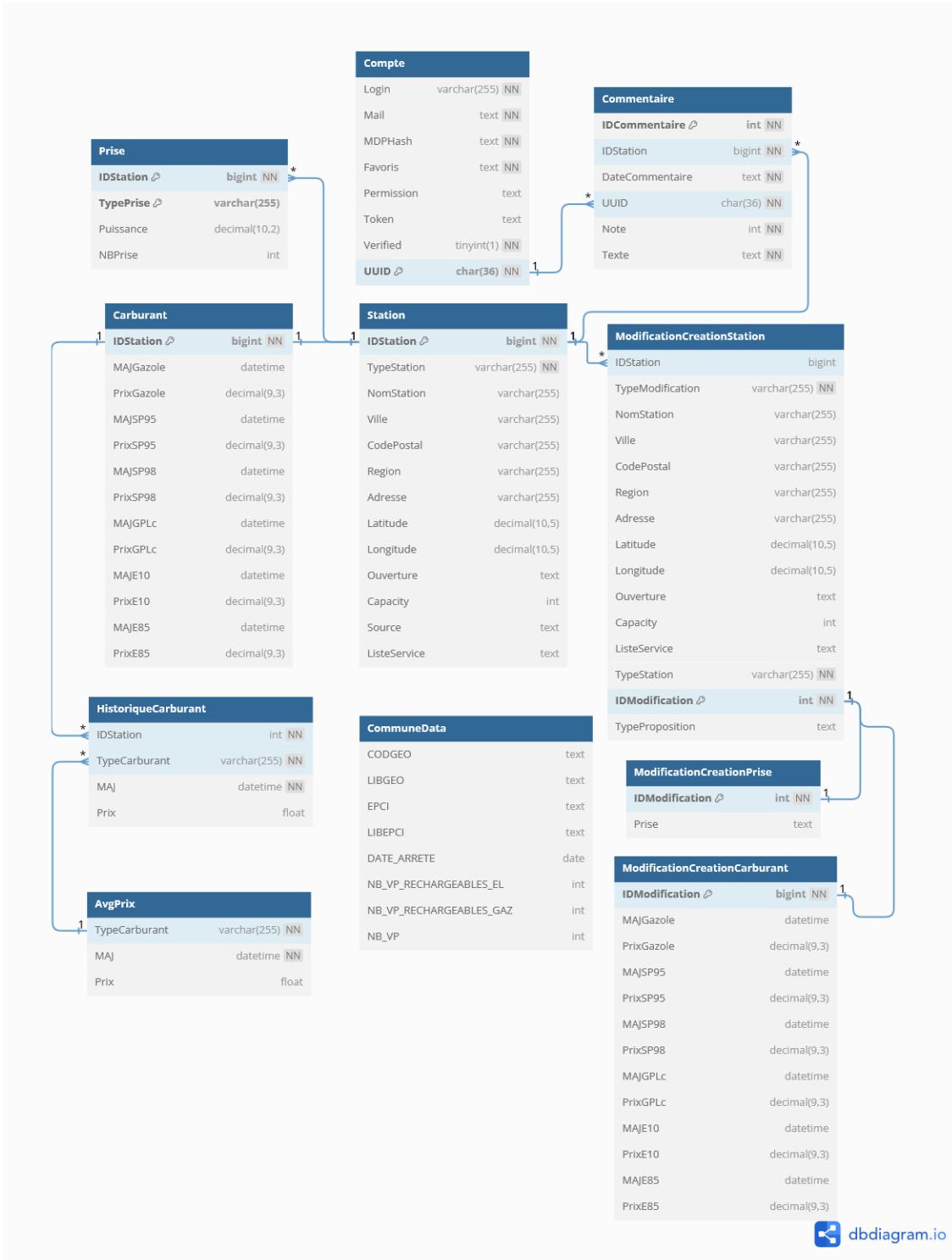


FIGURE 2.3 – Structure de la base de donnée

3 Développement de l'application

3.1 Généralités

3.1.1 Mode sombre

Notre application se veut donc centrée sur l'expérience utilisateur et, aujourd'hui, selon earthweb¹, plus de 80 % des utilisateurs disent préférer utiliser un mode sombre. Afin de plaire à cette très grande majorité, nous avons décidé de créer notre propre mode sombre, où darkmode. Pour ce faire, nous utilisons un mélange de CSS et de JavaScript. Celui-ci s'activera lorsqu'un utilisateur effectuera la combinaison de touche alt + d.

3.1.2 Interaction avec la base de données

L'application sera donc amenée à interagir avec une base de données à de nombreuses reprises. Pour ce faire, depuis un serveur nodeJS, il faut installer le package mysql. Celui-ci fournit tout le nécessaire pour pouvoir interagir avec une base de données. L'aspect monothread de JavaScript impose l'utilisation du mécanisme de promesse, et pour alléger le code, nous avons décidé de créer une fonction `requetteBDD()` présentée en figure 3.1. Cette fonction permet, grâce au module mysql de nodeJS, de se connecter à la base de données et d'interagir avec elle à l'aide de sockets. Puis, grâce au mécanisme de promesse, il est possible de soumettre la requête et de ne plus bloquer le thread, la suite du programme reprend lorsque l'on a obtenu une réponse de la base de données, peu importe l'issue de la requête. Il suffit ainsi de passer en paramètre la requête à exécuter sous forme de chaîne de caractère ainsi qu'une référence à une fonction à appeler en cas de réussite et une autre en cas d'échec.

Sur internet, les pirates et personnes mal intentionnées se font nombreux. C'est pourquoi il faut toujours faire attention à se protéger. Ici, dans le cadre de l'interaction avec la base de données, il paraît obligatoire d'utiliser la fonction `mysql.escape()` lorsque l'on utilise des données provenant d'un utilisateur afin d'éviter tout risque d'attaque par injection SQL. Une injection SQL permet à un utilisateur malveillant de passer outre certaines conditions et de récupérer, modifier, supprimer des informations présentes en BDD qui seraient normalement inaccessibles pour un utilisateur lambda.

3.1.3 Bibliothèque

Nous avons donc décidé de développer une application multipages, où chaque page possède son propre script JavaScript. Cependant, il paraît évident que certaines fonctionnalités ou options sont partagées par plusieurs pages. Par exemple, le mode sombre cité précédemment ou bien la possibilité de récupérer des informations quant à l'utilisateur connecté et de vérifier si celui-ci est administrateur. C'est donc avec des objectifs de

1. <https://earthweb.com/how-many-people-use-dark-mode/>

```

1. const mysql = require('mysql');
2.
3. const connectionBDD = mysql.createConnection({
4.   host: config.DB_HOST,
5.   user: config.DB_USER.split('@')[0],
6.   password: config.DB_PASS,
7.   database: config.DB_NAME,
8.   socketPath: config.DB_SOCKET,
9.   multipleStatements: config.DB_STATEMENT === 'TRUE'
10. });
11.
12. connectionBDD.connect((err) => {
13.   if (err) {
14.     console.error('Erreur de connexion à la base de données : ' + err.stack);
15.     reject(err);
16.     return;
17.   }
18.   console.log('Connecté à la base de données avec l\'ID ' + connectionBDD.threadId);
19. });
20.
21. /**
22. * Permet d'interagir avec la BDD et d'exécuter la requête passée en paramètre.
23. *
24. * @param {string} sqlQuery requête SQL sous la forme d'une chaîne de caractère.
25. */
26. function requeteBDD(sqlQuery){
27.   return new Promise((resolve, reject) => {
28.     connectionBDD.query(sqlQuery, (error, results, fields) => {
29.       if (error) {
30.         console.error('Erreur lors de l\'exécution de la requête : ' + error.stack);
31.         reject(error);
32.       }else{
33.         resolve(results);
34.       }
35.     });
36.   });
37. }

```

FIGURE 3.1 – Interactions avec la base de données

réutilisabilité du code et de ne pas se répéter (principe DRY²) que nous avons décidé de créer un script JavaScript, nommé biblio.js, inclus dans toutes les pages et qui propose une multitude de fonctions utiles.

3.2 Visualisation spatiale des stations

3.2.1 Choix des API et bibliothèques externes

Au cinquième semestre de Licence informatique, nous avions déjà eu l'occasion de travailler sur un projet de visualisation. Le but était de mettre en place une carte et

2. don't repeat yourself en anglais

des marqueurs sur celle-ci grâce à OpenStreetMap³ et Leaflet⁴, ce que l'on a reconduit pour ce projet. En effet, OpenStreetMap est une grande base de données libre utilisant des coordonnées et autres données gps. Leaflet quant à elle aussi une bibliothèque libre permettant la mise en place d'une carte interactive, notamment grâce à la mise en place de marqueurs.

Nous proposons également aux utilisateurs de directement rechercher une adresse, ville, code postal ou tout autre élément permettant une identification géographique afin de recentrer et zoomer la carte sur cet élément. Pour ce faire, nous utilisons Nominatim qui est un outil fonctionnant de pair avec OpenStreetMap, remplissant tout à fait notre demande en récupérant les latitudes et longitudes de l'élément renseigné.

3.2.2 Créations des marqueurs

Comme dit précédemment, conserver une carte claire, non surchargée et en adéquation avec la demande du client est une chose essentielle. Bien évidemment, le programme se doit de rester suffisamment performant, ou du moins de répondre à la demande de l'utilisateur dans un temps correct.

Nous utilisons donc Leaflet pour créer une carte et afficher des marqueurs à des coordonnées géographiques précises sur cette dernière.

Il est impossible pour Leaflet de garantir de bonnes performances lorsqu'il a à gérer à la fois l'affichage de milliers de marqueurs et le mouvement de l'utilisateur sur la carte. Pour contourner ce problème, nous avons décidé dans un premier temps de créer nos propres clusters de marqueurs. Les marqueurs appartenant à la même région étaient regroupés, puis ceux du même département puis du même code postal et enfin les villes avant de laisser places aux marqueurs en eux-mêmes. Le niveau de zoom sur la carte permettaient de choisir quelle échelle utiliser. Cependant, la quantité de donnée était trop importante, compte tenu du nombre de stations relativement grand. Le parcours de toutes ces stations était coûteux en temps, rendant l'application inutilisable. Il nous est donc venu l'idée d'afficher uniquement les stations présentes dans la portion de territoire affichée sur l'écran de l'utilisateur. Cette fois-ci, certes la quantité de donnée était moindre et permettait donc un traitement plus rapide, mais un trop grand nombre de requêtes était envoyé vers le serveur et la base de données. Si nous avions beaucoup de client connecté en même temps, cela n'aurait pas été une solution viable.

Nous nous sommes donc tournés vers l'existant et notamment les clusters⁵ de marqueurs de Leaflet. Cette méthode permet de créer une couche à l'aide de la fonction `L.markerClusterGroup()` de Leaflet puis d'ajouter tous les marqueurs grâce à leurs coordonnées géographiques. Cette couche est ensuite ajoutée sur la map. Ainsi, le regroupement de marqueurs proches est géré automatiquement selon le zoom, et ce, de manière performante. La figure 3.2 représente ces clusters de marqueurs symbolisés par des cercles de couleurs, le numéro indiqué à l'intérieur étant, bien évidemment, le nombre de marqueurs que chaque cluster représente.

3. OpenStreetMap : <https://www.openstreetmap.org>

4. Leaflet : <https://leafletjs.com>

5. Cluster de marqueurs : <https://leafletjs.com/2012/08/20/guest-post-markerclusterer-0-1-released.html>

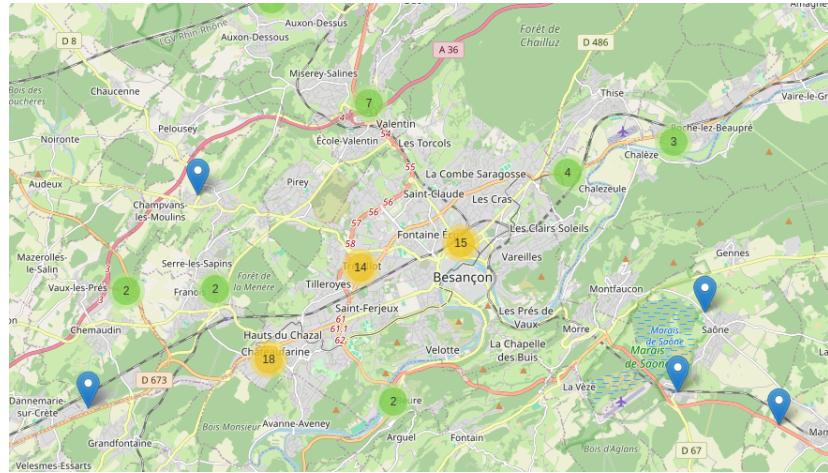


FIGURE 3.2 – Exemples de clusters et de marqueurs Leaflet sur la carte

3.2.3 Filtres

Nous avons donc divisé les filtres en trois catégories principales. Bien entendu, deux d'entre elles correspondent aux stations thermiques et électriques. Pour la troisième, il est proposé aux utilisateurs d'afficher les stations proches géographiquement parlant. Nous aurons l'occasion de donner plus de détails un peu plus tard.

La zone des filtres est placée à gauche de l'écran et est initialement masquée. L'utilisateur peut l'afficher ou la masquer à sa guise.

Une certaine hiérarchie se dessine parmi ces filtres. C'est-à-dire que l'utilisateur ne peut modifier le filtre sur les stations seulement si l'affichage de ceux-ci est effectif. Afin de pouvoir sélectionner uniquement les filtres que l'utilisateur souhaite, chaque champ est relié à une checkbox. Lorsque le serveur est interrogé, le programme commence par demander si oui ou non ce champ est sélectionné. S'il l'est, alors la valeur associée au filtre sera transmise, sinon ce filtre ne sera pas considéré.

Toujours sur cette volonté d'améliorer l'expérience utilisateur, il existe deux manières de modifier les valeurs numériques des filtres comme le présente la figure 3.3. D'une part, il est possible de les modifier à l'aide d'un input Range qui est quelque chose de plus confortable, et d'une autre grâce à un input number permettant plus de précisions. La présence de ces deux champs nous impose certaines contraintes. Par exemple, chaque Range et Number correspondant se doivent d'avoir la même valeur, et que celle-ci soit correcte (par exemple une distance négative n'a pas de sens ou un prix trop excessif n'a pas d'intérêt). Il semble par ailleurs utile de rendre impossible à l'utilisateur d'insérer autre chose que des valeurs numériques au sein de ces champs afin d'éviter une quelconque erreur.

Les filtres s'appliquent automatiquement, c'est-à-dire que dès qu'une modification des filtres a lieu, le serveur est interrogé pour obtenir les stations répondant à la demande. Une fois la réponse obtenue, le programme commence par retirer tous les marqueurs en supprimant la couche MarkerClusterGroup avant d'ajouter les marqueurs correspondant à la demande. Cependant, nous avons ajouté un délai avant d'interroger le serveur lorsque

FIGURE 3.3 – Exemple de champs pour les saisies numériques

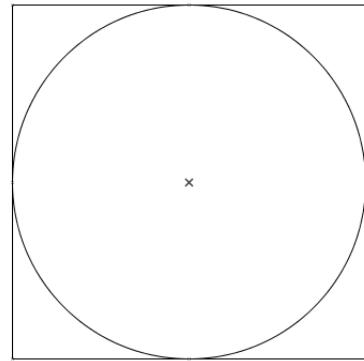


FIGURE 3.4 – Carré circonscrit à un cercle

certains champs sont modifiés, afin ne pas surcharger de requêtes ce dernier. En effet, les inputs Range peuvent changer extrêmement rapidement, la demande est donc transmise uniquement s'il n'y a pas eu de modification depuis 100 ms.

Filtre relatif à la distance

Afin de déterminer la position géographique de l'utilisateur, nous utilisons, si le navigateur le permet, la fonction `navigator.geolocation.getCurrentPosition()`. Suite à cela, si et seulement si l'utilisateur accepte de partager sa position, un filtre lui est proposé, permettant d'afficher les stations proches de ses coordonnées actuelles dans un rayon qu'il peut modifier. En effet, grâce à des formules de trigonométrie, il est possible de calculer les coordonnées du carré circonscrit à un cercle selon un rayon donné, comme présenté en figure 3.4. Pour de prochaines utilisations lors de cette session, la position de l'utilisateur est stockée côté client grâce à l'objet `sessionStorage` du navigateur.

Filtres relatifs aux stations électriques

Les filtres relatifs aux stations électriques retenus sont la capacité d'accueil, la puissance et les types de prises disponibles. Comme énoncé plus haut, il est possible de créer ou bien de modifier des stations. De ce fait, de nouveaux types de prises peuvent être ajoutés dans le futur. Cependant, les filtres se doivent de rester complets et de proposer tous les types de prises à l'utilisateur. Pour ce faire, à chaque fois qu'un client se connecte sur la page de la carte celui-ci reçoit un tableau des différentes prises présentes actuellement en BDD. Il ne reste ainsi plus qu'à créer des éléments HTML et de les ajouter aux filtres.

Filtres relatifs aux stations thermiques

Pour les stations thermiques, sont mis en place des filtres relatifs aux prix des carburants et des services proposés dans ces stations. À l'instar des différents types de prises, il est possible que de nouveaux services soient ajoutés. Cette contrainte est gérée avec

le même mécanisme que précédemment : un tableau contenant la liste des services est envoyé au client, permettant ainsi la création d'éléments HTML pour chaque service. Ces éléments servent de filtres pour affiner la recherche des utilisateurs.

3.2.4 Affichage des informations relatives à une station

L'affichage des informations relatives à une station se déclenche lorsqu'un clic est détecté sur son marqueur. Mais il restait à déterminer sous quel format celles-ci seraient affichées. Dans un nouvel onglet avec une page à part entière ? Un popup qui disparaît au bout de quelques secondes ? Ou bien l'option que nous avons choisi : un encart sur le côté, plus ou moins symétrique aux filtres.

Grâce à un écouteur de click mis en place sur le marqueur, le client envoie une requête au serveur afin d'obtenir les informations quant à cette station. À son tour, celui-ci soumet une requête à la base de données. Par la suite, le serveur transmet la réponse au client qui fait le nécessaire pour formater les réponses, dans le but de proposer une interface confortable et des données claires et explicites.

Pour les utilisateurs connectés, il est possible d'ajouter une station à ses favoris (ou bien de la retirer) en cliquant sur une icône d'étoile (grisé par défaut) qui se jaunit lorsque la station est désignée comme favorite. Il est également possible d'ajouter un commentaire et une note (plus de détails sont donnés dans la partie 3.4.3 "Page compte").

Pour les stations thermiques, un historique des prix sur les dernières années est disponible pour chaque carburant. Celui-ci se trouve sous la forme d'un graphique dans la section des informations relatives à une station. Les graphiques sont initialement masqués, mais peuvent être affichés en cliquant sur la flèche correspondant au carburant. Ce même mécanisme permet également de cacher le graphique.

Afin de faciliter la navigation vers une station sélectionnée, un bouton "Itinéraire" est ajouté à l'interface. En cliquant sur ce bouton, le système calcule automatiquement l'itinéraire depuis la position de l'utilisateur jusqu'à la station choisie, et ce, grâce à la bibliothèque Leaflet. Si la localisation est activée, l'itinéraire est calculé à partir des coordonnées GPS pour déterminer sa position actuelle. Dans le cas où la localisation est désactivée, l'emplacement est déduit à partir de l'adresse IP. L'affichage est ensuite intégré à l'interface où l'utilisateur peut modifier le lieu de départ en cas d'erreur dans les données GPS ou IP, ou bien ouvrir l'itinéraire dans une page Google Maps.

3.2.5 Modifications et créations de stations

Tout d'abord, nous tenons à souligner le fait qu'il ne s'agit que de propositions de modification ou de création et que l'ajout ou la modification définitive est soumis à l'approbation d'un administrateur. Nous aurons l'occasion d'en discuter davantage dans la section administrateur en 3.5.

Le fait de créer et modifier une station sont deux fonctionnalités très proches, d'autant plus qu'il est permis aux utilisateurs de modifier toutes les données propres à la station. Les seules différences résident donc dans le déclenchement de la procédure et le fait qu'une modification ait ses champs préremplis par les données de la station actuellement modifiée. Lorsque l'utilisateur demande une création ou une modification de station, un formulaire est affiché avec plusieurs champs remplissables par du texte ou des valeurs numériques.

Particularités des modifications de stations

La procédure de modification se déclenche donc lorsque l'utilisateur clique sur l'icône correspondant (symbolisé par un crayon dans un carré) présent dans la section des informations relatives à une station. Cette demande de modification concerne donc la station dont les informations sont affichées. À chaque fois qu'une modification est demandée, il faut donc préremplir les champs du formulaire avec les valeurs existantes dans la base de données. En effet, les données de la dernière station affichée sont stockées au moment où le serveur répond, suite au clic de l'utilisateur sur le marqueur d'une station. Il suffit donc d'utiliser ces données.

Particularités des créations de stations

Quant à la procédure de création, celle-ci se déclenche au moment où l'utilisateur clique sur le bouton symbolisé par un signe '+'. Le choix lui est alors laissé entre la création d'une station électrique et celle d'une station thermique ou bien d'annuler sa création.

Afin de pouvoir placer la station sur la carte, il est également imposé à l'utilisateur d'ajouter dans le formulaire d'ajout, les valeurs de latitude et longitude de la station qu'il souhaite créer.

Fonctionnement commun

Bien que la présence de la plupart des champs du formulaire soient binaires, c'est-à-dire qu'ils peuvent être soit présents, soit absents, les données relatives aux prises et aux services peuvent être présentes un nombre variable de fois, voire même absentes, ce qui est difficile à contrôler. Pour remédier à cela, nous avons mis en place une stratégie simple : un champ texte permet à l'utilisateur de saisir le libellé du service ou de la prise, et de l'ajouter à une liste représentée par un menu déroulant en cliquant sur un bouton dédié. De plus, l'utilisateur a la possibilité de supprimer un ou plusieurs services ou types de prise en les sélectionnant et en cliquant sur un autre bouton. Grâce à ce mécanisme, il n'y a aucune limitation quant au nombre d'ajouts ou de suppressions possibles. Ce mécanisme est exemplifié en figure 3.5 où l'on peut effectivement observer les différents champs.



FIGURE 3.5 – Exemple de modification ou de création des services

Si l'utilisateur souhaite ne pas renseigner de valeurs pour un champ, c'est possible. Il existe alors deux possibilités : si c'est un champ Texte, il suffit de laisser vide, s'il s'agit d'un champ Number, alors donner une valeur nulle équivaut à ne pas renseigner de valeur.

Une fois que l'utilisateur estime en avoir terminé avec sa modification ou sa création, il envoie sa demande au serveur en cliquant sur un bouton. Si tout est conforme (latitude et longitude présentes notamment) alors la demande est ajoutée en BDD par le serveur dans la table ModificationCreationStation, prête à être traitée par un administrateur.

3.3 Statistiques

L'analyse statistique des données constitue une partie essentielle du projet, visant à faciliter la compréhension et la visualisation des données. Pour ce faire, plusieurs techniques statistiques ont été mises en place.

Concernant les carburants, des statistiques spécifiques ont été créées pour suivre l'évolution des prix pour chaque station, accessibles en cliquant sur chaque station carburant de la carte. De plus, un calcul de la moyenne de tous les prix de carburant (gazole, SP95, SP98, E10, E85, GPLc) a été mis en place. Pour visualiser ces données de manière claire et interactive, l'application utilise la bibliothèque JavaScript Chart.js, qui permet de zoomer sur le graphique et d'obtenir le prix précis en passant la souris sur un point de la courbe.

Les données électriques et carburants disponibles dans la base permettent également d'accéder à la région d'implantation des stations, ce qui a conduit à la création d'une carte interactive. Cette carte utilise un fond de carte fourni par Leaflet avec une surcouche GeoJSON pour représenter les régions. Chaque région est colorée en fonction du nombre de stations qu'elle contient.

En ce qui concerne les statistiques liées aux véhicules électriques, nous avons élaboré un graphique visualisant les données fournies par Enedis, qui documentent l'évolution du nombre de stations électriques par trimestre ainsi que leur répartition d'accessibilité entre accès public, privé ou réservé aux entreprises. De plus, les données sur le nombre de voitures électrique par commune depuis le 1er trimestre 2020 ont permis la création de deux cartes interactives sur le même principe que les précédentes, mais avec une surcouche GeoJSON par département. La première carte représente la répartition des voitures électriques immatriculée par département (avec un détail du nombre de voitures par commune disponible en cliquant sur un département). La seconde carte illustre l'évolution du nombre de voitures électriques immatriculé par département (avec le détail du nombre de voitures électriques présentes en 2020 et 2023 accessibles en cliquant sur le département). Pour mieux comprendre la répartition de la puissance des prises de recharge électrique, un graphique présentant le nombre de prises par puissance est également disponible.

3.4 Comptes utilisateurs

Parmi les objectifs du projet, il semblait essentiel de garantir une expérience utilisateur complète et enrichissante. Dans cette optique, l'implémentation d'une fonctionnalité de compte utilisateur s'est avérée être une solution pertinente et cohérente. Cette fonctionnalité vise à offrir aux utilisateurs récurrents une plateforme plus interactive et personnalisée. Les comptes personnels permettent notamment aux utilisateurs de marquer des stations

comme favorite, et de partager leurs expériences en commentant et en notant les stations visitées. Cette initiative vise à renforcer l'engagement des utilisateurs et à favoriser un partage d'informations et d'expériences plus dynamique au sein de la plateforme.

3.4.1 Page connexion

Lors de la première utilisation, un utilisateur a la possibilité de s'inscrire en fournissant des informations telles qu'un nom d'utilisateur, une adresse e-mail et un mot de passe. Des contrôles sont effectués sur le nom d'utilisateur et l'adresse e-mail pour garantir leur conformité (minimum de caractères requis, absence de caractères spéciaux) et leur unicité afin d'éviter tout conflit entre les différents comptes utilisateur. Le mot de passe est sécurisé en étant "hashé" à l'aide de la bibliothèque bcrypt.

Une fois ces contrôles réalisés, pour assurer la sécurité et l'intégrité du processus d'inscription, tant pour l'utilisateur que pour l'application, un e-mail de vérification est envoyé pour finaliser l'inscription.

Lors de l'insertion du compte dans la base de donnée, un uuid⁶ associé à ce compte est généré. Lorsqu'un utilisateur est connecté, l'uuid associé à son compte est stocké dans le sessionStorage ou bien dans le localStorage. Cela permet de récupérer toutes les informations concernant l'utilisateur. En effet, l'uuid permet de ne pas stocker les informations de l'utilisateur en clair dans le navigateur, dans quel cas celui-ci aurait pu les changer et prendre par exemple l'identité d'un autre. Une multitude de fonctions permettant donc de récupérer des informations quant à un utilisateur depuis un uuid sont définies dans la bibliothèque `biblio.js`.

Lorsqu'un utilisateur est déjà inscrit, il peut se connecter en utilisant son adresse e-mail et le mot de passe défini lors de l'inscription. De plus, l'utilisateur a accès à une fonctionnalité lui permettant de réinitialiser son mot de passe en cas d'oubli. Ce processus, similaire à celui de l'inscription, commence par l'envoi d'un e-mail à l'utilisateur pour vérifier son identité avant de permettre le changement de mot de passe.

Lorsque l'utilisateur clique sur le lien contenu dans le mail, une page s'ouvre. Dans celle-ci se trouve un formulaire demandant à l'utilisateur son nouveau mot de passe ainsi qu'une confirmation de celui-ci.

Afin de faciliter l'expérience utilisateur, une option "se souvenir de moi" est également disponible. Cette option détermine si l'uuid de l'utilisateur doit être stocké dans le sessionStorage ou dans le localStorage. En cochant cette option, l'utilisateur peut rester connecté lors de ses prochaines visites, évitant ainsi d'avoir à saisir à nouveau ses informations de connexion à chaque utilisation. Cette fonctionnalité vise à rendre l'accès à la plateforme plus fluide pour les utilisateurs réguliers.

3.4.2 Fonctionnalités apportées

Lorsqu'un utilisateur consulte les informations d'une station, deux fonctionnalités facultatives sont mises à sa disposition. Il peut tout d'abord ajouter la station à ses favoris, lui permettant ainsi de la retrouver plus facilement ultérieurement, ainsi que la possibilité d'ajouter un commentaire sur la station pour partager son expérience ou ses impressions.

Ces fonctionnalités sont visibles par tous les utilisateurs, mais seuls ceux qui sont connectés peuvent les utiliser. Lors de l'ajout d'une station en favoris, son IDStation est

6. Universally Unique Identifier

ajouté dans un tableau contenant tous les ID des stations favorites de l'utilisateur. Les stations ajoutées en favoris sont ainsi accessibles depuis la page de compte de l'utilisateur, lui permettant ainsi de retrouver simplement toutes ses stations favorites.

Un utilisateur ne peut publier qu'un seul commentaire par station. Lors de la publication d'un commentaire, l'utilisateur doit obligatoirement attribuer une note à la station. De manière facultative, l'utilisateur peut ajouter un texte supplémentaire pour fournir des détails ou des observations plus approfondies.

Si un utilisateur est connecté et qu'un commentaire lui appartient, il peut alors le supprimer à tout moment.

3.4.3 Page compte

La page compte regroupe l'ensemble des fonctionnalités définies précédemment. Tout d'abord, elle permet à l'utilisateur de modifier ses informations personnelles telles que son adresse e-mail et son login, deux éléments importants respectivement pour la connexion et l'identité associée aux commentaires.

Ensuite, la page compte comporte un onglet permettant à l'utilisateur de visualiser les commentaires qu'il a postés. L'utilisateur peut également supprimer ses propres commentaires depuis cette interface. De plus, un lien cliquable "consulter" est disponible pour chaque commentaire, redirigeant l'utilisateur vers la carte avec la fenêtre de la station concernée. Lors de la redirection, nous avons pris la décision de laisser la carte vierge avec simplement la fenêtre de la station concernée, pour éviter toute complication liée aux filtres non nécessaires.

De même, la page compte offre un onglet permettant à l'utilisateur de visualiser les stations qu'il a ajoutées en favoris. Les noms des stations sont affichés, accompagnés des options pour consulter et supprimer les stations de la liste de favoris.

Enfin, un bouton de déconnexion est également disponible si l'utilisateur le souhaite, offrant ainsi un moyen simple de mettre fin à sa session.

3.5 Pages administrateurs

3.5.1 Protection d'accès

Les pages administrateurs se doivent, comme leur nom l'indique, d'être accessibles uniquement par les administrateurs. Le lien depuis la page d'accueil ne s'affichera donc que si l'utilisateur est connecté et qu'il s'agit d'un compte administrateur. Pour ce faire, nous utilisons un objet localStorage stockant l'uuid associé à l'utilisateur courant. Celui-ci nous permet de vérifier, chaque fois que nécessaire, si le compte courant est un compte administrateur grâce à une requête vers la base de données.

Comme protection supplémentaire, nous avons ajouté le fait que l'utilisateur soit directement redirigé vers l'accueil si jamais il n'est pas administrateur et qu'il accède à une de ces pages (par exemple en tapant directement l'URL de la page). Nous réalisons également cette vérification en amont lors d'une suppression ou modification en base de données.

3.5.2 Gestion créations/modifications

Le scénario est simple : lorsqu'un administrateur se connecte à la page de gestion de création et modification de station, celui-ci lui demande les informations présentes en base de données (par défaut les modifications, tout en sachant qu'il est possible de passer du mode "voir les modifications" au mode "voir les créations" et vice-versa en cliquant sur un bouton).

Dans le cas des modifications, nous récupérons pour chaque proposition de modification les données présentes dans la table Station (ie les données actuelles), en effet, cela permet d'afficher à l'écran de l'administrateur d'une part les données actuelles et d'une autre ce qui est proposé dans la modification. L'administrateur peut changer la provenance de ce qui est affiché en cliquant sur un bouton. Ceci lui permet de savoir ce qui a été modifié, ajouté ou supprimé.

Pour les créations, il n'y a aucune donnée relative à cette station en BDD. Sont donc affichées uniquement les propositions de créations données par l'utilisateur.

L'administrateur peut accepter, rejeter ou simplement ne pas traiter une modification ou une création.

Afin de pouvoir recharger les demandes sans avoir besoin de recharger la page directement et ainsi conserver le choix de création ou modification, une flèche a été ajoutée, un click sur cette flèche transmet la demande au serveur qui renvoie les données présentes actuellement en BDD.

De par notre volonté d'interface interactive et claire, nous avons eu l'idée de réaliser des cartes qui défilerait horizontalement. L'administrateur pouvant choisir de les parcourir de gauche à droite ou de droite à gauche. Il y a donc une carte centrale, avec laquelle l'administrateur pourra interagir, et deux autres présentes de chaque côté, où l'on pourra percevoir d'autres demandes de modification.

Il fallait donc pouvoir gérer à la fois le mode création et le mode modification, et pour ce dernier, le fait d'afficher soit la demande, soit le contenu actuel présent en BDD. De plus, les structures des tables ModificationCreationStation et Station étant très proches, conserver les données telles qu'elles étaient présentes en BDD pouvait être un avantage. Cependant, cette présentation des données n'est absolument pas quelque chose de présentable et de digeste pour l'administrateur, il fallait conserver le format d'origine des données, mais également les formater à la vue administrateur. Interroger le serveur à chaque mouvement pouvait être quelque chose d'envisageable, mais cela aller poser des problèmes de performances, et d'ordonnancement quant à l'affichage des infos.

Nous avons donc décidé de créer trois tableaux. Deux pour les demandes de modifications/créations et un dernier dans le cas des modifications, afin de conserver ce qui était présent actuellement en bdd. Tous les tableaux ont la même taille, et pour le même indice, désignent chacun les infos relatives à la même station. Dans le but d'éviter de devoir formater les données à chaque fois qu'il fallait les afficher et pour conserver les données, nous utilisons deux des trois tableaux : un contenant les données présentes dans la table ModificationCreationStation et l'autre contenant un élément HTML contenant toutes ces mêmes infos, mais formatées à l'affichage utilisateur. Pour le dernier tableau, il s'agit également de données formatées relatives à ce qui est actuellement présent dans la table Station. Une variable globale, faisant office d'indice, est incrémentée/décrémentée de façon à ce que l'indice reste toujours valide et corresponde à la carte principale, et ce, dès qu'un défilement ou une suppression a lieu.

Côté BDD, il fallait pouvoir créer un identifiant non utilisé pour chaque nouvelle création, car le champ n'est pas en auto-incrémentation dans la bdd de par les contraintes expliquées précédemment. Pour ce faire, nous avons choisi un nombre suffisamment grand pour ne pas altérer les données déjà présentes : 20 000 000 000. Puis, nous récupérons le plus grand identifiant supérieur à cette valeur, et s'il existe, nous l'enregistrons et partons de cette valeur pour la création de nouvel ID. C'est ce que fait la fonction `getGreaterID()`, issue du fichier `index.js` et présentée en figure 3.6. Cette opération s'effectue à chaque démarrage du serveur.

```

1. function getGreaterID() {
2.     let IDMiniCrea = 2000000000;
3.     let sqlID = "SELECT MAX(IDStation) AS IDMax FROM Station WHERE IDStation > "+IDMiniCrea;
4.
5.     requeteBDD(sqlID)
6.     .then(results => {
7.         if(results[0]['IDMax']){
8.             IDMiniCrea = results[0]['IDMax'];
9.         }
10.    })
11.    .catch(error => {
12.        console.error("Erreur : ", error);
13.    });
14.
15.
16.    return IDMiniCrea;
17. }
```

FIGURE 3.6 – Fonction `getGreaterID()`

Quelle que soit la décision de l'administrateur (accepter ou rejeter), il faut supprimer les données des tables `ModificationCreationStation` et `ModificationCreationCarburant` ou `ModificationCreationPrise`, selon le type de station. Pour ce faire, il suffit d'exécuter une requête SQL de `DELETE` très simple.

3.5.3 Gestion comptes

Cette page permet de visualiser l'ensemble des comptes créés, de savoir lesquels sont administrateurs ou non, de connaître les logins, mais également de supprimer un ou plusieurs comptes. À savoir que, lorsqu'un compte est supprimé tous les commentaires postés par cet utilisateur sont aussi supprimés. Il est aussi possible de changer les permissions d'un compte utilisateur en administrateur ou bien l'inverse. Cependant, il est impossible de supprimer ou de modifier les infos de son propre compte.

Les comptes sont triés par ordre alphabétique selon les logins, et pour pouvoir trouver un compte plus rapidement, il existe une fonction recherche. Celle-ci permet d'entrer une chaîne et d'afficher uniquement les logins commençant par cette chaîne. Cette recherche est réalisée grâce à des requêtes SQL de la forme suivante, où `pattern` représente la chaîne rentrée par l'utilisateur :

```
SELECT * FROM COMPTE WHERE Login LIKE pattern%
```

3.5.4 Gestion BDD

La gestion de la base de données occupe une place centrale dans notre projet, étant indispensable pour stocker et gérer efficacement les données nécessaires au bon fonctionnement de l'application. Afin de simplifier cette gestion, nous avons développé une interface d'administration de la BDD.

Cette interface agit comme un intermédiaire entre le serveur et l'administrateur. Elle permet d'effectuer diverses opérations cruciales pour la maintenance et la gestion de la base de données. Parmi ces opérations, nous retrouvons l'exportation, la suppression, l'affichage (limité aux 10 premières lignes) et l'ajout de tables, offrant ainsi une flexibilité totale dans la manipulation de la structure de la BDD.

En plus de cela, cette interface offre la possibilité d'effectuer des mises à jour manuelles sur des tables spécifiques, telles que CommuneData, l'historique des prix des carburants et les tables des stations (électriques et carburant).

4 Déploiement de l'application

À l'heure actuelle, la présence en ligne pour des sites web est essentielle pour se faire connaître. Notre objectif à travers le déploiement de ce projet est de se familiariser avec le principe de déploiement, de comprendre ce qu'est une VM, un serveur Apache, ainsi que les fichiers de configuration associés. Pour rappel, Le déploiement d'une application consiste à mettre en place et de rendre fonctionnelle l'application sur un serveur, lui permettant ainsi d'être utilisée par les utilisateurs finaux.

4.1 Connexion et utilisation d'une machine virtuelle

Dans notre contexte d'étudiants, la faculté nous fournit une machine virtuelle. Cette machine virtuelle est un environnement informatique autonome, isolé du système hôte, qui fonctionne comme un véritable ordinateur avec son propre système d'exploitation (Debian 11) et ses propres ressources. Cette machine virtuelle est hébergée sur le réseau privé de l'université, ce qui signifie qu'elle n'est pas directement accessible depuis l'extérieur de ce réseau.

Pour accéder à cette machine virtuelle depuis un emplacement extérieur au réseau de l'université, il est nécessaire d'utiliser un VPN¹. Un VPN permet de créer une connexion entre l'ordinateur local de l'utilisateur et le réseau privé de l'université, ce qui permet d'accéder à la machine virtuelle comme si l'utilisateur était physiquement connecté au réseau interne de l'université.

Pour accéder à cette machine virtuelle, nous utilisons le protocole SSH (Secure Shell), un protocole de communication sécurisé qui permet de se connecter à distance à la machine virtuelle à partir de son propre ordinateur. L'utilisation du terminal permet de se connecter en spécifiant l'adresse IP de la machine virtuelle et les identifiants d'utilisateur.

```
ssh etudiant@172.20.128.218
```

Une fois connectés à la machine virtuelle via SSH, il est possible de transférer des fichiers entre son ordinateur local et la machine virtuelle en utilisant le protocole SCP (Secure Copy). SCP est une commande intégrée au protocole SSH qui permet de copier des fichiers de manière sécurisée entre deux systèmes distants. Par exemple, pour copier un fichier de son ordinateur local vers la machine virtuelle, nous utilisons la commande suivante dans notre terminal local :

```
scp chemin/du/fichier utilisateur@adresse_IP_
machine_virtuelle:chemin_de_destination
```

L'utilisation de ces outils permet d'interagir simplement avec la machine virtuelle, de transférer des fichiers et d'effectuer toutes les opérations nécessaires pour déployer l'application.

1. Virtual Private Network.

4.2 Serveur web et installation

Nous avons choisi Apache en raison de sa popularité et de sa documentation abondante. Apache est largement utilisé et dispose de nombreux tutoriels disponible sur diverses plateformes telles que YouTube ou des sites communautaires. Cette disponibilité de ressources facilite l'apprentissage et la résolution de problèmes, ce qui en fait un choix fiable pour notre projet. Pour l'installer, nous avons mis à jour notre VM et installé apache via ces *commandes*² :

```
apt update  
apt upgrade  
apt-get install apache2
```

De plus, pour faciliter l'édition des fichiers de configuration, nous avons également installé l'éditeur de texte Vim ainsi que Git pour cloner le projet ainsi que mariadb pour générer la base de données.

Nous avons donc cloné le projet sur la VM et récupéré les fichiers .sql via le protocole SCP. Pour les intégrer dans la base de données, nous avons commencé par nous connecter avec

```
mysql -u root -p
```

Nous avons ensuite crée notre database et utiliser celle-ci

```
CREATE DATABASE ProjetL3;  
USE ProjetL3;
```

Et nous avons importé les fichiers sql via la commande

```
mysql -u root -p ProjetL3 < chemin_vers_le_fichier.sql
```

4.3 Configuration

Avant d'entrer dans les détails de la configuration d'Apache, prenons un moment pour comprendre son fonctionnement.

Apache utilise des ports pour écouter les demandes de connexion entrantes. Par défaut, le port 80 est utilisé pour le trafic HTTP, tandis que le port 443 est réservé au trafic HTTPS sécurisé. Ces ports servent de points d'entrée pour les requêtes HTTP ou HTTPS envoyées par les navigateurs web.

Lorsqu'une requête arrive, Apache utilise le concept de répertoire racine pour déterminer où trouver les fichiers à servir. Sur les systèmes Linux, par exemple, le répertoire racine est généralement situé à /var/www/. C'est à partir de ce répertoire que le serveur web recherche les fichiers demandés par les clients.

2. Les commandes qui suivent sont exécutés en tant que super utilisateur.

Cependant, nous avons rencontré un défi supplémentaire : l'application Node.js fonctionne sur le port 8080, tandis qu'Apache écoute par défaut sur le port 80.

Pour résoudre ce problème, nous avons utilisé le module de proxy d'Apache pour rediriger le trafic provenant du port 80 vers le port 8080, où l'application Node.js est en cours d'exécution. En parallèle, nous avons modifié le répertoire racine d'Apache pour qu'il pointe vers le répertoire contenant les fichiers statiques de notre application.

Ainsi, lorsque les utilisateurs accèdent au site via leur navigateur, Apache intercepte les requêtes sur le port 80 et les redirige vers le port 8080 où l'application Node.js est en écoute. Ensuite, l'application Node.js gère les réponses appropriées, qui sont ensuite renvoyées par Apache aux clients. Cette configuration permet aux utilisateurs de se connecter au site en utilisant simplement l'adresse IP sans avoir besoin de spécifier un port.

Cela est fait comme ceci dans la configuration apache

```
1. root@cr700-vfele-a-projet-01:/# cat /etc/apache2/sites-available/000-default.conf
2. <VirtualHost *:80>
3.   ServerAdmin projet13.noreply@gmail.com
4.   DocumentRoot /var/www/projet13
5.
6.   ErrorLog ${APACHE_LOG_DIR}/error.log
7.   CustomLog ${APACHE_LOG_DIR}/access.log combined
8.
9.   ProxyPass / http://127.0.0.1:8080/
10.  ProxyPassReverse / http://127.0.0.1:8080/
11. </VirtualHost>
12.
13.
14.
15. root@cr700-vfele-a-projet-01:/# cat /etc/apache2/sites-available/projet13.conf
16. <VirtualHost *:8080>
17.
18.   ServerAdmin projet13.noreply@gmail.com
19.   DocumentRoot /var/www/projet13
20.
21.
22.   ErrorLog ${APACHE_LOG_DIR}/error.log
23.   CustomLog ${APACHE_LOG_DIR}/access.log combined
24.
25.   <Directory /var/www/projet13>
26.     Options -Indexes +FollowSymLinks
27.     AllowOverride All
28.   </Directory>
29. </VirtualHost>
```

FIGURE 4.1 – Configuration Apache

Pour organiser la configuration des sites web, Apache utilise les répertoires sites-available et sites-enabled. Dans sites-available, sont stockées les configurations de tous les sites web disponibles, tandis que sites-enabled contient les liens symboliques vers les configurations actives. Cette structure facilite la gestion des différents sites hébergés sur le serveur.

Pour activer un site, nous utilisons la commande a2ensite suivie du nom du fichier de configuration du site, comme ceci :

```
a2ensite nom_du_site.conf
```

Pour désactiver un site, nous utilisons la commande a2dissite suivie du nom du fichier de configuration du site, comme ceci :

```
a2dissite nom_du_site.conf
```

Après avoir activé ou désactivé un site, il est nécessaire de recharger la configuration d'Apache pour que les changements prennent effet :

```
systemctl reload apache2
```

Afin de garantir le bon fonctionnement des applications Node.js, nous utilisons le package forever (installé via apt-get). Forever permet de relancer automatiquement une application en cas de plantage, assurant ainsi un fonctionnement continu des applications Node.js.

De plus, nous avons mis en place des scripts shell de démarrage et d'arrêt du serveur Node.js. Ces scripts peuvent contenir des commandes pour démarrer l'application avec forever.

Script de démarrage du serveur Node.js :

```
#!/bin/bash

# Chemin vers l'application Node.js
APP_PATH="index.js"

# Chemin vers les fichiers de logs
LOG_PATH="/var/www/projet13/logProjet13"
OUT_LOG="$LOG_PATH/out.log"
ERR_LOG="$LOG_PATH/err.log"
LOG_FILE="$LOG_PATH/log.txt"

# Créer le répertoire des logs s'il n'existe pas
mkdir -p "$LOG_PATH"

# Démarrer l'application avec Forever et rediriger les logs vers
# les fichiers
forever start --append -l "$LOG_FILE" -o "$OUT_LOG" -e "$ERR_LOG"
"$APP_PATH"

# Vérifier le statut de l'application
forever list
```

Script d'arrêt du serveur Node.js :

```
#!/bin/bash

# Arrêter le processus Forever en utilisant l'index
forever stop index.js
```

```
# Vérifier que le processus a bien été arrêté  
forever list
```

4.4 bilan du déploiement

L'un des principaux problèmes rencontrés était lié à une MTU (Maximum transmission unit) incorrecte, découlant de la mauvaise configuration du VPN. Cette configuration erronée a entraîné des problèmes de connectivité à la machine virtuelle. Après avoir identifié la source d'erreur, le problème a pu être résolu en ajustant la configuration de la MTU pour résoudre ces problèmes de connectivité et poursuivre le déploiement.

En dépit de cette difficulté majeur, le déploiement a abouti à un succès, avec une application fonctionnelle accessible aux utilisateurs qui dispose du VPN de l'université.

Conclusion

Pour conclure, ce projet a été une véritable expérience qui nous a permis de mettre en pratique les connaissances acquises au cours de notre formation, tout en arrivant à compléter les objectifs initialement fixés. En nous plongeant dans la réflexion puis la réalisation d'un premier projet d'envergure, nous avons pu non seulement développer nos compétences techniques, mais aussi découvrir les défis concrets auxquels nous confrontent les projets informatiques d'ampleur.

Sur le plan technique, ce projet nous a permis d'explorer diverses technologies, de la collecte initiale des données au déploiement final de l'application. Nous avons appris à nettoyer et à stocker les données pour les rendre exploitables, puis à concevoir et développer une architecture pour notre système et nous avons choisi le serveur web approprié et configuré notre environnement de déploiement pour effectuer une mise en production.

Nous avons pu réaliser l'importance de la collaboration au sein d'une équipe et ce que celle-ci apporte. La variété des compétences et des points de vue au sein de notre équipe a enrichi notre approche et a conduit à l'élaboration de solutions innovantes. La supervision et les conseils avisés de Mme Violeta FELEA ont également été utile dans la réussite de notre projet et dans l'accomplissement de tous nos objectifs. En guise d'évolutions possibles des fonctionnalités de l'application, l'ajout de signalement de commentaires ou de logins abusifs peut être intéressant afin de faciliter l'administration de la plateforme. Une nouvelle interface administrateur de gestion des commentaires pourrait ainsi être développée. De plus, la mise en place de comptes gérants de station permettrait d'assurer la véracité des informations de chaque station. De cette fonctionnalité pourrait découler la possibilité pour les gérants de répondre aux commentaires postés sur leur station, favorisant ainsi une meilleure interactivité.

Au-delà des compétences techniques acquises, ce projet nous a permis de développer des compétences transversales précieuses telles que la gestion du temps, la communication efficace et le travail en autonomie. Pour différentes parties du projet, les obstacles rencontrés nous ont permis de renforcer notre capacité d'adaptation et notre aptitude à résoudre les problèmes de manière autonome.

En fin de compte, cette expérience nous a non seulement permis de créer une plateforme fonctionnelle et conviviale, mais elle a par ailleurs éveillé notre sensibilité aux enjeux environnementaux et sociaux. Nous espérons que cette application contribuera à sensibiliser le public à la mobilité électrique et encouragera des changements positifs dans nos modes de déplacement.

En guise de bilan, ce projet restera pour nous une étape importante de notre formation, marquée par l'apprentissage, la collaboration et la découverte de l'impact concret de la technologie sur notre société.

Annexe

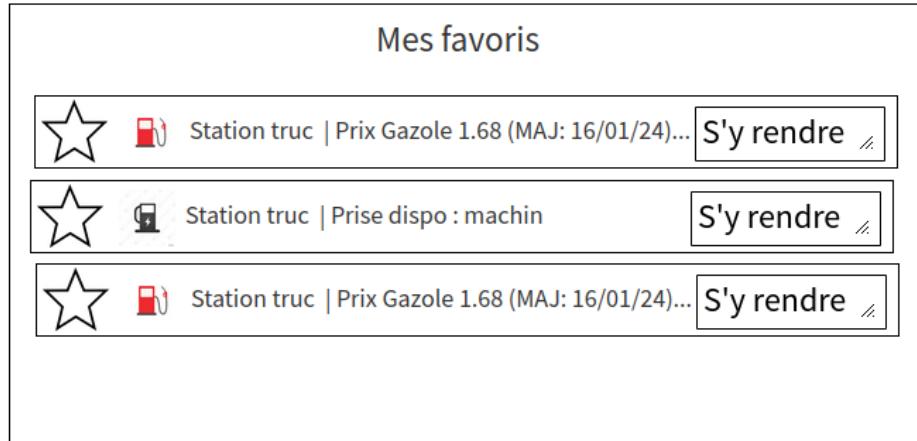


FIGURE 4.2 – Croquis des favoris

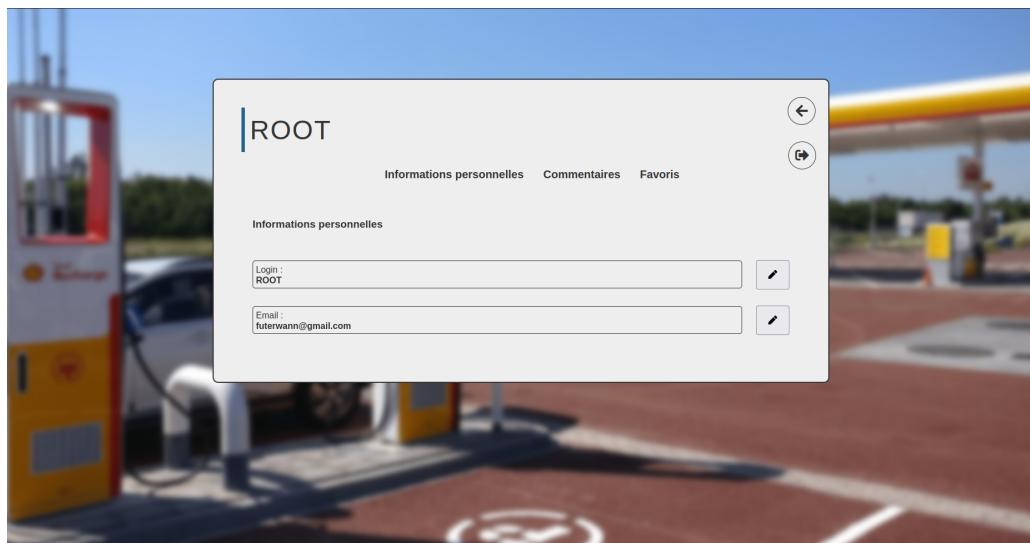


FIGURE 4.3 – Interface finale de la page compte

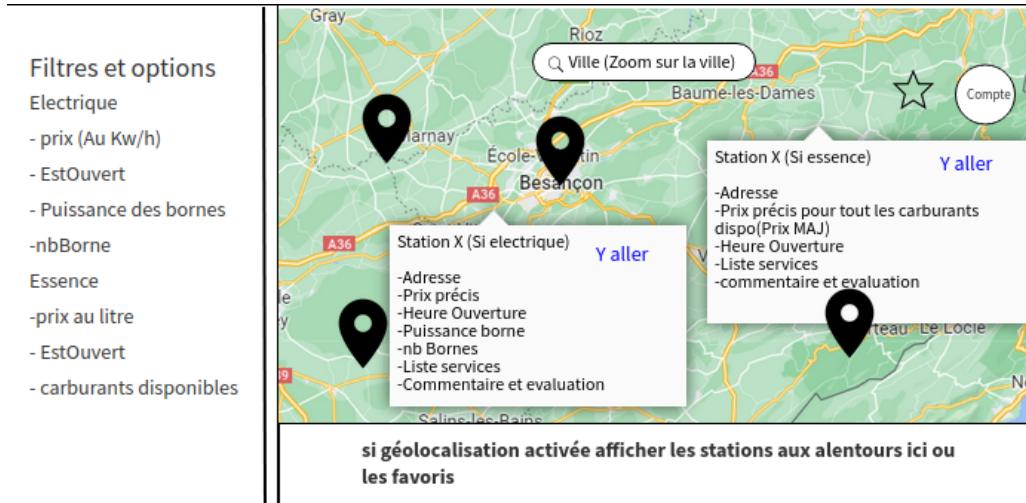


FIGURE 4.4 – Croquis de la carte

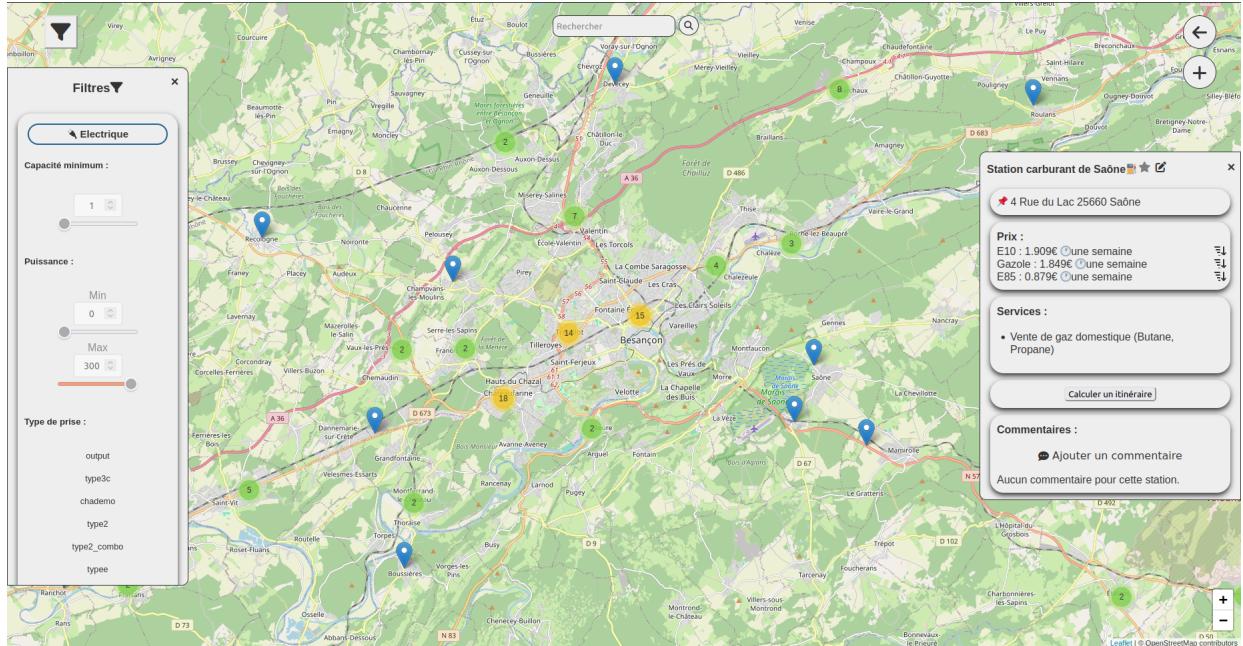


FIGURE 4.5 – Interface finale de la carte

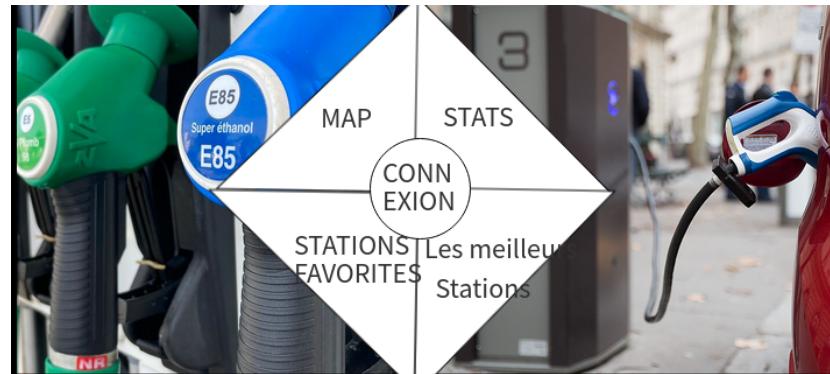


FIGURE 4.6 – Croquis de l'accueil



FIGURE 4.7 – Interface finale de l'accueil

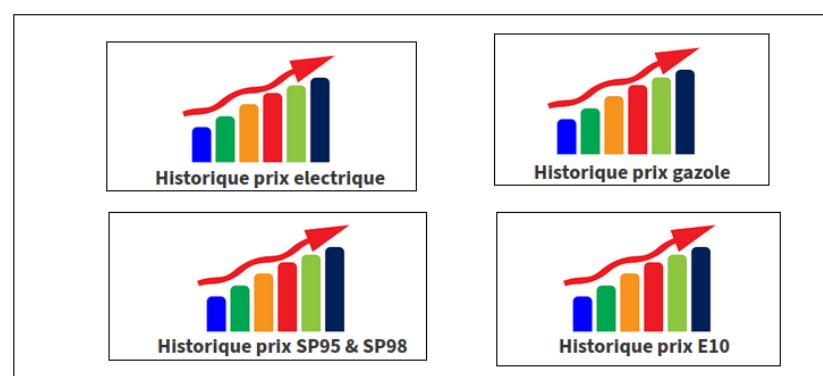


FIGURE 4.8 – Croquis des statistiques



FIGURE 4.9 – Interface finale des statistiques

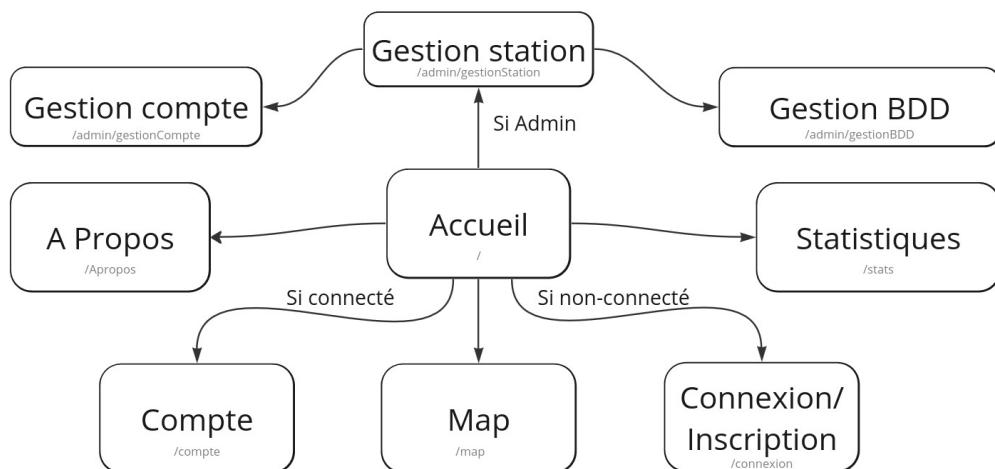


FIGURE 4.10 – Arborescence de l'application

```

6   "features": [
7     {
8       "type": "Feature",
9       "properties": {
10         "@id": "node/5321979984",
11         "amenity": "charging_station",
12         "operator": "Colruyt"
13       },
14       "geometry": {
15         "type": "Point",
16         "coordinates": [
17           5.5019562,
18           47.0977157
19         ]
20       },
21       "id": "node/5321979984"
22     },
23     {
24       "type": "Feature",
25       "properties": {
26         "@id": "node/5687122825",
27         "amenity": "charging_station",
28         "capacity": "2",
29         "fee": "yes",
30         "motorcar": "yes",
31         "network": "SRM2",
32         "opening_hours": "24/7",
33         "operator": "Freshmile",
34         "owner": "SYDED",
35         "parking:fee": "no",
36         "ref:EU:EVSE": "FR*525*P02502149",
37         "socket:type2": "2",
38         "socket:type2:output": "18kW",
39         "socket:typeee": "2",
40         "socket:typeee:output": "2kW",
41         "source": "data.gouv.fr:Etalab - 01/2020"
42       },
43       "geometry": {
44         "type": "Point",
45         "coordinates": [
46           5.7839204,
47           47.0348689
48         ]
49       },
50       "id": "node/5687122825"
51     },
52     {
53       "type": "Feature",
54       "properties": {
55         "@id": "node/5893310910",
56         "access": "customers",
57         "addr:city": "Chemaudin et Vaux",

```

FIGURE 4.11 – Structure des données open source map

Résumé

La troisième année de Licence informatique à l'université de Franche-Comté propose pour son sixième semestre une unité d'enseignement nommé Projet qui, comme son nom l'indique, consiste en la réalisation d'un projet semestriel. Le projet qui nous a été attribué consistait en la création d'une carte de France permettant de visualiser les emplacements des stations électrique mais également de proposer des statistiques quant à elles. Sujet que nous avons étendu aux stations thermiques. Ce rapport témoigne de notre démarche quant à l'élaboration de ce projet qui prendra la forme d'une application web utilisant JavaScript et l'environnement d'exécution NodeJS côté serveur.

Mots clés

Application web, NodeJS, Leaflet, sécurité, base de données, carte des bornes de recharge électrique, carte des stations thermiques, statistiques, réchauffement climatique, transition énergétique.

Abstract

The third year of the Computer Science Bachelor's program at the University of Franche-Comté includes a Project unit in its sixth semester. As the name suggests, this involves a semester-long project. Our assigned topic focused on creating a map of France displaying locations of both electrical and thermal stations, along with relevant statistics. This report outlines our approach to this project, which took the form of a JavaScript web application using server-side NodeJS.

Keywords

Web application, NodeJS, Leaflet, security, database, gas station map, map of electric charging stations, statistics, global warming, energetic transition.