

Camille Céleste Covarel

projet

watts up

watts-up.anamnesis.ovh

WILD CODE SCHOOL Toulouse

dossier projet

SOMMAIRE

2 Sommaire

3 - 4 Introduction

5 Liste des compétences

Analyse et conception

5 - 6 les besoins fonctionnels

6 - 7 analyse de la source de données

7 les besoins non-fonctionnels

8 - 11 maquette de l'application, ui et ux

12 - 13 gestion de projet et organisation

l'Environnement technique & le cœur du Backend

14 architecture technique globale

15 analyse de la stack technique

16 modélisation de la BDD

18 - 23 réalisation Backend 1 · le traitement de données

24 - 27 réalisation Backend 2 · logique métier et automatisation

Frontend et expérience utilisateur

28 - 30 réalisation Frontend 1 · architecture adaptative et accessibilité

31 - 32 réalisation Frontend 2 · logique métier et cycle de vie

33 - 34 réalisation Frontend 3 · interface d'administration et visualisation

Jeu d'essai · validation et limites de l'importation

35 scénario 1, le cas nominal succès sur données conformes

36 scénario 2, résilience aux données "sales" gestion d'erreurs

37 scénario 3, mise à jour de données update

38 - 39 scénario 4, cycle de vie de la réservation actif

40 - 41 scénario 5, début et fin de charge

42 - 43 Déploiement et industrialisation

Méthodologie organisation et veille

44 - 45 gestion de projet et travail collaboratif

46 - 47 la veille technologique les choix architecturaux

48 - 49 perspectives d'évolution

50 bilan du projet

I · INTRODUCTION

I · contexte du projet

Notre projet démarre avec la demande d'un prototype d'application web visant à créer une carte interactive à destination des utilisateurs de véhicules électriques.

L'objectif est de disposer d'un MVP fonctionnel pour notre présentation auprès de la société (fictive) Géocode, avec pour vocation d'être exportée en application mobile."

Ce projet vise à développer une application qui permet aux utilisateurs finaux de réserver une bornes de recharge pour leur véhicule de façon simple et localisée. Au travers d'une carte, l'utilisateur est invité à choisir sa station et ensuite sa borne.

L'utilisateur va devoir créer son compte et fournir ses informations personnelles ainsi que les informations de son véhicules.

Une page d'information est présente pour l'aider dans ces démarches. Ainsi que plusieurs indications lors de la procédure de création de compte.

Les administrateurs doivent pouvoir mettre à jour la BDD des stations/bornes à partir d'un fichier CSV.

II · la candidate

Graphiste et photographe de formation, je me lance dans une reconversion en tant que développeuse en mars 2025.

Ayant un intérêt spécifique pour la gestion de serveurs et le monde du devops depuis plusieurs années, cette formation est pour moi, un tremplin pour bien comprendre les besoins des devs (et pour dev mes propres applications). Toutefois, ayant fait de la chefferie de projet numérique quand j'étais en poste en tant que graphiste, j'ai une certaine vision de comment doit être gérée une demande client.

III · Liste des compétences

Les compétences du titre sont remplies par ces éléments

Activité type 1 · développer la partie Frontend

- | | |
|---|---|
| Maquetter une application | » Conception de l'expérience utilisateur UX parcours de réservation et d'administration.
» Réalisation des wireframes et maquettes graphiques avec une approche Mobile First pour assurer l'ergonomie sur tous supports. |
| Réaliser une interface utilisateur web statique et adaptable | » Intégration des interfaces en HTML5 / CSS3 / TS.
» Mise en œuvre d'une méthodologie CSS pour garantir la gestion du Responsive Design <i>adaptation fluide mobile, tablette, desktop.</i> |
| Développer une interface utilisateur web dynamique | » Développement d'une SPA complète avec React.
» Gestion de contexte avec ReactContext
» Gestion du cache avec Tanstack Query
» Notifications temps réel WebSockets. |

Activité type 2 · développer la partie Backend

- | | |
|---|--|
| Créer une base de données | » Conception du Schéma Conceptuel <i>MCD</i> et Logique <i>MLD</i> .
» Implémentation d'une base relationnelle PostgreSQL.
» Intégration de l'extension spatiale PostGIS pour la gestion des coordonnées GPS.
» Gestion du versioning de la base <i>migrations</i> avec SequelizeCli. |
| Développer les composants d'accès aux données | » Développement d'une logique d'accès aux données en couches Services / Contrôleurs.
» Requêtes pour réservation d'une borne en station
» Validation stricte des données entrantes pour garantir l'intégrité. |
| Développer la partie Backend d'une application web | » Architecture d'une API REST avec Node.js et Express.
» Développement de scripts de traitement lourd <i>Import CSV via Streams</i> .
» Mise en place d'une authentification sécurisée JWT stocké en Cookie <i>HttpOnly</i> et gestion des droits Admin/User. |

II · ANALYSE ET CONCEPTION

I · les besoins fonctionnels

Voici le backlog tel qu'il a été reformulé au démarrage du projet

En tant que visiteur, je peux :

- highest ≈ **accéder à la page d'accueil**
page de présentation du site
- high ↗ **voir les bornes de recharges autour de moi**
page carte - Avant de laisser l'utilisateur visiter la page, récupérer la position GPS (latitude et longitude) si le navigateur le permet sinon l'inviter à saisir manuellement le nom de sa ville. La carte doit se recentrer sur la position précédemment définie.
- low ↘ **accéder à la page informations**
différents types de prises, conseils...

En tant qu'utilisateur, je peux :

- high ↗ **m'inscrire**
INFORMATIONS CLIENT - Nom, Prénom, Email, Sexe, Date de naissance (le site contenant des paiement, est réservé aux majeurs), code postal, ville (vérifier la cohérence avec le code postal), nombre de véhicules électriques, mot de passe (et sa confirmation).
INFORMATIONS VEHICULE(S) - Marque, Modèle, Type de prise
- high ↗ **me connecter**
Page de connexion
- high ↗ **me déconnecter**
Retour sur la page d'accueil en tant qu'utilisateur non connecté.
- high ↗ **réserver une borne**
Page borne, uniquement accessible depuis la carte si l'utilisateur dispose au minimum d'un véhicule, permettant de réserver une borne de recharge pour une durée de 30 minutes, un montant X devra être exigé pour procéder. (une seule borne par utilisateur).
- high ↗ **mettre à jour mon profil**
Page profil permettant de modifier ou supprimer son profil.
- high ↗ **être géolocalisé**
Je souhaite pouvoir choisir d'être géolocalisé pour que la carte affiche les bornes à proximité de ma position.
- medium = **récupérer mon mot de passe oublié**
Réinitialisation du mot de passe
- medium = **annuler une réservation**
Page profil, onglet "mes réservations" permettant d'annuler une réservation de borne de recharge. Le montant exigé à la réservation n'est pas restitué.
- medium = **filtrer les bornes sur la carte**
Pour n'afficher que les bornes qui correspondent au type de prises, puissance ou au type de véhicule.
- medium = **accéder à la page de contact**
Page de contact (formulaire contenant une liste déroulante : Demande d'informations, Demande de partenariat, Autres). Ces données doivent être persistées.

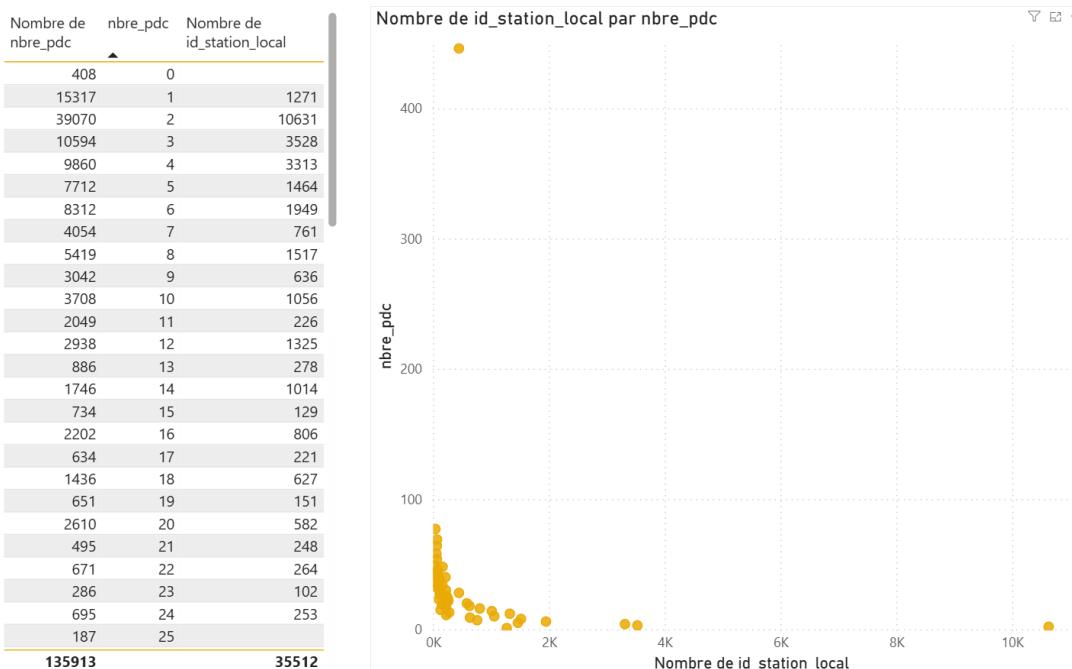
medium	=	ajouter une photo de profil Page profil, me permet d'ajouter, de modifier ou de supprimer une image de profil
low	▼	mettre à jour mon(mes) véhicule(s) Page véhicule permettant de modifier ou de supprimer mon ou mes véhicules.

En tant qu'administrateur, je peux :

highest	❖	accéder à une carte qui contient les bornes électriques Onglet de la page d'administration permettant de lire, modifier ou supprimer une borne de recharge
highest	❖	accéder au panel d'administration Page d'administration contenant toutes les informations annexes mettre à jour toute la liste des bornes électriques Onglet de la page d'administration permettant d'uploader un fichier CSV permettant de mettre à jour la liste complète des bornes.
medium	=	accéder aux statistiques du site Onglet de la page d'administration permettant de voir le nombre total d'utilisateurs inscrits, le nombre d'utilisateurs inscrits les 7 derniers jours, le nombre total de véhicules, le nombre total de véhicules par filtres (marques, modèles)
medium	=	accéder à la liste des utilisateurs Onglet de la page d'administration permettant de lire, modifier ou supprimer un utilisateur
low	=	accéder à la liste des véhicules Onglet de la page d'administration permettant de lire, modifier ou supprimer un véhicule

II · analyse de la source de données

À la lecture du backlog, j'ai décidé d'entamer un travail préparatoire sur les données. Je voulais comprendre les particularités que j'allais rencontrer avant d'écrire une seule ligne du script d'import. J'ai donc pris le temps de réaliser une analyse sur Power BI, dont voici les résultats



À première vue, la donnée semble relativement propre. Sur cette analyse, on peut voir un peu moins de 136k points de charge (PDC) pour environ 35.5k stations. Quelques aberrations statistiques sont toutefois présentes, comme ce point au-delà de 400 PDC. Certains PDC ne sont pas liés à des stations.

horaires	Nombre de horaires
24/7	103597
Mo-Su 00:00-23:57	17194
Mo-Su 00:00-24:00	8224
Mo-Fr 09:00-19:00	789
Mo 08:00-12:00, Mo 14:00-19:00, Tu 08:00-12:00, Tu 14:00-19:00, We 08:00-12:00, We 14:00-19:00, Th 08:00-12:00, Th 14:00-19:00, Fr 08:00-12:00, Fr 14:00-19:00, Sa 09:00-12:00, Sa 14:00-18:00	310
Mo-Su 06:00-01:00	194
Mo 00:00-23:59, Tu 00:00-23:59, We 00:00-23:59, Th 00:00-23:59, Fr 00:00-23:59, Sa 00:00-23:59, Su 00:00-23:59	173
Mo-Fr 07:30-12:00, Mo-Fr 14:00-19:00	155
Mo 00:00-23:59, Tu 00:00-23:59, We 00:00-23:59, Th 00:00-23:59, Fr 00:00-23:59	148
Mo 08:00-19:00, Tu 08:00-19:00, We 08:00-19:00, Th 08:00-19:00, Fr 08:00-19:00, Sa 09:00-18:00	85
Mo 08:00-12:00, Mo 14:00-19:00, Tu 08:00-12:00, Tu 14:00-19:00, We 08:00-12:00, We 14:00-19:00, Th 08:00-12:00, Th 14:00-19:00, Fr 08:00-12:00, Fr 14:00-19:00, Sa 08:00-12:00, Sa 14:00-19:00	79
Mo 09:00-12:00, Mo 14:00-19:00, Tu 09:00-12:00, Tu 14:00-19:00, We 09:00-12:00, We 14:00-19:00, Th 09:00-12:00, Th 14:00-19:00, Fr 09:00-12:00, Fr 14:00-19:00, Sa 09:00-12:00, Sa 14:00-18:00	78
Mo 08:30-12:00, Mo 14:00-19:00, Tu 08:30-12:00, Tu 14:00-19:00, We 08:30-12:00, We 14:00-19:00, Th 08:30-12:00, Th 14:00-19:00, Fr 08:30-12:00, Fr 14:00-19:00, Sa 09:00-12:00, Sa 14:00-18:00	71
Total	135913

Parmi les données non consolidées, il y a pas mal de souci en revanche. La majorité des stations sont propres mais environ 30% des PDC sont mal renseignés avec des valeurs qui pourraient être regroupées. Ce n'est pas une priorité, mais c'est bon à savoir. Analyse de juin 2025.

III · les besoins non-fonctionnels

Au vu du projet, nous devons inclure certains besoins particuliers

highest ↗ **volumétrie et performance spatiale**

Au vu de notre source de données (Base nationale des IRVE (Infrastructures de Recharge pour Véhicules Électriques)) nous allons avoir des besoins d'optimisation de la donnée géospatiale. De l'importation à l'affichage, une attention particulière sera donnée à la performance.

highest ↗ **dynamisme et réalisme de l'application**

Ce MVP devra être le plus réaliste possible. Il devra simuler une utilisation réelle et ne pas être bloquant. Mais vu que nous ne serons pas liés directement aux vraies bornes, une simulation de fin de charge devra être mise en place.

highest ↗ **sécurité**

Certaines fonctionnalités sont réservées aux utilisateurs enregistrés. Il faudra donc prévoir un environnement sécurisé pour l'utilisateur et les administrateurs.

highest ↗ **Accessibilité**

Pour une adaptabilité maximale, le site devra répondre aux critères d'accès numériques communément établis

IV · maquette de l'application, ui et ux

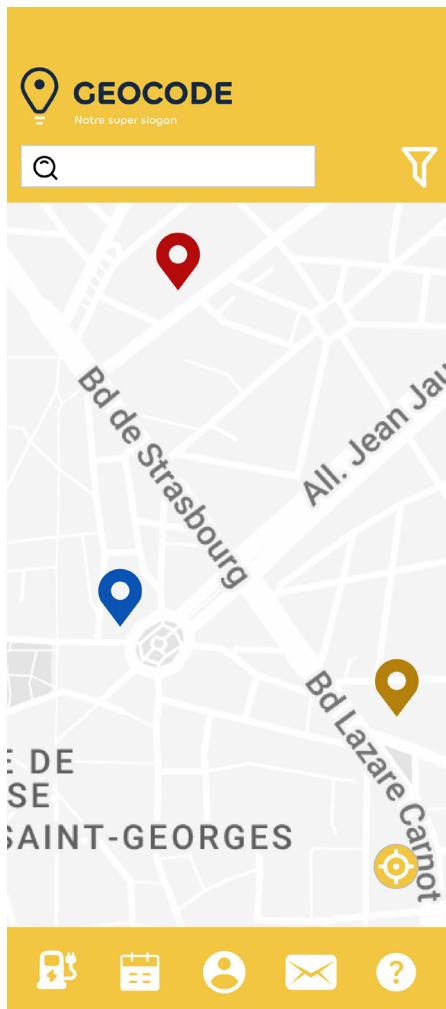
Compte tenu de l'objectif du backlog (site web responsive (exportation vers une application mobile probable) et de l'importance de la mobilité dans ce projet, nous nous sommes orientés vers une approche *Mobile First*. Cette Maquette est donc uniquement mobile. La version desktop a été réalisée avec une ergonomie différente. Maquette sur Figma.

I · charte graphique

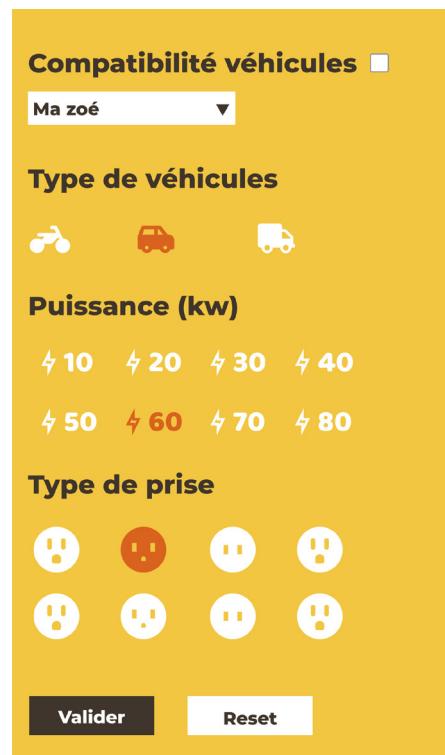


II · la maquette

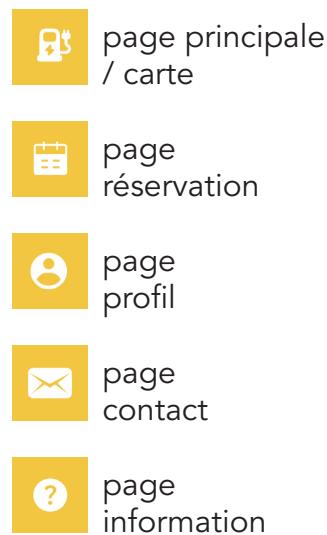
page principale



filtres



menu mobile



inscription

 **GEOCODE**
Notre super slogan

Mon profil



Télécharger une photo

Prénom
Tapez votre prénom

Nom
Tapez votre nom

Adresse mail
Tapez votre mail

Mot de passe
Tapez votre mot de passe

Confirmer le mot de passe
Confirmez votre mot de passe

Nom du véhicule
Tapez votre nom



Télécharger une photo

Marque
Tapez la marque du véhicule

Immatriculation
Tapez votre immatriculation

Type
Type de véhicule ▾

Type de prise
Type de prise ▾

Ajouter un autre véhicule

Valider mes informations



login

 **GEOCODE**
Notre super slogan

Adresse mail
Tapez votre mail

Mot de passe
Tapez votre mot de passe

Mot de passe oublié ?

Se connecter

Toujours pas de compte ?

S'inscrire !



station

 **GEOCODE**
Notre super slogan

La station

Station robert
Adresse : 12 Rue du grand
robert 31200 ROBERT

Les bornes de la station







Réserver **Annuler**

validation réservation

 **GEOCODE**
Notre super slogan

Merci de vérifier vos informations avant de finaliser la réservation.

Détails de la réservation

Station : Robert
Adresse : 12 Rue du grand robert 31200
ROBERT
Borne :
• Type de prise
• 40 kw
Créneau de réservation : 16h30 - 17h00
Paiement : CB

Valider ma réservation

Annuler ma réservation

confirmation réservation

 **GEOCODE**
Notre super slogan



Votre réservation à bien été prise en compte !

Vous allez recevoir par e-mail toutes les informations sur votre réservations.

Pour gerer vos reservations :

Voir mes réservations

réservations

Réserve en cours

Détails de la réservation

Station : Robert
Adresse : 12 Rue du grand robert 31200 ROBERT
Borne :

- Type de prise
- 40 kw

Créneau de réservation : 16h30 - 17h00
Paiement : CB

contact

Objet de votre demande

Selectionner ▾

Votre message

Taper votre message ...

Envoyer

message envoyé

Votre message a bien été envoyé !

Merci pour votre retour,
nous vous répondrons
dès que possible !

Votre message :

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec eros magna, fringilla vitae lacinia ultricies, tincidunt vitae erat. Praesent auctor lorem lacus, ac vestibulum quam tincidunt eu. Curabitur ligula arcu, laoreet in pretium aliquam, posuere vel ipsum. Vivamus ac ornare libero, in dapibus metus.

Donec eros magna, fringilla vitae lacinia ultricies, tincidunt vitae erat. Praesent auctor lorem lacus, ac vestibulum quam tincidunt eu.

Retourner à l'accueil

Historique de réservations

12 Juin 2025

06 Juin 2025

02 Juin 2025

15 Avril 2025

Un problème avec votre réservation ?

[Contactez-nous](#)

modal annulation réservation

Êtes vous sûr de vouloir annuler votre réservation ?

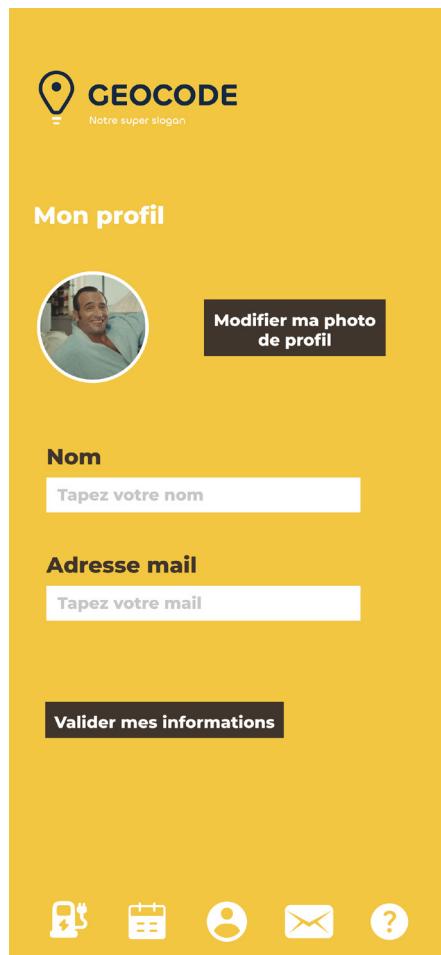
Oui

Non

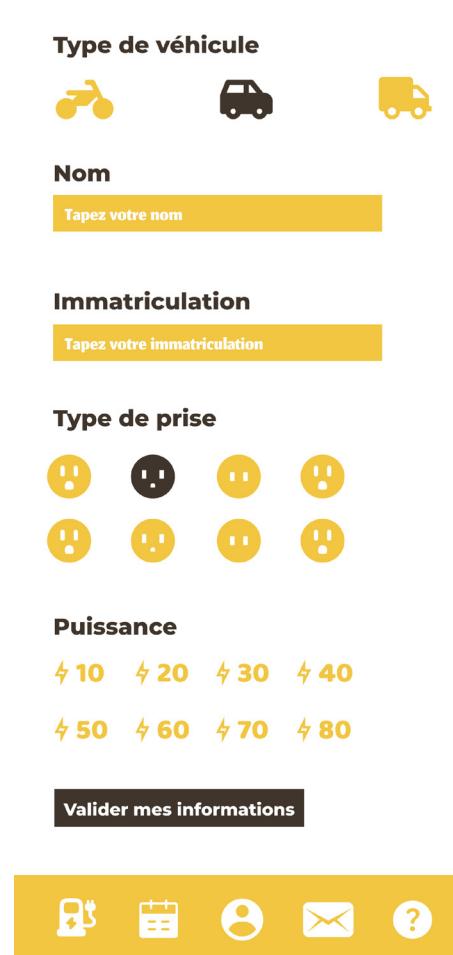
profil



modification profil



modification véhicule



III · evolution logo & nom

Le nom du projet fictif était Geocode. Nous avons cependant décidé de le modifier pour adopter un nom et un logo différents. Borne to be wild, La prise de la bastille, etc. Un long brainstorming pour valider notre nom "**Watts'up**". Un jeu de mots avec Watts à la place de What's. Avec notre logo, **conçu par Jonathan Dias**.

icon



logo complet



V · gestion de projet et organisation

I · l'équipe et notre rôle

L'équipe n'est constituée que de deux personnes. Jonathan Dias et Camille Céleste Covarel. Il a occupé le rôle de Scrum Master pendant que j'étais Product Owner. Nous avons souvent travaillé ensemble sur les fonctionnalités mais nous nous sommes spécialisés pour plus d'efficacité. Par exemple, Jonathan s'est principalement chargé du wireframe de l'application. J'ai défini la charte et l'ergonomie de l'application. Jonathan a été orienté front pendant que j'étais sur le back.

Jonathan Dias Scrum Master



Opérateur PAO issue d'un BTS design graphique à l'EPSESAAT.

Jonathan a travaillé sur la mise en place de la carte et des tuiles vectorielles, les filtres, l'authentification *login & inscription* ainsi qu'une grosse partie du design de l'application et son logo.

Camille Céleste Covarel Product Owner



Graphiste photographe issue d'un BTS photographie et d'un BAC communication visuelle plurimédia

J'ai travaillé sur l'adaptation du monorepo, le traitement complet de la données de l'analyse à l'import, la création de la BDD et du backend, la veille technique, la réservation des bornes, l'UX/UI desktop et mobile, une partie du design et le déploiement de l'application.

II · application des principes agile et scrum

Afin de garantir une livraison progressive, une transparence constante sur l'avancement du projet, et une capacité d'adaptation face aux incertitudes techniques notamment lors de l'import des données géospatiales, nous avons opté pour une approche **Agile** inspirée de **Scrum**

Cette méthode a structuré notre travail autour des principes suivants

Découpage en Sprints

Le projet a été planifié en sprints de 2 semaines. Chaque sprint avait des objectifs clairs et livrables *un incrément du produit*.

Backlog et Priorisation

L'ensemble des fonctionnalités et tâches techniques a été documenté dans un Backlog Produit maintenu par nous deux. La priorisation était effectuée avant chaque sprint pour s'assurer que les tâches à forte valeur métier ou les défis techniques bloquants (*comme le traitement PostGIS*) étaient traités en premier.

Synchronisation Quotidienne

Des points de synchronisation quotidiens (similaires aux Daily Scrums) permettaient

à l'équipe de se coordonner sur les avancements, d'identifier rapidement les obstacles techniques rencontrés et d'adapter immédiatement la suite de la journée si nécessaire.

Transparence

Chaque fin de sprint était clôturée par une revue informelle pour valider les développements et un retour sur l'efficacité de la méthode, permettant une amélioration continue de notre organisation.

III · outils de gestion et versionnement

Le suivi des tâches et la gestion du code ont été effectués via des outils professionnels, garantissant la qualité du code et l'efficacité de notre collaboration.

Nous avons utilisé **Jira** pour matérialiser notre approche **Agile**. L'outil a servi de support pour :

Le Backlog

Centralisation et classification des User Stories (réécriture des fonctionnalités).

Le Board Kanban/Scrum

Suivi visuel des tâches dans les différents états (à dev, en dev, dev, abandonnée).

Traçabilité

Chaque fonctionnalité ou bug était associé à un ticket unique, assurant la traçabilité complète de l'idée à sa mise en production.

Git a été l'outil central de notre collaboration pour maintenir l'intégrité du code. Le dépôt était hébergé sur **GitHub** et suivait une stratégie de branches rigoureuse :

Branches de Fonctionnalité

Chaque nouvelle fonctionnalité (ex: l'import de données, la réservation, le Trap Focus) était développée sur une branche dédiée.

Revue de Code et Fusion (Pull Request)

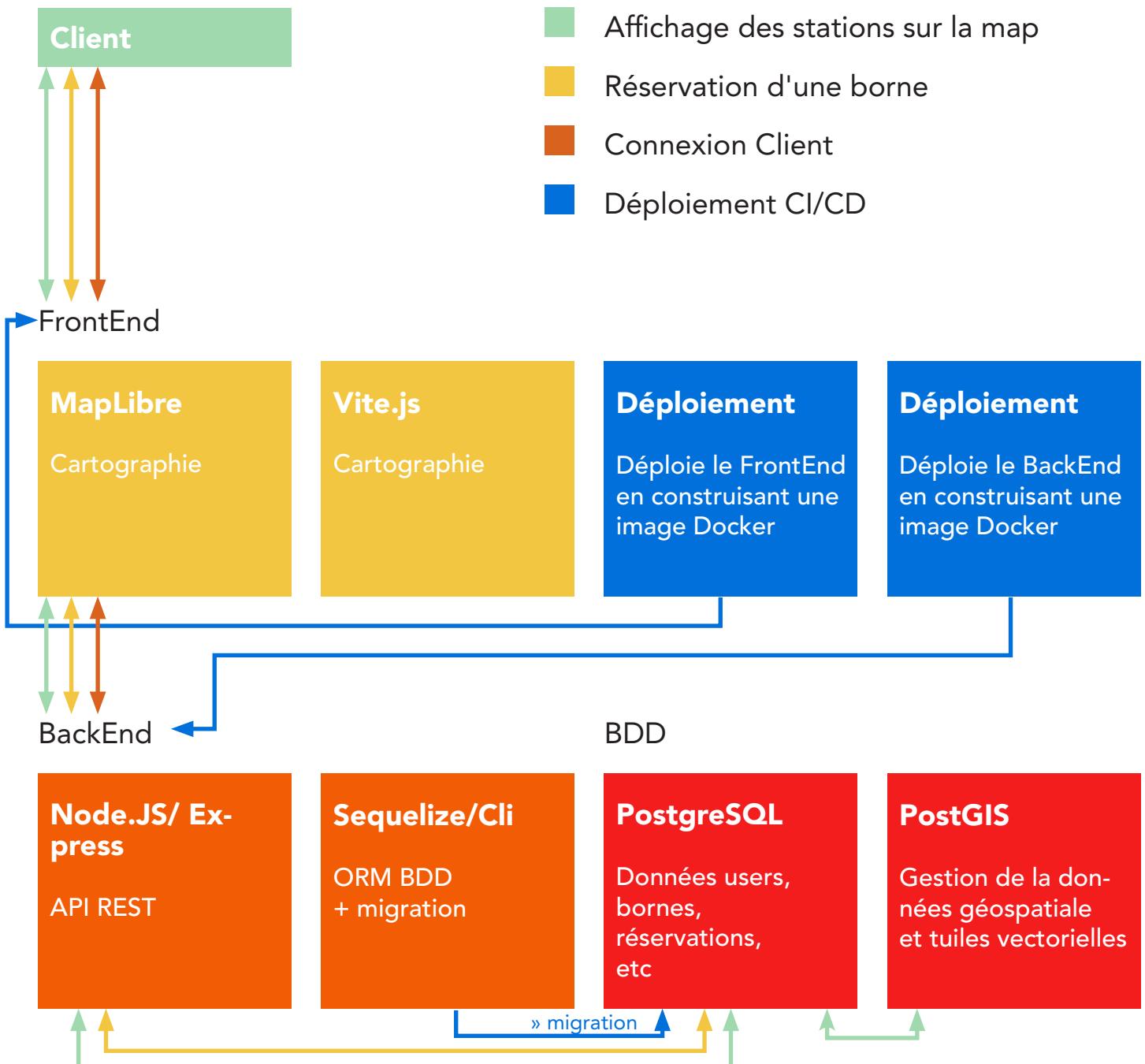
Le code n'était fusionné sur la branche principale (dev) qu'après une review par le second membre de l'équipe et la validation de l'absence de régression.

Déploiement (GitHub Action & GHCR)

Une fois le code fusionné sur dev, il était poussé sur la branche Staging de mon fork pour être déployé sur ma propre infrastructure. Je me suis occupée intégralement de la partie DevOps.

III · L'ENVIRONNEMENT TECHNIQUE & LE CŒUR DU BACKEND

I · architecture technique globale



II · analyse de la stack technique

I · frontend

Vite.js

Nous avons utilisé Vite.js comme build tool (outil de construction) moderne. Son architecture de Hot Module Replacement (HMR) a permis un environnement de développement extrêmement rapide et efficace.

MapLibre GL JS

Composant central de la cartographie, MapLibre a été choisi pour sa performance et sa capacité à afficher des tuiles vectorielles légères. Il est le récepteur direct des données optimisées par PostGIS.

Tanstack Query (React Query)

Pour gérer la communication avec l'API Backend, nous avons implémenté Tanstack Query. Cet outil gère la mise en cache, la synchronisation et l'invalidation des données (data fetching), réduisant ainsi la charge sur l'API et améliorant la réactivité de l'interface.

II · backend

Node.js / Express

Le cœur de notre API REST est construit sur Node.js avec le framework Express. Il gère le routage des requêtes HTTP, la logique métier (comme les réservations) et la communication sécurisée avec la base de données.

Sequelize / Cli

Nous avons choisi l'ORM Sequelize pour modéliser nos données et interagir avec PostgreSQL. Le Sequelize CLI a été un outil crucial pour gérer les migrations de la base de données, garantissant une structure de BDD versionnée et cohérente avec le code.

Bcrypt, JWT, express-rate-limit

Modules standards pour le hashage des mots de passe, la gestion des JWT et la protection contre le brute force.

III · base de données (BDD)

PostgreSQL

Choisi pour sa robustesse en tant que SGBD relationnel, il gère les données transactionnelles (utilisateurs, bornes, réservations).

PostGIS

L'extension PostGIS est la pierre angulaire de notre projet. Son rôle est de permettre le stockage et l'interrogation de données géospatiales à haute performance.

IV · CI/CD et déploiement

Docker

L'architecture CI/CD repose sur Docker pour conteneuriser les applications Frontend et Backend. La construction d'images Docker garantit que notre application s'exécute de manière identique en développement et en production.

GitHub Actions

Cet outil d'automatisation est utilisé pour notre processus d'Intégration et de Déploiement Continues (CI/CD). GitHub Actions déclenche automatiquement la construction des images Docker à chaque fusion de code et gère l'orchestration du déploiement vers le serveur.

GHCR (GitHub Container Registry)

Le GHCR est utilisé comme registre privé pour stocker les images Docker construites par GitHub Actions.

Secrets GitHub

L'ensemble des informations sensibles nécessaires au déploiement (identifiants du serveur SSH, clés de BDD, tokens d'API) est géré via les Secrets GitHub. Ce mécanisme garantit que ces informations ne sont jamais exposées dans le code source ou dans les logs des workflows CI/CD, respectant ainsi un standard de sécurité fondamental dans les pratiques DevOps.

III · modélisation de la BDD

I · choix et justification PostgreSQL & PostGIS

Lorsque nous avons commencé à préparer le projet durant les deux premières semaines. Après l'étude des données sur Power BI, je me suis interrogée sur la pertinence de rester sur MySQL (SGBD présent sur le monorepo fourni par notre école). Utilisant MariaDB depuis plusieurs années, je connais ses limites lorsqu'il s'agit de gérer un grand volume de données rapidement. En l'absence d'une extension géospatiale native et performante, MySQL/MariaDB aurait nécessité des calculs complexes et lourds pour la seule fonction de géocodage.

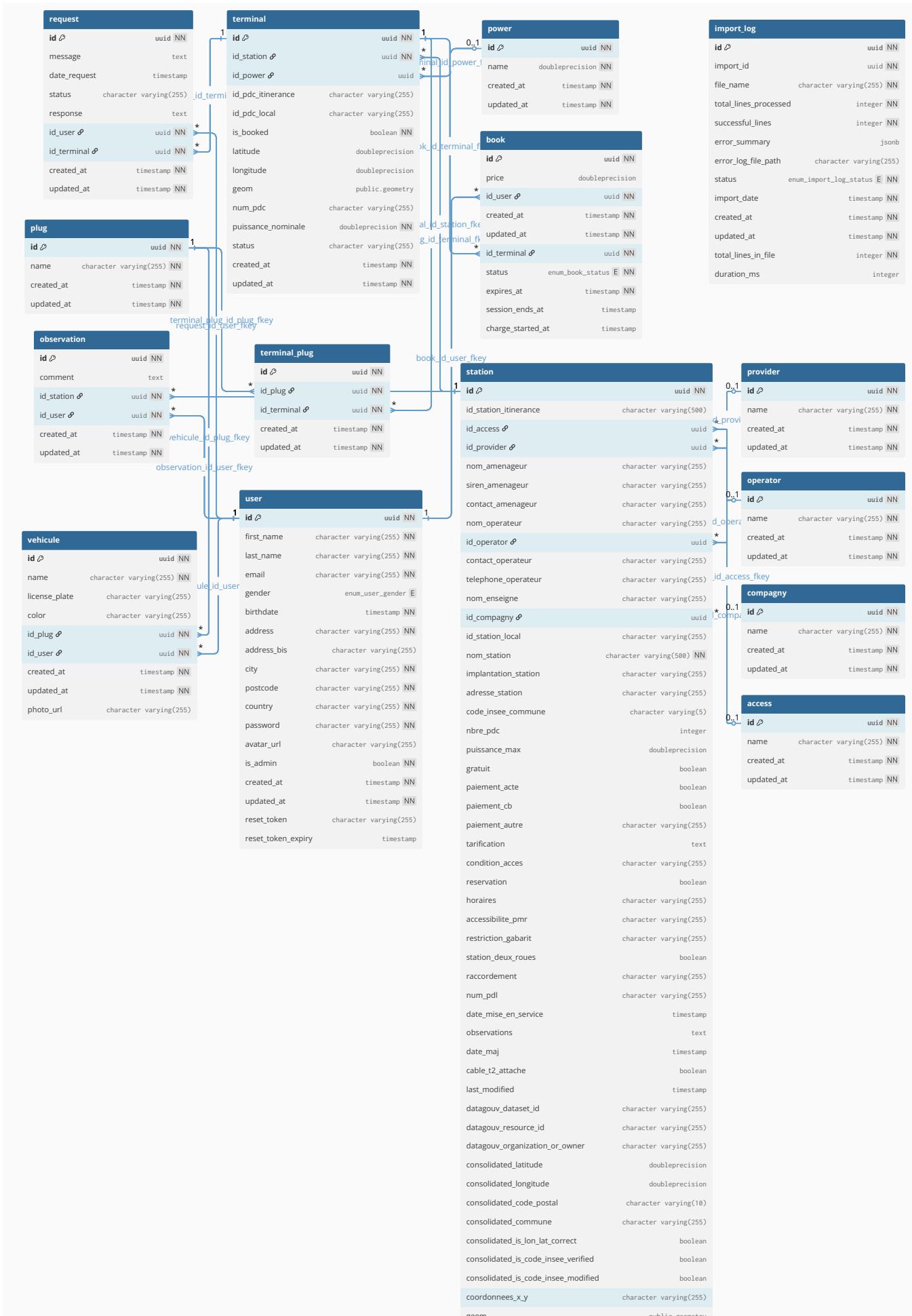
Au vu de la grande quantité de données (plus de 45 000 terminaux), une solution performante était indispensable. Un temps de chargement de plusieurs secondes était inacceptable. L'expérience montre qu'une mauvaise optimisation entraîne une perte d'utilisateurs. Nous nous sommes donc naturellement demandé comment réussir à gérer une telle quantité de data, en particulier pour les requêtes de proximité qui sont le cœur de l'application Frontend.

Nous avons eu la chance de bénéficier d'un échange précieux sur ce sujet.. Car d'anciens Wilders sont venus nous présenter leur entreprise (Flutilliant). Spécialisée dans la données, Tristan Laroye nous avait brièvement parlé d'un projet sur lequel ils travaillaient avec une carte et pas mal de données. C'était l'occasion parfaite de le contacter et discuter entre dev. La solution PostGIS s'est rapidement imposée au vu des alternatives.

Disposer d'un plugin ultra optimisé qui transforme PostgreSQL en un véritable SGBD Spatial (SGBDS) pour les données géospatiales, avec un traitement natif des tuiles vectorielles via des fonctions dédiées (comme ST_AsMVT) pour l'affichage MapLibre. Une indexation GIST (Generalized Search Tree) très performante pour accélérer les requêtes de recherche dans une zone géographique, et une données géométrique unique à charger. Ces atouts ont largement justifié le choix de cette solution et ne pas perdre trop de temps sur cette veille.

PostgreSQL s'est naturellement greffé derrière ça. Il suffisait alors d'adapter le Backend à utiliser cette solution, en implémentant le type de colonne GEOMETRY(Point, 4326) pour les entités station et terminal, et à faire les premiers tests.

II · MCD / MLD



IV · Réalisation Backend 1 · le traitement de données

I · préambule.

Cette fonction de traitement des données a été un véritable défi pour moi. n'ayant jamais travaillé sur une telle quantité de données, j'ai dû adapter ma stratégie à plusieurs reprises avant d'arriver à l'objectif. Mon but était d'intégrer au mieux les données avec le plus de données possible sans surcharger le serveur. Je voulais une fonction rapide et légère tout en sachant d'avance que je ne pourrais pas tout intégrer correctement dans le temps imparti.

Ma première idée a consisté à consolider et intégrer toutes les 5000 lignes lues dans une fonction simple non bloquante pour l'utilisateur. Mais le résultat était bien trop imparfait pour pour être utilisé (- de 25% de la données importée). Je vais donc vous présenter la formule qui m'a permis de réussir cet import.

II · performance, streaming et consolidation

I · lecture en streaming

Pour éviter de saturer la mémoire du serveur (le buffer overflow), le fichier CSV est lu directement en mode stream. L'utilisation du `fs.createReadStream` natif de Node.js, combinée à `fast-csv`, a permis de traiter le fichier ligne par ligne sans jamais le charger en entier.



```
const csvStream = fs
  .createReadStream(filePath)
  .pipe(fastCsv.parse({ headers: true }));

// Utilisation de la syntaxe asynchrone pour
// traiter chaque ligne
for await (const row of csvStream as
AsyncIterable<CsvRow>) {
  // ...
}
```

II · consolidation en mémoire optimisation des écritures

Plutôt que d'interroger la BDD pour chaque ligne, une phase de consolidation est effectuée en mémoire.

Les données sont regroupées dans une Map (`stagedStationData`) avant d'être écrites, transformant ainsi le fichier plat CSV en une structure hiérarchique (Station -> Terminaux) qui minimise le nombre d'opérations d'écriture.



```
const stagedStationData = new Map<
  string,
  StagedStationContent
>();

// ... (dans la boucle de streaming) ...

const stationId =
getStationCompositeId(transformedData.stationData);
if (!stagedStationData.has(stationId)) {
  // Si la station n'existe pas, on la crée
  stagedStationData.set(stationId, {
    stationData: transformedData.stationData,
    terminals: [transformedData.terminalData],
  });
} else {
  // Si la station existe déjà, on ajoute le nouveau terminal
  stagedStationData.get(stationId)!.terminals.push(transformedData.terminalData);
}
```

III · fiabilité & intégrité, nettoyage, cache et transaction

I · nettoyage et normalisation

```
● ● ●  
// Extrait de stringNormalizer.ts  
export function  
parseGeoJSONPoint(coordsString: string):  
Point | null {  
    // ... (Logique de validation du format  
GeoJSON) ...  
    if (  
        parts.length === 2 && !  
Number.isNaN(parts[0]) && !  
Number.isNaN(parts[1]))  
    ) {  
        const longitude = parts[0];  
        const latitude = parts[1];  
        // Validation des bornes Lat/Lon  
        if (latitude ≥ -90 && latitude ≤ 90 &&  
longitude ≥ -180 && longitude ≤ 180) {  
            return { type: "Point", coordinates:  
[longitude, latitude] };  
        }  
        // Log l'erreur et retourne null si  
invalide  
        console.warn(`[Parse Error] Coordonnées  
GeoJSON invalides: "${coordsString}"`);  
        return null;  
    }  
}
```

```
● ● ●  
// (findOrCreate - Extrait de dataTransformer.ts et  
importCache.ts)  
// Appel du Cache (dans dataTransformer.ts)  
const idAccess = accessName  
? await findOrCreateAccessByName(accessName)  
: null;  
// ...  
const idPower =  
puissanceNomiale !== null  
? await  
findOrCreatePowerByName(puissanceNomiale)  
: null;  
  
// Fonction Cache (dans importCache.ts)  
export async function findOrCreatePowerByName(name:  
number): Promise<string> {  
    // findOrCreate agit comme un cache  
    const [power] = await Power.findOrCreate({  
        where: { name },  
        defaults: { name },  
    });  
    return power.id;  
}
```

Les données brutes issues du CSV sont rarement propres. Avant l'écriture, chaque ligne passe par une phase de nettoyage et de transformation (**dataTransformer.ts**), qui est le point de contrôle de l'intégrité :

Nettoyage et Normalisation

La donnée brute du CSV est d'abord "nettoyée" par un service dédié (**stringNormalizer.ts**). Ce service valide et standardise les champs (ex: **parseNumber**, **parseBoolean**). Il gère les incohérences de format et l'échappement des caractères pour prévenir les injections.

Intégration PostGIS

La fonction **parseGeoJSONPoint** garantit que les coordonnées brutes du CSV sont validées et converties au format GeoJSON (**type: 'Point'**, **coordinates: [lon, lat]**), prêtes à être insérées dans la colonne PostGIS **geom**. Cette étape est cruciale pour le bon fonctionnement de l'indexation GIST.

Le Cache d'Importation

Gestion des Relations N:1

C'est la stratégie clé pour la performance. Pour gérer les relations (tables de typologie comme **plug**, **power**, **access...**) sans saturer la base de données, des fonctions **findOrCreate...ByName** ont été mises en place (**importCache.ts**). Ces fonctions agissent comme un cache côté BDD : elles évitent de créer des doublons et de solliciter inutilement la BDD en utilisant la logique **findOrCreate** de Sequelize. Chaque nom de plug est ainsi trouvé ou créé une seule fois, garantissant l'intégrité et un temps de traitement optimal.

Sécurité Transactionnelle (Rollback)

Enfin, pour garantir qu'aucune données orphelines n'est insérée, l'écriture de chaque station (qui inclut tous ses terminaux et ses plugs) est encapsulée dans une transaction SGBD (`sequelize.transaction()`). Le point fort de cette approche est le mécanisme de rollback : si l'insertion d'un seul terminal ou d'un plug échoue, la transaction entière est annulée. Cela garantit la cohérence des données pour l'entité complète et évite l'écriture de données partielles ou invalides.



```
// Snippet 5 (Transaction et Rollback - Extrait de importController.ts)
const stationTransaction = await sequelize.transaction();

try {
    // 1. Logique d'Upsert (findOrCreate) de la station
    const [station] = await Models.Station.findOrCreate({
        where: { id_station_itinerance: stationId },
        transaction: stationTransaction,
    });

    // 2. Traitement et Upsert des terminaux et relations (plugs)
    // ...

    await stationTransaction.commit(); // Si tout est OK
} catch (error) {
    // Si une erreur est levée sur l'Upsert/relations
    await stationTransaction.rollback(); // Annulation de l'opération entière
    logImportErrorToFile(/* ... */);
}
```

II · sécurité transactionnelle et logique upsert

Le danger d'un import par lots est qu'une seule erreur sur une ligne invalide peut corrompre le lot entier. La solution a été d'adopter un modèle d'Upsert (Update or Insert) et d'encapsuler chaque écriture dans une transaction SGBD dédiée.

Transaction par Station

Une transaction est ouverte pour chaque station (qui inclut tous ses terminaux et ses plugs). Si l'insertion d'un terminal échoue pour cette station, l'ensemble de l'opération est annulé (rollback), garantissant la cohérence des données pour cette entité.

Logique Upsert

L'identifiant unique d'itinérance (`id_station_itinerance`) a été utilisé comme clé métier pour l'Upsert (`Models.Station.findOrCreate`). Cette approche permet de mettre à jour les stations existantes (si le fichier est ré-importé) tout en créant les nouvelles.



```
// Extrait de importController.ts - Processus de transaction
const stationTransaction = await sequelize.transaction();

try {
    // 1. Logique d'Upsert de la station et de ses données (findOrCreate)
    const [station, createdStation] = await Models.Station.findOrCreate({
        where: { id_station_itinerance: stationId },
        // ... (données de la station)
        transaction: stationTransaction,
    });

    // 2. Traitement et Upsert des N terminaux de cette station

    // 3. Traitement des relations N:M (Plugs)
    await stationTransaction.commit(); // Si tout est OK
} catch (error) {
    // Si une erreur est levée sur l'Upsert/relations
    await stationTransaction.rollback(); // Annulation de l'opération entière
    logImportErrorToFile(/* ... */);
}
```

IV · traçabilité & suivi, logs, webSockets et contrôle

I · logique de journalisation dédiée (logger.ts)

J'ai créé un service de journalisation dédié (logger.ts) intégré au contrôleur, permettant d'aller au-delà des simples console.log

Niveaux de Criticité

Le service définit des niveaux (DEBUG, INFO, SUCCESS, ERROR, CRITICAL) qui permettent de filtrer l'information.

Audit d'Erreur

En cas d'échec de la transaction d'une station, le contrôleur utilise la fonction logImportErrorToFile qui enregistre l'erreur détaillée dans un fichier de log dédié (import_errors_[uuid].log). Cela permet à l'administrateur de constater les erreurs et de corriger les lignes problématiques sans interrompre le processus pour les lignes valides. Ce mécanisme a été d'une très grande

aide lors de l'amélioration du processus. Elle m'a permis de passer de 25% de réussite à 99,6%. Certaines erreurs restent à ce jour non expliquées. Ceci s'explique par un manque d'investigation approfondie.

Log BDD

L'ensemble du processus est tracé dans la table import_log, enregistrant le statut final (COMPLETED, FAILED, PARTIAL_SUCCESS) ainsi que la durée, ce qui est crucial pour le suivi et l'optimisation future.



```
// Extrait de importController.ts - Mise à jour du log final
await Models.ImportLog.update(
{
  status: LogLevelToImportStatus(finalStatus), // Ex: 'COMPLETED' ou 'PARTIAL_SUCCESS'
  successful_lines: successfullines,
  total_lines_processed: totalProcessedCsvLines,
  duration_ms: Date.now() - startTime.getTime(),
  error_summary: Object.values(errorsSummary).length > 0 ? errorsSummary : null,
},
{ where: { import_id: importUuid } },
);
```

II · suivi en temps réel et contrôle administratif

Suivi temps réel et réhydratation

Résilience UX

L'utilisation des WebSockets (**websocket.ts**) permet au Backend de pousser les mises à jour de statut au Frontend sans recharge. Une attention particulière a été portée à la continuité de service : une logique de "**réhydratation**" a été implémentée. Si l'administrateur quitte la page ou la rafraîchit, le serveur détecte la nouvelle connexion, recherche si un import est **IN_PROGRESS** en base de donnée, et renvoie immédiatement l'état d'avancement pour resynchroniser l'interface.



```
//(Réhydratation WebSocket - Extrait de websocket.ts)
wss.on("connection", async (ws: WebSocket) => {
    // Vérification à la connexion : y a-t-il un import en cours ?
    const ongoingImport = await Models.ImportLog.findOne({
        where: { status: "IN_PROGRESS" },
    });

    // Si oui, on "réhydrate" le client immédiatement
    if (ongoingImport) {
        const percentage = /* calcul du pourcentage */;
        // On renvoie l'état actuel au nouveau client connecté
        notifyProgress(
            ongoingImport.import_id,
            percentage,
            `Progression actuelle: ${percentage}%`,
            ongoingImport.successful_lines,
            ws // Envoi ciblé à ce client uniquement
        );
    }
});
```

Arrêt propre du processus

Une fonction de contrôle, **isStopRequested(importUuid)**, est vérifiée à chaque itération du traitement. Si l'administrateur demande l'annulation, cette vérification permet au worker de s'arrêter proprement, fermant les flux et marquant l'opération comme **CANCELLED** plutôt que de laisser des transactions orphelines.



```
// Extrait de importController.ts - Contrôle du flux
if (isStopRequested(importUuid)) {
    // Si l'admin a demandé l'arrêt
    console.warn(`Arrêt de l'import ${importUuid} demandé par l'utilisateur.`,
        LogLevel.WARN);
    finalStatus = LogLevel.ERROR;
    // ou un statut spécifique pour l'annulation
    break;
}

// Mise à jour de la progression vers le Front-End
if (totalProcessedCsvLines % PROGRESS_UPDATE_FREQUENCY === 0) {
    notifyProgress(
        importUuid,
        totalProcessedCsvLines,
        totalLinesFromMetadata,
    );
}
```

III · Mise en œuvre des migrations (rigueur du schéma)



```
// (package.json - Scripts de Migration)
"scripts": {
    "migrate:dev": "cross-env NODE_ENV=development sequelize-cli db:migrate",
    "migrate:prod": "cross-env NODE_ENV=production sequelize-cli db:migrate",
    "db:seed:prod": "cross-env NODE_ENV=production sequelize-cli db:seed:all",
    // ...
},
```



```
// (Migration - Définition PostGIS)
// Extrait de 20250702095341-create-initial-tables.js

// ...
await queryInterface.createTable("station", {
    // ...
    id_station_itinerance: {
        type: Sequelize.STRING(255),
        allowNull: true,
        unique: true,
    },
    // Déclaration du champ géospatial
    geom: {
        type: Sequelize.GEOMETRY("POINT", 4326),
        allowNull: true,
    },
    created_at: {
        allowNull: false,
        type: Sequelize.DATE,
    },
    // ...
});

await queryInterface.createTable("terminal", {
    // ...
    // Le terminal reçoit également un champ géospatial
    geom: {
        type: Sequelize.GEOMETRY("POINT", 4326),
        allowNull: true,
    },
    // ...
});
```

La gestion du schéma de la base de données a été traitée avec la même rigueur que le code applicatif. Pour garantir la cohérence et la reproductibilité de l'environnement, le schéma n'a pas été créé manuellement. Il est entièrement **versionné et contrôlé par l'outil Sequelize-CLI**.

Versioning et Séparation des Environnements

L'utilisation de **sequelize-cli** permet de suivre chaque modification de structure via un fichier de migration daté (**20250702095341-create-initial-tables.js**). Les scripts **npm** dans le **package.json** exploitent la librairie **cross-env** pour garantir que les migrations sont appliquées avec les variables d'environnement spécifiques à la **production (migrate:prod)** ou au développement (**migrate:dev**).

implémentation du Choix PostGIS

Le fichier de migration initial est la preuve que le choix technique de PostGIS a été concrétisé au niveau de la structure. Il est vital de déclarer explicitement le type de colonne géospatiale lors de la création des tables **station** et **terminal** pour que PostGIS puisse l'indexer efficacement. Le type **Sequelize.GEOMETRY("POINT", 4326)** définit un point géographique utilisant le système de référence WGS 84 (SRID 4326).

V · Réalisation Backend 2 · logique métier et automatisation

Cette section présente le développement du cœur fonctionnel de l'application : le système de réservation. Contrairement aux opérations CRUD standard, ce module implémente des règles métier strictes, une simulation de charge *IoT* et des tâches planifiées *Cron* pour garantir l'autonomie du système.

I · le workflow de réservation transactions et règles métier

La réservation d'une borne ne se limite pas à une insertion en base de données. C'est une opération critique qui doit garantir qu'aucun conflit n'existe. J'ai développé le `reservation.controller.ts` pour gérer cette complexité.

1 · Atomicité et Règles Métier

Chaque demande de réservation est encapsulée dans une transaction SGBD. Avant de valider, le contrôleur effectue des vérifications bloquantes :

Règle 1 Unicité

Un utilisateur ne peut avoir qu'une seule réservation active (**status: ACTIVE ou IN_USE**). Si une réservation existe déjà, la transaction est annulée (**rollback**).

Règle 2 Disponibilité Complexe

La recherche d'une borne disponible ne se fait pas seulement sur le statut, mais aussi sur la compatibilité de puissance et de connecteur (via des sous-requêtes SQL `sequelize.literal` sur la table de jointure **terminal_plug**).

2 · Simulation de l'*IoT* Logique de Charge

N'ayant pas de véritables bornes connectées, j'ai dû simuler le comportement physique de la charge dans le Backend **startCharge**. J'ai implémenté un algorithme qui calcule le temps de charge nécessaire en fonction :

De la puissance de la borne *puissance_nominale*

Du moment de la journée

(Charge plus lente/rapide simulée selon si c'est le jour ou la nuit, via des constantes **NIGHT_START_HOUR**). Cela permet de générer une date de fin de session **session_ends_at** réaliste, rendant l'application vivante pour l'utilisateur.



```
// Logique de transaction et
vérification (Extrait de
reservation.controller.ts)
const transaction = await
sequelize.transaction();
try {
    // Vérification règle métier : Déjà
    une réservation ?
    const existingReservation = await
Book.findOne({
    where: { id_user: userId, status:
{ [Op.in]: [ReservationStatus.ACTIVE,
ReservationStatus.IN_USE] } },
    transaction,
});
if (existingReservation) {
    await transaction.rollback(); //
Annulation immédiate
    return res.status(409).json({
message: "Réservation déjà en cours."
});
}
// ... Recherche terminal et
création ...
await transaction.commit();
} catch (error) {
    await transaction.rollback();
}
```

II · automatisation et fiabilité tâches cron

Pour garantir que les bornes ne restent pas bloquées indéfiniment si un utilisateur ne se présente pas ou si une charge est terminée, j'ai mis en place un système d'automatisation via `node-cron`. C'est le "garant" de la disponibilité du parc.



```
Tâche Cron d'expiration (Extrait de
cron.service.ts)
const expireReservations = async (): Promise<void> => {
    // Recherche des réservations
    actives dont la date est passée
    const expiredBookings = await
    Book.findAll({
        where: {
            status:
                ReservationStatus.ACTIVE,
            expires_at: { [Op.lt]: new
                Date() }, // Op.lt = Less Than (Plus
                petit que maintenant)
        },
        transaction,
    });
    // Mise à jour en masse et
    libération des bornes
    // ...
};

// Planification toutes les minutes
cron.schedule("* * * * *", () => void
expireReservations());
```



```
// extrait de router.ts

const apiLimiter = rateLimit({
    windowMs: 15 * 60 * 1000,
    max: 10,
    standardHeaders: true,
    legacyHeaders: false,
    message: 'Trop de requêtes envoyées
    depuis cette IP, veuillez réessayer
    après 15 minutes.',
});
```

Le service `cron.service.ts` exécute deux tâches critiques chaque minute

Nettoyage des "No-Show"

`expireReservations`

Si une réservation dépasse sa date d'expiration `expires_at` sans que la charge n'ait débuté, le système passe automatiquement le statut à **EXPIRED** et libère la borne `is_booked: false`. Cela empêche le blocage abusif des ressources.

Finalisation des Charges

`completeChargeSessions`

Lorsque l'heure de fin de charge simulée `session_ends_at` est atteinte, le script clôture automatiquement la session `status: COMPLETED` et rend la borne disponible.

III · protection de l'API

rate limiting

Bien que des tests de charge complets n'aient pas été réalisés, la sécurité de l'API a été renforcée par l'implémentation d'un Rate Limiting ciblé.

J'ai utilisé **express-rate-limit** pour protéger les points d'entrée sensibles contre le spam et les attaques par force brute, notamment le formulaire de contact.

Configuration

Une limite de 10 requêtes par fenêtre de 15 minutes par IP a été définie.

Application

Ce middleware est appliqué spécifiquement sur la route **POST /contact**, protégeant ainsi le service d'envoi de mails `nodemailer` contre la saturation.

IV · refactorisation du service de notification Mailtrap & Templating

Initialement, la fonctionnalité de contact avait été implémentée par mon binôme *Jonathan* pour envoyer des emails réels. Bien que fonctionnelle, cette approche ne me semblait pas adaptée au contexte d'un MVP qui demande des ajustements. Elle rendait les tests fastidieux et forçait l'utilisation d'un vrai service mail avec toutes ses contraintes.

J'ai donc décidé de réécrire et d'adapter ce module pour le rendre moins rigide.

Intégration de Mailtrap

Environnement de Test

J'ai modifié la configuration du transporteur **nodemailer** pour intégrer Mailtrap.

Cette solution de "sandbox" intercepte les emails sortants dans une boîte de réception virtuelle. Cela m'a permis de tester l'ensemble du flux de notification (mot de passe oublié, contact) en toute sécurité, sans jamais spammer de véritables adresses.

Templating HTML et Amélioration de l'UX

La version originale envoyait le message au format texte brut, ce qui manquait de professionnalisme pour l'administrateur recevant la demande. J'ai donc profité de cette refonte pour développer un template **HTML/CSS responsive** directement dans le contrôleur **userActions.ts**. Le mail injecte désormais dynamiquement le contexte de l'utilisateur (son nom, mais aussi la liste de ses véhicules et le type de prise associé), offrant une lecture claire et immédiate au support.

Gestion via Secrets GitHub

Les valeurs concrètes pour ces variables (**EMAIL_HOST**, **EMAIL_USER**, etc.) ont été stockées dans les Secrets GitHub et injectées au moment du déploiement, respectant ainsi les bonnes pratiques de sécurité (aucune clé API dans le code source).



```
//(Configuration Mailer et Templating  
- Extrait de userActions.ts)
```

```
// 1. Template HTML (Mon ajout  
personnel)
```

```
const emailHtml = `  
    <!DOCTYPE html>  
    <html>  
        <div class="summary-section">  
            <h2>Informations de  
l'utilisateur</h2>  
            <ul>${userVehicles}</ul>  
        </div>  
        <p><strong>Message :</strong>  
        ${message.replace(/\n/g, "<br>")}</p>  
    </html>  
`;
```

```
// 2. Utilisation des variables  
d'environnement (Configurées par mes  
soins en Prod)
```

```
const transporter =  
nodemailer.createTransport({  
    host: process.env.EMAIL_HOST, //  
Défini via GitHub Secrets → Mailtrap  
    port:  
    Number(process.env.EMAIL_PORT),  
    auth: {  
        user: process.env.EMAIL_USER,  
        pass: process.env.EMAIL_PASS,  
    },  
});
```

V · Architecture et audit de sécurité maîtrise des risques

Mon rôle en matière de sécurité a consisté à définir le cadre architectural préventif et à effectuer la revue technique des modules sensibles développés en équipe. Cette démarche visait à transformer les choix de développement collaboratifs en une application dont les failles structurelles sont contrôlées.

Sécurité structurelle

Mes choix d'architecture

Cette couche de sécurité, que j'ai mise en place, agit indépendamment du module d'authentification pour prévenir les failles les plus courantes :

Prévention contre l'injection SQL

J'ai imposé l'utilisation de l'ORM **Sequelize** sur l'ensemble du Backend.

» Impact

Ceci garantit que toutes les requêtes BDD sont paramétrées et que les entrées utilisateurs sont assainies avant l'exécution, nous protégeant efficacement contre les injections SQL.

Protection contre l'Énumération

J'ai généralisé l'usage des **UUID v4** pour toutes les clés primaires.

» Impact

Les IDs non-séquentiels empêchent l'énumération des ressources (ex: utilisateurs) et réduisent le risque de **référence directe non sécurisée (IDOR)**.

Audit technique et identification de la dette

J'ai réalisé une revue des mécanismes d'authentification implémentés par mon binôme, ce qui a permis de valider les bonnes pratiques existantes et d'identifier la dette technique à adresser en V2.

Hashage des mots de passe

Utilisation de **Bcrypt**

Critique

Bien que ce soit un standard, j'aurais imposé **Argon2** sur un projet neuf (plus moderne et robuste). C'est une amélioration prévue en V2.

Stockage du Token

JWT dans un **Cookie HttpOnly**

Validation critique

Ce choix est essentiel. Le drapeau **HttpOnly** rend le token inaccessible au JavaScript client. C'est l'unique parade pour contrer le risque majeur de vol de session par XSS (Cross-Site Scripting).

Gestion de session

JWT à durée limitée (24h)

Flaw identification

L'absence de Refresh Token compromet la sécurité de l'utilisateur et complexifie la gestion du cycle de vie des sessions. Cela constitue une dette technique que je propose de combler en V2.

Conformité OWASP générale

L'application respecte les principes de sécurité critiques grâce aux mécanismes suivants :

Gestion des Logs et Monitoring

OWASP A10

L'application intègre un service de log et un système de **WebSocket pour le monitoring d'import**, facilitant la détection et le suivi des anomalies.

Protection des ressources

Le Rate Limiting est appliqué sur les endpoints publics (**/auth/login**, **/contact**) pour prévenir les attaques par force brute (défense contre OWASP A07 : **Identification Failure**).

Validation des entrées

Le service de normalisation neutralise les entrées HTML et les caractères spéciaux, agissant comme une première ligne de défense contre les injections XSS sur les données non structurées.

IV. FRONTEND ET EXPÉRIENCE UTILISATEUR

I · Réalisation Frontend 1 · Architecture adaptative et accessibilité

Le projet a débuté avec une approche Mobile First classique, indispensable pour une application destinée aux conducteurs en déplacement. Cependant, j'ai rapidement identifié qu'une simple adaptation CSS (Responsive) ne suffirait pas à offrir une expérience optimale sur tous les supports.

I · le défi UX du responsive à l'adaptatif

J'ai défendu une stratégie d'ergonomie différenciée pour répondre aux contraintes spécifiques de chaque support

Sur Mobile Usage séquentiel

L'espace est limité. L'utilisateur doit se concentrer sur une tâche à la fois (chercher, puis choisir, puis réserver). La navigation doit donc être pleine page.

Sur Desktop Usage contextuel

L'écran permet le multitâche. J'ai considéré qu'il était critique de ne pas perdre le contexte géographique. Recharger la page pour afficher le détail d'une station ferait perdre la carte de vue, ce qui est une friction UX majeure sur un grand écran.

Ma Solution

Le Système d'Overlay J'ai proposé et implémenté un système **d'Overlay** (panneau superposé). Sur Desktop, la carte reste toujours visible en arrière-plan, et les interactions (Détails, Réservation) s'ouvrent par-dessus. Sur mobile, le routeur bascule sur une navigation classique.

II · Implémentation technique *Route Handles*

Pour ne pas dupliquer le code des pages, j'ai utilisé une fonctionnalité avancée de React Router : les **handles**. J'ai marqué certaines routes comme étant "affichables en Overlay".

Dans le composant racine **App.tsx**, j'ai développé une logique conditionnelle qui agit comme un aiguillage.



```
// Logique d'affichage Adaptative (Extrait de App.tsx)
function AppContent() {
  const matches = useMatches();
  // Détection : Cette route est-elle candidate pour l'Overlay ?
  const isOverlayRoute = matches.some((match) => match.handle?.isOverlay);

  // Décision UX basée sur le device (isMobile) ET la route
  // Sur Desktop : On ouvre l'Overlay par-dessus la carte
  const shouldShowOverlay = !isMobile && isOverlayRoute;
  // Sur Mobile : On navigue vers la page dédiée (Outlet standard)
  const shouldShowMobilePage = isMobile && !isRootPath;

  return (
    <div className="app-container">
      {shouldShowOverlay && <Overlay title="">{/* Contenu de la route */}</Overlay>}
      {shouldShowMobilePage && <div className="main-page-container">{/* Contenu plein écran */}</div>}
      {/* La carte est toujours rendue en fond sur Desktop */}
      <LandingPage />
    </div>
  );
}
```

III · Le défi de l'accessibilité Refactorisation V1.1

L'introduction de modales non-natives (Overlays) a créé un problème d'accessibilité majeur : la navigation au clavier (**Tab**) pouvait "s'échapper" de l'overlay, laissant l'utilisateur naviguer à l'aveugle sur la carte en arrière-plan.

J'ai dû effectuer une refactorisation complète du composant **Overlay.tsx** pour y intégrer un **Focus Trap** (Piège à focus) manuel. Ce n'était pas présent dans la V1 collaborative.

J'ai codé un algorithme qui intercepte la touche **Tab** et force le curseur à boucler uniquement sur les éléments interactifs de l'Overlay tant qu'il est ouvert, garantissant une expérience navigable pour tous.



```
// Focus Trap Manuel (Extrait de Overlay.tsx)
useEffect(() => {
  const handleKeyDown = (event: KeyboardEvent) => {
    // Si Touche TAB et Overlay ouvert
    if (event.key === "Tab" && overlayRef.current) {
      const focusable = overlayRef.current.querySelectorAll('button, input, [href] ... ');
      const first = focusable[0];
      const last = focusable[focusable.length - 1];

      // Si Shift+Tab sur le premier élément → Renvoi au dernier
      if (event.shiftKey && document.activeElement === first) {
        last.focus();
        event.preventDefault();
      }
      // Si Tab sur le dernier élément → Renvoi au premier
      else if (!event.shiftKey && document.activeElement === last) {
        first.focus();
        event.preventDefault();
      }
    }
  };
  document.addEventListener("keydown", handleKeyDown);
}, []);
```

II · Réalisation Frontend 2 · logique métier et cycle de vie

Si la carte permet de trouver une station, c'est la **page de réservation** qui constitue le cœur fonctionnel de l'application. J'ai été chargée de développer l'intégralité de ce module, de la sélection de la borne jusqu'au suivi en temps réel de la charge.

I · agrégation de données *simplifier le choix (UX)*



```
// Algorithme d'agrégation (Extrait de
stationDetails.tsx)
const terminalGroups = useMemo(() => {
  if (!station?.terminals) return [];
  const groups = new Map<string,
TerminalGroup>();

  for (const terminal of station.terminals)
  {
    // Création d'une clé unique basée sur
    // la puissance et les prises
    const groupKey =
`${terminal.puissance_nominale}-${plugNames}
`;

    // Logique de comptage et de
    // regroupement
    if (groups.has(groupKey)) {
      // Incrémentation des compteurs
      existingGroup.count++;
      if (!terminal.is_booked)
existingGroup.availableCount++;
    } else {
      // Initialisation du groupe
      groups.set(groupKey, { /* ... */ });
    }
  }
  return Array.from(groups.values());
}, [station]);
```

L'API renvoie une liste brute de tous les terminaux d'une station (par exemple, 4 terminaux identiques de 22kW).

Afficher cette liste telle quelle aurait été répétitif et confus pour l'utilisateur.

Mon algorithme de regroupement

Dans le composant **StationDetails.tsx**, j'ai implémenté une logique de traitement de données côté client. J'utilise le hook **useMemo** pour agglomérer les terminaux possédant les mêmes caractéristiques (Puissance et Type de Prise).

Résultat UX

Au lieu de choisir "Borne 1" ou "Borne 2", l'utilisateur choisit un "**Type de charge**" (ex: "22kW - Type 2"), et l'interface lui indique simplement combien sont disponibles (ex: "2/4").

II · gestion du temps réel cycle de vie de la réservation

Une fois la réservation effectuée, l'interface doit devenir "vivante". L'utilisateur ne doit pas avoir à recharger la page pour savoir s'il lui reste du temps.

Logique de Timers

`setInterval`

Sur la page **ReservationPage.tsx**, j'ai développé une logique de suivi qui s'adapte au statut de la réservation :

Statut ACTIVE

Un compte à rebours calcule chaque seconde le temps restant avant l'expiration de la réservation (`expires_at - now`).

Statut IN_USE

Un chronomètre affiche la durée de la charge en cours (`now - charge_started_at`).

Cette logique permet une expérience fluide et rassurante, sans solliciter le serveur à chaque seconde.



```
// Gestion des Timers (Extrait de ReservationPage.tsx)
useEffect(() => {
  if (!activeReservation) return;

  // Mise à jour de l'interface toutes les secondes
  const interval = setInterval(() => {
    if (activeReservation.status === "ACTIVE")
      // Calcul du temps restant avant expiration
      setRemainingTime(getRemainingTime(activeReservation.expires_at));
    else if (activeReservation.status === "IN_USE")
      // Calcul du temps écoulé depuis le début de la charge
      setRemainingTime(getElapsedTime(activeReservation.charge_started_at));
  }, 1000);

  return () => clearInterval(interval); // Nettoyage à la destruction du composant
}, [activeReservation]);
```



```
// Intégration du Feedback dans les Mutations (Extrait de stationDetails.tsx)
const { showToast } = useToastStore(); // Hook du store global

const reservationMutation = useMutation({
  mutationFn: createReservation,
  // Gestion centralisée du succès
  onSuccess: (data) => {
    // Redirection ou Feedback positif
    navigate(`/reservation/success/${data.id}`, { ... });
  },
  // Gestion centralisée des erreurs (Feedback non-bloquant)
  onError: (err: unknown) => {
    const message = err instanceof Error ? err.message : "Erreur inconnue";
    // Déclenchement du Toast rouge via le store
    showToast({ type: "error", message });
  },
});
```

III · qualité et feedback

refactorisation

J'ai retravaillé l'interaction utilisateur en intégrant un système de **Feedback Global**.

Pour garantir un niveau de finition professionnel, j'ai développé un système de Feedback Global via les composants `ToastContainer.tsx` et les modales de confirmation. J'ai intégré le hook

`useToastStore` dans les mutations **TanStack Query**. Désormais, chaque interaction réussie ou échouée déclenche une notification non-bloquante ("Toast"), guidant l'utilisateur sans interrompre sa navigation.

Les opérations de confirmation, modification, annulation sont gérés par des modal du composant **Modal.ts**

III · Réalisation Frontend 3 · interface d'administration et visualisation

L'administration d'une application gérant des milliers de données ne doit pas être une "boîte noire". Pour l'importation massive du fichier IRVE (plus de 45 000 lignes), j'ai développé un tableau de bord de monitoring en temps réel (**CsvImporter.tsx**) qui traduit les flux techniques asynchrones en indicateurs visuels intelligibles.

I · feedback visuel dynamique *algorithme d'interpolation*



```
// Calcul dynamique du dégradé (Extrait de CsvImporter.tsx)
const getProgressBarGradient = (percentage: number) => {
  const redRgb = [211, 47, 47]; // Rouge
  const yellowRgb = [251, 192, 45]; // Jaune
  const greenRgb = [56, 142, 60]; // Vert

  if (percentage <= 50) {
    // Interpolation mathématique du Rouge vers le Jaune (0-50%)
    const factor = percentage / 50;
    const currentColor =
      interpolateRgb(redRgb, yellowRgb, factor);
    return `linear-gradient(to right, ${red}, ${currentColor})`;
  }
  // Interpolation du Jaune vers le Vert (50-100%)
  const factor = (percentage - 50) / 50;
  const currentColor =
    interpolateRgb(yellowRgb, greenRgb, factor);
  return `linear-gradient(to right, ${red}, ${yellow} ${yellowStopPosition}%, ${currentColor})`;
};
```

Pour offrir un feedback immédiat sur la "santé" du processus d'importation, j'ai voulu aller au-delà d'une simple barre de chargement bleue standard. J'ai conçu une barre de progression dont la couleur évolue organiquement en fonction de l'avancement.

Mon algorithme frontend

J'ai implémenté une fonction mathématique d'**interpolation RGB** directement dans le composant React.

Logique

La barre démarre en **Rouge** (0%), transite par le **Jaune** (50%) et termine en **Vert** (100%).

Technique

Le composant recalcule dynamiquement la valeur CSS **linear-gradient** à chaque mise à jour du pourcentage reçu via WebSocket. Cela crée une transition fluide qui informe visuellement l'administrateur de la progression vers le succès.

II · console de monitoring et logs

Pour rassurer l'administrateur sur le bon déroulement des opérations en arrière-plan, j'ai intégré une "console" virtuelle dans l'interface.

Architecture

Le composant s'abonne au contexte d'import (**useImport**) qui reçoit les lignes de log du Backend via WebSockets.

Rendu

Les logs défilent en temps réel. J'ai appliqué un style d'opacité dégressif sur les anciennes lignes pour focaliser l'attention sur l'action immédiate tout en conservant l'historique visible.

III · monitoring, contrôle et gestion des erreurs

Contrôle de l'import

J'ai intégré des contrôles permettant à l'administrateur d'interagir avec le processus d'importation en cours. Un bouton d'annulation déclenche la fonction `isStopRequested` côté serveur, permettant un arrêt propre du worker sans laisser de transactions orphelines.



```
// client/src/components/admin/CsvImporter.tsx
// Bouton d'annulation
<button
  onClick={handleCancelImport}
  disabled={!isImporting}
  className="..."
>
  Annuler l'import
</button>

// Handler
const handleCancelImport = () => {
  if (importId) {
    // Appel API pour demander l'arrêt
    fetch(`^/api/import/${importId}/cancel`, { method: 'POST' })
  }
}

// server/src/controllers/importController.ts

// Fonction vérifiée à chaque itération
export const isStopRequested = (importUuid: string): boolean => {
  const import_ = await ImportLog.findByPk(importUuid);
  return import_.status === 'CANCELLING';
}

// Dans la boucle de traitement
if (isStopRequested(importId)) {
  stream.destroy();
  await ImportLog.update({ status: 'CANCELLED' }, { where: { id: importId } });
  return;
}
```

Gestion des erreurs visuelles

Le composant affiche également un récapitulatif des erreurs à la fin de l'import, avec le nombre de lignes réussies et échouées, permettant à l'administrateur d'identifier rapidement si un import nécessite une correction du fichier source.



```
// CsvImporter.tsx
{importResult && (
  <div className="import-summary">
    <p>✓ Lignes réussies :<br/>
      {importResult.successfulLines}</p>
    <p>✗ Lignes en erreur :<br/>
      {importResult.errorLines}</p>
    {importResult.errorLogFile && (
      <a href={`^/api/logs/${importResult.errorLogFile}`}>
        Télécharger le fichier
        d'erreurs
      </a>
    )}
  </div>
)}
```

IV · Jeu d'essai · validation et limites de l'importation

Afin de valider la fonctionnalité la plus critique de l'application (l'importation massive des données IRVE), j'ai mis en place un protocole de test basé sur un extrait représentatif du fichier source officiel (**consolidation-etalab-schema-irve-statique.csv**).

Consciente que cette fonction a été optimisée pour un schéma de données précis, ce jeu d'essai vise à vérifier le comportement nominal, la résilience aux erreurs de saisie, et à documenter honnêtement les limites actuelles face aux évolutions structurelles de l'Open Data.

I · Scénario 1, le cas nominal succès sur données conformes

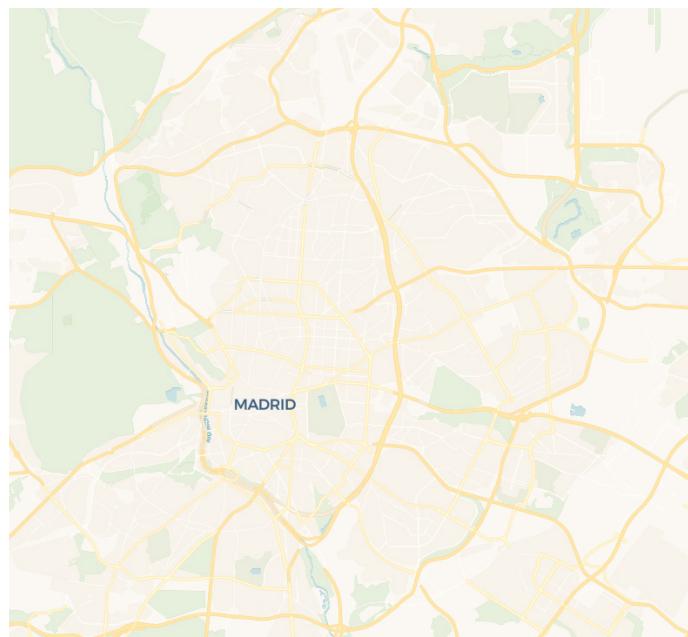
Objectif

Valider que le système crée bien de nouvelles entrées (et pas seulement des mises à jour).

Données en entrée

Un fichier **test_madrid.csv** contenant 3 stations fictives localisées à Madrid (Plaza Mayor, Bernabéu, Atocha) avec des IDs uniques (**ES001...**).

Ce jeu de données est volontairement situé hors de France pour être visuellement distinct sur la carte.



Données Obtenues

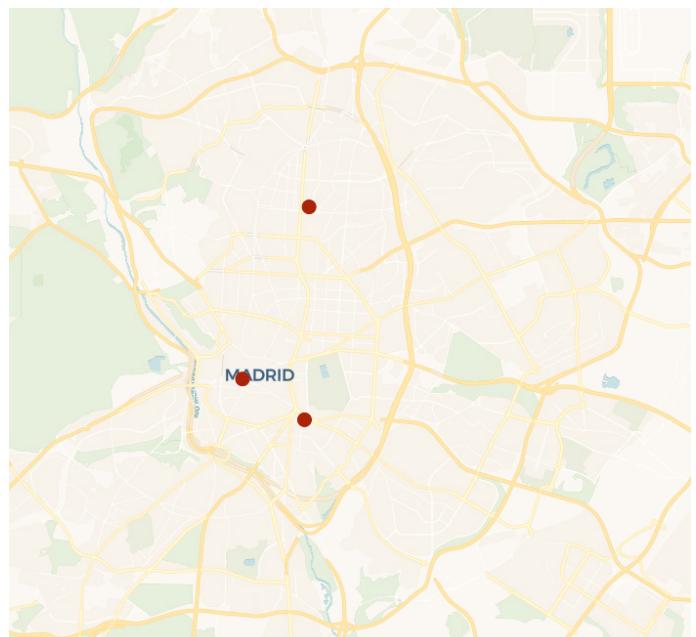
Les 3 stations madrilènes sont visibles sur la carte interactive. Les bornes sont correctement importées.

Analyse

Conforme.

Données attendues

- » Création de 3 stations et 4 terminaux en BDD.
- » Statut final : COMPLETED.
- » Apparition des marqueurs sur la carte de l'Espagne (zone vide auparavant).



Remarque

Une erreur dans l'import a été remonté. Après vérification, il s'agit d'une erreur dans la fonction qui compte les erreurs. Le script étant fait pour grouper les bornes, cela a causé souci sur le compte de borne importées. Le fonctionnement est nominal.

II · Scénario 2, résilience aux données "sales" gestion d'erreurs

Objectif

Vérifier que le code du service d'import rejette proprement une ligne invalide à cause d'une erreur de validation (ici, une coordonnées géographique impossible) sans interrompre l'importation des autres lignes de données valides.

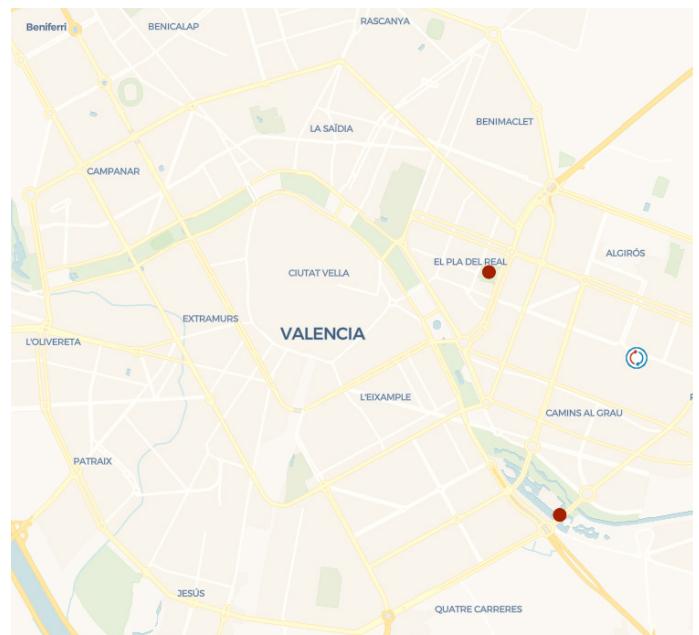
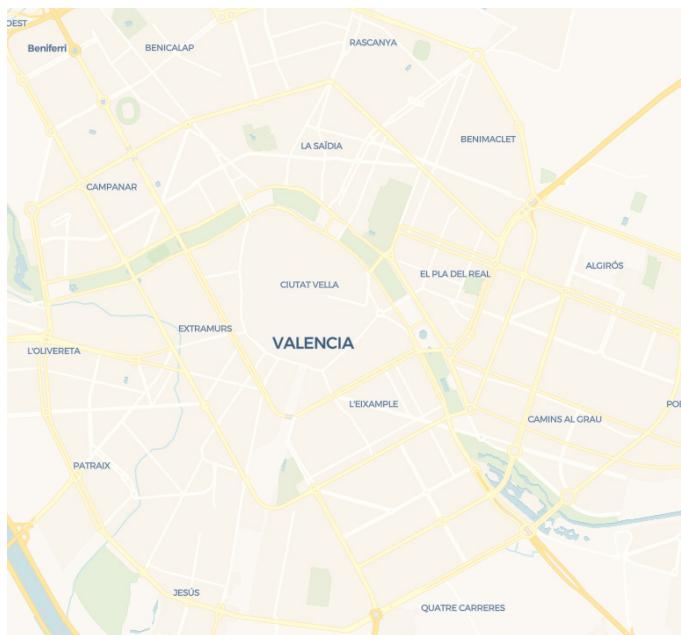
Données en entrée

Un fichier **test_valencia.csv** contenant 3 stations fictives localisées à Valence (Cité des Arts, Gare du nord et Mestalla) avec une grossière Erreur de coordonnées sur la station 2 [-0.377400, 200]

Ce jeu de données est volontairement situé hors de France pour être visuellement distinct sur la carte.

Données attendues

- » Création de 2 stations avec une ligne en Erreur.
- » Statut final : PARTIAL_SUCCESS.
- » Apparition des marqueurs sur la carte de l'Espagne (zone vide auparavant). Pas de crash.



Données obtenues

Les 2 stations valenciennes sont visibles sur la carte interactive. Les bornes sont correctement importées et le script ne s'est pas arrêté.

Erreur remontée

L'interface affiche 1 erreur et celle-ci a été isolée de l'import. L'import portait l'indentifiant **import_errors_10220e9f-9044-4b94-a855-06268114d3e7.log**. L'Erreur a été inscrite comme **16:42:56 | Ligne 2**
| Type: INVALID_GEOJSON_FORMAT
| Message: Format GeoJSON invalide pour coordonneesXY. | Colonne CSV:
coordonneesXY | Valeur: "[-0.377400, 200]"

Analyse

Conforme.

Remarque

R.A.S

III · Scénario 3, mise à jour de données update

Objectif

Vérifier que la ré-importation d'un fichier CSV contenant des identifiants de stations/terminaux déjà existants déclenche correctement la mise à jour des attributs modifiés (logique findOrCreate), sans créer de doublons.

Données en entrée

Un fichier **test_madrid_maj.csv** contenant une mise à jour de données sur la station **Madrid - Plaza Mayor** avec un changement horaire **24/7 » 24/7 365 j/an**

Ce jeu de données est volontairement situé hors de France pour être visuellement distinct sur la carte.

Données attendues

- » Lancement de l'import
- » Statut final : COMPLETED.
- » Mise à jour de la station

Madrid - Plaza Mayor

Plaza Mayor 28012 Madrid

Implantation: Voirie

Horaires: 24/7

Accès PMR: Oui

Points de charge: 2

Paiement par CB: Oui

Autre paiement: false

Tarification: 0.30€/kWh

Observations: Station Test Madrid Centre

Madrid - Plaza Mayor

Plaza Mayor 28012 Madrid

Implantation: Voirie

Horaires: 24/7 365 j/an

Accès PMR: Oui

Points de charge: 2

Paiement par CB: Oui

Autre paiement: false

Tarification: 0.30€/kWh

Observations: Station Test Madrid Centre

Données obtenues

Le champ horaires affiche bien 365 j/an
Aucune duplication dans le processus d'importation

Analyse

Conforme.

Remarque

La question de la création d'une station en doublon lors d'un nouvel import a été pris en compte au début du développement.
En effet, c'est un point que j'ai balisé dès le début en liant l'UUID de la station à son nom mixé à ses coordonnées GPS. Créant donc une identification unique que le script reconnaît très bien à chaque import. Sauf changement primaire dans la données, une station ne peut être dupliquée lors d'une opération de mise à jour de la data. Cela est également vrai pour toutes les autres informations comme la marque, le gestionnaire et les prises disponibles.

IV · Scénario 4, cycle de vie de la réservation actif

Objectif

Vérifier le succès du processus de réservation, la redirection vers la page de confirmation avec le minuteur de validité, le lancement de la charge avec le timer de fin et la capacité du système à bloquer les tentatives de double réservation pour un même utilisateur.

Étapes de l'Essai

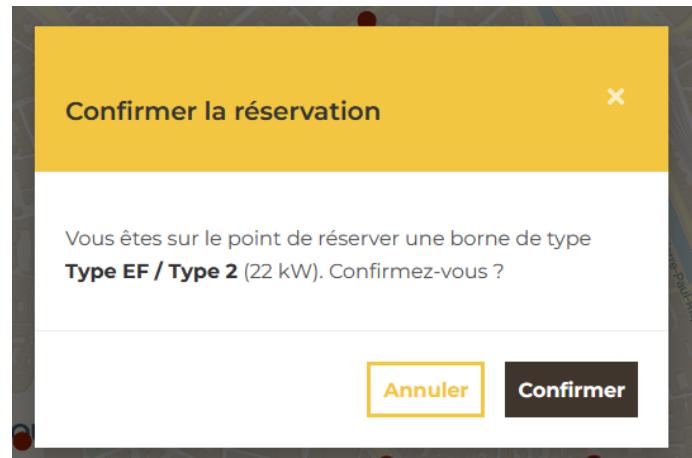
» Lancement de la réservation

Choix de la borne, modal de confirmation, redirection vers ReservationSuccessPage.tsx

» La page donne une indication dynamique de la limite de temps pour la réservation et invite à lancer la charge.

» une fois sur la page réservation, les informations sont présente et on peut lancer la charge avec le timmer de fin.

» pendant ce temps, on essaye de lancer une autre réservation.



TOULOUSE - François Verdier

Boulevard Lazare Carnot, 31000 TOULOUSE

Implantation: Voirie

Horaires: Mo-Su 00:00-23:57

Accès PMR: Oui

Points de charge: 2

Paiement par CB: Non

Autre paiement: False

Choisir ma borne

Réserver **Annuler** **Supprimer la station**

*Le bouton "supprimer la station" n'est disponible qu'en tant qu'administrateur

Réservation confirmée !

Votre borne est prête et vous attend. Voici un résumé de votre réservation.

TOULOUSE - François Verdier

Boulevard Lazare Carnot, 31000 TOULOUSE

Borne

Puissance: 22 kW

Prises: Type EF, Type 2

Réservation

Date: 23 novembre 2025

Statut: RÉSERVÉE

Votre borne est réservée jusqu'au **23/11/2025 à 18:21**.

Voir toutes mes réservations

En cours

Borne réservée

Station

Boulevard Lazare Carnot, 31000 TOULOUSE

Borne	Réservation
Puissance: 22 kW Prises: Type EF, Type 2	Date: 23 novembre 2025 Statut: RÉSERVÉE

La réservation expire dans **28m 59s**

Commencer la charge **Annuler**

Charge en cours

Station

Boulevard Lazare Carnot, 31000 TOULOUSE

Borne	Réservation
Puissance: 22 kW Prises: Type EF, Type 2	Date: 23 novembre 2025 Statut: EN CHARGE

Temps de charge : **00:01:59**
Fin de charge prévue à : 19:37:56

Arrêter la charge

QPARK - TOULOUSE - JEANNE DARC

8 PLACE JEANNE DARC 31000 TOULOUSE

Implantation: Voirie

Horaires: 24/7

Accès PMR: Oui

Points de charge: 1

Paiement par CB: Non

Autre paiement: true

Observations: Conditions accès aux bornes dépendantes des accords et formules de votre EMSP

Choisir ma borne



Réserver

Annuler

Supprimer la station

Vous avez déjà une réservation en cours.

Données obtenues

La procédure de réservation a fonctionné de façon nominale. La modal de confirmation s'est bien affichée, la redirection s'est effectuée, les données dynamiques de réservation se sont bien affichées (exemple le **minuteur de 30 minutes** avant expiration de la réservation(**calcul expires_at - createdAt** (30 minutes) est correct)).

La double réservation a été empêchée, le message (Vous avez déjà une réservation en cours.) s'est bien affiché.

Analyse

Conforme.

Remarque

R.A.S fonctionnement optimal

V · Scénario 5, début et fin de charge

Objectif

Valider la transition des statuts de la réservation depuis l'état ACTIVE (Réservée) à IN_USE (En charge) puis à COMPLETED (Terminée), et le retour à la disponibilité de la borne.

Étapes de l'essai

- » **Début de la charge** passage de **Borne Réservée à Charge en cours** (déjà validé dans scénario 4). **ACTIVE** à **IN_USE** en BDD.
- » **Fin de charge**, passage de **Charge en cours** à **Aucune réservation** La réservation disparaît de l'état actif pour le client. **IN_USE** à **COMPLETED** en BDD.
- » **Vérification de la disponibilité**. La station doit redevenir **Disponible** et visible comme telle sur la carte pour tous les utilisateurs.

The screenshot shows a mobile application interface for managing a charging session. At the top, it says "En cours". Below that, under "Charge en cours", it displays the station details: "104 AVENUE DE GRANDE BRETAGNE, 31300 TOULOUSE". It also shows the "Borne" and "Réservation" sections with specific details: "Puissance: 300 kW", "Prises: Combo CCS, Type 2, Chademo", "Date: 23 novembre 2025", and "Statut: EN CHARGE". Below this, it shows the time remaining: "Temps de charge : 00:00:07" and the end time: "Fin de charge prévue à : 18:26:24". A yellow button at the bottom right says "Arrêter la charge".

RELAIS TOULOUSE GRANDE BRETAGNE

104 AVENUE DE GRANDE BRETAGNE, 31300 TOULOUSE

Implantation: Voirie

Horaires: 24/7

Accès PMR: Oui

Points de charge: 7

Paiement par CB: Non

Autre paiement: True

Choisir ma borne



Réserver

Retour à la carte

Supprimer la station

En cours

Aucune réservation active

Vous n'avez pas de réservation en cours. Trouvez une borne et réservez-la dès maintenant !

[Trouver une borne](#)

Historique

23 novembre 2025	TERMINÉE
Station	
104 AVENUE DE GRANDE BRETAGNE, 31300 TOULOUSE	
Borne	Réservation
Puissance: 300 kW	Date: 23 novembre 2025
Prises: Combo CCS, Type 2, Chademo	Statut: TERMINÉE
23 novembre 2025	TERMINÉE
23 novembre 2025	TERMINÉE
23 novembre 2025	TERMINÉE
23 novembre 2025	TERMINÉE

Un problème ?

Si vous rencontrez un souci avec une réservation, n'hésitez pas à nous le signaler.

[Contactez-nous](#)

RELAIS TOULOUSE GRANDE BRETAGNE

104 AVENUE DE GRANDE BRETAGNE, 31300 TOULOUSE

Implantation: Voirie

Horaires: 24/7

Accès PMR: Oui

Points de charge: 7

Paiement par CB: Non

Autre paiement: True

Choisir ma borne



[Réserver](#)

[Retour à la carte](#)

[Supprimer la station](#)

Données obtenues

Début de charge

Transition réussie vers l'état **IN_USE**. Le composant **ReservationPage.tsx** affiche le **Temps écoulé**.

Fin de charge

Transition réussie vers l'état **COMPLETED** (**Terminée** pour le client).

Disponibilité

La station/borne est immédiatement **Disponible** sur la carte.

Analyse

Conforme.

Le cycle de vie complet de l'utilisation d'une borne (ACTIVE » IN_USE » COMPLETED) est validé.

Remarque

R.A.S fonctionnement optimal

V. DÉPLOIEMENT ET INDUSTRIALISATION

I · Genèse

Dès le lancement du projet, j'avais pour idée de déployer le projet. Étant autodidacte dans l'infrastructure et la gestion serveur depuis 10 ans, je voulais fortement comprendre comment les projets étaient déployés avec les **pipelines CI/CD**. À partir du moment où la migration a été mise en place pour le BDD (étape qui marquait la fin de la refactorisation du backend). J'ai prévu **3 environnements** pour tester le projet et faire en sorte de prévoir les bugs avant le déploiement. Il y avait l'environnement **dev** (démarrable avec **npm run dev**), **staging** (**docker compose up -d**) et **prod** avec les scripts **deploy-backend.yml** & **deploy-frontend.yml** arrivés bien plus tard. L'idée étant de se rapprocher le plus possible d'un environnement **ISO-Prod**. A n'importe quel moment, nous pouvions vérifier le fonctionnement de notre application sur docker et donc préparer le déploiement. La configuration est assurée par des fichiers .local (**.env.development.local** & **.env.staging.local**) et des **github secrets** pour la production. J'ai personnellement beaucoup travaillé sur l'environnement de dockérisé. Ce qui a parfois posé des soucis de compréhension avec mon camarade. Ayant une version de code et de BDD plus récente. Mais cela nous a permis de détecter quelques bugs grâce à ce fonctionnement.

II · Docker

C'était une solution toute trouvée pour ce projet. **Facile à configurer, beaucoup d'image préconçue, architecture micro-service, persistance des données et configuration automatique d'un réseau interne**. Docker c'était vraiment une super solution pour l'environnement ISO-Prod. Mon collègue étant débutant sur Docker, je ne voulais pas non plus partir sur une configuration à base de minikube qui l'aurait perdu. C'était vraiment la solution idéale pour s'assurer que le projet aurait un déploiement facilité.

III · L'Automatisation (CI/CD) avec GitHub Actions

Le déploiement s'effectue grâce à une **pipeline** pour le **Back** et le **FrontEnd**. Cela assure une meilleure optimisation et une sécurité en cas de défaut sur un des builds. Cela assure également de ne build que le côté qui a reçu une modification et d'éviter les temps de build trop long. Les deux scripts ont la même fonctionnement global avec quelques particularités.

L'action se lance

Uniquement si l'il y a un **push** ou une **pullrequest** sur la branche **staging** du répo.

On vérifie tout

On va lancer node et **installer toutes les dépendances** du répo **npm ci**, pour ensuite **vérifier le code** (lint, types, etc) avec **npm run check**. Si tout est bon, un **audit de sécurité** est lancé avec la commande **npm audit --audit-level=medium** pour s'assurer qu'il n'y a **pas de vulnérabilité** sur des paquets.

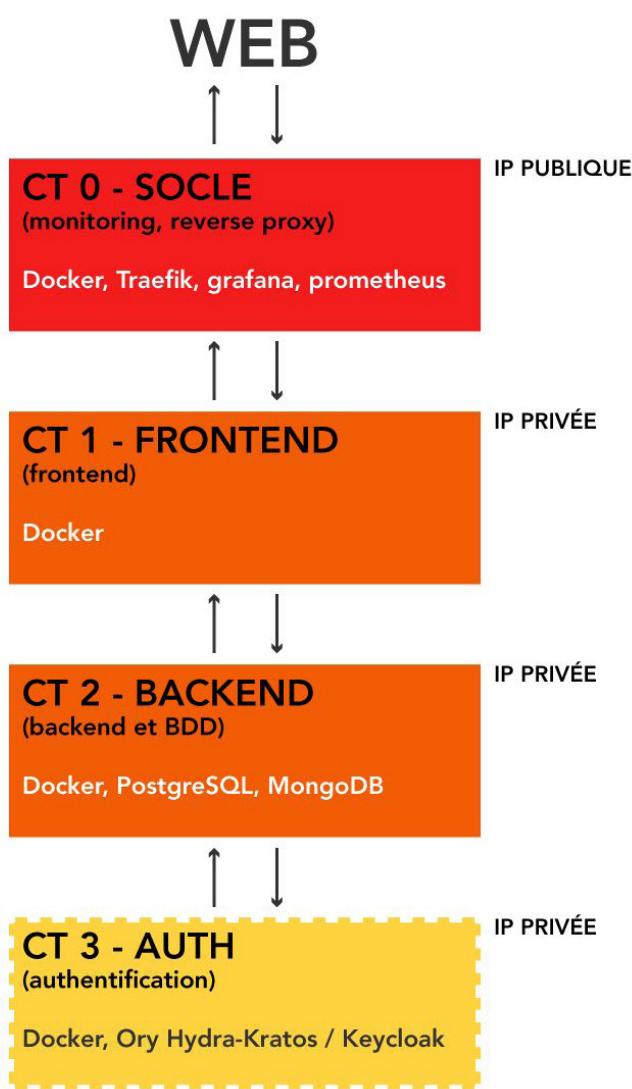
On construit l'image

Pour chaque script, on va focus ou sur **./serveur** ou sur **./client** pour le build. Pour la configuration, le script va se connecter à mon compte github grâce à mes **Personal access tokens**. Il va build l'image avec comme nom **server:latest** ou **client:latest** et va l'**envoyer à GHCR (GitHub Container Registry)** sous le nom **ghcr.io/camille-ricon/*****:latest**. Le build est **versionné** avec la **signature** du **SHA** du **commit (github.sha)**. Ce qui permet donc un retour en arrière facile en cas de bug.

Déploiement

Une fois le build prêt, l'image est tirée par le **docker-compose.prod.yml** de chaque CT à partir de GHCR et remplace la précédente image tirée. Les migrations sont assurées par **entrypoint.sh**. Ce n'est pas moins de **21 secrets github** qu'il faut pour **déployer** le **back** et le **FrontEnd sur mon infrastructure**.

IV · Infrastructure



Mon infrastructure est composée de **3 CT sous Debian 13 toutes dockérisées**. Le Frontend et le Backend ne sont pas directement exposé au web. Grâce à des proxy interne et l'utilisation de **Traefik**, je n'ai pas besoin de produire plusieurs configuration. Les **certificats SSL** sont assurés par lui et j'utilise **Nginx** comme **reverse proxy** dans chaque image.

Les **secrets** sont convertis en fichier **.env** exécutés sur chaque machine et limité à leur propre image. Ce qui limite grandement les soucis de sécurité et de configuration.

Contrairement à l'environnement dev/staging, **PostgreSQL n'est pas redéployé** à chaque fois. **Une seule version** est utilisée et maintenue.

PS : trouvant l'authentification par Token (**JWT**) peu sécurisé, j'ai entamé la recherche d'une meilleure solution d'authentification. Après discussion avec des collègues développeurs, mon choix s'est arrêté sur les solutions **d'ORY**. Une quatrième CT (AUTH) verra donc le jour prochainement. J'en parle car c'est **une possibilité d'évolution pour ce projet**. Et cela sera une évolution nette pour les miens.

VI. MÉTHODOLOGIE, ORGANISATION ET VEILLE

I · gestion de projet et travail collaboratif

Organisation et suivi des tâches Jira

Nous avons utilisé Jira comme outil central de pilotage.

Backlog

Nous avons d'abord effectué un tri complet du backlog initial, en reformulant les besoins et en redéfinissant les priorités.

Découpage

Le projet a été segmenté en 22 Epics, correspondant à nos User Stories principales. Chaque Epic contenait plusieurs tickets techniques.

Répartition

Initialement, nous nous sommes répartis les rôles par domaine (moi sur le Backend, Jonathan sur le Frontend). Cependant, au fil du projet, nous avons adopté une approche plus "Full Stack", travaillant ensemble sur les deux volets selon les besoins du sprint.

<input type="checkbox"/>	Tickets	Personne assignée	Priorité	↓	État
<input type="checkbox"/>	⚡ P3-21 Je peux mettre à jour toute la liste des bornes électriques (Admin)	CT Constance-Tlse	↗ Highest	DEV ▼	
<input type="checkbox"/>	⚡ P3-20 Je peux accéder à une carte qui contient les bornes électriques...	CT Constance-Tlse	↗ Highest	DEV ▼	
<input type="checkbox"/>	⚡ P3-17 Je peux accéder au panel d'administration (Admin)	CT Constance-Tlse	↗ Highest	DEV ▼	
<input type="checkbox"/>	⚡ P3-4 Je peux accéder à la page d'accueil	JD jonathan dias	↗ Highest	DEV ▼	
<input type="checkbox"/>	⚡ P3-26 Je peux avoir accès à une barre de recherche	JD jonathan dias	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-25 Je peux être géolocalisé.	JD jonathan dias	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-12 Je peux réserver une borne	CT Constance-Tlse	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-8 Je peux me connecter	JD jonathan dias	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-7 Je peux m'inscrire	JD jonathan dias	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-6 Je peux voir les bornes de recharges autour de moi	JD jonathan dias	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-5 Je peux mettre à jour mon profil	CT Constance-Tlse	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-4 Je peux me déconnecter	CT Constance-Tlse	↗ High	DEV ▼	
<input type="checkbox"/>	⚡ P3-56 Je peux récupérer mon mot de passe oublié	JD jonathan dias	↔ Medium	DEV ▼	
<input type="checkbox"/>	⚡ P3-28 Je peux filtrer les bornes sur la carte	JD jonathan dias	↔ Medium	DEV ▼	

Rituels et communication

Bien que nous travaillions souvent dans la même pièce, facilitant les échanges directs, nous avons maintenu des rituels stricts

Discord

Utilisé pour le partage de ressources et la communication asynchrone.

Daily meetings

Ces points quotidiens étaient cruciaux, compte tenu de notre forte cadence de production. Et moi qui travaillais intensivement sur le Back-End, y compris hors des heures de cours. Compte tenu de la complexité du projet, il était important de savoir où en était l'autre à tout moment. Surtout quand j'étais en remote pendant quelques semaines.

La retrospective

En fin de semaine, on se lançait dans une retrospective du travail effectué. C'était l'occasion de commenter et préparer les futurs sprint.

Workflow de développement Git Flow

Nous avons mis en place un Git Flow strict pour éviter les conflits et garantir la qualité du code sur la branche principale **dev**

Feature Branch

Création d'une branche par fonctionnalité.

Pull Request (PR)

Ouverture d'une PR vers la branche de développement (dev) une fois la fonctionnalité terminée.

Code Review

Validation obligatoire par le binôme (et initialement par le formateur) avant le merge.

Autonomie

En fin de projet, au vu de notre avancement par rapport au reste de la promotion, nous avons obtenu les droits pour valider nos propres PR, nous responsabilisant sur la qualité du code fusionné.

Gestion des erreurs

L'interdiction du "push direct" sur staging ou dev a été respectée (après quelques erreurs d'inattention corrigées via **git push --force**, qui ont servi de leçons définitives sur la gestion des historiques Git).



il y en a qui ont essayé mais ils ont eu des problèmes :(

II · la veille technologique les choix architecturaux

La complexité du projet nous a imposé de valider nos choix techniques par une veille active, en consultant des sources variées (documentation officielle, StackOverflow, MDN, W3Schools, cours Odysee/Wild Code School, collègues développeurs).

I · l'architecture de données, le choix de l'ORM

Problématique

Comment manipuler une base de données complexe et garantir le typage fort (TypeScript) sans perdre de temps au vu du projet

SQL pur (pg / node-postgres)

Bien que performant, écrire le SQL à la main pour le nettoyage et l'insertion de 45 000 lignes aurait demandé un temps de développement excessif et une maintenance lourde.

Prisma

Très populaire et moderne. Cependant, ma veille a révélé qu'il nécessitait l'apprentissage d'un langage de schéma spécifique (DSL). Compte tenu de la durée limitée du projet, la courbe d'apprentissage représentait un risque.

Drizzle ORM

Une solution prometteuse testée en phase de recherche, mais écartée car l'écosystème et la documentation semblaient moins matures à l'époque pour nos besoins spécifiques.

Solution retenue

Sequelize.

Justification

J'ai opté pour Sequelize couplé à sequelize-typescript. C'est un ORM mature, doté d'une documentation exhaustive. Sa robustesse pour gérer les transactions et les validations de modèles (indispensables pour notre script d'importation) a été décisive. Ce choix pragmatique nous a permis de démarrer le développement immédiatement, en capitalisant sur mes connaissances existantes pour tenir les délais.

II · La Performance géographique, PostgreSQL + PostGIS

Problématique

L'application doit pouvoir filtrer et afficher des milliers de points de charge autour de l'utilisateur instantanément.

Calculer ces distances côté serveur en JavaScript sur 45 000 entrées aurait saturé le processeur à chaque requête.

Recherche

Je me suis renseignée sur les moteurs de base de données capables de gérer nativement la spatialité. Si MariaDB/MySQL offrent des options, **PostgreSQL** est la référence industrielle grâce à son extension **PostGIS**.

Mes recherches ont confirmé que PostGIS permet d'indexer les colonnes géographiques (Index GIST) et d'exécuter des requêtes spatiales (**ST_DWithin**, **ST_Distance**) directement dans le moteur de base de données, de manière infiniment plus rapide que n'importe quel script JS.

Solution retenue

Migration vers **PostgreSQL avec PostGIS**.

Ce choix a nécessité une adaptation du monorepo (validée avec le référent technique), mais il a transformé la base de données en un véritable moteur de calcul spatial, garantissant la fluidité de la carte côté Frontend.

III · sécurisation de l'authentification, JWT & HttpOnly

Problématique

Comment persister la session utilisateur de manière sécurisée côté client ?

Recherche

Nous savions que le stockage du Token JWT dans le localStorage exposait l'application aux failles XSS (Cross-Site Scripting), car n'importe quel script JS peut y accéder.

Solution retenue

Nous avons opté pour le stockage du JWT dans un Cookie HttpOnly.

Ce cookie n'est pas accessible via JavaScript, neutralisant les attaques XSS.

Côté Backend, nous avons implémenté une architecture basée sur des **Middlewares** ("Auth Controller"). Plutôt que de sécuriser chaque route individuellement, nous avons créé un "mur" (Wall) dans le routeur : toutes les routes définies après ce middleware nécessitent obligatoirement une authentification valide. Cette approche, similaire à notre middleware **isAdmin**, garantit qu'aucun endpoint critique n'est oublié et simplifie la maintenance.

IV · performance cartographique, MapLibre vs Leaflet

Problématique

Afficher plus de 45 000 points de recharge sans ralentir le navigateur. Recherche : La librairie standard Leaflet gère les marqueurs comme des éléments du DOM. Avec des milliers de points, le DOM devient lourd, entraînant des ralentissements majeurs.

Solution retenue

Nous nous sommes tournés vers **MapLibre GL JS**.

Contrairement à Leaflet, MapLibre utilise **WebGL** pour le rendu graphique via la carte graphique (GPU).

Elle permet l'utilisation de **tuiles vectorielles** (Vector Tiles), ce qui est beaucoup plus performant pour charger et afficher de grands volumes de données géographiques dynamiques.

Ce choix, couplé à notre base de données PostGIS, a permis de maintenir une fluidité de navigation optimale malgré la densité des données.

III · perspectives d'évolution

Bien que l'application soit fonctionnelle et performante pour une démonstration technique, le contexte de l'examen (délais courts, équipe réduite) nous a contraints à prioriser le développement des fonctionnalités "cœur" au détriment de certaines finitions.

Si le projet devait être industrialisé pour une vraie mise en production, voici la feuille de route prioritaire que j'ai identifiée

I · qualité logicielle et robustesse, priorité absolue

C'est le chantier le plus critique

Tests automatisés

Actuellement, aucune fonctionnalité ne bénéficie de tests unitaires ou d'intégration (E2E). Cette absence de filet de sécurité a causé plusieurs régressions fonctionnelles durant le développement. La priorité serait de couvrir les parcours critiques (Import, Réservation, Auth) avec des tests rigoureux. Encore que l'import est sans doute la feature la plus testée du projet.

Gestion des environnements

Améliorer la stratégie de conteneurisation pour conserver les anciennes images Docker, ce qui permettrait un rollback instantané en cas de déploiement défectueux.

Chasse aux bugs graphiques

Une passe de finition est nécessaire sur le Frontend, notamment pour corriger des défauts d'affichage persistants à l'image de la vue de certaines stations.



Ici des bornes avec trop de plug déforment la zone de sélection.

II · sécurité et authentification

Refonte de l'auth

Le système actuel doit être renforcé. L'objectif est d'implémenter les Refresh Tokens pour une meilleure gestion des sessions, ainsi que l'authentification à double facteur (2FA) pour sécuriser les comptes utilisateurs et administrateurs.

Mieux filtrer ce qu'on reçoit.

Mise en place d'une validation stricte des informations utilisateur à l'inscription pour bloquer les informations aberrantes. Même si le back est protégé contre les attaques qui viseraient la BDD. Actuellement, l'utilisateur est trop libre sur les informations qu'il donne. Exemple, j'ai pu m'inscrire avec une date de naissance en 1553.

III · fonctionnalités utilisateur et communication

Emails fonctionnels

Actuellement, l'utilisateur n'est pas notifié hors de l'app. Il faut développer de nouveaux emails pour confirmer chaque étape clé comme. l'inscription, alerte de connexion suspecte, confirmation de réservation, facture de fin de charge, etc.

Support client

La structure de base de données (table support) a été anticipée et existe déjà, mais l'interface de gestion des tickets et le back-office associé restent à développer intégralement.

Administration avancée

Ajouter des fonctionnalités de gestion pour le suivi précis des utilisateurs et permettre la suppression de compte en "self-service" (RGPD).

Gestion du RGPD et mise en conformité

Actuellement il n'y a pas d'auto-suppression des compte après 2 ans sans nouvelle, pas de demande d'accord pour l'utilisation des données personnelles dans le cadre du service. Pas de charte RGPD. Le projet, hors du fait qu'il est une démo sympathique de certification. Ne pourrais pas sortir en l'état.

IV · intégrité et uniformisation des données

Correction des données sources

J'ai identifié que certaines bornes sont importées sans être correctement liées à leurs prises (problème de structure dans l'Open Data). Une investigation approfondie du script d'import est nécessaire pour traiter ces cas limites.

Internationalisation des horaires

Les horaires fournis par l'Open Data sont hétérogènes et souvent en anglais. J'ai déjà développé un script de normalisation/traduction, mais il a été conçu sur une version antérieure de la base de données. Il doit être mis à jour, testé et déployé pour afficher des horaires uniformisés en français.

V · vers une application commerciale

Liason réelle avec les bornes

Pour rendre le système opérant, il faudrait qu'il soit lié aux vraies bornes. Cela demanderait un sacré développement en plus. Surtout niveau sécurité

Solution de paiement

Dans le cadre d'un vrai développement, il nous faudrait avoir une solution de paiement aussi. Au vu de la complexité, cela viendrait probablement d'un prestataire. Dans ce cadre, je pense également qu'il faudrait plusieurs audit de sécurité sur l'ensemble du projet et une équipe bien plus grande, encadrée par des dev séniors.

IV · bilan du projet

Bilan technique

Ce projet a été un véritable défi technique qui m'a poussée dans mes retranchements.

Maîtrise de la data

J'ai appris à être "Reine de ma Data". L'utilisation stricte du typage (TypeScript) nous a épargnés de nombreux bugs. J'ai dû imposer des choix structurants, comme l'utilisation de l'ORM Sequelize pour le nettoyage et l'insertion des données, là où du SQL pur aurait été ingérable pour une petite équipe dans le temps imparti.

Ambition technique

Nous n'avons pas choisi la facilité. L'intégration de PostGIS, de MapLibre et des tuiles vectorielles a demandé un investissement énorme, mais a permis de livrer un produit performant. Je sors de ce projet avec le sentiment d'être devenue une véritable développeuse fullstack (avec une préférence pour la backend évidente <3).

Bilan humain

J'ai endossé le rôle de Lead Tech, une position exigeante qui a nécessité de faire des choix d'architecture complexes et de porter la responsabilité de la stabilité du Backend.

Travailler en binôme a demandé une grande confiance mutuelle. J'ai choisi une architecture ambitieuse, parfois difficile à appréhender pour mon binôme, mais la communication a permis de garder le cap.

S'imposer en tant que Lead Tech sur un projet aussi dense a demandé une charge de travail considérable, frôlant parfois la saturation, mais je suis fière d'avoir mené ce navire à bon port et d'avoir livré une application fonctionnelle sur un sujet jugé "prestigieux" et difficile par notre formateur.