

Dossier Professionnel



MINISTÈRE CHARGÉ
DE L'EMPLOI



Concepteur / Développeur d'applications

Jérémy Breuillard – CS2

Sommaire

Remerciements

Introduction

Résumé

- Liste des compétences couvertes par le projet

- Lexique

Présentation

- Présentation personnelle

- Présentation du projet

Cahier des charges

Organisation du projet

Conception du front-end

- Wireframe

- Contenu

- Environnement de travail

- Arborescence

Développement du front-end

- Partie mobile

- Partie web

Conception du back-end

- Mise en place de la base de données

- Concevoir la base de données

Développement de la partie back-end

- Arborescence

- API

- Architecture

- Sécurité

Préparer et exécuter le déploiement d'une application

- Machine Virtuelle

- Alwaysdata

Exemple de jeu de manipulation de données

Exemple de composants métiers : React-Native

Développer une interface utilisateur de type desktop

Conclusion

Remerciements

Je remercie Paul Poulain le gérant de l'entreprise Biblibre de m'avoir donné la chance d'exercer ce métier et mon tuteur Julian Maurice de m'avoir appris de nombreuses choses dans le cadre du développement web et personnel.

Je remercie mes camarades de classe avec qui j'ai eu l'opportunité d'apprendre et de créer dans une ambiance agréable. Je remercie aussi Shlimon David avec qui j'ai élaboré le projet qui va suivre et qui m'a partagé sa vision du développement.

Je remercie aussi l'école La Plateforme qui m'a accepté en formation CS2

Introduction

- Résumé

Ce projet consiste à référencer les bornes électriques installées à Marseille grâce à une application. Nous avons choisi de simuler un projet qui répondrait aux attentes d'une entreprise de déploiement de bornes électriques.

Cette application contient un site web qui est l'interface back-office et d'une application mobile. Les langages de développement qu'on a utilisé pour le back-office et la partie mobile sont le HTML/CSS, Javascript, NodeJS et les frameworks sont principalement le Pug, React-Native, TailwindCSS, expressJS. L'IDE utilisé est Visual Studio Code.

Le service informatique aura donc un compte administrateur sur le back-office pour créer des emplacements de bornes électriques dans lesquelles il faudra y installer des bornes.

L'administrateur devra se connecter pour avoir possibilité de voir la liste de toutes les bornes électriques et il pourra, s'il le décide, créer une nouvelle borne, modifier, ou la supprimer via un formulaire post. La même fonctionnalité existe pour créer des bornes électriques.

Les techniciens chargés d'installer les bornes électriques devront passer par l'application pour se rendre à un emplacement de borne existant et renseigné sur la fonctionnalité Map après s'être connecté avec leur nom et prénom sur l'application.

- Liste des compétences du référentiel couvertes par le projet

Maquetter une application

La modélisation s'est faite avec Figma : nous avons réalisé un wireframe et une maquette du site web ainsi que de l'application mobile

Développer une interface utilisateur de type desktop

La création d'un Sokoban dans le langage Python m'a permis de réaliser ma première application Desktop, c'est un petit jeu vidéo de réflexion japonais assez rapide à mettre en place

Développer des composants d'accès aux données

En me connectant à une API je peux faire un CRUD en JSON qui me permet de récupérer des données de manière sécurisée. De même que récupérer des données en base par le biais d'une requête SQL

Développer la partie front-end d'une interface utilisateur

Pour accéder à mon site je passe par localhost sur le navigateur qui m'affiche la page à base de composants WEB que j'ai créé visuellement

Développer la partie back-end d'une interface utilisateur

La BDD faite avec phpMyAdmin puis nous avons créé une API pour avoir des données en JSON. Nous avons utilisé NodeJS.

Concevoir une base de données

On peut utiliser le logiciel en ligne Whimsical pour modéliser une base de données : La méthode Merise nous sert à présenter de manière différente (MCD, MLD, MPD) la structure de la BDD afin que celle-ci soit lisible par le client comme pour le développeur

Mettre en place une base de données

En utilisant PhpMyAdmin il est facile et rapide de créer des tables et des relations entre elles. L'accès ne doit pas être public et doit être sécurisé de plusieurs manières pour empêcher les intrusions

Développer des composants dans le langage d'une base de données

Revient à créer des requêtes en SQL qui interagissent avec la BDD

Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement

Les outils GIT tels que Gitlab ou Github s servent à faire du versionning de logiciel ; c'est-à-dire qu'il stocke sur un espace sécurisé accessible les différentes versions d'un projet.

Concevoir une application

Revient à rassembler la partie front-end avec la partie back-end pour former l'application ; et les rendre compatibles.

Développer des composants métiers

Le propre du travail de concepteur/développeur d'applications : créer des composants réutilisables qui peuvent servir à tous sans avoir à modifier le code de base.

Construire une application organisée en couches

Le front-end est articulé par des composants qui façonnent des pages ; tout part du menu puis il faut ensuite empiler des pages pour naviguer ou dépiler des pages pour revenir à la couche/page précédente

Développer une application mobile

Avec le framework React-Native nous avons pu développer et voir le résultat sur mobile/emulateur Android directement

Préparer et exécuter les plans de tests d'une application

Nous avons créé des tests unitaires sur Postman et avec du Bash pour vérifier la sécurité de notre application

Préparer et exécuter le déploiement d'une application

Au cours de l'année scolaire nous avons réalisé un déploiement de VM à l'aide d'un terminal et d'une interface graphique

- Lexique

SGBD : Système de Gestion de Base de Données est un logiciel système destiné à stocker et à partager des informations dans une base de données, en garantissant la qualité, la pérennité et la confidentialité des informations, tout en cachant la complexité des informations.

phpMyAdmin : phpMyAdmin est une application Web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB, réalisée principalement en PHP et distribuée sous licence GNU GPL.

CRUD : L'acronyme informatique anglais CRUD (pour create, read, update, delete) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données. Plus généralement, il désigne les opérations permettant la gestion d'une collection d'éléments.

SQL : Structured Query Language. Langage de requêtes, basé sur l'algèbre relationnelle, utilisé pour manipuler les données dans une base de données relationnelles.

XSS : Le cross-site scripting est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page.

Cookie : Il s'agit d'un fichier texte généré par le serveur du site web que vous visitez ou par le serveur d'une application tierce (régie publicitaire, logiciel d'analyse du trafic internet, etc.). Il ne pourra par la suite être réutilisé que par le serveur qui l'a déposé en premier lieu.

MindMap : Permet de représenter visuellement et de suivre le cheminement associatif de la pensée.

Interface : Dispositif matériel ou logiciel qui permet à un usager d'interagir avec un produit informatique. C'est une interface informatique qui coordonne les interactions homme-machine, en permettant à l'utilisateur humain de contrôler le produit et d'échanger des informations avec le produit.

Agile (Scrum) : La méthode Scrum tire son nom du monde du rugby, scrum = mêlée. Le principe de base étant d'être toujours prêt à réorienter le projet au fil de son avancement. C'est une approche dynamique et participative de la conduite du projet. La réunion dans la méthode Scrum relaie la métaphore.

Primary key (clé primaire) : Une clé primaire est la donnée qui permet d'identifier de manière unique un enregistrement dans une table. Une clé primaire peut être composée d'une ou de plusieurs colonnes de la table.

API REST : (également appelée API RESTful) est une interface de programmation d'application (API ou API web) qui respecte les contraintes du style d'architecture REST et permet d'interagir avec les services web RESTful

API : Ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. Elle est parfois considérée comme un contrat entre un fournisseur d'informations et un utilisateur d'informations, qui permet de définir le contenu demandé au consommateur (l'appel) et le contenu demandé au producteur (la réponse).

Architecture REST : (Representational State Transfer) est un ensemble de principes architecturaux adapté aux besoins des services web et applications mobiles légers. La mise en place de ces recommandations est laissée à l'appréciation des développeurs. Une application est dite RESTful lorsqu'elle respecte six recommandations architecturales.

Architecture SOAP : (Simple Object Access Protocol) est un protocole standard initialement conçu pour que des applications développées avec différents langages sur différentes plateformes puissent communiquer. Comme il s'agit d'un protocole, il impose des règles intégrées qui augmentent la complexité et les coûts, ce qui peut ralentir le chargement des pages. Cependant, ces standards assurent la conformité et sont ainsi privilégiés pour certains scénarios d'entreprise.

Terminal : Variété de périphérique réseau placé à l'extrémité d'un nœud. Le terminal est un point d'accès de communication entre l'homme et un ordinateur central ou un réseau d'ordinateurs.

Méthode Merise : Méthode de conception des systèmes d'information créée dans les années 70 par une équipe de chercheurs, encore très utilisée aujourd'hui pour la conception des bases de données. Elle propose de considérer 4 niveaux : MCD, MOD, MLD et MPD.

Architecture MVC : L'architecture MVC (Model-View-Controller), est l'une des architectures logicielles les plus utilisées pour les applications Web. Elle permet de créer une application web pour bien gérer la structuration d'un projet en trois parties. Elle se compose de trois modules : modèle, vue, contrôleur.

Modèle : Noyau de l'application qui gère les données, permet de récupérer les informations dans la base de données, de les organiser pour qu'elles puissent ensuite être traitées par le contrôleur.

Vue : Composant graphique de l'interface qui permet de présenter les données du modèle à l'utilisateur.

Contrôleur : Composant responsable des prises de décision, gère la logique du code qui prend des décisions, il est l'intermédiaire entre le modèle et la vue.

Route : Ce sont des segments de ce que propose l'API. Dans le jargon du web, on appelle les segments également des routes d'API.

JWT : Les « JSON Web Token » ou JWT sont des jetons générés par un serveur lors de l'authentification d'un utilisateur sur une application Web, et qui sont ensuite transmis au client. Ils seront renvoyés avec chaque requête HTTP au serveur, ce qui lui permettra d'identifier l'utilisateur. Il est composé de 3 parties : header.payload(les « claims ».signature

Git : Logiciel de gestion de versions (Version Control System) qui suit l'évolution des fichiers sources et garde les anciennes versions de chacun d'eux sans rien écraser.

Tests unitaires : Les tests unitaires permettent de vérifier le bon fonctionnement d'une petite partie bien précise (unité ou module) d'une application. Ils s'assurent qu'une méthode exposée à la manipulation par un utilisateur fonctionne bien de la façon dont elle a été conçue.

Tests d'intégration : Test qui se déroule dans une phase d'un projet informatique suivant les tests unitaires. Il consiste, une fois que les développeurs ont chacun validé leurs développements ou leurs correctifs, à regrouper leurs modifications ensemble dans le cadre d'une livraison

Composants métier : Développé par l'entreprise, le composant métier correspond aux différentes tâches effectuées par un utilisateur : créditer ou débiter un compte, alimenter une base de données, etc.

JSX : JSX est une extension React de la syntaxe du langage JavaScript qui permet de structurer le rendu des composants à l'aide d'une syntaxe familière à de nombreux développeurs. Il est similaire en apparence au HTML.

Middleware : Un middleware est un logiciel qui fournit aux applications des fonctionnalités et des services communs que le système d'exploitation n'offre pas. La gestion des données, les services d'application, la messagerie, l'authentification et la gestion des API sont des services communément gérés par les solutions de middleware.

Un middleware permet d'améliorer l'efficacité des développeurs qui créent les applications. Il joue le rôle de lien entre les applications, les données et les utilisateurs.

URI : Uniform Resource Locator (URL) est un URI qui, hormis le fait qu'il identifie une ressource sur un réseau, apporte les moyens d'agir sur une ressource ou d'obtenir une représentation de la ressource en décrivant son mode d'accès primaire ou «emplacement» réseau.

Présentation

- Présentation personnelle

Je m'appelle Jérémy, j'ai 33 ans et mes études ont été commerciales essentiellement. Avec un Bachelor Marketing-Communication en 2015, j'ai travaillé dans la vente majoritairement en stations de ski ou en métropole, je pensais que le secteur tertiaire me conviendrait mais j'avais tort. Il y a 2 ans j'ai décidé de me réorienter dans le secteur du numérique que je ne me pensais pas accessible ; j'ai commencé par une formation de développeur web à l'Afpa Marseille dans laquelle je me suis épanoui et décidé de continuer dans cette voie en suivant la formation supérieure actuelle : concepteur/développeur d'applications. J'effectue cette année en alternance dans l'entreprise Biblibre qui travaille sous licence de logiciel libre.

- Présentation du projet

Le choix de projet personnel s'est porté sur une application d'entreprise destinée à la mise en place de bornes de recharge électriques pour véhicules dans la ville de Marseille. Le but étant que les employés désignés comme administrateur puissent se connecter sur le backoffice et y ajouter des futurs emplacements de bornes de recharge pour que, côté technicien, on puisse se rendre à ces emplacements et y installer les bornes. Les techniciens souvent en déplacement auront l'aide de cette application mobile pour trouver les emplacements et mettre à jour les informations par borne installée ainsi qu'une photo !

En passant par une connexion sécurisée, les administrateurs de l'entreprise peuvent se connecter sur le backoffice et y découvrir un dashboard ainsi que 3 autres menus : Emplacements, Bornes électriques et Carte.

'Emplacements' liste les emplacements déjà choisis dans la ville sur lesquels il y a ou il y aura une borne de recharge.

'Bornes électriques' sont les endroits dans la ville où sont déjà implantées les bornes et 'Carte' nous affiche la carte de la ville ainsi que les positions des emplacements ET des bornes. Il est possible via le menu de créer, modifier ou supprimer des bornes ET emplacements de borne.

L'application mobile qui sera téléchargeable via le backoffice sera utilisable par les techniciens pour retrouver les emplacements prédéfinis afin d'y installer des bornes de recharge électriques. Ils se connecteront grâce à leur prénom et leur nom sur l'application et pourront créer, modifier ou supprimer une borne. Ils ont également la possibilité de joindre une photo de la borne directement après installation.

Cahier des charges

Présentation projet

Le projet a pour but de simuler une entreprise qui déploie des bornes électriques dans la ville de Marseille. Les techniciens devront lorsqu'ils déploient les bornes, ajouter sur l'application mobile une nouvelle borne pour pouvoir la référencer sur le site internet. Ces données seront stockées dans une base de données. L'utilisateur du site et de l'application mobile aura la possibilité de voir grâce à des marqueurs les différentes bornes installées avec leurs emplacements. L'activité principale du site est donc le déploiement de bornes électriques. La date de création du projet est le 03/01/2022.

Analyse de l'existant

Les langages de programmation utilisés : HTML/CSS, Javascript (et NodeJS), SQLPug, NodeJS, React-Native, Mysql

Système de Gestion de base de données : MySQL/

Framework : expressJS, React-Native, TailwindCSS

Les versions des langages : Pug v3.0.2, NodeJS v16.15.0, React Native v0.64.1, MySQL v5.7.33, expressJS v4.17.3

Hébergement utilisé : AlwaysData

Type de site : Pour entreprise en interne

Les personnes engagées sur ce projet : BREUILLARD Jérémy et SHLIMON David

Les objectifs de l'application et les cibles

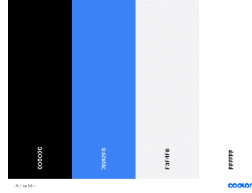
Les objectifs de l'application site web est de référencer les emplacements de bornes électriques ainsi que les bornes électriques installées, l'application mobile sera pour l'équipe technique qui sera sur place et devra ajouter de nouvelles bornes électriques lorsqu'ils installent la borne sur place. Pour au final avoir un rendu sur une liste, et une carte les bornes électriques déployées dans la ville de Marseille Cette application aura pour but de référencer les bornes électriques installées et donc l'entreprise aura un visuel sur les bornes installées

Périmètre du projet

Notre site ne sera pas multilingue pour l'instant étant donné qu'il sera utilisé par une entreprise qui utilise seulement le français. Le site a donc une version site web et une version application mobile chacun pour un métier précis.

Charte graphique

➤ Colorimétrie



➤ Logo



➤ Typographie

DejaVu Serif

Contraintes techniques

L'application mobile sera accessible sur les systèmes d'exploitation Android dans un premier temps puis sur iOS par la suite.

L'application web, elle, sera accessible sur tous les navigateurs.

Il existe **deux parties** pour ce projet, une partie back-office (interface web) et une partie application mobile :

- La **partie application** mobile sera utilisée par un seul type d'utilisateur : l'équipe technique qui va sur place installer les bornes de recharge. Cette partie permet donc le déploiement des bornes. Il aura la liste des bornes déployées et il pourra créer, modifier ou supprimer des bornes.
- La **partie site web** est utilisé uniquement par les employés qui ont les droits d'administrateur. Sur cette espace l'utilisateur pourra créer, modifier, supprimer des emplacements de bornes de recharge, et pourra aussi faire cela aux bornes de recharge. Il trouvera aussi une liste complète des emplacements crée et des bornes de recharges crée.

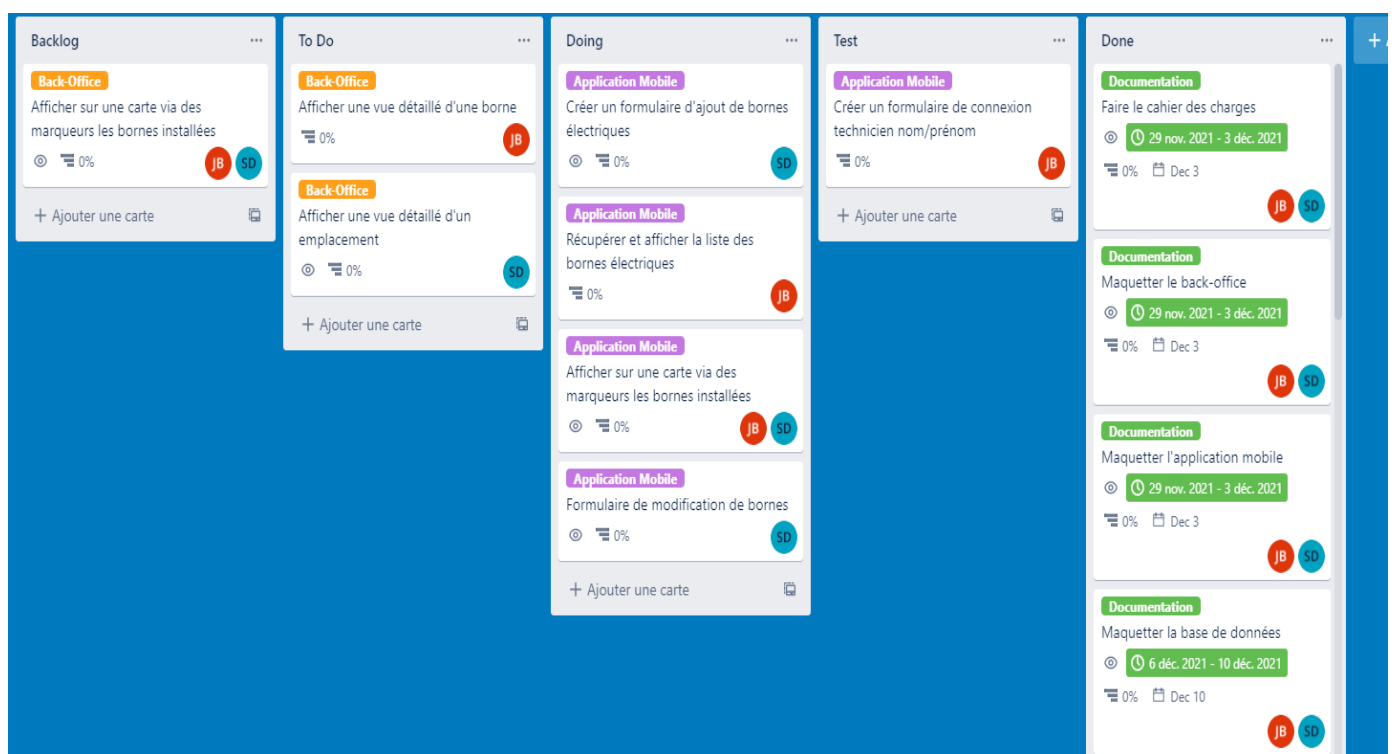
Organisation du projet

Trello

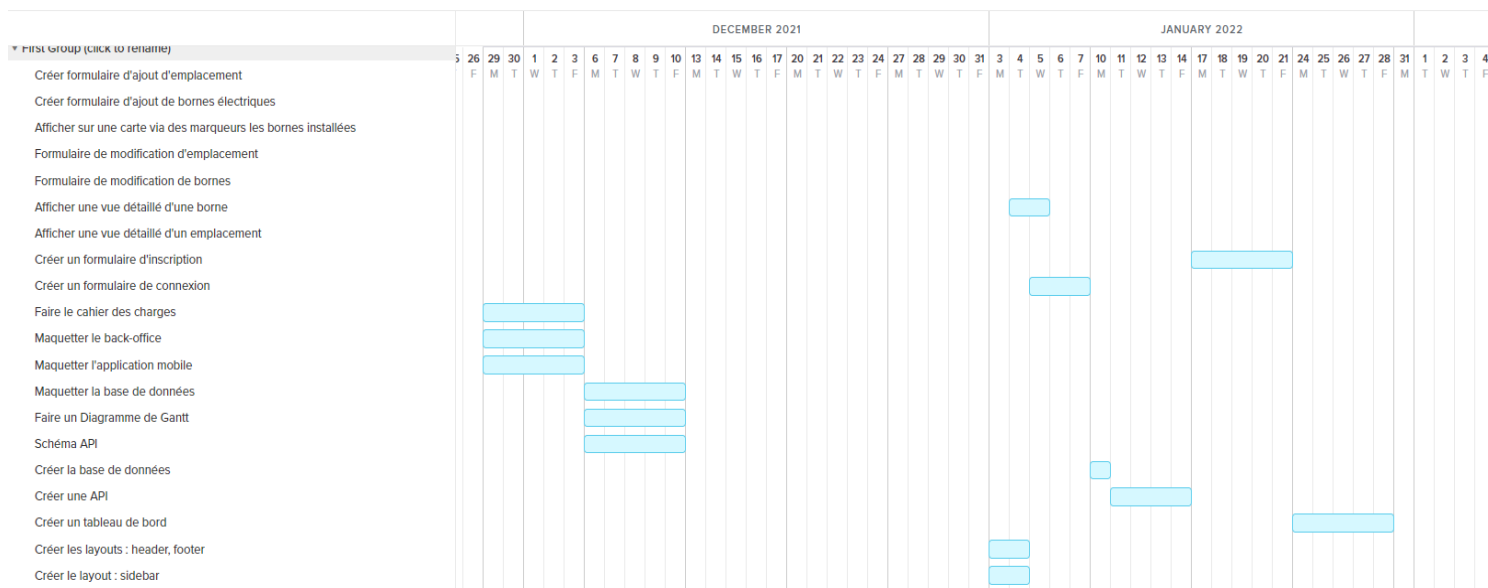
Ce projet a été réalisé en groupe durant ma période d'école. Il a fallu être bien organisé et bien gérer notre temps de travail, prioriser les bonnes pratiques...

Pour commencer nous avons utilisé l'agilité grâce au Trello. Cet outil nous a permis d'attribuer des tâches à chacun et d'avoir une vue d'ensemble des tâches à effectuer. Il permet aussi de définir une date de début de tâche et date de fin, de gérer les priorités des différentes tâches. Cela nous a aidé à suivre l'avancement de notre projet en temps réel.

Chaque semaine on prenait du temps pour visualiser le Trello, afin de suivre les tâches qui nous sont attribuées et de changer leur statut en fonction du travail fourni ; on peut le voir ci-dessous :



Dans Trello se trouve un plugin permettant de générer un diagramme de Gantt issu d'un espace de travail. Ci-dessous la liste des tâches avec les dates de début des différentes tâches et les deadlines que nous avons fixé :



Github

Nous avons décidé d'utiliser Github, outil de versionning phare pour partager notre travail et y accéder à tout moment. Comme nous travaillons chacun de notre côté la majeure partie du temps il était essentiel pour nous d'utiliser cette plateforme.

Une fois le dépôt distant créé en visibilité privée, nous avons accès mon collaborateur et moi-même uniquement à celui-ci.

A chaque modification de fichier ou ajout de code, une branche était créée à partir d'une version de base de notre application.

Ayant travaillé sur GitLab via Linux dans le cadre de mon contrat de professionnalisation, j'ai appris à utiliser l'invite de commande pour sauvegarder mon travail, voici quelques commandes :

git init : initialise git dans un dossier déjà existant

git add <fichier> : ajoute un fichier à l'index git

git commit : créé un commit, c'est à dire un snap avec commentaire des fichiers ajouter à l'index, en local uniquement

git push <nom du dépôt> : envoi les fichiers commité sur un dépôt distant

git log : affiche un journal des derniers commits effectués

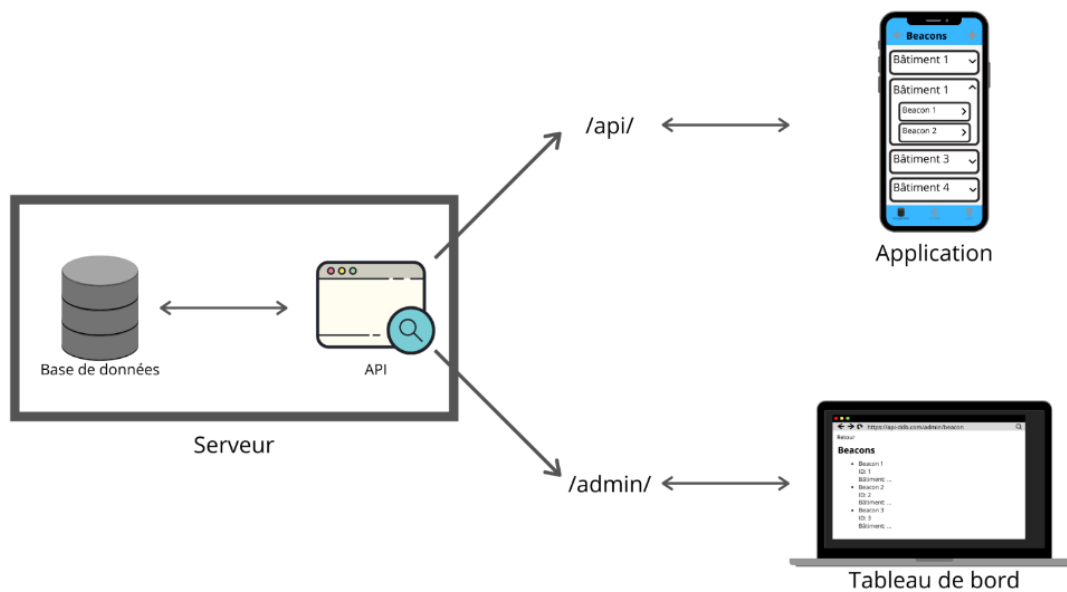
git branch : liste les branches locales existantes

git checkout <nom_de_la_branche> : change de branche

git rebase <nom_de_la_branche> : permet de remettre une branche au même niveau que celle indiquée

git reset --soft | --hard : permet d'annuler les derniers changements et de se faire un backup du dernier commit

Architecture logicielle



L'application mobile sera réservée uniquement aux techniciens, l'administrateur informatique aura accès à l'application mobile et au site web.

L'application web et l'application mobile utilisent la même API, et l'API communique avec la base de données « chargemap ».

Ressources utilisées



Team Viewer pour faire du peer-programming



Trello qui accompagne la méthode Agile, pour planifier nos travaux à court terme.



Figma pour créer nos maquettes et Wireframe



Whimsical pour modéliser la base de données



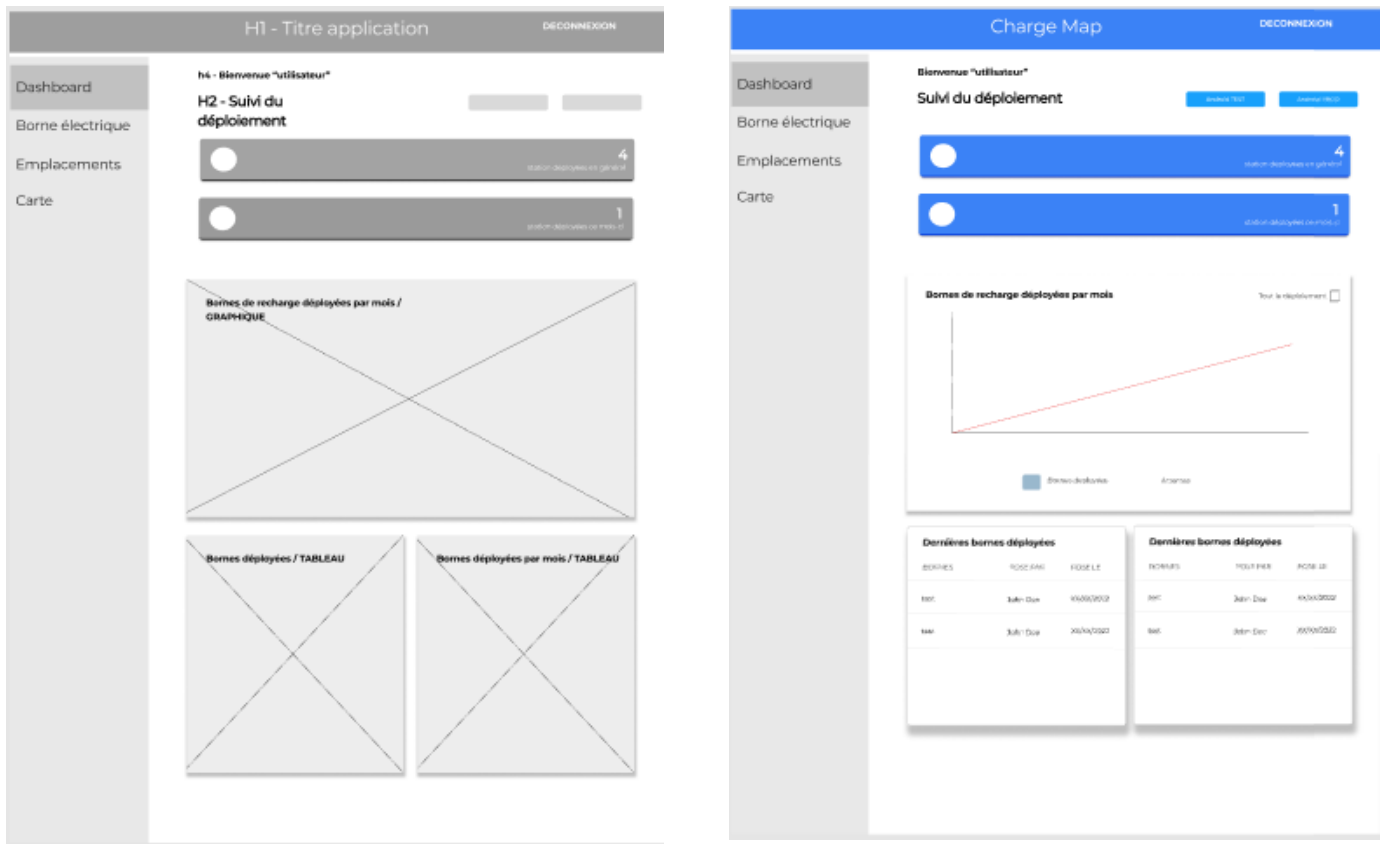
Google Drive pour stocker des fichiers



Alwaysdata qui est un hébergeur gratuit < 100Mo

Conception du front-end

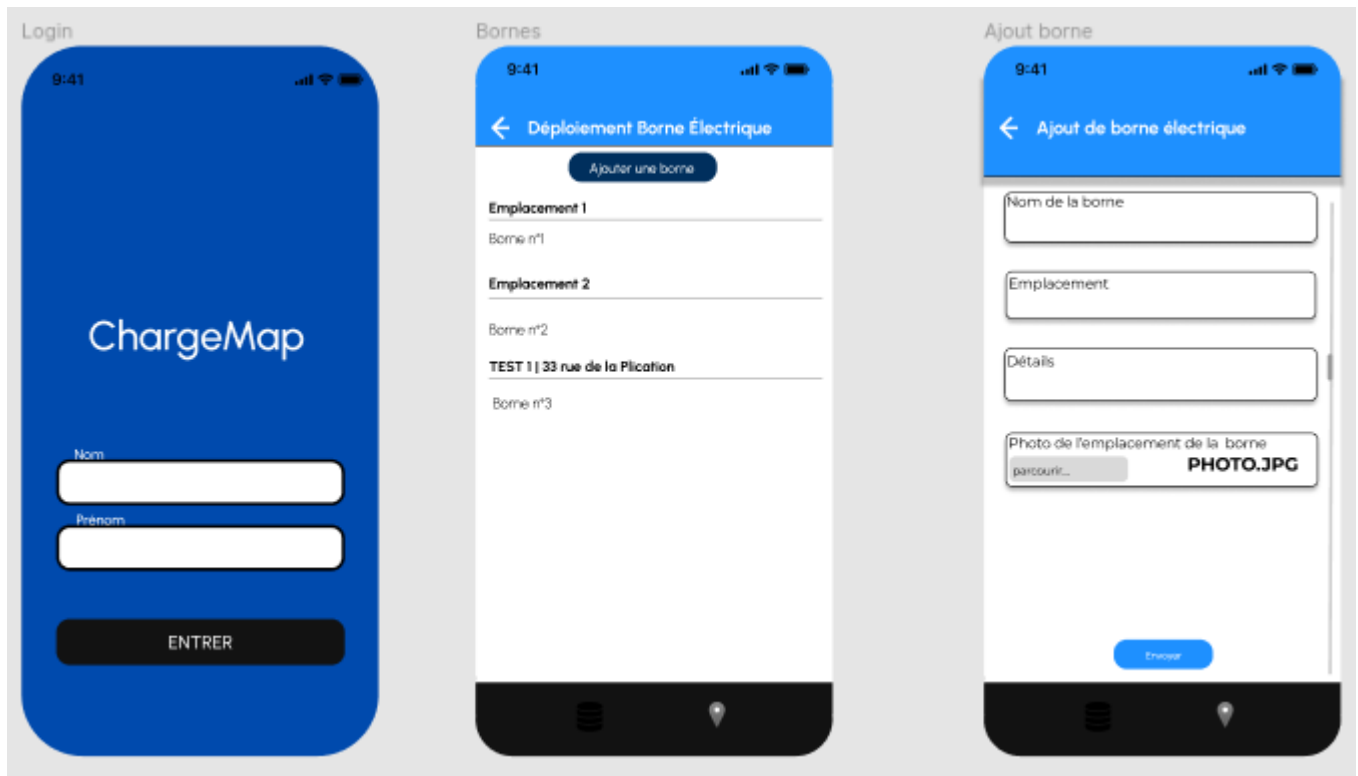
- Wireframe et maquettage



Ci-dessus le wireframe à gauche et la maquette à droite du site web accessible par les administrateurs.

Les maquettes ont été réalisées avec l'outil en ligne **Figma**

Ci-dessous la maquette de l'application mobile utilisable par les techniciens et les administrateurs



- Informations des contenus

Côté back-office :

Login – Espace de connexion réservé aux administrateurs du site

Dashboard – Page d'accueil du back-office qui est une synthèse des données en base

Emplacements – Page permettant l'ajout, la modification et la suppression d'un emplacement de borne électrique

Bornes électriques - Page permettant l'ajout, la modification et la suppression d'une borne électrique de recharge

Carte – Page affichant Une carte ainsi que les emplacements de borne et les bornes installées sous forme de marqueurs

Côté mobile :

Login – Espace de connexion pour les techniciens de l'entreprise qui doivent installer des futures bornes électriques

Emplacements – Page listant les emplacements disponibles

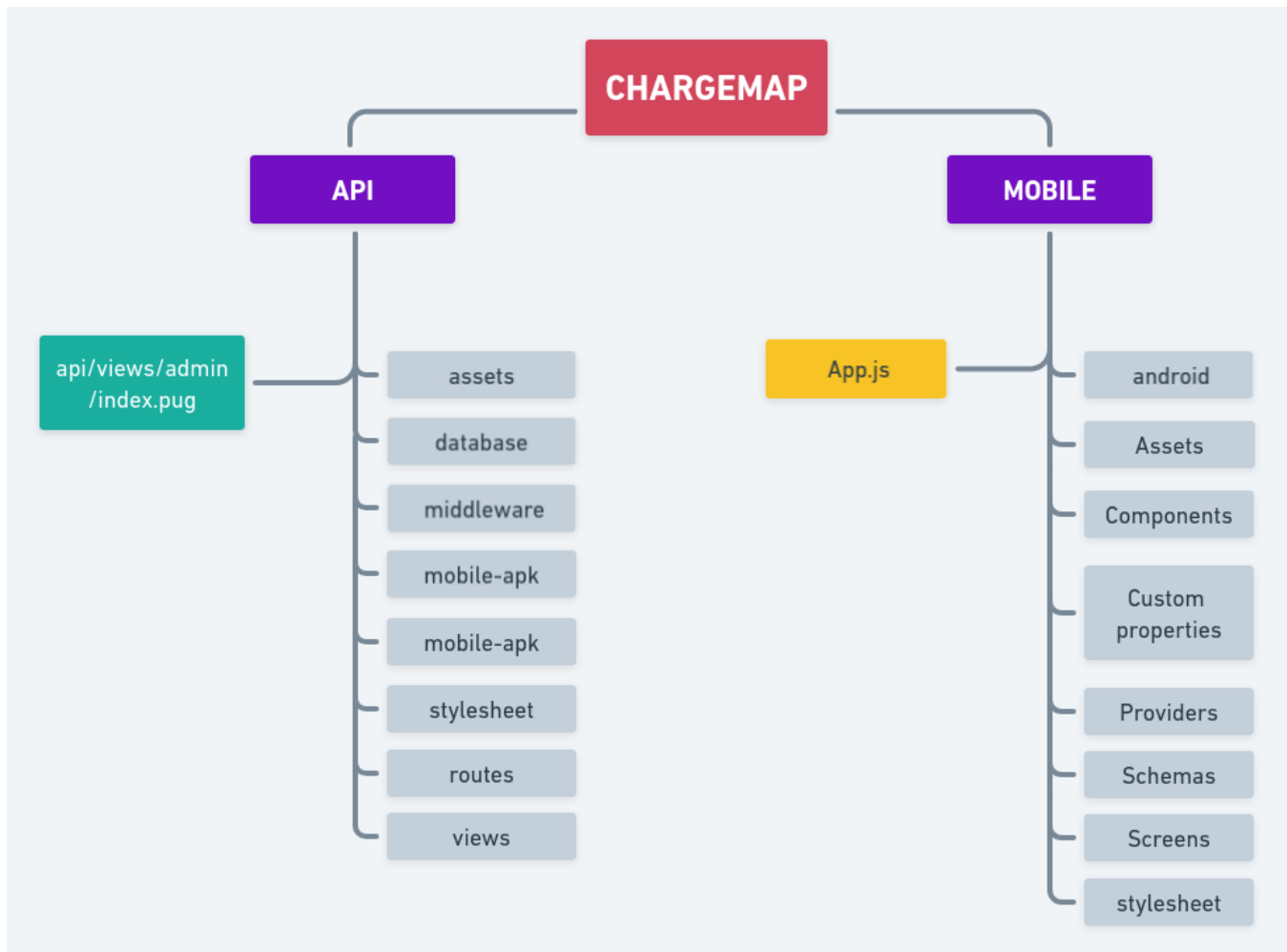
Ajout de borne électrique – Permet d'ajouter une borne sur un emplacement défini et d'ajouter une photo du travail fini

- Environnement de travail

- **Visual Studio Code** : un IDE performant et ergonomique, de plus il possède de nombreuses extensions qui permettent d'avoir du confort suivant les langages qu'on utilise
- **Laragon** : environnement de développement web rapide et pratique à utiliser sur Windows
- **Android Studio** : IDE spécifique pour développer les applis android, il donne des informations supplémentaires notamment pour le développement mobile que l'on ne trouve pas sur d'autres IDE comme VSCode et il permet de télécharger facilement les outils de debug
- **phpMyAdmin** : système de gestion de base de données avec interface graphique
- **Terminal/Powershell** : console qui permet un accès rapide à mon ordinateur
- **Swagger** : langage de description d'interface qui m'a permis de documenter l'API
- **NPM** : gestionnaire de paquets pour l'environnement JavaScript et NodeJS

- **React-Native** : framework JavaScript pour créer des applications mobiles
- **Alwaysdata** : hébergeur en ligne
- **TailwindCSS** : framework CSS avec des classes

- Arborescence du projet



Le projet est présenté sous 2 parties, API et MOBILE.

API représente le back-office ainsi que l'API et les routes.

MOBILE représente l'application mobile.

Développement du front-end

- Partie mobile

Composants

Avec le framework React-Native on peut créer des composants (components) : ce sont comme des fonctions Javascript réutilisables à n'importe quel moment à condition d'importer le fichier sur lequel la fonction est contenue dans la page sur laquelle on a besoin du composant.

```
import { TextInput, Button } from 'react-native-paper';
```

 voici l'import

```
<TextInput
  label='Prénom'
  mode='outlined'
  error={firstnameError}
  onChangeText={text => {
    setFirstnameError(false);
    checkAndSetFirstname(text);
  }}
  style={styles.textInput}
/>
<Button
  mode='contained'
  onPress={handleEnter}
  style={styles.centeredButton}
  color='black'
>
  Entrer
</Button>
```

Grâce auquel on peut appeler les composants **TextInput** et **Button**.

On leur passe en **props** les informations que l'on souhaite voir affichées

Nous avons aussi la classe/le composant `Stack.Screen` qui est appelé sur la page, ce dernier possède 2 attributs dont le **name** et le **component**. On passe au composant ce qu'on appelle une **props** (propriété) qui va renvoyer des éléments React décrivant ce qui doit apparaître à l'écran, ces éléments sont des attributs JSX sous forme d'objet.

C'est une information qui vient de « l'extérieur » du composant, généralement, de son parent direct (mais pas toujours).

```
<Stack.Screen
  name='Stations Screen'
  component={StationScreen}
/>
<Stack.Screen
  name='Edit Station Screen'
  component={EditStationScreen}
/>
```

Navigation mobile / Navigation par stack

La navigation mobile repose entièrement sur un modèle d'empilement d'écrans, on appelle ça Stack-Based Navigation.

```
<Stack.Navigator
  screenOptions={{
    headerShown: false
  }}
>
  <Stack.Screen
    name='Welcome Screen'
    component={WelcomeScreen}
  />
  <Stack.Screen
    name='Borne Screen'
    component={BorneNavigation}
  />
  <Stack.Screen
    name='Add Borne Screen'
    component={AddBorneScreen}
  />
```

Le principe est d'imaginer une **pile d'écrans**, le premier étant l'écran de login.

Pour naviguer d'un écran à l'autre il faut "empiler" l'écran de destination en haut de la pile, c'est-à-dire que lorsqu'on clique sur le bouton "entrer" on laisse l'écran de login derrière nous et un 2nd écran fait son apparition... l'écran d'accueil ; et ainsi de suite.

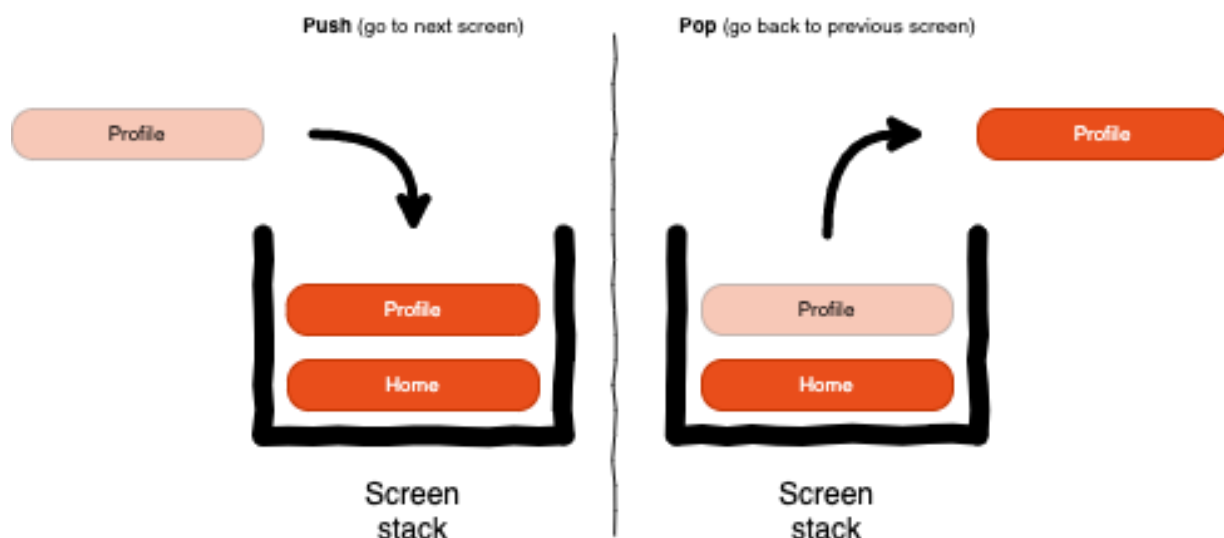
Si on revient à l'écran précédent le dernier de la pile est retiré pour laisser place à celui d'avant.

À noter que l'écran tout en haut de la pile sera toujours l'écran visible par l'utilisateur.

On l'a dit, en mobile, tout repose sur un modèle d'empilement d'écrans.

Deux nouveaux termes à connaître ici. Le premier, c'est **Stack**, qui désigne la totalité des écrans actuellement empilés. Le second, c'est **Navigator**, qui désigne le système gérant la **Stack** (ajout ou suppression d'écran par exemple).

Commençons par illustrer ce fonctionnement, analysons le schéma suivant :



On retrouve les deux actions essentielles à une navigation, à savoir, pouvoir naviguer vers un nouvel écran, mais aussi revenir sur l'écran précédent.

Ces écrans sont accessibles grâce au composant `Stack.Screen`

- Partie web

Grâce au moteur de template Pug nous avons pu générer rapidement des composants qui ont fini par créer des pages.

La syntaxe est assez particulière aux premiers abords mais se prend facilement en main : seul bémol c'est qu'il y a des classes qui agissent comme du CSS ce qui donne un visuel un peu chargé mais pas moins efficace. On pense notamment au framework Bootstrap qui agit un peu de la même manière.

```
.mr-8.mb-5
  .flex.flex-row.justify-between
    h1.text-lg.font-bold(class='lg:text-3xl') Suivi du déploiement
    .flex
      a.bg-blue-400.text-white.font-semibold.py-2.px-4.rounded-md.mr-5(href='/admin/download-apk-test' class='hover:bg-blue-500') Télécharger App Android-test
```

« Le parent est une div avec un *margin-right* de 8 et un *margin-bottom* de 5

- la div enfant est de display flex

- - le h1 enfant est en gras »

Comme en python, l'indentation est cruciale si on veut le rendu souhaité. On note que le **h1** et la **div** de classe « `.flex` » sont tous deux enfants du même parent **`.flex.flex-row.justify-between`**

Conception du back-end

- Mise en place de la base de données

Pour ce projet, il est nécessaire d'avoir une base de données afin de stocker les données concernant les bornes de recharges, leurs emplacements ainsi que les utilisateurs. N'ayant aucune contrainte concernant le choix de bases de données, nous avons fait le choix de travailler sur les bases de données MySQL car c'est celle que nous avons l'habitude d'utiliser, que ce soit interface graphique ou via un terminal

- Concevoir la base de données

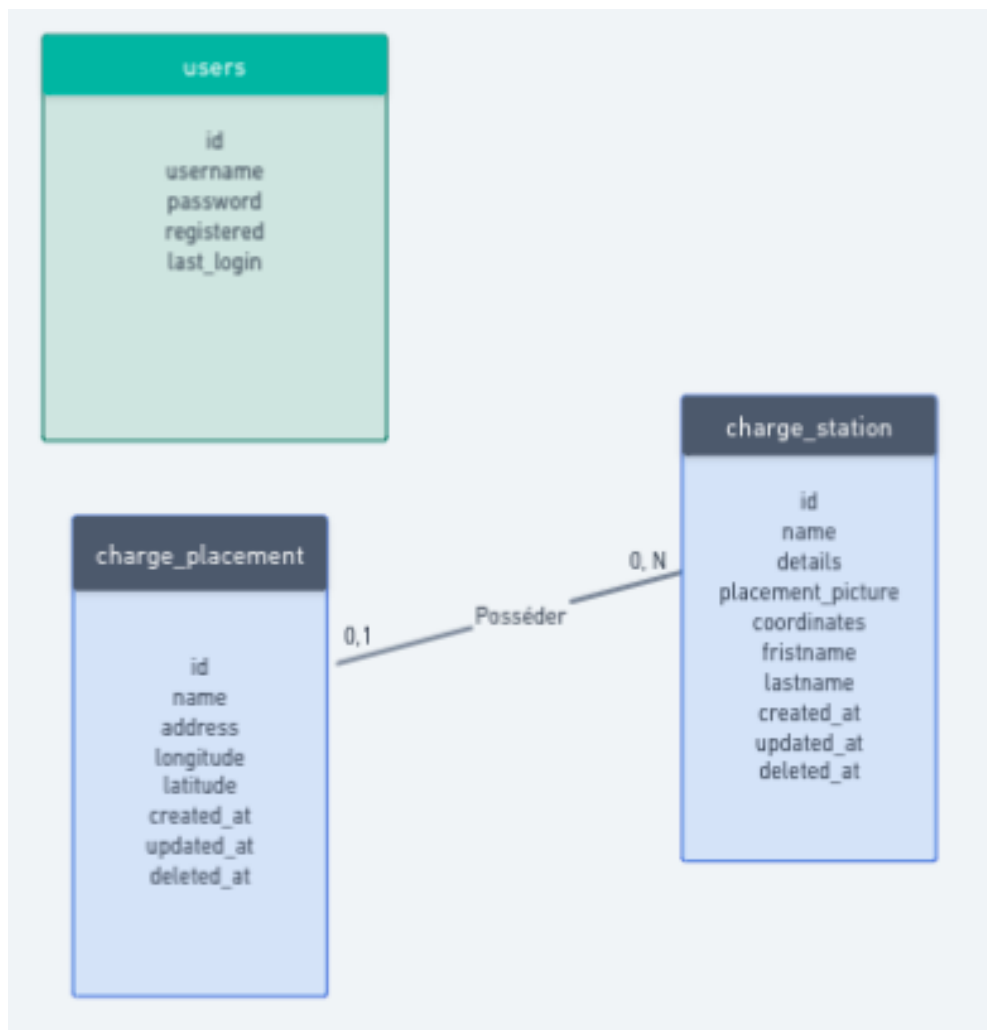
Pour concevoir notre base de données nous avons utilisé la **méthode Merise**.

Nous avons d'abord fait une liste des données importantes à stocker dans la base de données.

- Un *utilisateur*, avec un *nom d'utilisateur*, un *mot de passe sécurisé*, une *date d'inscription* et *date de dernière connexion* afin de retracer l'utilisateur
- Des bornes de recharges, avec leur noms, l'identifiant de l'emplacement relié à l'emplacement, un espace de détails afin que le technicien puisse ajouter des détails lors de l'installation, une photo de l'emplacement avec la borne installée, et les coordonnées GPS. Avoir la possibilité de retracer l'utilisateur qui a installé la borne avec leur nom, prénom, la date d'installation, la date de modification et la date de suppression
- Les emplacements des bornes, avec un nom d'emplacement, une adresse, la longitude et latitude pour les coordonnées GPS, et la date de création, modification et suppression pour le traçage.

Modèle conceptuel de données (MCD)

Créer un modèle conceptuel de données, c'est-à-dire un schéma de base de données qui peut être compris et lu par tous. Avec l'outil en ligne **Whimsical** nous voyons les différentes tables avec leur nom, leurs entités et les cardinalités entre les tables.



On remarque le lien entre la table *charge_placement* et *charge_station*. Dans le **MCD** On ne sait pas encore quelle entité représente *charge_placement* dans la table *charge_station*.

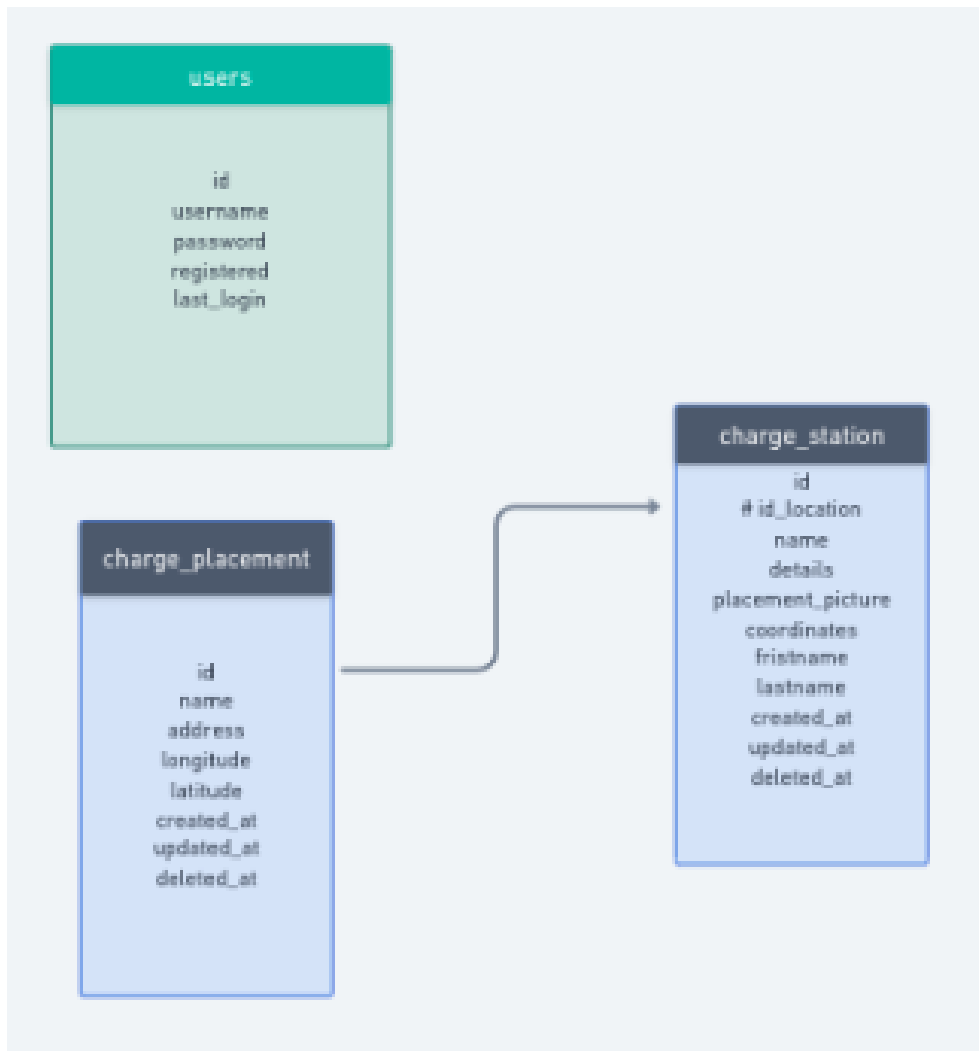
Les **cardinalités** indiquent qu'un emplacement possède au minimum 0 bornes de recharge ou au maximum N bornes de recharges, et une borne possède au minimum 0 emplacement ou au maximum 1 emplacement

Modèle logique de données (MLD)

Dans la 2ème étape qui s'appelle le Modèle Logique de Données.

Les verbes de liaison ont disparu et les entités équivalentes d'une table à l'autre sont reliées entre elles.

On voit apparaître les clés étrangères et les clés primaires.

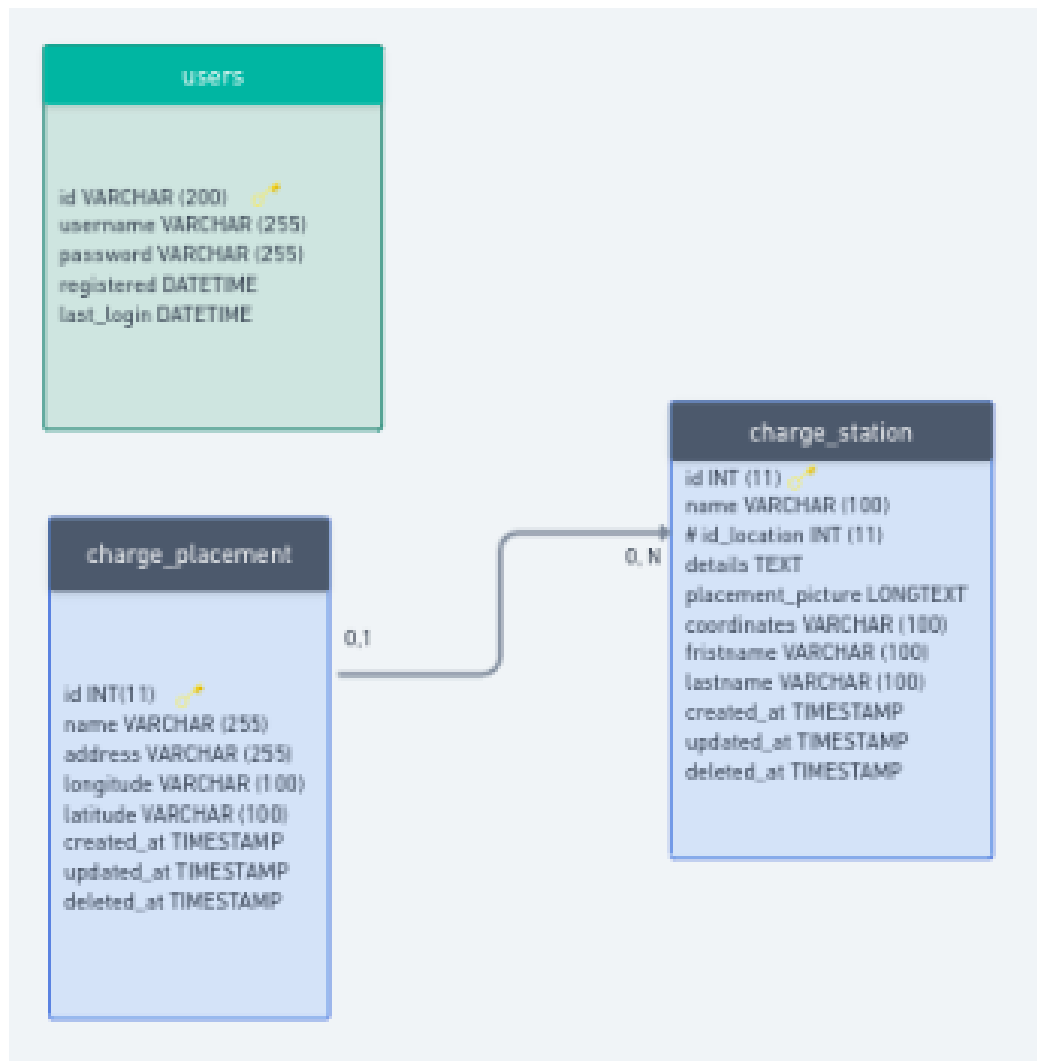


Le verbe a disparu, la **clé primaire** de *charge_placement* fait apparaître son équivalent dans la table *charge_station*.

Modèle physique des données (MPD)

Pour cette dernière étape, on analyse les relations entre les tables. N'ayant pas de cardinalités de type N/N (Many To Many), je n'ai donc pas besoin de table de liaison. Puis nous avons créé la clé étrangère, cette clé garantit l'intégrité référentielle entre deux tables, par exemple nous ne pouvons pas supprimer un emplacement (table parent) sans avoir supprimé ses bornes de recharges reliées (table enfant). Par la suite on donne un type à chaque entité.

Ce MPD servira de support de création de la base de données au développeur car il constitue une maquette utilisable.



Développement du back-end

Le développement du back-end a été fait à la fois pour la partie application web et application mobile. Nous avons créé une **API** qui communique avec les deux applications.

- Arborescence

Nous avons utilisé le motif d'architecture logicielle **MVC** : model / view / controller, ce motif est composé de trois types de modules ayant trois responsabilités différentes : les **modèles**, les **vues** et les **contrôleurs**.

Notre dossier route contient des fichiers liés au contrôleurs, le dossier database contient des fichiers de modèles et le dossier view pour les différentes vues/affichage.

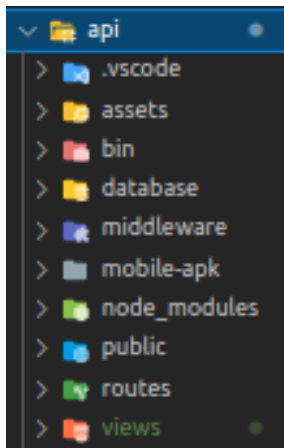


Illustration de l'arborescence du back-end de l'application

- API

Fonctionnement API

Lorsque le client envoie une requête sur l'API, le routeur analyse l'URL. En fonction de la route et de la méthode, un controller est appelé seulement si celui-ci à bien renseigné le *username* et le *password* dans l'URL, l'accès au backoffice est sécurisé avec un token JWT, le mot de passe est haché et l'utilisateur bénéficie d'un uID. Côté mobile cela n'est pas sécurisé mais étant donné que cette API a pour but une utilisation en interne et ne contient pas de données importantes, nous pouvons nous le permettre.

Architecture API

Les différentes méthodes **http** utilisées dans ce projet sont :

- GET pour récupérer les données via l'url
- POST pour enregistrer les données que l'on soumet dans un formulaire
- PUT Pour mettre à jour l'intégralité des informations d'une donnée
- PATCH Pour mettre à jour seulement une donnée
- DELETE pour supprimer totalement une donnée

Documentation API

Il est important de documenter son API de manière à ce que notre code puisse être lu par un développeur sans difficultés. Swagger est un ensemble d'outils permettant d'aider à la conception et la documentation. Il est un support très utile quand on parle d'API.

Ci -dessous l'interface graphique permettant d'interagir avec l'API :

charge_station		^
GET	/charge_station	Permet la récupération de toutes les bornes de recharge
PUT	/charge_station	Permet la modification intégrale d'une borne de recharge
PATCH	/charge_station	Permet la modification intégrale ou partielle d'une borne de recharge
POST	/charge_station/create	Permet la création d'une borne de recharge
DELETE	/charge_station/{id}	Permet la suppression d'une borne de recharge

Routage

Routage fait référence à la définition de points finaux d'application (URI) et à la façon dont ils répondent aux demandes client. Pour le routage nous avons utilisé le routeur de *express.js*. Car il possède un middleware qui gère facilement le routage. C'est une instantiation de la classe *express.Router*

Voici comment se définit une route :

Route = router.**méthode** (**chemin**, **fonction** (req,res))

Router est une instance de **express** : on appelle une **methode** http puis en paramètre on lui passe le **chemin / endpoint** vers notre API ainsi qu'un appel à une **fonction callback** qui requiert une (request) et un (result)

```
/* GET index Api - view */
router.get('/', userMiddleware.isLoggedIn, (req, res)
```

Comme on peut le voir ci-dessus, nous utilisons la méthode GET.

Le point de terminaison étant déjà dans le /admin, il n'est pas nécessaire de préciser avec le /index.

La fonction **isLoggedIn** issue d'un middleware de connexion sécurisé permet à la route de ne pas laisser des utilisateurs non connectés se rendre sur cette page.

Liste des routes :

```
var indexRouter = require('./routes/index');
var apiRouter = require('./routes/api');
var adminRouter = require('./routes/admin');
var loginRouter = require('./routes/login');
var registerRouter = require('./routes/register');
var logoutRouter = require('./routes/logout');
var cors = require('cors');
var app = express();
```

Dans le fichier **App.js** on peut voir que les routes sont déclarées indépendamment afin de séparer chaque route par fichier

```
app.use('/', indexRouter);
app.use('/api', apiRouter);
app.use('/admin', adminRouter);
app.use('/login', loginRouter);
app.use('/logout', logoutRouter);
app.use('/register', registerRouter);
```

L'URL /admin est défini, alors on peut se rendre dans le routeur **adminRouter**.

« app » est une instance de express et la fonction use permet d'associer un routeur à un point de terminaison

- Architecture

Modele

Dans les modèles on peut trouver la connexion à la base de données, et toutes les requêtes SQL. Nos requêtes sont listées dans le fichier **requests.js**

Voici sur l'exemple ci-dessous la connexion à la base de données

```
const mysql = require('mysql');
const config = require('../config.json');

const db = mysql.createPool({
  host: config.db.host,
  user: config.db.user,
  password: config.db.password,
  database: config.db.database_name
});
```

Connexion avec **mysql.createPool** à qui on passe des paramètres pour établir la connexion

Comme on peut le voir ci-contre j'ai renseigné les différentes coordonnées de connexion à la base de données dans le **config.json**.

```
{
  "db": {
    "host": "localhost",
    "user": "root",
    "password": "root",
    "database_name": "charge_map",
    "tables": {
      "charge_station": {
        "name": "charge_station",

```

Avec la commande **npm install mysql** j'ai pu installer la dépendance mysql qui permet de lier mon serveur NodeJS à MySQL

Composant dans le langage d'une base de données

La méthode query prend en paramètre du SQL et un tableau facultatif de paramètres qui seront transmis à la requête.

L'objet **Promise** (pour « promesse ») est utilisé pour réaliser des traitements de façon asynchrone. Une promesse représente une valeur qui peut être disponible maintenant, dans le futur voire jamais.

Cette fonction renvoie un objet Promise, cet objet permet d'effectuer une vérification. La promesse est résolue lorsque la requête est terminée. S'il n'y a pas d'erreur alors la promesse est résolue, s'il y a une erreur alors elle sera rejetée.

```

/** GET all */
const getAll = (table, order_by = 'created_at') => {
  console.log('GET', table, 'CALL');

  return new Promise((resolve, reject) => {
    db.query(
      `SELECT * FROM ${table} WHERE deleted_at IS NULL ORDER BY ${order_by} DESC `,
      (error, result) => {
        if (error) {
          console.log('ERROR: ', error);
          reject({ type: 'ko', code: 500, message: error });
        }
        resolve({ type: 'ok', code: 200, data: result });
      }
    );
  });
};

```

Comme on peut le voir ci-dessus, nous avons créé une fonction `getAll()`, permettant de l'utiliser quand on veut pour récupérer toutes les informations d'une table choisie grâce à la requête

*SELECT * FROM \${table}* - le nom de la table en paramètres,

WHERE delete_at IS NULL - pour récupérer seulement les informations des lignes qui ont le champ *deleted_at* égal à 0

ORDER BY \${order_by} DESC - en affichant les dates de création dans l'ordre décroissant

Controller

Dans notre fichier **api.js** se trouvent toutes les fonctions nécessaires au projet, ce que l'on appelle un **Controller** : il cherche dans le **Modèle** les requêtes que l'on peut formuler et se sert de ces requêtes pour récupérer des informations

```

/** GET 'charge_station' by ID */
router.get('/charge_station/:id', (req, res) => {
  if (
    req.query.username !== config.api.username &&
    req.query.password !== config.api.password
  ) {
    return res.status(401).json({
      type: 'ko',
      code: 401,
      error: 'user not authenticated'
    });
  }
  requests
    .getById(db_tables.charge_station.name, req.params.id)
    .then(result => {
      return res.status(result.code).json(result);
    })
    .catch(error => {
      return res.status(error.code).json(error);
    });
});

```

Ici la route `'/charge_station/:id'` a besoin de récupérer une donnée de la table `charge_station` grâce à un ID qu'on lui passe en paramètre...

On note ici que la fonction **getById** est déjà défini dans le fichier **request.js**

getById peut être utilisée à n'importe quel moment tant qu'elle est déclarée puis exportée grâce au **module.exports**

```
module.exports = {  
  getAll: getAll,  
  postNew: postNew,  
  getById: getById,  
};
```

```
/** GET info by ID */  
const getById = (table, id) => {  
  console.log('GET', table, 'BY ID CALL');  
  
  return new Promise((resolve, reject) => {  
    var idError = checkID(id);  
  
    if (idError) return idError;  
  
    db.query(`SELECT * FROM ${table} WHERE id=` + id, (error, result) => {  
      if (error) {  
        console.log('ERROR: ', error);  
        reject({ type: 'ko', code: 500, message: error });  
      }  
      if (!result[0])  
        reject({  
          type: 'ko',  
          code: 400,  
          message: `${table.slice(0, -1)} with id ${id} does not exist`  
        });  
      resolve({ type: 'ok', code: 200, data: result[0] });  
    });  
  });  
};
```

- Sécurité

Nous avons décidé de sécuriser un maximum notre back-office de manière à éviter les attaques, par exemple Les injections SQL ou les Attaque DDOS, un détail de ces attaques est disponible à la fin de ce projet.

Pour la sécurité, plusieurs actions ont été mises en place.

1. Chiffre/Cryptage des données liées au compte de l'administrateur :

Le **mot de passe** est haché, et l'**ID** est un **uid** qui veut dire identifiant unique se présentant sous forme de hashage également.

id	username	password
ba9aa6b4-907d-4d13-9e00-badf2136969b	administrateur	\$2a\$10\$tgIZD/dBgvsK3EOiu0j9a.Op9pbdlCft6qX2GGHR5k8...

2. JWT

Pour une authentification sécurisée, on utilise le Jeton Web Token. Lorsque l'utilisateur se connecte à l'application web, il obtient alors un token JWT qui lui permet de naviguer librement.

Qu'est-ce que le JWT ?

C'est un jeton qui permet l'échange des informations sur l'utilisateur de manière sécurisée. Et composé de 3 parties différentes



Pour sécuriser chaque route nous avons dû créer un système de droits qui ne permet pas à tout le monde de naviguer sur le site, seulement aux administrateurs.

```
const jwt = require('jsonwebtoken');

module.exports = {

  isLoggedIn: (req, res, next) => {
    const token = req.cookies.token;
    try {
      const decoded = jwt.verify(
        token,
        'SECRETKEY'
      );
      req.userData = decoded;
      next();
    } catch (err) {
      res.clearCookie("token");
      return res.redirect("/login");
    }
  }
};
```

On crée une constante qui va contenir notre JWT puis on exporte le module qui va contenir la fonction « isLoggedIn » chargée de vérifier l'utilisateur pour lui permettre l'accès. On crée une constante `token` dans laquelle il y a la requête `cookie token`, puis on fait un `try and catch` pour vérifier le token.. si le token est valide alors on continue, sinon il y a une erreur et l'utilisateur sera redirigé vers la page de login.

Condition qui permet de vérifier l'utilisateur et son mot de passe puis redirection vers la page « admin ». Si statut 400 : une erreur est renvoyée...

```
bcrypt.compare(req.body.password, result[0]['password'], (bErr, bResult) => {
  if (bErr) {
    throw bErr;
    return res.status(400).send({
      message: "Nom d'utilisateur ou mot de passe incorrect !",
    });
  }

  if (bResult) { //password match
    const token = jwt.sign(
      {
        username: result[0].username,
        userid: result[0].id,
      },
      'SECRETKEY',
      {expiresIn: 12000}
    );
    res.cookie('token', token, {
      httpOnly: true,
      maxAge: 900000000000,
    });
    db.query(
      `UPDATE users SET last_login = now() WHERE id = '${result[0].id}';`
    );
    return res.redirect('admin');
  }
}
```

Ci-dessus le code du fichier *login.js*. C'est le code qui permet de se **connecter**.

Pour le hachage on a utilisé la fonction *bcrypt* basée sur un algorithme de chiffrement, et pour vérifier ces mots de passe il faut utiliser la fonction *bcrypt.compare* : la fonction compare le mot de passe rentré dans l'input avec celui enregistré en base ; si ce dernier existe on lui assigne un token JWT d'une durée de 3 heures (12000sc). De plus, on update le champ « last_login » qui contiendra la date et l'heure à laquelle la personne s'est connectée pour la dernière fois.

Comme on peut le voir, j'utilise *bcrypt* pour hasher mes mots de passes, et donc pour les vérifier, il faut utiliser *bResult* : en comparant le mot de passe entré dans l'input et le mot de passe existant dans la base de données.

Il y aura aussi une requête pour update le champ « last_login » qui contiendra la date et l'heure à laquelle la personne s'est connectée pour la dernière fois.

Problématique rencontrée

J'ai rencontré un problème avec **mysql.createCollection** qui n'acceptait pas ma connexion car seule une connexion est permise avec cette fonction, j'ai donc utilisé **createPool** en faisant des recherches sur le site SITE ANGLAIS et cette fonction permet plusieurs connexions.

J'ai d'abord créé une connexion avec **mysql.createConnection** mais j'ai été déconnecté car c'est une connexion unique. Quand on exécute une requête on ne peut pas en exécuter d'autres en parallèle. Après quelques recherches sur internet je suis tombé sur le site *web-technology-experts-notes* en anglais qui conseillait d'utiliser **mysql.createPool** qui est plus adapté à notre utilisation : lorsqu'une connexion est occupée avec une requête, d'autres peuvent désormais s'exécuter.

Recherche sur site anglais

web-technology-experts-notes.in/2020/04/difference-between-mysql-createconnection-and-mysql-createpool-in-nodejs.html

Login Gmail

- BootStrap JS Videos
- MongoDB Videos
- MySQL Videos
- Node JS Videos
- PHP Videos

Interactive Diagram Library

GoJS Open >

Google Search

Search

Go Interactive

mysql.createConnection

When you create a connection with **mysql.createConnection**, you only have one connection and it lasts until you close it OR connection closed by MySQL.

A single connection is blocking. While executing one query, it cannot execute others. hence, your application will not perform good.

Example:

```
var mysql = require('mysql');
var connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'root',
  password  : '',
  database  : 'mydb',
});
```

mysql.createPool

mysql.createPool is a place where connections get stored.

When you request a connection from a pool, you will receive a connection that is not currently being used, or a new connection.

If you're already at the connection limit, it will wait until a connection is available before it continues.

A pool while one connection is busy running a query, others can be used to execute subsequent queries. Hence, your application will perform good.

Example:

- Tests

Il existe plusieurs types de tests à faire sur un projet, on peut y faire des

- tests unitaire (sur une seule unité),
- test d'intégration (vérifient que plusieurs de mes tests unitaires fonctionnent en les regroupant),
- tests end to end (la totalité de l'application).

TEST unitaires

Bash

```
test.sh
1  ## Testing connexion with good password
2  curl 'http://localhost:8082/login' -X POST -s --data-raw 'username=administrateur&password=' -o response.txt
3
4  if [ $(grep -c 'Found. Redirecting to admin' response.txt) ]
5  then
6  echo 'test connexion ok'
7  else
8  echo 'test connexion fail'
9  fi
10
```

J'ai créé un test unitaire en **bash** pour voir la réponse qui était renvoyée. Il s'agit d'un test d'identifiants de mot de passe qui nous renvoie la réponse dans un fichier **response.txt**.

Cette réponse que la commande '*grep <abcde>*' doit nous retourner est celle renvoyée par le navigateur quand on cherche dans l'inspecteur F12.

Pour ce test j'utilise CURL qui est une interface en ligne de commande, destinée à récupérer le contenu d'une ressource accessible par un réseau informatique.

CURL avec option :

- s permet un affichage standard une fois la commande exécutée
- o *response.txt* me renvoie la réponse dans ce fichier

GREP -c compte le nombre de fois ou l'occurrence « string » apparaît dans le *response.txt*

```
[ jeremy-Latitude-E5450 ] ~/Documents/MesDevs/Chargemap bash test.sh
test connexion ok
```

On note que le test est passé (ligne 6 du fichier **test.sh**)

```
POST /login 302 112.249 ms - 27
```

On remarque aussi que le serveur a renvoyé une *réponse 302 - Not modified*

Type de réponse de requête rencontrés :

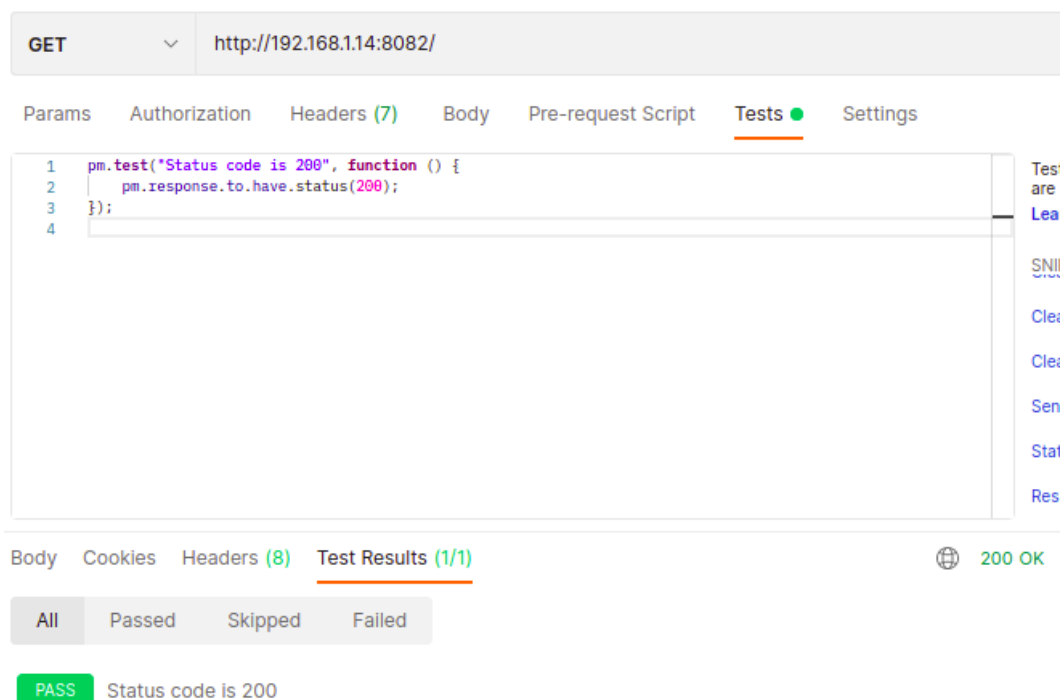
-200 : réussite de la requête

-404 : page inexistante

-304 : pas besoin de retransmettre les ressources demandées. C'est une redirection implicite vers une ressource mise en cache.

-400 : requête mauvaise ou mal formulée

Postman



Un test de réponse statut 200 a été effectué... On envoie la requête puis la page renvoie bien une réponse en statut 200... le test est validé !

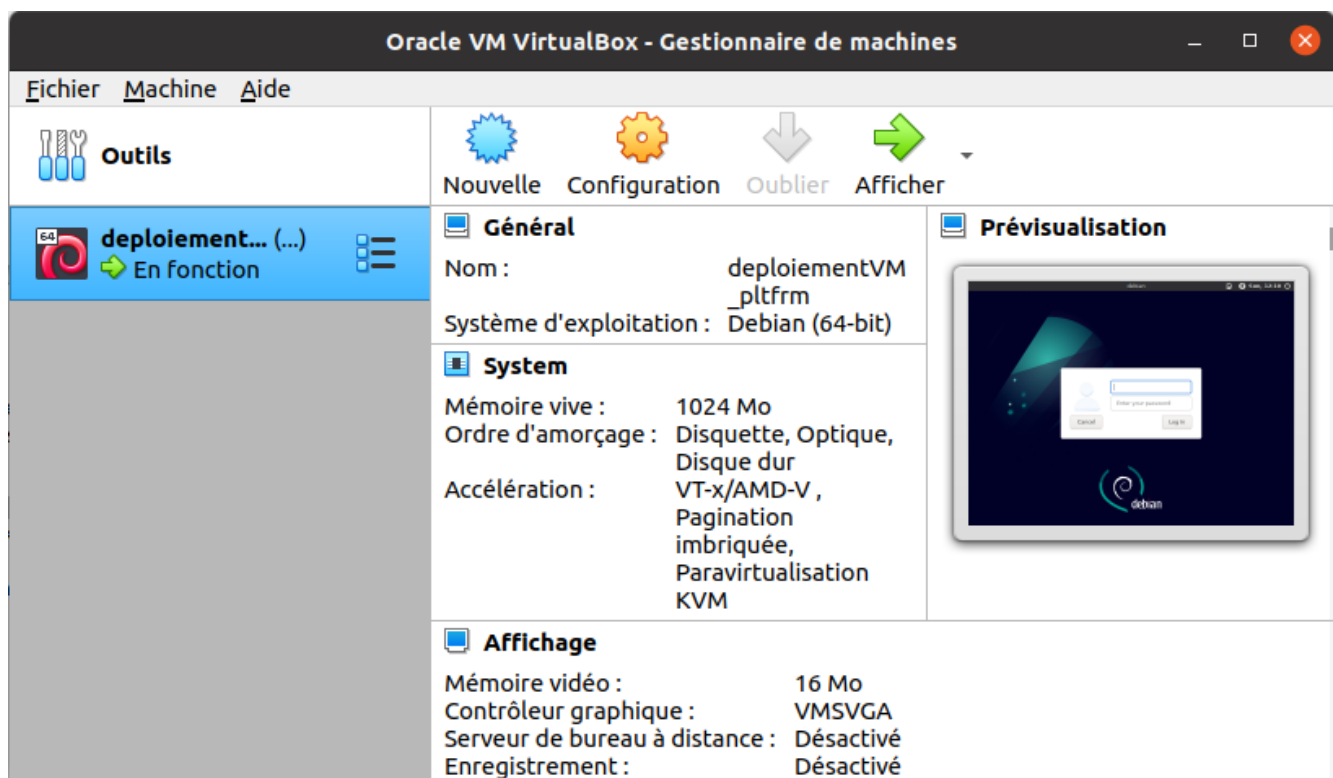
Préparer et exécuter le déploiement d'une application

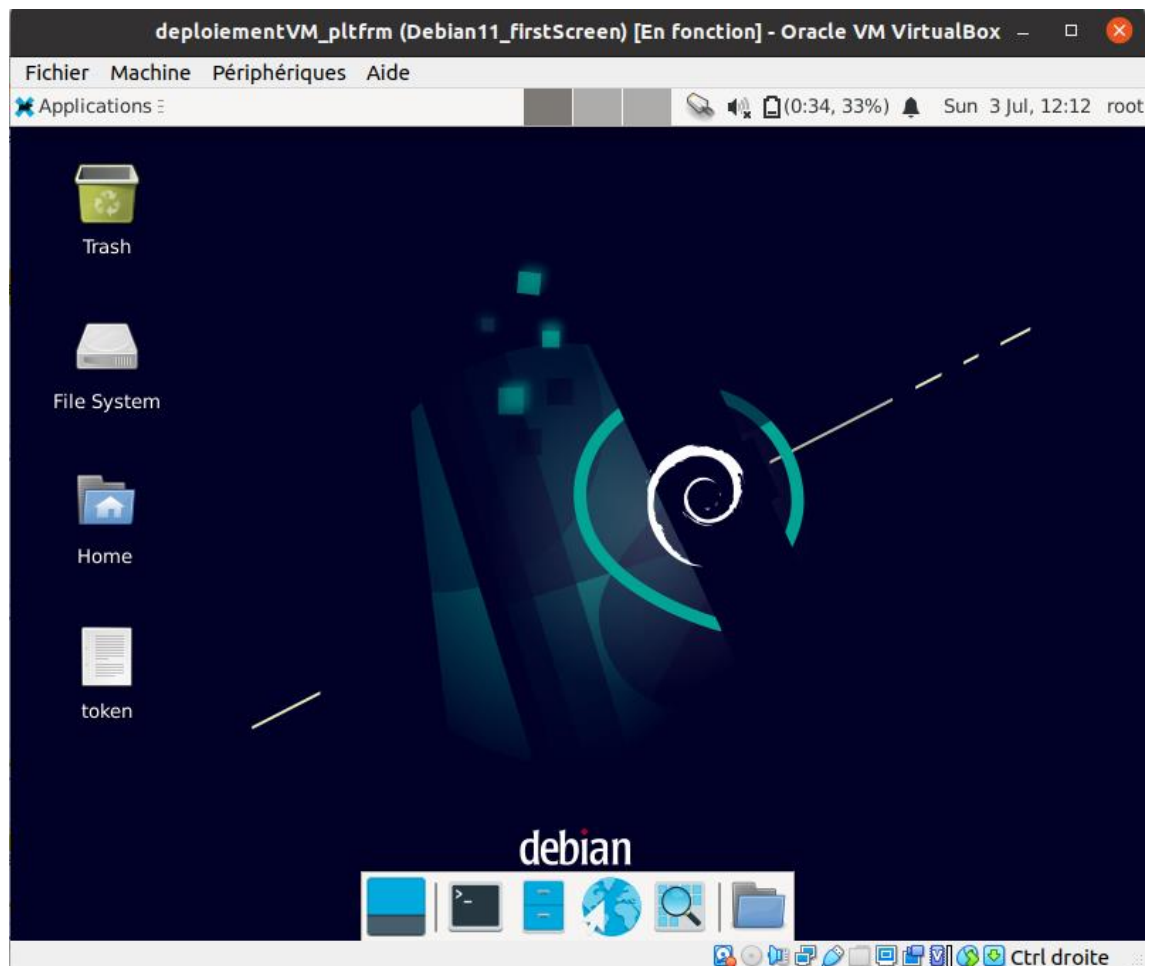
- Machine virtuelle

Au cours de l'année nous avons appris à déployer une Machine Virtuelle sur notre ordinateur. Le but d'une VM est de simuler un autre système d'exploitation que celui de base installé sur notre propre pc.

Lors de l'exercice on a dû créer une VM qui est de base vierge : c'est-à-dire que l'on commence par choisir un certain espace de stockage à allouer à cette VM, pour être tranquille j'ai choisis 30go.

Ensuite nous avons installé un système d'exploitation Debian 64-bit...



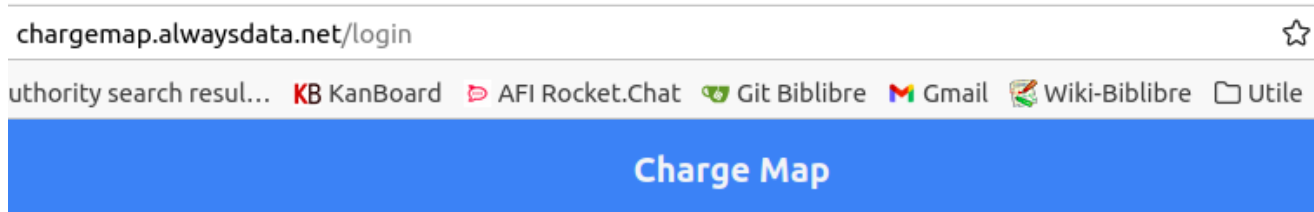


La fenêtre dans laquelle la Machine Virtuelle est ouverte

- Alwaysdata

Nous avons décidé d'héberger notre projet sur l'hébergeur en ligne **alwaysdata** qui propose un hébergement gratuit tant que l'ensemble des fichiers pèse moins de 100Mo

<http://chargemap.alwaysdata.net/login>



Connexion

Nom d'utilisateur

Mot de passe

Connexion

Exemple de jeu de manipulation de données

```
/** GET all */
const getAll = (table, order_by = 'created_at') => {
  console.log('GET', table, 'CALL');

  return new Promise((resolve, reject) => {
    db.query(
      `SELECT * FROM ${table} WHERE deleted_at IS NULL ORDER BY ${order_by} DESC `,
      (error, result) => {
        if (error) {
          console.log('ERROR: ', error);
          reject({ type: 'ko', code: 500, message: error });
        }
        resolve({ type: 'ok', code: 200, data: result });
      }
    );
  });
};
```

Comme décrit plus haut dans le paragraphe « Controller » il y a un fichier **request.js** dans le projet qui répertorie toutes les requêtes formulées en SQL... c'est notre Modèle !

On remarque que dans les requêtes il y a des variables qui changent d'état en fonction de la table qui est appelée. On peut appeler la fonction **getAll()** sur n'importe quelle table ; c'est une requête simple qui permet de récupérer toutes les entrées d'une table. On évite ainsi le **DRY** et le code est plus propre

Autre exemple mais cette fois-ci côté mobile... On remarquera qu'à la fin de la requête le résultat et l'erreur sont changées en JSON lignes 181 & 184

```
166 ///////////////GET/////////////////
167 /* GET all 'emplacement' from db */
168 router.get('/emplacement', (req, res) => {
169   if (
170     req.query.username != config.api.username &&
171     req.query.password != config.api.password
172   )
173     return res.status(401).json({
174       type: 'ko',
175       code: 401,
176       error: 'user not authenticated'
177     });
178   requests
179     .getAll(db_tables.charge_placement.name)
180     .then(result => {
181       return res.status(result.code).json(result);
182     })
183     .catch(error => {
184       return res.status(error.code).json(error);
185     });
186 });
```

Exemple de composants métiers : React-Native

La caractéristique principale de React est la composition de composants... les composants écrits par des personnes différentes fonctionnent correctement ensemble. Nous pouvons ajouter des fonctionnalités à un composant sans impacter le reste de la base de code, il est possible d'introduire un 'état local' grâce à `useState()` sans changer les composants qui l'utilisent

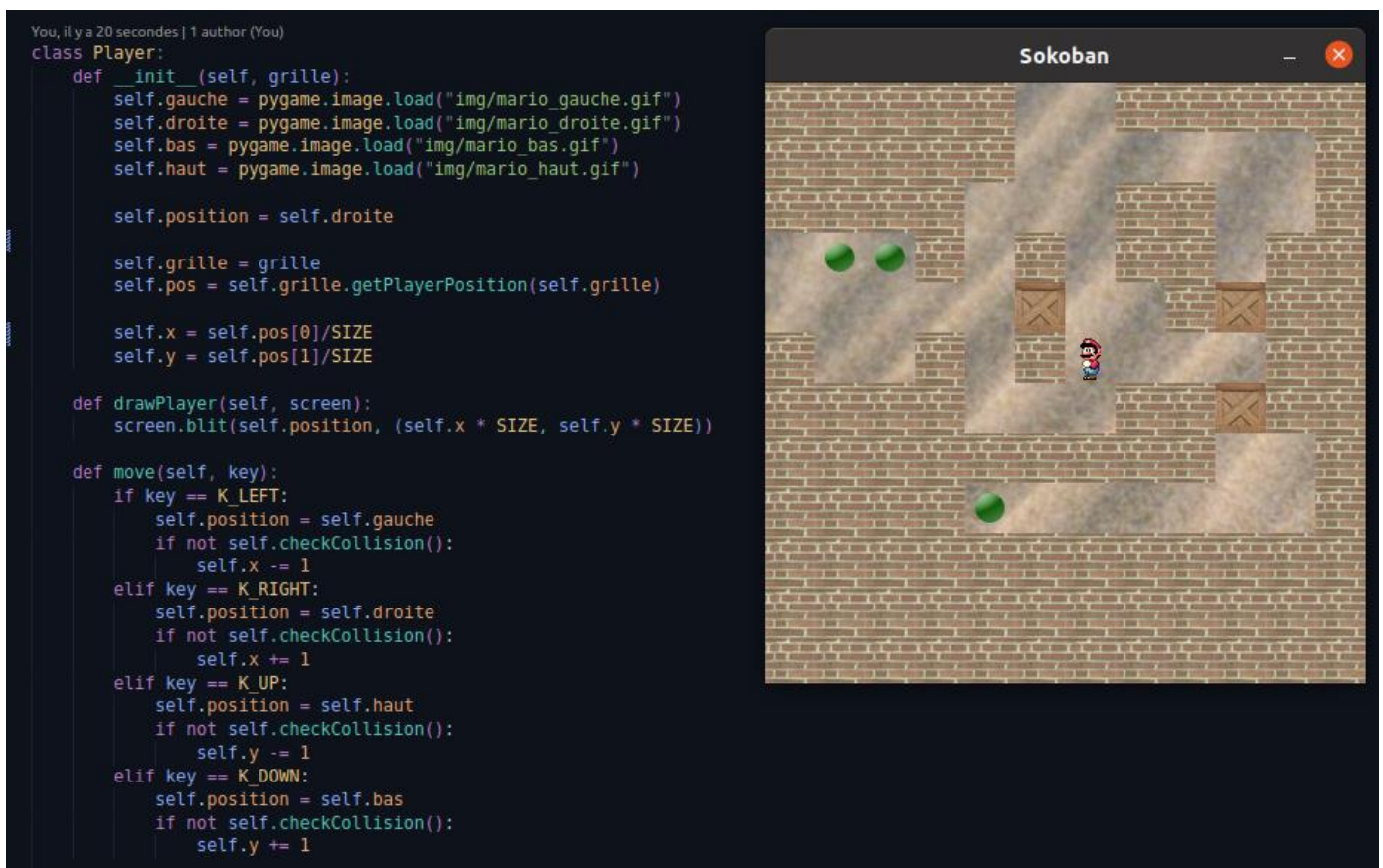
```
<View>
  <Button
    mode='contained'
    onPress={handleCamera}
    style={styles.photoButton}
    color='#353535'
    compact
  >
    Prendre une photo
  </Button>
  <Button
    mode='contained'
    onPress={handleLibrairy}
    style={styles.photoButton}
    color='#353535'
    compact
  >
    Choisir dans la bibliothèque
  </Button>
</View>
```

On remarque sur la gauche le composant **View** et ses enfants les composants **Button** : Ce sont les mêmes mais avec un état différent qui ne leur donne pas la même utilité dans l'application. Le View permet l'affichage du parent sur la page

Développer une interface utilisateur de type desktop

Pendant mon année de formation nous avons eu l'opportunité de créer une application desktop : c'est un programme directement utilisé pour réaliser une tâche ou un ensemble de tâches.

Ce programme est un **Sokoban**, c'est un jeu vidéo de réflexion japonais qui signifie « garde-d'entrepôt » : il consiste à déplacer chaque caisse à l'emplacement désigné, quand un niveau est terminé on passe à un autre avec une difficulté supérieure



On peut voir ici qu'à chaque instanciation de la classe Player, le constructeur définit les différentes images qui vont animer le personnage en fonction de ses mouvements.

La fonction *drawPlayer* permet de dessiner le personnage sur une surface source choisie.

La fonction *move* sert à faire les déplacements du personnage.

Veille technologique

Les vulnérabilités de sécurité,

Nous avons tous un point faible, même les machines et les ordinateurs. J'ai fait une recherche sur les vulnérabilités de sécurité courantes et je me suis aperçu qu'il y avait plusieurs failles exploitables dans un site web non à jour. De nouvelles méthodes sécurisantes voient le jour mais aussi de nouvelles façons de les déjouer, en voici quelques exemples :

- ⇒ Les **failles XSS** de son nom complet **Cross-Site Scripting** est une faille qui permet d'injecter du code HTML et/ou Javascript dans des variables ou des bases de données mal protégées. Il consiste à injecter du code dans une variable afin de faire en sorte que le site visé se connecte à un autre site pirate distant (Cross-site) contenant du code malveillant. Une fois le code malveillant exécuté le site « pirate » pourra accéder aux cookies laissés par les utilisateurs sur le site d'origine.
- ⇒ Les **injections SQL** sur un site non-sécurisé permettent d'injecter des données ou d'en supprimer, ce qui peut avoir de graves conséquences.
- ⇒ L'attaque par **force brute** permet à un pirate de tester des identifiants et mots de passes à l'aide d'un script très rapide sur un formulaire par exemple.
- ⇒ L'**Upload** de fichier consiste à faire passer un fichier pirate PHP par exemple pour une image.
- ⇒ Le **Buffer Overflow** est une technique complexe qui consiste à utiliser les limitations du langage C pour provoquer des dépassements de mémoire et permettre l'exécution de code malveillant.

Le hachage en SHA1 et en MD5 est devenu obsolète car il est rapidement déchiffrable par une machine avec forte puissance de calcul.

Une machine peut décrypter n'importe quel mot de passe, il faut lui compliquer la tâche :

On utilise désormais le `password_hash()` (ou le `crypt()`) qui permet de crypter le mot de passe et de rajouter un « grain de sel », une donnée additionnelle qui renforce significativement la puissance du hachage.

Une autre manière de sécuriser les input est la fonction `htmlspecialchars()` qui va donner une version texte de la valeurs des inputs dynamique du site.

Pour la lecture du mot de passe lors d'une connexion on utilise `password_verify()`

Conclusion

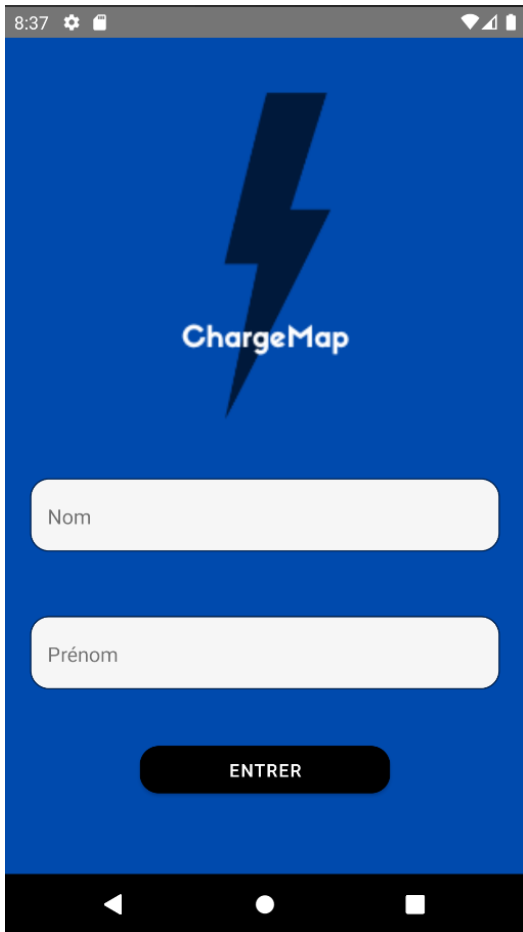
En conclusion, le site fait avec David Shlimon m'a appris à travailler sur un projet en équipe et aussi la création de ma première application. J'ai eu la chance d'apprendre de nouveaux langages et frameworks comme le Pug en frontend et le NodeJS, le shell en backend.

Cette application d'utilité privée et publique sera un outil de choix pour l'entreprise concernée, les administrateurs peuvent gérer les nouveaux emplacements de bornes électriques via leur dashboard et dans leurs locaux, pendant que les techniciens sur le terrain reçoivent les infos en temps réel, installent les bornes sur site et les sauvegardent sur l'application qui envoi une requête post à l'API.

Lors de mon alternance j'étais sur des technologies différentes (Linux, Perl) qui m'ont permis de comprendre la syntaxe, la construction et l'architecture du langage de programmation dans sa globalité ; j'ai réalisé que pour apprendre il faut s'y intéresser régulièrement en faisant de la veille constamment, en suivant l'ascension de la tech et en codant régulièrement des projets personnels.

Je suis content d'en être arrivé là et je souhaite continuer mes études dans le développement pour prendre de l'expérience.

Annexe



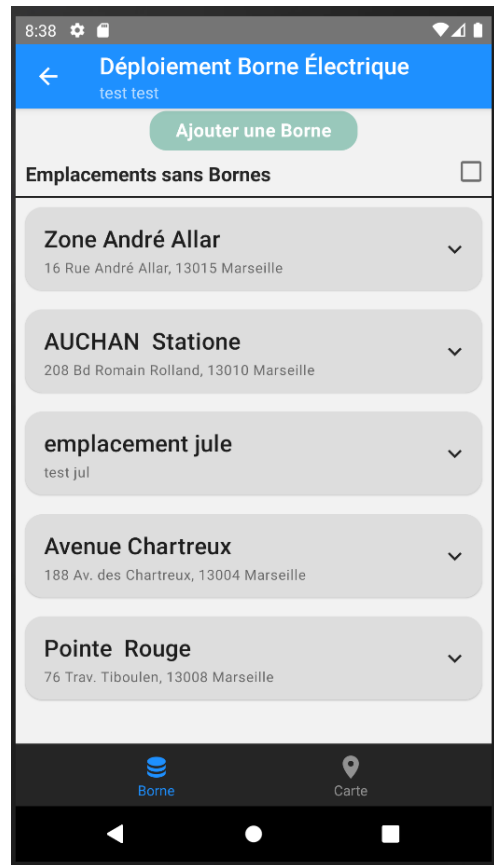
8:37

ChargeMap

Nom

Prénom

ENTRER



8:38

← **Déploiement Borne Électrique**
test test

Ajouter une Borne

Emplacements sans Bornes

Zone André Allar
16 Rue André Allar, 13015 Marseille

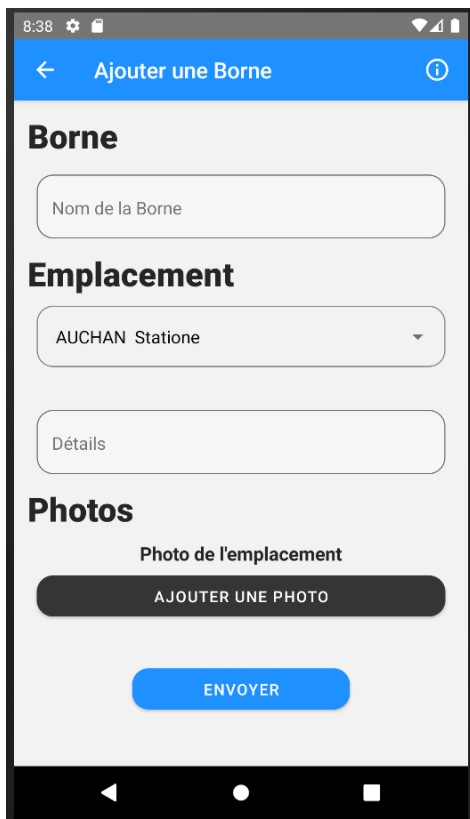
AUCHAN Statione
208 Bd Romain Rolland, 13010 Marseille

emplacement jule
test jul

Avenue Chartreux
188 Av. des Chartreux, 13004 Marseille

Pointe Rouge
76 Trav. Tiboulén, 13008 Marseille

Borne Carte



8:38

← **Ajouter une Borne** ⓘ

Borne

Nom de la Borne

Emplacement

AUCHAN Statione

Détails

Photos

Photo de l'emplacement

AJOUTER UNE PHOTO

ENVOYER



8:39

← **Ajouter une Borne** ⓘ

Borne

Nom de la Borne

Emplacement

Ajouter une photo

PRENDRE UNE PHOTO

CHOISIR DANS LA BIBLIOTHÈQUE

ANNULER

Photo de l'emplacement

AJOUTER UNE PHOTO

ENVOYER