REMERCIEMENT

17 janvier 2020

BORIBOUN Steven

ProxiDej

3 Rue carre de vigne, 44690, Monnières

# Table des matières

[I. Table des matières 2](#_Toc39190992)

[II. Partie Site WEB – Florian ENET 3](#_Toc39190993)

[1. Contexte 3](#_Toc39190994)

[2. Page Suivie de Livraisons 3](#_Toc39190995)

[3. Page de la chambre Froide 5](#_Toc39190996)

[4. La page d’historique 6](#_Toc39190997)

[5. La page Profil 7](#_Toc39190998)

[6. La page Administration 7](#_Toc39190999)

[7. La barre de navigation 7](#_Toc39191000)

# Partie Site WEB – Florian ENET

## Contexte

La partie site WEB doit pouvoir permettre la consultation des différents relevés présentes dans la base de données. Si un utilisateur non-connecté souhaite accéder au site, le site signale qui doit se connecter afin d’accéder au page. Nous ne pouvons pas créer de compte directement. Seul un administrateur peut vous créer un compte. Ce site possède une carte qui retrace les coordonnées GPS des différents véhicules. Les tracés peuvent être changé de couleur via la page profil. Si pendant le trajet, l’un des véhicules relève une température qui ne respecte pas la chaîne du froid, la carte affichera un marqueur relevant l’erreur. Un overlay permet d’afficher ou non, les différents tracés des véhicules, l’entreprise Proxidej et les marqueurs. La page chambre froide affiche un graphique selon la date choisi, correspondant à la température de la chambre froide de Proxidej au cours du temps. La page historique affiche toutes les données du jours de la chambre froide par défaut. Ensuite, selon vos envies, vous pouvez choisir si vous souhaitez accéder aux infos de la suivie de livraison et selon le jour. La page profil permet de modifier les couleurs des tracés, comme cités précédemment. La page Administration est disponible seulement si vous avez les droits administrateurs. Elle affiche tous les utilisateurs de la base de données avec la possibilité de créer un compte, d’éditer un compte, réinitialiser le mot de passe et de pouvoir le supprimer. La dernière page « add\_data », permet aux Raspberry des véhicules d’insérer des champs dans la base de données.

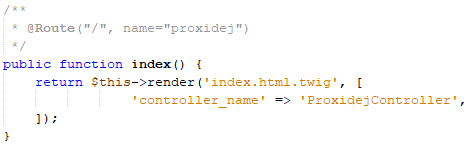
## COVID-19

Durant la réalisation de ce projet, nous étions en période de confinement à la suite du scandale sanitaire du Coronavirus. Nous avons donc dû réaliser notre projet chez nous. Nous avons pu communiquer avec le groupe grâce à différent moyen de communication. Nous avons commencé avec la création d’un github, puis finir avec un Discord. GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Il nous a permis de mettre à jour notre projet. Discord est un logiciel propriétaire gratuit de VoIP. Discord nous a permis de faire des réunions, afin de nous aider mutuellement.

## Installation de Symfony

Afin d’utiliser Symfony, il demande plusieurs prérequis. D’abord, avoir php7.1.9, donc pour se faire, je vais installer WampServer. Car de base, il possède php7.2 et en plus, il nous faut une base de données. Il nous faudra aussi l’outil Composer. Composer permet de télécharger des librairies et de les ajouter directement dans notre projet. Une fois téléchargé, je peux exécuter la commande : « composer create-project symfony/skeleton proxidej ». Cette commande permet de créer un projet Symfony nommé « Proxidej ». On accède au nouveau dossier Proxidej et exécuter « symfony server :start » dans notre invité de commande afin de lancer notre site web en local. Il est accessible sur notre navigateur web en allant sur « 127.0.0.1 :8000 ». Je vais utiliser PhpStrom et Netbeans afin d’éditer les pages du projet Symfony. Ils sont tous les deux adaptés pour faire du HTML, du CSS et du PHP.

Au-dessus de chaque fonction où je souhaite afficher une page, il faut définir une route. Une route permet de définir l’url de la page. Par exemple, ci-dessous, ce sera mon index, car elle nécessite aucune route pour y accéder. Le nom écrit à sa droite permet d’identifier le chemin. On pourra utiliser le nom pour faire des redirections. Par exemple en utilisant « {{path('proxidej') }} », celle-ci nous renverra sur l’index. Ce qui est entre crochet défini les variables php que l’on veut transmettre dans la page.



Nous pouvons voir toute les routes possible avec la commande « php bin/console debug :router ». Ci-dessous se sont toutes les pages disponibles sur le site.

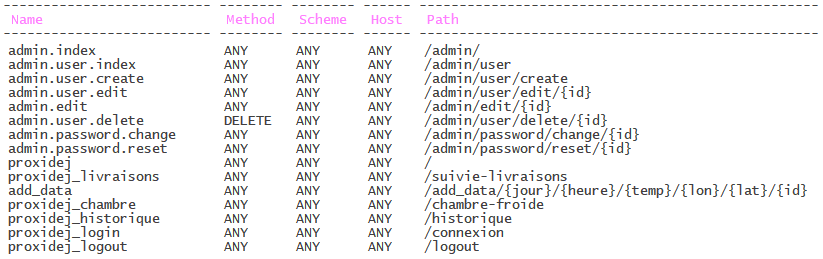


Figure Route des pages

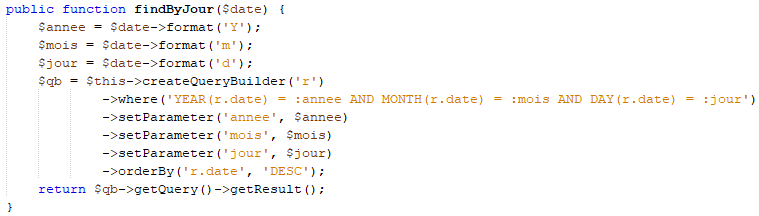
Les pages de notre site sont en « .html.twig ». Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le framework Symfony. Pour utiliser des références Twig, il suffit d’ouvrir deux accolade tel que : « {{app.user.pseudo}} » pour afficher le pseudo de l’utilisateur. Si nous souhaitons faire utiliser des fonctions, il faut utiliser une accolade suivit d’un pourcentage tel que : « {% if app.user %} », pour savoir si l’utilisateur est connecté. Pour commencer avec la création de nos premières pages, je dois configurer notre « template » de base ou modèle. Il se nomme « base.html.twig », celui-ci possède le style css. Ce qui permet que toute les pages possèdent le même style. Il suffit juste de mettre cette ligne avant chaque début de page : «{% extends 'base.html.twig' %} ». J’ai décidé que sur chaque page, nous aurions une « navbar » permettant de changer de page. Cliquer sur le logo de Proxidej nous permet de retourner sur l’index et selon sur quelle page nous sommes, la « navbar » change. Voici un aperçu du « navbar » ci-dessous. Le bouton à droite permet de nous déconnecter.



Figure Barre de navigation

## Base de données

Pour utiliser une base de données avec Symfony, il faut créer une doctrine. Cette doctrine nous permettra d’accéder et de modifier une base de données. D’abord, nous devons l’installer avec Composer. Puis indiquer l’URL de la base de données dans le «. env ». Pour l’instant, elle reste en local et est égal à : DATABASE\_URL=mysql://root:@127.0.0.1:3306/bddproxidej?serverVersion=5.7. Maintenant que la liaison est faite, nous pouvons la créer directement avec un invité de commande. Chaque nouvelle table créé, c’est deux nouveaux documents créés dans notre projet. Un objet php dans le dossier « entity », qui possède le nom des colonnes, avec ces accesseurs. J’ai utilisé la commande « php bin/console make:entity –overwrite » afin d’ajouter des nouvelles colonnes à mon objet. Puis il faut faire une migration, Il compare toutes vos entités à l'état actuel de la base de données et génère le SQL nécessaire pour les synchroniser. Le second document est un « repository ». Le repository permet d’accéder aux informations de la base de données. Nous avons la fonction « findAll() » de base, comme indiqué, elle récupère toute les information de l’entité. Mais pour mes pages, je souhaite avoir les données d’une entité selon le jour. Pour mes deux entités « relevé » et « vehicule », j’ai une colonne date de type DateTime. Grâce au formulaire de date, que je vais vous expliquez au chapitre des formulaires, je reçois le choix de date de type DateTime. Donc pour récupérer les informations du jour, j’utilise cette fonction :



Pour pouvoir accéder aux données, il suffit d’indiquer le repository et quelle fonction utilisé avec ses paramètres.



La variable « chambreDonnee » est un tableau contenant toutes les données du jour.

Pour travailler dans des conditions réelles, il me faut des données dans la base. Grâce à un bundle de Symfony nommé Fixture. Fixtures (jeu de données) est un ensemble de données qui permet d’avoir un environnement de développement proche d’un environnement de production avec des fausses données. J’ai besoin de composer pour ajouter cette librairie avec cette commande « composer require --dev orm-fixtures ». Puis dans les fichiers de fixtures d’entités, je fais une boucle for pour avoir plusieurs données. Je met les valeurs souhaité avec les accesseurs. Je finis la fonction load avec un « flush », cela me permet d’envoyer les données vers la base. Pour finir avec la commande dans un invité « php bin/console doctrine:fixtures:load » dans le dossier Proxidej. Grâce à cette librairie, il est beaucoup plus rapide d’envoyer des fausses données dans la base.

## Page Suivie de Livraisons

Cette page doit contenir une carte et j’ai choisi OpenStreetMap qui est gratuit, contrairement à l’API[[1]](#footnote-2) de Google (google maps). Ce plan est affiché avec Javascript, donc après avoir téléchargé les scripts en local, on peut l’incorporer. Afin que la carte s’affiche dès le lancement de la page, il suffit de lancer la fonction d’affichage lors du chargement de la page (onload). Une fois la carte affichée, je centre la carte sur l’entreprise Proxidej, en utilisant la position en latitude et en longitude de l’entreprise. Maintenant que la carte est prête, il faut afficher les tracés. Pour les tracés GPS, dans la base de données, je possède seulement des données de latitude et de longitude. Ce qui ne peut pas être traduit par OpenstreetMap directement. Donc j’ai créé un fichier GPX par véhicule, comportant les données GPS. Un fichier GPX permet de faire de l’échange de coordonnées GPS. C’est une succession de point qui forme un itinéraire. Pour chaque tracé, il faut rajouter un calque. Ce calque permet de nommer le nom du tracé, donner l’emplacement du fichier GPX, donner son format et sa couleur. La création du fichier GPX se fait à chaque chargement de page. Les fichiers GPX fonctionnent sous forme de balise tels que :

<?xml version="1.0" encoding="UTF-8"?>

<gpx version="1.0">

<name>Example gpx</name>

<trk><trkseg>

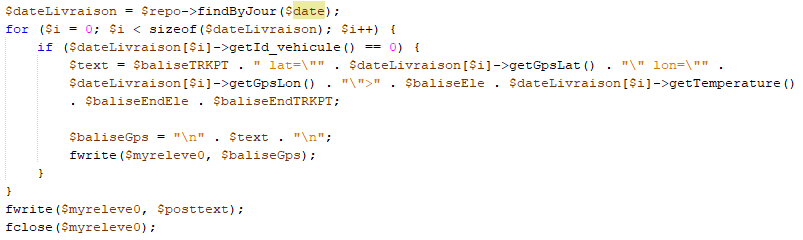
<trkpt lon="11.707633137702942" lat="44.35309951704362"><ele>51.0</ele><time>2010-12-26T17:07:40.421Z</time></trkpt>

[…]

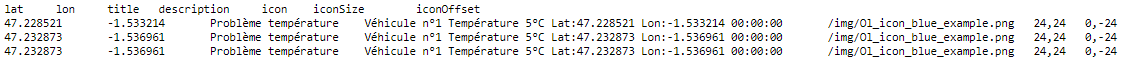
</trkseg></trk></gpx>

A chaque chargement de la page, les fichiers deviennent vides et les fichiers ajoutent les premières balises. Pour récupérer les données de la base de données, nous n’envoyons pas de requête SQL « normal ». Nous utilisons le repository expliqué dans le chapitre « Base de données ». Maintenant, dans une boucle for allant de zéro, jusqu’au nombre de données GPS du jour choisi via un formulaire ou par défaut c’est date du jour. Dans cette boucle, il y a une condition permettant de filtrer les données des différents véhicules. Nous devons réaliser trois fois cette opération, car nous ne pouvons pas modifier deux fichiers en même temps. Une fois les données GPS sont inscrites dans le fichier, on ferme les balises et on l’enregistre. Voici une boucle for pour l’écriture du fichier GPX pour le véhicule 0 comme exemple avec la valeur des différentes balises utilisées.





A la fin de l’écriture des données GPS, nous refermons les balises. Le calque peut maintenant être opérationnel. Afin de savoir si le véhicule a dépassé la limite de température imposé, des points d’intérêts seront placé. Pour la création des points d’intérêt, il nous faut créer un autre calque. Ce calque indique juste l’emplacement du fichier txt. Ce fichier, est un tableau créé avec des tabulations. Il comprend, les colonnes de latitude, de longitude, du titre, de la description, de l’emplacement de l’icône, la taille de l’icône et du compenseur. La description comprend, le numéro du véhicule, la température mesurée, les coordonnées GPS et l’heure. Seul inconvénient, il ne prend pas en compte la dernière ligne. Il suffit juste de réécrire la dernière ligne une fois de plus. Ci-dessous voici le fichier txt, afin de créer deux points d’intérêt.



Voici la map avec deux tracés de différentes couleurs et deux points d’intérêts. En haut à droite, nous pouvons naviguer et zoomer avec les clics de notre souris. A droite, nous avons l’overlay, celui-ci nous permet d’afficher ou non les calques. Si nous cliquons sur un point d’intérêt, voici le texte qui peut nous afficher.

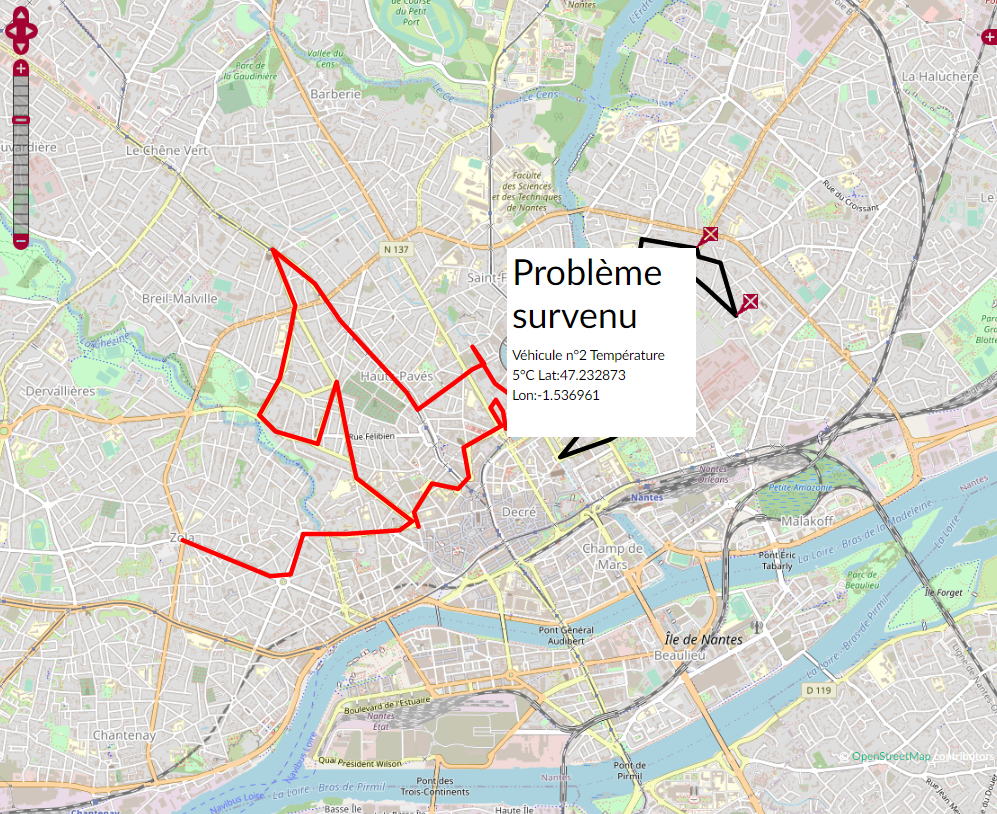
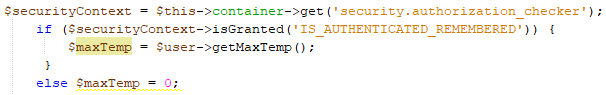


Figure Carte de la page Suivie de Livraison

Cependant, il y a un problème lorsque l’on veut accéder à cette page lorsque l’on n’est pas connecté. Car il ne peut pas accéder au « getTempMax » qui permet de savoir à quelle température l’utilisateur souhaite être notifier d’un problème de température. A la base, j’avais c’est deux lignes ci-dessous. Ces lignes permettent d’accéder aux informations de la base de données pour la table user. Mais vu que si l’utilisateur n’est pas connecté et que ce code est présent dans le controller. Ceci nous affiche une erreur, car il n’arrive pas à avoir « maxTemp ».

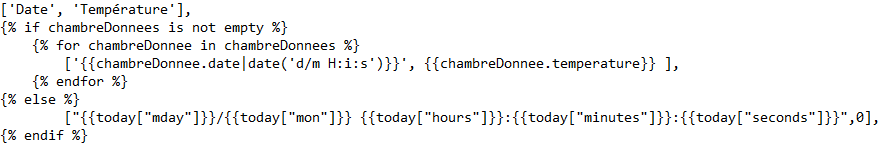


Donc pour résoudre ce problème, il nous suffit de savoir si un utilisateur est connecté avant de récupérer la valeur « maxTemp ». Ces lignes permettent de résoudre le problème.



## Page de la chambre Froide

Cette page contiendra un graphique de relevé de température de la chambre froide. J’ai choisi d’utiliser Google Charts qui permet de visualiser les données sur un site Web. Ce dont j’ai besoin c’est un graphique en ligne. Les ordonnées seront les températures relevées et abscisses les seront la date des relevées. Pour mettre des données dans le graphique, ça se fait sur Javascript. Mais je ne peux pas ajouter des valeurs via php, donc je vais utiliser Twig. Sur la page, les informations des relevées sont envoyées en paramètres par php. Avant d’ajouter des données, je vérifie que les relevés ne sont pas vides. Si elles ne le sont pas, on passe dans une boucle for, afin d’ajouter toutes les valeurs au graphique. Si au contraire, il n’y a aucune valeur, on rajoute une valeur par défaut. Voici l’aperçu de cet algorithme ci-dessous, « chambreDonnees » est un tableau possédant les valeurs de la base de données.



Si « chambreDonnees » est vide, le titre changera et ajoutera « (aucune donnée pour ce jour) ». Sinon, de base le titre est « Température de la chambre Froide ». Voici un aperçu du graphique de la chambre froide ci-dessous.

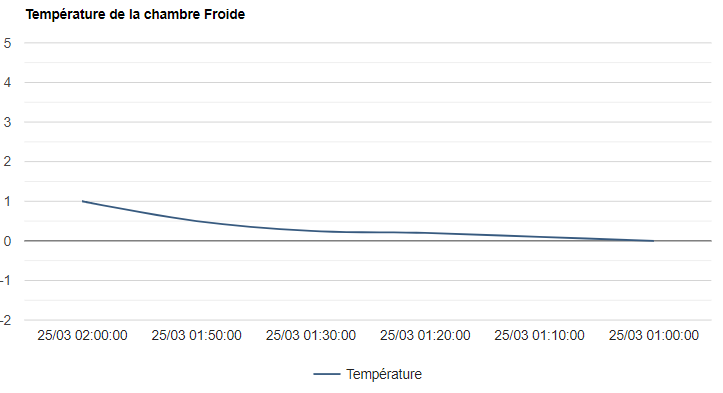


Figure Graphique de la chambre froide avec des données

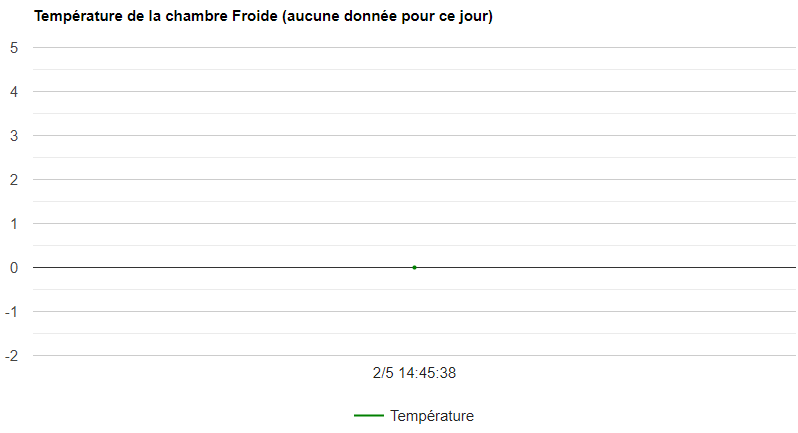
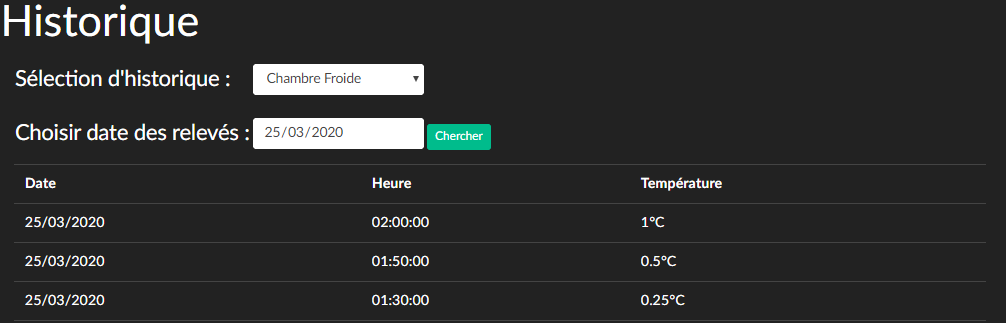


Figure Graphique de la chambre froide sans données

Le même problème est arrivé pour la page suivie de livraison sur cette page. Mais pour notre cas, ce problème ne se situait pas dans le controller, mais dans le javascript. Donc il suffisait juste de mettre un « {% if app.user %} pour savoir si un utilisateur était connecté.

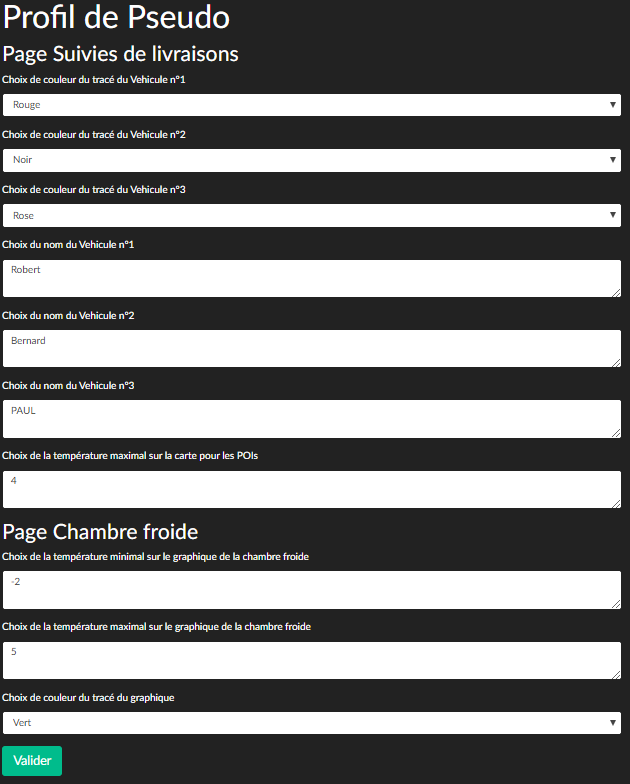
## La page d’historique

Cette page permet de voir toutes les données brutes de la base de données. Elle affiche dans un tableau les différentes données. Elle permet aussi de trier les valeurs selon la température, afin de savoir si la chaîne de froid a été brisé ou non. Grâce à un formulaire, nous pouvons choisir entre la table « Chambre froide » et « Suivie de livraisons », le jour de la prise de relevé.



## La page Profil

La page profil, permet à l’utilisateur de personnalisé le site. Il peut modifier les couleurs des différents tracés/courbes. Il peut changer les noms des véhicules sur l’overlay, modifier la valeur de maximal température acceptée. Pour la chambre froide, on peut modifier l’échelle de l’ordonnée. Toutes ces informations sont stockées sur la base de données. Quand nous accédons à cette page, nous n’avons pas besoin de recentrer nos valeurs.



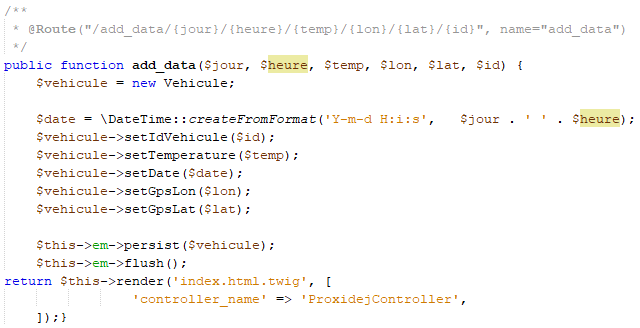
## La page Administration

Comme cité précédemment, l’inscription ne peut que se faire avec un compte administrateur. J’ai pris cette décision, pour éviter à toute personne inconnue puisse se créer un compte pour ensuite accéder à des informations confidentielles. Elle affiche les différents utilisateurs et leur(s) rôle(s). Cette page nous permet de créer un utilisateur, en éditer, réinitialiser leur mot de passe et de les supprimer. Lors de la création d’un utilisateur, il possède les droits utilisateurs. Il y a deux grades supérieurs disponible, le grade « Admin » pour créer des comptes et les éditer. Le grade « Gérant » permettant de créer un nouvel utilisateur accessible dans la page profil, mais ne peut les éditer. Pour créer un profil, nous créons un nouvel objet avec la ligne « $user = new User ». Une fois le formulaire complété, j’encode le mot de passe, afin qu’il ne soit pas écrit en brut dans la base de données. J’ai choisi l’encodage « bcrypt » qui est le plus utilisé. Je lui assigne les valeurs par défaut sur les différents champs. Il ne reste plus qu’à envoyer ces informations sur la base de données et l’utilisateur est créé.



## Page d’ajout de données

Cette page permettra à M.BORIBOUN d’envoyer des données des différents véhicules. Ça lui évite de créer des requêtes SQL avec les Raspberrys. Pour réceptionner les différentes données pour la table véhicule, il me les transfèrera via l’URL, tels que : « /add\_data/{jour}/{heure}/{temp}/{lon}/{lat}/{id} ». Ce qui est entre accolade est les différentes variables qu’il me donnera. Mais les variables de temps auront un format particulier. Le jour devra être sous forme « Y-m-d », par exemple « 2020 – 12 – 30 » pour le 30 décembre 2020. Pour l’heure, il sera de format « H:i:s », le « i » indiquant les minutes. Le format du temps m’a posé quelques soucis, car la base de données n’accepte pas de prendre une chaîne de caractères pour un DateTime. Avec l’aide de M.HOURDIN, nous terminons avec cette fonction.



1. *Application Programming Interface ou interface de programmation applicative* [↑](#footnote-ref-2)