

Rapport de laboratoire

Site de gestion de station service

|  |  |
| --- | --- |
| Département : | TIN |
| Unité d’enseignement : | Ruby on Rails |

|  |  |
| --- | --- |
| Auteur | Aebischer Luc |
|  | Salomon Quentin |
| Professeur | Lefrancois Eric |
|  |  |
| Date | 13 juin 2016 |

Table des matières

[1 Introduction 3](#_Toc453598697)

[2 Cahier des charges 4](#_Toc453598698)

[3 Structure du logiciel 5](#_Toc453598699)

[4 Implémentation 6](#_Toc453598700)

[4.1 Description de AJAX 6](#_Toc453598701)

[4.1.1 Explication du code 6](#_Toc453598702)

[4.2 Librairies utilisées 8](#_Toc453598703)

[4.3 Eléments remarquables 8](#_Toc453598704)

[5 Gestion de projet 9](#_Toc453598705)

[6 Etat des lieux 9](#_Toc453598706)

[7 Conclusion 9](#_Toc453598707)

[8 Annexes 9](#_Toc453598708)

# Introduction

Dans le cadre du cours de Web Rails, il nous a été demander de faire un projet à l’aide de ruby. Nous sommes donc partit sur un site pour les stations essences et le suivi des véhicules. Le but de notre projet est de voir un prix pour toutes les stations existantes. Ainsi que de voir la position de chaque station. En parallèle, la possibilité d’avoir un compte pour gérer un ou plusieurs véhicules permettant de suivre les consommations de ceux-ci.

# Cahier des charges

## But

* Connaitre le prix des différents carburants des stations à proximité (trouver la moins chère).
* Pouvoir enregistrer pour un utilisateur la consommation de son véhicule.
* Visualiser la position des différentes stations du site
* Réalisation d’une page principale avec AJAX et visualisation jolie

## Cas d’utilisations

* Visiteur
  + Consulter la liste des prix des stations
  + Consulter les positions des stations
* Utilisateur enregistré
  + Enregistrer des voitures personnelles
  + Enregistrement des statistiques du véhicule
  + Modifier le prix des carburants
  + + droit visiteur
* Administrateur
  + Ajout de nouvelles stations
  + Gestions des utilisateurs (Supprimer)
  + Gestion des types d’essences à disposition
  + + droit utilisateur

# Structure du logiciel

# Implémentation

## Description de AJAX

Dans notre projet l’AJAX a été utiliser pour permettre la recherche des stations de manière asynchrone. Sur la page principale l’on dispose d’une barre de recherche à la mode Google qui permet la recherche des stations dans la base de donnée. L’AJAX permet de les afficher et rafraichir cette liste sans recharger de page. Il est possible de rechercher une station par nom ou par « marque ».

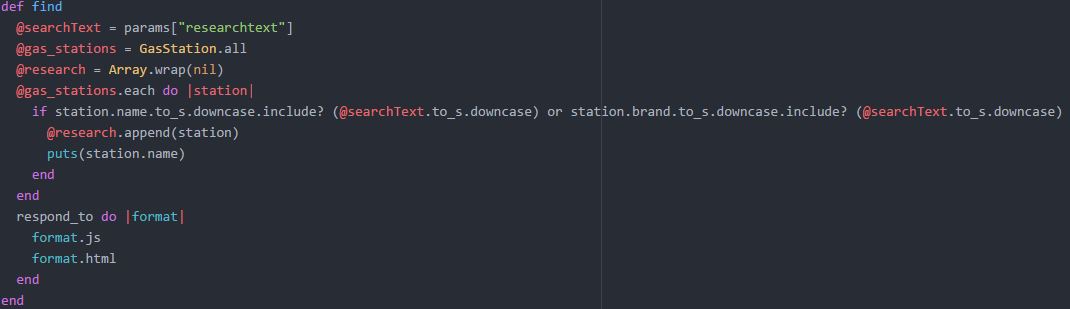
### Explication du code

Pour réaliser cette partie AJAX nous avons procéder de la manière suivante :

Lors de modification du champ de recherche une fonction, que nous avons appelé find, est appelée. C’est cette fonction que se charge de faire une nouvelle recherche dans la base de donnée et d’afficher les nouveaux résultats. Pour l’affichage des résultats nous avons trouvé judicieux d’utiliser un rendu partiel. Ceci dans le but de programmer une seul fois le rendu du résultat qui sera ensuite répéter plusieurs fois. Il nous a parut plus simple et compréhensible de faire ceci en HTML à l’aide de rails plutôt qu’en JavaScript.

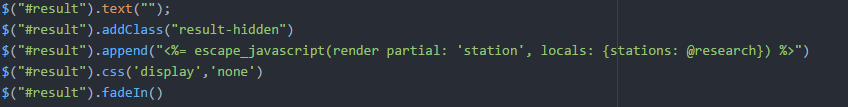
Fonction find :

La fonction find permet de faire la recherche dans la base de donnée et de faire la comparaison avec le texte présent dans la barre de recherche. Tous les résultats sont stockés dans un tableau qui est transmis à la fonction find mais JavaScript.



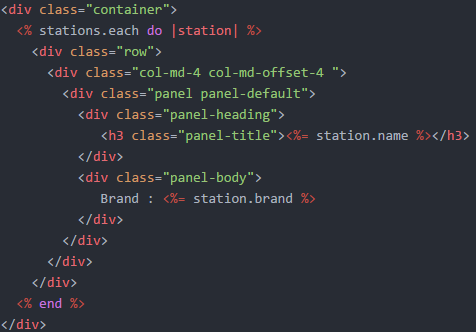
Fonction find.js

Cette partie JavaScript n’as pas un rôle des plus essentielle puisqu’elle s’occupe principalement d’appeler le rendu partiel et de la petite animation d’apparition. L’on peut constater l’utilisation du « escape\_javascript » pour faire le rendu.



Rendu :

Cette partie de code s’occupe de faire le rendu de chaque station. Il permet la création et la mise en forme des panel d’affichage. Rien de bien spécial là-dedans. Il reçoit en paramètre la station à afficher via le JavaScript ci-dessus.



## Librairies utilisées

Les librairies utilisées sont listées dans le bundle du projet. Nous pouvons tout de même noter la présence de libraire utiliser dans le laboratoire 5 pour la géstion des accès et des roles (CanCanCan et Device). Pour l’aspect visuel nous avons utilisé Bootsrap. Aucune autre librairie si ce n’est les standards ont été utilisées.

## Eléments remarquables

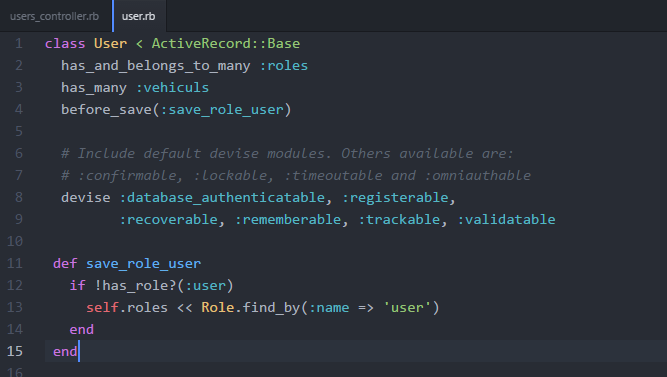
Dans ce projet il existe quelques portions de code qui nous ont demandé (beaucoup) plus de temps que prévu et qui méritent d’être detaillée.

### Ajout automatique d’un rôle

Nous avons établi la gestion des rôles sur le principe définit dans le cahier des charges. C’est-à-dire avec un rôle utilisateur et un rôle administrateur. Visiteur n’étant pas rôle puisqu’il désigne juste une personne allant sur le site.

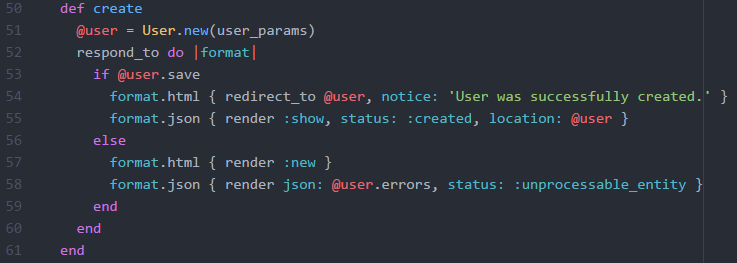
Lorsqu’une personne crée un compte, il doit obtenir automatiquement le rôle d’utilisateur. Et cela s’est avéré plutôt fastidieux étant donné qu’il a été consacré 1.5h seul + 1h avec le professeur pour trouver une solution fonctionnelle. Ceci est dû en partie à l’utilisation de la gem devise car il n’a pas été évident de trouver ou elle passait dans le code lors de la création d’un compte.

Le code final reste assez simple. Il est dans user.rb et donne ceci :



La fonction save\_role\_user est appelé avant la sauvegarde d’un utilisateur grâce à la fonction before\_save. Pour des raisons inconnues, le before\_save est appelé plusieurs fois. C’est pourquoi, on check si l’utilisateur ne possède pas déjà le rôle utilisateur ( if !has\_role?(:user) ).

Au début, le code était placé dans le contrôleur user (user\_controller.rb) lors d’un create :



En plaçant le code dans le « if @user.save », pour savoir si le compte avait bien été créé, cela ne fonctionnait pas. Pourquoi ? parce que la gems devise contourne le user controller et par conséquent, le programme ne passait jamais par ici.

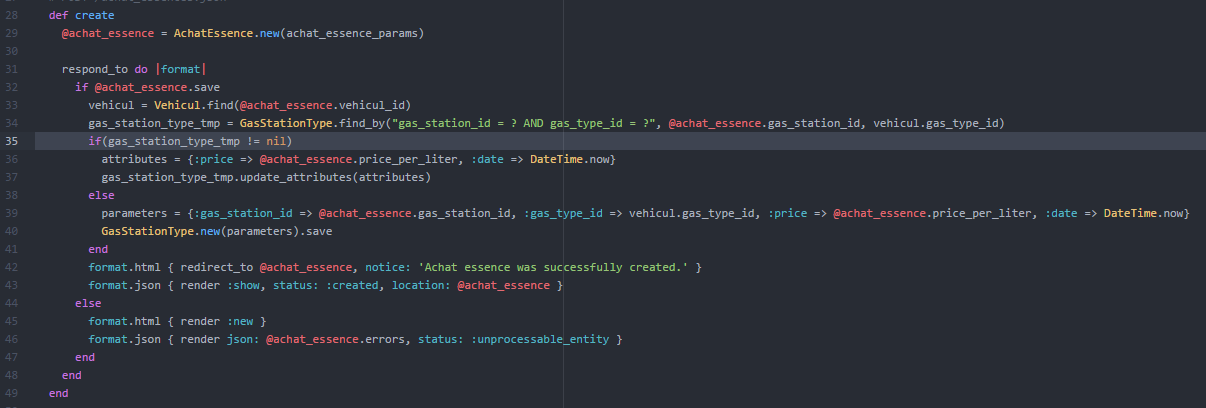
### Mise à jour automatique des prix des essences

Pour mettre à jour automatiquement le prix des essences, il a fallu modifié légèrement notre schéma relationnelle en ajoutant une liaison entre les tables « véhicules » et « type d’essences » . Avec ce nouveau schéma, on obtient qu’un véhicule à un type d’essence :



Grâce à cela, il a été possible de changer automatiquement le prix de l’essence d’une station lorsqu’un utilisateur fait un achat. C’est un point très pratique car plus il y a d’utilisateur, plus les prix sont à jours et pour chaque stations indépendamment. Remarque, la date est aussi mise à jour permettant aux utilisateurs de savoir si les prix sont récents.

On obtient le code suivant dans « achat\_essences\_controller.rb » :

En enregistrant l’achat, si on le sauvegarde, on met à jour le prix ou on le crée s’il n’existe pas. Ceci est fait avec les lignes 33-41.

Si un utilisateur édite un achat, le prix et la date sont aussi mis à jour de la même manière (lignes 56-64).

### Check box pour la sélection des essences

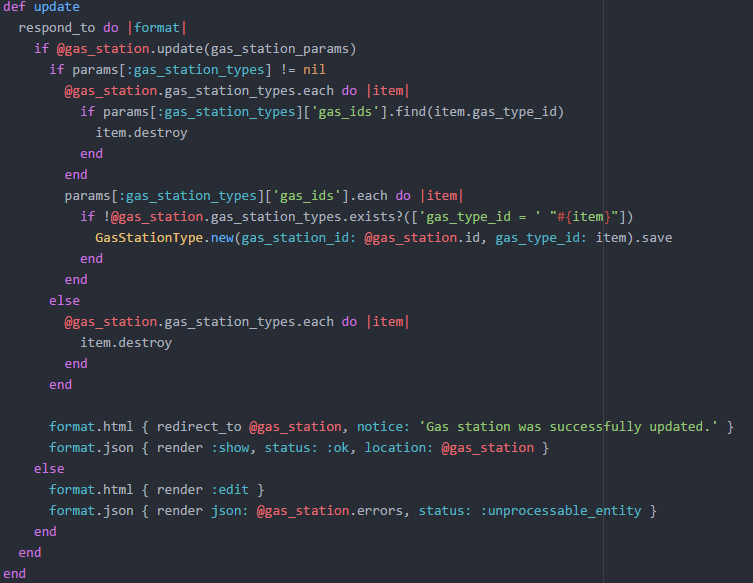
En allant dans l’édition des stations essences il est possible de sélectionner les essences disponibles dans la stations à l’aide de checkbox. Cette partie qui peut paraitre pourtant banal mais nous a donné beaucoup de fils à retordre. Cette solution nous a obligé à nous attaquer à différente partie du code. Cette relation étant n-n et est faite avec une table d’association. Elle demande donc d’aller explorer plusieurs parties de la base de donnée.

Pour réussir cette prouesse il nous a fallu ruser d’astuces. Premièrement nous avons créé un nouvel objet dans lequel allais être stocker l’ensemble des états des checkbox. Pour construire cette objet il nous à fallut d’abords aller rechercher quelle types d’essences était disponible pour la station.



Le code ci-dessus nous permet de mieux comprendre le mécanisme. Si le type d’essence est présent alors le flag est passé à true, dans le cas contraire on le laisse à false. La modification des cases à coché va directement affecter cet objet. Ainsi quand la demande de l’update est faites l’objet est passé à la fonction update pour mettre à jour.

Ci-dessous le code de la fonction update qui contrôle si les check box ont été modifier et effectue les modifications nécessaires dans la table d’association.



# Gestion de projet

Pour la gestion du projet nous nous somme partager le travail de la manière suivante.

Quentin : Réalisation des scaffolde et construction de la base de donnée

Modification des scaffolde pour afficher les informations correctes

Gestion des accès (CanCanCan, Device)

Luc : Réalisation du thème et de la partie graphique

Réalisation de la séléction des essance de manière simple par check box

Réalisation de la partie AJAX

# Etat des lieux

Dans sa configuration actuelle le projet est utilisable. Il est possible d’accéder à tous les éléments importants et les fonctions de base fonctionne. La base de donnée est construite et est utilisable. Elle laisse la possibilité à de nouvelle évolution dans le futur.

Il reste cependant beaucoup d’amélioration possible. Le coté visuel peut être nettement améliorer surtout pour les formulaires standard. La possibilité de voir les données de manière plus convivial serais un gros plus. On entend par là par exemple la possibilité de rajouter des graphiques que permettent de suivre l’évolution du prix ou de la consommation sur un mois. Il serait également possible.

Une autre amélioration à apporter est la possibilité de pouvoir visualiser sur une map google la position des stations.

# Conclusion

Pour nous qui sommes EAI cet exercice s’est révélé relativement compliqué étant donné que depuis le début du semestre nous avons pratiquement passé autant de temps à apprendre rails que le Web est ces caractéristiques. Nous sommes plutôt satisfaits du résultat même si nous espérions aller un peu plus loin encore. Néanmoins cela nous a permis de nous plonger dans ces technologies relativement complexes et ce sera sans doute un plus pour notre formation étant donné la quantité de chose qui passe par le web de nos jours.

# Annexes