HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

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

Dans le cadre du cours de Web Rails, il nous a été demandé de faire un projet à l'aide de Rails. Nous sommes donc partis sur un site pour les stations essence 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.

2 Cahier des charges

2.1 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

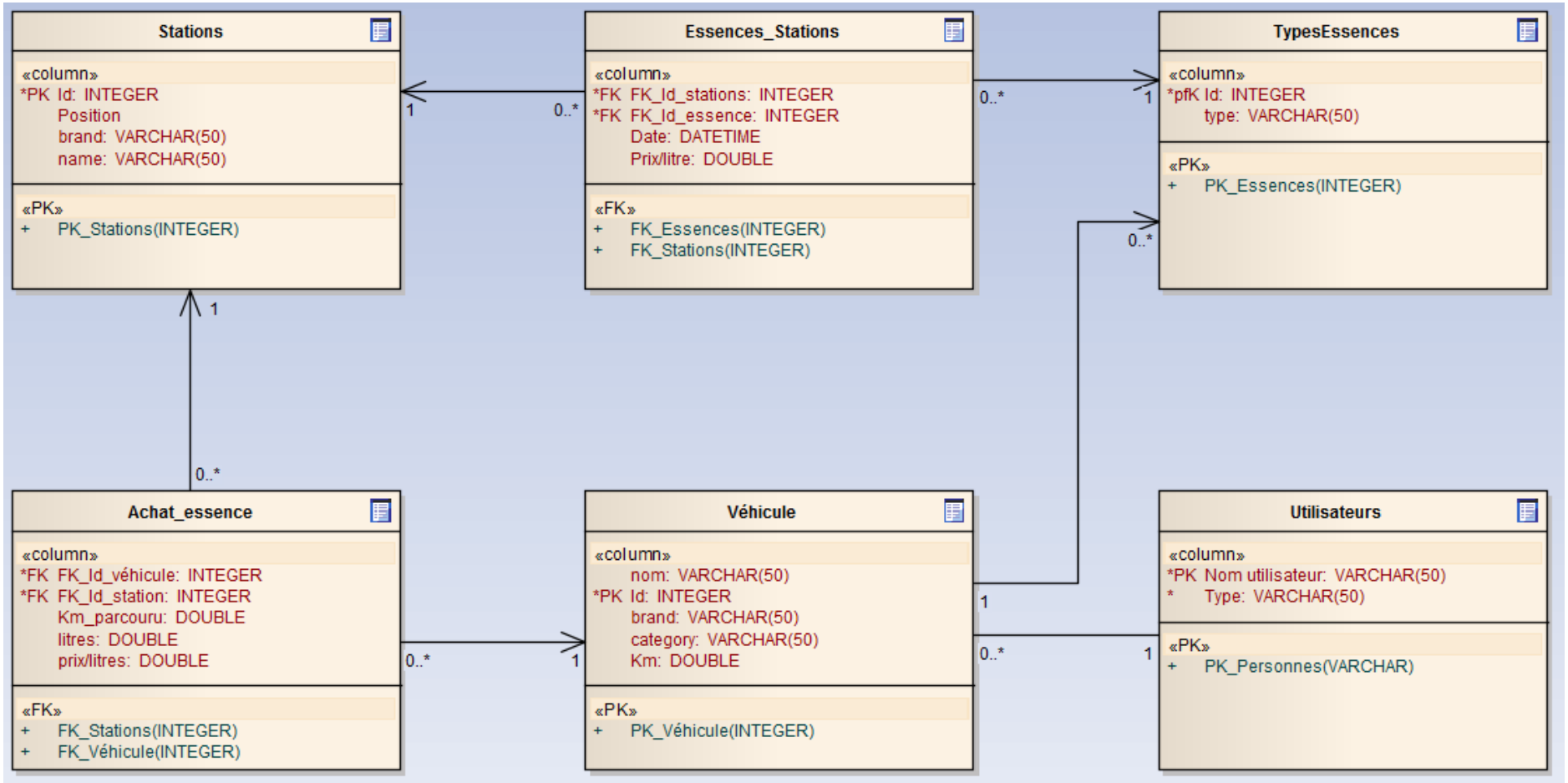
2.2 Cas d'utilisations

- 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

3 Structure du logiciel



4 Implémentation

4.1 Description de AJAX

Dans notre projet l'AJAX a été utilisé 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 données. 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 ».

4.1.1 Explication du code

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

Lors de modification du champ de recherche, une fonction, que nous avons appelée find, est appelée. C'est cette fonction qui se charge de faire une nouvelle recherche dans la base de données 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 seule fois le rendu du résultat qui sera ensuite répété plusieurs fois. Il nous a paru 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 données 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.

```
def find
  @searchText = params["researchtext"]
  @gas_stations = GasStation.all
  @research = Array.wrap(nil)
  @gas_stations.each do |station|
    if station.name.to_s.downcase.include? (@searchText.to_s.downcase) or station.brand.to_s.downcase.include? (@searchText.to_s.downcase)
      @research.append(station)
      puts(station.name)
    end
  end
  respond_to do |format|
    format.js
    format.html
  end
end
```

Fonction find.js

Cette partie JavaScript n'a pas un rôle des plus essentiel 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.

```
$("#result").text("");
$("#result").addClass("result-hidden")
$("#result").append("<%= escape_javascript(render partial: 'station', locals: {stations: @research}) %>")
$("#result").css('display','none')
$("#result").fadeIn()
```

Rendu :

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

```
<div class="container">
  <% stations.each do |station| %>
    <div class="row">
      <div class="col-md-4 col-md-offset-4 ">
        <div class="panel panel-default">
          <div class="panel-heading">
            <h3 class="panel-title"><%= station.name %></h3>
          </div>
          <div class="panel-body">
            Brand : <%= station.brand %>
          </div>
        </div>
      </div>
    </div>
  <% end %>
</div>
```

4.2 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 gestion des accès et des rôles (CanCanCan et Device). Pour l'aspect visuel, nous avons utilisé Bootsrap. Aucune autre , si ce n'est les standards, n'a été utilisée.

4.3 Éléments 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 détaillées.

4.3.1 Ajout automatique d'un rôle

Nous avons établi la gestion des rôles sur le principe défini 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 :

users_controller.rb | user.rb

```ruby
class User < ActiveRecord::Base
  has_and_belongs_to_many :roles
  has_many :vehiculs
  before_save(:save_role_user)

  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

 def save_role_user
   if !has_role?(:user)
     self.roles << Role.find_by(:name => 'user')
   end
 end
```

La fonction save_role_user est appelée 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)).

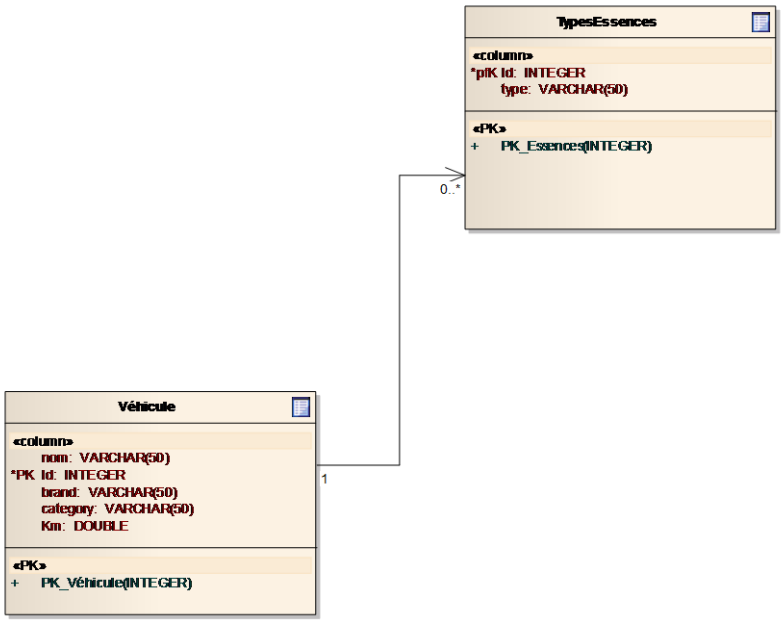
Au début, le code était placé dans le contrôleur user (user_controller.rb) lors d'un create :

```ruby
def create
  @user = User.new(user_params)
  respond_to do |format|
    if @user.save
      format.html { redirect_to @user, notice: 'User was successfully created.' }
      format.json { render :show, status: :created, location: @user }
    else
      format.html { render :new }
      format.json { render json: @user.errors, status: :unprocessable_entity }
    end
  end
end
```

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 usercontroller et par conséquent, le programme ne passait jamais par ici.

4.3.2 Mise à jour automatique des prix des essences

Pour mettre à jour automatiquement le prix des essences, il a fallu modifier légèrement notre schéma relationnel 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'utilisateurs, plus les prix sont à jour et pour chaque station 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 » :

```
28  def create
29    @achat_essence = AchatEssence.new(achat_essence_params)
30
31    respond_to do |format|
32      if @achat_essence.save
33        vehicul = Vehicul.find(@achat_essence.vehicul_id)
34        gas_station_type_tmp = GasStationType.find_by("gas_station_id = ? AND gas_type_id = ?", @achat_essence.gas_station_id, vehicul.gas_type_id)
35        if(gas_station_type_tmp != nil)
36          attributes = {:price => @achat_essence.price_per_liter, :date => DateTime.now}
37          gas_station_type_tmp.update_attributes(attributes)
38        else
39          parameters = {:gas_station_id => @achat_essence.gas_station_id, :gas_type_id => vehicul.gas_type_id, :price => @achat_essence.price_per_liter, :date => DateTime.now}
40          GasStationType.new(parameters).save
41        end
42        format.html { redirect_to @achat_essence, notice: 'Achat essence was successfully created.' }
43        format.json { render :show, status: :created, location: @achat_essence }
44      else
45        format.html { render :new }
46        format.json { render json: @achat_essence.errors, status: :unprocessable_entity }
47      end
48    end
49  end
```

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).

4.3.3 Check box pour la sélection des essences

En allant dans l'édition des stations essence, il est possible de sélectionner les essences disponibles dans la station à l'aide de checkbox. Cette partie qui peut paraitre pourtant banale, mais nous a donné beaucoup de fils à retordre. Cette solution nous a obligés 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 données.

Pour réussir cette prouesse, il nous a fallu ruser d'astuces. Premièrement nous avons créé un nouvel objet dans lequel allait être stocké l'ensemble des états des checkbox. Pour construire cet objet, il nous a fallu d'abord aller rechercher quels types d'essences étaient disponibles pour la station.

Le code ci-dessous 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 cocher va directement affecter cet objet. Ainsi quand la demande de l'update est faite l'objet est passé à la fonction update pour mettre à jour.

```
<% test = Array.wrap(nil) %>
<table>
  <thead>
    <tr>
      <% @gas_types.each do |item| %>
        <th><%= item.name %></th>
        <% test.append(item) %>
      <% end %>
    </tr>
  </thead>
  <tbody>
    <% test.each do |head| %>
    <% flag = false %>
      <% @gas_station.gas_station_types.each do |item| %>
        <% if head.id == item.gas_type.id %>
          <% flag = true %>
        <% end %>
      <% end %>
      <td>
        <%= check_box_tag("gas_station_types[gas_ids][]",head.id,flag) %>
      </td>
    <% end %>
  </tbody>
</table>
```

```
def update
  respond_to do |format|
    if @gas_station.update(gas_station_params)
      if params[:gas_station_types] != nil
        @gas_station.gas_station_types.each do |item|
          if params[:gas_station_types]['gas_ids'].find(item.gas_type_id)
            item.destroy
          end
        end
        params[:gas_station_types]['gas_ids'].each do |item|
          if !@gas_station.gas_station_types.exists?(['gas_type_id = ' "#{item}"])
            GasStationType.new(gas_station_id: @gas_station.id, gas_type_id: item).save
          end
        end
      else
        @gas_station.gas_station_types.each do |item|
          item.destroy
        end
      end

      format.html { redirect_to @gas_station, notice: 'Gas station was successfully updated.' }
      format.json { render :show, status: :ok, location: @gas_station }
    else
      format.html { render :edit }
      format.json { render json: @gas_station.errors, status: :unprocessable_entity }
    end
  end
end
```

Ci-dessus le code de la fonction update qui contrôle si les check box ont été modifié et effectue les modifications nécessaires dans la table d'association. L'on peut remarquer sont accès aux paramètres pour aller rechercher l'état de chaque check box. En cas d'ajout ou de suppression d'un type d'essence, il faut alors aller affecter la table d'association.

5 Gestion de projet

Pour la gestion du projet, nous nous sommes partagé le travail de la manière suivante.

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

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 sélection des essences de manière simple par check box

Réalisation de la partie AJAX

6 État 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 fonctionnent. La base de données est construite et est utilisable. Elle laisse la possibilité à de nouvelles évolutions dans le futur.

Il reste cependant beaucoup d'amélioration possible. Le côté visuel peut être nettement amélioré surtout pour les formulaires standard. La possibilité de voir les données de manière plus conviviales serait un gros plus. On entend par là par exemple la possibilité de rajouter des graphiques que permettre 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.

7 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 choses qui passent par le web de nos jours.

8 Annexes