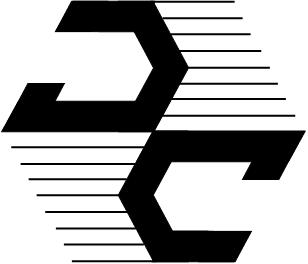
National College of Ireland

MSc in Web Technologies

2013/2014



Derek Caprani

X09106561

[derek.caprani@student.ncirl.ie](mailto:Derek.caprani@student.ncirl.ie)

dcaprani@me.com

DeeSee Car Lot

Technical Report



Contents

[Executive Summary 3](#_Toc375121585)

[Introduction 3](#_Toc375121586)

[1.2 Ruby on Rails Framework 3](#_Toc375121587)

[1.3 Ruby 4](#_Toc375121588)

[1.4 Rails 4](#_Toc375121589)

[Implementation of Features 4](#_Toc375121590)

[1.5 Allow a user to upload the details of a vehicle including a photo. 5](#_Toc375121591)

[1.6 Allow a user to Login/Logout & only allow logged in users to purchase. 5](#_Toc375121592)

[1.7 Allow a user to tick various options using the Decorator Pattern. 5](#_Toc375121593)

[1.8 Allow a user to log if a vehicle had been crashed using the Singleton Pattern 5](#_Toc375121594)

[1.9 Allow the system to check the “Crash Log” and display a response 5](#_Toc375121595)

[1.10 Allow a user to search the database by certain criteria and display matching results. 6](#_Toc375121596)

[1.11 Allow a user to select a Vehicle to view its full details and then allow the user to purchase this Vehicle if he wishes. 6](#_Toc375121597)

[1.12 Enable the system to show details of a purchased Vehicle including finance options. 6](#_Toc375121598)

[Conclusions 7](#_Toc375121599)

# Executive Summary

The main aim of the DeeSee Car Lot application is to enable users to either upload vehicles for sale to, or search available vehicles to buy from, the website.

Initially it was thought that it would be both aesthetically pleasing and would make for an excellent user experience, if, while a user was searching the database, the results of his choices were displayed in the form of a Spring Graph, as per the diagram below.



However it became apparent, when researching the Arbor API’s which were going to enable the use of Spring Graphs, that these were primarily a “Front End” technology and they wouldn’t really address any of the learning outcomes for the WAF Module. Having arrived at this conclusion the Spring Graphs were dropped from the project.

The dropping of the Spring Graphs then enabled me to progress much quicker with the development of the project and I set about designing an application within the Ruby on Rails framework to achieve the following:

* Allow a user to upload the details of a vehicle including a photo.
* Allow a user to login/Logout & only allow logged in users to purchase
* Allow a user to tick various options using the Decorator Pattern.
* Allow system to log if a vehicle had been crashed using the Singleton Pattern.
* Allow the system to check the “Crash Log” and display if a vehicle was or wasn’t crashed.
* Allow a user to search the database by certain criteria and display matching results.
* Allow a user to select a vehicle to view it’s full details.
* Allow a user to purchase a selected vehicle.
* Enable the system to show details of a purchased vehicle including finance options over 3, 4 or 5 years.

By and large most of the above listed features were achieved fully, and those that weren’t fully implemented still offer a degree of functionality and I’m certain that, given only a little more time, the application could be brought to the level of a fully functioning commercial Web Application.

# Introduction

## Ruby on Rails Framework

The Framework used for the project is Ruby on Rails. And the aim of the project is to illustrate the relative ease with which a fully functioning, dynamic Web Application can be designed and constructed using Ruby on Rails and implementing the MVC (Model View Controller) architecture.

## Ruby

Ruby is a powerful Object Oriented Programming language which offers all the functionality of the likes of Java, C# or Objective C, but it’s less verbose and less complex (Syntactically Speaking). For example variables are defined and interpreted on the fly, much like the var feature in JavaScript and are assigned their datatype based on their context within the given class or algorithm.

## Rails

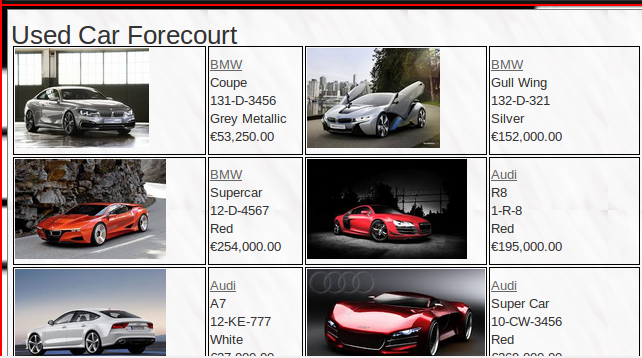
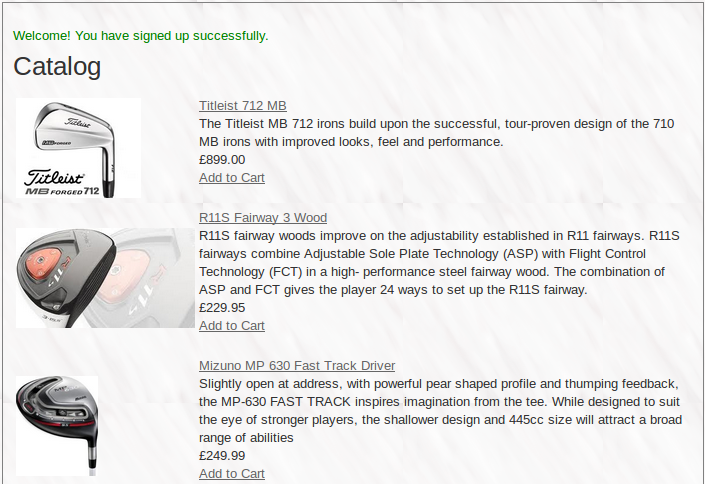
Rails is a combination of HTML and Ruby, unlike the .NET framework in which each “Front End” or “View” page has it’s corresponding “Code Behind” page where all the logic resides. The “View” in Rails can combine HTML and Ruby code on the same page.

Equally the Model, View, Controller architecture is further emphasized with separate folders for the Model, View and Controller code files. And once a “Scaffold” is created for a given Model or Table within the database, each folder is populated with the relevant template files relating to that Model/Table.

# Implementation of Features

Although the Golf Store exercise was used as a template upon which to build this project, it has to be said that a number of alterations and changes were made which entailed researching various potential solutions to achieve the desired effect.

The most significant of these were the image upload feature which was added to the crud functionality of the Vehicles/new template and is dealt with in the items below. Added to this was the formatting of the Vehicles page, which shows all vehicles in stock, to dynamically display the vehicles in a grid 2 columns wide, as opposed to the single column format in the Golf Store application, as illustrated below.

Showroom Stock List 1 Golf Catalogue 1

## Allow a user to upload the details of a vehicle including a photo.

Whilst the initial part of this feature was quite simple and only involved the construction of a scaffold to build the necessary form to enable the relevant input, the most significant element, the photo, proved to be a little less straightforward!

After a fair amount of searching, I discovered a wonderful Gem called ‘carrierwave’ and along with the gem and the help of a lovely man, who sounds like Ned Flanders, and his fabulous RailsCast on using the ‘carrierwave’ gem to enable a user to upload a photo, I managed to get that little, but nonetheless critical, feature working.

## Allow a user to Login/Logout & only allow logged in users to purchase.

This was achieved using the “Devise” gem and its associated features and functionality.

## Allow a user to tick various options using the Decorator Pattern.

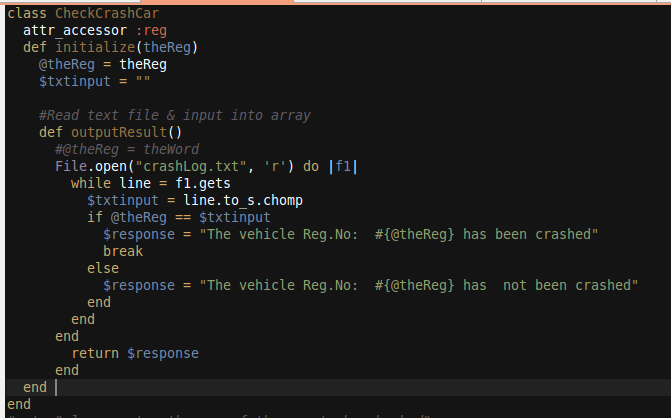
Having completed the Decorator Pattern, it became obvious how this Design Pattern could be utilised in an application such as the DeeSee Car Lot. It also illustrated, to me, a definite benefit that the Rails Framework has over .NET. This was due to the fact that I’d implemented a similar feature in a previous project using .NET and it was a far more complex affair than the Rails implementation.

## Allow a user to log if a vehicle had been crashed using the Singleton Pattern

By modifying the Logger.rb class to accept the vehicle reg as its input, I used the Singleton Pattern to call the crashLog method when the “Car Crashed” check box is ticked and log the reg of the vehicle being uploaded to the Crash Log text file.

## Allow the system to check the “Crash Log” and display a response

By completing and understanding the “Analyser” exercise, I modified the code and created a **custom gem** to read the “crashLog.txt” file, and instead of checking each line for spaces, words etc,, it simply checks each line for a match with the reg of the car being viewed.



## Allow a user to search the database by certain criteria and display matching results.

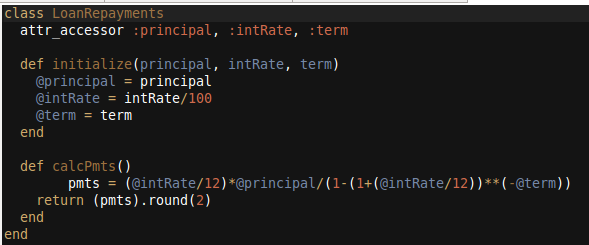
Functionality was built to include the necessary functionality to enable users to search the “Vehicles” model and display the returned results.

## Allow a user to select a Vehicle to view its full details and then allow the user to purchase this Vehicle if he wishes.

The Golf Shop Application was used as a template for this feature, however code was modified to alter the layout.

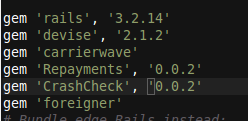
## Enable the system to show details of a purchased Vehicle including finance options.

A **custom gem** called “LoanRepayments” was built to take the Price of the vehicle as a parameter, along with an Interest Rate and a Tem and then calculate the corresponding monthly repayment.



# Conclusions

In all five external gems were used in the development of this project, two of which were custom built.

Devise was used to implement the Login/Logout functionality.

Carrierwave provided excellent functionality enabling the downloading/uploading of image files.

Foreigner was installed to allow me to add a foreign key to an existing table.

Both Repayments and CrashCheck were custom built gems and were essential to the successful implementation of the requirements, as listed in the executive summary.

In conclusion, I have revised my opinion of the Ruby on Rails framework several times during the course of the module and this opinion has swung like a pendulum swinging through the full spectrum from Positive to Negative to Not Bad, to Extremely Negative and finally to Very Positive.

Its plus points (speed of development, pre-construction of necessary MVC templates and files, amongst others) certainly outweigh it’s bad points (hiding of said files in obscure places, tendency to find errors randomly and then find none from the same blocks of code). Needless to say the previous comment has a “Tongue-in-Cheek” quality to it.

What appealed to me most about Ruby on Rails was the simplicity by which a developer can add functionality to his/her application through the inclusion of a gem which has been developed to achieve just the desired result. Alternatively, if the developer needs to build his/her own gem to achieve something, he can then share this for others to use.

Having said that, if I were to be asked to choose between .NET and Rails at this point in time, I would choose .NET. I would make this choice purely on the basis that I have spent more time working with .NET and I find that having direct access to the database, and being able to manipulate and modify foreign key relationships directly inside the database, is better from my point of view.

Although in all fairness, I feel if I were to spend more time using Rails on a day to day basis, I would become more comfortable with its intricacies and would, more than likely, be able to take better advantage of its various capabilities and the benefits the framework, as a whole, has to offer.