Object Mapper

James Nicholson

BSc. Software Design 2018

Athlone Institute of Technology

# Declaration

I hereby certify that the material, which is submitted in this report towards the award of BSc. Software Design, is entirely my own work and has not been submitted for any academic assessment other than part fulfilment of the above-named award.

Future students may use the material contained in this report provided that the source is acknowledged in full.

Signed…………………………………………….

Date………………………………………………

# Abstract

# Acknowledgements

##### 

# Table of Contents

[Declaration ii](#_Toc508299632)

[Abstract iii](#_Toc508299633)

[Acknowledgements iv](#_Toc508299634)

[Table of Contents v](#_Toc508299635)

[List of Tables vii](#_Toc508299636)

[List of Figures vii](#_Toc508299637)

[Chapter 1: Introduction 1](#_Toc508299638)

[1.1 Introduction 1](#_Toc508299639)

[1.2 Context & Rationale 1](#_Toc508299640)

[1.3 Proposal 1](#_Toc508299641)

[1.4 Proposed Technologies 3](#_Toc508299642)

[1.5 Research Aims and Objectives 3](#_Toc508299643)

[Chapter 2: Background Research 4](#_Toc508299644)

[2.1 Introduction 4](#_Toc508299645)

[2.2 Feasibility 4](#_Toc508299646)

[2.3 Pre-existing Software 4](#_Toc508299647)

[2.3.1 Map Marker 4](#_Toc508299648)

[2.4 Technology 5](#_Toc508299649)

[2.4.1 Mapping API 5](#_Toc508299650)

[2.4.2 PHP Framework 6](#_Toc508299651)

[2.4.3 Database Language 7](#_Toc508299652)

[2.4.4 Mobile Application 7](#_Toc508299653)

[Chapter 3: System Design 10](#_Toc508299654)

[3.1 Introduction 10](#_Toc508299655)

[3.2 Requirements 10](#_Toc508299656)

[3.2.1 Use Cases 10](#_Toc508299657)

[3.2.1.1 Plotting & Saving 10](#_Toc508299658)

[3.2.1.2 Salesperson & Customer 11](#_Toc508299659)

[3.2.2 Functional Requirements 12](#_Toc508299660)

[3.2.3 Non-Functional Requirements 14](#_Toc508299661)

[3.3 Architecture 15](#_Toc508299662)

[3.3.1 Model View Controller 15](#_Toc508299663)

[3.3.2 MySQL Database 16](#_Toc508299664)

[3.4 Design 16](#_Toc508299665)

[3.4.1 Frontend Design 16](#_Toc508299666)

[3.4.2 Backend Design 17](#_Toc508299667)

[3.5 Implementation 18](#_Toc508299668)

[3.5.1 Laravel & MySQL 18](#_Toc508299669)

[3.5.2 Google Maps JavaScript API 19](#_Toc508299670)

[3.5.3 JSON Objects & Ajax 19](#_Toc508299671)

[Chapter 4: Testing and Evaluation 23](#_Toc508299672)

[4.1 Introduction 23](#_Toc508299673)

[4.2 Testing 23](#_Toc508299674)

[4.2.1 Unit Testing 23](#_Toc508299675)

[4.2.2 Integration Testing 23](#_Toc508299676)

[4.2.3 Functional Testing 23](#_Toc508299677)

[4.2.4 Usability Testing 24](#_Toc508299678)

[4.2.5 Regression Testing 24](#_Toc508299679)

[4.3 Evaluation 24](#_Toc508299680)

[Chapter 5: Conclusions 25](#_Toc508299681)

[5.1 Introduction 25](#_Toc508299682)

[5.2 Reflection 25](#_Toc508299683)

[5.3 Recommendations 25](#_Toc508299684)

[References 26](#_Toc508299685)

[Glossary 27](#_Toc508299686)

[List of Abbreviations 28](#_Toc508299687)

[Appendix Title uses ‘Heading 1’ 29](#_Toc508299688)

[Appendix sub-title uses ‘Heading 7’ 29](#_Toc508299689)

# List of Tables

[Table 1.4.1: Proposed Technologies to be Utilized 3](#_Toc508299690)

# List of Figures

[Figure 2.3.1: ‘Map Marker’ Main View 5](#_Toc508299691)

[Figure 2.3.2: ‘Map Marker’ Menus 5](#_Toc508299692)

[Figure 2.3.3: ‘Map Marker’ Edit Info & GPS 5](#_Toc508299693)

[Figure 3.3.1: Model View Controller Diagram 15](#_Toc508299694)

[Figure 3.4.1: Marker Table Schema 18](#_Toc508299695)

[Figure 3.5.1: Converting to JSON 19](#_Toc508299696)

[Figure 3.5.2: Ajax Routing 20](#_Toc508299697)

[Figure 3.5.3: JSON Parse 20](#_Toc508299698)

[Figure 3.5.4: Returning JSON Output in Console 21](#_Toc508299699)

[Figure 3.5.5: Object Plotting 22](#_Toc508299700)

# Introduction

## Introduction

This thesis document titled ‘Object Mapper’ describes in detail topics such as background, purpose, research and development and other areas such as difficulties that have been encountered throughout the development process of the Object Mapper application.

For this project, it was decided that a mapping application using Google Maps would be created. From a regular user’s point of view, objects cannot be placed on a map and saved. This project aims to prove that a web based application can be created which allows a user to place objects in the form of markers and then be saved.

The intent for the finished application is for it to be used in an industry setting. For example, companies within the telecoms industry may wish to create markers to represent their infrastructure on a map. Companies within the automotive industry, particularly vehicle dealerships, may want to locate vehicles within a large compound.

Taking the various environments into consideration in which this application can be applied to, It was decided that the application would be based on a vehicle dealership setting as a proof of concept. From here on in, the application will be contextualised with regards to a vehicle dealership experiencing difficulties in locating vehicles on a large premise.

## Context & Rationale

The idea of this application stems from a common problem experienced in most large-scale vehicle dealerships. In most of these dealerships, their premises are attached to large compounds which consists of many parking areas for vehicles. However, these scenarios are, vehicle are moved on a regular basis whether it be due to test drives, movement to different spaces or being transferred from different departments such as service or valeting within the dealership. This constant moving of vehicles can create a problem when it comes trying to locate a specific vehicle particularly for sales staff when interacting with customers.

## Proposal

The aim of this project is to create a solution for this problem that dealerships encounter. It consists of a web and mobile application using various technologies including but not limited to a PHP Framework, JavaScript and the Google Maps JavaScript AP. A combination of these technologies will give a user the ability to plot points on a map of their compound which represent individual vehicles. The map represents the compound. Each plotted point can be updated with details such as manufacturer, model, year and registration. The user can search for specific vehicles and be shown their location and perform other data manipulation events such as edit and delete.

Key features:

* Plot marker to represent vehicles’ positions on a map.
* Insert details about each vehicle.
* Search for a group of vehicles by manufacturer and model name.
* Search for a specific vehicle and display it on the map.
* Read details about each vehicle.
* Edit details for each vehicle.
* Update vehicle latitude and longitude position by its draggable marker.
* Delete vehicle (marker) from the map.

## Proposed Technologies

Proposed Technologies to be Utilized

|  |  |
| --- | --- |
| Google Maps JavaScript API | Laravel Framework |
| JavaScript | Android SDK |
| SQL | JSON |
| PHP | Android |
| JQuery | Ajax |

## Research Aims and Objectives

The research carried out is focused on finding out which technologies should be used such as frameworks, programming languages, which mapping API to be utilized and whether a functioning Android application is feasible within the given timeframe.

Each technology was chosen based on many factors such as current experience, the learning curve and most importantly there needed to be an acceptable amount of well-written and well-structured documentation.

Once comfortable with the chosen technologies, producing coding examples and small prototypes could be put together. If issues presented themselves a decision could then be made to carry out further research to improve the conceptualization of the application.

# Background Research

## Introduction

Research played a significant role in the project both prior to and during the development of it. It was essentially the determining factor of whether or not the application could be feasible and applied to an industry setting. It was always important to realise that the research would carry on throughout the development of the application.

## Feasibility

Initially a lot of time was spent crawling the web to see if a similar web application existed where it could be applied a specific industry as mentioned previously. While carrying out the initial research, it was evident there was no pre-existing application that would a user to plot objects, in this case, plot vehicles on a map and save them. However, there are applications which provide live tracking of vehicles using GPS which is a feature that will not be incorporated.

After the initial research, the project could progress onto the next stage of research which was to initiate dialogue with an individual within automotive industry. This point in time provided an excellent opportunity to get feedback regarding the feasibility; *“…would an internally based vehicle mapping application be able to assist in the day to day operations of a large-scale vehicle dealership?”* was one question that was asked*.* After learning that the application could be applied in a dealership setting, the next stage of research proceeded onto researching what technologies are available and which could be utilized efficiently while paying close attention to the developer documentation.

## Pre-existing Software

It was important to know if there were any existing applications available on the web or on the Google Play Store. As mentioned previously, there is currently no web based application that provides similar functionality as mentioned in section 1.3.

### Map Marker

During development, It was discovered that an Android application named ‘Map Marker’ (theandroidseb, 2018) was recently developed. This app has implemented within it most of the features mentioned in section 1.3. With that in mind, the application was then analysed and tested to see how the Object Mapper could improve on it.

|  |  |  |
| --- | --- | --- |
| ‘Map Marker’ Main View | ‘Map Marker’ Menus | ‘Map Marker’ Edit Info & GPS |

The app itself does not a provide a context for how it could be utilized within an industry setting based on initial observations. One such observation is that the menus and sub menus which can be seen in Figure 2.3.2: are not necessary. It is possible the application can be improved on by having interactive info windows on the map; like the web based application without the need for a list of menus and on-screen buttons. Most importantly, by improving on it in this way, it may be possible to apply it to an industry setting with easy to plot markers, refined access to information and integrated search field(s).

## Technology

The bulk of the background research was done in relation into which technologies would work best. It was crucial to understand if and how the certain technologies could interact with each other to develop a fully functioning application.

### Mapping API

Choosing the right mapping API was important. It needed to be user and developer friendly. Having no experience using a mapping API, it was important to take the quality of the developer documentation and community support into consideration. The two main APIs that had been narrowed down two were Foursquare and Google Maps.

Foursquare (Foursqaure, 2018) already provides GPS tracking for ride sharing company Uber. Therefore, at the time it could be assumed that by using Foursquare there would be a good starting point when it comes to development. However, that was not the case. Upon further research it was clear that the community and most importantly the documentation would not suffice. The community on the popular developer oriented website Stack Overflow was nearly non-existent. Most of the Foursquare related questions went unanswered which gave a bad impression on the API. The developer documentation was far from acceptable. Upon signing up as a developer and examining the documentation it became clear that any code related examples on how to implement any features were nowhere to be found and for the short examples that were there, the descriptions for each were vague and did not offer any help as to what the code was doing (Foursqare, 2018). It became clear that due to a combination of the inadequate documentation and community support a decision was made had to drop the idea of incorporating Foursqaure’s Map API.

The Google Maps API (Google, 2018) was my latter option. Like Foursqaure, Google provides mapping functionality for on-demand transportation companies such as Lyft. This also gave the assumption that there may be a good starting point with previous examples however, that assumption turned out to be incorrect. Albeit in contrast, the Google Maps API community and the documentation was excellent. Most of the coding related questions were given solutions to on Stack Overflow. The code examples within the developer documentation was abundant with well written descriptions of what the samples do (Google, 2018). Due to an abundance of developer related information, the decision was made to use the Google Maps JavaScript API. This API will help provide and incorporate the needed functionality with the chosen framework.

### PHP Framework

Choosing a framework was not as cumbersome in comparison to choosing a mapping API. The Laravel PHP framework had already been in mind from the beginning. Having worked with Laravel before albeit the 5.0 version, the version being used, Laravel 5.5 introduced some minor changes but nothing that of immediate importance. The file structure, the database and work environment set up and the package dependencies such as Eloquent ORM remain familiar (Laravel, 2018). Eloquent ORM proves very useful as it helps shorten down code length and simplifies things in general for PHP.

*“The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.”* (Laravel, 2018)

The documentation supplied for Laravel is superb with easy to read code and descriptions. There is also a very active community base on the Laravel forums, their developer podcast ‘Laracasts’ and on Stack Overflow. Amongst the developer community, Laravel is considered to be very developer friendly thus making it an easy choice to go with it.

### Database Language

Like with the framework that had been chosen, the database to go with didn’t need much consideration. MySQL was chosen, the reasoning being; having adequate experience in using it, provides high performance and works very well alongside PHP to which is ideal for web programming.

### Mobile Application

The biggest feasibility and technical concern was to produce a complete and working android application to coincide with the development of the main web application as Android development was certainly not a strong point. It made sense to develop an Android application as a salesperson may wish obtains vehicle locations and information on-the-go.

Several different techniques of reproducing a working application on Android based on the web version had to be analysed. Some of the options turned out to be more complex than I had originally thought.

There are three options I considered in this regard:

1. Create the Android application separately to the web applications
2. Use a hybrid framework such as Ionic or PhoneGap which both incorporate Apache Cordova.
3. Run the web application in an Android WebView whilst connected to the server.

Naturally, the trio of options would introduce their own set of advantages and disadvantages.

Creating the applications separately would allow for greater control over the app; fewer technologies would have to be implemented. However, it may prove time consuming getting to understand the Android SDK and its requirements for implementing the core functionality of the application.

Using a hybrid framework which incorporates Apache Cordova essentially allows for the work to be carried out on one application with a wrapper surrounding it. This wrapper allows the application to be ported over and work on a different platform albeit using Angular thus introducing another learning curve. Technologies ported from the web application may not be compatible and other solutions will need to be developed.

The idea of running the application in an Android WebView whilst connected to the server only came at a much later stage into development. It allows for all the exact same functionality that’s in the web app but scaled down to fit on a smaller screen. It can also be configured to be loaded from within in an Android app. The only changes needed were to adjust Ajax URL’s and directory routes to suit the Apache server the application will be running on.

Deciding on which option to proceed with would rely on further research and investigation and experimentation in terms of developing prototypes. One research objective that I set during this stage was to understand if there could be any solutions that are applicable to the problems I would encounter.

However, upon carrying out further reading and coding examples, the wrapper allows for compatibility within a web browser on Android, not allowing for a native app which can be accessed easily on the Android home screen. The most suitable solution to go with was to run the application in a WebView

Working with Google’s first party software, Android Studio introduced its own issues which had to be adapted to. Using the Android Studio and an emulator were quite resource intensive on the hardware that was available to me thus making it difficult to be productive at certain time due to system slowdowns and crashes.

# System Design

## Introduction

In this chapter, use cases are defined, functional and non-functional user requirements that and end user would need are discussed. The process involved in developing the application is also discussed and how certain technologies were implemented along with issues that may have occurred and how there were rectified.

## Requirements

The use cases and requirements outlined in this section represent both applications.

### Use Cases

#### Plotting & Saving

This use case describes how a salesperson will interact with the application to store a vehicle and have its position and details displayed on the map.

Actors

* Dealership Salesperson

Preconditions

* The device has the system requirements capable of running the Object Mapper.
* The dealership has the details of vehicle.

Basic Flow of Events

1. The use case begins when the salesperson plots a marker on the map.
2. The sales person then clicks the plotted marker.
3. An info window then appears where the salesperson can input information regarding that vehicle marker.
4. The salesperson clicks the save button where the vehicle’s details and position are stored in a database.
5. The use case ends successfully.

Alternative Flows

* Duplicate Vehicle: If a user tries to enter a vehicle with the same registration number or VIN, an error message saying, “Duplicate vehicle: Check reg/VIN” will be displayed.

Post-Conditions

* Successful Query: The user can view the plotted vehcile marker on the map.
* Failure Condition: The database has been updated accordingly.

#### Salesperson & Customer

This use case describes how the salesperson will interact with a customer by using the Object Mapper.

Actors

* Dealership Salesperson
* Dealership Customer

Preconditions

* The device has the system requirements capable of running the Object Mapper.
* The dealership has vehicles stored and displayed on map.

Basic Flow of Events

1. The use case begins when the customer enquiries about a vehicle.
2. The salesperson wishes to show the customer the vehicle.
3. The salesperson is unsure of where the vehicle is located on the premises.
4. The salesperson interacts with the map to locate the vehicle by cross checking registration numbers or VIN.
5. The salesperson successfully locates the vehicle and proceeds to walk to it along with the customer.
6. The use case ends successfully.

Alternative Flows

* Vehcile Not Found: If the salesperson searches for a vehicle that is not stored in the system, an error message saying, “Vehicle not found” will be displayed.

Post-Conditions

* Successful Query: The salesperson can view a vehicle on the map including its position as well as its details which can be communicated to with the customer.
* Failure Condition: The database does not hold the vehicle the salesperson was looking for.

### Functional Requirements

REQ-1: #1000 Plot Vehicle Marker

Description: The user shall interact with and click on a position on the map to place a marker.

Dependencies: None

REQ-2: #1001 Open Vehicle Marker Info Window

Description: The user shall click the plotted marker when they will be presented with an info window popup.

Dependencies: #1000 Plot Vehicle Marker

REQ-3 #1002 Insert Marker Details

Description: Upon opening of the info window, the user shall insert details regarding the vehicle marker they have placed using dropdowns and text input boxes.

Dependencies: #1001 Open Vehicle Marker Info Window

REQ-4 #1003 Save Vehicle Marker

Description: The user shall click on the save button where the details of the vehicle marker will be stored in a database.

Dependencies: #1002 Insert Marker Details

REQ-5 #1004 Display Vehicle Marker(s)

Description: The user wishes to have the saved marker(s) displayed back to them on the map.

Dependencies: #1003 Save Vehicle Marker

REQ-6 #1005 Read Vehicle Marker Info Window

Description: The user wishes to read information regarding a displayed marker. The user shall click on a marker to have an info window displayed with details regarding that marker.

Dependencies: #1003 Display Vehicle Marker(s)

REQ-7 #1006 Edit Vehicle Marker

Description: The user shall click on a displayed marker where an info window will be displayed with an edit option. The user clicks edit.

Dependencies: #1003 Save Vehicle Marker, #1004 Display Vehicle Marker(s), #1005 Read Vehicle Marker Info Window

REQ-8 #1007 Edit Vehicle Marker Details

Description: The user will be presented with input fields where text information can be changed.

Dependencies: #1006 Edit Vehicle Marker

REQ-9 #1008 Edit Vehicle Marker Position (latitude & longitude)

Description: While editing, the user shall reposition the marker by dragging it.

Dependencies: #1006 Edit Vehicle Marker, #1007 Edit Vehicle Marker Details

REQ-10 #1009 Save Edited Details

Description: When the user is finished with editing details and or position, the user shall click the save button where the updated information will be updated in the database.

Dependencies: #1006 Edit Vehicle Marker, #1007 Edit Vehicle Marker Details, #1008 Edit Vehicle Marker Position (latitude & longitude)

REQ-11 #1010 Delete Vehicle Marker

Description: The user may wish to remove a vehicle marker from the map. The user will open a marker info window, where they can then click the delete button. The deleted marker will then be removed from the database and the map will be updated to reflect the changes.

Dependencies: #1004 Display Vehicle Marker(s), #1005 Read Vehicle Marker Info Window

REQ-12 #1011 Search Vehicles by Manufacturer

Description: For example, the user may wish to search for all Audi vehicles and have only Audi vehicles displayed on the map. To do so, the user shall choose which manufacturer to display via a dropdown menu and the click the search button.

Dependencies: #1004 Display Vehicle Marker(s)

REQ-13 #1012 Clear Search

Description: The user wishes to clear a search and have all vehicles markers displayed on the map again. After a search function has taken place, the user will be presented with a clear search button. The user will click this button in order to have all marker displayed on the map.

Dependencies: #1004 Display Vehicle Marker(s), #1011 Search Vehicles by Manufacturer

REQ-14 #1013 Vehicle Information List with Links

Description: A list with all the stored vehicles will be presented to the user which can be clicked. The list will provide details such as manufacture, model, registration and timestamps.

Dependencies: #1003 Save Vehicle Marker,

REQ-15 #1014 Display Individual Vehicle Marker

Description: The user shall click a specific vehicle’s link in the vehicle information list. The vehicle marker only will be displayed on the map while all other have been removed.

Dependencies: #1013 Vehicle Information List with Links

REQ-16 #1015 Focus on Individual Vehicle Marker

Description: When the user has clicked on the vehicle link from the vehicle information list, the map will focus on the position of the chosen vehicle.

Dependencies: #1013 Vehicles Information List with Links, #1014 Display Individual Vehicle Marker

REQ-17 #1016 Clear Individual Search

Description: After displaying an individual vehicle marker, the user will be presented with a clear search button. When the button is clicked, all vehicle marker will reappear on the map again.

Dependencies: #1014 Individual Vehicle Marker

### Non-Functional Requirements

REQ-18 #2000 Required Fields

Description: If a user tries to submit a form with empty fields, a “Required” alert will appear beside the empty fields to alert the user to fill in the required fields.

Dependencies: #1002 Insert Marker Details

REQ-19 #2001 Back-End Error Alerts - Add

Description: If the user tries to add a vehicle marker but an error occurs in the application’s back-end code, an error flash message will appear stating “Unable to add vehicle marker. Please try again.”.

Dependencies: #1003 Save Vehicle Marker

REQ-20 #2002 Back-End Error Alerts - Edit

Description: If the user tries to edit a vehicle marker but an error occurs in the application’s back-end code, an error flash message will appear stating “Unable to edit vehicle marker. Please try again.”.

Dependencies: #1009 Save Edited Details

REQ-21 #2003 Back-End Error Alerts - Delete

Description: If the user tries to delete a vehicle marker but an error occurs in the application’s back-end code, an error flash message will appear stating “Unable to delete vehicle marker. Please try again.”.

Dependencies: #1010 Delete Vehicle Marker

REQ-22 #2004 Timestamps

Description: The user can shall be informed of when a vehicle marker was created and last updated at in the vehicles list in the form of timestamps.

Dependencies: #1013 Vehicle Information List with Links

REQ-23 #2005 Close Info Window

Description: When a vehicle marker info window is open, the user shall click a close button located in the info window to close it.

Dependencies: #1001 Open Vehicle Marker Info Window

REQ-24 #2006 Scrollable Vehicles List

Description: The vehicles list will be scrollable when a user interacts with it.

Dependencies: #1013 Vehicle Information List with Links

REQ-25 #2007 Info Window & Map Bounds

Description: When an info window is opened, the map’s positioning will change to ensure the marker and the information within the info window are clearly visible and centered.

Dependencies: #1005 Read Vehicle Marker Info Window

REQ-25 #2007 Session Error

Description: If a session error occurs, the user will be redirected to a page which will give them the option to return to the Dashboard page via a link.

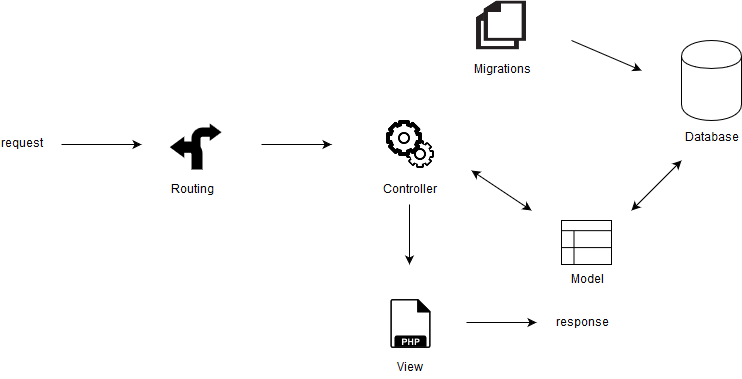
Dependencies: None

## Architecture

This section will discuss the architectural pattern of the system and the database structure.

### Model View Controller

The architecture behind the application relies on the Model View Controller(MCV) architectural pattern which is built into the Laravel PHP Framework. The architecture allows for solutions to be created while being easily adapted to the application’s specifications.



Model View Controller Diagram

The *model* is an essential component for the pattern. It manages all the data, system rules, logic to update changes and objects that is passed through the system. The *view* contains information from the model and then visualized in the form of chats, diagrams and text for the user to interpret. The *controller* works in between the model and the view. It updates the displayed model and view information when data has changed.

The way in which the Object Mapper works can be seen in Figure 3.3.1. A user makes a request which is routed to the controller and then to the specified model. From thereon, information regarding the model is then retrieved from the database, passed back through the controller and then displayed on a view and visualized for the user to interpret.

### MySQL Database

MySQL was an ideal database programming language to use. It’s easy to work with and is supported very well within the Laravel framework.

## Design

In this section, design aspects are discussed in terms of frontend and backend and how these aspects benefit the end user and the developer.

### Frontend Design

Design is important when it comes to user experience. The user must be aware of what is happening on the screen and be informed if something is wrong and not be overwhelmed by too many features on the screen. The design was kept simple by having one map and one list of vehicles. The user is not overwhelmed with information, input fields and buttons. It sticks to a simple white layout with blue accents on links and icons from the font-awesome icon library.

What the user sees is put together by using Laravel’s blade system. Part of the page such as the header, and side menu are coded in separated blade files. Each section can be implemented by using @section('body') and @extends('layouts.sidenav') lines of code. This method of coding is useful as it allows code to be separated and accessed and modified more efficiently. Thus, the page that is compiled consists of significantly less code with the benefit of not being convoluted; which is extremely helpful for the developer.

The frontend has one core file for handling the visualization of data. Map.js handles the code needed for initializing the map, displaying markers and other information. The file utilizes Ajax for communicating with the background in JSON format. JQuery is incorporated to handle modification of the DOM elements on the client side.

### Backend Design

The backend design is based off the model view controller design pattern as mentioned in section 3.3.1. The web application’s backend is written in PHP and utilizing some of Laravel’s included packages including Eloquent.

There are three core files which allow the application to function are

1. Marker.php – serves as the marker Eloquent model. Within the file, attributres that are mass assigned are defined and attributes that should be hidden for arrays are defined.
2. MarkerController.php – is the main controller for handling all data regarding markers. This controller handles the functionality for communicating with the database. Within it are functions for storing, editing, deleting and pulling data in JSON format. The controller also includes code specific to Laravel’s framework for handling the creation and updating of timestamps.
3. Web.php – routes all information towards the specified controller. Included in the file are other defined routes ranging from login routes and link routes to marker routes which can carry variables defined by the user’s their input.

The MySQL database tables are created using the included database schema builder. The marker table attributes are defined within the same file.



Marker Table Schema

Figure 3.4.1 shows the makeup of the marker table. The php artisan migrate command can be executed where the table will then be created. On the client side, the id and latitude and longitude values will not be visible to the user.

## Implementation

The way in which technical features were implemented and how they work with each other proved challenging at times. With the use of creative solutions, these challenges could be overcome. How certain solutions were implemented from start to finish and how certain problems were overcome will be mentioned in the following subsections.

### Laravel & MySQL

The first thing on the list to do was set up a development environment followed by the creation of a base Laravel framework using Composer. Referring to documentation, this was done with ease.

Following initial set up of the framework, database configuration files could then be modified so Laravel could can communicate with the MySQL database. In Figure 3.3.1 migrations can be seen connected to the database. Migrations are not core to the MVC architectural pattern, but it is a feature unique to Laravel. Migrations allow for database schemas to be easily modified and shared if needed. Developers can quickly access the way in which tables are created and modify them.

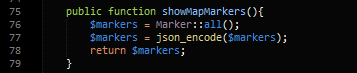
When migrations were completed, the marker controller could then be created. Within the controller are functions to handle the creation, editing, deleting and reading of markers. In it are also functions to manage the timestamps of markers stored in the database.

### Google Maps JavaScript API

The API was implemented with the aid of documentation. The first step is to obtain an API key from the Google Developers console. This key is inserted into an async defer script source tag in the HTML. Async defer will allow HTMl elements to load first and only then will the map load. The map is initialized and modified through the map.js file. This file also handles the plotting of markers and other features.

### JSON Objects & Ajax

The first major problem to overcome was to figure out a way of how markers could be read from the database and be displayed on the map as JSON objects. The first step to solving this was to create a function in the backend where all markers are returned as JSON objects.



Converting to JSON

Above in Figure 3.5.1 show the solution to converting to marker to JSON. Using Eloquent’s format, all marker records can be returned in a single collection using the all() method and assigned to $markers. With all the marker records now retrieved from the database, they then needed to be encoded in JSON format. That was done by using the json\_encode() method and passing in the collection name as a parameter. Lastly the collection of markers is returned in JSON form.

At this point each marker has now been JSON encoded. The next step was to pass the markers to the map and display them using JavaScript. This was achieved with the help of Ajax. Ajax allows for the retrieval of data from a server and for the sending of data to a server. It also allows for a web page to be updated without the page being reloaded. The way in which Ajax is applied here, it is given a URL pointing to a method in the PHP backend.



Ajax Routing

In figure 3.5.2 above, this part of the Ajax method is executed each time the page is loaded. The URL points to the method shown in figure 3.5.2 where the GET method is used to retrieve the data. Upon successful retrieval of the data, a success method is implemented. Put simply, this method decides on what should happen next to the data. The idea was to hopefully just have the data displayed on the map using Google Map’s marker function. Unfortunately, that was not the case and lead to another issue that needed to be solved even if it did take some time to figure out.

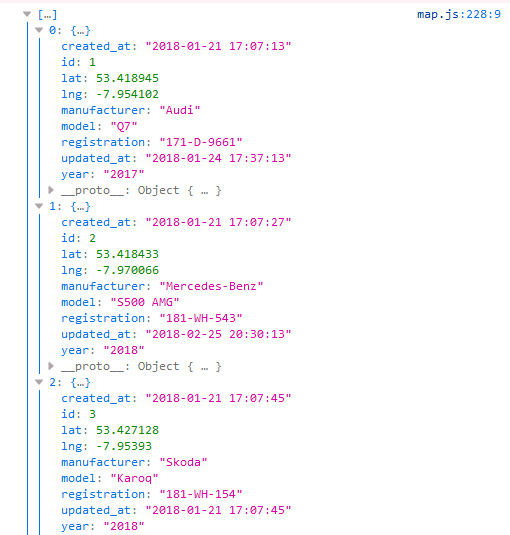
Now that the data has been successfully returned, a solution was needed to correctly read the data in JSON format. It took some to figure out. Eventually it was discovered that the way to read in JSON data was to use the parse function. The following image shows how the parse function is implemented.



JSON Parse

The data that is successfully retrieved from the Ajax route is passed into the success function. It is then passed into a JSON.parse() function which is assigned to a variable named also name data. After the data has been parse, it is finally possible to read the collection of data as JSON.

The scaffolding for Ajax, JSON data and plotting functionality are now working. The role these technologies play in the application are faulty significant and allow for the integration of more features using the same and or similar techniques.

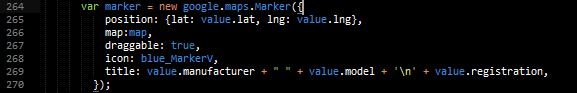


Returning JSON Output in Console

The above image represents the first three objects and their attributes from the collection of data printed out in the console.

The final issue with regards to displaying data came in the form of plotting markers. At this stage it was possible to get one marker to display and that however despite number of records within the collection, that marker would always the most recent entry in the database. To overcome this, with the help of Stack Overflow it was determined the best way to solve this was to split the objects into key-value pairs. In figure 3.5.3, it shows the each() function being utilized to iterate through the data and creating key-value pairs.

Now that a collection of records has been retrieved, JSON encoded, passed into our Ajax function, parsed in JSON format and then finally iterated through and assigned key-pairs, each object can now be accurately represented on the map. This was achieved by using method including in the Google Maps JavaScript API as in the image below.



Object Plotting

Figure 3.5.5 demonstrates how the plotting of objects are handled in the form of markers. Plotting each marker would not be possible if it were not for the implementation of key-value pairs. With this, other information such as manufacturer, model and registration can be access with ease.

# Testing and Evaluation

## Introduction

Testing was imperative to ensure the application was functioning correctly. Several forms of testing were carried out during the development of the Object Mapper including:

1. Unit Testing
2. Integration Testing
3. Functional Testing
4. Usability Testing
5. Regression Testing

Each of the above methods used a white box code and fix approach to testing where code is implemented and if bugs are found, they are fixed afterwards by the developer.

## Testing

### Unit Testing

For each of the functions implemented in the front and back end of the Object Mapper, unit testing was carried out to ensure the application was producing expected output against given input. For example, to ensure the correct marker coordinates were being obtained when adding a new marker, the coordinates were accessed and logged in the debugger console so their values could be seen.

### Integration Testing

Integration testing was applied to components that are combined to produce a desired outcome. In the case of the function that handles the searching of markers, a function that removes markers that are not relevant to the search parameters had to be implemented. This form of testing wasn’t limited to only a couple of components being implemented together, it was also applied system wide when all functions were implemented together.

### Functional Testing

At this stage of testing, the specified functional and non-functional requirements required listed in section 3.2.2 and section 3.2.3 were tested against. All requirements were tested successfully.

### Usability Testing

Usability testing was performed with the user in mind. This stage of testing involves testing the user interface to see if it is user friendly, easy to use and if a user can carry out tasks efficiently by using it. By testing it extensively, it was concluded that the UI is well designed and well implied and would satisfy a user’s needs.

### Regression Testing

During and after the testing methods used mentioned up to this point had regression testing performed in one way or another. Regression testing occurred when a single or a group of components were modified to correct bugs. It ensured that the modifications implemented did not interfere or negatively impact other system components

Regression testing was not confined to being when bugs are fixed, it was also applied when new features were implemented. For example, regression testing was carried out when conditions were implemented for plotting marker icons which involved using different images based on a vehicle’s manufacturer.

## Evaluation

Quite some time was spent on the testing the application both during and post development. This was a positive aspect and not a negative one. It ensured that system components could be implemented and they could work efficiently together.

# Conclusions

## Introduction

## Reflection

## Recommendations

# References

Foursqare, 2018. *Foursquare Developer.* [Online]   
Available at: https://developer.foursquare.com/  
[Accessed 5 November 2017].

Foursqaure, 2018. *Foursquare Enterprise.* [Online]   
Available at: https://enterprise.foursquare.com/  
[Accessed 3 November 2017].

Google, 2018. *Google Developer Documentation.* [Online]   
Available at: https://developers.google.com/maps/documentation/  
[Accessed 5 November 2017].

Google, 2018. *Google Maps Enterprise.* [Online]   
Available at: https://enterprise.google.com/maps/  
[Accessed 6 November 2017].

Laravel, 2018. *Laravel.* [Online]   
Available at: https://laravel.com/  
[Accessed 25 October 2017].

Laravel, 2018. *Laravel Docs.* [Online]   
Available at: https://laravel.com/docs/5.5/eloquent  
[Accessed 26 October 2017].

theandroidseb, 2018. *Map Marker.* [Online]   
Available at: https://play.google.com/store/apps/details?id=com.exlyo.mapmarker&hl=en  
[Accessed 8 February 2018].

# Glossary

# List of Abbreviations

# Appendix Title uses ‘Heading 1’

## Appendix sub-title uses ‘Heading 7’