Hitch a Ride - Online Carpooling Application

Final Report for CS39440 Major Project

Author: Alex Roan (alr16@aber.ac.uk)
Supervisor: Fred Labrosse (ffl@aber.ac.uk)

10th April 2012
Version: 2.0 (Release)

This report was submitted as partial fulfilment of a MEng degree in
Software Engingeering (G601)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature .....

Date .....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature .....

Date .....

Acknowledgements

I am grateful to Aberystwyth University for providing the resources necessary to produce this project.

I'd like to thank Fred Labrosse for his guidance and assistance with the project whenever I required it. I'd also like to thank everyone else whom I have discussed my project with and helped me along the way.

Abstract

In the world we live in today, many people need to travel long distances across the UK. Often, these people do not have access to their own mode of transportation. With public transport prices in the UK rising steadily, making these journeys could be quite costly.

This project aims to bring the drivers of vehicles together with others who have little choice of transportation, to reduce the cost of travelling across the country. It provides a platform in which drivers can share journeys for passengers to hitch in the same or similar directions.

CONTENTS

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Overview

As a student studying in a university away from home, I have often needed to travel back home during the holidays or on weekends. My motivation for this project comes from the requirement that many people have to travel frequently from one place to another.

I have always driven to Cardiff from Aberystwyth and back during the holidays in a car with 4 spare seats. If there was some way I could offer those spare seats to other students or people travelling in the same or similar direction the cost of fuel could be split, meaning a cheaper mode of transport.

With the cost of public transport soaring ever higher [10], travelling around the country is becoming a more expensive task year after year. Not only are bus and train prices increasing, but petrol prices have sky rocketed in the last century [21] as well. This website aims to bring people travelling in similar directions together, to save on the cost of fuel instead of having to pay a full train fair or travelling alone.

A one way train ticket between Aberystwyth and Cardiff costs around £54.80 [19], a journey that in a small car can cost as little as £20 in fuel. If that journey is shared between a driver and a passenger, who would otherwise have no option but to take the train, they would each only need to pay £10 for the journey. The driver would save £10 on their journey, and the passenger would save £44.80.

1.1.2 Example Use Case

Upon opening the home page, the user will be confronted with the option of either registering, or logging in using existing details. Either action done successfully will log the user in.

Once logged in, the user is taken to their home page, which displays predicted journeys that the site thinks the user might be interested in using their preferences. From this page, they can access all features within the site, which include:

- 'My Activity' - Information about upcoming rides and hitches related to the user. The option to accept or decline hitch requests for their journeys.
- 'Messages' - Messages to and from other users.
- 'My Profile' - Personal details about the user, available for editing.
- 'Find a Journey' - A search page used to search for journeys from one location to another.
- 'Post Journey' - The page used to post new journeys that the user will be partaking in.

When a user posts a journey from location A to location B using the 'Post Journey' page, it will display a map route which needs to be accepted by the user. Once this is accepted, the journey will be entered into the database, where it will be returned in other user's searches if the parameters match the details of the journey. These details could be two locations along the journey that are not the origin and destination, but somewhere in between.

If another user requests to hitch a ride on the journey, the driver will be prompted in the 'My Activity' page, and will have the option of accepting or declining the hitch request.

The hitch request could be from the origin to the destination, but it could also be from origin to a new location along the route of the original journey. Similarly, it could be from a location along the route of the journey to the journey destination, or even from one location along the route somewhere to another. It is up to the driver to accept or decline the hitch request, depending on how it changes the route that the driver takes.

If the driver accepts this new hitcher, the route will be altered if necessary to include the new pickup / drop off points and saved into the database with one of the spare spaces now filled.

1.1.3 Existing Services

There are existing websites that offer the same sort of carpooling service. The leading services are Carpooling.com [7] and BlaBlaCar.com [2]. These sites provide a platform for people to offer rides to other people needing to travel in similar directions.

Carpooling.com This is a worldwide service which offers registered users the chance to share rides to other users.

Blablacar.com BlaBlaCar is the largest car sharing network in Europe and rates users on their experience usign the site.

1.2 Analysis

1.2.1 Original Goals

The original goals of the project were to produce an online service that enables people to share journeys with people who are looking to travel to and from similar locations or locations along the route of the driver, this was the key marker as to the success of the project. The service must recognise when two locations in a search lie along the route of an existing journey with different

origin and destination, not just recognise the searched locations as the beginning and end of a journey. In other words, the service must account for picking people up and dropping people off at any point in the journey, should the driver accept the hitcher.

The most important goals of the project are as follows:

1.2.2 Requirements

- Register as a new user to the site
- Log in with existing details
- Edit profile details
- Post a journey that the user will be driving from location A to location B
- Search for journeys between two locations. The search must return journeys in which the searched location appear along some points on the journeys
- Request to hitch a journey as a passenger
- Accept or decline a hitcher as the driver
- Send messages to other users on the site

Ideally, provided a more extensive time period, the requirements would involve more in-depth services. These include the like of integration with social networking sites and the introduction of interoperability such as OAuth [18] to extract data from existing user account to different services. Also, a service like this would benefit from being accessible away from a traditional desktop computer, with a mobile application being developed. Unfortunately, time restrictions and priority on the main purpose of the project meant that these features were omitted from the initial requirements stage.

1.3 Process

1.3.1 Overview

Planning and research was performed before the implementation of this project. Before the development could begin, the method in which to develop needed to be established. There were a few methodologies considered, detailed in the Methodology section of this chapter.

Research into the types of technologies I needed to use was also required, prompting reading into APIs, code libraries and programming languages. These are detailed in the Planning and Research sections of this chapter.

1.3.2 Methodology

Agile methodologies were considered for the development of this project. The decision lied mainly between choosing Feature Driven Development (FDD) along with Test Driven Development (TDD) as the method of coding the site, or using the more structured Waterfall method of development. There were pros and cons to both of these methods.

Feature Driven Development would allow a very flexible environment in which to develop each feature iteratively, without much thought into future architecture. Although this would account for any future changes in specification or problems encountered during the implementation stage, I did not feel as though it would suit the development of the site. This is due to the lack of architectural planning involved in agile methodologies such as FDD. I feel that a project as reliant on its data storage method such as this requires more thought towards the design of such data storage methods. A tweaked version of FDD was still an option, allowing for a plan and design of the database, with the incremental development of each of the features implemented over time using the original database design which would be developed first. This however, defeats the object of agile development because it would not really allow for any future changes due to the architecture being set in stone from the beginning.

The Waterfall methodology requires that specifications and design are compiled well before implementation begins, and I feel that this suited the project more. The architecture of the database, and program controlling data flow to and from it, could be planned in advance and implemented within a short period of time. Although this did not account much for future changes, the architecture could be discussed in detail and designed with planning rather than be developed on the fly, resulting in a more normalised relational database.

1.3.3 Research

The vast majority of this service depends on its ability to recognise, locate and route directions from an origin to a destination. It was clear from the beginning that either a routing algorithm needed to be developed using real map data, or an existing or multiple existing APIs needed to be utilized to produce routing data that could be saved locally to the site's database for processing.

Google Directions API [11] was found to be a valuable asset in the projects development. Using it enables routes to be created from an origin, destination and way points. The result that gets returned can be specified by the parameters parsed to it, for example: JSON, XML, etc. It details the global position of the origin and destination names parsed to it, and the global position of each step taken along the route. This proved to be perfect for the types of requests that this project needs to make.

Another valuable API that was researched was the Google Geocoding API [12]. This API returns the geographical location in latitude and longitude of any place name given as a parameter, useful when searching using the geographical location of places along the route of journeys.

1.3.4 Planning

The planning that was carried out followed requirements specified by the Waterfall development methodology. Initially, a project specification document was developed to outline the requirements of the project. Once they had been established, the design of the project began to develop, resulting

in some basic UML diagrams describing the database structure and website. The Waterfall process deviated slightly from its normally strict processes here where a few prototypes were developed to test out some of the design theories compiled during the design stage. Any issues or flaws in the design were corrected at this stage, enabling for the smoothest implementation possible.

Chapter 2

Design

2.1 Overview

The design chapter outlines the technologies used and a detailed explanation of the overall architecture of the project.

2.2 Technologies

2.2.1 PHP

PHP was used as the main language for the object oriented control of the data flow to and from the database. It is also used as the language to dynamically produce the website output to the user's browser.

2.2.2 JQuery

JQuery is featured on the website as a means of displaying certain features. Graphical maps generated by Google Directions [11] service are retrieved and displayed using JQuery. It is also used to control divs on the site, seen on the 'messages.php' web page.

2.2.3 PostgreSQL Database

Using an object relational database management system is the most efficient method of storing dynamic data for websites. The PSQL database is handled by the object oriented database controllers used by the website.

2.2.4 Github

Github [17] version control web hosting was chosen as the desired method for version control.

2.2.5 Programming Environment

Developing an object oriented application in PHP meant that a suitable IDE was required for the development process. Netbeans IDE 7.3.1 was the selected tool for this task, as its PHP plug-in provides a very efficient interface for PHP development.

2.3 Overall Architecture

2.3.1 Overview

The data storage system the website uses is a PostgreSQL database which contains the following tables:

- Person - Personal details of each user.
- Journey - Details of journeys that users have posted.
- Journey_Step - Each journey has many journey steps. This table holds the geographical location, the related journey and the order of each step.
- Journey_Step_Temp - A temporary table used when the journey steps for a particular journey change. This may occur when a hitch request is accepted which alters the route of the journey. The old steps are moved to this table, so new steps can be inserted into the 'Journey_Step_Temp' table.
- Hitch_Request - Details about a hitch request made from a person relating to a particular journey.
- Message - Messages sent from user to user.

All actions performed on the database are done via a connection from the PHP database controller classes. Each table has a representative PHP model class mirroring the table. Controller classes control these classes to insert, retrieve, update and delete records from each of the tables. The website instantiates the database controller classes to enable the site to produce dynamic output and allow the user to access all of the site's features once logged in.

2.3.2 Other Considered API-Like Structures

The final structure of the database controller that is present in the final system was not originally the first choice for it. Investigative prototypes were developed that completely separated the website from an API that interacted with the database.

The prospect of a procedural API was considered, prompting a basic prototype to be built. It worked well to an extent but because of its procedural nature, it did not follow any sort of design pattern. The method of data transferral between API and website was JSON data, which was sent via HTTP POST to a single API URL. This JSON data would contain a 'request_type' field, which would indicate to the API what kind of operation needed to be performed on the rest of the JSON data. A data encoder method would parse this data, call a specific function which would perform

some kind of action of the JSON data and query the database, then echo JSON data back to the website with the result of the request. This structure worked fairly well, but the development over time slowly became more of a hassle as the entropy of it outgrew the benefits. Any slight change in database structure or table meant changing every function that queried the database, which made the code almost impossible to re-factor or update. It was deemed that an Object Oriented approach was the preferred approach to follow.

If the API was going to separate from the website, it would probably be in the form of a Representation State Transfer architecture (REST). RESTful web services explicitly use HTTP methods to perform all queries on the database. They are usually used for high performance services with many queries being passed to the server from many sources. Implementing a RESTful API was considered in detail but deemed not right for the project because of the scale of it. If there were other methods of access to the system that would be made available, for example mobile applications and other third party websites, REST would be a perfect solution. But because the only service accessing the API is this website which would be based on the same server, there was no need to separate the website from the API. This is how the current structure of database controller classes and website files came to be. The website files are not object oriented and simply maintain a session whilst the user is browsing. The database controller files are object oriented and control data flow to and from the database. The website gives the user an interface which uses database controller classes to control data flow to and from the database.

2.3.3 Structure of Database Controller

The database controller is the name given to the object oriented structure of PHP classes used to control data flow to and from the database. For each of the main tables in the database, there is a corresponding PHP class with the same name: 'Hitch_Request', 'Journey', 'Journey_Step', 'Message' and 'Person'. These classes model their corresponding tables and have attributes matching those of the table fields. They all contain the methods: 'Create()', 'Load()', 'Update()', 'Delete()'; following the CRUD persistent storage technique [9].

The Create method in each of the classes uses the class attribute values to create a new entry in the corresponding database table. The Update methods update the related entry already in the corresponding table using the attributes in the class object, the primary key will have been retrieved from the database upon executing the Create method and stored as one of the attributes. The Load method retrieves the attributes to the class object from the corresponding table in the database using the primary key as a parameter. The Delete method simply deletes the related entry from the corresponding table in the database using its primary key stored in the class attributes. The Delete method also resets all of the attribute values in the object just in case the object is used again by its controller for a different entry in the database. All of these methods dynamically construct SQL queries within them to query the database depending on the current state of the object itself. The return value from the method depends on the success of the query to the database.

Upon instantiating each of the model class objects, a non-compulsory parameter may be parsed to the constructor as the primary key to that table. If something is parsed in this parameter, the constructor will call the Load method, which attempts to populate the class object's attributes with values from the corresponding table using the parameter as the primary key.

Each of the table model classes are utilized by controller classes. These classes instantiate their model classes to manipulate the entries in the database. Each of the controllers are unique

to the tasks that need to be performed on each of the tables and often involve interaction between the controllers. For example, when a new Journey is posted by a user, the 'Journey_Controller' class would be instantiated. This object would then instantiate a 'Journey' class object, populate its attributes with values entered by the user on the website, and use its 'Create()' method to insert the data into the database. It would then instantiate a 'Journey_Step_Controller' class, which in turn, would instantiate and populate a series of 'Journey_Step' classes depending on the number of steps in that journey. Each of the entries would then be inserted into the database by the controller calling the Create method in each of the 'Journey_Step' objects.

A UML class diagram of the controller and model classes can be found in Appendix D.

2.4 Website Design

2.4.1 Overview

The website is produced by the collection of 25 PHP files which dynamically output HTML depending on the data that is received from the database controller classes. They also provide a platform for users to access all of the features available to them that the site offers. Other technologies and libraries are also used in the productions of the dynamic website, including Bootstrap CSS library [5] and JQuery Library [14].

2.4.2 PHP

The PHP files responsible for dynamically outputting the website maintain a session throughout the user's time on the site. This session allows the site to maintain a user logged on as they navigate through the site. The only data that is stored continuously as a session variable is the user's email address, which is used when they log in or register to the site. If there is no session present, or the session variable containing the email is empty, any page that is accessed redirects the browser to the index page.

Once logged on, there are six main pages accessible via the menu bar spanning the top of the web page. These pages are:

Home Clicking this redirects the browser to 'home.php'. This page is basically a splash page which makes 3 journey suggestions that the site thinks the user may be interested in hitching a ride with. It calculates this by instantiating a 'Person' object from the database controller classes, parsing the email address stored in session to the 'Load()' method, and retrieving the details stored in the 'Person' object's attributes. If their profile has not yet been fully completed to enable predictions, the user will be prompted. Otherwise, these details are parsed into a 'Journey_Controller' object which searches for any journeys that may be relevant to them. Each suggested journey has two buttons: 'View Journey Details' and 'Request to Hitch'. The first takes the user to a new page, 'journey_view.php' where more details about the journey are displayed, including a map of the route. The second redirects the browser to 'request_hitch.php' which attempts to make a request on the user's behalf and redirects the user back to 'activity.php', the Activity page.

Activity Clicking the 'Activity' button redirects the browser to 'activity.php'. This page displays all of the current user's shared journeys on the left hand side and hitch requests on the right.

If there has been any change in the status of a journey, it will be highlighted red with a prompt message also in red inside it. This will occur if someone has made a hitch request to it. Journeys are retrieved from the database by the page instantiating a 'Journey_Controller' object and parsing the email address in the session variable into the 'GetMyJourneys()' method, which returns an array of journeys posted via that email address. The hitch requests are obtained in much the same way but instead of using the 'Journey_Controller.php' class, it instantiates a 'Hitch_Request' object and parses the email address into the 'GetMyHitchRequests()' method.

Each journey and each hitch request are click-able if the user wishes to view them in more detail. Clicking on a journey will redirect the browser to 'journey_view.php' which displays more details and a map relating to the route of the journey. If a hitch request is clicked, it redirects the user to 'hitch_view.php' which displays more details about the journey and driver; details of the other accepted hitchers, whether the request has been accepted or not, and a map of the route including the way points that hitchers will be picked up / dropped off at.

Messages Clicking the 'Messages' menu button will redirect the user's browser to 'messages.php'. This page allows the user to see messages sent from other users, messages sent to other users and allow them to send more. It does this by instantiating a 'Message_Controller' object from the database controller classes and parses the email from the session variables into the 'LoadMyMessages()' method to load the messages to the object. It then calls 'GetMessages()' to retrieve them as an array. Similarly, the same process is repeated to retrieve sent messages except instead of parse the email into the 'GetMyMessages()' method, it parsed it into the 'GetSentMessages()' method. When a new message is compiled and submitted by the site, it parses the new data fields into the 'SendMessage()' method inside the 'Message_Controller'. This page uses JQuery 'Hide()' and 'Show()' methods to display each div. These divs include: 'New Message', 'Inbox' and 'Sent Messages'. Clicking on the buttons on the left shows the corresponding div relating to the button and hides the others.

Profile Clicking on the 'Profile' button redirects the the browser to 'profile.php'. This page retrieves the user's data by instantiating a 'Person' object from the database controller classes, then parsing the email address in the session variable into its 'Load()' method and retrieving the attribute values. It then displays all user data on the page. The page provides a button at the bottom of the page captioned 'Edit Profile'. This button redirects the browser to the 'edit_profile.php' page which displays all of the same information as the 'profile.php', but instead of just text, the data is displayed in editable data fields. A submit option is present at the bottom of the page which, when clicked, submits the form data to 'perform_edit_profile.php'. This page uses the 'Person' object in the database controller classes to load the person data using the email address in session, update its attributes with the posted form data, and call 'Update()' in its functions to update the entry in the database. The browser is then redirected back to 'profile.php' to view the new person details.

Find a Journey Clicking 'Find a Journey' on the menu bar will redirect the browser to 'search_journey.php'. This page is a form which asks the user to input the parameters of their search. Once filled in and submitted, the input is posted to the 'perform_journey_search.php' page. This page instantiates a 'journey_controller' object and sets the controller's search attributes to the search parameters posted from the form. The controller's 'SearchJourney()' method is called, which returns an array of search results. These results are displayed on the page, each with two buttons. One gives the option of immediately requesting a hitch, the other

giving the option to view more information about the journey. Clicking the 'Request Hitch' button will redirect the browser to 'request_hitch.php' which attempts to request a hitch for that journey using a 'Hitch_Request _Controller' object, and parsing the email stored in the session and data relating to the journey in question into the 'CreateHitchRequest()' method. the browser is then redirected to the 'activity.php' page. if the user clicks the 'View Journey Details' button, they are redirected to 'hitch_view.php', which uses database controller object 'Hitch_Request_Controller' to retrieve all data relating to the journey in question. There are two buttons present on the page for the user to click at this point: 'Request Hitch' and 'Find Another'. The 'Find Another' button redirects the browser back to the previous search results on the 'perform_journey_search.php' page. The 'Request Hitch' button redirects the browser to 'request_hitch.php' which uses the database controller class 'Hitch_Request_Controller.php' to store the hitch request in the database and then redirects the browser back the 'activity.php' page.

Post a Journey Clicking on the 'Post a Journey' button on the menu bar redirects the browser to 'post_journey.php'. This page is a form, requiring the user to fill in the fields and submit the data using the button at the bottom of the form. This button posts the form data to the 'post_journey_preview.php' page, which using the Google Directions API [11] and JavaScript displays a preview map outlining the route so that the user may check the correct place names have been used. If the place names were not found, the user is redirected back to the form on the 'post_journey.php' page. if the user confirms the route, the submit redirects the browser to 'perform_journey_post.php', which using the database controller object 'Journey_Controller' adds the new journey to the database.

2.4.3 Bootstrap

Bootstrap [5] is a standard open source CSS library which makes the development of structures web pages easy. Bootstrap uses a mathematical based system for measuring divs across a page. It considers that every div on the page, including the base container of the whole page, is made up of 12 identically sized segments covering the width of the div. If a div inside the container is given the class 'col-lg-12', it will cover the entire width of the container. If it is given the class 'col-lg-6' it will cover half the width of the page. If a div given the class 'col-lg-6', is another div given the class 'col-lg-6', it will fill half of the div it lies within; where if it was given class 'col-lg-12' it would fill the whole width of the div it lies within, which could be of class 'col-lg-6' so it would only fill half of the page. Using this method enables easy segmentation and management of divs around the website.

It contains a standard CSS file, with a large online repository of additional CSS files available for download for additional effects. The styles I have used from the online repository in this project are as follows:

'Jumbotron' Jumbotron [4] style was used on 'index.php'. The header across the top from this style is also maintained throughout the website on all pages.

'Dashboard' Dashboard [3] is used as a side menu system. It was used on the 'messages.php' page along with JQuery to separate the Inbox, Sent items and New Message.

'Signin' Signin [6] is used for form control. It is used on every page where data is entered in the style of forms: 'search_journey.php', 'post_journey.php' and 'index.php'.

2.4.4 JavaScript and JQuery Library

JavaScript is generated dynamically by the website PHP files that generate web pages for the user. The route maps that are displayed on the journey pages are generated by using JavaScript and the Google Maps API with JavaScript [16]. Jquery is used on the 'messages.php' page using the functions 'Hide()' and 'Show()' to display the correct div for the user when sending, reading inbox, or viewing sent items from it.

Chapter 3

Implementation

3.1 Overview

There were a few changes in though during the implementation of the project. A number involved the structure of the API / database controller written in PHP.

3.2 PHP

A few languages were considered for this project; these include Ruby, Python and Perl. Perl was the first to be discarded. This was mainly due to its performance and usability when used in an Object Oriented fashion. Ruby with Ruby of Rails provides a very stable platform to develop upon, but does not quite have the flexibility that I intended to yield in the development. The final decision came between Python and PHP. Python is a clean language with very good performance and is easy to use. However, the flexibility that PHP provides when developing web pages, and its good Object Oriented capabilities, meant that I sided with it even though its performance may not be as good as Python.

The implementation of the PHP application began based on the principle that the website files generating the HTML and JavaScript output for the interface would be completely separate from the API / database control type structure that would control all access to the database. Both would be developed in PHP and would use some form, or many forms, of HTTP protocol to transfer between the two. During the implementation stage, prototypes were developed which made use of HTTP Post to post JSON test data across from a mock website to an API which handled the database. Although this worked, it started to seem slightly pointless separating the two so much. If the user interface was a mobile based application, an API would need to be developed in order for the application to access the database. The website however, did not need to be entirely separated from the API, and if it was, it would mean that all interaction would have to be done via HTTP protocols. Instead, while the website files are located separately from the database controller files, the website files do have access to the controller classes so that functions inside the controller classes can be called upon by the website files.

3.3 Google Directions API

Google Directions API was considered the most applicable API for generating the journey steps that were needed to store for each of the journeys. Implementing the use of this API within the application proved to be a bit of a struggle when parsing the output retrieved from the API. But once the layers of arrays were mastered, slotting the desired information into the database tables came quickly after.

The API allows for an origin, a destination and up to 8 way points parsed to it. It then returns a JSON string detailing the global positioning in latitude and longitude of the origin, destination and way points, and an array of route data for each of the routes calculated. The API can be set to only output a single route if necessary. The route data hold an array of every leg of the journey. The legs of the route are split by each of the way points, meaning if no way points are parsed to the API, there will only be one leg in the resulting route data. Each leg holds an array of steps. These steps describe the geographical location and html instructions of each of the main steps on the journey. It is this step data that my service must be able to retrieve and store to compile a full series of journey steps for each journey. I need this step data to perform more complex searches upon journeys.

The step data stored in the 'Journey_Step' table is used during the searching of journeys. If a user searches for a Journey between 'Brecon' and 'Cardiff', the search would not only look through the main 'Journey' table for journeys with that origin and destination data, but it looks through journey step data too. So this search could find a journey from Aberystwyth to Bristol which has step data running through or near 'Brecon' and 'Cardiff'.

The problem with these journey steps comes into play with journeys involving long single-motorway journeys. The API only produces a step when a prompt is issued to the driver along the journey. At 'Brecon', the API may issue a step which issues a direction to 'Go Over the Roundabout' in the middle of the town. The API logs this as a major event, considers it as a step and logs the geographical location in the JSON. If the journey is simply a single motorway from A to B then there won't be any major events along the way, meaning the distance between steps could be tens if not hundreds of miles. I found this out when posting a journey from 'Cardiff' to 'London'. The vast majority of this journey is spent travelling on a single motorway; the M4. When testing the search function, I wanted to hitch a ride from 'Reading', which lies just off the route between 'Cardiff' and 'London', to 'London'. Unfortunately, there were no steps anywhere near 'Reading' on the journey from 'Cardiff' to 'London' because it was a straight motorway drive past 'Reading'. The previous step was located too far to the west, before the motorway, and the next step was located too far to the east, near 'London' for the location of 'Reading' to be picked up in the search.

There are other limitations to using this API also. There is a maximum limit of eight way points that can be parsed through the URL. If a journey was shared by a user with a people carrier of more than four spare seats, they could have many more passenger hitch their journey. If each of the passengers has two way points, meaning that their pickup and drop off points are different to the journey origin and destination, the API will not accept the additional way point parameters. As well as this, all Google APIs have a limitation on an IP address' daily usage allowance to 2,500 requests per day. This is a high number but with a decent size user base, it could very easily cap this minimum, prompting the purchase of the Business license [15] which takes the limit up to one hundred thousand requests per day and allows 23 way points in total per request.

One of the parameters that can be parsed to the Directions API is the ability to request multiple routes. If this parameters is parsed to the API, three routes with their full routing details are returned. Had the project had a larger time scale, I would have developed the project so that the journey steps from all three routes were saved in the 'journey_step' table with a marker separating them. This could enable more potential passenger matches for partial routes. Once a hitch was accepted onto one of the routes, the alternatives would be deleted from the journey step table in favour of the new accepted altered route with the hitcher as a way point.

3.4 Google Geocoding API

The Google Geocoding API [12] was needed for the search function I implemented to work properly. Once the journey and the journey steps had been saved to the database, the more complex search method needed to be developed. Each journey step in the 'Journey_Step table has a latitude and longitude which needed to be utilised to perform searches involving partial journeys.

If a journey from 'Aberystwyth' to 'Bristol' routed through 'Brecon' and 'Cardiff', A search for a journey between 'Brecon' and 'Cardiff' would need to return that journey. The Geocoding API was used to retrieve the geographical location, in the form of latitude and longitude, of the two search locations. These locations could then be used to search not only through the 'Journey' table, which stores the latitude and longitude of the origin and destination, but also through the 'Journey_Step' table, which holds the latitudes and longitudes of journey steps of each step on each journey. This enables the search to retrieve larger journeys of partial routes.

3.5 Database

The aim of the database design was to make the most simple design possible, which would still carry out the desired functions of the site. It was one of the main issues that contributed to the decision of using a traditional development methodology instead of an agile one. A full database diagram can be found in the appendix.

The database structure and relation underwent a slight change from the original idea. Originally, as the link between 'hitch_request' entries and 'journey' entries there was going to be a 'hitch' table, which simply stored the primary keys of both the 'hitch_request' entry and the related 'journey' entry. Instead, the 'hitch_request' table has an attribute called 'hr_jr', which stores the primary key of the entry in the 'journey' table that the hitch request relates to.

3.6 Database Controllers

The database controllers refer to the group of object oriented classes that control data flow to and from the database. Initially, during the prototypes and early planning, this was the API that would be access via HTTP protocols from the website files. There were many deliberations over how this API or database controller would be structured.

At first there were prototypes developed which were entirely procedural. They would accept JSON via HTTP post and deal with it by decoding it and parse the data to the required specific

function, which would encode the result. This worked very well in early testing but began to cause a lot of problems when it came to refactoring or attempting to add new features or edit existing features. Eventually, the development of the current form of interface came to be. In MVC terms, the Model and the Controllers are bunched together, with the website files acting as the View.

3.6.1 Searching and Hitching Journeys

As stated in the requirements, one of the most important aspects of the project was to produce a search which not only returned journeys which matched origin and destinations, but returned journeys which passed through places that were being searched. As explained in the 'Google Directions API' section of this chapter, the Google Directions API was used to store journey steps as well as journeys. This part of the functionality proved to be the most difficult to get right.

The method, in which the search function performs its search, involves compiling a large SQL query to database. It combines five aspects of the search into one set of results. These five aspects are as follows:

Origin and Destination Names The first aspect of the query searches for journeys with the same origin and destination names as the search parameters.

Origin and Destination Proximity The second aspect searches for journeys where the origin and the destination are within twenty kilometres of the search origin and destination respectively, using the latitude and longitude retrieved from the Google Geocoding API.

Origin proximity, Drop Off Location The third aspect searches for journeys where the origin is within twenty kilometres of the searched origin, and searches the resulting journey steps of that journey for a journey step within twenty kilometres of the searched destination.

Destination proximity, Pick Up Location The fourth aspect searches for journeys where the destination is located within twenty kilometres from the searched destination location, and the resulting journey steps are searched to find a journey step which is within twenty kilometres of the searched origin.

Journey Steps The fifth aspect searches through the journey steps of each of the journeys to find a route which has steps within twenty kilometres of both the searched origin and destination in the correct order.

The search function compiles an SQL query string with these five aspects as separate "select" statements, and uses "union" to accumulate all of the results. As explained in the 'Google Geocoding API' section of this chapter, the latitude and longitude of the place names in the search was retrieved from the Geocoding API and inserted into the 'Journey' table for the origin and destination positions. Using the Google Directions API each journey step geographical location was into the 'Journey_Step' table.

Because the locations of the places are in latitude and longitude, some calculation was required to calculate the distance between locations. 'About.com' describes each degree of latitude to be approximately 111 kilometres, varying between 110.567 kilometres at the equator, and 111.699 kilometres at the poles. It also describes each degree of longitude to vary between 111.321 kilometres at the equator and gradually shrinks to zero at the poles [1]. At forty degrees north or

south, the distance between a degree of longitude is 85 kilometres. Since the UK lies at around 50 degrees north, I assumed that the distance between a degree of longitude is approximately 80 kilometres.

By dividing a degree of latitude by 111 kilometres, roughly the mean of the two extremes of latitude distance, it can be said that 1 kilometre is roughly 0.09 degrees in latitude. By dividing a degree of longitude by 80 kilometres, it can be said that 1 kilometre is roughly 0.125 degrees of longitude. It is these rough measurements that the search function uses to construct the SQL query. It compares the locations of the journey origin location, destination location, and journey step locations to the locations of the searched origin and destination.

Once the search function had been developed and was working correctly: returning journeys which not only started and ended at the searched locations, but journeys which passed through such searched locations; the hitch request was subject to acceptance or denial by the driver of the journey. If the driver accepted the hitch and it involved only a part of the full journey, the journey steps needed to be modified in the database to include the new way points. The management of journey steps in the database and PHP database controller classes proved quite difficult, but was eventually accomplished by the communication between controller classes. The acceptance of hitch requests follow these steps:

- Move current journey steps relating to journey into 'journey_step_temp' table
- Retrieve all accepted hitch requests for journey from 'hitch_request' table
- Use the origin, destination and waypoints from the accepted hitch requests to retrieve new route data from google directions API
- Parse the resulting JSON into journey steps
- Insert the new journey steps into the 'journey_step' table
- Change the response of the hitch request to true
- Decrement the amount of spaces left on the journey
- Delete the old journey steps from the 'journey_step_temp' table

Provided a more generous time-scale, I would have included a feature counting the spaces available during each part of the journey. For example, adding a 'js_spaces_available' field to the 'journey_step' table to account for pick ups and drop off decrementing and incrementing the spaces left as the journey travels along the route.

3.7 Website

Communication with the API started as a JSON interface over HTTP post. This proved to be little more than an over-separation of the two structures and over complication of the design. The development of the website did take some time, but once the foundations of the database controller classes were laid down, it was simply a case of developing the website files which would make use of them.

The use of the Bootstrap CSS library really helped the development of the website. Simplicity was key and Bootstrap steered the site towards a formalised visual structure so that each page seemed to appear structurally identical to the last. It was also very easy to maintain and manage when needing to change the look of things on a page.

3.8 Review

All of the basic requirements of the project were fulfilled in the end. However, more requirements would have been introduced had the timespan of the project been greater. This would have made the project much more usable and would have encouraged more users to join up had they been introduced.

Chapter 4

Testing

4.1 Overview

Testing was performed in the following ways. The main strategies I have used involve feature testing against the original requirements and performing PHPUnit tests on the Database controller classes. The outputs of which can be viewed in the appendix.

4.2 Database Controller Testing

4.2.1 Overview

The testing for the database controller was done by unit testing the model and controller classes. The feature testing of the database controller was performed during the feature testing of the website during use. Each feature described in the project specification (Appendix) was tested as a feature

4.2.2 Unit Tests

PHPUnit tests were written for the testing of the model classes which represent tables in the database. The reason for this was to ensure that data was being transferred and handled correctly to and from the database into the model classes. The CRUD methods, in each model classes in the database controller, needed to work perfectly for the higher level functions in the corresponding controller classes to function properly.

Each model class, 'Person.php', 'Message.php', 'Journey.php', 'Journey_Step.php' and 'Hitch_Request.php' were tested for their 'Create()', 'Load()', 'Update()' and 'Delete()' methods to ensure that data was being parsed correctly. The model tests followed the following steps to test these functions:

Instantiate Instantiating the object and set the attributes to desired values

Create() Use the attribute values to insert a new record into the database table via the 'Create()' method in the class.

Load() Use the 'Load()' method to retrieve the new entry in the table and assert the retrieved values are equal to the initial attribute values used to insert into the database table.

Update() Change one of the attributes in the model class and use 'Update()' to update the entry in the database table with the new attribute values and assert that the new retrieved values are equal to that of the updated ones in the class.

Delete() Use delete to remove the entry in the database table and assert that the action returns true.

As well as the testing of the model classes, the controller classes have each been tested for their individual functions to ensure that they are utilizing the database table model classes correctly. The results of all of these can be seen in the appendix.

4.3 Website Testing

4.3.1 Overview

The website testing was performed by following the functional requirements set out in the project specification document compiled at the beginning of the project. The full feature test results table can be found in the appendix.

4.3.2 Functional Tests

The functional tests of the website are high level functional requirements that involve all sections of the project. The reason for performing these feature tests was to make sure that the website performed the tasks that it was set out to do from the beginning of the project. The basic results are as follows:

Test Code	Description	Result
FR.I.01	Register Form - The page must provide an easy to access register form for personal details to be entered upon registration. Basic details needed are: email address, first name, last name and password. More specific details can be entered after registering to the site	Pass
FR.I.02	Log in Form - There must be a section of the page dedicated to logging in existing users. It must be easily viewed and apparent to users.	Pass
FR.H.01	Menu - There should be a menu giving the user access to all of the site's functions separated out into pages that group functionality	Pass
FR.H.02	Logout - A logout option must be present	Pass
FR.H.03	User Journeys - Users must be able to see a list of their shared journeys and view their status. This could show how many spaces they have left and new requests others have made to hitch the journeys	Pass

FR.H.04	User Hitches - User must be able to view a list of the hitches they have requested to other journeys and view their status. The status of the hitch request depends on whether or not it has been accepted by the driver	Pass
FR.H.05	Post Journey - The user must be able to share journeys by posting the details of the journey up on the page. the site must provide a map-style preview of the journey so that the user can be sure the correct origin and destination have been set	Pass
FR.H.06	Search For Journeys - The site must provide a search function for the user to find journeys from one location to another. This search function must include date parameters	Pass
FR.H.07	Make a Hitch Request - If a user a selected a journey after searching, they must be able to make a hitch request provided there are spaces remaining	Pass
FR.H.08	Accept/Decline Hitch Requests - When another user requests to hitch a journey that a user has shared, the site must prompt them to accept or decline the new hitcher	Pass
FR.H.09	Suggested Journeys - The site must provide suggested journeys that it predicts a user may be interested in due to their profile details	Pass
FR.H.10	Messages - The ability to send and receive messages to and from other users on the site must be available	Pass
FR.H.11	Profile - The user must be able to view and edit their personal and profile details	Pass
FR.H.12	Journey Cancelling - The user must be able to cancel a journey at any time	Pass
FR.H.06.01	Distance Search - The search function must search not only for journeys which originate and terminate at the exact searched locations, but must return journeys which originate and terminate at locations near to that of the search. For example, within ten kilometres	Pass
FR.H.06.01	Partial Journey Search - The search function must return journeys in which the searched origin and destination are, or near to (see FR.H.06.01), points along the journey	Pass

These feature test results show that the main features specified at the beginning of the project were completed and work as they are expected to.

Chapter 5

Evaluation

5.1 Original Goals

The original goals set out in the project specification were set in such a way that they could be realistically accomplished within the time set for the project.

Before the project was fully under way, part of the original goals was to aim the site at students in university, not just the general public. There was also the discussion of producing a mobile application along with the website that would be used by users to share and hitch journeys. It became apparent that this was far too much work to be completed in the time frame available. Therefore, it was decided that the website, with the specific search features and journey alterations, be identified as the priority.

I feel that the requirements of the project were identified fairly well. The management of the journeys, the combination of pick ups and drop off points at certain points of journeys, was identified as one of the core concepts in the requirements for good reason. They enable users to accept or decline depending on their own preferences, and can view the edited route should they accept the hitcher. This enables them to make an educated decision as to whether it is cost effective / worth the effort of picking the hitcher up.

5.2 Accomplishments

I believe that the original goals that were set out from the beginning were accomplished to a good standard. I also believe however, that the project deserves a more extensive result. The purpose of the website was always to provide a solution to a wide range of people, and to provide a wide range of services so as to encourage a large user base. Unfortunately, I think the intended size of the project I had planned from the beginning, before requirements were set, was perhaps larger than was realistically possible which resulted in the specifications being as small as they are.

The site performs the type of search that I emphasised in the requirements; it returns journeys which cover points along the routes, and then changes the routes depending on pick up points and drop off points if the hitch request is accepted by the driver.

5.3 Possible Improvements

5.3.1 Environment

The PSQL database I used to create my data storage system did cause time to be wasted slightly. This was mainly due to the fact that I did not have any access to a database management software system, such as 'phpMyAdmin' [8], to manage the database tables themselves; everything had to be performed in the command line. Although this does provide a lot of flexibility, I feel as though time could have been saved with some sort of graphical user interface instead of text based.

5.3.2 System

There are a few things I would improve in hindsight. The website interface leaves a bit to be desired when it comes to graphical design. The initial idea was to make sure that all the pages worked correctly, and then introduce more interesting design features. However, time constraints meant that functionality came before producing more attractive pages.

In terms of the project's actual functionality, I would probably add a feature which acts like a 'wanted' add. Currently, the site accommodates for drivers haring journeys that they plan on driving, and offer them to potential passengers. The passengers then search for journeys between locations they wish to travel and hope that a journey exists already. The 'wanted' journey add would enable users to post up a journey that they wish they could travel, but can't because of no vehicle / lack of money. Drivers could see them and decide that they would want to take part in that journey. This approach could increase use in the site because both the drivers and hitchers share their wishes to travel, where as currently the functionality only allows drivers to share, and hitchers to search.

I think I would improve the method in which the website works with the database controllers. The code that dynamically generates the HTML and JavaScript for the user's browser is quite messy. This is because it needs to make use of the database controllers almost like a 'main' method does in a Java program.

If starting the project from scratch again, I would seriously consider developing a completely Model-View-Controller approach to the entire system.

5.4 Future Development

The future development of this project would involve the following features:

- The improvement of the website interface. Currently, the website does perform the tasks it was set out to provide to the user, but could be improved by analysing the current state of interaction between human and site with questionnaires.
- Interoperability using protocols such as OAuth to extract existing user data from accounts the user already maintains. This would include social media accounts such as Facebook [13] and Twitter [20]
- Improved messaging system.

- Ability to post 'wanted' type adds as a hitcher. This would enable drivers to see the adds and possibly offer to be the driver.

5.5 Design Choices

5.5.1 Database Controller

I think the choices I made regarding design of the project could have been better. In hindsight, the design of the database controllers and website could have been much more structured in the form of a standard design pattern. Although it is a tweaked version of the Model-View-Controller pattern, I think it came to be as a result of indecision regarding the design of the system. If more research was done towards Representational State Transfer (RESTful) services, I would have probably designed the project to follow that kind of structure instead. This is due to RESTful services being so easily expandable, and would probably have allowed me to more easily add extra features to the site once the core of the project had been developed.

5.5.2 Website

The website developed was completely procedural, and made use of the object oriented controller classes when required. Although this allowed me to develop a completely customised website to my own specifications on each page, it did mean that each file that I produced had to be written from scratch. There are obvious advantages to this, such as being able to develop customised pages for whatever function the page is intended to provide. However, the time it took to create them, and the amount of separate files that needed to be created, was a potential hindrance to the project.

5.5.3 Database

I feel the design of the database was as simple as could have been possibly made. It enables the site to store as much important information as possible whilst reducing the complexity of the queries that need to be made to extract related information from multiple tables. Other tables and features could have been added had the requirements stated so, but the design of the database enables the system to function according to the specifications that were finalised during the planning stage.

5.6 Approach

The approach to the design of this project could have been performed to a better standard. Initially, not sure whether I wanted to develop using an agile methodology or a more structured one like the Waterfall method, I deviated from following any methodology when developing prototypes to test some features without any planning. Although I did gain some valuable information from this, I feel that it may have wasted time that could have been better spent planning, researching and designing the system at a higher level. Doing more research could have resulted in me realising that more features could have been added to the specifications and introduced to the system.

In hindsight, I do believe that the Waterfall method was the correct method to develop this project. Although I did not follow its steps strictly, I believe that it enabled me to make architectural decisions regarding the database which would not have developed with a sensible structure had I used an agile approach such as Feature Driven Development. Specifications were outlined at the beginning, and design documents and diagrams were produced.

Appendices

Appendix A

Third-Party Code and Libraries

1.1 JQuery Library

The JQuery library was used on parts of the website without modification and is available from JQuery Foundation [14].

1.2 Bootstap CSS Library

The Bootstrap CSS library was used to develop the graphical user interface of the website and is available from 'getbootstrap.com' [5] [3] [6] [4]

1.3 Google Maps Javascript API

The Google Maps Javascript API was used to develop parts of the website which embeds Google Maps into it. Details can be found on the API documentation pages [16].

1.4 Google Directions API

The Google Directions API was used to gather routing data. The details can be found on their API pages [11].

1.5 Google Geocoding API

The Google Geocoding API was used in the project to gather location data about place names. The details can be found at the API pages [12].

Appendix B

Code samples

2.1 SQL Search Query Generator

This is the PHP generation of the search query which queries the database for journeys when searching with an origin name, destination name, origin latitude and longitude, destination latitude and longitude, and time frame.

```
$sql_string = "select * from (select * from journey where
    jr_origin = '$this->jr_origin' and jr_destination = '$this->
    jr_destination'
            and jr_etd >= '$this->search_date_1' and jr_etd <= '
                $this->search_date_2' and jr_spaces_available > 0
                 and jr_is_cancelled = 0 union

            select * from journey where jr_origin_latitude -
                $this->jr_origin_lat < 0.04532
            and jr_origin_latitude - $this->jr_origin_lat > -
                0.04532 and jr_origin_longitude - $this->
                jr_origin_lng
            < 0.041 and jr_origin_longitude - $this->
                jr_origin_lng > -0.041 and
                jr_destination_latitude
            - $this->jr_destination_lat < 0.04532 and
                jr_destination_latitude - $this->
                jr_destination_lat > -0.04532
            and jr_destination_longitude - $this->
                jr_destination_lng < 0.041 and
                jr_destination_longitude
            - $this->jr_destination_lng > -0.041 and jr_etd >= '
                $this->search_date_1' and jr_etd <= '$this->
                search_date_2' and jr_spaces_available > 0 and
                jr_is_cancelled = 0 union
```

```
select * from journey where jr_pk in (select
    distinct js_jr from journey_step where js_jr in (
    select jr_pk from journey where
    jr_origin_latitude - $this->jr_origin_lat <=
    0.04532
and jr_origin_latitude - $this->jr_origin_lat >= -
    0.04532 and jr_origin_longitude - $this->
    jr_origin_lng
<= 0.041 and jr_origin_longitude - $this->
    jr_origin_lng >= -0.041) and js_latitude - $this
    ->jr_destination_lat >= -0.3153
and js_latitude - $this->jr_destination_lat <=
    0.3153 and js_longitude - $this->
    jr_destination_lng >= -0.35 and
js_longitude - $this->jr_destination_lng <= 0.35)
    and jr_etd >= '$this->search_date_1' and jr_etd
    <= '$this->search_date_2' and jr_spaces_available
     > 0 and jr_is_cancelled = 0 union

select * from journey where jr_pk in (select
    distinct js_jr from journey_step where js_jr in (
    select jr_pk from journey where
    jr_destination_latitude - $this->
    jr_destination_lat <= 0.04532
and jr_destination_latitude - $this->
    jr_destination_lat >= - 0.04532 and
    jr_destination_longitude - $this->
    jr_destination_lng
<= 0.041 and jr_destination_longitude - $this->
    jr_destination_lng >= -0.041) and js_latitude -
    $this->jr_origin_lat >= -0.3153
and js_latitude - $this->jr_origin_lat <= 0.3153 and
     js_longitude - $this->jr_origin_lng >= -0.35 and
js_longitude - $this->jr_origin_lng <= 0.35) and
    jr_etd >= '$this->search_date_1' and jr_etd <= '
    $this->search_date_2' and jr_spaces_available > 0
     and jr_is_cancelled = 0 union

select * from journey where jr_pk in (select
    distinct a.js_jr from journey_step a inner join
    journey_step b on a.js_jr = b.js_jr
where a.js_latitude - $this->jr_origin_lat >= -0.1
    and a.js_latitude - $this->jr_origin_lat <= 0.1
    and
a.js_longitude - $this->jr_origin_lng >= -0.09009
    and a.js_longitude - $this->jr_origin_lng <=
    0.09009 and
```

```
b.js_latitude - $this->jr_destination_lat >= -0.1
    and b.js_latitude - $this->jr_destination_lat <=
    0.1 and
b.js_longitude - $this->jr_destination_lng >=
    -0.09009 and b.js_longitude - $this->
    jr_destination_lng <= 0.09009)
and jr_etd >= '$this->search_date_1' and jr_etd <= '
    $this->search_date_2' and jr_spaces_available > 0
     and jr_is_cancelled = 0) journey order by jr_etd
    ";
```

Appendix C

Development Process Documentation

3.1 Project Specification Document

Project Specification

Alexander Roan

May 9, 2014

Version 2.0 Release

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the prospective product in detail, outlining specific requirements that it must adhere to.

1.2 Definitions

- Journey - A proposed journey following a route between an origin and destination by a person.
- Hitch - When a person joins a journey by taking up a spare seat on a journey.

1.3 System Overview

The purpose of the Online Carpooling Service is to provide drivers with the ability to share the details of prospective journeys, enabling non-drivers to travel in the same direction by joining them on their journey. Many people drive long distances from one location to another without knowing they could make the journey together and save money. My motivation for this project comes from this very situation. I wish to bring together people who travel from place to place, often in parallel to each other. This site aims to bring together these people, enabling them to spend less money on fuel which in turn could help free up clogged roads and reduce air pollution.

2 Overall Description

2.1 Product Perspective

The website will be the interface for the user base. Using it, they will have access to all of the features that the service offers and will be able to update their profile enabling suggestions to be put forward to them by the service.

2.2 User Characteristics

People who sign up to the site will provide personal information which will be saved in the database. This personal information will aid the service in producing Journey and Hitch suggestions.

The users accessing the site will use it to share journeys that they are partaking in with the aim of attracting other users to join them in those

journeys. Conversely, the site is also used for users who wish to find journeys that other people are driving, so that they can hitch a journey.

2.3 Product Functions

The main functions of the site will be:

- Register as a new user to the site
- Log in with existing details
- Edit profile details
- Post a journey that the user will be driving from location to location
- Search for journeys between two locations. The search must return journeys in which the searched location appear along some points on the journeys.
- Request to Hitch a journey as a passenger
- Accept or decline a hitcher as the driver
- Send messages to other users on the site

3 Specific Requirements

The website must provide the following services:

Index Page The index page must provide:

FR.I.01 Register Form - The page must provide an easy to access register form for personal details to be entered upon registration. Basic details needed are email address, first name, last name and password. More specific details can be entered after registering to the site

FR.I.02 Log in Form - There must be a section of the page dedicated to logging in existing users. It must be easily viewed and apparent to users

Home Page When a person logs into their account, they must be provided with a home page with the following:

FR.H.01 Menu - There should be a menu giving the user access to all of the site's functions separated out into pages that group functionality

FR.H.02 Logout - A logout option must be present

FR.H.03 User Journeys - Users must be able to see a list of their shared journeys and view their status. This could show how many spaces they have left and new requests others have made to hitch the journeys

FR.H.04 User Hitches - User must be able to view a list of the hitches they have requested to other journeys and view their status. The status of the hitch request depends on whether or not it has been accepted by the driver

FR.H.05 Post Journey - The user must be able to share journeys by posting the details of the journey up on the page. the site must provide a map-style preview of the journey so that the user can be sure the correct origin and destination have been set

FR.H.06 Search For Journeys - The site must provide a search function for the user to find journeys from one location to another. This search function must include date parameters

FR.H.07 Make a Hitch Request - If a user a selected a journey after searching, they must be able to make a hitch request provided there are spaces remaining

FR.H.08 Accept/Decline Hitch Requests - When another user requests to hitch a journey that a user has shared, the site must prompt them to accept or decline the new hitcher

FR.H.09 Suggested Journeys - The site must provide suggested journeys that it predicts a user may be interested in due to their profile details

FR.H.10 Messages - The ability to send and receive messages to and from other users on the site must be available

FR.H.11 Profile - The user must be able to view and edit their personal and profile details

FR.H.12 Journey Cancelling - The user must be able to cancel a journey at any time

Specific Search Requirements The 'Search For Journeys' requirement outlined in FR.H.06 needs to include the following:

FR.H.06.01 Distance Search - The search function must search not only for journeys which originate and terminate at the exact searched locations, but must return journeys which originate and terminate at locations near to that of the search. For example, within ten kilometres

FR.H.06.02 Partial Journey Search - The search function must return journeys in which the searched origin and destination are, or near to (see FR.H.06.01), points along the journey

3.2 Design Specification Document

Design Specification

Alexander Roan

May 9, 2014

Version 2.0 Release

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to detail the design of the Online Carpooling Service describing the functional workings of the database and code.

1.2 Objectives

The objectives are to describe the overall system and its workings

2 Database Structure

The decomposition of the database describes each of the tables and their attributes.

Person :

ps_email character varying, not null, Primary Key

ps_first_name character varying, not null

ps_last_name character varying, not null

ps_password character varying, not null

ps_home_1 character varying

ps_home_2 character varying

ps_frequent_destination_1 character varying

ps_frequent_destination_2 character varying

ps_frequent_destination_3 character varying

ps_current_location character varying

ps_register_date date, not null, default = Now()

The 'Person' table holds the personal details of each of the website's users. The 'home_1' through to 'current_location' are not required to be filled in upon registration to the site but are required if the user wishes to receive suggested journeys. The primary key in this table is the user's email address.

Message :

ms_pk big serial, not null, Primary Key

ms_ps_sender character varying, not null, references Person(ps_email)

ms_ps_receiver character varying, not null, references Person(ps_email)

ms_title character varying

ms_body character varying

ms_sent_date date time

ms_read_date date time

The 'Message' table holds all message information between the users on the site. The primary key is generated as the next available primary key in the table.

Journey :

jr_pk big serial, not null, primary key

jr_ps_email character varying, not null, references Person(ps_email)

jr_origin character varying, not null

jr_destination character varying, not null

jr_etd date time, not null

jr_eta date time, not null

jr_spaces_available integer, not null

jr_total_spaces integer, not null

jr_description character varying

jr_register_date date time, default = Now()

jr_extra_distance real, default = 30.0

jr_origin_latitude real, not null

jr_origin_longitude real, not null

jr_destination_latitude real, not null

jr_destination_longitude real, not null

jr_total_distance real, not null

jr_is_cancelled small integer, not null, default = 0

The 'Journey' table is responsible for storing the majority of information relating to a journey posted by a user. The primary key is generated as the next available integer in the table. 'jr_ps_email' refers to the entry in 'Person' table which posted the journey i.e. The driver of the journey.

Journey_Step :

js_pk big serial, not null, primary key

js_jr integer, not null, references Journey(jr_pk)

js_step_order integer, not null

js_latitude real, not null

js_longitude real, not null

The 'Journey_Step' table stores data regarding each step of journey. The primary key is generated as the next available integer primary key. The 'js_jr' refers to the journey that the step belongs to by referencing the 'Journey' table's primary key. The 'js_step_order refers to the order of the step in the journey.

Journey_Step_Temp :

st_pk integer, not null, primary key

st_jr integer, not null, references Journey(jr_pk)

st_step_order integer, not null

st_latitude real, not null

st_longitude real, not null

The 'Journey_Step_Temp' table is identical to the 'Journey_Step' table except for the field names and the fact that the primary key is not generated, but rather inserted.

Hitch_Request :

hr_pk big serial, not null, primary key

hr_jr integer, not null, references Journey(jr_pk)

hr_ps_email character varying, not null, references Person(ps_email)

hr_request_date date time, not null, default = Now()

hr_response_date date time

hr_response small integer

hr_waypoint_1 character varying

hr_waypoint_2 character varying

The 'Hitch_Request' table stores data relating to a hitch request that a user makes to a journey. The primary key is generated as the next available integer in the field. 'hr_jr' references the journey that is being requested to hitch. 'hr_ps_email' refers to the user that made the request by referencing the 'ps_email' field in the 'Person' table.

3 PHP

3.1 Overview

The client side application is split into two structures of PHP files. One group, called the 'Database Controller' acts as the gateway of interaction with the database. This structure is an object oriented representation of the database itself, with controller classes which perform the complex tasks that the client side website utilizes.

The other group will be the website files. These files are a collection of procedural PHP files which dynamically generate HTML data depending on the data received from the database controller classes.

The design is almost a tweaked version of Model-View-Controller, where the controllers and the models are the database controller, and the view is comprised of the website files.

3.2 Database Controller

Each of the tables in the database has a corresponding PHP class, a 'model' class, in the database controller. Each of these classes representing a table contain attributes of the same names as its table counterpart, and overwrite four functions from the 'Table' class they extend. These functions are named 'Create()', 'Load()', 'Update()', 'Delete()'.

Create() inserts a new record into the corresponding table using an SQL procedure and the values that the class' attributes are set to.

Load() takes a parameter as a string, and uses it as the primary key to the corresponding table to retrieve an entry from the table. If an entry is found, it will set the class' attributes equal to that of the corresponding values retrieved from the database.

Update() updates the table using the attributes in the class, of which will include the primary key to the table being updated. This method

assumes that changes have been made to the attributes in the class object before hand.

Delete() attempts to delete a record from the table where the primary key matched the same attribute stored in the class object. It then resets the values of all of the attributes in the object in case the object is used again.

As well as each of the model classes, controller classes instantiate and control them. The model classes are structurally very similar to each other with the only difference being which tables / model classes they represent. Controller classes are structured depending on how they make use of the model classes. The 'Journey_Controller' class needs to perform many actions using the 'Journey' model class and interact with the 'Journey_Step_Controller', so there are far more methods in these classes that in the 'Message_Controller' for example.

3.2.1 API.php

All classes in the database controller classes extend 'API.php'

__construct() constructor

API() constructor

ConnectToDatabase()

CloseConnection()

Get()

Set()

GetAll() Returns All attributes from an object

3.2.2 Journey_Controller.php

extends 'API.php'

__construct() constructor

Journey_Controller() constructor

GetMyJourneys()

Private ReturnJourneys() Function used by other functions in the class to format the return structure as an array

FillSpace() Fills a space in a journey, decrements the amount of spaces available for a particular journey

LoadJourney() Loads a journey object to the controller

GetJourneyDataAll()

CancelJourney()

SearchJourney()

CreateJourney()

CreateJourneySteps()

ModifyJourneySteps()

Private MoveOldJourneySteps()

Private DeleteOldJourneySteps()

SetWaypointArray()

BuildDirectionsUrl()

Geolocate()

Private SetGeolocation()

Private BuildSearchString()

3.2.3 Journey_Step_Controller.php

extends 'API.php'

__construct() constructor

Journey_Step_Controller() constructor

MoveTempJourneySteps()

DeleteTempJourneySteps()

CreataeJourneySteps()

3.2.4 Message_Controller.php

extends 'API.php'

__construct() constructor

Message_Controller() constructor

SendMessage()

LoadMessage()

LoadMyMessages()

GetMessages()

GetNumberOfUnreadMessages()

3.2.5 Hitch_Request_Controller.php

extends 'API.php'

__construct() constructor

Hitch_Request_Controller() constructor

GetHitchRequestsForJourney()

GetNewRequests()

GetMyHitchRequests()

Private ReturnHitchRequests()

CreateHitchRequest()

LoadHitchRequest()

AcceptHitchRequest()

DeclineHitchRequest()

3.3 Website

The website files are the files that are accessed by the user when interacting with the website. They comprise of 7 main files which display the main features of the site. These files are as follows:

index.php The main page that the user comes to when accessing the site. Has a register form and a log in form on it. If a session for the site already exists then the browser is redirected to 'home.php'

home.php The home page when logged on to the site. If there is no session already then the browser is redirected to 'index.php'. This page shows suggested journeys for the user.

activity.php Displays the user's journeys and hitch requests and prompts any changes that have occurred for them to respond to.

messages.php Shows the user's inbox, sent messages and enables them to send a new message.

profile.php Shows the user's profile details.

search_journey.php A form with search parameters for the user to fill in when searching for journeys.

post_journey.php A form page with fields which the user is required to fill in before sharing a journey on the site.

Each of these pages require other files to function properly. For example, the 'profile.php' page has a button on the bottom of the page labelled 'Edit Profile'. This button takes the user to 'edit_profile.php' where the static fields from 'profile.php' are editable, and can be submitted to the database using the 'Submit' button which activates database controller classes.

3.3 Feature Testing Document

Feature Testing

Alexander Roan

May 8, 2014

Version 2.0 Release

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to outline the feature requirements set at the beginning of the project for the website and describe how the website performs against these specifications.

1.2 Objectives

The objective is to details how successful the development of the project has been in achieving its goals that were set from the beginning.

2 Feature Tests

The following table describes each feature and how the website reacts to test input for each feature.

Code	Description	Input	Output	Result
FR.I.01	Register Form - The page must provide an easy to access register form for personal details to be entered upon registration. Basic details needed are Email address, First name, Last name and Password. More specific details can be entered after registering to the site	Correctly formatted details	User registered and logged into the site	PASS

		Wrongly formatted email address	User not registered and told to re enter details	PASS
FR.I.02	Log in Form - There must be a section of the page dedicated to logging in existing users. It must be easily viewed and apparent to users	Correct details already in database	Logged into site	PASS
		Details not already in database but formatted correctly	not logged into site, redirected back to home page to re enter details	PASS
		Badly formatted email address	not logged into site and asked to re-enter details	PASS
FR.H.01	Menu - There should be a menu giving the user access to all of the site's functions separated out into pages that group functionality	Clicking the menu buttons	redirects the page to the correct one	PASS

FR.H.02	Logout - A logout option must be present	Logout button pressed	user redirected to index page where they are required to log in or register	PASS
FR.H.03	User Journeys - Users must be able to see a list of their shared journeys and view their status. This could show how many spaces they have left and new requests others have made to hitch the journeys	'Activity' button pressed in menu	Taken to activity page where journeys are displayed. Each one can be viewed in detail by pressing on the journey div itself	PASS
FR.H.04	User Hitches - User must be able to view a list of the hitches they have requested to other journeys and view their status. The status of the hitch request depends on whether or not it has been accepted by the driver	'Activity' button pressed in menu	Taken to activity page where user hitch requests are displayed. Each one can be viewed in detail by pressing on the journey div itself	PASS

FR.H.05	Post Journey - The user must be able to share journeys by posting the details of the journey up on the page. the site must provide a map-style preview of the journey so that the user can be sure the correct origin and destination have been set	Data with corect place names inserted into form and submitted	preview page shows correct map preview and option to submit or go back a page	PASS
		Data with incorrect place name inserted into form	preview page shows incorrect map as expected and option to go back	PASS

FR.H.06.01	Distance Search - The search function must search not only for journeys which originate and terminate at the exact searched locations, but must return journeys which originate and terminate at locations near to that of the search. For example, within ten kilometres	'Penglais' to 'Penarth' search inserted into form	Journeys from 'Aberystwyth' to 'Cardiff' are returned in search results	PASS
FR.H.06.02	Partial Journey Search - The search function must return journeys in which the searched origin and destination are, or near to (see FR.H.06.01), points along the journey	search from 'Brecon' to 'Builth Wells' inserted into form and submitted	Search results include journeys from 'Aberystwyth' to 'Cardiff'	PASS
FR.H.07	Make a Hitch Request - If a user a selected a journey after searching, they must be able to make a hitch request provided there are spaces remaining	'Request Hitch' button clicked on journey	Hitch now appears in 'Activity' page as awaiting response from driver	PASS

FR.H.08	Accept/Decline Hitch Requests - When another user requests to hitch a journey that a user has shared, the site must prompt them to accept or decline the new hitcher	'Accept Hitcher' button clicked on journey page	Hitch is now part of journey, journey route updated and spaces left decreased	PASS
		'Decline Hitcher' button pressed	Hitch request no longer shows up and no change to journey	PASS
FR.H.09	Suggested Journeys - The site must provide suggested journeys that it predicts a user may be interested in due to their profile details	User logged onto site or returns to home page with full profile completed	3 suggested journeys displayed	PASS

		User navigates to home page without full profile completed	messages prompting to complete profile for suggested journeys shown	PASS
FR.H.10	Messages - The ability to send and receive messages to and from other users on the site must be available	Message sent to other user using email address	Email sends successfully and appears in 'Sent Items' folder	PASS
FR.H.11	Profile - The user must be able to view and edit their personal and profile details	User navigates to 'Profile' page	profile details displayed	PASS
		'Edit Profile' button clicked	User directed to page where details can be edited after pressing 'Submit' button	PASS

FR.H.12	Journey Cancelling - The user must be able to cancel a journey at any time	User clicks 'Cancel Journey' button on journey page	Journey cancelled and redirected to 'Activity' page where journey no longer displayed	PASS

3.4 Unit Testing Results

PHPUnit 4.0.18 by Sebastian Bergmann.

..

Time: 112 ms, Memory: 3.00Mb

OK (2 tests, 1 assertion)

PHPUnit 4.0.18 by Sebastian Bergmann.

.

Time: 302 ms, Memory: 3.00Mb

OK (1 test, 8 assertions)

PHPUnit 4.0.18 by Sebastian Bergmann.

..

Time: 117 ms, Memory: 3.00Mb

OK (2 tests, 3 assertions)

PHPUnit 4.0.18 by Sebastian Bergmann.

.

Time: 177 ms, Memory: 3.00Mb

OK (1 test, 12 assertions)

PHPUnit 4.0.18 by Sebastian Bergmann.

.

Time: 244 ms, Memory: 3.00Mb

OK (1 test, 30 assertions)

PHPUnit 4.0.18 by Sebastian Bergmann.

.

Time: 151 ms, Memory: 3.00Mb

OK (1 test, 12 assertions)

PHPUnit 4.0.18 by Sebastian Bergmann.

....

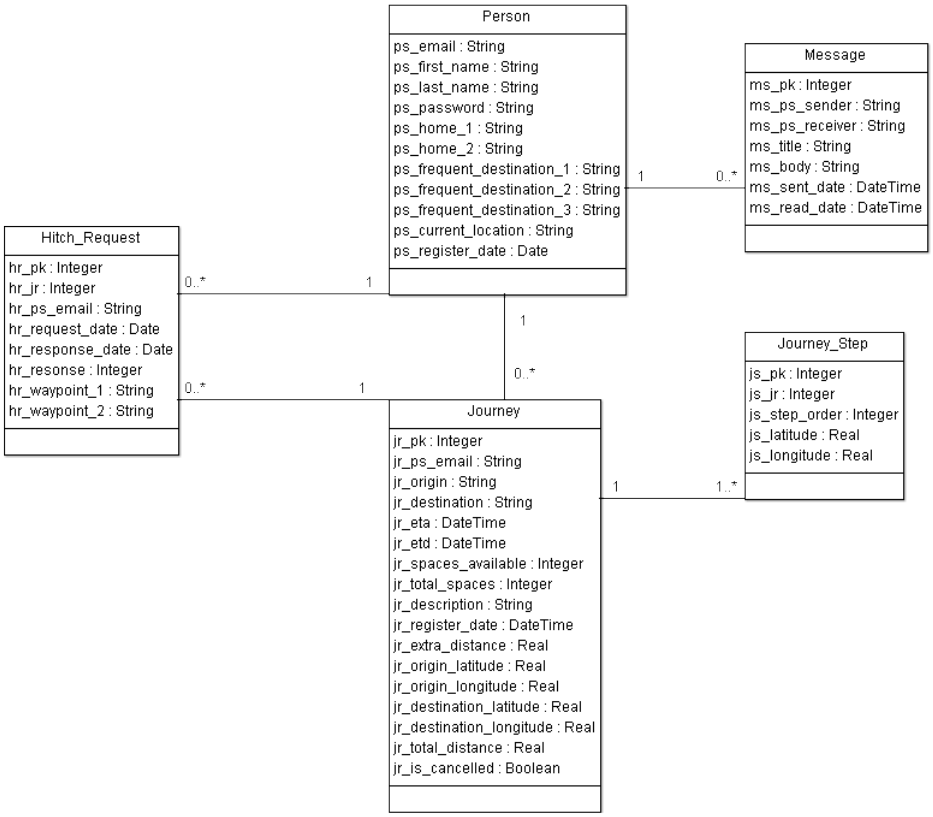
Time: 206 ms, Memory: 3.00Mb

OK (4 tests, 26 assertions)

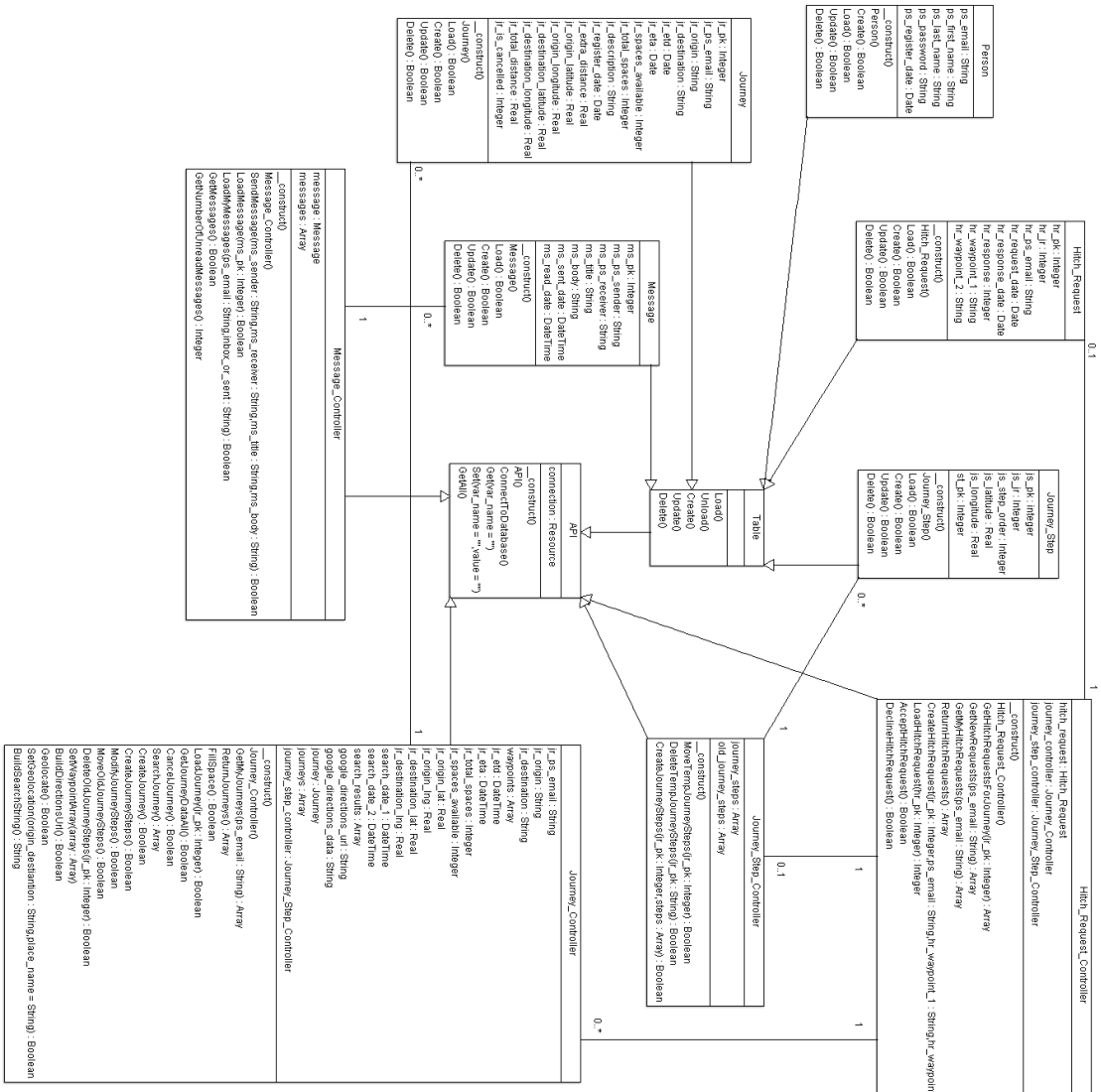
Appendix D

Design Diagrams

4.1 Database UML



4.2 Database Controller Classes



Annotated Bibliography

[1] About.com, "What is the distance between a degree of latitude and longitude?" http://geography.about.com/library/faq/blqzdistancedegree.htm, 2014.

[2] BlaBlaCar.com, "Blablacar.com - online carpooling website," http://www.blablacar.com/, 2014.

[3] Bootstrap, "Bootstrap dashboard," http://getbootstrap.com/examples/dashboard/, 2014.

[4] ——, "Bootstrap jumbotron," http://getbootstrap.com/examples/jumbotron/, 2014.

[5] ——, "Bootstrap library," http://getbootstrap.com/, 2014.

[6] ——, "Bootstrap signin," http://getbootstrap.com/examples/signin/, 2014.

[7] Carpooling.com, "Carpooling.com - online carpooling website," http://www.carpooling.co.uk/, 2001.

[8] P. Contributors, "Phpmyadmin," http://www.phpmyadmin.net/home_page/index.php, 2014.

[9] DatabaseAnswers.com, "Crud matrix," http://www.databaseanswers.org/data_migration/crud_matrix.htm, 2014.

[10] U. G. Department of Transport, "Transport statistics great britain 2011," https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/11833/Transport_Statistics_Great_Britain_2011___all_chapters.pdf, p. 4, 2011.

[11] developers.google.com, "Google directions api," https://developers.google.com/maps/documentation/directions/, 2014.

API used to retrieve routing data

[12] ——, "Google geocoding api," https://developers.google.com/maps/documentation/geocoding/, 2014.

API used to retrieve location data using individual place names

[13] Facebook, "Facebook," https://www.facebook.com/, 2014.

[14] J. Foundation, "Jquery library," http://jquery.com/, 2014.

[15] Google, "Google api usage limits," https://developers.google.com/maps/documentation/directions/, 2014.

'Usage Limits'

[16] ——, "Google maps javascript api," https://developers.google.com/maps/documentation/javascript/, 2014.

[17] G. inc, "Github," https://github.com, 2014.

Version control service

[18] oauth.net/, "Oauth," http://oauth.net/, 2014.

An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications

[19] The Train Line, "TheTrainLine.com Train Times Booking. Aberystwyth - Cardiff," http://www.thetrainline.com/buytickets/combinedmatrix.aspx?Command=TimeTable#Journey/AYW/CDF/01/05/14/13/0/Dep/////Dep/1/0//0;0;0, 2014.

A query to thetrainline.com made by myself about the prices of train fair between Aberystwyth and Cardiff on the 1st of May 2014 at 13:30. The price that was shown was £54.80.

[20] Twitter, "Twitter," https://twitter.com/, 2014.

[21] A. Vial, "Uk fuel price: how has it changed over time?" http://www.theguardian.com/news/datablog/2012/nov/12/uk-fuel-price-over-time, 2012.