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: 1.0 (Draft)

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...

I’d like to thank...

Abstract

Include an abstract for your project. This should be no more than 300 words.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

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 [3], 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 [7] as well. This website aims to bring people together who are travelling in similar directions 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 [6], 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 saves £10 on their journey, and the passenger saves a massive £44.80 on theirs!

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 ability 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 users searches if the parameters match the details of the journey.

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 location A to location B, but it could also be from location A to a new location C. Similarly it could be from location C to location B, or even location C to location D. 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 [2] and BlaBlaCar [1]. 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 a journey already with different origin and destination, not just recognise the searched origin and destination 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.

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 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

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 [**?**] to extract data from existing user account to different services. 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 estblished. There were a few methodologies considered, detailed in the Methodology section of this chapter.

Reserach 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 developing 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 or 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 would follw a strict design 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 was 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 [4] was found to be a valuable asset in the projects development. Using it enables routes to be created from an origin, destination and waypoints; perfect for the types of requests that this project will need to make.

Another valuable API that was researched was the Google Geocoding API [5]. 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 that of the requirements of 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 explination of the overall architecture of the project.

2.2 Technologies

2.2.1 PHP

PHP is used as the main language for the object oriented controlling of the data flow to and from the database. It is also used as the language to dynamically display produce the website output that the user sees.

2.2.2 JQuery

JQuery is featured on the website as a means of displaying certain features. Graphical maps generated by Google Directions [4] 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 [**?**] 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 an suitable IDE was required for the development process. Netbeans IDE 7.3.1 was the selected tool for this task, as its PHP plugin 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 the step.
- Journey_Step_Temp - A temporary table used when the journey steps for a particular journey change. This may occur is a hitch request is accepted which alters the route of the journey.
- Hitch_Request - Details about a hitch request made from a person 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 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 it procedural nature did not follow any sort of design pattern as it was not object oriented. The method of data transferral between API and website was that of 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 called on the rest of the JSON data. A data encoder method would parse this data, call a specific procedure 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 refactor or update. It was deemed that Object Oriented was the right path to follow.

If the API was going to separate from the website it should probably be in the form of a Representation State Transfer architecture, or 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 that would be made available, for example mobile applications and other third party websites, REST would be a perfect solution. But because only service accessing the API is this website, there is 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 to take advantage of the database controller classes and use their methods to perform actions of 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 contain the 4 methods: Create(), Load(), Update(), Delete(); following the CRUD persistent storage technique [**?**].

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 the method call 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 [**?**] and JQuery Library [**?**].

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 6 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 considering to hitch 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 highlights 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 user 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 if 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 simple 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 with two buttons each giving the option of immediately request to hitch, or 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 button present on the page for the user to click at this point: 'Request Hitch' and 'Find Another'. The 'Find Another' button simply 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 [4] 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 [**?**] 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 inside 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 the central classes with a large online repository of additional CSS files available for download for additional effects. The styles used in this project and where they are as following:

'Jumbotron' Jumbotron [**?**] 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 [**?**] 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 [**?**] 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 [**?**]. 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 sent items from it.

Chapter 3

Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

3.1 Overview

There were a few quite large scale 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 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 seperately 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 waypoints, 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 then 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 journey 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 [**?**] which takes the limit up to one hundred thousand requests per day and allows 23 way points in total per request.

3.4 Google Geocoding API

The Google Geocoding API [5] 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 adding attempting to add new 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 serached. 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.

Once the serach 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 acceptede 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 waypoints. The management of journey steps in the database and PHP database controller classes proved quite difficult, but was evetually 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 reponse to 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 encourage more users to join up had they been introduced.

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on real users? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

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 class in the database controller needed to work perfectly for the higher level functions in the corresponding controller class functioned 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:

Instanciate Instanciating the object and set the attributes to desired values

Create() Use the atribute 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 fnctional 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 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

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

5.1 Original Goals

5.2 Accomplishments

5.3 Future Improvements

5.4 Future Development

5.5 Design Choices

5.6 Approach

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the clients existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Appendix B

Code samples

Appendix C

Development Process Documentation

Appendix D

Design Diagrams

Annotated Bibliography

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

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

[3] 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.

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

    API used to retrieve routing data

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

    API used to retrieve location data using individual place names

[6] 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.

[7] 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.