BSc (Hons) Computer Science

A Full-Stack System Framework for Rollercoaster Tracking

By [redacted]
Supervised by Dr Paraskevas Yiapanis

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

Signed _____

Acknowledgement

I would like to thank my supervisor, Dr Paraskevas Yiapanis for his support throughout this project. I would also like to thank the users who participated during the testing of the project. Finally I'd like to thank my family for their support.

Abstract

This project is focused on the design and development of a web framework for creating, viewing and editing a theme park enthusiast's ride count (an in-depth list of the rides that have been ridden). Extra features such as ride statistics, user leader boards, user login accounts system, a ride recommendation feature and a new geolocation feature were implemented. The product created for this project has many parts that link to the framework. This includes existing mobile apps that needed to be substantially modified to add the new features and the creation of a new website allowing the users to access their data on any platform.

Table of Contents

List of Figures

List of Tables

Abbreviations

GPS	Global Positioning System
MoSCow	Must do, Should do, Could do, Won't do
UML	Unified Modelling language
HCI	Human-Computer Interaction
RFID	Radio Frequency Identification
DVD	Digital Versatile Disc
API	Application Programming Interface
TEA	Themed Entertainment Association
IDC	International Data Corporation
VB	Visual Basic
B4A	Basic 4 Android
B4i	Basic 4 iOS
IDE	Integrated Development Environment
HTML5	Hypertext Mark-up Language 5
PHP	PHP: Hypertext Pre-processor
MVC	Model View Controller
RDBMS	Relational Database Management System
SQL	Structured Query Language
GUI	Graphical User Interface
AWS	Amazon Web Services
TDD	Test Driven Development
CUST-	Custom-
ARC	Auto Ride Count
ARCC	Auto Ride Count Cloud (development name for this project)
UI	User Interface
WCF	Windows Communication Foundation
REST	Representational state transfer
ERD	Entity-Relationship Diagram
OWASP	Open Web Application Security Project
EC2	Elastic Compute Cloud
IIS	Internet Information Services
JSON	JavaScript Object Notation
IEEE	Institute of Electrical and Electronic Engineers
SWEBOK	Guide to the Software Engineering Body of Knowledge
ms	millisecond
CPU	Central Processing Unit
BPB	Blackpool Pleasure Beach
AT	Alton Towers

1 Introduction

1.1 Project Background

Rollercoaster enthusiasts (people who like going to theme parks and riding the rides repeatedly) often count the number of rides they have ridden for both personal reference and to compare with others. This is commonly referred to as a 'ride count'. This was traditionally done using notebooks with tally charts, spreadsheets or simple click counters. A selection of basic apps and websites have been launched that attempted to solve this issue of counting rides ridden in an compact and easy to use format, however they often had features missing. A few years ago, a basic mobile app was launched by the author (available on Android and iOS) that would allow the users to count the rides they go on and store it locally on the device. The app included a prototype of a feature that could recognise which rides you go on using GPS information such as the device's latitude, longitude, speed and altitude – as well as accelerometer data on some rides. This feature had to be started manually before going on each ride – which was cumbersome and time consuming to use.

To expand and improve the app – some users suggested having a web companion to the app that would allow users to store their ride count information remotely so that they could view it on a range of devices such as desktop computers. This would also allow for social media style account options where you can compare ride counts. There were also a range of other areas in the app that could be improved that were suggested by users, including: improved usability and user interface design to make the app easier to understand, and changes to the geolocation feature, so that it could run all day without interaction from the user.

For this project, a website was created that allows the users to upload their ride counts to an online account. Using the site they can then check to see their own ride counts and ride information for each trip, look at leader boards of who has ridden the most rides as well as

browse their own overall ride statistics (e.g. the tallest ride they rode last year). The website also includes the facility to edit the parks that they visit, for example adding new custom rides to an existing park or creating custom parks. This allows the user full control over what they would like to count in the app.

The project also includes a framework that allows the existing mobile apps created by the author to integrate into the website. These apps were substantially modified to allow them to connect to the site – this included modifications to how the data is retrieved on the phone, as well as substantial modifications to the user interfaces to add some of the new features. The project also includes the introduction of a remotely hosted ride list that the app pulls down from the website to use. This replaced the existing ride list that was hardcoded into the app, meaning whenever a park added or removed a ride, the app itself had to be updated which was slow and often resulted in an out of date ride list. The system created for the project also had to be designed and built with consideration to hosting and maintainability of the system, as the website will have to be suitable to be kept running for a number of years.

New features were also added to the system, this included the addition of a ride recommendation system that can recommend rides to users based on how they rate rides they have ridden. The system then looks for similar users and the rides that they have rated highly, which are then used to suggest a suitable ride to the user. Another new feature that was included in the system was the creation of the ‘Full Auto’ geolocation system. This allows the user to start the app running at the beginning of their day at a park, then by the end of the day the app will have calculated a list of the rides that the user has ridden. This feature makes use of new sections of functionality, as well as a modified version of the existing geolocation code that is already in the app, that allows the geolocation features to be run in the background of the phone without considerable negative impact on the user’s battery life.

The usability of all the features had to be carefully considered, as the system needs to have a similar look and feel across all of the platforms that the users may use. How the users interact with the system also needed to be considered, looking at the importance of useful help and information to allow the user to easily use all the features of the system.

1.2 Aims

This project aimed to create a structured framework for a web connected ride count system. This included a data driven website and suitable protocols to connect to the author's existing mobile apps. There were also substantial modifications to the apps to make them interface with the framework. New features were implemented in the system, such as the 'Full Auto' geolocation ride detection system, and a ride recommendation feature.

1.3 Objectives

1. Identification and prioritisation of user requirements and features. This could include MoSCow analysis etc.
2. Choose a software development design process (e.g. Waterfall, Agile)
3. Investigate and choose development languages and technologies for website and database.
4. Design of system using suitable design patterns to make the website well-structured, maintainable and secure. These designs will include relevant UML, database diagrams etc.
5. Research on user interface style – including some HCI investigations.
6. Research into recommendation algorithms for the ride recommendation system.
7. Implementation of system taking into account the completed research. This will consist of multiple parts:
    a. Database implementation using a suitable database management system
    b. Data service back-end
    c. Website front-end
    d. Mobile data service web back-end
    e. Recommendation system
    f. Modifications and addition of new web features to mobile apps
    g. Geolocation modifications to the apps
8. Creation and use of a range of testing strategies – this could include using black box testing, automated unit tests, usability testing, and overall acceptance testing.
9. Choosing and deploying to a suitable hosting platform. This includes research into the suitable platforms.
10. Production of the project report and presentation the finished system.

1.4 Report Structure

The rest of the report is structured as follows:

- Chapter 2 – Literature Review: Assesses and reviews existing academic work into the areas of recommendation systems, usability and geolocation as well as discussing the features and flaws of existing systems that are available.
- Chapter 3 – Design: Covers the choice of tools and techniques that were used in the implementation of the software as well as discussing the designs and plans for the system using a range of diagrams and presentation techniques.
- Chapter 4 – Implementation: Discusses the steps and decisions that were made whilst implementing the different elements of the software project.
- Chapter 5 – Testing: Reviews the rage of methods that are available for testing software and discusses those used on this project - and their results.
- Chapter 6 – Evaluation: Investigates the user survey that was used to measure how well the system met its aims.
- Chapter 7 – Conclusion: Assesses the project as whole, including personal views from the author. Finally, further work suggestions are given for what could be done to improve the system in the future.

2 Literature Review

This chapter investigates the research done to gain knowledge of the best practices of the three most important areas that this project aims to improve on based on the requests from the users. These areas are: the recommendation feature which automatically suggests rides to the users. The usability of the system, ensuring that the user easily navigate and complete tasks within the system without any issues. Finally geolocation, which is used by the improved mobile apps to identify where the user is and what rides they have ridden as part of an automatic system that requires no user input.

This chapter also looks at the existing ride count apps and websites that are currently available and discusses some of their positive attributes as well as what features they are lacking and how the system created for this project resolves these issues by creating a system that combines the best features.

2.1 Recommendation Systems

A recommendation system is a tool that can suggest items to the user based on what they might find useful, to assist them in making decisions (Ricci, et al., 2011). Recommendation systems can be found in a range of applications from e-commerce sites such as eBay to sites such as MovieLens which offers non-commercial movie recommendations (GroupLens Research - University of Minnesota, 2016).

A common example of recommendation systems can be found on online shopping websites such as Amazon. By using recommendation systems in e-commerce, businesses can help increase sales by converting browsers into buyers, increase cross sell and increase loyalty (Schafer, et al., 1999). Part of the system created for this project includes a ride recommendation system with the aim of helping to increase user engagement and encourage users to spend longer interacting with the site. The system for this project recommends rides,

taking into consideration how the user has rated the rides that they have already ridden and what other rides similar users have been on and rated highly. This is, as far as the author knows, the first example of using such a system to recommend theme park rides to users.

There are three main ways of creating a recommendation system: Collaborative filtering, content based and a hybrid of the two. Collaborative systems are often used by e-commerce sites such as Google Play and show recommendations based on the preferences and similarities with other customers (Sarwar, et al., 2000). Content-based filtering uses the individual user's preferences and description of items in the system to show recommendations and can be seen in sites such as the Internet Movie Database. (Pazzani & Billsus, 2007). Hybrid recommendation systems can combine the two previous recommendation systems to provide more accurate recommendations (Melville, et al., 2002).

For this system, a collaborative filtering recommendation system was chosen, as it would provide more genuine recommendations than ones that are based on ride statistics using a content based system. This is because some rides, although having very similar statistics, are very different rides in reality and would not make suitable recommendations. For example, Grand National and Nickelodeon Streak at Blackpool Pleasure Beach have very similar statistics (similar age, height, speed etc.) however due to track profiling, train design and other features, Grand National is recognised as a 'white knuckle' thrill ride whereas Nickelodeon Streak is known as a family rollercoaster in the children's area of the park. A recommendation system that used content-based filtering may deem these two rides to be similar and generate an unsuitable recommendation. Whereas a collaborative based system that takes into account the user's rating of a ride and matches the user with other users who

have rated rides in a similar way, then generating recommendations based on rides that the other users have rated highly. This recommendation system is more appropriate for this project.

The speed at which a recommendation system can generate recommendations is a potential issue when working with large amounts of data. This is can be seen on sites such as Amazon, whose recommendation algorithms are focused on working with a large number of users and products whilst mainlining quick update speed when a user changes their data, for example purchasing an item (Linden, et al., 2003). This highlights the importance of regularly updating the user's recommendations when a change occurs that could affect them.

A key issue with generating recommendations regularly is that it could use large amounts of CPU and processing resources. One way around this, demonstrated by YouTube, is to generate the recommendations in batches then store the data. This allows the recommendation algorithm to process large amounts of data for the recommendations without monopolizing resources. This can then be presented to the user quickly as the data has already been generated (Davidson, et al., 2010). Another issue discussed here is the idea of keeping the recommendation system separate from the main site. This not only makes debugging and maintaining the code easier, but allows for the system to fail in a graceful way should another part of the system fail. In the system created for this project, the recommendation system generates recommendations live when the user requests it. This may have to be changed as the number of users grows and traffic increases, or when the recommendations are deemed to be too slow.

Some recommendation systems (such as the ones in YouTube and Amazon) provide information as to explain why an item has been suggested for them. This extra transparency in a recommendation system can help users trust the system more (Sinha & Swearingen, 2002). However it is suggested that although recommendation systems have been useful for entertainment purposes – such as YouTube, the systems are not often used for more important things. It is suggested that the lack of transparency in systems could prevent users from trusting systems with recommendations on bigger things – such as a honeymoon destination (Herlocker, et al., 2000). This shows that providing an explanation of why a recommendation system has chosen something is important, as it will help the user to trust the system and the recommendations it gives. For the system created for this project, a brief explanation of how the system works has been provided to the users on the recommendation page of the website, to allow them to gain an understanding of why a ride might be recommended to them. The majority of the users that participated in the user evaluation survey agreed this made them feel like they could trust the recommendation system.

2.2 Usability

Usability is the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (British Standards Institute., 1998). It is important to ensure that the final users find the system easy to use. The system created for this project consists of three main parts that the user can interact with. These are: the mobile app on Android, the mobile app on iOS and the website. Each of these platforms has varying requirements and standards for usability.

2.2.1 “Eight Golden Rules of Interface Design”

Ben Shneiderman proposes that there are "Eight Golden Rules of Interface Design" (Shneiderman, et al., 2010). These provide some guidelines that can be followed when considering the usability of a computer system. They are as follows:

1. *Strive for consistency.*
2. *Enable frequent users to use shortcuts.*
3. *Offer informative feedback.*
4. *Design dialog to yield closure.* (This is the idea that a user can get a sense of achievement)
5. *Offer simple error handling.*
6. *Permit easy reversal of actions.*
7. *Support internal locus of control.* (Make users feel like they are in control)
8. *Reduce short-term memory load.*

These cover many important aspects, which were considered during the design and implementation stage of the project. For example, rule 2 – “Enable frequent users to use shortcuts”. This is difficult to do on the mobile apps where space is limited, but the web system contains shortcuts (for example on the logged in home page) to make things easier for the users. Rule 3 - “Offer informative feedback” and rule 5 “Offer simple error handling” link together when error messages are shown to the user. The messages should contain useful information about how to recover, and are coloured based on their severity – with red being the most severe. Some of the rules, for example rule 6 “Permit easy reversal of actions” have been considered in the system, for example – delete buttons will show a confirmation box first before deleting items. Two of these rules (1 and 8) stand out as being particularly relevant to this project, and have been covered in more detail.

2.2.1.1 "Strive for Consistency"

The first of the eight golden rules is to "strive for consistency". This will enable the users to have an understanding of how to use a system because they know how other screens will look. It is difficult to provide this consistency when different app platforms have a range of user interface approaches and conventions that should be adhered to (Leroux & Charland, 2011). Therefore, although the features and information that the system provides may be similar throughout the platforms, the way it is presented may need to be altered to suit each platform. These will be specific to the individual platform. For example, the Material design guidelines (Google, 2016) which show the best practices for creating a usable design on Android – and can also be used on the web. This includes guidelines on things such as colour, typography, motion and layout choices. These rules were created to help standardise mobile apps and websites. Apple's iOS Human Interface Guidelines feature similar demonstrations on how to create designs for their mobile platform that the users will find easy to use and provide a good experience, as well as maintaining consistency with existing iOS apps (Apple, 2016).

2.2.1.2 "Reduce Short Term Memory Load"

An example of how a mobile platform increases the importance of Shneiderman's interface design rules is point eight – "Reduce short-term memory load" (Shneiderman, et al., 2010). This is especially important on a mobile device more than a desktop device – as the user may be busy with other things (Tarasewich & Gong, 2004). This is a key consideration for this project, as the user will be at a theme park - therefore they will want to enjoy themselves in the park. This will take precedence over their interactions with the ride count system, therefore the screens used should be simple and not overly complicated – containing only essential information, with the most important at the top where it is easier to find.

2.2.2 Intuitive design

It is very important that all aspects of the design are easy to understand and use. Steve Krug suggests that if a design is not obvious to a user, it could cause users to lose confidence in the site – if the designers didn't make it obvious how to use the site, then they don't care (Krug, 2006). This is an important consideration as the product created for this project stores information from the users, therefore it's crucial that they feel that the site and apps are trustworthy and secure.

2.2.3 Existing users

As this project will include the modification of existing mobile apps, it is important that the existing design of those apps are considered when altering them for this project. Jakob Nielsen says that when working on usability, it is a good idea to test the existing design. This will highlight the good and bad points that should be kept or removed (Nielsen, 2012). It is important that the changes to the app are done in such a way that the user does not have to re-learn how to use the pre-existing features of the app. Instead the new features should be additional to these. This has been an important consideration when making modifications to the apps, as there are users who already know how to use them. For example, adding new items to tabs with similar pre-existing screens.

2.3 Geolocation

This project includes the expansion and upgrading of an existing app that includes prototype geolocation features to identify which theme park a user is at. The app can then at selected parks, identify which rides the user has been on using geolocation data. This data is processed on the device and compared with a range of information about the ride that has been collected by the author. This project includes the creation of a 'Full Auto' system. This is where the user can start the app running at the beginning of the day, then by the end of the day they will have a full list of the rides that they have ridden. This means that the user will not have to

open the app to manually add rides throughout the day, meaning that they can enjoy their day at the park without the need to constantly use their phone to manually record the rides they go on. This was an important aspect of user feedback from the pre-existing prototype geolocation features that were already included in the app. This option needed to be started by the user manually before each ride, which was time consuming and cumbersome. The creation of the 'Full Auto' feature for this project overcomes these issues and provides a much improved user experience. The app is currently the only app to have a geolocation facility available to identify and count the rides that a user has ridden at a range of parks.

Some parks themselves use, or have used systems in the past that can keep track of the rides that people visit throughout the day for a range of purposes. For example the YourDay system at Alton Towers used RFID wristbands to work out where the riders were, then film them on the rides for a souvenir DVD (Towers Times, n.d.). Blackpool Pleasure Beach uses barcoded wristbands and barcode readers to allow or deny riders access to the rides (Blackpool Pleasure Beach, 2017). However these systems are controlled by the parks and are not useable by the public for getting their personal ride count data. Therefore a solution using GPS in a mobile app that is not integrated with the park's technologies was devised.

Many mobile apps utilise geolocation for a range of uses. This can include mobile tracking for marketing - which has been implemented in many apps. In a recent survey, 66% of marketing professionals consider location based advertising the most exciting mobile opportunity in 2016 (Internet Advertising Bureau UK, 2016). Location data can also be used in social networks, such as the 'check-in' feature on Facebook – where the user can tag themselves at a location such as a hotel or restaurant. This allows users to recommend places to others and get information about a place whilst there (Phelan, et al., 2013). Whilst a similar feature for getting information about nearby rides was not part of the scope of this project, it is something which could be considered for adding at a later date in future work.

Geolocation can also be used in other ways in apps. Using location data from phones and a recommendation algorithm, an app has been created to recommend social events that are near to a user (Crowcroft, et al., 2010). As location data for some of the rides will be held in the system for part of this project, future work could include using the ride location when recommending a ride. For example, the rides that a user is given as recommendations could be sorted by distance relative to the current location of the user.

There are however downsides to location tracking usage in mobile apps. Privacy is a major concern for many users. It is important to inform users when their location is being used, and allow them the option to opt out if necessary. It is also important that the data collected is kept secure (DeLuca, 2015). Both of these points were considered in the design of the mobile apps in this project, with all location processing done on the device with no location data being transmitted from it. The apps also follow the operating system's methods and guidelines for gaining access to the user's location.

It is also important to consider the battery usage of geolocation systems. GPS chips in phones can be very power intensive, so other solutions such as assisted GPS can be used to reduce battery consumption. Assisted GPS is where network location is used to assist the GPS chip (Djuknic & Richton, 2001). An example of an implementation of this can been seen in Google's location APIs such as the fused location provider which has low power modes (Google, n.d.). This location provider uses the APIs in Google Play services and therefore is more battery efficient. This was selected to be used as part of the 'Full Auto' system on Android, as it would allow the user's location to be used throughout the day without severely impacting their battery life.

2.4 Existing ride count systems

Counting the number of rides that have been ridden is commonly done by many rollercoaster enthusiasts. Some enthusiasts use a notebook to write down the number of rides that they have ridden whilst at the park. Other enthusiasts use Excel spreadsheets to track the rides when they get home (Weisenberger, 2016).

Since the increase in popularity of smartphones and the internet, a number of systems have become available to keep count of the rides that the users go on. These have a limited range of features available, which may restrict the user and make the tool less useful. The most notable of these existing ride count systems have been selected for review in this section.

2.4.1 Coaster Counter (Android app) (Nyphoria, 2016)

Coaster Counter can keep track of the number of times a user has ridden each ride in total. There is no differentiation between when a ride is ridden – so it is not possible to see for example, how many rides were ridden on a specific date. A good feature of this app is that it can show statistics about the rides that the user has ridden, the app then adds these statistics up for the number of times the ride has been ridden – giving totals such as the total time spent on the ride (Figure 1).

Coaster Counter does not support custom parks. This is where the user can create or edit a park and edit the list of rides that are available there. This means that users of this app cannot create a park that may be missing from the app, or update the ride list at a park. Coaster Count also currently only supports Alton Towers, this could be a disadvantage for users as there are parks that are more popular than Alton Towers that they may want to count rides at. Attendance figures from 2015 shows that many more parks are more popular around the world with Alton Towers being the 12th most visited park in Europe during 2015 (Themed Entertainment Association (TEA), 2016). Therefore by supporting more of the popular parks on the list, the app could potentially gain more users. Another way of gaining more users

would be to support more platforms. Currently this app is only available for Android which could put it at a disadvantage, because fewer users can access the app.

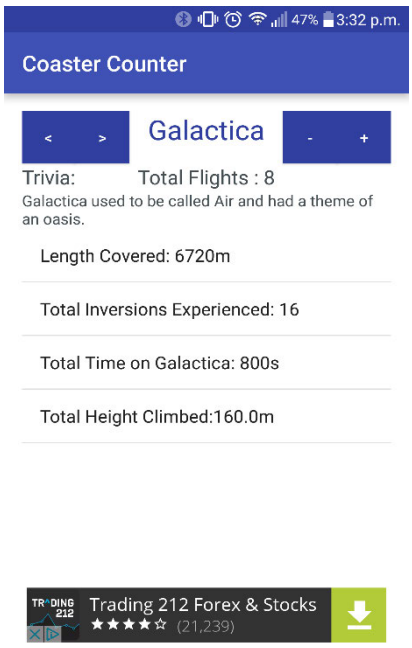


FIGURE 1 - SCREENSHOT OF COASTER COUNTER SHOWING RIDE STATISTICS

2.4.2 Ridecount Worldwide (iOS and Android apps) (Theme Park Guide, 2015)

Ridecount Worldwide is a cross-platform mobile app which, unlike the previous app, contains many parks around the world. It also offers paid features such as advert removal, unlocking more parks and custom parks. This app is available on both iOS and Android unlike the previous app which was Android only. These two platforms are the most popular mobile platforms – with Android having a 87.6% usage share and iOS having 11.7% worldwide (IDC, 2016). This app however does not offer the feature for users to transfer their data across their devices using a feature such as a login system.

2.4.3 Coaster-Count.com (website and iOS app) (Sauer & Thumann, 2014)

Coaster-Count.com is a website with iOS companion app is specifically for counting and keeping track of rollercoasters – therefore does not offer the feature of tracking other rides such as water rides and spinning rides (for example Sky Force at Blackpool Pleasure Beach).

This may be a disadvantage if the user wants to count all the rides they go on, not just the rollercoasters.

Unlike the previous apps, this website has user accounts which allows users to access their data from any web browser. The site also has an iOS app allowing users to access their accounts whilst on-the-go. This is important with the current increasing trend in mobile usage. According to a study in 2014, over 60% of digital media time is spent using mobiles and tablets in the US (Lella & Lipsman, 2014). Therefore it is important that websites have a mobile friendly site and possibly a matching mobile app. This is especially important as many rollercoaster enthusiasts may wish to access their accounts on their phones whilst at parks.

The website also has leader-boards and ranking so you can see who has been on the most rides. This competitive feature could help to increase user engagement, making them want to spend more time using the app or website (Shackelford, 2016).

2.4.4 ridecount.com (online only) (park.expert, 2007)

Users of ridecount.com can log the rides that they go on in a trip based format - meaning that they can see the total times they have ridden each ride per visit as well as per year and an overall total. This is unlike Coaster Counter which only offered a total overall count.

Like coaster-count.com, there is a social aspect where users can be featured on a leader board based on how many rides they have ridden. This can be filtered to monthly, weekly etc. (Figure 2). Leader-boards allow the users to compare and rank themselves with others (Zichermann & Cunningham, 2011). This could help engagement as users may want to try to beat others to rise up the leader-boards, therefore increasing the usage of the website.

This website however does not have a mobile app – therefore it is not very easy to use when at a park. With recent statistics showing that 83% of US theme park guests owning a smart

phone (Lennox, 2016) it is important to utilize mobile apps to make a product easier to use whilst on the go.

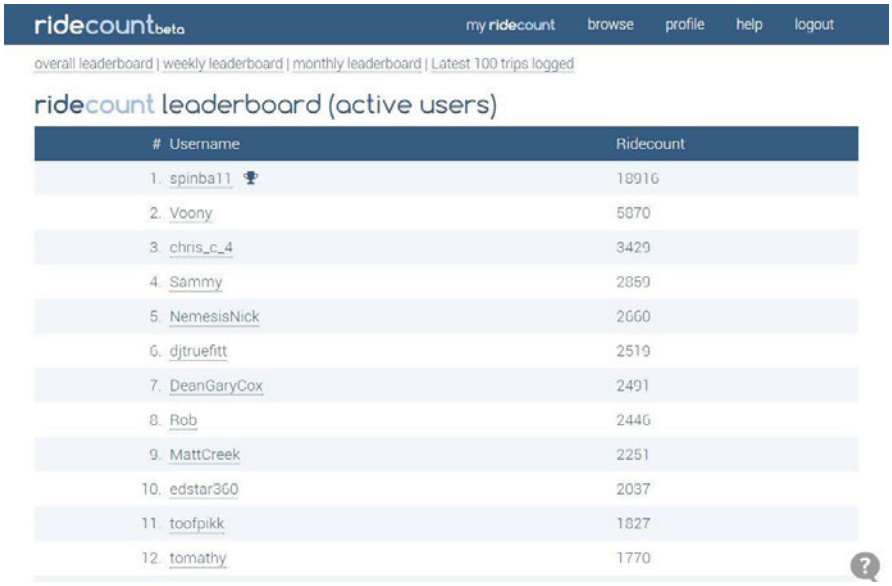


FIGURE 2 - THE OVERALL LEADER BOARD IN RIDECOUNT.COM

2.4.5 Comparison of Existing Systems

Table 1 shows a comparison of the systems discussed in this chapter. The features that are included in the existing mobile apps have also been added in the penultimate column. The last column shows the features that the system created for this project includes.

Feature	Coaster Counter	Ridecount Worldwide	Coaster-Count.com	ridecount.com	Existing system	System for this project
Wide selection of parks & rides – including custom parks & rides		✓		✓	✓	✓
iOS, Android and web						✓
Ride statistics	✓					✓
Leader boards & ranking			✓	✓		✓
Accounts to access data anywhere			✓	✓		✓
Ride recommendation						✓
Automatic, all day, ride recognition with geolocation					Not all day	✓

TABLE 1 – COMPARISON OF EXISTING RIDE COUNT SYSTEMS AND FEATURES

There is currently a range of apps and websites available for counting the rides that people go on, all with a variety of features. The existing products however were lacking in functionality, or only have a limited range of features available. This project aimed to improve on these limitations with the inclusion a wider range of features, as well as new services that are unique to the market. A key feature of the product created for this project was the use of user accounts to allow the accessing and logging of statistics across multiple devices such as iOS and Android apps as well as the web. Some of the existing apps and websites use parts of these features, however the system created for this project is the only one that will be accessible on Android, iOS and as a website.

Only one of the existing apps included ride statistics. This project also included ride statistics, which is an interesting tool to engage the user, making them want to spend more time using the system. Leader boards and user rankings will also help increase user engagement. Other key features such as a wide selection of parks and rides, and the ability to create custom parks

and custom rides enable the users to use the app at any park they desire. This could also make this project appeal to a wider range of users.

The system also includes some unique features that have never been used before in similar systems. This includes a ride recommendations feature, where users can get calculated recommendations of rides they might like. The system in this project also introduces the 'Full Auto' geolocation feature, where the app can generate a list of the rides a user goes on automatically over the day - a feature that is unique to ride counting apps.

2.5 Summary

This chapter covered the investigation that was undertaken to gain insight and a greater understanding into key areas of the project, as well as the range of existing features that are already available to users. The results of this investigation allowed for the design and development of a counting solution which made use of a research driven design to achieve a high quality end result.

3 Design

This chapter covers the choices of tools and techniques that were used during the implementation of the project as well as the reasoning why these were chosen. These include the choices of languages and frameworks for the mobile and web and how this effects the hosting platform choices. The database system also had to be selected, which was again related to the language and framework choices. Different development methodologies were considered, ensuring that they would be suitable for the way the project was going to be developed. The way tasks were going to be planned has also been investigated, as this would ensure that the design and development of the solution could be done effectively.

This chapter also covers the plans for the implementation of the project, including its structure, content and layout. This includes overall system architecture diagrams, use case diagrams and use case specifications that illustrate how the user interacts with the system. Database design and website structure diagrams were also produced to allow the development to go smoothly. Finally, interface design mock-ups for the website and mobile apps have been produced that take into consideration the design decisions that were affected by the research in the previous chapter.

3.1 Language Choices

3.1.1 Mobile

The mobile apps for this system was written in a derivative of VB that has been implemented in a range of cross platform development tools including B4A (Basic4Android) and B4i (for iOS). These tools allow for the development of applications on a range of platforms, whilst being able to share sections of code. The B4A and B4i IDEs (Integrated Development Environments) both compile the apps down to the respective native language for each

platform allowing the resulting application to run efficiently making optimal use of the phone's resources.

This tool has been chosen over developing apps natively as it allows code to be re-used between the two platforms, making development faster. B4A and B4i were chosen over other cross platform methodologies such as web apps developed using HTML5 as the apps that are produced by B4A and B4i are native, allowing them to be more efficient than HTML5 web apps. They can also integrate into the host operating system better, and make full use of the operating system's tools and APIs. An example of this is the requirement that the geolocation features must be able to run in the background of the user's phone whilst the phone is locked and securely inside their pocket whilst on a ride. This cannot be achieved with a HTML5 web app, however it can be achieved with native code – like the code produced by B4A and B4i. Also by using native libraries, such as the fused location provider mentioned in the previous chapter, a more battery efficient geolocation solution can be created.

3.1.2 Web

There is a range of different technologies that can be used when creating websites. This includes a range of languages such as PHP and Java, to the range of frameworks available such as MVC. For this project, C# was chosen as when coupled with Visual Studio (the IDE from Microsoft) it allows for rapid and efficient development of applications. Visual Studio also has a range of powerful debugging solutions for C# that are not as easily available in languages such as PHP – such as breakpoints.

There are currently 4 Microsoft ASP.NET programming models that can be used. These include: MVC, Web Pages, Single Page Applications and Web Forms. For this project, Web Forms has been selected as they allow for good separation between the front end code, and the logic code – which makes code more reusable, easier to debug and easier to maintain.

Web forms also have a large range of support and documentation available – which makes writing and troubleshooting the code easier.

3.2 Database Choices

MySQL has been chosen as the RDBMS (Relational Database Management System) for this project. Like other relational database systems, different tables are used to store data in. These can then be linked together if necessary when queried. Interaction with MySQL is done using SQL (Structured Query Language). MySQL is used by some larger companies such as Facebook and Flickr and has a wide following in the developer community resulting in a large amount of support and other resources being available for assistance in using the tool. It is also widely used due to the fact that it is open-source and free – and is therefore supported by many hosting platforms, often very cheaply. This wide availability is one of the main reasons why it has been chosen for this project.

Microsoft SQL Server is an alternative RDBMS created by Microsoft. Like MySQL data is stored in tables and can be queried using SQL. Both solutions offer tools to assist in the maintenance and running of the database such as backup solutions, GUI (Graphical User Interface) tools to access the database, solutions for concurrent data access etc. However, for this project, the wide availability of MySQL makes it more suitable that Microsoft SQL Server.

3.3 Hosting Platform

Due to the languages chosen for this project, the hosting the web side of the system will be on a Windows server. Although it is possible to use tools such as Mono to host ASP.NET sites on Linux, the extra complexity and setup time makes it unfeasible for this project. AWS (Amazon Web Services) offers a range of cloud computing solutions – including their

flexible server solutions. Due to the scale and popularity of AWS, it is often marketed as being a cost effective solution for hosting sites. A small Windows server can be commissioned for free for the first year of use, therefore it is the ideal choice for this project as it allows the initial setup and experimentation that will occur during development and testing to be done for free.

3.4 Development Methodologies Choices

3.4.1 Waterfall

The waterfall development model, as shown in Figure 3, is a progression of different stages that follow on from each other. This is a well-used model when developing software as it clearly defines each task and the order that they should be done in. This development methodology makes it more difficult, for example, to change the requirements of the product once the implementation stage has been completed. This can be seen as a bad thing, especially in the current climate where software and updates are released frequently.

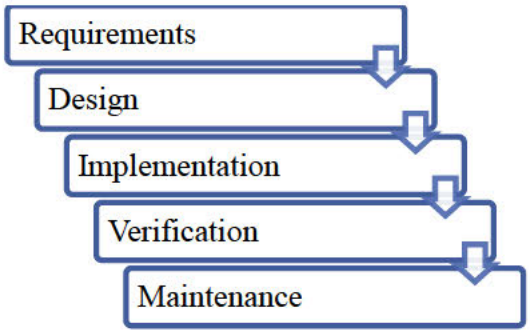


FIGURE 3 - THE WATERFALL MODEL STAGES

3.4.2 Agile

An alternative to the waterfall methodology is agile. Often in this model, shorter iterations are completed where smaller changes are implemented and tested. This allows the product to be flexible for adaptations that could change over time.

For this project, a ‘hybrid agile’ methodology was followed. This is a hybrid between Agile and Waterfall. Each product in the system will have to be reactive to any changes that might occur in the others, therefore this element from agile is important. However for each element of the product, the progression and format of waterfall was followed. This allows for a proper structure for the creation of the product.

From a more code specific view, a TDD (Test Driven Development) style was followed for the development of the central parts of the system involving core logic. This area has a large number of unit tests to ensure it functions correctly, and therefore a TTD style was used. This is where the tests are written first, then the code itself is written and modified until the tests pass. This ensures a higher code quality for the central parts of the system that is shared across the different parts of the system.

3.5 Task Planning Methods

It is important that all the individual tasks for this project are planned in advanced, as this allows for a consistent flow of work during the implementation process. By sizing tasks, it is possible to work out an estimate of how long each task will take relative to each other, and therefore how many tasks can be finished over a period of time (e.g. how many tasks can be completed per week). This helps to maximise the amount of tasks completed overall. Tasks were sized using Fibonacci numbers (1, 2, 3, 5, 8 etc.). This is commonly used in ‘Planning Poker’, a way of estimating the size of tasks in software development. Fibonacci number are useful for sizing tasks because they demonstrate the difficulty in sizing tasks that are larger in scale.

The tasks for this project were also prioritised using MoSCoW prioritisation (Must do, Should do, Could do, Won’t do). This means that throughout the development of the project,

it was clear which features were the most important. This links into an agile strategy, as the tasks that are for example classed as ‘Should do’ might be done after all the ‘Must do’ tasks have been completed and tested.

All the tasks were tracked on a Kanban style board in Trello. This was a good way to visualise how the tasks were progressing and would ensure that tasks did not get forgotten, whilst also maintaining a steady workflow.

3.6 System Design Diagram

Figure 4 shows how the different parts of the system link together. On the left are the three different platforms that the user will use. Going left to right, the app code is shown next. This is the code that is written in VB and will be downloaded from the relevant app stores by the user to run natively on the user’s device.

The next section for the apps is a mobile data service link. This runs on the server, and acts as a link between the apps and the logic layer. This aims to resolve issues with versioning between the apps and the logic layer where a user may not update the version of the app that is running on their device, however the logic part of the system may have been updated. Running parallel to this is the web presentation layer. This is the code that displays the data onto the website and also includes the front-end code that the user will interact with.

The next part of the system is the logic layer. This is the bulk section of the code, containing all the logic for the system shared across the apps and the website. Next to this is the database connection which connects to the database. The next section is the database itself, which will hold the data and the necessary procedures for fetching, updating and inserting the data.

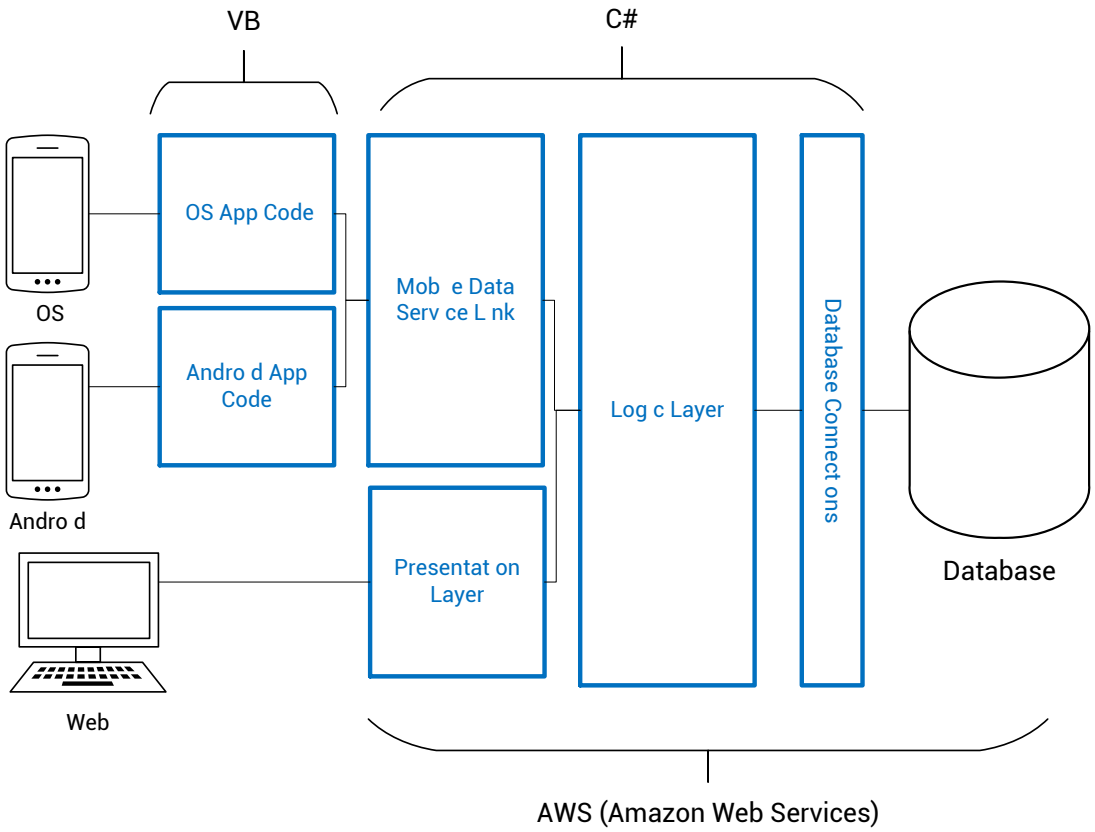
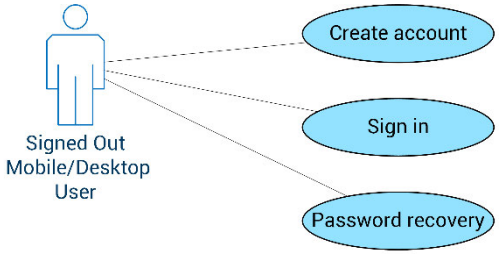
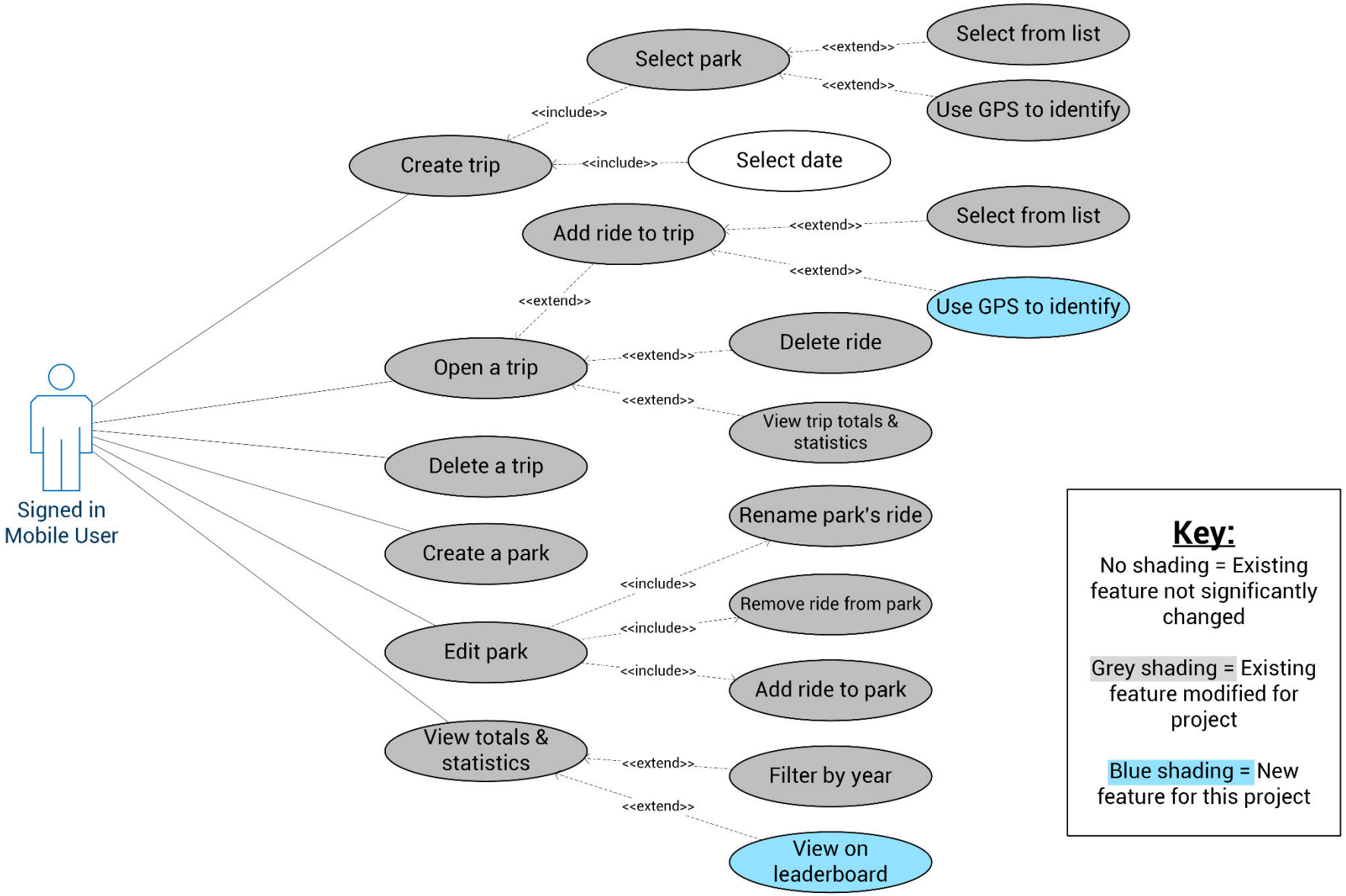


FIGURE 4 - SYSTEM DIAGRAM SHOWING THE DIFFERENT CODE BASES

3.7 Use Case Diagram

A use case diagram shows the actions that each actor (a user) can do. For this system, I have three users - a Signed out mobile/desktop user (as their actions will be the same on each platform) and a signed in user for each platform. This assumes that they have already logged in. This diagram has been spread across three pages due to its size, however it is all part of Figure 5.


Create account
Sign in
Password recovery
Signed Out
Mobile/Desktop
User


Signed in Mobile User
Select park
Select from list
Use GPS to identify
Create trip
Select date
Add ride to trip
Select from list
Use GPS to identify
Open a trip
Delete ride
View trip totals & statistics
Delete a trip
Create a park
Edit park
Rename park's ride
Remove ride from park
Add ride to park
View totals & statistics
Filter by year
View on leaderboard
<<extend>>
<<include>>
Key:
No shading = Existing feature not significantly changed
Grey shading = Existing feature modified for project
Blue shading = New feature for this project

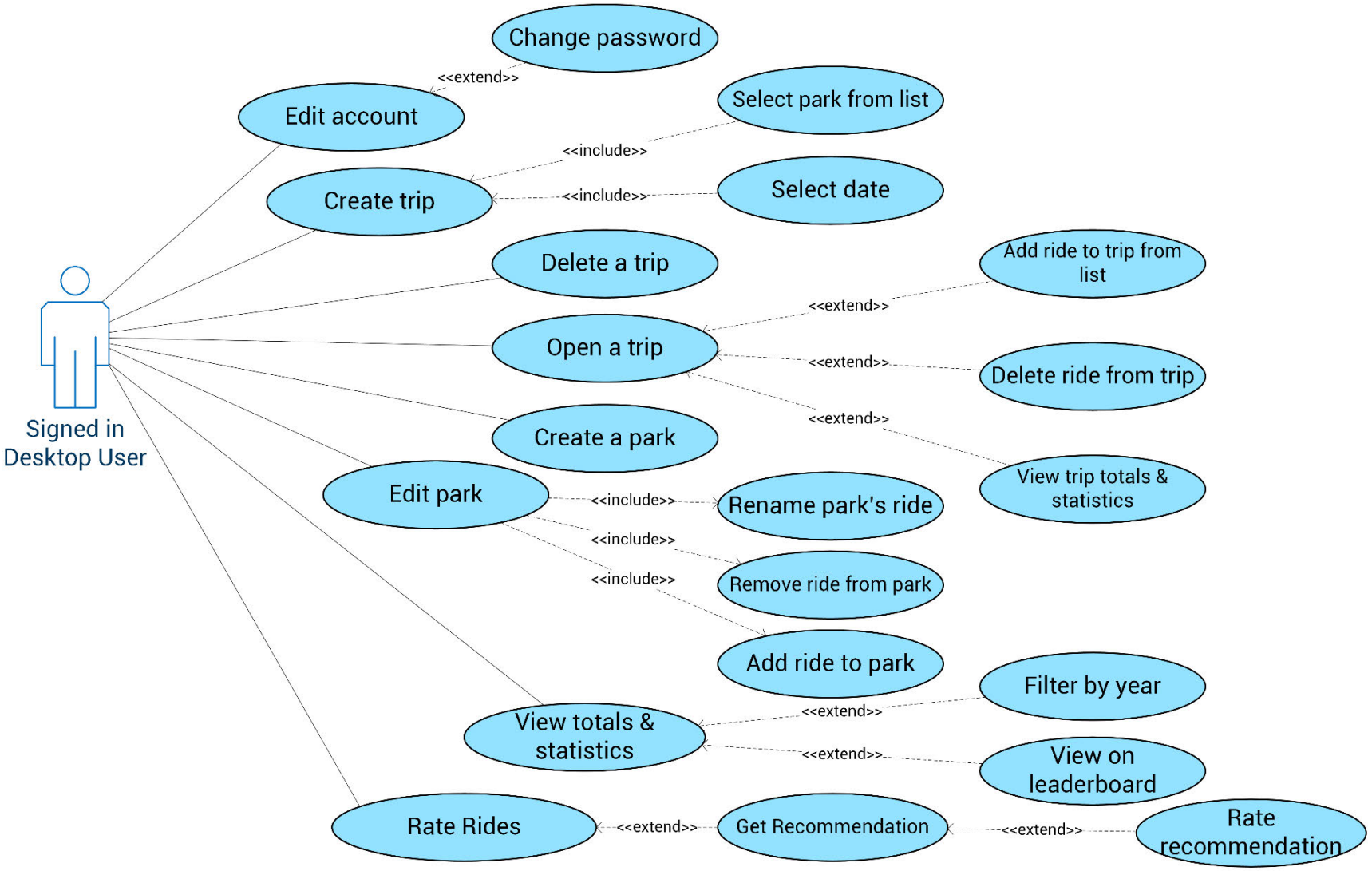


FIGURE 5 - USE CASE DIAGRAM FOR THE WHOLE PRODUCT

3.8 Use Case Specifications

For specific parts of the system that contain a more complex set of paths, use case specifications have been created to aid in the development of the system.

3.8.1 Create Account

Pre-Conditions

- User does not already have an account

Post Conditions

- Account is created in the database and the user is registered
- Failure condition - the user has not been registered

Primary Path

1) The user clicks the link to register
2) On the register page, the user enters their selected username, email address and password (twice). Front end validation is run as the user types.
3) User clicks the button to register.
4) Server side validation checks are run.
5) The user is created in the database.
6) The user is re-directed to the trips page

Alternative Path

- **Username is taken:** If stage 2 or 4 on the primary path fails because the user's input for their chosen username is already taken, the user will be prompted to choose a new username.
- **Invalid email address:** If stage 2 or 4 on the primary path fails because the user's input for their email address is invalid, the user will be prompted to re-enter their email address.
- **Passwords do not match:** If stage 2 or 4 on the primary path fails because the user's input for their passwords do not match, the user will be prompted to re-enter their passwords.
- **Any part of the form is incomplete:** If stage 2 or 4 on the primary path fails because the user has not entered any data for any of the fields, the user will be prompted to enter data into the fields they left blank.

3.8.2 Login

Pre-Conditions

- The user has an account
- The user is logged out (if they are logged in, the 'Log In' button will be changed for a 'Log Out' button.

Post Conditions

- The user is logged into their account
- Failure condition - the user does not gain access to their account

Primary Path

1) The user clicks the link to log in
2) On the log in page, the user enters their username and password. Front end validation is run to check that the user has typed something in the boxes.
3) User click the button to register.
4) Server side validation checks run, system checks to see that the username and passwords match.
5) The user is logged in and is re-directed to the trips page

Alternative Path

- **Username or password do not match:** If stage 4 on the primary path fails because the user has answered the wrong username or password, the user will be shown a message saying that the logging in was unsuccessful.
- **Any part of the form is incomplete:** If stage 2 or 4 on the primary path fails because the user has not entered any data for any of the fields, the user will be prompted to enter data into the fields they left blank.

3.8.3 Creating a custom park (mobile and desktop)

Pre-Conditions

- The user is logged in
- The park the user wants to create is not already in the app/website

Post Conditions

- A park is created in the database for the user to use
- Failure condition – the park is not created

Primary Path

1) The user selects the option to create a custom park.
2) The user types the name of the custom park and presses a 'submit' button.
3) The data that the user has typed is validated, then the database is checked to see that the park has not been created before.
4) The park name is abbreviated (taking the first letter of each name then converted to upper case) then prefixed with the term 'CUST-'. A hyphen and number is added to the end which is incremented if the abbreviation has used before. For example if the user chooses to add a custom park for 'Blackpool Tower', the abbreviation will be: 'CUST-BT-1' if they have not added a park with the abbreviation of 'BT' before.

Alternative Path

- **The park already exists:** If stage 3 on the primary path fails because the user has entered a park that is already in the system, the user will be shown a message saying that the park already exists.
- **The user does not enter any characters into the park name box:** If stage 3 on the primary path fails because the user has not entered any characters into the park name text box, the user will be prompted again to enter a park name into the box.
- **The user has entered an invalid character:** If stage 3 on the primary path fails because the user has entered a reserved character (commas, hyphens, slashes, tilde characters, vertical bars, negation or asterisks) into the text box, then they will be informed which characters they cannot enter and will be asked again for the park name.

3.8.4 Adding a custom ride to a park (mobile and desktop)

Pre-Conditions

- The user is logged in
- The user is editing a park
- The ride is not already listed on the park's ride list

Post Conditions

- The ride is added to the park's ride list in the database
- Failure condition – the ride is not added

Primary Path

1) The user selects to add a custom ride
2) A box is shown for the user to type the name of the ride into
3) The user types a ride name and presses the 'submit' button.
4) The entered text is validated and checked to ensure that the ride is not already on the park list.
5) The ride is added to the ride list for the park in the database

Alternative Path

- **The ride already exists:** If stage 4 on the primary path fails because the user has entered a ride that is already on the park list, the user will be shown a message saying that the ride already exists at the park.
- **The user does not enter any characters into the ride name box:** If stage 3 on the primary path fails because the user has not entered any characters into the ride name text box, the user will be prompted again to enter a ride name into the box.
- **The user has entered an invalid character:** If stage 3 on the primary path fails because the user has entered a reserved character (commas, hyphens, slashes, tilde characters, vertical bars, negation or asterisks) into the text box, then they will be informed which characters they cannot enter and will be asked again for the ride name.

3.9 Database Design Diagram

Figure 6 shows the design of the tables used in the system, and how they are linked together.

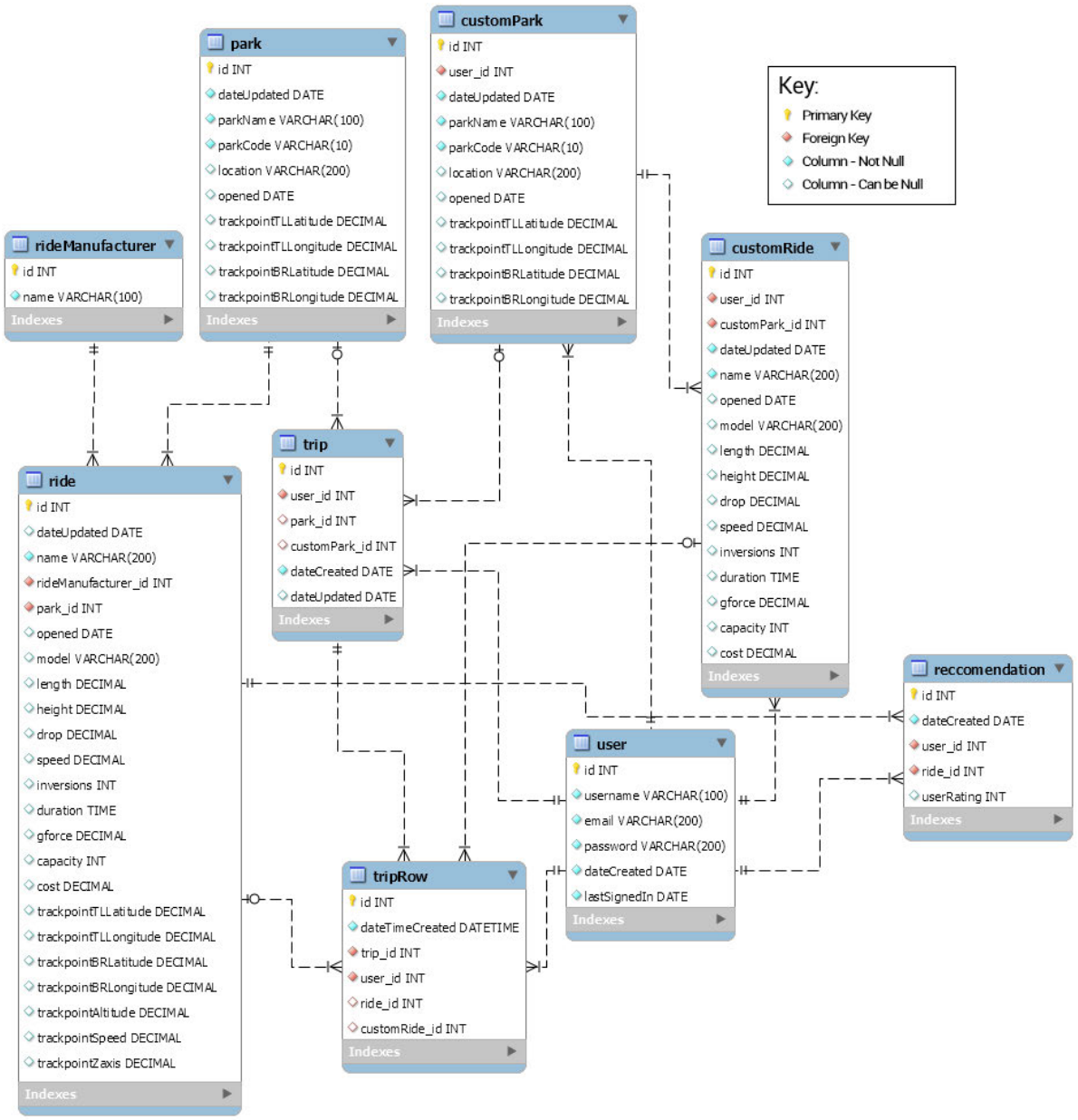


FIGURE 6 - DATABASE DIAGRAM AND KEY

3.10 Web Structure

Figure 7 shows the structure of each separate web page and how they are linked from other pages. All of the pages on the second line are on the navigation bar or footer and will therefore be accessible from each other. Some of the lower level items may not be separate

pages but may be additional pages shown as pop-up light boxes – for example adding a new ride. The pale blue shaded items represent pages that can only be viewed when logged out, the non-shaded items represent pages that can only be accessed when logged in. Grey shaded pages can be accessed at all times, regardless of logged in status. The home page is shaded in a blue/grey cross because it can be accessed at all times, but alters depending on if the user is logged in or not (e.g. the sign-in button is only showed when the user is not logged in.)

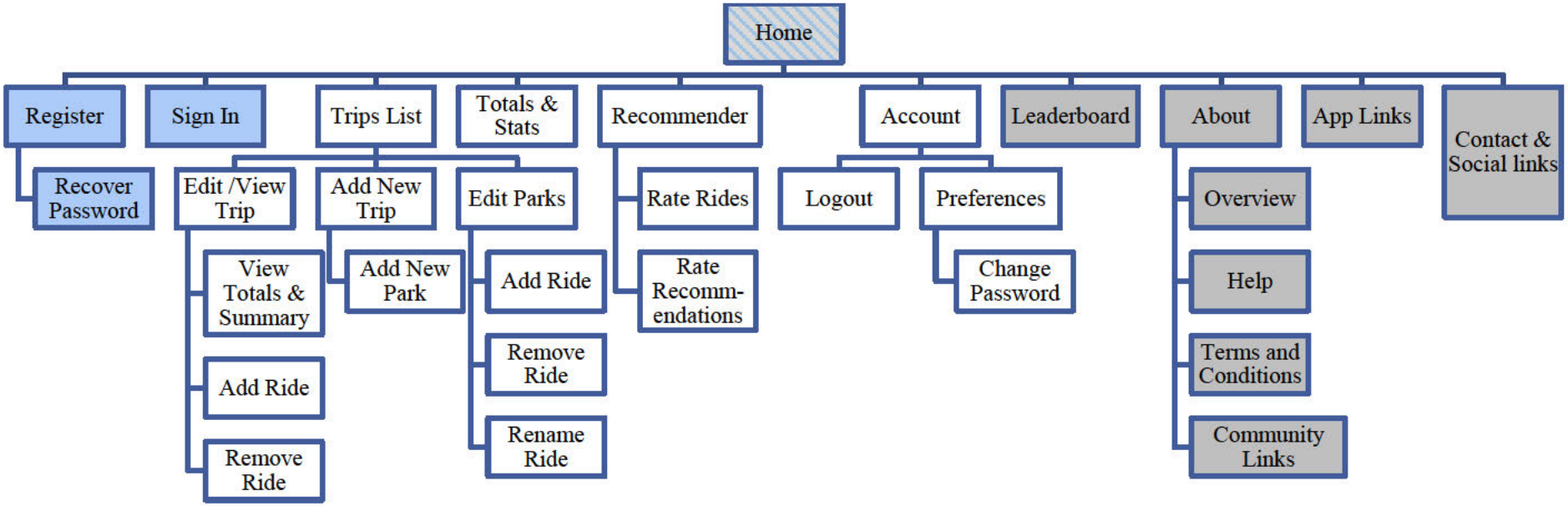


FIGURE 7 - PAGE STRUCTURE FOR THE WEBSITE CREATED FOR THE PRODUCT

3.11 Interface Design Mock-ups

3.11.1 Web

Bootstrap was chosen as the framework for the front end design of the website. This makes the design and the layout of the website easier as it will provide templates and tools to make a responsive web design. The prolific use of Bootstrap across various websites will also ensure that the website is easy to use and navigate as parts of it may feel familiar to users, which is important as it will reduce the user's short term memory load – a key consideration from the usability research.

Bootstrap also has a range of colours and styles used for items such as buttons, error notifications etc. that allow the severity of the action to be conveyed. For example a built in 'alert-danger' class by default makes an alert message display in red, showing that it is important to the user. This also links into the usability research.

3.11.1.1 Standard layout

These navigation elements of the website are the same for all pages. This familiar and consistent layout will make the website easier to navigate, a key part of the usability research. Figure 8 shows the design of the navigation. There is a main navigation bar at the top of the page. This contains the logo in the top left – which will link to the home page. The other links are distributed across. A drop down box will appear under the account link when the user clicks on it. This provides a range of links that are related to their account. Grouping these links together ensures that the user understands that these links are all part of their account management, making them easier to find and understand. The links on the navigation bar will change where necessary to reflect the user's logged in status (for example, if the user is not logged in, the links are removed, and 'Sign In' and 'Register' links are added instead).

The footer of the page contains easy to find links to the leader board and information pages as well as download links for the mobile apps and links to social networking sites. The links in

the footer are not as important as the links in the header and consequently will be used less by the users. Therefore, it is important to put them in the footer rather than the header as so not to overload the user with links at the top of the page. As with the navigation page, this will be displayed on all pages. All of the links in the footer can be accessed regardless of the user's logged in status.

The main content of the page is split into a central section on the left which spans most of the page. A sidebar on the right contains extra content or links that are relevant to the content in the main section.

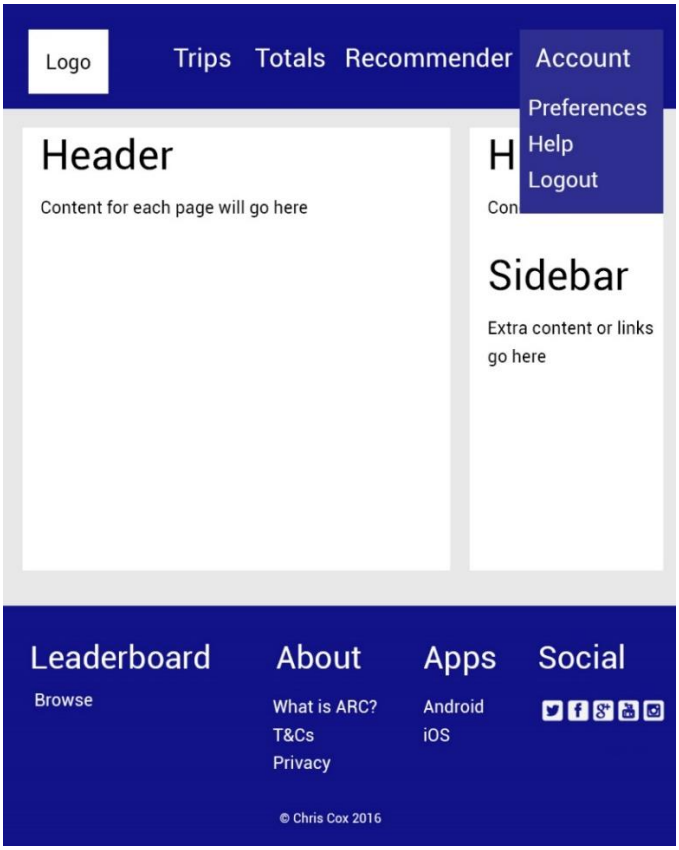


FIGURE 8 - NAVIGATION DESIGN FOR WEBSITE

3.11.1.2 Standard List View

This list view is used for a range of pages, including:

- Trips page

    - Containing park name, trip date, total ride count and ‘Edit’ and ‘Delete’ buttons. This is shown in Figure 9.
- Trip view page
    - Containing ride name, time and ‘Delete’ button.
- Trips totals and statistics page
    - Containing ride name, total ride count for the ride or statistic.
- Editing a park page
    - Containing ride name with ‘Edit’ and ‘Delete’ buttons.
- Recommender page
    - Containing a list of rides that the user has rated with an ‘Edit’ button to adjust the rating.
    - The recommendations themselves are in the sidebar content.
- Totals and stats page
    - Containing ride/park name, total count for the ride/park or a statistic.
- Leaderboard page
    - Containing username and total ride count.

Using a standardised page template that includes a few small modifications for each page (e.g. the adding of a delete button on the trips page, but not on the leaderboard page) allows the website to be easier to use, as the user will become more familiar with the layout of the site. This ensures that they can easily navigate around the system, as they know how best to interact with the lists.

Figure 9 shows a mock-up of the standard list view. In this instance, it shows the trips page – where users can view all their trips, select a trip to edit, or delete a trip. The user can also use the controls on the right of the screen to filter the list contents by year, or sort it in a range of sorting options (ascending and descending dates, and ascending and descending ride counts).

All of the list views are similar – but with the central content altered to reflect the information that is needed in that list. For ease of presentation, this mock-up does not show the navigation or footer – however, they are included as shown in Figure 8.

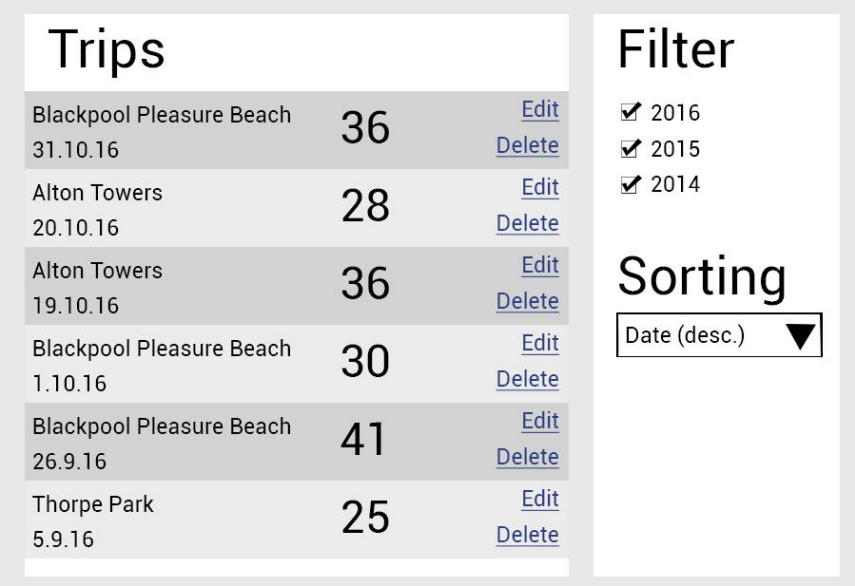


FIGURE 9 - TRIPS LIST DEMONSTRATING THE LIST VIEW

3.11.1.3 Login and Sign Up

Login and sign up forms contain the relevant text boxes and submit buttons for each. For the login form this consists of: A username text box, a password text box and a 'Sign In' button as well as a 'Forgotten Password' link. For the sign-up form, an additional text box for the user's email address is also needed, as well as a 'Retype password' box to check the user's password and a 'Sign Up' button instead of a 'Sign In'. The 'Forgotten Password' link is also replaced with a link to the terms and conditions that the users agree to by signing up.

Care has been taken to ensure that the two forms are easily identifiable so that users do not get mixed up and use the wrong form, for example trying to sign in using the sign-up form. The form contains both server and client side data validation to maintain security and helpful

user information, for example the length of the password being longer than six characters. This helps improve the usability of the page and offers informative feedback and simple error handling, which are two of the Shneiderman's "Eight Golden Rules of Interface Design" as mentioned in the previous chapter.

3.11.1.4 Preferences

The preferences screen (Figure 10) contains the facility for a user to change their password. Containing 3 text boxes, for the old password (for security) as well as two boxes for their new password to ensure they have typed it correctly. Again, this creates a good user experience, providing simple error handling as the user cannot accidentally change their password to something that they have accidently miss-typed, as they have to type their new password twice.

There is also a submit button to send the form. Like the login and signup forms, there is both server side and client side validation. A demonstration of this is shown in the bottom text box where the user has not re-typed the password. Here a useful sentence will be shown explaining the issue which is in red to show the severity of the issue.

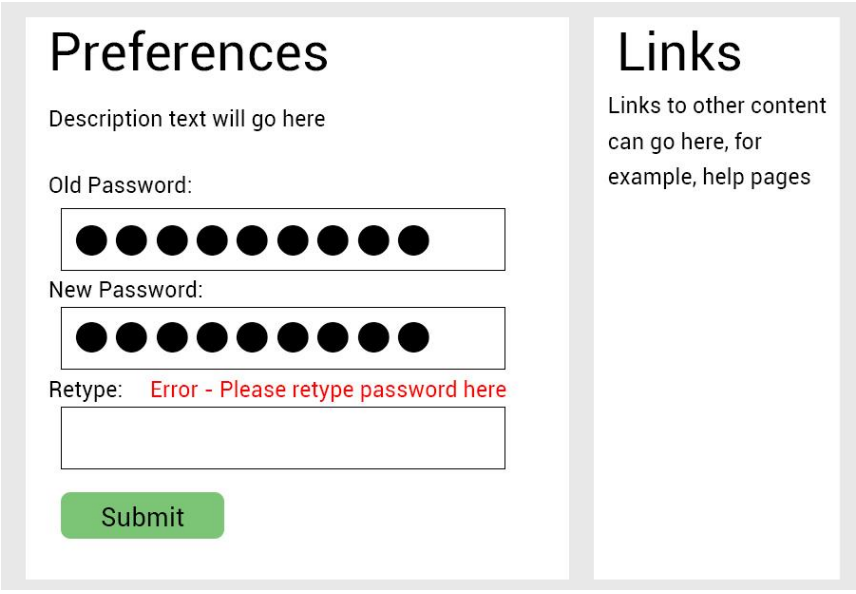


FIGURE 10 - PREFERENCES SCREEN ON WEBSITE

3.11.1.5 Text Pages

The help and information pages are be mainly text, with the occasional link or image to help provide more information about the content. This follows the same layout as the previous pages, with the body text being in the left-hand main section – and links to other relevant pages showing on the right in the sidebar.

3.11.1.6 Light box

Pop-up light boxes can be used in a range of situations where more user controls are needed to be shown, but where it is not necessary to go to a new page in the website, which can split up the user journey, increasing their short-term memory load (a key part of the usability of the system) These are used in:

- Adding a new ride
    - Containing auto-complete text box for choosing the ride name, a button to select the date and a numeric stepper to select the number of times a ride is ridden. This is shown in Figure 11.
- Confirmation of deleting item on lists
    - Containing coloured yes and no buttons.
- Adding new parks
    - Containing a text box for typing the park name, and a button to confirm the action.
- Creating trips
    - Contains an auto-complete text box for choosing the park name, and a date picker to select the date.
- Renaming rides
    - Contains a text box with the existing ride name auto filled in and a button to save.

Figure 11 shows an example of the light box that is shown when adding a new ride to a trip. In this example, the page behind the popup box is distorted, and the box has a shadow under it. This helps to create separation from the page itself, making it stand out to the user. The close and confirm buttons are strategically coloured to help show at a glance what they do (e.g. red to show a destructive action, green to show confirmation). This example also shows a range of other UI elements, such as an auto predict text box for typing in the ride name. This shows a selection of ride names as the user types, these can be selected – making it quicker for adding rides with long names. There is also a number increment element for the number of times to add a ride – and a button to show a time picker.

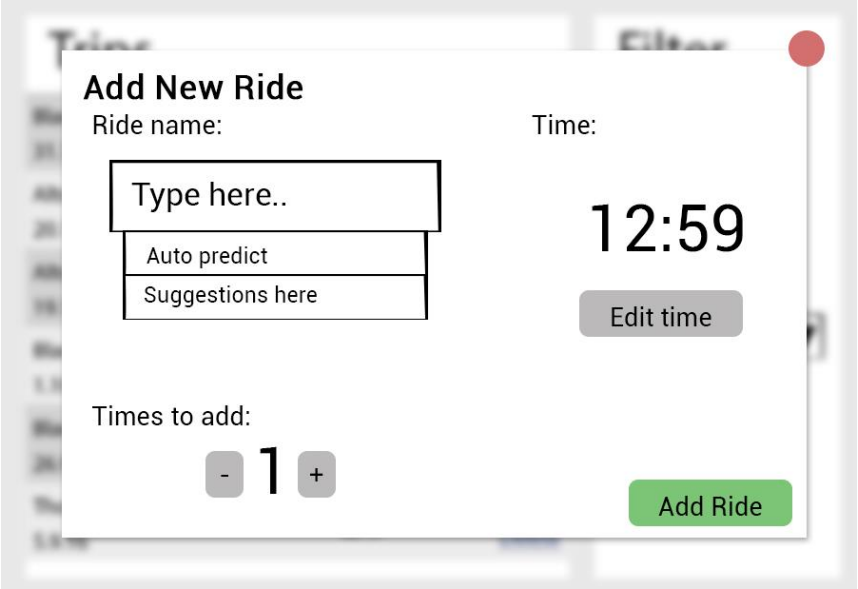


FIGURE 11 - ADD NEW RIDE LIGHT BOX ON WEBSITE

3.11.2 Apps

Many of the changes to both the Android and iOS apps do not affect the user interface. For example, the list of parks that a user can create a trip at will, to the user, look the same as it has done in the past despite the way the data is fetched being completely changed. Keeping

the user interface the same will ensure that existing users who have already used the app and know where everything is will not have to re-learn how to use the app – a key point from the usability research. Some screens have had additional data added to them containing the statistics features, therefore I have only included screenshots of these changes.

All of the changes maintain the same colours, fonts and other styles that were previously used in the app, which themselves reflect the styles used by default in the respective platform. Maintaining this consistency ensures that the user is familiar with the app and its features, making it easier to use. There are however similarities between all of the platforms (including the web). For example, the blue action bar at the top of the screens is the same shade of blue used in the navigation bar on the top of the website. The logo is also maintained across all devices. The mobile devices also share very similar layouts. All of these similarities ensure cross platform consistency as described in the usability section.

3.11.2.1 Sorted Ride List

A set of tabs has been added to the top of the screen under the blue action bar. The existing screen is shown by default, and when the ‘Totals’ tab is selected. This ensures that existing users are not confused with the changes. Figure 12 shows the ‘Statistics’ view that can be accessed by pressing the ‘Statistics’ section of the tabs. The on-screen list contains a range of statistics for that trip, for example the total number of inversions that have been experienced.

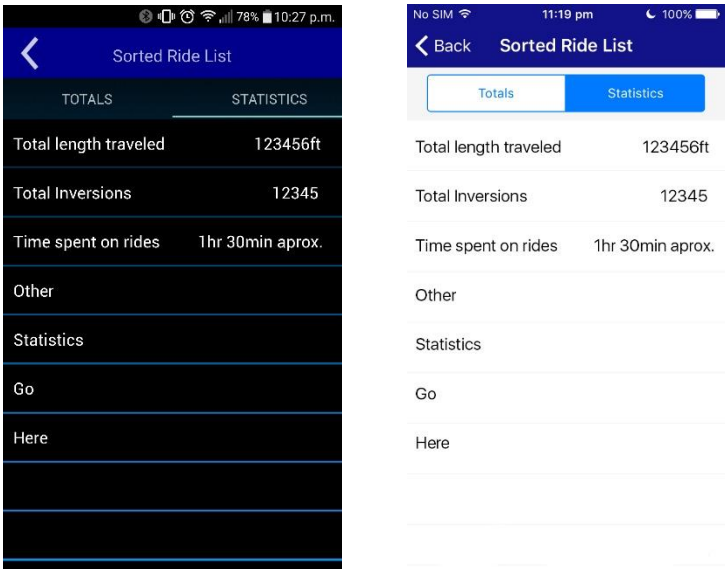


FIGURE 12 - STATISTICS TAB ON THE SORTED RIDE LIST (ANDROID ON LEFT, IOS ON RIGHT)

3.11.2.2 Summary View

The 'Summary View' in the mobile apps shows the all the combined rides that the user has logged. It already had a set of tabs at the top of the screen, allowing users to select what information they would like to see (an overall total, total list of parks visited or total list of rides ridden) The buttons in the top right corner can be used to filter the data by year (the funnel icon) or sort the data in ascending, descending etc. order (the lines icon).

Figure 13 shows the leader board added to the bottom of the screen. This list contains the users and their total ride count. The information on this list is sorted and filtered according to the current sorting and filtering selected.

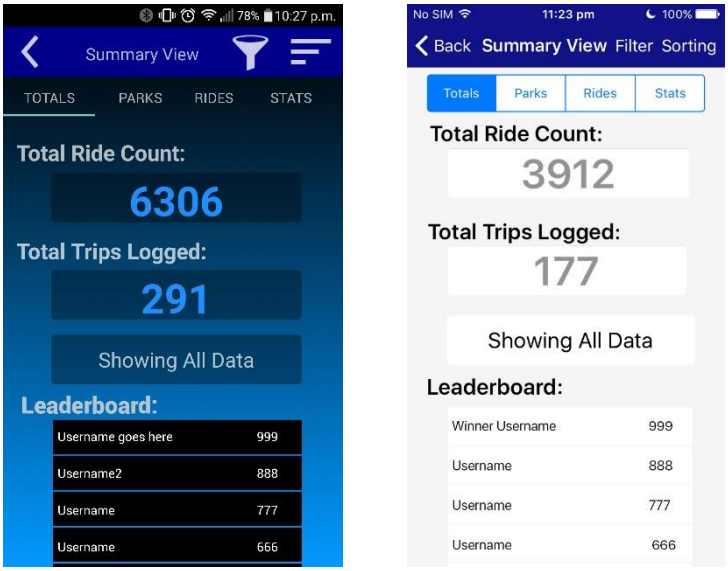


FIGURE 13 - SUMMARY VIEW TOTALS SCREEN

The new statistics tab shows a range of statistics from all the user's data. As with the other data shown in the Summary View, it is filtered and sorted according to the currently selected preferences. This is shown in Figure 14. This has been integrated into the existing screen through the use of an additional tab (similar to the Sorted Ride List screen) that can be selected for viewing after the default screen (the first 'Totals' tab) is shown. This ensures that existing users are not confronted with changes that they have to re-learn.

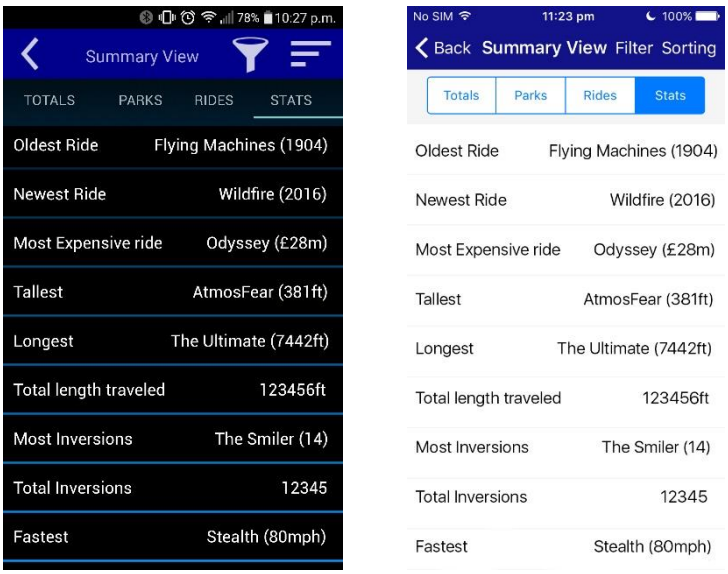


FIGURE 14 - IOS SUMMARY VIEW STATISTICS TAB

3.12 Summary

The system design presented in this chapter demonstrates the considerations that were made to ensure the implementation of the production were efficient and successful. A well thought out design also improves the structure of the final product, which will make it easier to maintain in the future.

4 Implementation

This chapter covers the steps that were taken in the creation of the system for this project. This includes the overall setup of the project and the connections between sections, including connections to the database. There is also a discussion on how each section of the system functions on each of the platforms and how it was created to achieve its purpose.

4.1 Project Setup

Following the system diagram that was outlined in the design chapter of the project (Figure 4), the different code bases were setup. The logic layer was created as a Windows Communication Foundation (WCF) project. This allows the other projects created to call methods from this service in an easy to maintain way. These references can then be updated when the logic layer is altered so that new method calls can be used.

The other projects for the web system were also created, this includes ‘ARCCDataService’- the logic layer that connects the database to the code. ‘ARCCMobileDS’, the mobile Dataservice that connects the mobile to the central Dataservice logic layer. This was created as an ASP.NET Web API project, as this allows for the efficient creation of a REST API. The ‘ARCCWebFront’ project was created, this is the ASP.NET Web forms project that is used for the front end of the website. Also a test project called ‘DataServiceTest’ was created. This houses all the unit tests for the Dataservice. The mobile projects and web project files were all stored in separate Git repositories hosted in Bitbucket. This allowed for a good level of version control, allowing changes to be tracked as part of the agile development strategy.

The classes used throughout the system that represent items, such as rides, parks and users were all created in the Dataservice project. This allowed for them to be re-used throughout the web system, allowing for efficient movement of data throughout parts of the system. A full list of these types can be seen in Figure 15.

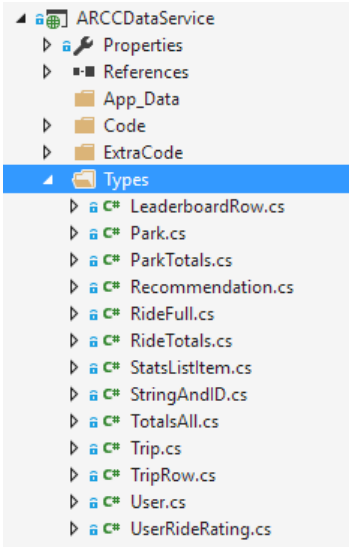


FIGURE 15 - SCREENSHOT SHOWING THE DATA TYPES IN THE ARCCDATASERVICE PROJECT

An example of one of the classes is shown in Figure 16. This shows the tags that have been added to the class as part of the serialisation for the WCF project. This allows the calls to the Dataservice project to let classes be passed in as parameters as well as returning classes to the code that calls it. The class also shows the use of the C# getters and setters, which make accessing the member variables of the class easier. There are also two constructors, an empty one for initialising an instance of the class, and one that accepts a range of parameters for setting all the variables inside the class. Different classes also have a range of constructors that contain less parameters, for use where all the data is not needed.

```
[DataContract]
65 references | chris_c_4, 4 days ago | 1 author, 2 changes
public class Recommendation {
    [DataMember]
    4 references | chris_c_4, 4 days ago | 1 author, 1 change
    public int id { get; set; }
    [DataMember]
    1 reference | chris_c_4, 4 days ago | 1 author, 1 change
    public DateTime dateCreated { get; set; }
    [DataMember]
    4 references | chris_c_4, 4 days ago | 1 author, 1 change
    public int userID { get; set; }
    [DataMember]
    6 references | 0/1 passing | chris_c_4, 4 days ago | 1 author, 1 change
    public int rideID { get; set; }
    [DataMember]
    2 references | chris_c_4, 4 days ago | 1 author, 1 change
    public string rideName { get; set; }
    [DataMember]
    5 references | 0/1 passing | chris_c_4, 4 days ago | 1 author, 1 change
    public int? userRating { get; set; }


    //constructor
    7 references | 0/5 passing | chris_c_4, 4 days ago | 1 author, 1 change
    public Recommendation(int pid, DateTime pdateCreated, int puserID, int prideID, string prideName, int? puserRating) {
        id = pid;
        dateCreated = pdateCreated;
        userID = puserID;
        rideID = prideID;
        rideName = prideName;
        userRating = puserRating;
    }
    1 reference | chris_c_4, 4 days ago | 1 author, 1 change
    public Recommendation() {
    }
}
```

FIGURE 16 - SCREENSHOT SHOWING RECOMMENDATION CLASS IN ARCCDATASERVICE

4.2 Database

The database is hosted in the UK. The tables for the database are generally the same as the ones included in the design chapter. However there are some slight changes, the updated ERD is shown in Figure 17. Some more columns were added to some tables to store additional data that was useful for the system. An example of this included the ‘deleted’ columns on the trip row, custom ride and custom park tables. These allow for the items to be deleted by the user, but still retained in the system for future use at a later date. New tables were also created, these include: ‘PasswordRecovery’, a table to store attempts to use the password recovery page and ‘userRideRating’, a table to store the user’s ratings for individual rides that can then be used in the recommendation feature.

All of the information about the column names, data types and sizes, primary and foreign keys, whether a column can be null or not can be found on Figure 17.

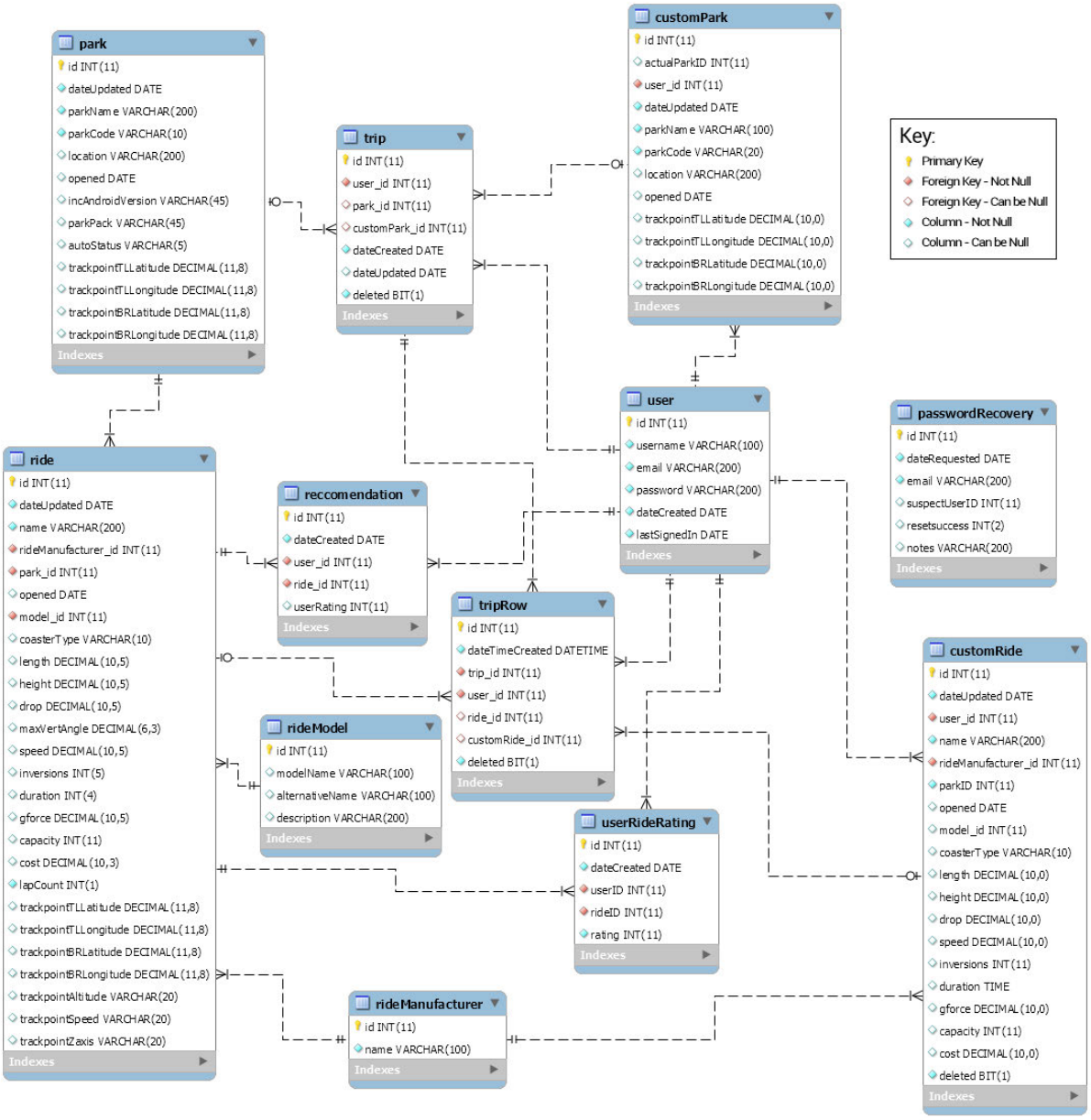


FIGURE 17 - ERD DIAGRAM SHOWING THE LIVE DATABASE

Connections to the database were all done through the database connections in the Dataservice (logic layer). These were maintained as a connection string in the project

settings. The server address was added to the hosts file on the development machine and live server. This adds security, as the address of the database server is not inside the code.

All SQL queries are done through the use of stored procedures. This has many advantages, including:

- Good separation of code. By separating the C# logic code and the SQL query code, cleaner and more easily understood code could be produced that does not mix concerns.
- Protection against SQL injection attacks. Injection attacks are listed as the most critical web application security risk (OWASP, 2017). By using stored procedures with parameterised queries, the risk of SQL injection attacks has been reduced.
- Having the stored procedures on the database server itself can help increase the speed of queries and calls to the database.

4.3 Hosting

A T2 micro EC2 Windows instance was selected as this is included in the free trial tier. A key problem that had to be overcome with this server was that the date and time were both incorrect, and in the incorrect format, and this caused issues with the dates and times shown on the website. Following this, Microsoft Internet Information Services (IIS) manager was installed and setup. This included setting up the sites themselves and binding them onto the relevant web ports (80 for the main website, 8903 for Dataservice and 6576 for the mobile Dataservice). Since Dataservice is not intended to be publically accessible on its own, instead being used by the other projects internally, this port was left closed on the firewall – whereas the others were opened up to be used externally.

4.4 Webpage Implementation (Trips page)

The following section will give an example of how the trips list page is setup in the project. This is a list based page that was also included in the design chapter. This page makes use of 2 sets of code in the ARCCWebFront code, the front end .aspx file and the 'code behind' .aspx.cs file. The code also relies on calls to the Dataservice code in ARCCDataservice, which in turn relies on stored procedures that are hosted on the database. Figure 18 shows a screenshot of what the implemented page looks like.

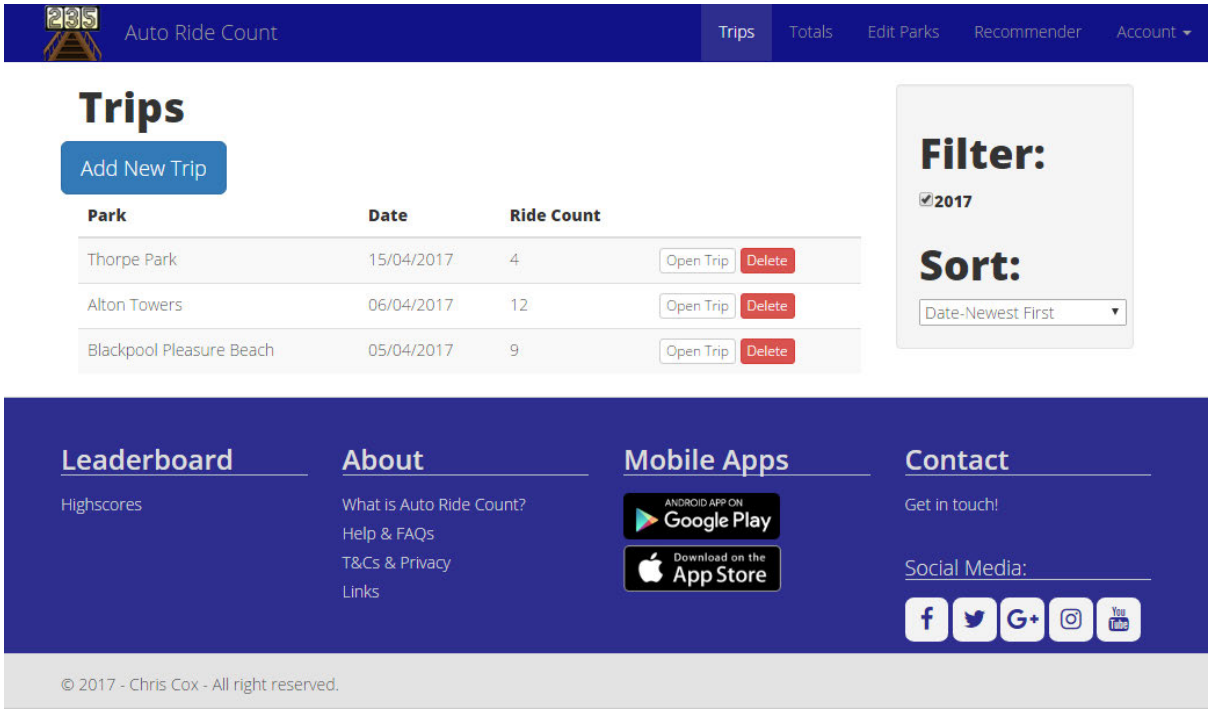


FIGURE 18 - SCREENSHOT SHOWING THE TRIPS PAGE

4.4.1 ARCCWebFront code

Figure 19 shows the front end .aspx file, which uses HTML5 and inline ASP.NET C# code to setup the layout of the page. An asp:Repeater combined with the inline ‘Eval’ method is used to create the list view within the table code. The data is bound to this component in the ‘code behind’ file. Control components such as buttons, dropdown lists and check boxes are all generally ASP.NET types, as they can be linked to the ‘code behind’ automatically.

However, some HTML input types are used on other pages when the link to the ‘code behind’ is not needed – for example when jQuery is called to open a pop-up box.

Near the end of the page, there is an optional asp:Content section for page specific JavaScript or jQuery code. In this example, there is the relevant code for the deletion confirmation message box.

```
<%@ Page Title="Trips" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Trips.aspx.cs" Inherits="ARCCWebFront.Trips"
%>

<%@ Import Namespace="ARCCDataService" %>

<asp:Content runat="server" ID="BodyContent"
ContentPlaceHolderID="MainContent">
    <hgroup class="title">
        <h1><%: Title %></h1>
    </hgroup>

    <div class="row">
        <a href="<%$RouteUrl:routename=AddTrip %>" class="btn btn-primary btn-
lg" role="button" runat="server">Add New Trip</a>
    </div>

    <article>
        <asp:Repeater runat="server" ID="rep">
            <HeaderTemplate>
                <table class="table table-striped">
                    <thead>
                        <tr>
                            <th>Park</th>
                            <th>Date</th>
                            <th>Ride Count</th>
                            <th></th>
                        </tr>
                    </thead>
                    <tbody>
            </HeaderTemplate>

            <ItemTemplate>
                <tr>
                    <td>
                        <%# Eval("parkName") %>
                    </td>
                    <td>
                        <%# Eval("dateOfTrip").ToString().Replace("
00:00:00","") %>
                    </td>
                    <td>
                        <%# Eval("rideCount") %>
                    </td>
                    <td>
                        <asp:Button ID="btnOpenTrip" runat="server" Text="Open
Trip" CssClass="btn btn-default btn-xs" />
```

```
                        <asp:Button ID="btnDeleteTrip" runat="server"
Text="Delete" CssClass="btn btn-danger btn-xs" data-toggle="confirmation" data-
popout="true" />
                    </td>
                </tr>
            </ItemTemplate>

            <FooterTemplate>
                </tbody>
        </table>
            </FooterTemplate>
        </asp:Repeater>

        <div id="ErrorNoTrips" class="alert alert-info" runat="server"
visible="false">
            There are no trips here. Please adjust your filter criteria, or add
a new trip to see some trips!
        </div>
    </article>
</asp:Content>

<asp:Content runat="server" ID="Content1"
ContentPlaceHolderID="SidebarContent">
    <div>
        <h1>Filter:</h1>
        <asp:CheckBoxList ID="chkblFilter" runat="server"
OnSelectedIndexChanged="chkblFilter_SelectedIndexChanged"
AutoPostBack="True"></asp:CheckBoxList>
    </div>
    <div>
        <h1>Sort:</h1>
        <asp:DropDownList ID="drpSorting" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="drpSorting_SelectedIndexChanged">
            <asp:ListItem Text="Date-Newest First" Value="0" />
            <asp:ListItem Text="Date-Oldest First" Value="1" />
            <asp:ListItem Text="Ride Count - Ascending" Value="2" />
            <asp:ListItem Text="Ride Count - Descending" Value="3" />
        </asp:DropDownList>
    </div>
</asp:Content>


<asp:Content runat="server" ID="Scripts"
ContentPlaceHolderID="ExtraScriptsContent">
    <%--  http://bootstrap-confirmation.js.org/ --%>
    <script>
        $('[data-toggle=confirmation]').confirmation({
            rootSelector: '[data-toggle=confirmation]',
// other options

        });
    </script>
</asp:Content>
```

FIGURE 19 - TRIPS PAGE FRONT END CODE

Figure 20 shows the backend .aspx.cs 'code behind' file. This uses C# and contains website specific logic code, event handlers, links to the relevant Dataservice calls etc. The page_load

method is automatically called when the page loads, it contains the code that checks to see if the user is logged in, if they are not then they are re-directed to the login page. Security functions built into asp.net are used, as they offer a reliable login management provider that is secure and easy to use. Following this, the data is called from the Dataservice. Next the page is setup to display the data.

There is a section of code marked by the comment "Event handlers" that shows where the handlers for button clicks, check box selects, drop down interactions etc. are handled. One line of code to note is the line:

"`Response.RedirectToRoute("TripView", new { tripid = itemToOpen });`" within the btn_OpenClick method. This demonstrates the use of the routes that have been setup for this project. This means that instead of going to www.[url].co.uk/trips.aspx, the user can instead go to www.[url].co.uk/trips. This makes the website easier to navigate for users as they do not have to know about the .aspx extension.

At the bottom of the page, under the comment "other functions" there is a function that connects to Dataservice to carry out an action.

```
using System;
using System.Collections;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections.Generic;
using ARCCDataService;
using System.Web.Security;

namespace ARCCWebFront {
    public partial class Trips : Page {
        List<Trip> tripList = new List<Trip>(); //this is the list of all the
trips that the user has
        public List<Trip> displayTripList = new List<Trip>(); //this is the
list that is currently shown to the user (i.e. with filtering/sort)
        List<int> selectedYears = new List<int>(); //this is the years that are
currently shown

        protected void Page_Load(object sender, EventArgs e) {
            //try and get the user login info
            User tempUser = new User();
            try {
```

```
                tempUser.id = int.Parse(Context.User.Identity.Name);
            } catch (Exception) {

                FormsAuthentication.RedirectToLoginPage();
                return;
            }

            //getting the data
            Dataservice1.iDataservice1 dsLink = new
Dataservice1.iDataservice1Client();
            tripList = dsLink.GetAllTrips(tempUser);


            //only get the years from the trips on the first run-through
            if (!(IsPostBack)) {
                List<int> yearList = new List<int>();
                foreach (Trip trip in tripList) {
                    int year = trip.dateOfTrip.Year;
                    if (yearList.Contains(year) == false) {
                        yearList.Add(year);

                        //add to check boxes, and tick it by default
                        chkblFilter.Items.Add(year.ToString());
                        chkblFilter.Items[chkblFilter.Items.Count - 1].Selected
= true;

                        //add to list of selected years
                        selectedYears.Add(int.Parse(year.ToString()));
                    }

                    //here is an iteration through the trips
                }
            }


            //get the selected years
            updateCheckBoxYearList();

            //filtering
            //if the trip date is in the year that is on the selectedYears
list, then display it
            foreach (Trip trip in tripList) {
                if (selectedYears.Contains(trip.dateOfTrip.Year)) {
                    displayTripList.Add(trip);
                }
            }

            //sorting
            switch (drpSorting.SelectedItem.ToString()) {
                case "Ride Count - Ascending":
                displayTripList.Sort(delegate (Trip t1, Trip t2) { return
t1.rideCount.CompareTo(t2.rideCount); });
                break;
                case "Ride Count - Descending":
                displayTripList.Sort(delegate (Trip t1, Trip t2) { return
t2.rideCount.CompareTo(t1.rideCount); });
                break;
                case "Date-Oldest First":
                displayTripList.Sort(delegate (Trip t1, Trip t2) { return
t1.dateOfTrip.CompareTo(t2.dateOfTrip); });
                break;
                default:
                //this will also be "Date-Newest First"
```

```
                displayTripList.Sort(delegate (Trip t1, Trip t2) { return
t2.dateOfTrip.CompareTo(t1.dateOfTrip); });
                break;
            }

            if (!IsPostBack) {
                BindRepeater();
            }

            foreach (RepeaterItem ritem in rep.Items) {
                Button btnDelete = ritem.FindControl("btnDeleteTrip") as
Button;
                btnDelete.Click += new EventHandler(btn_DeleteClick);

                Button btnOpen = ritem.FindControl("btnOpenTrip") as Button;
                btnOpen.Click += new EventHandler(btn_OpenClick);
            }


            //show the 'no trips' message box if there are no trips visible
            ErrorNoTrips.Visible = (displayTripList.Count == 0);
        }

        private void BindRepeater() {
            rep.DataSource = displayTripList;
            rep.DataBind();
        }


        //Event handlers
        void btn_DeleteClick(object sender, EventArgs e) {
            Button btn = (Button)sender;
            RepeaterItem ritem = (RepeaterItem)btn.NamingContainer;
            int itemToDelete = displayTripList[ritem.ItemIndex].id;

            //delete on DB
            deleteTripID(itemToDelete);

            //remove from items to display
            displayTripList.RemoveAt(ritem.ItemIndex);


            BindRepeater();
        }

        void btn_OpenClick(object sender, EventArgs e) {
            Button btn = (Button)sender;
            RepeaterItem ritem = (RepeaterItem)btn.NamingContainer;
            int itemToOpen = displayTripList[ritem.ItemIndex].id;

            Response.RedirectToRoute("TripView", new { tripid = itemToOpen });
        }

        protected void chkblFilter_SelectedIndexChanged(object sender,
EventArgs e) {
            //just call the method to update the selectedYears list
            updateCheckBoxYearList();
            BindRepeater();
        }

        private void updateCheckBoxYearList() {
            //this will update the selectedYears list with the currently
selected years from the checkboxes
            for (int i = 0; i < chkblFilter.Items.Count; i++) {
```

```
                int year = int.Parse(chkblFilter.Items[i].Value.ToString());
                if (chkblFilter.Items[i].Selected) {
                    selectedYears.Add(year);
                } else {
                    selectedYears.Remove(year);
                }
            }
        }

        protected void drpSorting_SelectedIndexChanged(object sender, EventArgs
e) {
            BindRepeater();
        }


        //other functions
        public string deleteTripID(int tripID) {
            Dataservice1.iDataservice1 dsLink = new
Dataservice1.iDataservice1Client();
            bool sucess = dsLink.DeleteTrip(tripID, true);

            return "";
        }


    }
}
```

FIGURE 20 - TRIPS PAGE 'CODE BEHIND' FILE

4.4.2 ARCCDataservice code

When a remote function call to the Dataservice has been initialised, the relevant code is executed in the Dataservice project, then passed back to the code that called it. The example shown in Figure 21 shows the “getAllTrips” function. This is called from the above web code, and has been written to get a list of all the user’s trips. Here, a ‘using’ statement, with the MySqlConnection passed in is used to connect to the database. The database connection strings are stored centrally in the project settings. This allows them to be modified centrally if the database configuration changes.

This function calls a stored procedure on the database called “GetAllTrips_V001”, and passes in the parameter “userID”. The values from the result are then parsed back to C# datatypes and added to a new ‘Trip’ object, which is then added to the list of trips which is part of the return statement.

```
        public List<Trip> getAllTrips(int userID) {
            List<Trip> retData = new List<Trip>();

            using (MySqlConnection conn = new
MySqlConnection(ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionS
tring)) {
                conn.Open();

                string procName = "GetAllTrips_V001";
                MySqlCommand cmd = new MySqlCommand(procName, conn);
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@userID", userID);

                MySqlDataReader rdr = cmd.ExecuteReader();
                while (rdr.Read()) {
                    int id = (int)rdr.GetInt64("id");
                    int parkID = (int) rdr.GetInt64("parkID");
                    string parkName = rdr.GetString("parkName");
                    DateTime dateCreated = rdr.GetDateTime("dateCreated");
                    DateTime dateUpdated = rdr.GetDateTime("dateUpdated");
                    int rideCount = (int)rdr.GetInt64("rideCount");

                    Trip retTrip = new Trip(id, userID, parkID, parkName, dateCreated,
dateUpdated, rideCount);

                    retData.Add(retTrip);
                }
                rdr.Close();
            }

            return retData;
        }
```

FIGURE 21 - GETALLTRIPS CODE IN THE DATASERVICE

4.4.3 Stored Procedure

Figure 22 shows the GetAllTrips_V001 stored procedure that is stored on the database. This is called by the previous Dataservice call shown above. Version numbering (the _V001 suffix) has been used to maintain a history of stored procedures as they are changed, this reduces the need to do changes to stored procedures on the live database which could be potentially damaging to the system.

This stored procedure selects all the non-deleted (deleted = 0) trips from the database where the ‘userID’ matches the one passed as a parameter. The ‘parkID’ and ‘parkName’ columns use the ‘CONCAT_WS’ function to join the park and custom park columns. As only one of

these can contain a value at once, it means that this will return whichever is being used. Nested selects have been used where necessary to get the information required, for example the total ride count for the trip.

```
CREATE DEFINER=`chrisc4`@`%` PROCEDURE `GetAllTrips_V001`(IN userID INT(11))
BEGIN

SELECT `trip`.`id`,
    `trip`.`user_id`,
    CONCAT_WS('',park_id,customPark_id) as 'parkID',
    CONCAT_WS('',
        (SELECT parkName FROM park WHERE id = park_id),
        (SELECT parkName FROM customPark WHERE id = customPark_id)) as 'parkName',
    `trip`.`dateCreated`,
    `trip`.`dateUpdated`,
    (SELECT COUNT(*) FROM tripRow
        WHERE trip_id = trip.id
        AND (ride_id IS NOT NULL OR customRide_id IS NOT NULL )
        AND deleted = 0)
    as 'rideCount'
FROM `trip`
WHERE user_id = userID
AND deleted = 0;

END
```

FIGURE 22 - GETALLTRIPS_V001 STORED PROCEDURE SQL

4.5 Mobile Implementation

The implementation of getting data onto a screen on the mobile apps is similar to the website, however an additional layer- the mobile Dataservice (ARCCMobileDS) is used in between the app code and the Dataservice (shown in Figure 23). This is an ASP.NET Web API project that offers a REST API style interface for the Dataservice calls that can be used by the mobile app. This has been separated from the standard Dataservice to allow for separate versioning of calls – as users might be slow to download updates for apps, therefore some users may be outdated Dataservice calls. All calls to the system include a version parameter to allow for this versioning. The calls also include username and password parameters, which are checked on each call. This increases the security of the system.

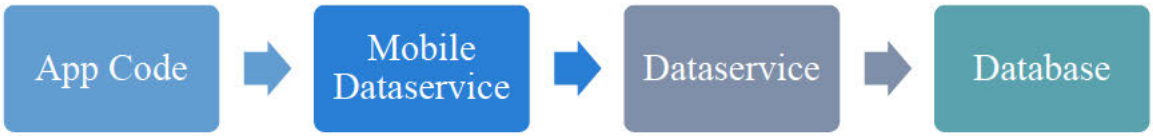


FIGURE 23 - DIAGRAM SHOWING A CALL FROM THE APP CODE TO THE DATABASE

Inside the mobile code, an asynchronous http job is created (demonstrated in Figure 24). Calling the job asynchronously ensures that the user interface does not freeze and lock up whilst loading the data, which would be a bad user experience, especially if the user was in a place that had slow internet – such as a theme park. The URL for the call is generated using a function in the CloudHelper code module. This takes in a range of parameters and formats them in the style needed. Doing this centrally allows for easier changes to the calls in the future if necessary. In this example, the call's function is to get the list of trips that the user has. This is done by calling this URL:

[URL]/api/TripsList?version=[version]&userName=[username]&userPassword=[userPassword]

```
Sub CalculateParkList
     'call the http job
     Dim jobTripsList As HttpJob
     jobTripsList.Initialize("jobTripsList", Me)
     jobTripsList.Download(CloudHelper.GetAPIcallURL("TripsList",1,""))

     ProgressDialogShow2("Loading trips, please wait", False)
End Sub
```

FIGURE 24 - CODE SHOWING THE CREATION OF A MOBILE HTTP JOB

The code inside the mobile Dataservice that is called by this call is shown in Figure 25. It takes in the parameters that are passed in the URL parameters as part of the get request. The

code validates the user's credentials, then calls the relevant Dataservice call to get the data. The data is passed back to the app as a string of JSON. This was selected over other options such as XML as it is more lightweight, as well as there being a large selections of plugins and tools to automatically generate and parse. It is also human readable, which is useful for debugging. To change the output to JSON, the line of code in Figure 26 was added to the WebApiConfig.cs file.

```
    public class TripsListController : ApiController {
        //Getting a list of the trips
        public IEnumerable<Trip> Get(int version, string userName, string
userPassword) {
            //check the user
            UserHelper uh = new UserHelper();
            User currentUser = uh.checkUserCredentials(userName, userPassword);

            //do the call to get the park list
            if (version == 1) {
                Dataservice1.iDataservice1 dsLink = new
Dataservice1.iDataservice1Client();
                return dsLink.GetAllTrips(currentUser);
            } else {
                throw new HttpResponseException(HttpStatusCode.NotImplemented);
            }
        }
    }
```

FIGURE 25 – CODE SNIPPET INSIDE MOBILE DATASERVICE FOR GETTING THE LIST OF TRIPS

```
            config.Formatters.JsonFormatter.SupportedMediaTypes
                .Add(new MediaTypeHeaderValue("text/html"));
```

FIGURE 26 - CODE SNIPPET TO CHANGE MOBILE DATASERVICE'S OUTPUT TO JSON

Each activity within the app has a method which is called when the http job completes and returns a value, the name of the job is checked, and then the relevant method is called to process the returned data. Figure 27 shows the returned data (a string called data) being parsed using a JSONParser and being mapped to a TripsListRow object which is then added to the TripsList. This list is then used to update the interface with the list of the trips.

```
      Dim parser As JSONParser
      parser.Initialize(data)
      Dim root As List = parser.NextArray
      For Each colroot As Map In root
            Dim dateOfTrip As String = colroot.Get("dateOfTrip")


            'the display item
            Dim tempLine As TripsListRow
            tempLine.ParkCode = colroot.Get("parkName")
            tempLine.RideCount = colroot.Get("rideCount")
            tempLine.TripID = colroot.Get("id")
            DateTime.DateFormat = "yyyy-MM-dd'T'HH:mm:ss"
            Dim dateToShowTicks As Long =
DateTime.DateParse(dateOfTrip)
            DateTime.DateFormat = "dd.MM.yyyy"
            tempLine.TripDate = DateTime.Date(dateToShowTicks)
            TripsList.Add(tempLine)
      Next
```

FIGURE 27 - CODE SNIPPET SHOWING THE RETURNED TRIPS JSON DATA BEING PARSED

4.6 Geolocation

The 'Full Auto' part of the geolocation modifications that were added as part of this project include the addition of a service that runs and monitors the user's location throughout their day at a park. This additional service uses Google's fused location provider on Android, meaning it has a low power consumption. When the service detects that a user is near a ride, the existing 'AutoGuesser' service is started automatically. This service has a high power consumption, therefore timers and checks have been added to ensure that it is only run when necessary.

As the code itself is too long and complex for displaying as part of this document, a pseudocode representation has been provided below to demonstrate the logic used by the new geolocation feature.

1. User presses the start button
2. ‘FullAuto’ service is started, this starts the low power location listener.
3. When a location is found, it is checked against the locations of all the ride stations at the park the user is at.
    a. If the location does not match a ride, then the app waits for the next location update.
    b. If the location matches a ride, then the exiting ‘AutoGuesser’ service is called. A 10 minute timer is also started.
        i. If the ‘AutoGuesser’ service recognises a ride, then it is added to the ride list automatically. The ‘AutoGuesser’ service is then stopped, and a 2 minute timer is started. Any ‘FullAuto’ location updates are discarded during this time. The time period was selected as it sufficient enough to prevent adding the same ride multiple times over the course of the user being on the ride, but short enough to allow for the user to go back on the same ride quickly. Once this 2 minute timer has finished, the ‘FullAuto’ location updates are resumed.
        ii. If the ‘AutoGuesser’ service does not locate a ride and the 10 minute timer passes, the user’s last know location from the ‘FullAuto’ is checked to see if it matches a ride station.
            1. If this location matches a ride’s vicinity (e.g. if the user is still waiting to get on the ride), then the ‘AutoGuesser’ service is left running.
            2. If the location is not near a ride, then the ‘AutoGuesser’ service is stopped – as this means that the user is no longer near the ride (e.g. they were just walking past).

4.7 Recommendation System

After consideration during the research stage of this project, it was decided that the system should use a collaborative filtering based recommendation system, as this would provide more accurate ride recommendations to the user. It was also decided that the recommendation system would be run when the user requests it, meaning a live recommendation had to be added into the database and shown to the user very quickly. This was done as it would give the user an instant recommendation. However the possibility of future changes to the project had to be considered, so that the code could be modified in the future to support running the recommendation system at a separate time, with multiple recommendations being added into the system for the user to view at a later date. This would reduce the loading time when the user requests a recommendation, as it just has to be collected from the database. However the user could run out of recommendations for them, which would be a negative user experience.

For the implementation of the recommendation system, a library was selected to assist in the generation of recommendations. The NReco system was chosen, as it provided a C# implementation that could be easily implemented within the Dataservice code, as well as a large amount of support documentation available – which assisted in the deployment of the system.

The recommendation system used a list of all the existing user's ride rating data to generate recommendations for a user. One drawback of the library was that it could only accept user ride rating data as a text file, therefore a system to output the database contents to a text file for the system to read had to be implemented. Figure 28 shows the implementation of the recommendation system. First, the file path is loaded. This has been stored in the project settings, as this allows for the easy, central, modification of the path. A Dataservice function is then called which returns all the of the user's ride ratings. This is then formatted into a tab

delimited file and outputted to the file system. Due to the current scale of this system, it was decided that this step would be run every time a recommendation is requested – as this would allow for more up-to-date data to be used, therefore producing more accurate recommendations. The recommendation system is then setup, and a single recommendation is then requested. This is then added to the database.

```
        private bool runRecommendationGenerator(int userID) {
            //calculating the file path
            string filePath = Properties.Settings.Default.RecommenderFileOut;

            //get all the ride ratings
            List<UserRideRating> allUserRideRatings =
getOverallAllUserRideRatings();

            //generate the text file
            var fileContents = new StringBuilder();
            foreach (UserRideRating rating in allUserRideRatings) {
                string line = string.Format("{0}\t{1}\t{2}", rating.userID,
rating.rideID, rating.rating);
                fileContents.AppendLine(line);
            }
            File.WriteAllText(filePath, fileContents.ToString());


            //get the recommendation
            int numberOfItems = 1;
            var model = new FileDataModel(filePath);
            var similarity = new LogLikelihoodSimilarity(model);
            var neighborhood = new NearestNUserNeighborhood(3, similarity,
model);
            var recommender = new GenericUserBasedRecommender(model,
neighborhood, similarity);
            var recommendedItems = recommender.Recommend(userID,
numberOfItems);


            //delete the text file
            File.Delete(filePath);


            //add the recommendations to the database
            try {
                int recRideID =
int.Parse(recommendedItems[0].GetItemID().ToString());
                if (recRideID > 0) {
                    Recommendation newRec = new Recommendation(0, DateTime.Now,
userID, recRideID, "", null);
                    return insertRecommendation(newRec);
                }
                return false;
            } catch (Exception) {
                return false;
            }
        }
```

FIGURE 28 - CODE SNIPPET SHOWING THE IMPLEMENTATION OF THE NRECO LIBRARY

4.8 Design Variations

Some slight design alterations were made during the implementation stage of the project. These have been detailed below:

4.8.1 Add New Trip

The 'Add New Trip' button on the Trips page was initially supposed to call a pop-up light box. This was however changed to be a separate page. This was done because the asp:Calendar item uses post-backs to the server, which would have caused the light box to disappear when the page is re-loaded. Whilst a solution was found to this (which was later used to edit rides on the ParkEdit page), other benefits of separating the functionality to a separate page were noted. These included:

- Reduced load speed for the Trips page. If a pop-up light box were to be used, the list of parks would have to be loaded whilst loading the Trips page. This would add extra overhead to an already large page to load. Whilst AJAX could be used to load the list asynchronously, this functionality was not included as part of the scope of the project.
- Cleaner mobile interaction. Having the page separate makes selecting the park and date easier as the whole screen is dedicated to that functionality, as opposed to a popup which only takes up a large percentage of the existing page.
- Ability to bookmark the page. Splitting this functionality to a separate page means that it could be bookmarked by the user and used as a quick shortcut for if they would like to add a trip to their account.

4.8.2 Add New Ride

The ‘Add New Ride’ pop-up light box was designed as part of Figure 11, and included an auto-complete textbox to select the ride, a time selector and an option to select how many times to add the ride. The popup box that was implemented (shown in Figure 29) keeps the time selector, however does not have the option to select how many times to add the ride. This was removed to help alleviate the load onto the server if this function was used. The auto-complete dropdown ride selector was also replaced with a dropdown box to select the ride. This was done because it is a lot easier to use on a mobile device. It also means that the user does not have to know the name of the ride they want to add before they start using the form. This could be useful when visiting parks that they are not familiar with. This also links to the research done as part of the literature review, which suggested reducing the ‘short term memory load’ on the user. By using a dropdown box instead of an auto-complete text box, the user no longer has to concentrate on remembering, then typing the name of the ride, instead they can just pick it off a list.

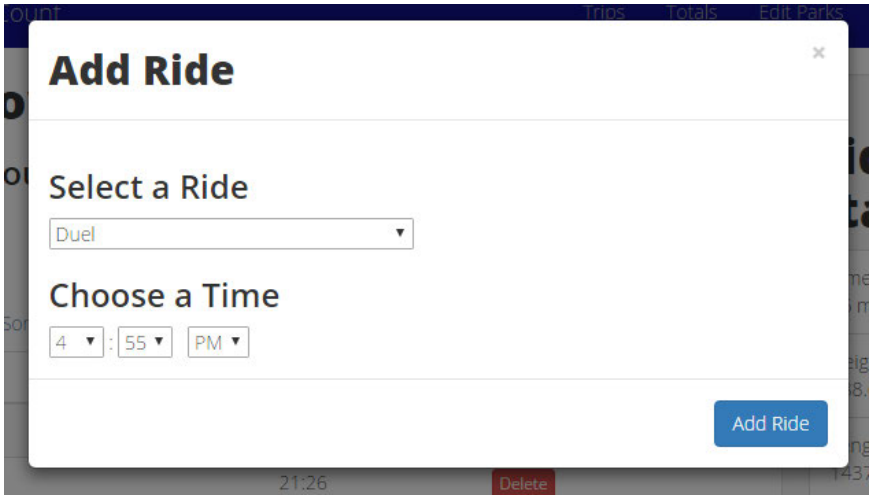


FIGURE 29 - SCREENSHOT SHOWING THE ADD NEW RIDE POP-UP LIGHTBOX

4.8.3 ‘Edit Parks’ link in Navigation bar.

As part of the initial design, the option to add a new park was going to be linked to when adding a new trip. The option to edit the ride list at a park was going to be linked when adding a ride to a trip. This followed the way that this functionality is linked to on the existing mobile app.

When this was trialled on the website, the flow of the pages did not work, and it was confusing to locate the links when needed. As the website has a different navigation structure than the mobile app, with space for more links at the top of the page, a link to the ‘Edit Parks’ page was added to the navigation bar. This makes the functionality easier to find and easier to use. A link to the page was still retained when adding a new trip as so not to confuse any existing users who are familiar with where the link is on the existing apps.

4.8.4 Button Colour Changes

During the design stage, the colour of the buttons were chosen to be green for confirmation actions and red to show a destructive actions. These colours were changed slightly. The red to represent a destructive action was retained, however the button to close a pop-up light box was changed to grey, as closing a pop-up box is not destructive, because the information entered is retained if the user opens the pop-up box again. The green colour for confirmation was also changed to blue. This was because the blue colour was more in-keeping with the rest of the colour scheme of the website and existing mobile apps.

By adjusting the colour of the buttons, a more consistent and easy to use web interface was created. This is reflected by the positive feedback for usability in the user survey results in the evaluation chapter.

4.8.5 Leader boards on Mobile

The positioning of the leader board containing a sorted list of the users and their total ride count was initially going to go on an existing tab. This was shown in Figure 12 in the design

section. This was moved to a separate screen during the implementation stage. This was done for a number of reasons, which are as follows:

- Avoids confusion for existing users. Adding the leader board to the existing screen may have been confusing to existing users of the mobile app, which was a key point to avoid, according to the research considered in the literature review. By adding the new functionality to a separate screen, this confusion can be avoided.
- Keeps a clean separation with personal totals, and leader board. Adding the leader board with other users on, to the same screen as personal totals and statistics might cause confusion, therefore keeping them separate would be easier to understand for the users.
- Enables space to be used in future. Whilst working on the totals page for the website, more statistics were generated which could be added to the mobile totals page in the future. As this screen already contains some of the data, it would make sense to add the extra data here too – as it is related to the existing data. Therefore this area should be kept empty for this data to be added as part of a later project.

5 Testing

This section discusses a range of techniques that are available to test software, as well as those that were chosen for use in this project. Testing is an important part of the development of software as it helps to maintain the quality of the system produced, hopefully ensuring that it can be used by the user without any issues. One definition of testing is: "…the process of executing a program with the intent of finding errors." (Glenford, et al., 2011). Many techniques to try to find bugs or errors have been developed to help create a structured approach to testing. These more methodical approaches ensure that testing is as thorough as possible, helping to reduce the number of errors that are in the system.

5.1 Black box testing

Black box testing is a technique where the tester is concerned with the behaviour of the software externally. There is no consideration of the logic behind the item being tested, just its functionality. Here a set of expected results will be compared with the actual results from the tests to ensure that the system functions correctly.

This project makes use of black box testing across a range of different test levels – such as automated unit testing of the Dataservice logic layer.

5.2 White box testing

White box testing is where checks to the internal system are used, internally checking the algorithms that produce results. Key points such as monitoring the flow through the program are examples of white box testing.

For this project mainly black box testing was used, as the automated tests, which are black box, form a fundamental part of the testing plan. White box testing was used during development for:

- Control flow checks
- Loop count checks
- Null data checks

5.3 Levels of testing

There are a range of different levels at which software can be tested. These are based around the stages of the development process and the aspect of the system that is being targeted. According to IEEE's SWEBOK (Guide to the Software Engineering Body of Knowledge) these levels comprise of three stages of testing. These are: Unit, integration and system testing (Bourque & Fairley, 2014).

5.3.1 Unit testing

This is where small sections of the software are tested as individual units. In this system, automated unit tests were created to test the Dataservice section of the code. These can be run inside the IDE to ensure that each function call to the Dataservice works as it should. This is an example of Black Box testing.

The automated unit tests made calls to the Dataservice with demo data passed in where necessary. The automated tests are then classed as ‘passed’ if the returned data matches with the expected results.

5.3.1.1 List of Unit Tests

There are a total of 82 unit tests that were written for the project, these are shown in Table 2. The final column shows the result, with a ‘Y’ showing that the tests have passed. All of the tests pass when run.

Test File	Test Name	Result
AccountTests	ChangePasswordFailsWithInvalidPassword	Y
AccountTests	ChangePasswordFailsWithSamePassword	Y
AccountTests	ChangePasswordFailsWithWrongOldPassword	Y
AccountTests	ChangePasswordSucsessWithValidData	Y
AccountTests	GetUserFromIDFailWithInvalidID	Y
AccountTests	GetUserFromIDSucess	Y
AccountTests	GetUserFromUsernameFailWithInvalidUsername	Y
AccountTests	GetUserFromUsernameSucess	Y
AccountTests	GetUsersFromEmailFailWithInvalidEmail	Y
AccountTests	GetUsersFromEmailSucess	Y
AccountTests	RecoverPasswordFailsWithWrongEmailUsernameCombination	Y
AccountTests	RecoverPasswordSucsessWithValidData	Y
AccountTests	RegisterUserFailWithExistingUser	Y
AccountTests	RegisterUserFailWithInvalidData	Y
AccountTests	RegisterUserSuccess	Y
AccountTests	SignInUserFailsWithWrongPassword	Y
AccountTests	SignInUserFailsWithWrongUsername	Y
AccountTests	SignInUserSuccessWithValidData	Y
CustomParksTests	AddCustomParkFailsWithExistingCustomPark	Y
CustomParksTests	AddCustomParkFailsWithExistingIncludedPark	Y
CustomParksTests	AddCustomParkFailsWithInvalidParkName	Y
CustomParksTests	AddCustomParkPassesWithVaidParkDataDuplicateParkCode	Y
CustomParksTests	AddCustomParkPassesWithValidPark	Y
CustomParksTests	AddRideToParkFailsWithExistingRide	Y
CustomParksTests	AddRideToParkFailsWithInvalidRideName	Y
CustomParksTests	AddRideToParkPassesWithValidDataCustomPark	Y
CustomParksTests	AddRideToParkPassesWithValidDataIncludedPark	Y
CustomParksTests	DeleteRideFromParkFailedWithInvalidRide	Y
CustomParksTests	DeleteRideFromParkSucess	Y
CustomParksTests	RenameRideFailsWithInvalidData	Y
CustomParksTests	RenameRideSucessWithJustEditStats	Y
ParkDataTests	GetAllParksIncCustomFailsWithInvalidUser	Y
ParkDataTests	GetAllParksIncCustomReturnsAllData	Y
ParkDataTests	GetAllParksReturnsAllData	Y
ParkDataTests	GetParkFromIDFailsWhenCustomParkandUserIDsDoNotMatch	Y
ParkDataTests	GetParkFromIDFailsWithInvalidParkID	Y
ParkDataTests	GetParkFromIDPassesWithValidDataCustomPark	Y
ParkDataTests	GetParkFromIDPassesWithValidDataStandardPark	Y
ParkDataTests	GetRideListForParkFailsWithInvalidUser	Y
ParkDataTests	GetRideListForParkReturnsAllData	Y
ParkDataTests	GetRideListForParkReturnsAllDataCustomPark	Y
ParkDataTests	GetRideListForParkReturnsAllDataWithCustomRides	Y
RecommendationsTests	GetListOfUnratedRidesFailInvalidUser	Y
RecommendationsTests	GetListOfUnratedRidesSucess	Y

RecommendationsTests	GetOverallAllUserRideRatingsSucess	Y
RecommendationsTests	GetRecommendationsFailsWithInvalidUser	Y
RecommendationsTests	GetRecommendationsReturnsData	Y
RecommendationsTests	GetRideRatingsForUserFailInvalidUser	Y
RecommendationsTests	GetRideRatingsForUserSucess	Y
RecommendationsTests	InsertRecommendationFailDuplicateRideID	Y
RecommendationsTests	InsertRecommendationFailInvalidRideID	Y
RecommendationsTests	InsertRecommendationFailInvalidUserID	Y
RecommendationsTests	InsertRecommendationSucess	Y
RecommendationsTests	InsertRideRatingSucess	Y
RecommendationsTests	RateRecommendationFailsNoRecommendation	Y
RecommendationsTests	RateRecommendationSucess	Y
RecommendationsTests	UpdateRideRatingFailInvalidRatingID	Y
RecommendationsTests	UpdateRideRatingSucess	Y
TotalsTests	GetLeaderboardReturnsAllData	Y
TotalsTests	GetTotalsInvalidUserFails	Y
TotalsTests	GetTotalsReturnsAllDataNoYear	Y
TotalsTests	GetTotalsReturnsLessDataWithYear	Y
TotalsTests	GetTotalsReturnsNoDataWithInvalidYear	Y
TripTests	AddRideToTripPassesWithValidData	Y
TripTests	CreateTripFailsWhenAlreadyExists	Y
TripTests	CreateTripFailsWithInvalidData	Y
TripTests	CreateTripPassesWithValidData	Y
TripTests	DeleteRideFromTripFailedWithInvalidRide	Y
TripTests	DeleteRideFromTripSucess	Y
TripTests	DeleteTripFailsWithInvalidTrip	Y
TripTests	DeleteTripSucess	Y
TripTests	GetAllTripsFailsWithWrongUser	Y
TripTests	GetAllTripsReturnsAllData	Y
TripTests	GetTripContentsFailWithInvalidTrip	Y
TripTests	GetTripContentsSucessfullWithValidTrip	Y
TripTests	GetTripDoesNotErrorWithEmptyTrip	Y
TripTests	GetTripFailsWithInValidTrip	Y
TripTests	GetTripPassesWithValidTrip	Y
TripTests	GetTripTotalsDoesNotErrorWithEmptyTrip	Y
TripTests	GetTripTotalsFailWithInvalidTrip	Y
TripTests	GetTripTotalsSucessfullWithValidTrip	Y
UnclassifiedTests	GetAllManufacturersReturnsValues	Y

TABLE 2 - LIST OF UNIT TESTS CREATED TO TEST DATASERVICE

5.3.2 Integration testing

Integration testing covers the areas where smaller areas of the system are integrated and tested together. This aims to show any errors that occur with the interaction between different

parts of the system. As this system consists of many different components, it is important to test that everything integrates properly. For this project, integration testing was carried out manually by checking key areas of the site that cover a large amount of integration between the components of the system, in a black box methodology.

These key areas are:

- Creating an account
- Signing in
- Creating a trip and opening it
- Adding a ride to a trip
- Deleting a ride from a trip
- Viewing the trip totals and statistics
- Creating a custom park
- Adding and removing custom rides from the park
- Viewing overall totals and statistics, with filtering and sorting applied
- Viewing the leader board
- Getting and rating a ride recommendation

These tests were repeated where necessary for all three of the platforms (Android app, iOS app and website). These tests could have been automated through the use of tools such as Selenium etc. as this would increase the speed at which the tests could be completed, as well as potentially increasing their accuracy. However for the scope of this project, the tests were completed manually – as the overhead of setting up the testing system on three platforms would have taken a considerable amount of time.

5.3.3 System testing

This stage of testing looks at the system as a whole to check that it meets the requirements outlined when the system was designed. As the components of the system cover a wide range of platforms, a selection of manual and automated checks were selected to ensure that important areas where issues could occur are checked. These are detailed below.

5.3.3.1 Website Specific Tests

The website side of the project also has specific considerations that could affect the performance of the system.

- The website was tested in a range of browsers to ensure that the site looks and acts the same across a selection of platforms
- Testing of the website on mobile devices ensured that the responsive design of the website allowed the user to use the site properly whilst using a mobile device.
- Page load times were minimised to ensure that the users did not have to wait for data to load. This could be done through the use of techniques such as ensuring correct image sizes and compression to reduce the time taken to load images.

5.3.3.2 Load and Speed Tests

The system needed to be tested to ensure that the site could load quickly, including when a number of users were all accessing the site together. Many theme parks are located in places where mobile phone signal is limited and therefore the internet connection will be slower, therefore it is important that the system can react quickly. The system was also put under load testing to see if it could handle a number of concurrent users.

To complete these tests, automatic tests in Apache JMeter were created that tested the speed and average page speed on a range of pages with 10 concurrent users. Here a selection of tasks were performed that were particularly intensive on the database and server setup, to ensure that the vulnerable parts of the system can deal with higher volumes of traffic. These areas are:

- Viewing the leader board
- Loading the trips page
- Selecting and viewing a trip, including the trip statistics and sorted ride list
- Viewing overall ride totals and statistics on the totals page
- Loading the list of custom parks and parks available for the user to edit
- Viewing the recommender page, including the rides that the user has rated.
- The test also included the home page (a static page with no data on it) as a benchmark, to compare with the other data.

Table 3 shows the results of the load tests. For comparison they were run on both the local development version of the site that was running on the same computer that the tests were run on, and the live server. The average page size in bytes was also recorded for comparison.

Page	Local Page Load (ms)	Live Page Load (ms)	Average Page Size (bytes)
Home Page (static page)	3	1885	11097
Leader board	165	2703	13949
Trips Page	105	1519	15937.5
View Trip Page	410	5615	34458.5
Totals	289	2373	27645.5
Custom Parks	177	2153	56642.5
Recommender	355	3318	28071
AVERAGE	**215**	**2795**	**26828.75**

TABLE 3 - LOAD TESTS ON LOCAL AND LIVE SITE

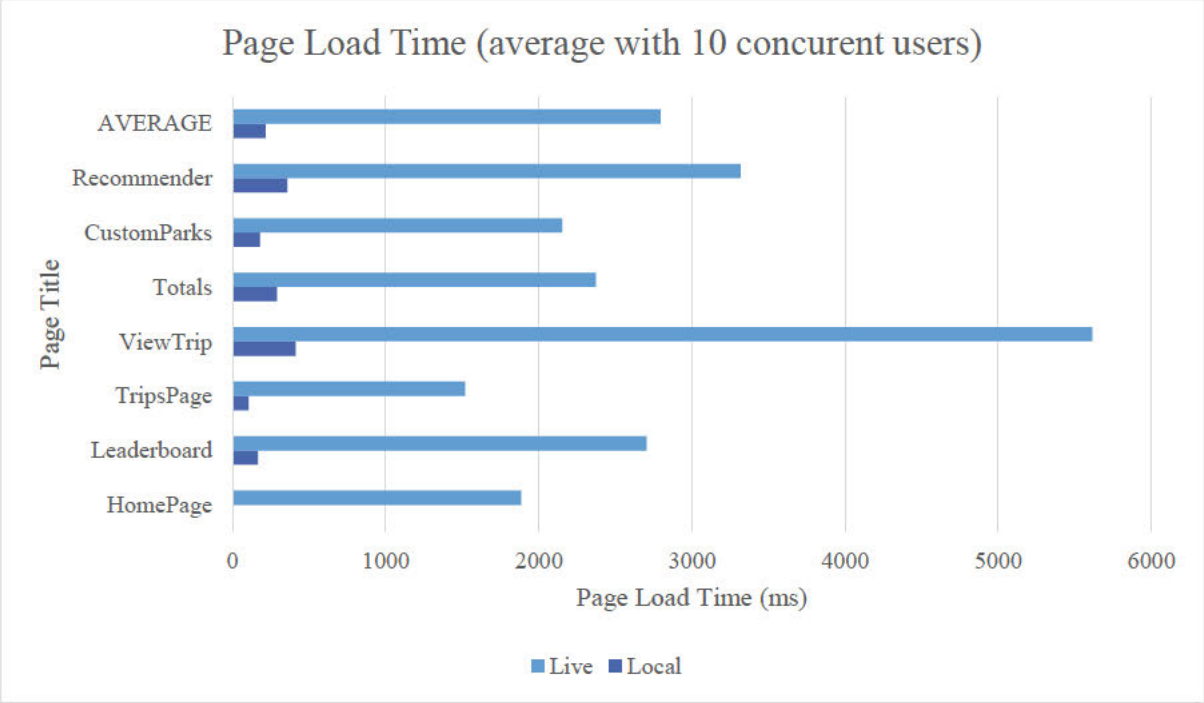


FIGURE 30 - GRAPH SHOWING THE AVERAGE PAGE LOAD TIME

These results shown in Figure 30 show that there was a significant speed difference on the live server compared with the local development server. This points to there being an issue with the live system setup rather than the code behind the site itself. There also seems to be little correlation between the page load time and the size of the page – as the largest page (custom parks) was one of the quicker pages to load, therefore this can be used to rule out major issues with the speed of the internet connection between the computer where the tests were being run, and the live server.

To investigate this issue further, I put the leader board page (as this is a data driven page that can be accessed when logged out) through Google's PageSpeed Insights tester tool. Figure 31 shows that the results from Google also suggested that the main problem causing slow load times were related to the server itself rather than the code, as the code fixes were listed as lower priority fixes.

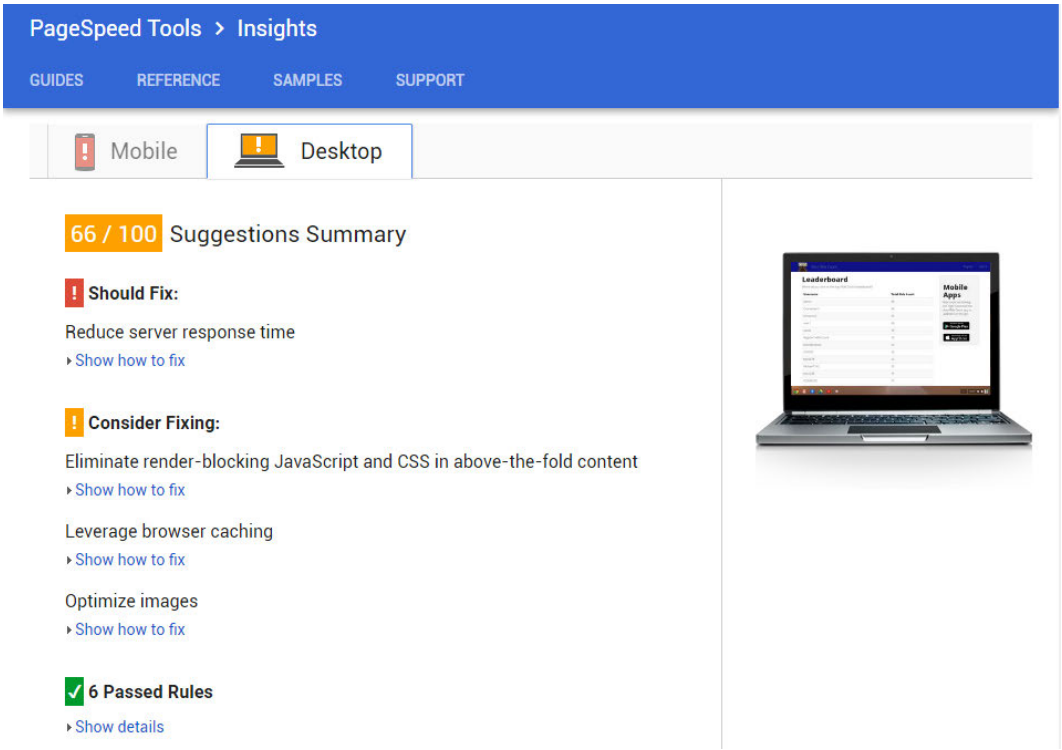


FIGURE 31 - GOOGLE'S PAGESPEED INSIGHT TOOL RESULTS

Logging onto the server showed that the memory usage was consistently high - around 80% (Figure 32), whereas the CPU was low. This could point to an issue with the server not being powerful enough. Therefore to improve the speed of the system, a server with more memory could be commissioned.

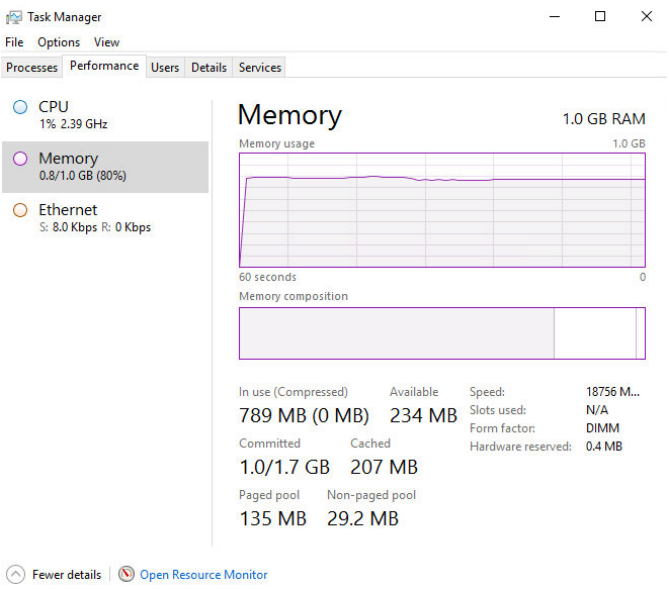


FIGURE 32 - SCREENSHOT OF WINDOWS TASK MANAGER ON THE LIVE SERVER SHOWING MEMORY USAGE

Another issue could have been the location of the server. The server was located in AWS's US West datacentre, whereas the database server was located in the UK. The tests were also being run in the UK. The poor site speed therefore could have been caused by network latency that was increased due to the scattered locations of the live servers used for the project. Therefore, moving the site's hosting to a UK server, together with combining the website code and database hosting into a combined hosting solution (meaning that the traffic would be inside one datacentre) could also help to increase the speed of the live site.

5.4 Geolocation Testing

To test the geolocation part of the project, a testing version of the app was installed onto a selection of different mobile devices (this ensures that the feature worked on phones of different age, specification and operating system). The devices were then taken around the park and onto the rides to emulate how an average user would use the app. Information about the state of the geolocation services (e.g. what rides are near to the user) was logged to the device so that it could be viewed later, to gain an understanding of any points of failure in the code. Due to the limited resources available, the system was only able to be tested at two parks, Blackpool Pleasure Beach (BPB) and Alton Towers (AT).

Table 4 shows the results of the tests, with the full test results being available in appendix 2. Each test was run on three devices during each testing day. This is because the all of the testing devices were not available for every park visit. Where a device has not been available N/A has been inserted into the results. The initial tests of the code at Blackpool Pleasure beach were 33.4% accurate on average. Following this test, improvements to the code were made. These included: Moving ride station coordinates to be nearer to the ride station and closer to the start of the ride, and the modifications to a timer that leaves the full power ride identifier running if the user is still in the general ride area (for example in a queue). Following these changes, the accuracy was on average higher, achieving 48.3% at Blackpool Pleasure Beach, and 75% at Alton Towers.

Device Name	Code Version 1	Code Version 2	
	BPB	BPB	AT
Samsung Galaxy S3 (2012)	28.6%	N/A	75%
LG G4 (2015)	28.6%	61.1%	62.5%
Vodaphone Smart Mini 7 (2016)	42.9%	33.3%	87.5%
iPhone 5 (2012)	N/A	50%	N/A
Average	**33.4%**	**48.3%**	**75%**

TABLE 4 - GEOLOCATION TESTING RESULTS

Blackpool Pleasure Beach was generally less accurate than Alton Towers due to the compact nature of the park. Blackpool Pleasure Beach has 41 rides in a 42 acre site – whereas Alton Towers has 40 rides in a 910 acre site. Because of this, the geolocation code has more criteria that the rides need to satisfy at Blackpool Pleasure Beach to help differentiate rides that are close together. These could be the height or speed of the ride. These criteria are often harder to meet – as things such as the speed of the ride can vary depending on what the weather is like, how many people are on the ride etc. Therefore the lower accuracy of the geolocation feature at Blackpool Pleasure Beach is to be expected.

On investigation of the testing logs, there were no specific rides that did not work in the geolocation features – all of the rides tested worked at least once. When a ride was not identified it was generally due to the inaccuracy of the GPS data that meant that the user was not reported as being within a ride area. As the accuracy of GPS cannot be improved, these results are considered acceptable. Rides were generally not miss-identified. There were a couple of instances were a ride was added that the user did not ride, however these were exceptional circumstances where the rider had gone into the ride area but not fully completed the ride (e.g. during a ride breakdown) – therefore these results can be discarded.

5.4.1 Battery Usage

One of the key points from the research was the importance of minimising the battery usage when using the geolocation feature. This was a hard task, as the geolocation had to be recorded to a high level of accuracy throughout the entire day. On the Android version of the app, Google's Fused Location API was used, which allowed for lower power consumption when getting the user's location. Figure 33 shows that on Android, the testing version of the Auto Ride Count app had a relatively low impact on the battery life – with 13% battery usage being reported for the Samsung S3 device, and 3% being reported for the newer Vodaphone Smart Mini 7 test device.

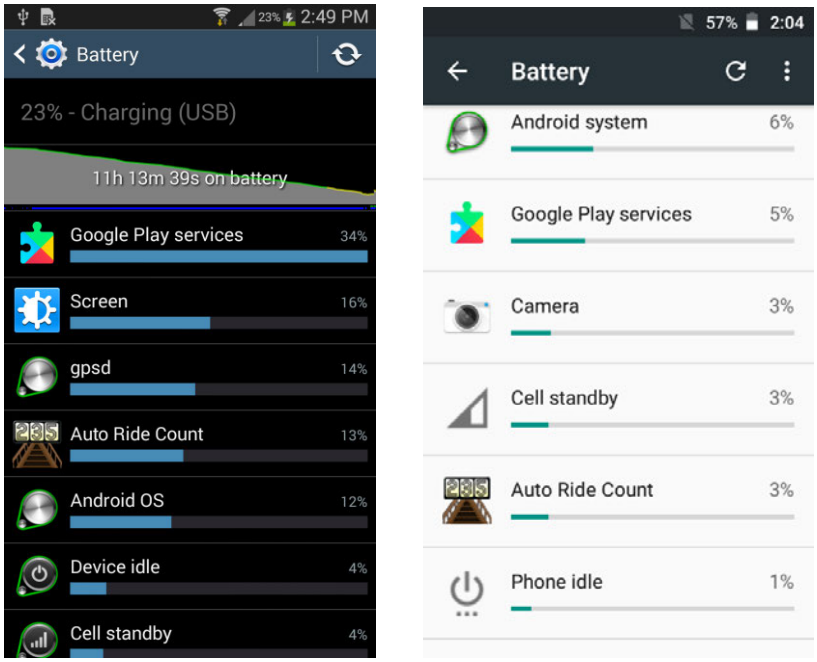


FIGURE 33 - SCREENSHOTS FROM THE S3 (LEFT) AND VODAPHONE (RIGHT) TEST PHONES SHOWING BATTERY USAGE

The iPhone version of the app had a higher battery usage rate, as Google's Fused Location API, which includes low power location features, was not available on iOS. This resulted in a 62% battery drain on the iPhone 5 test device (Figure 34). As an Android specific, proprietary API was used on the Android app – it is not possible to reduce the power consumption by using that system on the iOS platform. Therefore, since the standard geolocation APIs were used throughout the entire day at a theme park, a high level of battery usage is to be expected.

FIGURE 34 - SCREENSHOT OF THE IPHONE 5 TEST PHONE SHOWING BATTERY USAGE

5.5 Summary

This project made use of a range of testing techniques to help ensure that the system functions correctly, including in expected use. This included the use of automated unit tests, white box and integration tests, as well as website specific tests regarding the compatibility and viewing of the site on a range of platforms. The tests also included speed and load tests, to ensure that the site can cope under load. The results of the speed tests on the live site were not as expected, so the reasoning behind this was investigated. The project also made use of specific testing and inspection of the geolocation features and the power consumption efficiency of the used location APIs.

6 Evaluation

This chapter looks the user survey that was used to investigate what the users thought of the system that was created and to check if the system met its objectives outlined in the introduction. Reflections on the project, its success and what could be improved upon in the future are also included where relevant.

6.1 Survey Plan

The system underwent acceptance testing, this is where the system is tested with real users to ensure that it conforms to the user's expectations. Users can be part of alpha testing, where the users are part of a private testing group that are typically closely involved with the developer, or beta testing, where a more open audience are used. In this project, a small group of 13 alpha testers were selected.

For this project, users were questioned using an online survey to measure the success of the system in relation to its goals set out in the design, as well as checking features from the literature review for their effectiveness and accuracy. The recommendation algorithm was also tested to see if the recommendations generated are appropriate. Users that are knowledgeable about rollercoasters were selected to test this, as they were able to provide an insight into the suitability of the recommendation system.

The survey was split into different sections, each one covering a different aspect of the project. There were also sections to gather information about the participant (e.g. if they were an existing user of Auto Ride Count, and their main phone operating system). The sections were as follows:

- Introduction
    - Containing user information questions.
- The usability of the web system

    - Here the users were asked to complete a list of tasks on the site, then asked questions on elements of the site using a Likert scale (Strongly Disagree, Disagree, Neither, Agree, Strongly Agree) and a not applicable option to measure their opinions.
- The recommendation system
    - Similar to the usability section, the users were asked to rate aspects of the recommendation system on the Likert scale.
- The usability of the mobile system
    - Here the users were shown screenshots of the changes made to the mobile app on Android and iOS.
    - Like the other sections, they were then asked to rate aspects of the changes on the Likert scale.
- The geolocation feature
    - Here users were shown a short video demonstrating the 'Full Auto' geolocation changes at Alton Towers.
    - The users were then asked if they would use the geolocation feature (With Yes/No/Maybe/Other selection options). Following this, the users were then asked to rate a set of reasons that might affect their decision for using the 'Full Auto' feature on a scale from 'Not Important' to 'Very Important'.
- Other comments section
    - This section contained text boxes for users to write down any problems they experienced during the test as well as a box for any other comments regarding the system.

6.2 Survey Results

Key points from the survey have been selected that are specifically linked to the research carried out as part of the literature review. The entire set of results can be found in Appendix D.

6.2.1 Usability

The test users used the site on a range of devices (Figure 35), some users selected more than one option, meaning that they had viewed the site on more than one platform. 100% of the users tested selected 'strongly agree' when asked if all the content in the site was displayed correctly. This shows that there were no errors with the displaying of the site, even on platforms with different screen sizes such as mobile devices and tablets.

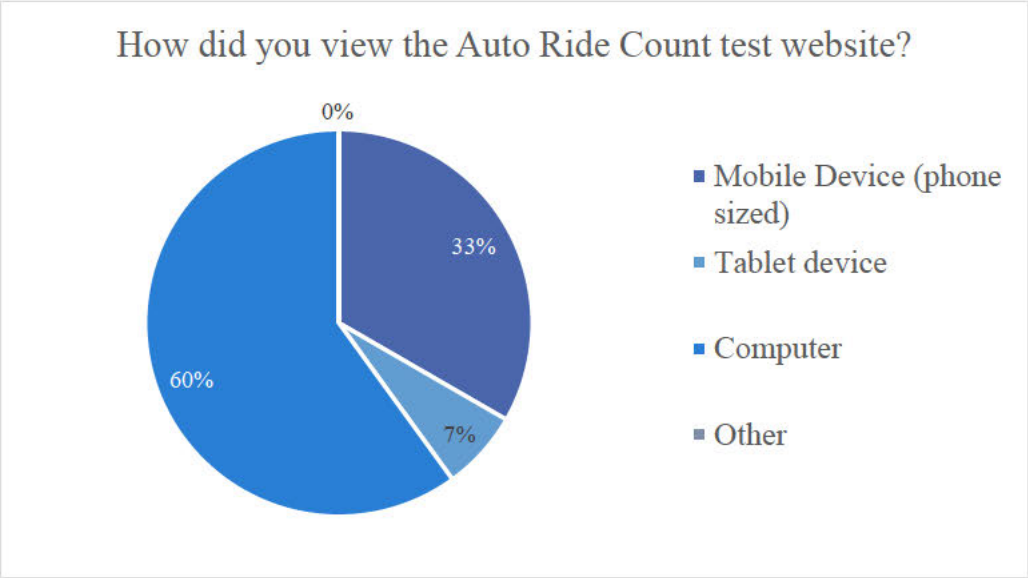


FIGURE 35 - GRAPH SHOWING THE DEVICES THE USERS USED TO VIEW THE TEST SITE

All of the users felt that they were given enough guidance on how to use the site, with 69% saying they strongly agreed, and the remaining 31% agreeing. 92% of users also agreed or strongly agreed that it was easy to locate things on the site. This shows that the content of the site is well displayed in an intuitive way, and contains the correct amount of guidance on how to use the site.

92% of users agreed or strongly agreed that if an error occurred, they could recover and continue what they wanted to do. This was a key point from the usability research, especially linking to rule three and five from Shneiderman's Eight Golden Rules of Interface Design.

One area that the users did not fully agree with was the design consistency between platforms. This was another key point from the usability research, and was the first rule of the rules of interface design. The majority of users did think that the design was similar across all platforms – however 23% of users did not agree (Figure 36).

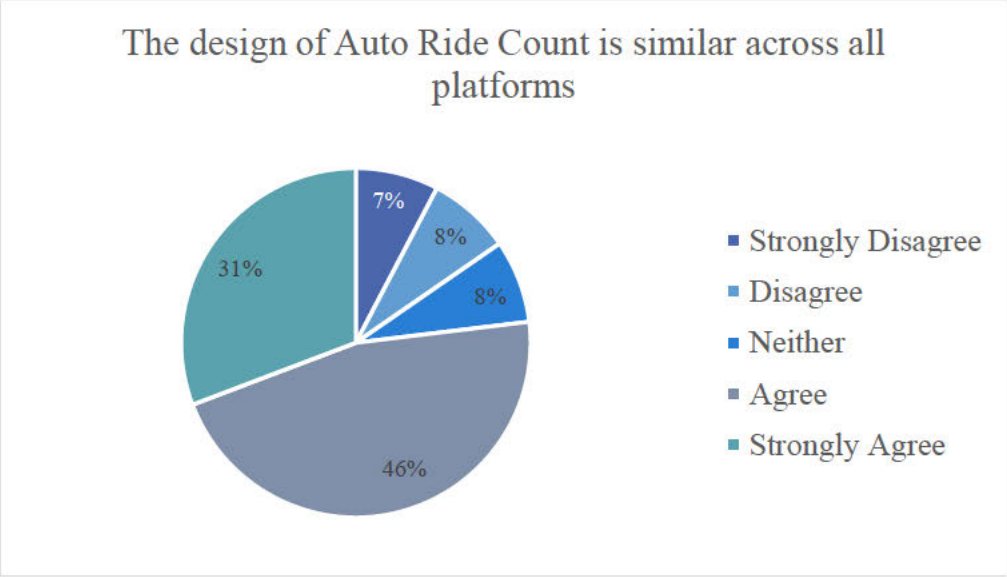


FIGURE 36 - GRAPH SHOWING USER RESULTS OF DESIGN CONSISTENCY

This was supported in the comments section of the survey, with one user writing: *"The ios and android apps need to be themed the same. The white modern theme on the ios app looks much cleaner than the blue/black on android."* This is a valid point, as the pre-existing Android version of the app does use darker colours, especially for the lists, than the iOS app. Therefore it may be considered that the UI colour choices for Android need updating to match the ones on iOS, however this was not within the scope of this project therefore can be considered for future work.

Another key point from the research was the focus on ensuring that existing users of the system do not have to re-learn how to use it. This is especially important in this system as there are users of the existing mobile apps who will therefore receive the new features that have been created for this project. Figure 37 shows that 8% of the users replied with not applicable which suggests that they are not an existing user. The rest of the replied positively to the question, with 84% strongly agreeing that the additional features had been added in a non-intrusive way. This shows that the designs of the system successfully catered for existing users.

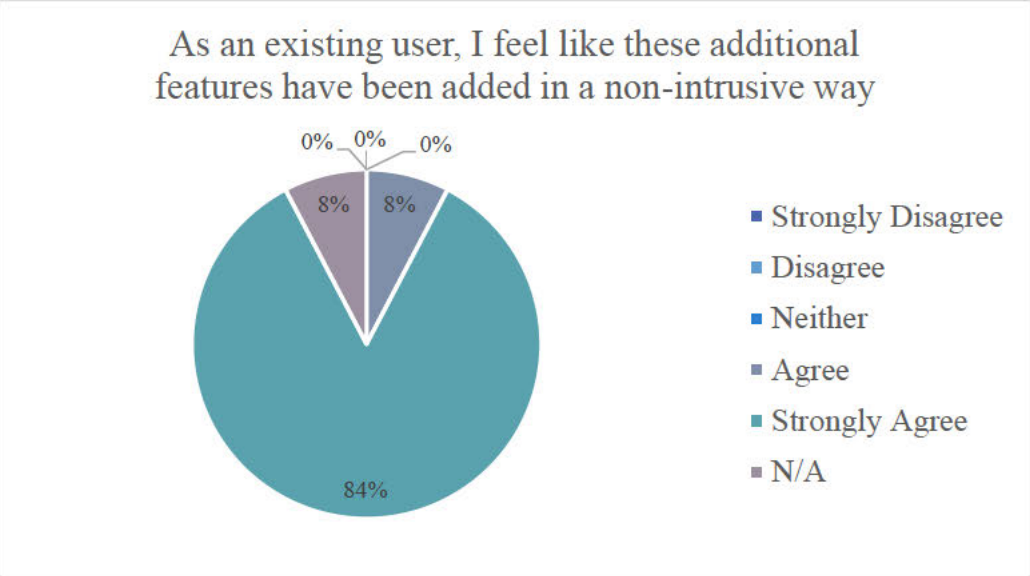


FIGURE 37 - GRAPH SHOWING EXISTING USER'S THOUGHTS ON CHANGES TO THE SYSTEM

Another part of the research that was validated using the survey was the importance of cross-platform support for the ride counting system and also the importance of having the data accessible on all devices. All of the users selected that they 'Agree' or 'Strongly Agree' that their trip data is accessible on all the devices. This helps to further show that the synchronisation of user data – a key part of this project – is very important to the users.

6.2.2 Recommendations

The recommendation part of the system had a separate section in the user survey. This ensured that the key points of this system could be reviewed separately from the rest of the system. Overall, the users found the recommendations feature fairly easy to use, with 77% of users either selecting ‘agree’ or ‘strongly agree’. There were some points of user feedback in the comments section of the survey. One user reported that they were able to add the same ride multiple times. On further testing, it seems possible that if the button to add a ride rating is clicked multiple times in quick succession before the page refreshes, it is possible to add a ride multiple times. This bug can be fixed in the next update of the site. Another user comment was an idea of pre-populating the page with the list of rides that the user has ridden, instead of making the user add the rides to the list manually. This comment needs careful consideration, as it is important to not overwhelm the user with a long list of rides as soon as they view this page.

The majority of users (77%) agreed or strongly agreed that they could trust the recommendation system to give a good ride recommendation, and all of the users responded that they agreed or strongly agreed that they understood how the recommender worked. This was a key point from the research of the recommendation systems – that it is important that the users feel that they can trust the recommendation system.

The load speed of the recommendations was rated less positively. Figure 38 shows that only 38% of users agreed that the recommendations loaded quickly. Due to the overall site speed issues mentioned in the previous chapter, it is important to consider that this could have had an impact on this result. Therefore if more work was to be done on this system, it would be important to investigate this further to see the speed results on a system that was running at a higher overall response time. The research in the literature review did suggest that some sites found that speed of recommendations was an issue, and so chose to run the recommendation

system separately, and store a set of recommendations for the user for them to view at a later time. If, on further investigation into the speed of the rest of the site, it is deemed that the recommendations are noticeably slower than the rest of the site, then this approach could be used.

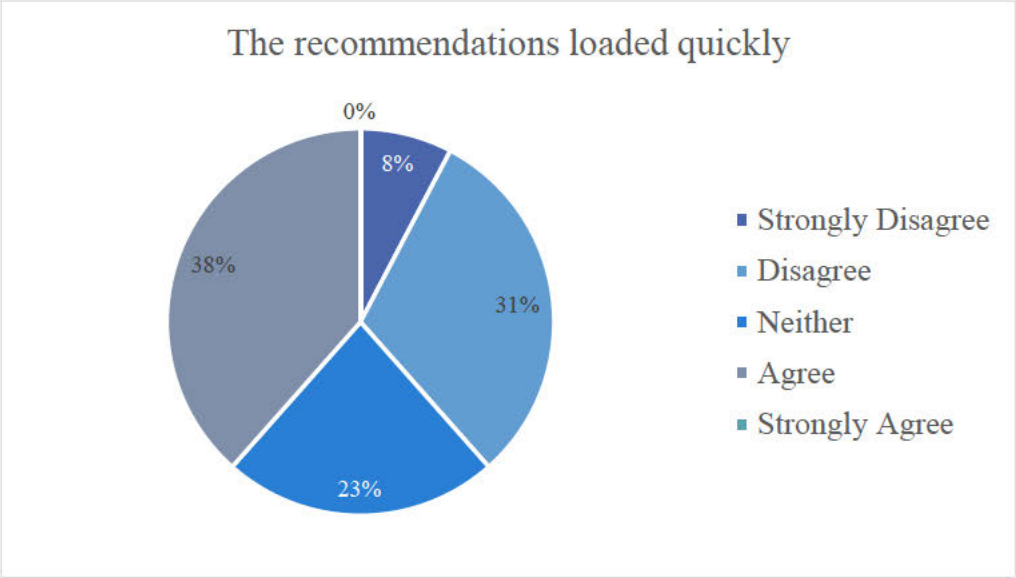


FIGURE 38 - GRAPH SHOWING USER'S OPINIONS ON RECOMMENDATION SYSTEM SPEED

There was one user who reported that the recommendation feature did not work for them when they tried rating rides at Lightwater Valley. On further inspection, it seems that there are few ride ratings in the database for Lightwater Valley. Since the recommendation system uses collaborative filtering to get a recommendation, this lack of data would explain why the user could not get a recommendation. This issue would naturally resolve itself when the system is sent live to a wider audience, as the more data in the system will lead to better recommendations. A solution for testing purposes could have been to limit the parks that the user could use in the recommendation system further, so that there would be more data for the part that they were testing.

The other users that tested the system found the recommendations to be accurate. Figure 39 shows that the majority of users agreed that the recommendations were accurate, with 38% strongly agreeing. This shows that, despite a few issues, the recommendation system was overall a success.

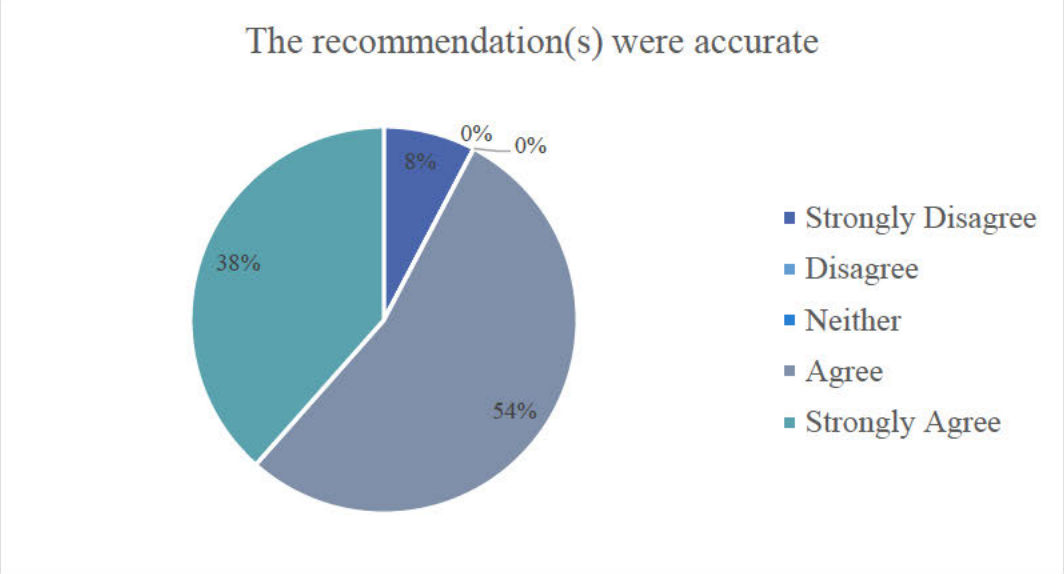


FIGURE 39 - GRAPH SHOWING USER'S OPINIONS ON THE ACCURACY OF THE RECOMMENDATION SYSTEM

6.2.3 Geolocation

After watching the video that explained and demonstrated the 'Full Auto' changes to the geolocation side of the mobile app, 77% of users said that they would be interested in using this feature, with the remaining 23% saying that they might be interested in using it. This shows that there is an interest in this feature.

The users were also asked to rate what would affect their decision on using the 'Full Auto' feature, with each reason having a rating from not important to very important. The results in Figure 40 show that battery usage and accuracy show a high level of importance for the users, with all of the users selecting either 'Important' or 'Very Important'. The rest of the points: Park availability, ride selection and privacy of the geolocation data seemed of mixed

importance. Linking this to the geolocation research completed for this project, it seems surprising that the users tested were not very concerned about the privacy of their location data. Therefore as part of future work, sending the user's location data to the server to be processed there may not be as much of a concern as initially thought.

The high concern over battery usage highlights the importance of using low power consumption location APIs – such Google's fused location provider that was used as part of the modifications to the geolocation part of the Android app. It is therefore important that in future work, a solution to a battery efficient geolocation solution is found, as high power consumption on iOS was found, during testing, to be one of the flaws with the modifications to the system that were part of this project.

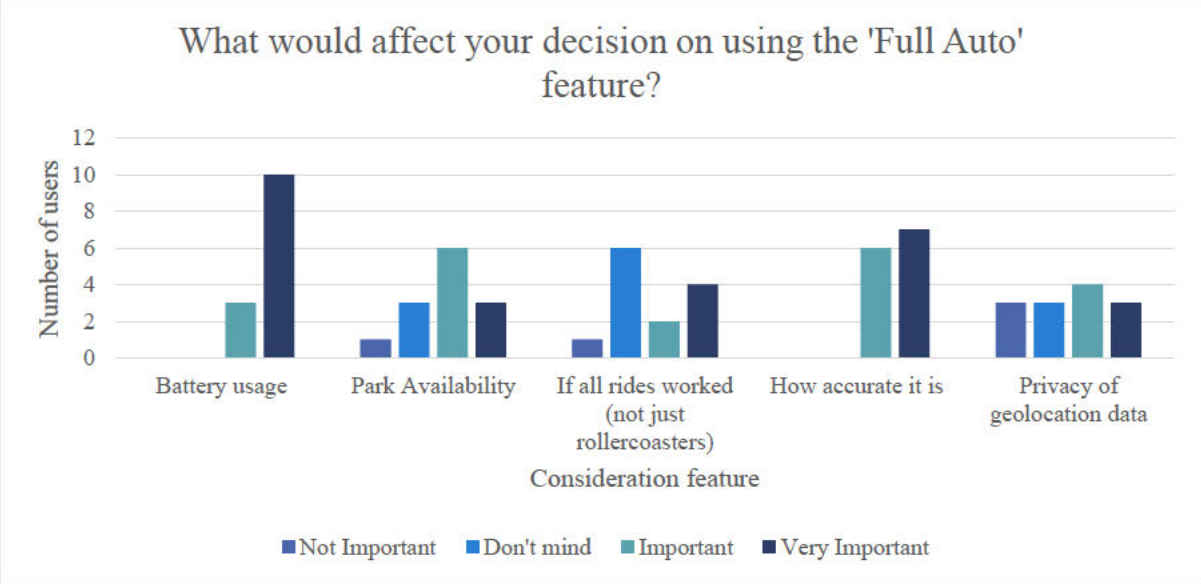


FIGURE 40 - GRAPH SHOWING USER'S OPINIONS ON THE 'FULL AUTO' FEATURE

6.3 Summary

This section shows that the user's response to the project was very positive overall. Specific points and suggestions were raised that can be addressed as part of improvements to the system in further work. Many of the results to the questions covered show positive views on

the features that were implemented following concepts and ideas that were investigated as part of the research section. This shows a good support of research driven improvements to the project area.

7 Conclusion

This chapter aims to recap over the project as a whole, looking into each stage of the project and discuss some of the key point that contributed to the progression and outcome of the project.

The chapter also discusses some of the personal views of the author. This includes a commentary of what aspects of the project went well, and what parts of the project could have gone more smoothly.

Finally, there is a discussion about what could be done in the future that is related to this project. This includes the improvements and expansion of the project itself, as well as additional areas related to the project which could be explored.

7.1 Review

The main basic objective of the project was to take an initial concept, investigate the areas of the topic to find research based solutions and best practices then use these to design and implement a solution. This solution was then tested and evaluated using a range of techniques to ensure its suitability.

The initial concept for the project was based around the construction of a web framework for ride counting at theme parks. This would then involve the creation of a website, as well as the substantial modifications to existing mobile apps. A key consideration during these modifications was the usability of the system. This was also a key part of the investigation into existing literature, with many important points that would affect the design of the system being raised. This included Shneiderman's "Eight Golden Rules of Interface Design", these rules were examined and assessed for their impact into the system. Points such as consistency across platforms and the reduction of information shown to the user were observed and then later applied in the designs and then implementation of the project. This resulted in a key discussion point with the user testing group during the evaluation chapter, where the

consistency between the three platforms (Android, iOS and web) was assessed by the users. Suggestions and ideas for future work were also generated regarding this point. Other concepts uncovered during the usability research were ideas regarding the careful integration of new features into existing systems. This was especially important for this project as substantial modifications had to be made to the existing mobile apps, both to allow it to be integrated into the web framework as well as the implementation of new features such as the new 'Full Auto' geolocation system. The success of this was measured in the evaluation section to ensure that existing users were not inconvenienced with the addition of the new features.

Other new features were added to the project. One of these was the recommendations system. The existing research materials that were investigated as part of the literature review heavily influenced the design and implementation of the project. In this project, the recommendation system is a unique feature which increases user engagement with the site, helping increase how much time the users spend on the site. Different types of recommendation systems were considered, with the chosen one being a collaborative based system that would take into account other users' preferences when recommending a ride. This resulted in a recommendation system being implemented that created high quality ride recommendations. The speed of recommendation features was investigated, and what considerations have been made to improve the speed of existing recommendation systems. This resulted in a recommendation system being created that would generate live recommendations, which could be adapted in the future if the need arose, to run separately from the web system at a time when less of a speed impact would be noticeable.

Another new feature that was added to the mobile apps was the 'Full Auto' geolocation feature. This was also investigated in the literature review stage, which included an overall look at geolocation on a mobile device, as well as a closer look at some of the side effects of

location-aware features, such as battery life and privacy concerns. These were then considered further in the design chapter, with the discussion of the languages and platforms that would allow these features to be implemented. The implementation and testing chapter then looked at how these features were added to the app, as well as looking at their resulting accuracy for detecting the rides that had been ridden. The implications of these features were then discussed further in the evaluation chapter, with the users reacting positively to the feature but also highlighting issues – such as battery concerns – which formed part of the research and design of the project.

By considering existing systems in the literature review, the system created for this project could take into consideration a range of features that were currently available to users, then these features could be added, and improved upon where necessary in the system for this project. These points were also evaluated with the users during the evaluation section of this project. The overall positive response from the test users shows that the project was a success. This is largely thanks to the careful research, design, implementation and testing which this project focused on.

7.2 Views from the author

The introduction chapter outlined the initial aims for this project as follows: *"This project aimed to create a structured framework for a web connected ride count system. This included a data driven website and suitable protocols to connect to existing mobile apps. There were also substantial modifications to the apps to make them interface with the framework. New features were implemented in the system, such as the 'Full Auto' geolocation ride detection system, and a ride recommendation feature."* All of these points have been successfully met, which I believe is a major achievement. The scale of this project was one of the main tasks that had to be overcome. The creation and management of a system that runs over numerous

code bases, on numerous platforms, in a range of languages was a large task, however the use of modular code with carefully designed links, alongside careful task and code management using appropriate tools such as Trello and Git allowed for the project to be constructed in a carefully architectured way that followed the design successfully. I think this was one of the areas that was the most challenging for me, but was an area which proved to be key to the project's success.

This project covered areas that I had a basic understanding such as geolocation, as well as areas that were completely new to me such as the recommendation system and the advanced ideas and reasoning behind usability and HCI concepts. By doing this research driven project, I have been able to learn a lot about these areas. This understanding of the concepts and ideas behind the system has been an interesting part of the development of the project. I have also learnt a lot of practical code skills. These include: Key C# techniques and coding practices, the use of unit tests, some technologies and frameworks – such as WCF, ASP.NET Web API and Web Forms. I have also learnt a great deal about server setup and security. This practical knowledge will be very useful in the future, both as part of the expansion and growth of this project in the future as well as in my career.

There are some areas of the project that did not go as well as hoped. A major one of these is the speed of the live system that was used for testing. The reasoning behind this has been investigated in the testing chapter. It was beyond the scope and timescale of this project to rectify the server issues, and this has been suggested in the further work section. There were also a few other concerns that came out of the user testing, such as the reliability of the recommendation system when there was a small amount of data to work within the system, the design of the Android app and how its styles integrate into those of the iOS app and those created for the website for this project. There were also concerns regarding high battery usage whilst using the 'Full Auto' feature on iOS. All of these issues however have been

investigated as part of this project, and changes to them would be beyond the scope of this project, and therefore could be considered for future work.

I think that if I was to approach this project again with the knowledge and understanding that I have now, I would make a few alterations. I would remove the focus of the mobile apps from the project, moving the focus of the project onto the website with a small aspect of the framework for apps. As these were existing products, making modifications to them to integrate them into the project was complicated and a particularly difficult part of the project. I also feel that a standalone project that didn't involve the existing apps would make the overall project clearer and easier for others to understand. Having said that, having the mobile apps as a part of the project has taught me a lot about the modifications required when integrating new code into existing 'legacy code', ensuring that the modifications are done in a clear way that integrates carefully with the existing code. This is an invaluable lesson which will serve me throughout further projects and my career.

7.3 Further work suggestions

As this project was designed to be a real, working system, there are a wide range of changes and improvements that can be made to this system in the future to ensure that it remains useful, convenient and fresh for users. Many of these suggestions have been outlined in this and previous chapters as key issues from this project that could be improved upon. These include: Speed improvements achieved through the re-organisation of servers and improved battery consumption on the 'Full Auto' feature on iOS. Other new features that could be investigated are further increases into the ride statistics that are available to the users, an improved ride ratings system, taking into consideration the suggestions that were provided by the users during the evaluation stage of this project.

Another part of future works could be the integration of other existing apps that the author has created into the web framework that was created for this project. These existing apps also function using ride statistic information, so connecting them to this centralised framework would bring a range of improvements to the users, such as: New features, quicker updating ride content (e.g. when a park adds a new ride) and possible user account integration.

Other future work could look at further investigations into recommendation systems, with an aim of integrating user location and distance to the user's current location to provide ride recommendations whilst at a park. This could be integrated into either the existing mobile apps, or as new apps that integrate into the web framework that was setup for this project.

7.4 Summary

This project's aims were to implement a web framework for ride counting, including the addition of new features such as ride recommendations and geolocation – as well as the substantial modifications of existing mobile apps. Through the use of research in the key areas of the project, as well as analysis of existing solutions, a high quality product was designed and implemented. This implementation was then tested and evaluated – including the surveying of test users to ensure the system's suitability, which resulted in an overall positive response. As this project has met its aims and requirements it can be considered successful. Areas for improvement, such as the speed of the server, were highlighted – and these, along with other suggestions around further addition features, further research and apps form part of the further work suggestions.

8 References

Apple, 2016. *iOS Human Interface Guidelines.* [Online]

Available at: https://developer.apple.com/ios/human-interface-guidelines/

[Accessed 13 November 2016].

Blackpool Pleasure Beach, 2017. *Copyright.* [Online]

Available at: https://www.blackpoolpleasurebeach.com/copyright/

[Accessed 17 April 2017].

British Standards Institute., 1998. *Ergonomic requirements for office work with visual display terminals (VDTs). Guidance on usability.* s.l.:British Standards Institute..

Crowcroft, J. et al., 2010. *Recommending social events from mobile phone location data.* s.l., IEEE, pp. 971-976.

Davidson, J. et al., 2010. *The YouTube Video Recommendation System.* New York, ACM, pp. 293-296.

DeLuca, K., 2015. *Selling or Spying: The Legal Implications of Target Marketing Through Geolocation Technologies.* s.l.:s.n.

Djuknic, G. M. & Richton, R. E., 2001. Geolocation and assisted GPS. *Computer,* 34(2), pp. 123-125.

Glenford, M. . J., Sandler, C. & Badgett, T., 2011. *The art of software testing.* 3 ed. New Jersey: John Wiley & Sons..

Google, 2016. *Material Design.* [Online]

Available at: https://material.google.com/

[Accessed 13 November 2016].

Google, n.d. *Getting the Last Known Location.* [Online]

Available at: https://developer.android.com/training/location/retrieve-current.html

[Accessed 7 January 2017].

GroupLens Research - University of Minnesota, 2016. *MovieLens home.* [Online]

Available at: movielens.umn.edu

[Accessed 19 November 2016].

Herlocker, J. L., Konstan, J. A. & Riedl, J., 2000. *Explaining Collaborative Filtering Recommendations.* New York, ACM, pp. 241-250.

IDC, 2016. *Smartphone OS Market Share, 2016 Q2.* [Online]

Available at: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[Accessed 30 October 2016].

Internet Advertising Bureau UK, 2016. *Mobile Advertiser Snapshot Study 2015.* [Online]

Available at: http://www.iabuk.net/research/library/mobile-advertiser-snapshot-study-2015

[Accessed 2016 November 2016].

Krug, S., 2006. *Don't Make Me Think! A Common Sense Approach to Web Usability.* 2nd ed. California: New Riders.

Lella, A. & Lipsman, A., 2014. *The U.S. Mobile App Report.* [Online]

Available at: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report?cs_edgescape_cc=GB

[Accessed 30 October 2016].

Lennox, A., 2016. *Industry Reports - The Wifi Experience.* [Online]

Available at: http://interpark.co.uk/the-wifi-experience/

[Accessed 23 November 2016].

Leroux, B. & Charland, A., 2011. Mobile application development: web vs. native. *Communications of the ACM*, May, pp. 49-53.

Linden, G., Smith, B. & York, J., 2003. Amazon.com recommendations: item-to-item collaborative filtering,. *IEEE Internet Computing*, Volume 7, pp. 76-80.

Melville, P., Mooney, R. J. & Nagarajan, R., 2002. *Content-Boosted Collaborative Filtering for Improved Recommendations.* Edmonton,, s.n., pp. 187-192.

Nielsen, J., 2012. *Usability 101: Introduction to Usability.* [Online]
Available at: https://www.nngroup.com/articles/usability-101-introduction-to-usability/
[Accessed 20 November 2016].

Nyphoria, 2016. *Coaster Counter on Google Play.* [Online]
Available at:
https://play.google.com/store/apps/details?id=com.coastercounter.nyphoria.coastercounter
[Accessed 30 October 2016].

OWASP, 2017. *OWASP Top Ten for 2017 Release Candidate.* [Online]
Available at:
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2017_Release_Candidate
[Accessed 18 March 2017].

park.expert, 2007. *ridecount.* [Online]
Available at: https://ridecount.com/
[Accessed 30 October 2016].

Pazzani, M. J. & Billsus, D., 2007. Content-Based Recommendation Systems. In: P. Brusilovsky, A. Kobsa & W. Nejdl, eds. *The Adaptive Web.* Berlin: Springer, pp. 325-341.

Phelan, K. V., Chen, H.-T. & Haney, M., 2013. “Like” and “Check-in”: how hotels utilize Facebook as an effective marketing tool. *Journal of Hospitality & Tourism Technology*, pp. 134-154.

Ricci, F., Rokach , L. & Shapira, B., 2011. Introduction to Recommender Systems Handbook. In: *Recommender Systems Handbook.* s.l.:Springer, pp. 1-35.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J., 2000. *Application of Dimensionality Reduction in Recommender System - A Case Study,* Minneapolis,: GroupLens Research Group / Army HPC Research Center.

Sauer, V. & Thumann, T., 2014. *Coaster Count Home.* [Online]
Available at: http://coaster-count.com/
[Accessed 30 October 2016].

Schafer, J. B., Konstan, J. & Riedl, J., 1999. *Recommender Systems in e-Commerce.* New York, NY, USA, ACM, pp. 158--166.

Shackelford, A., 2016. *7 Ways to Increase Member Engagement with Your App.* [Online]
Available at: http://associationmediaandpublishing.org/sidebar/7-Ways-to-Increase-Member-Engagement-with-Your-App?&Sort=
[Accessed 31 October 2016].

Shneiderman, B., Plaisant, C., Cohen, M. & Jacobs, S., 2010. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* 5th ed. s.l.:Pearson Education.

Sinha, R. & Swearingen, K., 2002. *The Role of Transparency in Recommender Systems.* New York, ACM.

Tarasewich, P. & Gong, J., 2004. *Guidelines for handheld mobile device interface design.* s.l., s.n., pp. 3751-3756.

Theme Park Guide, 2015. *Ride Count Worldwide on Google Play.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.vb.themeparkguides
[Accessed 30 October 2016].

Themed Entertainment Association (TEA), 2016. *TEA/AECOM 2015 Theme Index and Museum Index: The Global Attractions Attendance Report 2015.* [Online]
Available at: http://www.teaconnect.org/images/files/TEA_160_611852_160525.pdf
[Accessed 30 October 2016].

Towers Times, n.d. *YourDay.* [Online]
Available at: http://www.towerstimes.co.uk/history/from-the-archives/yourday/
[Accessed 13 March 2017].

Weisenberger, N., 2016. *Download the Coaster101 Coaster Counter Template Now.* [Online]
Available at: http://www.coaster101.com/2016/11/03/download-coaster101-coaster-counter-template-now/
[Accessed 19 November 2016].

Zichermann, G. & Cunningham, C., 2011. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps.* 1st ed. California: O'Reilly.

9 Appendices

9.1 Terms of reference

Title

A Full-Stack System for Rollercoaster Tracking

Student: [redacted]

Supervisor: Dr Paraskevas Yiapanis

Course specific learning outcomes

- To develop an understanding of the nature of databases and be able to develop, maintain and interrogate databases;
- To develop knowledge of computer hardware and an understanding of how the selection of hardware will affect the performance of an application;
- To investigate the interaction between hardware and software and the influence of this interaction on the design of computer systems;
- To study the fundamentals of computer network communications and communication protocols; and
- To study the management and security of networked systems.

Project Background

Rollercoaster enthusiasts (people who like going to theme parks and riding the rides repeatedly) often count the number of rides they have ridden for both personal reference and to compare with others. This is often done with notebooks with tally charts, spreadsheets etc. A few years ago, a mobile app was launched by the author (available on Android and iOS) that would allow the users to count what rides they go on and store it locally on the device. The app automatically recognises which rides you go on using GPS information such as the device's latitude, longitude, speed and altitude – as well as accelerometer data on some rides.

To expand and improve the app – some users have suggested having a web companion to the app that would allow users to store their ride count information remotely so that they could view it on a range of devices. This would also allow for a social media style account options where you can compare ride counts.

In this project, a website will be created that allows the users to upload their ride counts to an online account. Using the site they can then compare ride count, look at high scores and leader boards. The project will also include a framework that will allow the website to integrate into the existing apps. These apps will have to be substantially modified to allow them to connect to the site – this will include modifications to how the data is stored and retrieved on the phone, as well as major modifications to the user interfaces. The project will also include the introduction of a remotely hosted ride list that the app will pull down from the website to use. This will also involve modifications to the app, as it is currently hardcoded in. The project will also cover the choices made with regard to hosting, security and maintainability of the system – as the website will have to be suitable for me to keep running in the future.

Subject to satisfactory completion of the above. The project will also include the addition of a ride recommendation system. This will look at the characteristics of each ride (e.g. the manufacturer, classification, construction year etc.). The system could also take into consideration what other users have ridden that is similar – for example if user A has ridden ride 1, 2 and 3. Whereas user B has ridden 1 and 2 then user B might receive the recommendation of ride 3.

Aim

This project aims to create a structured, data driven website with suitable protocols to connect to existing mobile apps. There will also have to be substantial modifications to the apps to make them interface with the site.

Objectives

1. Identification and prioritisation of user requirements and features. This could include MoSCow analysis etc.
2. Choose a software development design process (e.g. Waterfall, Agile)
3. Investigate and choose development languages and technologies for website and database.
4. Design of system using suitable design patterns to make the website well-structured, maintainable and secure. These designs will include relevant UML, database diagrams etc.
5. Research on user interface style – including some HCI investigations.
6. Research into recommendation algorithms for the ride recommendation system.
7. Implementation of system. This will consist of multiple parts:
    a. Database implementation using MySQL
    b. Data service back-end
    c. Website front-end
    d. App connection and modifications in a cross platform environment
8. Creation and use of a range of testing strategies – this could include using black box testing, automated unit tests, usability testing, and overall acceptance testing.
9. Choosing and deploying to a suitable hosting platform. This will include research into the suitable platforms.
10. Produce the project report and present the finished system.

Required Resources

- Visual Studio IDE
- B4A and B4i (IDEs for cross platform mobile development)
- MySQL database
- Suitable hosting for project

Tasks and Timescale

Item	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Terms of reference document																									
Ethics form document																									
System analysis																									
System design and research																									
Literature Survey document																									
Testing strategy design																									
Product design document																									
Product implemented																									
Evaluation design																									
Implementation of recommendation system																									
Testing and launch																									
Report outline																									
Draft slides																									
Practice presentation																									
Report and product submitted. Presentations held																									

Key

Yellow - University documentation

Blue - Project Item

9.2 Geolocation test results

Blackpool Pleasure Beach 12/3/2017 (Version 1 of code)

Ride	S3	G4	Vodaphone
Steeplechase			
Revolution		Got	
Avalanche	Got	Got	
Infusion			Got
Big One		Got	
Skyforce	Got	Got	Got
Ice Blast	Got		
Grand National			
Big Dipper			Got
Infusion	Got		Got
Revolution			
Blue Flyer			
Nickelodeon Streak			Got
Big Dipper			Got
Totals	*4/14 = 28.6%*	*4/14 = 28.6%*	*6/14 = 42.9%*

Overall success = 33.4%

Alton Towers 25/3/2017 (Version 2 of code)

Ride	S3	G4	Vodaphone
Nemesis	Device unavailable	Got	Got
Nemesis		Got	Got
Nemesis		Got	Got
Nemesis			Got
Runway Mine Train	Got	Got	Got
Congo River Rapids	Got		Got
Oblivion		Got	Got
Th13teen	Got		
Totals	*3/4 = 75%*	*5/8 = 62.8%*	*7/8 = 87.5%*

Overall success = 75%

Blackpool Pleasure Beach 9/4/2017 (Version 2 of code)

Ride	G4	iPhone	Vodaphone
Nickelodeon Streak			Got
Steeplechase	Got		
Skyforce	Got		Got
Revolution			
Big One	Got	Got	
Big One	Got	Got	

Big One	Got	Got	
Big Dipper	Got		Got
Infusion	Got		Got
Big One			
Grand National	Got	Got	
Wild Mouse	Got	Got	
Big Dipper		Got	
Avalanche		Got	
Grand National		Device unavailable	
Big Dipper	Got		Got
Infusion	Got		Got
Infusion			
Totals	*11/18 = 61.1%*	*7/14 = 50%*	*6/18 = 33.3%*

Overall success = 48.3%

9.3 User survey questions

Please note that this was constructed as an online web form – which has been outputted to a print version for inclusion in this project. Therefore some formatting and styles may have been altered.

Final Year Project - Questionnaire

*Required

About You

1. What is your main phone operating system? * Mark only one oval.

- ◯ Android
- ◯ iOS
- ◯ Windows Phone
- ◯ _____ Other:

2. Have you used Auto Ride Count before? * Mark only one oval.

- ◯ Yes (mainly iOS)
- ◯ Yes (mainly Android)
- ◯ No

3. Have you used any other ride counting solutions? (select as many as apply) Tick all that apply.

- ☐ Spreadsheets/notes app/other digital solution
- ☐ Notepad/written solution
- ☐ Other mobile app
- ☐ Other ride counting websites
- ☐ _____ Other:

Using the system

Please now (in another tab) open the website and complete the following tasks:

You may also browse the rest of the site, however it is important that you complete as many of these tasks as possible as there will be questions on them.

4. Task checklist *

Check if you complete the tasks Tick all that apply.

- [ ] Register for an account
- [ ] Browse to the trips page
- [ ] Create a new trip (or more than one!) at one of the following parks: Alton Towers/Blackpool Pleasure Beach/Lightwater Valley/Flamingo Land/Drayton Manor/Thorpe Park/Chessington World of Adventures
- [ ] Open the trip and add some rides to the trip (including major rollercoasters)
- [ ] Ensure you have at least 10 rides added to your account
- [ ] View the totals page to see your ride count statistics
- [ ] Go to the recommendations page and get a recommendation for a ride

5. Please say how much you agree with the following statements

* Mark only one oval per row.

	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Not applicable
I found it easy to locate things on the website	◯	◯	◯	◯	◯	◯
There is a large selection of parks and rides on the website	◯	◯	◯	◯	◯	◯
I felt like I was given enough guidance on how to use the site	◯	◯	◯	◯	◯	◯
If something went wrong, I could recover and continue what I wanted to do	◯	◯	◯	◯	◯	◯
The website worked well overall	◯	◯	◯	◯	◯	◯
All of the content of the website was displayed properly (e.g. no text being cut off)	◯	◯	◯	◯	◯	◯

6. How did you view the Auto Ride Count test website? * Tick all that apply.

- [ ] Mobile Device (phone sized)
- [ ] Tablet device
- [ ] Computer
- [ ] _____ Other:

Recommendations

These questions are based on the Ride Recommender feature only.

7. Please say how much you agree with the following statements * Mark only one oval per row.

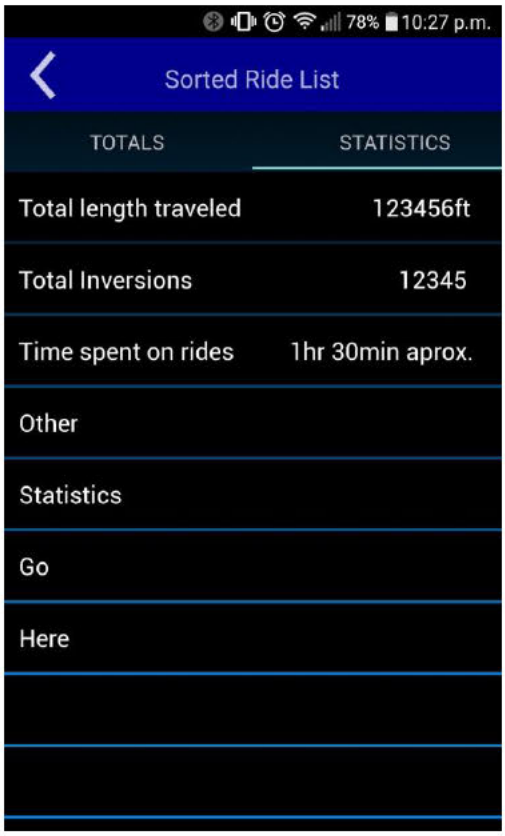
	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Not applicable
This feature was easy to use	◯	◯	◯	◯	◯	◯
The recommendations loaded quickly	◯	◯	◯	◯	◯	◯
The recommendation(s) were accurate	◯	◯	◯	◯	◯	◯
I feel like I trust the recommendation system to give a good ride recommendation	◯	◯	◯	◯	◯	◯
I felt like I understood how the recommender worked	◯	◯	◯	◯	◯	◯
I would use the ride recommender again	◯	◯	◯	◯	◯	◯

Mobile app

Please view the following two images of changes to the iOS and Android app, both show the addition of a statistics tab to the screens that are shown.

Summary View - iPhone

Sorted Ride List - Android



8. Please say how much you agree with the following statements * Mark only one oval per row.

	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Not applicable
It is important to me that I can use Auto Ride Count on a mobile device						
It is important that my trip data is accessible on all my devices						
I like being able to view ride statistics (e.g. how many inversions experienced)						
The design of Auto Ride Count is similar across all platforms (including website)						
As an existing user, I feel like these additional features have been added in a non-intrusive way						
I think it is important that Auto Ride Count is free and advert free						

Geolocation Demo Video

(Video was shown here)

Please watch this short demonstration of the 'Full Auto' Feature before continuing to the next set of questions.

9. Would this feature be something that you might use? * Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ Maybe
- ◯ _____ Other:

10. What would affect your decision on using the 'Full Auto' feature

* Mark only one oval per row.

	Not Important	Don't mind	Important	Very Important
Battery usage	◯	◯	◯	◯
Park Availability	◯	◯	◯	◯
If all rides worked (not just rollercoasters)	◯	◯	◯	◯
How accurate it is	◯	◯	◯	◯
Privacy of geolocation data	◯	◯	◯	◯

Other

11. Did you encounter any issues/bugs when using the website?

_____

_____

_____

_____

_____

12. Do you have any other comments regarding the website, app, or other part of the Auto Ride Count system?

9.4 Full user survey results

Section 1 - About You

1. What is your main phone operating system?	
iOS	6
Android	6
Proprietary	1

2. Have you used Auto Ride Count before?	
Yes (mainly iOS)	6
Yes (mainly Android)	4
No	3

3. Have you used any other ride counting solutions? (select as many as apply)	
Spreadsheets/notes app/other digital solution	5
Notepad/written solution	3
Other mobile app	3
Other ride counting websites	1
Other (text)	None, Clear (app)

Section 2 – Using the System

(Question 4 was a check list for user's guidance only)

5. Please say how much you agree with the following statements:						
	I found it easy to locate things on the website	There is a large selection of parks and rides on the website	I felt like I was given enough guidance on how to use the site	If something went wrong, I could recover and continue what I wanted to do	The website worked well overall	All of the content of the website was displayed properly (e.g. no text being cut off)
Strongly Disagree	0	0	0	0	0	0
Disagree	1	0	0	0	0	0
Neither	0	0	0	1	0	0
Agree	3	2	4	4	5	0
Strongly Agree	9	11	9	8	8	13
N/A (not applicable)	0	0	0	0	0	0

6. How did you view the Auto Ride Count test website? (multi-select)	
Mobile Device (phone sized)	5
Tablet device	1
Computer	9
Other	0

Section 3 - Recommendations

7. Please say how much you agree with the following statements:						
	This feature was easy to use	The recommendations loaded quickly	The recommendation(s) were accurate	I feel like I trust the recommendation system to give a good ride recommendation	I felt like I understood how the recommender worked	I would use the ride recommender again
Strongly Disagree	0	1	1	1	0	1

Disagree	2	4	0	2	0	1
Neither	1	3	0	0	0	1
Agree	6	5	7	3	6	5
Strongly Agree	4	0	5	7	7	5
N/A	0	0	0	0	0	0

Section 4 – Mobile App

8. Please say how much you agree with the following statements:						
	It is important to me that I can use Auto Ride Count on a mobile device	It is important that my trip data is accessible on all my devices	I like being able to view ride statistics (e.g. how many inversions experienced)	The design of Auto Ride Count is similar across all platforms (including website)	As an existing user, I feel like these additional features have been added in a non-intrusive way	I think it is important that Auto Ride Count is free and advert free
Strongly Disagree	0	0	0	1	0	1
Disagree	0	0	0	1	0	1
Neither	0	0	0	1	0	3
Agree	1	4	3	6	1	4
Strongly Agree	11	9	10	4	11	4
N/A	1	0	0	0	1	0

Section 5 – Geolocation

9. Would this feature be something that you might use?	
Yes	10
No	0
Maybe	3

10. What would affect your decision on using the 'Full Auto' feature:					
	Battery usage	Park Availability	If all rides worked (not just rollercoasters)	How accurate it is	Privacy of geolocation data
Not Important	0	1	1	0	3
Don't mind	0	3	6	0	3
Important	3	6	2	6	4
Very Important	10	3	4	7	3

Section 6 - Other

11. Did you encounter any issues/bugs when using the website? (free text)
Website was running slightly slowly, and the recommender didnt work right away (I had to keep adding more rides until the recommender worked)
Slow to load. (iOS)
I was able to add the same ride multiple times into the recommendation
No
No
Speed of updating data on Rating New Rides feature
No everything worked well,

The recommendations would not work 😳 Current amount of rides added but it wouldn't make a recommendation. (Lightwater Valley)
no
No
No
Very slow but i'm assuming that's just how you're currently hosting the site.

12. Do you have any other comments regarding the website, app, or other part of the Auto Ride Count system?
Menu text should be on the right, so it can be used with one hand. (iOS)
I would populate the ride recomender page so that all the rides I had been on were already listed and grouped by type and each ride has a default null value which could then be changed - it just save clicks. Also on the trip details page, having a back button on the page to navigate back to the trips page would improve the flow of the site for me!
The text in the top blue bar on the website blends in with the background too much, so is quite hard to notice that it is there until you hover over it
After posting data back in a form, the server can take a while to respond.
I'm really impressed with Auto ride count and the concise clear way it works.
Looks good, well presented, generally works well. ☺
Great App. Would be great to see all these extra features rolled out in future.
The ios and android apps need to be themed the same. The white modern theme on the ios app looks much cleaner than the blue/black on android. The text in the header of the website is hard to read i thought. I feel the auto ride count online homepage is a bit bland and lacks the correct branding to feel truly like it's part of the same company. But apart from that great stuff.