GoClub Android Application

An Android Application for local nightclubs to promote special deals to students.

Ross Phillips

BSc Computer Science

Submission Date: 24th April 2013

Supervisor: Graham Roberts

Abstract

This project involved the development of an Android mobile application to allow nightclubs to promote deals to students, and to help students travel to the club. The application is supported by a website that allows clubs to input the adverts displayed on the application. The aim of the project was to learn how to develop Android applications, as well as database-driven websites and use gained knowledge to develop the system. The main goal was to develop a successful and launch-able application, with a range of functionality: mapping, augmented reality, connections to social media, notifications, etc. The main challenge of the problem was to ensure users reach the club. This was achieved by providing routes to the club and an Augmented Reality view that allows users to identify clubs with their mobile phone camera.

Weekly objectives were set and development, of both the application and website, was approached in an iterative manner. The work built on existing knowledge of Java and involved learning and using new technologies including: Android, PHP + MySQL, Facebook SDK and the Augmented Reality SDK.

The result was a fully functional and heavily tested system that comprised of an Android application and website. The Application followed a number of Android's recommended design patterns. A number of new and advanced technologies have been learnt and successfully demonstrated through this project.

Contents

Chapter 1

Introduction

1.1 The Problem

People often have a wide range of choice when deciding what nightclub to go to. It often may be hard to decide where to go. Depending on the night, a club's entry price, drinks prices and event can vary greatly. A club also may not be open on some nights.

To overcome this trouble deciding, people may navigate to a couple of club's websites (if they have a computer accessible) to check what event the club has on and the entry price; however club websites are sometimes poorly maintained, holding incorrect information. If the person does not have a computer accessible, this may mean deciding on a club that they like from past experience and going there without the knowledge of entry price or information regarding the event that is happening.

To get to the club, people may use Transport for London's website prior to leaving to plan out the best route on a computer or mobile, or alternatively, they may just head to the area which they know the club is in (e.g. Leicester Square).

If the area has a high proportion of clubs, street promoters employed by the different local clubs will attempt to coerce people into going to their club. They are likely to offer discounted entry and cheaper drinks; often once at the club these special 'deals' do not materialise.

Promoters are paid by commission and pressure to get customers is high. Promotion tactics can be underhanded. They often put undue pressure on customers and sometimes rely on lying; for example, "oh you're going to Zoo Bar?" "It's just down here; I'll show you the way" then they take you to a different club.

Occasionally, special deals may be in place and a good deal can be obtained through promoters; however promoters are seen as a great nuisance by most. Promoters are usually avoided as they are not worth the hassle.

In summary, it may be hard to decide what club to go to, it is better to have knowledge of the club's event before you arrive and it is possible to get discounts through promoters, however this is usually avoided.

1.2 The Solution

To develop an Android application that will operate as an advertising platform for nightclubs to advertise discounts and special offers to students, via their mobile phones. Students can search (with a range of criteria) through adverts placed by local clubs, based on the student's current location, so the deals of nearby clubs can be found. Once the student has chosen which deal to redeem, the App will guide them to the club, via a route on a map.

This will incorporate an online advert management portal, where club management can add, edit and delete offer listings for their club. Adverts will display on the application or alternatively clubs can push adverts as notifications that will appear on the user's phone even when the application is not open.

1.3 Why I chose this project?

I chose to deal with this situation as it is something I had experienced first-hand. I and others have experienced all of the problems listed above many times. There is a clear need for a solution.

Before the individual project, I felt there was a gap in the market for an application acting as the solution, which would also be a successful commercial product. I have previously written a business plan (as part of COMPGC18 – Entrepreneurship: Theory and Practice) for the system developed in this project. To briefly explain the revenue stream, there is a lot of money to be made through promoting clubs. If the application is to launch after this project, it is envisaged clubs will either pay to place their adverts on the application, pay to 'push' adverts (more details later), or pay per customer that gets to the club (Current commissions paid to promoters range between £1 - £5 per customer).

Furthermore, there may be social benefit to existence of the application. With people travelling to clubs located throughout London, often in areas with which they are not familiar, it is important people do not get lost; especially as they may be intoxicated and it may be the middle of the night. It is also better that they are aware that the club is open and the price of entry.

1.4 Challenges

Below details the more challenging aspects of the problem:

1) One problem identified above is that reduced entry or drinks promised to customers before they enter the club may not materialise. For example, after being promised "£5 entry", once at the clubs entrance, the club denies knowledge of this and wants to charge £15 entry. The system needs to ensure that any deal placed is honoured by the club. This is so that users of the application receive benefit from using it and are satisfied with their user experience.

2) Current partial solutions (Transport for London's Journey Planner) for getting directions to the club may direct you to nearby the club. This may be to the postcode or street it is on. People may not be sure where the intended club is, just from being navigated to the vicinity. Smaller clubs are not usually marked and are sometimes a doorway down to a basement. Furthermore, promoters may attempt to make you believe one club is actually another. There needs to be a way for a person to be sure they are entering the correct club.

3) For recouping commission payments (in the future) from the clubs, there needs to be a mechanism in place that records whether a user has reached the club. The club may allow entry to the user, but not report that the user reached the club, meaning no commission is gained.

1.5 Aims and Goals

1.5.1 Aims

- To learn to develop Android applications incorporating a wide range of advanced specialised components.
- To learn to develop a database-driven website using PHP & MySQL.
- To use gained knowledge to develop a successful product that allows nightclubs to promote deals to students and direct students to nightclubs.

1.5.2 Goals

Below details the goals for the project; this gives a good overview of what was to be delivered. The system is examined in a lot more detail in the requirements document (appendix A).

1.5.2.1 Application

Application: Basic Goals	
1	The application should connect up to a webserver to retrieve adverts.
2	The application should store these adverts on an internal database for quick access.
3	The application should filter and display a list of adverts matching the user's inputted search criteria.
4	The application should display pictures of the clubs.
Application: Advanced Goals	
5	The application should sort ads by the location of the user. (Adverts are sorted nearer to further).
6	The application should provide a map with directions to the club from the user's location.
7	The application should allow users to view ratings and reviews of clubs.
8	The application should allow users to input ratings and reviews of clubs.
9	The application should post to Facebook (if given permission to) when people choose to redeem a certain offer.
10	The application should track the success of adverts, reporting statistics to the clubs through the website.
11	The application should be able to receive adverts as notifications based on the user's location (only adverts of nearby clubs are shown).
12	The application should provide an augmented reality view allowing the user to view on-screen markers over clubs when they open the devices camera through the application.

1.5.2.2 Website

Website: Basic Goals	
1	The website should provide an account system; with login and registration. Registered club promoters can log in to their own advert management dashboards.
2	The website should allow promoters to add, edit and delete adverts.
3	The website should allow promoters to upload a picture of their club.
4	The website should allow promoters to edit details stored about their club.
5	The website should look professional enough that industry would be prepared to use it.
6	The website should email clubs on registration, ensuring they validate their email before they can log-in.
7	The website should provide an admin dashboard that allows for new account approvals and

	the moderation of flagged comments.
Website: Advanced Goals	
8	The website should provide a preview of what adverts will look like on the app.
9	The website should report statistics on advert performance.
10	The website should allow advert statistics to be displayed in graphs.
11	The website should allow promoters to be able to push certain adverts; targeting the user's that receive the adverts by university.

1.6 How the project was conducted

Throughout this project, an iterative approach was followed. An initial working product with very limited functionality was first developed and then further functionality was then added throughout the project, building on what was already implemented.

The project plan document outlined a work plan for completion of the project. This work plan was later revised in the interim report. The work plan outlined work that was to be completed each week and what objective/goal this work came under. As work was completed, the individual pieces of work would be ticked off and the progress would be regularly monitored.

Weekly meetings were scheduled with the project's supervisor; these gave the opportunity to report progress, discuss problems, receive feedback and weekly demonstrations were given from the beginning. A notebook was used to record all details relating to the project. For example: questions to ask in the weekly meetings, weekly plans for work, notes from research and details of implementation.

Certain functionality could be implemented in different ways. Advantages and disadvantages of each method of implementation would be examined before deciding which method to use. This is covered further at later points (notably, the reporting of whether a user has got to the club, in section 4.2.7).

1.7 Structure of this report

Following this introductory chapter, the report is structured into chapters. Below gives a brief outline of the chapters of this report. **Chapter 2** covers background information and related work. It outlines the main sources of information that were used through this project. It then goes onto look at similar applications that are in direct competition with GoClub. This section finally outlines the software, tools, library code and frameworks that were used. **Chapter 3** covers requirements and analysis. A summary of both the requirements and Use Cases is given (with the respective full documents in the appendix) and ER diagrams are displayed. These represent the data stored by the application and website. **Chapter 4** covers design and implementation. First, a description of the systems complete architecture is given, before each of the components and their sub-parts are covered in detail. Detail of specific implementation is also given. **Chapter 5** gives a description of the testing strategy conducted during this project. For both the application and website components of the system, each different method of testing undertaken is detailed. **Chapter 6** contains the conclusions that can be drawn and evaluation of the project. Further developments to the project are then looked at before a final conclusion to the report and project is given.

Chapter 2

Background information and related work

2.1 Sources of information

Android application development and using PHP/MySQL for database driven websites (the two major parts of this project) were completely new to me at the start of the project. To be well prepared, time was spent over the summer to introduce myself and get familiar with these technologies. Detailed below are the main resources that were repeatedly referred to in this period and also throughout the project.

2.1.1 Learning Android

Learning Android[1] introduces the structure of an Android application to a beginner, as well as a range of further more complex features, such as content providers and SQLite databases. A practical example of a Twitter-style application is developed adding further functionality as the book progresses.

This book introduced the Android Activity; a single screen that the user can interact with and the lifecycle of such Activities. The lifecycle of an Activity specifies how they react to being opened, closed, paused, resumed and so on. The book also detailed how information could be passed from one Activity to another, via the use of an Intent. The final main concept learnt was Android's use of Services. This is a way for an application to execute longer running operations that may not be linked with an Activity and possibly not in the UI thread and so not blocking the user's interactions with the application. All these concepts form the basic structure of the developed application.

2.1.2 Android Developers

Android Developers [2] provide documentation on the developer SDK and information on how to get started. The Getting Started guide guides the reader through the process of downloading the Android SDK and getting it set up with the Eclipse IDE. The basic tutorials were followed to introduce myself to Android programming. Android Developers was an invaluable resource as it contains the complete class listings for the Android SDK; often these would be referred to, to fully understand the capabilities of a class.

Furthermore Android Developers provides design guidelines. These are a set of patterns, principles and recommendations identifying how an application could be developed for the best possible user experience. These guidelines were followed where possible, for example, in the development of the application's icon (See *section 4.2.10.1*).

2.1.3 The New Boston Android Application Development

The New Boston[3] is the username and now website of a YouTube user who produces a range of different tutorials on a range of programming topics. The website includes a tutorial series on Android development, which consists of 200 different tutorials. Tutorials are simple screencasts of the Eclipse IDE with a voice-over to explain what is happening. To begin with, the introduction tutorials were viewed consecutively. As experience was gained, tutorials were only viewed if they were in a particular area of development that was being focused on.

Much of my knowledge of databases on Android devices came from the 11 tutorials dedicated to SQLite databases. These tutorials taught the function calls that were required and basic implementation patterns.

2.1.4 Android Application Testing Guide

Android Application Testing Guide [4] examines a wide variety of testing techniques, frameworks, and tools that can be used to properly test an application. Each technique covered provides practical examples on the basic concepts of that technique, allowing a reader with little knowledge to follow the steps to create and run their first tests.

Through reading the book, a wide variety of testing strategies were explored, before deciding on the final testing strategy for this project. A number of different testing methodologies were researched and experimented with before the final strategy was chosen. The book also gave an introduction to Robotium, a testing framework that was used in the final testing strategy of this project.

2.1.5 Building a Database Driven Website using MySQL and PHP [5]

This book introduces PHP and MySQL, and then explains how to go onto use them together to create a database-driven website. It is primarily aimed at beginners to both PHP and MySQL and as a result, has all the practical installation information required; including downloading and setting up a LocalHost. Concepts are taught by an example running through the book, where the reader sets up a user-provided content website, adding increasingly complex functionality.

I used this book to introduce myself to PHP and MySQL; these were two technologies that I had not been taught previous to the start of the project. The book acted as the main resource when developing the database-driven website part of the system; many concepts learnt could be applied directly to the system.

2.1.6 Stack Overflow

Stack Overflow [6] is a popular question and answer website where users can ask questions relating to a wide range of programming topics. It has a substantial android section, with over 295,238 questions (at the time of writing) relating to Android programming.

Many problems that were experienced in this project were also experienced in the past by Stack Overflow users, who had posted them as a question. The public viewable answers were a useful resource to solve problems in the project. On a couple of occasions problems were experienced that had not yet been asked, so I asked these questions. Help obtained was very good and on one occasion I was praised by other users for a question that was well written and showed clear research effort.

2.2 Similar Products

Similar existing mobile applications are detailed below. Each table covers a different application. For each application a brief description of the application is given, and how the application differs from GoClub is covered.

Name	**London Clubs**
Operating System	iOS
Description	The application displays clubs on a basic Google Maps interface, clicking a club's map marker provides pictures of the club and an email form that allows users to email the club; so users place themselves on the guest list.
Reviews	Poor industry reviews, which comment "[the app] is trying to sell their own events" [7]. Poor user reviews: One user stated it was "Not Even Decent", awarding it one out of five stars. [8]
Target market	All Club goers
Further Details	https://itunes.apple.com/gb/app/london-clubs/id372580063?mt=8
How my application is different	
Functionality this application does not have, that GoClub has.	• Displays or allows the redemption of special discounts. • Routes to selected club. • Distance information. • Reviews/ratings. • Augmented reality view. • Posting to social media.
Functionality this app and GoClub share.	• Map view of clubs. • Event details.
Functionality this application has, that GoClub does not.	• More pictures can be viewed. • Higher quality pictures can be viewed. • Allows guest lists to be booked.

Name	**ClubTables**
Operating System	iOS
Description	This application allows for tables and guest lists to be booked at over 150 clubs who have agreed to work with the application. From taking the user's inputted details (including party size, desired time, booking date), the application will list available guest list spaces and bookable tables on an interactive map. When in map view, clicking on a club will result in more details being shown. The user can also search for venues nearest to them.
Reviews	Very good user reviews on the ITunes App Store. No industry reviews.
Target market	Extremely high-end market. Only contains very expensive clubs.
Further Details	http://www.clubtables.com/
How my application is different	
Functionality this application does not have, that GoClub has.	• Displays or allows the redemption of special discounts. • Routes to selected club. • Event details – Only clubs are listed, users do not know the event that is happening or whether it is open. • Distance information. • Reviews/ratings.

	• Posting to social media • Augmented reality view.
Functionality this app and GoClub share.	• Map view of clubs.
Functionality this application has, that GoClub does not.	• Allows guest lists/tables to be booked. • Higher quality pictures can be viewed.

Note: There are a number of other applications that serve an almost identical purpose to the ClubTables application detailed above. ClubTables has been chosen to represent these applications.

Name	**Time Out London**
Operating System	iOS, Android
Description	TimeOut London, the well know magazine, has produced an application that gives details on where to eat, drink and also club. The user can search, filtering to just search clubs only, and the application will return all the clubs with events today. Search results are sorted by distance. Once viewing an individual event, it is possible to view a map detailing your location and the location of the event.
Reviews	Good user reviews.
Target market	Time Out London readers / General Londoners
Further Details	https://play.google.com/store/apps/details?id=com.timeout.android
How my application is different	
Functionality this application does not have, that GoClub has.	• Routes to selected club. • Reviews/ratings. • Augmented reality view. • Posting to social media
Functionality this app and GoClub share.	• Event details – The actual event at a club on a certain night can be listed. • Distance information. • Displays or allows redemption of special discounts. • Posting to social networks.
Functionality this application has, that GoClub does not.	• None

All the applications looked at so far were present in the market when this project was started. GoClub added a range of functionality that was not included in these applications. However in December 2012[9], an initial version of an application called Socio was released (a more user-ready/bug-free version was then released in February 2013). This application shares a lot of the same functionality as GoClub. However, additional functionality was added to GoClub, such as Augmented Reality, that this application does not offer.

Name	**Socio**
Operating System	iOS, Android (However the Android version is currently unavailable)
Description	Socio connects the user with deals (uploaded by club promoters) at the nearest pubs, bars and clubs. The user can search on a range of criteria

	including location. Once at the venue, the user can share that they are there to social networks.
Reviews	No industry reviews. No user reviews.
Target market	General club goers.
Further Details	http://lovesocio.com/
How my application is different	
Functionality this application does not have, that GoClub has.	• Routes to selected club. • Reviews/ratings. • Augmented reality view.
Functionality this app and GoClub share.	• Event details – The actual event at a club on a certain night can be listed. • Distance information. • Displays or allows redemption of special discounts. • Posting to social networks.
Functionality this application has, that GoClub does not.	• None

In summary, there are a wide range of applications that are available in the nightlife/clubbing area. Many of these can be grouped into the applications that serve high-end clubs and for clients that are looking to book a table at a usually members-only club, this is not the target market of the GoClub application, which is the student market. Students are likely to be on a budget and not able to afford to go to these high-end clubs.

If positioned in the market in September 2012 (the start of the project), GoClub would have been the only application to provide special offers and discounts to mobile phone users. Furthermore there were no well-regarded applications serving the non-high-end club market.

The newly developed application, Socio, has become a direct competitor with GoClub as they both are based on the same concept, marketing special deals to the application's users. Although they are both based on the same concept, there are a number of valuable features that GoClub includes that Socio does not; these include providing a route to the club, user reviews and ratings of every club using the application and also an Augmented Reality view, so users can identify nearby clubs. The emergence of Socio into the market has confirmed that GoClub provides a desired service that current apps were not capable of doing.

Taking into consideration the whole market at the time of writing, there is no Android based application that offers the same functionality as GoClub.

2.3 Supporting Tools, Software, Libraries and frameworks

Application functionality:

2.3.1 Wikitude (SDK)

Wikitude is an award-winning Augmented Reality browser that also provides an SDK that allows applications to plug into its functionality. This functionality allows for on-screen computer generated markers to be placed, over a live-view of the device's camera view, to mark points of interest. These markers can be clicked on and further information displayed. The Wikitude SDK allows the connecting application to pass it multiple data objects, these objects are referred to as 'Points of Interest'. Points of Interest include a title, description and longitude and latitude of the object. Given the users location, along with these points, the SDK will calculate the location of these points respective to the user. It displays the points of interest over the device's camera view that is being shown on the screen. It will allow these points to be clicked, opening a new activity created in the application connecting with it.

There are a range of different Augmented Reality libraries that can be integrated into existing applications, including Metaio (http://www.metaio.com/products/sdk/), ARviewer (http://www.libregeosocial.org/node/24) and AR-Kit (https://github.com/haseman/Android-AR-Kit), however Wikitude was chosen for a number of reasons. Firstly, it specialises on location-based Augmented Reality; other SDKs, for example Metaio, specialise on image-recognition and generation of 3D graphics, something out of the scope of the project. Furthermore, it was found Wikitude has the best documentation and also a helpful support team, which were corresponded with during the integration of the SDK in the application. Finally not all SDKs have a free pricing tier. Wikitude allows for an education license to be obtained by students for free.

More information: http://www.wikitude.com/developer/

2.3.2 Facebook (SDK)

The Facebook SDK provides functionality required for the integration of Facebook within an application. This functionality includes user authentication and also allows for interactions with the Facebook Graph API. This Graph API is the way of retrieving or posting data to Facebook.

The Graph API provides two pieces of functionality that were integrated into the application. Firstly, it allows the application to retrieve information on a user once they have provided authentication details. Secondly, the Graph API provides the mechanism for posts to be made to the user's Facebook feed. More details on using the Facebook SDK for these functions are given in section 4.2.6.

Although many social networking sites were available and offer their own equivalent SDKs, Facebook was chosen primarily with the user in mind. Facebook, compared to Twitter, focuses more on the connections between existing friends. The application gives the user the option to tell their friends they are going to a certain club. Using Facebook means it is likely for people's close friends to see that they have gone to a certain club (while also providing free advertising for the club and application). Twitter statuses are usually public and are not targeted to anyone (e.g. friends) in particular. Users of the application may be less willing to share their activities and locations on Twitter as it is visible to the world.

More information: http://developers.facebook.com/android/

2.3.3 Google Directions API + Google Maps API (API)

The Google Directions API calculates directions between locations. Requests are given as HTTP requests. A wide variety of parameters can be set by placing these in the URL.

Requests take the format:

```
http://maps.googleapis.com/maps/api/directions/output?parameters
```

Parameters for the origin and destination are given as longitude and latitude values or as the textual addresses. Output can either be XML or JSON. The returned result provides a number of details about the returned route. Routes are given as longitude and latitude of each change in direction of the route between the origin and destination and also a polyline (A series of straight lines on a map) representing each segment of the route. Results can be parsed and overlaid on a map supplied by the Google Maps API. The Google Directions API and Google Maps API are used when a route to a club is provided to a user from their location.

More information: https://developers.google.com/maps/documentation/directions/

User Interface design:

2.3.4 Twitter Bootstrap (Framework)

Twitter Bootstrap is a free collection of HTML and CSS design templates for website user interface design. Twitter Bootstrap also provides JavaScript files for commonly needed event handling actions. One of its main benefits is the CSS it provides is cross-browser compatible. It also allows for partial compatibility (graceful degradation); important information displayed by the website will always be displayed, however compatible the viewing device. Twitter Bootstrap allows web developers, with a limited amount of time, to produce a professional looking user interface. Its integration was useful in this project, as it allowed more time to be spent on the required functionality. By using the design templates, a professional cross browser compatible user-interface could be developed with relative ease.

More information: http://twitter.github.com/bootstrap/

Testing:

2.3.5 UI/Application Exerciser Monkey (Tool)

UI/Application Exerciser Monkey is a command-line tool supplied with the Android SDK. It can be used for automated stress-testing of an application. Monkey generates high-speed random streams of events while operating on an application (on either the emulator or a real-device). Monkey provides a number of options to the user, including the number of events to generate and the starting seed to generate random events from (starting with the same seed will generate the same random events). Monkey will report back to the user on a successful run of events, or a stack trace of any unhandled exception received (an unsuccessful run of events).

Monkey was used in this project as an automated approach to stress test the application.

More information: http://developer.android.com/tools/help/monkey.html

2.3.6 Selenium (Tool)

Selenium IDE is a testing tool allowing for the recording - and automated playback – of interactions with a website's user interface. It allows for test scripts to be written, and then played back, to test a website's core functionality. Selenium's integration is very useful in iterative projects that add further functionality onto existing functionality. When further functionality is added, scripts can be re-run automatically checking that no existing functionality has been lost or altered.

Selenium IDE is a Firefox extension which allows for tests to be recorded via following your interactions with the screen. You act out a test case for it once and it can then repeat this unlimited times checking for certain results. If a test fails, the reports will highlight what part of the test failed and so allows for problem areas to be identified.

Selenium was used during the development of the website. As additional functionality was added, existing test scripts would be run to ensure no changes to the expected results. If there was a need for a new test case given the new functionality, this was also added along with the functionality.

More information: http://docs.seleniumhq.org/

2.3.7 Robotium (Framework)

Robotium allows for the creation of automated black-box JUnit-driven tests of possible user interactions with an android application. It serves effectively the same purpose for Android application testing as Selenium (described above) does for web interface testing. What Robotium adds to a traditional JUnit test is the solo object, which conducts interactions. Using a solo object and its related functions, Robotium can interact, like a user would, with the different Android activities and the elements that they hold. Robotium tests are written in code following the JUnit class structure (setUp(), test, and tearDown() methods). An example test could be that the solo object may be told to input some text, click a button and move to a new activity, and then confirm the text just entered is displayed successfully. As Robotium uses JUnit, 'assert' statements are used to check the results of a test and either pass or fail a test.

Robotium was used to write functional tests as the project was in progress. These tests would be re-run whenever new functionality had been added, to confirm existing functions were still in-tact.

More information: https://code.google.com/p/robotium/

Version/Source Control:

2.3.8 BitBucket (Tool)

As a version control system, BitBucket provides storage of a complete archive of the entire projects history, so that certain versions can be reverted back to if required. Work is committed to the repository from the local machine that it is held on. Each commit can be accessed from anywhere. BitBucket was used to store the code for both the application and the website components of the project. It offered a way to ensure that code was suitably backed up, on a device other than the computer it was written on, in case of hardware failure.

More information: https://bitbucket.org/

Chapter 3

Requirements and Analysis

3.1 Requirements

Below gives a description of the process of developing requirements and the structure followed when recording them. See Appendix A for the full requirements document.

3.1.1 Generation

The Introduction outlined the problem statement (*Section 1.1*); knowledge of this and personal experience of the problems experienced when visiting clubs was used to initially generate requirements. It was considered how an application could act as a solution to these problems. Existing applications were investigated to examine the functionality they offered, as it was desired to provide additional value to the user which was not already available. Once an initial concept for the application and its basic key functionality had been developed, people fitting the target market were then surveyed to ascertain whether they thought the application and its proposed key functionality was desirable. From feedback received, requirements were further refined.

3.1.2 Structure

The requirements document has been broken down into two sections for clarity; the application and the website components of the system. Inside each section, there is a substantial list of functional and non-functional requirements for the particular component.

Each requirement is given a unique ID, following the format:

[FR | NFR] [A | W] [Category Number].[Item Number]

[FR | NFR] specifies whether a requirement is a functional requirement (FR) or non-functional requirement (NFR). [A | W] identifies whether the requirement corresponds to the application (A) or website (W). The category a requirement is in is represented by the category number, and then the item number gives the individual requirement. Requirements are grouped into categories that follow the main goals of the project. Categories are useful as similar requirements can be grouped together; this allows the reader to easily inspect one of the key areas of the project and all the related requirements surrounding it.

For example, the first application functional requirement would be FRA1.1. Assigning unique IDs to each requirement is useful as the project progresses, especially in testing, where individual requirements are referenced.

Each Requirement is prioritized using the MoSCoW prioritisation system[10]. Requirements can be grouped into Must Have, Should Have, Could Have or Would Have. Must Have requirements are fundamental to the system; if the system is delivered without them, it will be useless. Should Have requirements are less vital requirements that add extra functionality to an already working system; if an initial system is delivered without them, it may still be useful. It should be expected that a very large proportion of Should Have requirements are to be delivered. Could Have requirements are requirements that could easily be left out. Their inclusion depends on whether they can be easily implemented without jeopardising higher priority requirements. Would Have requirements are requirements that can wait until later; they are noted for further developments to the system.

3.2 Use Cases

3.2.1 Application - Use Case Diagram

Fig. 3.1 shows the Use Case Diagram for the application component of the project.

Note: Actor definitions, corresponding with the actors in the diagram, can be found in *Appendix B*.

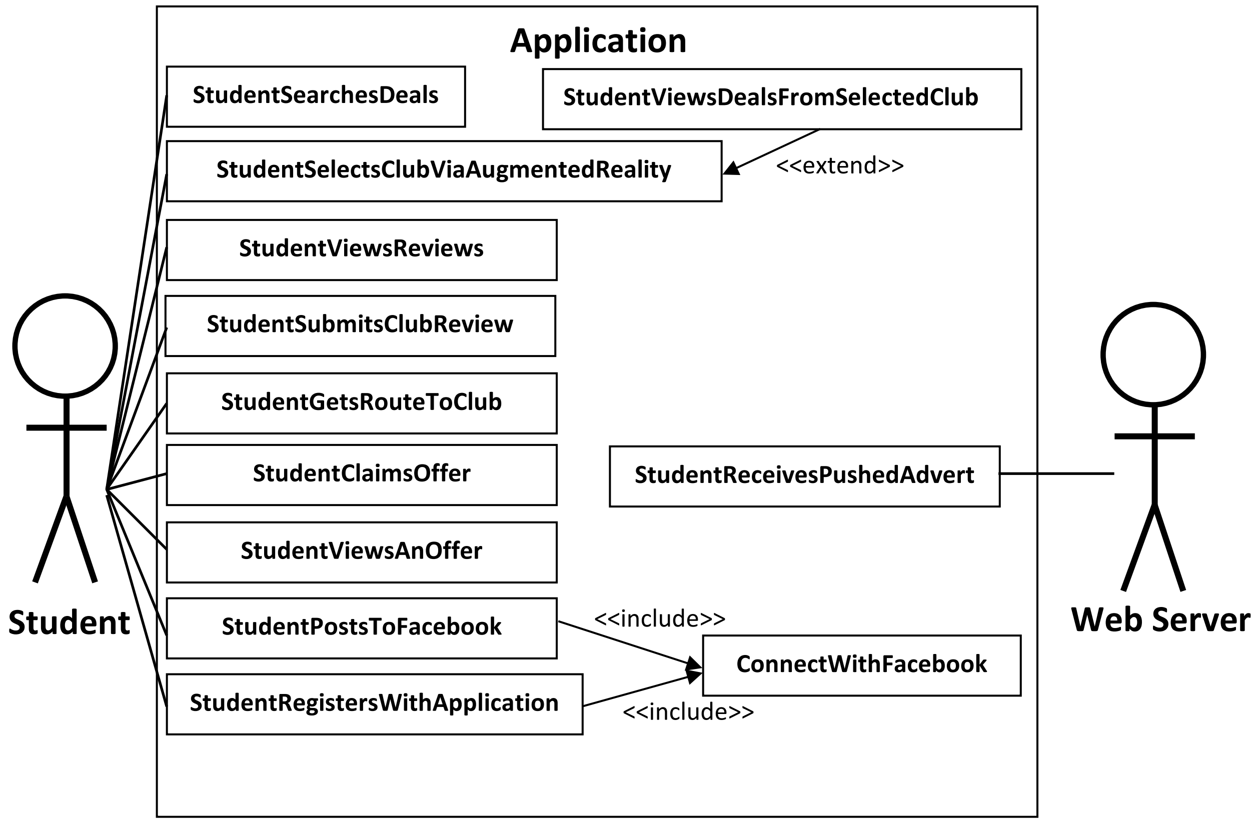


Figure 3.1 – Application Use Case Model

3.2.2 Application – Use Case Titles

A list of Use Case Titles for the application can be seen below. A full Use Case Document can be seen in *Appendix B.*

StudentRegistersWithApplication	StudentPostsToFacebook
StudentSelectsClubViaAugmentedReality	StudentGetsRouteToClub
StudentViewsDealsFromSelectedClub	StudentClaimsOffer
StudentSearchesDeals	StudentSubmitsClubReview
StudentViewsAnOffer	StudentReceivesPushedAdvert
StudentViewsReviews	ConnectWithFacebook

3.2.3 Website – Use Case Model

Fig. 3.2 shows the Use Case Diagram for the website component of the project.

Note: Actor definitions, corresponding with the actors in the diagram, can be found in *Appendix B*

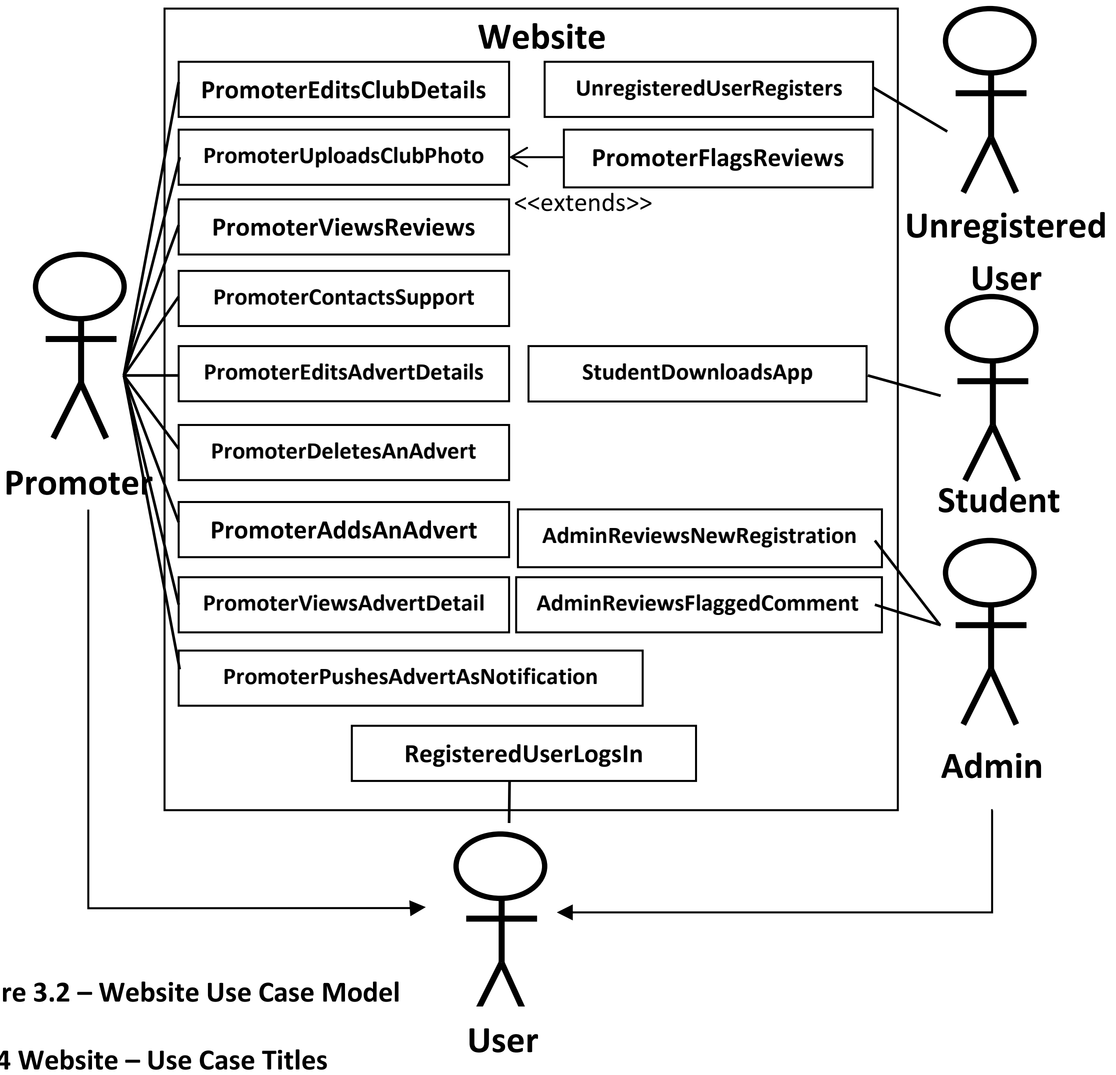


Figure 3.2 – Website Use Case Model

3.2.4 Website – Use Case Titles

A list of Use Case Titles for the website can be seen below. A full Use Case Document can be seen in *Appendix B.*

UnregisteredUserRegisters
RegisteredUserLogsIn
PromoterEditsClubDetails
PromoterUploadsClubPhoto
PromoterViewsReviews
PromoterFlagsReviews
PromoterContactsSupport
PromoterDeletesAnAdvert
PromoterAddsAnAdvert
PromoterViewsAdvertDetails
PromoterEditsAdvertDetails
PromoterPushesAdvertAsNotification
AdminReviewsNewRegistration
AdminReviewsFlaggedComment
StudentDownloadsApplication

3.3 Entity-Relationship (ER) Diagrams

3.3.1 Application Entity-Relationship (ER) Diagram

ER diagrams represent Entities and the relationships between them. Entities are items about which data is stored. Details stored about Entities are referred to as Attributes. Entities can interact with one another; this is called a Relationship. These Relationships can have a number of participants involved. [11]

The diagram in Fig. 3.3 shows the Entities involved in the application, these are: *Club, Advert, Review and ValidOn.* One club can place zero or more adverts to display on the application. Each advert is valid on one or more days. Finally each club can have zero or more Reviews. Note: distance could be placed in the Club entity – for an explanation as to why it is in the advert entity, see section 4.2.2.1.

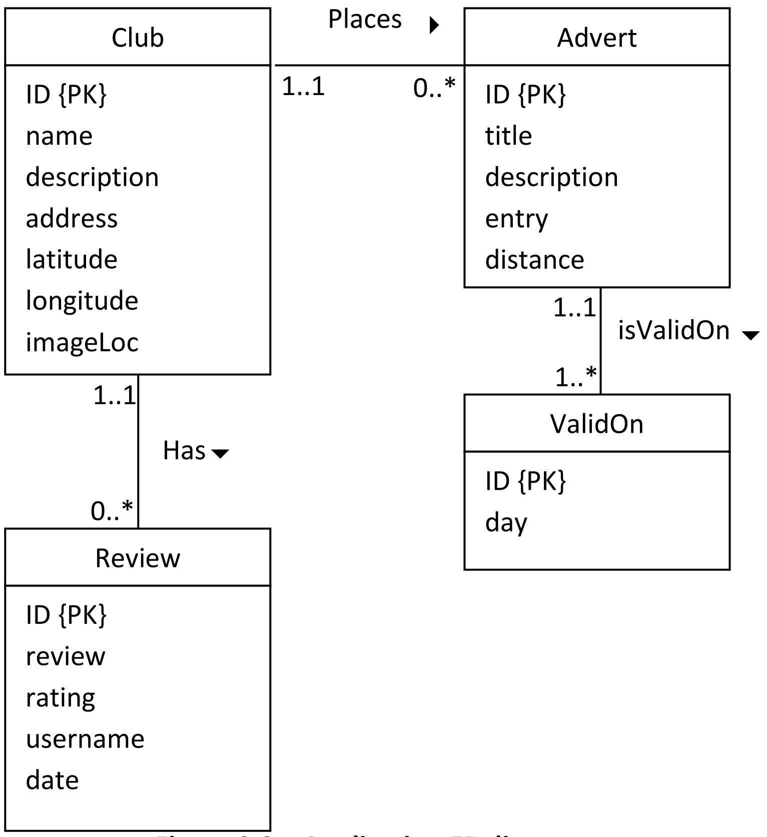


Figure 3.3 – Application ER diagram

3.3.2 Website Entity-Relationship (ER) Diagram

The website's ER diagram contains more entities than the application's ER diagram. A lot more data has to be stored on the webserver than on the application, due to additional functionality and because more data is stored on the webserver instead of directly on the application (the application then connects to the webserver and this data is retrieved to be displayed). A *user* entity exists as the website allows users to log in to their own personal accounts. There can be different types of *user*; administrators and club promoters. Each club promoter works for one *club.* A *club* can receive a *review,* and place an *advert* which is valid on a certain *day;* this is represented in the same way on the application. Clubs have a *rating* stored about them. The website also stores *statistics* on each *advert* recording the advert's usage. The website also stores details of notifications that clubs intend to send to local users. A *Notification* contains details about the *advert* that is being promoted. A user of the device (known as a *gcm_user*) can receive these notifications and can also be at an identified *location*.

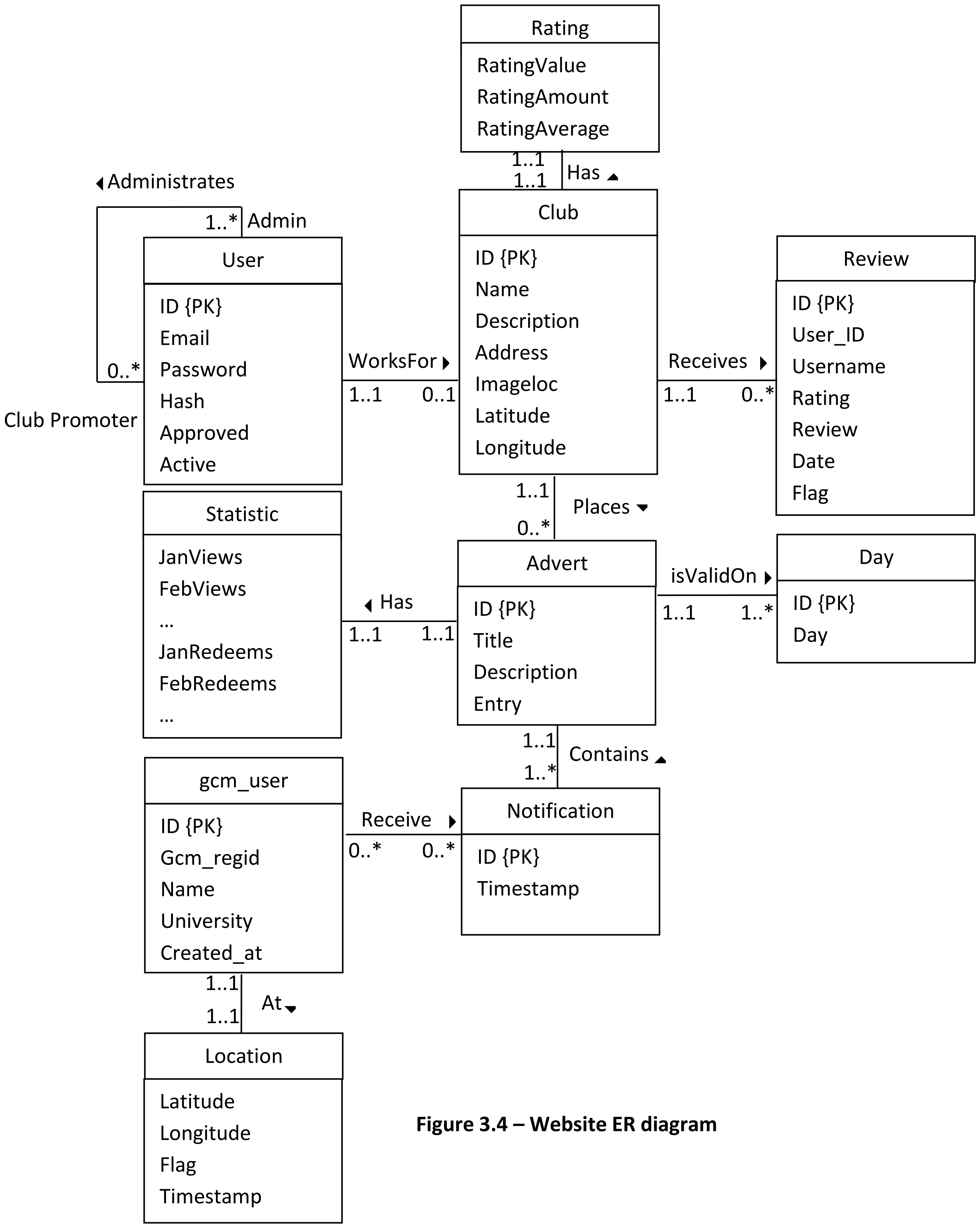


Figure 3.4 – Website ER diagram

Chapter 4

Design and Implementation

4.1 System Architecture

The system can be separated into two main components; the Android application and the back-end webserver that supports the application. The diagram in Fig. 4.1 represents this structure.

The **webserver** is shown at the top and each of its scripts that interact with the application are detailed and numbered. Each script is labelled so it can be referred to. Note: the webserver also contains the actual website that allows further functionality, however only parts that support the application directly are shown; the website functionality is not included in the diagram. The **application** can be seen at the bottom of the diagram. Each of the application's activities are shown in square boxes. The arrows between them represent the flow of execution through the application (how the user can transition between activities). The actual class name of each activity is displayed at the bottom of each box.

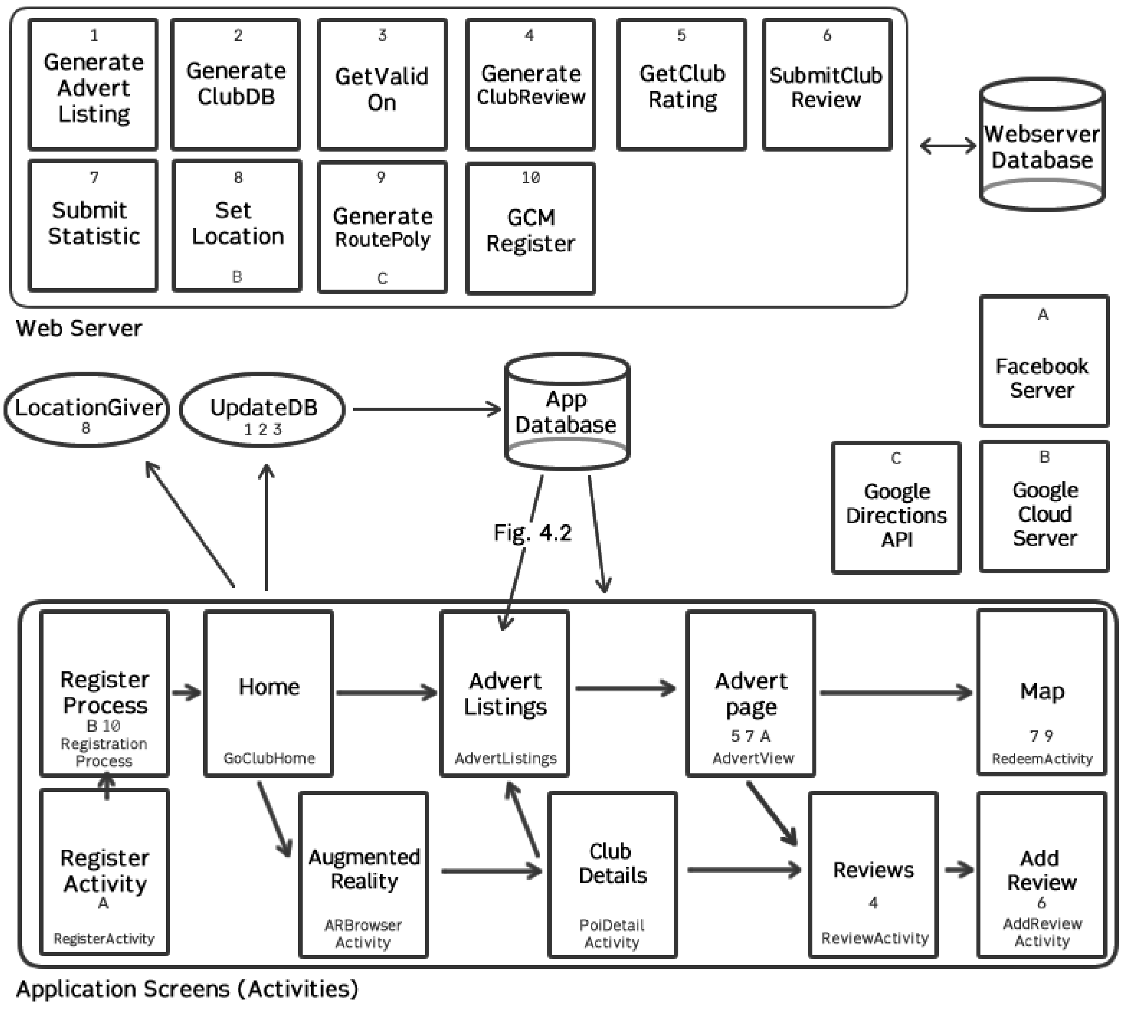


Figure 4.1 – System Architecture Overview

If one component of the diagram uses another component, this is noted by a number in the lower portion of the box. For example, the Review page on the application uses the 'GenerateClubReview' script on the webserver. These numbers show how the application utilises the webserver and third party services (labelled with letters (A,B,C)).

The home screen creates two services (shown in ellipses). One of these, 'UpdateDB', updates the database with information. The application database stores data for the applications use, with many activities retrieving data from it. More complex access is required by the Advert Listings as this requires the results in a ListView, more details on this can be found in section 4.2.3.

As Android applications are developed in Java, the application is structured into distinct classes. Each activity (shown in the diagram) requires its own class. Appendix C.1 contains a class listing, which includes descriptions of these activity classes as well as all supporting classes that were not in the diagram (details on all the applications classes are provided).

There are several different categories that the application's classes fall into. The application contains classes that extend the Activity class provided by the Android SDK. These classes (shown in Fig. 4.1) hold a User Interface (defined in XML), which the user can interact with. The Android SDK Activity class provides a number of methods that are extended to manage the Activity's lifecycle (what happens when certain events occur). The application contains two services ('*UpdateDB*' and '*LocationGiver*'); these extend the Android SDK Service class. A Service class is used to perform processes that do not interact with the user directly. Adapter classes (extending the Android SDK *SimpleCursorAdapter* class) are used to format data from the database into a list. Certain classes, such as the *AdvertCursorAdapter* class, also extend the *ContentProvider* SDK class; these are used to provide access to the database to Loaders (see section 4.2.3 for more on Adapters, ContentProviders and Loaders). There are also helper classes, such as the *DbHelper* class which is used to aid database operations as recommended by Android (see section 4.2.2.2 for more on this).

When required, an activity will call a service provided by the webserver. Appendix C.2 contains textual descriptions of each of the webserver scripts that are shown in Fig. 4.1.

4.2 Application Components

The important components that make up the application are detailed below.

4.2.1 Connecting to the webserver

The application contains a number of methods which manage transactions between the device, back-end web server and other servers. These are available throughout the application and used where required.

These functions include:

- **isNetworkConnected() –** This method checks whether an internet connection is present; returning true if present, and false otherwise. The method uses an instance of the Android *ConnectivityManager* class that can be queried regarding the network state.

- **ConnectToWebAddress() –** This method connects to the webserver using HTTP GET. The response is taken from this class and placed into an *InputStream,* then converted to a string by *inputSteamToString().*
- **PostToServer() –** This connects to the webserver using HTTP POST. The response is converted from an *InputStream* to a string using the *inputStreamToString* method.[12]
- **inputStreamToString() –** This method takes an *InputStream* and returns a *StringBuilder* (that can be converted to a string by calling *toString()*). [13]

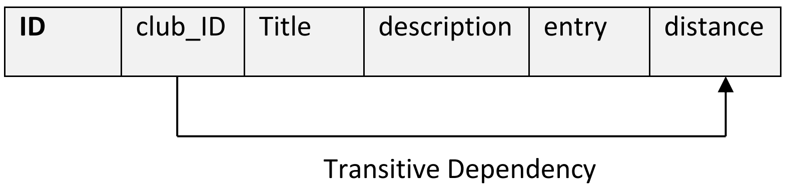
4.2.2 Database

4.2.2.1 Structure

The structure of the application's database tables are below. The diagrams show any outstanding dependency that could be removed to allow for further normalisation of the table. [14]

- All tables are in **First Normal Form (1NF)** as the intersection between rows and columns contain one and only one value.
- All tables have single-attribute primary keys. This means they are automatically in **Second Normal Form (2NF).**
- All tables, except the *advert* table are in **Third Normal Form (3NF).** Third Normal Form means that there exists no non-primary-key attribute which is transitively dependent on the primary key (meaning you can work out the attribute from another non-primary key attribute). See below as to why the advert table is not in 3NF.
- All tables (except the advert table as it does not meet the conditions of 3NF - see below), are in **Boyce-Codd Normal Form (BCNF).** Boyce-Codd Normal Form means that all determinates (an attribute that other attribute are fully functionally dependant on) are candidate keys. The only determinate in each of the tables (again excluding the advert table) is the primary key, meaning the conditions of BCNF are met.

Advert table



Originally the distance column (which represents the distance of a club from the user) was intended to be part of the *Club* table, as the distance is an attribute of the club, not of the advert. Due to a constraint, imposed by Android's *CursorLoader* class (a class for querying and returning results from the database to be formatted into a list), distance had to implemented as part of the *Advert* table. It was desired for adverts to be sorted by their distance from the user. The *CursorLoader* class only allows the results to be sorted based on a column in the table that was being queried, so the distance column has to be in the *Advert* table.

Club table

ID	name	description	location *(address)*	latitude	longitude	imageLoc

Review table

ID	club_ID	username	review	rating	date

ValidOn Table

ID	advert_ID	Day

4.2.2.2 Implementation

Android supports SQLite databases. SQLite databases are different to usual SQL databases as they require no configuration and are server-less. No configuration means that any program on an android device that can access the disk can access the database, which is stored in a single file. Server-less means any read or write execution is done directly to this file and not through a server. This has the advantage that there is no additional separate server to manage. [15]

The GoClub application follows the recommended method by Android [16] that to manage a SQLite database a Helper class (In GoClub, named *DbHelper*) is created, that extends SQLiteOpenHelper.

To do reads and writes of the database, *getWritableDatabase()* and *getReadableDatabase()* are called to retrieve a SQLiteDatabase object which represents the database. This object provides further functionality to interact with the database.

The GoClub application has an *UpdateDB* service, which runs each time the application is started, to update the database. *UpdateDB* connects to the webserver to retrieve a JSONArray of any data to store, this is then looped through and placed into the database row by row. An *ASyncTask* is used as the process can be time consuming and may block if the webserver is slow to respond.

As the file system of the device has to be accessed when the database is accessed, database operations can be slow. The next section looks at both overcoming the problem of speed, and accessing and making use of the data stored previously.

4.2.3 Content Provider and advert listings

The user is able to select criteria by which they want to search for deals. Clicking search will then display a list of matching adverts. This list of adverts is retrieved from the database and displayed on screen as a *ListView.* Fig. 4.2 outlines the overview of this process. In this section, the three main components will be covered: 1) The Content Provider 2) The Loader 3) The Adapter.

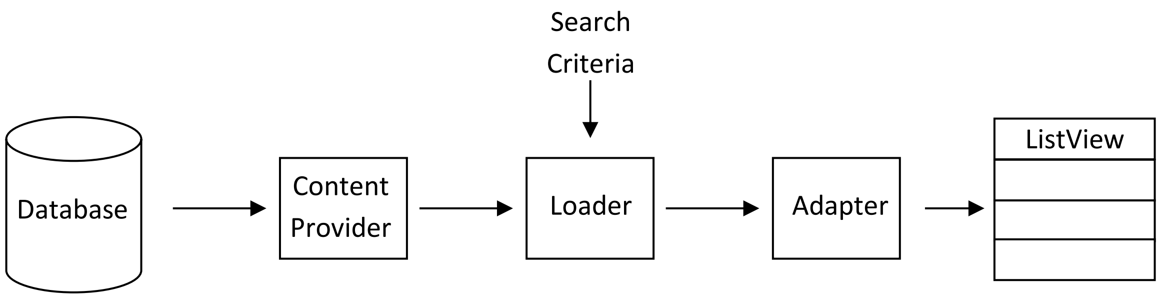


Figure 4.2 – Overview of Search Process

4.2.3.1 Content Provider

A Content Provider manages access to the database. Content Providers have a *content URI* that identifies the provider. Content Providers are recommended for providing access to a database to external applications; however they can also be used to supply access internally by querying the provider's *URI*. It is recommended by Android that a *CursorLoader* is used in conjunction with a Content Provider[17], this recommendation has been implemented. One benefit of using a Content Provider is that if in the future it is desired for GoClub to provide its database of adverts for other applications to use, removing the *android:exported="false"* statement from the provider declaration in the Android Manifest will mean that the provider is available to other applications.

In the next section, *Adapters* will be examined before we look at connecting up the *Adapter* with data from the *Content Provider.*

4.2.3.2 Adapter

An *Adapter* can be used to format any data provided to it into an Android *ListView*. In this case, the data is provided in the form of a *Cursor,* so a *CursorAdapter* is implemented. A *Cursor* is Android's standard interface used to provide access to the results returned by a query to the database. [18] Each result row of the *Cursor* is formatted into one item/row in the *ListView*.

The application uses a Custom *CursorAdapter* (named *AdvertCursorAdapter*)*,* which extends Android's *SimpleCursorAdapter.* The *SimpleCursorAdapter* class is an adapter to map columns from a cursor to TextView's defined in an XML file. This class is extended so the *BindView* method can be overridden. *BindView* specifies what information to display in the *ListView* rows.

Once data is supplied to the *AdvertCursorAdapter,* the overridden *BindView* method can be executed for each row to map the data to the elements. The method takes the title column (*DbHelper.A_TITLE)* value from the data and binds this directly to the title view. It queries the database with the *DbHelper.A_CLUBID* column value to get the club name. It also sets one element of the XML to display the distance to the club. In all cases, the method ensures data of different lengths are appropriately displayed without causing problems.

4.2.3.3 Loaders

The application uses a *Loader* to load data asynchronously. The *Loader* retrieves data from the database *Content Provider* and, once loaded, gives this data to the *Adapter* for formatting. Using a *Loader* means the user interface does not block while the data is loaded (database accesses may take some time, so may result in the UI blocking if not done through a *Loader*).

As the data is being retrieved from the database, the data is stored into a *Cursor.* It is the *Cursor* that, once loaded, is provided to the *Adapter*. Callback methods need to be implemented. These are supplied by the *LoaderManager.LoaderCallbacks<Cursor>* interface and specify what should happen at particular points in the loader's lifecycle. It is good practice[19] that the activity that uses a *Loader* implements the callback methods, which is the case in this application.

The *initLoader* method is called and the *LoaderManager* calls the method *onCreateLoader(int, Bundle)*. This method returns the actual *Loader* that starts loading the data. Any search criteria is placed into the *CursorLoader*'s database query that will generate the *Cursor* so only the matching results are loaded into the *Cursor*. In preparation for the final CursorLoader instantiation, the *onCreateLoader* method completes a number of queries depending on the search criteria specified:

- A query on the *VALIDON* table (the table that stores the day of the week that each advert is valid on) returns all the Advert IDs valid on the day the user has specified in the search criteria. The Advert IDs are stored into an array and used in the final *CursorLoader*.
  So that only these Advert IDs are displayed, the *CursorLoader* has:
  ..AND ID + " IN ("+ makeQuestionMarks(size) + ")… in its *selection* SQL statement.
  This is a parameterised query, where the question marks are replaced by data from an array provided as another parameter. The *MakeQuestionMark* method generates the number of question marks to display, that are then filled with the appropriate ID values.
- For all the adverts that match the search criteria the method loops through calling *getDistanceToClub(Integer).* This method calculates the distance (see *section 4.2.4 - Calculating distance* for how this is done) from the user's location to the advert's location. This is then stored in the *A_DISTANCE* column of the advert table.
  Results are then sorted on this A_DISTANCE value.

The *CursorLoader* is instantiated and then returned. It then begins loading the required data. Once finished, the appropriate adapter is updated with the Cursor data.

The search results have now been loaded and display in the *ListView*. The user can click on any one of the list items and open a new activity.

4.2.4 Calculating distance

The application uses distances in a number of situations, such as: 1) Sorting advert listings (so that nearer clubs are shown before further clubs) 2) Calculating whether a user has reached a club.

Below discusses two methods considered when calculating distance:

1. Using the Haversine formula [20] to calculate an 'as-the-crow-flies' distance between two longitude/latitude points. This takes account of the curvature of the earth, so achieves a slightly more accurate distance calculation than simply calculating an 'as-the-crow-flies' distance.
2. Connecting to the Google Directions API (Discussed later in *4.2.5 – Mapping and Routes*), providing it the longitude and latitude of the start and end point and parsing the XML/JSON results to get the distance returned of a possible route.

Method 2 generates a much more accurate distance measure. However, this method requires a lot more computation. In both methods, the application must retrieve the location of the club from the database and obtain the user's location. In method 1, the application has to perform simple arithmetic to calculate a distance. In method 2, the application has to connect to the Google Directions API, and read and parse the results. Method 2 requires an internet connection meaning adverts cannot be sorted (and hence accessed) without an internet connection. Furthermore, multiple calls would have to be made to the webserver and results are likely to be delayed.

Method 1 was chosen as distance can be calculated quickly and distance can also be calculated without an internet connection. If the user goes onto redeem an offer, they will be displayed an exact route, so will get a better idea of the distance to the club.

4.2.5 Mapping and routes

The RedeemActivity provides the user with a map and route from the user's location to the selected club. The route is generated by the process detailed below.

When the RedeemActivity is started, the *onCreate()* method (the method that is run when the activity is started) calls the *onLocationChanged()* method using the device's last known location as input. The *onResume()* method starts a LocationManager, which requests location updates every 30 seconds and 50 metres. When locations are returned by the LocationManager; this also invokes the same *onLocationChanged()* method as above. This follows the design pattern specified by Android whereby, a less accurate location (the last known location) is used initially and this is updated as more accurate locations are available (when the location manager returns the current location). [21]

An *onPause()* method is implemented to cancel the location updates when the activity is paused (when the activity is not on screen). *onResume()* restarts the location retrieval whenever the activity comes back into the foreground. This saves wasted processing (and battery) getting the location when it is not needed.

The *onLocationChanged()* method moves the map to focus the centre on the user's location. The method also starts an *AsyncTask* called *DirectionGetter*, which generates the route to follow.

The *DirectionGetter* builds a web address to connect to. There were two options at this point:

1. Connect directly to the Google Directions API.
2. Connect first to the back-end server and have that connect to the Google Directions API, returning any results to the device through the back-end server.

Option 1 has a direct connection to and from the Google Directions API, meaning the entire result is downloaded and parsed on the device. Option 2 attempts to remove work from the device, which as Android applications can be run on a wide variety of phones, with varying capabilities, is desirable. Option 2 allows the backend to filter out only required information. The disadvantage of option 2 is that two separate servers operating successfully are relied upon; the backend server and the Google Directions API server.

To show the difference in the file size returned to the device, given an example route to calculate, option 1 returns a result which is 19925 characters, whereas option 2 returns 1450 characters.

Option 2 was chosen and the application connects to the *generateroutepoly.php* file on the webserver. This formulates the appropriate Google Directions API request. *cURL* [22] is then used to request and retrieve the response from the Google Directions API. *cURL* is a library that lets you make HTTP requests in PHP. The Google Directions API returns a result, with a recommended route.

The returned JSON includes data that is not required for the purpose of this application. *Generateroutepoly.php* strips out the polyline values. Polylines store a set of lines that join up (at turning points) to form a route. The turning points have their longitude and latitude given. [23] An array of polyline values is returned to the application.

Returning to the *DirectionGetter* method that has now received an array of polylines as a response. If the route returned is not empty, the *drawPath* method is called, taking as one of its parameters the array of polylines. This method then loops each polyline and completes two steps:

1. *decodePoly* is called. This method returns a list of GeoPoints that each polyline encodes. [24]
2. *drawPathWithPoly* is called. This iterates through the list of Geopoints, instantiating the RouteOverlay class. The RouteOverlay class is introduced in a tutorial at synyx.de. [25] RouteOverlays draw a line between two points, which is then added to the map overlay. The different RouteOverlays connect to make a route.

Finally, after *drawPath* returns, the map is re-drawn displaying an overlay route. The route is recalculated in the background every time the device returns a new location and is drawn on the screen once ready.

4.2.6 Facebook

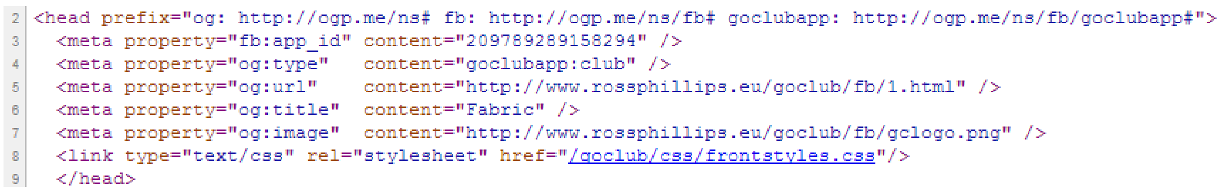
The application gives the user the ability to post to their Facebook news feed to inform their friends that they are going to a certain club.

All entries to a user's new feed or timeline are stories generated through the Facebook Open Graph. The Facebook Open Graph allows users to share stories about activities they are doing. The Open Graph allows a user to act out an *action* on an *object*. For example: *"Ross shared a link"*, the *action* is to 'share' and the *object* is the link. The Facebook Graph API allows developers to read and write data to Facebook on behalf of a certain user.

Below details how the GoClub application uses the Open Graph to make posts to user's Facebook pages. As stated above, a post is made up of an Action and an Object; Objects will be covered first.

4.2.6.1 Open Graph Objects

Objects are items that can be acted on. A new object "Club" was created to represent the different clubs that a user can interact with. When a post is made about a certain Object, Facebook needs to retrieve information on this object. To allow this, each instance of the Club Object (All the different clubs - Fabric, Ministry of Sound etc) has a webpage (hosted on the GoClub webserver) that includes information about the club. Each Club on the application requires a separate webpage to hold its respective information. A segment of the page for the club Fabric is shown in Fig. 4.3.



```
<head prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb# goclubapp: http://ogp.me/ns/fb/goclubapp#">
  <meta property="fb:app_id"  content="209789289158294" />
  <meta property="og:type"   content="goclubapp:club" />
  <meta property="og:url"    content="http://www.rossphillips.eu/goclub/fb/1.html" />
  <meta property="og:title"  content="Fabric" />
  <meta property="og:image"  content="http://www.rossphillips.eu/goclub/fb/gclogo.png" />
  <link type="text/css" rel="stylesheet" href="/goclub/css/frontstyles.css"/>
  </head>
```

Figure 4.3 – Fabric Object Page

Generation of object Pages

The generation of object pages is done automatically as manual creation of each page would be time consuming. When a clubs registration is approved by the admin on the website, a PHP script runs to generate the appropriate page for the new club.

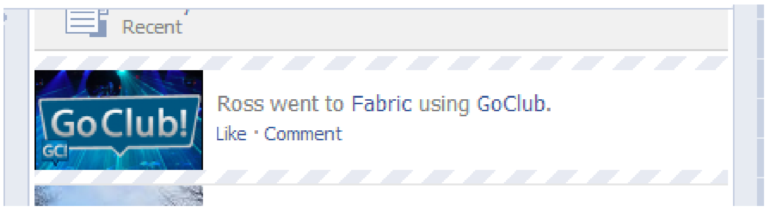
A default .html file contains the general structure that all object pages share. Where a value differs depending on the club the page represents, the default page has a variable marker enclosed in parenthesis, for example, “{clubname}”. Inside the *approve.php* script, the contents of the default file is retrieved, and all default values are replaced with real values. The page is then stored, with the name *[clubid].html*. Object pages contain the meta-tags and also a basic landing page for the club.

Fabric’s object page: http://rossphillips.eu/goclub/fb/1.html

4.2.6.2 Open Graph Actions

Actions define how a user can interact with GoClub’s *club* object. A custom ‘GoTo’ action has been made for the GoClub application. This is so that a user can “[Action: GoTo] a [Object: Club]”, for example, “Ross is going to Fabric” where “going to” is the action (in present tense) and “Fabric” is an object of the type Club. The action has been configured, so the sentence is correct whatever tense it is presented in. For example, once the action has expired (after a set time), the action will become past tense (“Ross went to Fabric”).

Once an Action has been set up with the associated object, it can be called by a request to the graph API. This is done by calling: *me/goclubapp:go_to* (a unique path for the action) with a *club* object specified as a POST parameter. The result of a successful post can be seen in Fig. 4.4.



Ross went to Fabric using GoClub.
Like · Comment

Figure 4.4 – A successful post to Facebook.

How the GoClub application connects to the Graph API, through the Facebook SDK, to make such posts will be covered in the next section.

4.2.6.3 Application Code

The Facebook SDK provides a *Session* class. This class is used to authenticate users and manage their session with Facebook. A callback interface is also used to handle changes in session state. Before the request to the Graph API can be made, a *Session* must be successfully opened for the user. Once opened, *handleAnnounce()* is called to handle the post to Facebook's Open Graph.

handleAnnounce() checks whether a valid session exists and then that the required Facebook permissions have been granted to the GoClub application. A *publish_actions* permission is required. To obtain this the *requestPublishPermissions* method is called.

- The *requestPublishPermissions* method calls the SDK method *requestNewPublishPermission.* This verifies with the user that they wish to grant the permission.
- When the original activity is returned to, the session state has changed as the new permission has been granted; as a result, the callback method *onSessionStateChange* is called. This method checks whether an announcement is in process and if so, *handleAnnounce()* method is called again.

Assuming the *handleAnnounce()* method now passes the permission check, the method starts building the Announcement. This is done in an *Asynctask* as the process of sending a request and retrieving a response may be time-consuming.

Two objects are created: One object to represent the *Graph Action* that the user wishes to undertake and another to represent the *Club object* that the action will act on.

A *GoToAction* object is created and then the *populateOGAction* method is called. This method populates the action with a *Graph object* to act upon. In this method, a *ClubGraphObject* has its web address variable set, so Facebook Open Graph knows where to find details on the particular object. The *ClubGraphObject* is then assigned to the action. An expiry time is set on the action; so that the action can change from present to past tense when displayed on a user's facebook feed. The *GoToAction* object has now been populated with all the required information.

A request object is instantiated. A Request is a single request to the Facebook Graph API from an application. The previously created *GoToAction* which holds the ClubGraphObject representing the chosen club is added to the Request and the Request is then executed and a response is waited for. This is shown in the code below:

```
request.setGraphObject(goToAction);
request.executeAndWait();
```

Once a Response is received, it is handled by the *onPostActionResponse,* which, in the case of a successful post, finishes the interactions with Facebook's SDK, or in the case of an error, calls the *handleError()* method.

4.2.7 Reporting at a club

One of the challenges identified in the introduction was that clubs are unlikely to declare application users reached the club, to avoid commission (if payment is taken per-customer who reaches the club in the future). To overcome this, there has to be a mechanism for knowing whether a user reached a club or not. Furthermore, it is beneficial for the clubs that are placing adverts to see how many users reach the club from each advert that they place.

Two possible methods to check whether the user is at the club were considered:

1. When Redeem is clicked to show directions to the club, the application starts up an asynchronous service. This service will, every 5 minutes, check the user's location to see if it is within a certain distance of a club. This service will continue running until the user is calculated as being at the club or until a certain time period has expired.
2. The redeem/map page could have a 'claim now' button that the user is instructed to press once they are at the club. This creates a pop-up detailing the terms of the deal they have selected for the cashier to view and also runs a method to check their location.

Method 1 relies on the creation of services that run every 5 minutes; this will drain the battery. Furthermore, users may click 'redeem' on multiple offers to see the route to the club before deciding which one to go to; multiple asynchronous services could be created to handle location getting and checking. This would be very inefficient.

It was decided that method 2 was to be implemented. Distance is calculated in the method discussed in section 4.2.4. If the user is within 0.2km of the actual clubs location, the application will report to the server that the user has reached the club.

4.2.8 Notifications

The GoClub system gives clubs the opportunity to distribute special deals to GoClub application users in the area nearby to their club. These special deals appear as a notification on the user's phone. The application does not have to be open for notifications to be received. Clicking on a notification will open the application and show the details of the particular advert. GoClub notifications use the Google Cloud Messaging (GCM) service for Android. This *"is a service that allows you to send data from your server to your users' Android-powered device".* [26]

There are a number of steps involved in a user successfully receiving a notification sent by a club from the website. These are summarised in the diagram in Fig. 4.5. Steps **1)** and **2)** are part of the registration and only occur when the application is first installed. Steps **A)** and **B)** are the device providing the server with its location and this happens every 15 minutes. Steps **C)** and **D)** are a notification being sent to the phone, this only happens when the server deems there is a notification local to the user waiting to be sent. Each step will be looked at in further detail below.

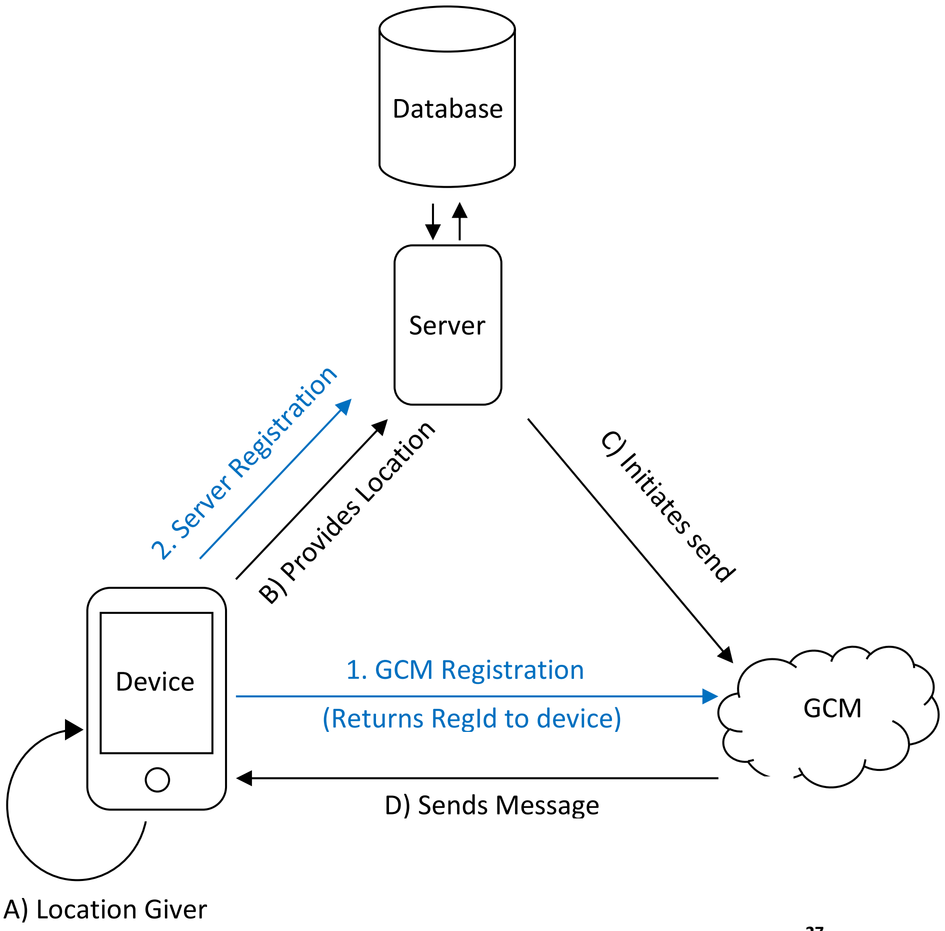


Figure 4.5 – GCM Overview (Adapted from similar AndroidHive [27] diagram)

Registering the device:

1. GCM Registration

When the application is first downloaded, the user has to complete a registration process in order to receive notifications. To register, the user fills out a form with their details. It is checked whether the device has a GCM Reg ID; if it does not, it is registered and a Reg ID is returned. The unique Reg ID will allow notifications to be sent to just that device. We need to save the Reg ID to the GoClub backend server, so that it is accessible when required, this is done in the 'Server Registration' step.

2. Server Registration

For unregistered devices, registration with the backend server is initiated. The server stores the device's GCM Reg ID (along with user-supplied details) into the *gcm_users* table of the database.

Receiving Notifications:

A) Location Giver

If the user does allow notifications (note: users can opt out – this is stored as a SharedPreference, so remains between sessions of use of the application), the *LocationGiver* service is started. This service retrieves the user's location and reports it to the server. It then calls the *scheduleNextUpdate()* method to schedule when it is next due to run. *scheduleNextUpdate()* uses Android's *AlarmManager* class. This class provides access to the functionality of a device's alarm service, and is used to Schedule the *LocationGiver* method to run every 15 minutes between 6pm and 2am. *AlarmManager* is used over other available timing operations, such as timeouts, as the *AlarmManager* will continue to run even when the application is closed and the device is locked. Furthermore, timeouts/waits mean there is a continuous process running for the duration of the timeout, whereas, using *AlarmManager,* between the Scheduled runs of the service there is no additional process running, which is a lot more efficient and saves battery power.

B) Provides Location

The *LocationGiver* service provides the device's location to the *setlocation.php* script, which is part of the backend server. This script updates the *location* table with the user's location. The *CheckDistanceOfAdverts* method is then called; this method retrieves all the notifications that clubs want to be 'pushed' as notifications on the night in question. If the user is within 1km of the advert's location and has not already received the notification, the notifications ID is added to an array.

C) Initiates Send

Continuing in the *setlocation.php script,* if there is more than one advert that could be sent to the user, one advert is currently chosen at random to send. In future, when a payment system is in place (see section 6.3 - future developments), the priority of which advert to send could be decided by which club is paying the most per-notification-sent.

A new row is added to the *received* database table, so that the same advert cannot be received again on that night. Automated PHP Cron jobs wipe the *notification* and *received* table after each night.

The *send_notification* method uses *cURL* to communicate with the Google Servers and provides the required *Reg ID* (that identifies the user) and *message* to send.

D) Sends Message

When the GCM server sends a message to the device, it is received by the Android *GCMBroadcastReceiver* class*,* this is an extension of the *BroadcastReceiver* class. A *BroadcastReceiver* registers for updates to a certain system event, in this case, a GCM message reaching the device. The *GCMBroadcastReceiver* is notified when this happens and delivers the message to the *GCMIntentService* class.

On receiving a message to the *GCMIntentService* class, the *onMessage()* method is called. This gets the data out of the Intent parameter; this data holds the contents of the notification to display. The *sendNotification* method is then called with this data.

The *sendNotification* method uses the Android *NotificationManager* class to produce a notification. The *NotificationManager* class is used to tell users of events that have happened. The details of the advert are placed into a notification (of class *Notification*). When the notification is clicked, an intent will cause the application to open displaying a certain advert. The notification is set so it will cause the phone to play the default notification sound and vibrate to alert the user of the incoming notification.

4.2.9 Augmented Reality

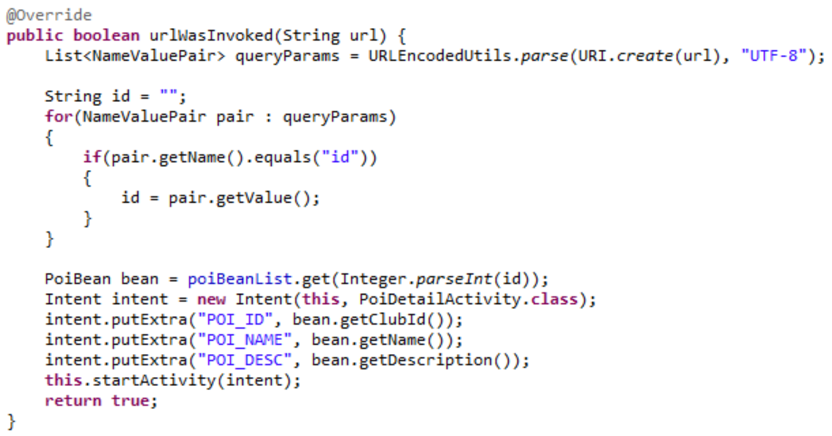
In the GoClub application, Augmented Reality allows the user to hold up their camera, displaying what it is recording, and overlaying markers over clubs. The positions of clubs (known as 'points of interest') are calculated relative to the user by providing the longitude/latitude of both the user and the point of interest. [28] These points of interest can be clicked to retrieve more information. The application uses the Wikitude SDK that was introduced in *Section 2.3.1.* The SDK supplies two classes that allow access to functionality of the Wikitude API.

The first SDK class is *ArchitectView,* which allows for the setting up and loading of data into the *ARchitect World*. The *ARchitect World* is the name for the augmented reality view provided by the *ARchitect API* technology. [29]

To set up the ARchitect World, an HTML file is loaded into it. This includes an internal CSS style sheet. This style sheet is used to alter the appearance of the *ARchitect World.* For example, the text which holds the club's name can be set to a chosen colour through this CSS. JavaScript statements give instructions on what to do on certain events, for example, when an item is clicked. Data is retrieved from the database and stored in an array. It was then converted to JSON and callJavascript() is then called to pass points of interest to the *ARchitect World*.

We also register a URL listener with the *ArchitectView*. This URL listener interface, *ArchitectUrlListener*, provides one method; this allows the *ARchitect World* to communicate an event back to the application. When events occur, the Architect World will call a URL with the *architectsdk://* protocol. The *ArchitectUrlListener* receives the requested URLs and can parse the name value pairs that are stored within it. For example, when a user clicks for more information on a club, the URL "*architectsdk://opendetailpage?id=*" is invoked and then parsed by the URL listener.

Fig. 4.6 shows the overridden *urlWasInvoked* method. This takes the invoked URL as a parameter, which is then parsed into a list of name-value pairs. An ID is found and from this more information found from the array that was passed as input to the *ARchitect World*; this data is then placed into an Intent and a new activity is started using the Intent data.



```java
@Override
public boolean urlWasInvoked(String url) {
    List<NameValuePair> queryParams = URLEncodedUtils.parse(URI.create(url), "UTF-8");

    String id = "";
    for(NameValuePair pair : queryParams)
    {
        if(pair.getName().equals("id"))
        {
            id = pair.getValue();
        }
    }

    PoiBean bean = poiBeanList.get(Integer.parseInt(id));
    Intent intent = new Intent(this, PoiDetailActivity.class);
    intent.putExtra("POI_ID", bean.getClubId());
    intent.putExtra("POI_NAME", bean.getName());
    intent.putExtra("POI_DESC", bean.getDescription());
    this.startActivity(intent);
    return true;
}
```

Figure 4.6 – ArchitectUrlListener method

4.2.10 User Interface Design

4.2.10.1 Launcher Icon

The launcher icon is displayed on a devices application screen. The GoClub icon follows a number of guidelines, specified by android [30], these include: 1) The use of a distinct silhouette 2) Icons should appear 3D 3) The view of the icon should be from the front; with a slight perspective as if the icon is viewed from above.

Figure 4.7 – GoClub Launcher icon

4.2.10.2 Custom Buttons

To increase the applications individuality and improve the consistency of the user interface, custom buttons were created. To ensure the background images that you provide can handle being stretched in any direction, the background image must be a nine-patch image (file extension: .9.png). These files specify a 1 pixel wide black border around areas which are stretchable. 9-patch images can be produced in the tool supplied with the android SDK called draw9patch (shown in Fig. 4.8).

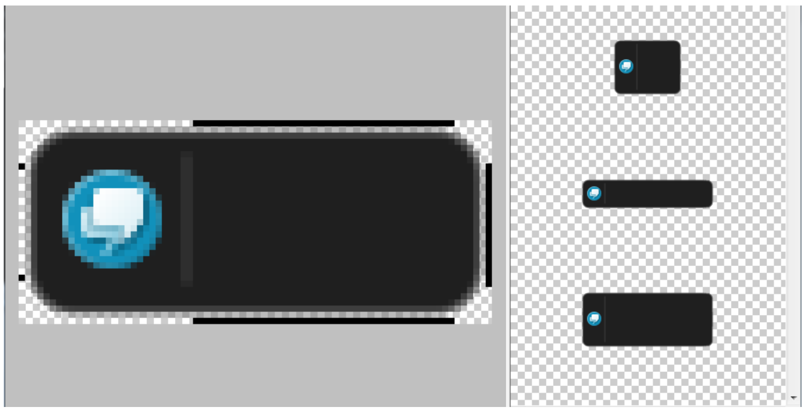


Figure 4.8 – Ratings/Reviews button

Fig. 4.8 shows the ratings/reviews button used on the AdvertView activity. The positioning of the black border allows the button to stretch, however the 'Comment icon' always remains the same so the icon does not distort or become non-circular.

4.2.10.3 Custom Rating

A custom rating bar was designed because the standard Android rating bar did not fit with the rest of the UI design. A segment of the UI, including the custom rating bar is shown in Fig. 4.9.

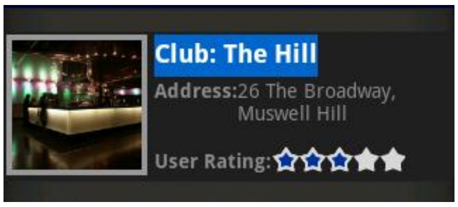


Figure 4.9 – Custom GoClub Rating Bar

To customise the rating bar, a new style had to be defined. Styles are a collection of XML properties that specify how a certain user interface element should display. [31] In the GoClub application, the custom style, named *starRatingBar,* inherits from the standard Android rating bar style. This is done by specifying the parent attribute, as seen in Fig. 4.10.

```
<style name="starRatingBar" parent="@android:style/Widget.RatingBar">
    <item name = "android:progressDrawable">@drawable/star_ratingbar_full</item>
    <item name = "android:minHeight">18dip</item>
    <item name = "android:maxHeight">18dip</item>
</style>
```

Fig 4.10 – Custom StarRatingBar style

Any conflicting properties declared in the custom style will overwrite properties that have been inherited.

In Fig. 4.10, it can be seen that *ProgressDrawable* has been declared. This provides reference to a further .XML file, called *star_ratingbar_full*. *Star_ratingbar_full* defines how the rating bar should be drawn. A rating bar is made up of a background, and the stars that are drawn. This file references two further files detailing how each of these (the background and stars) are drawn. The files define the different states (pressed, focused and selected) each can be in and depending on the state what image file to draw. Finally, the style is referenced from the *RatingBar* to be customised.

A benefit of storing styles in the file that is external from all layout files, is that custom styles can easily be used throughout the application, without duplication of code, by adding the *"style ="* property to the element. Furthermore, if changes are required, code only has to be changed once instead of in multiple places in the application, ensuring consistency. The custom style was used on a number of different rating bars, throughout the application.

4.3 Website Components

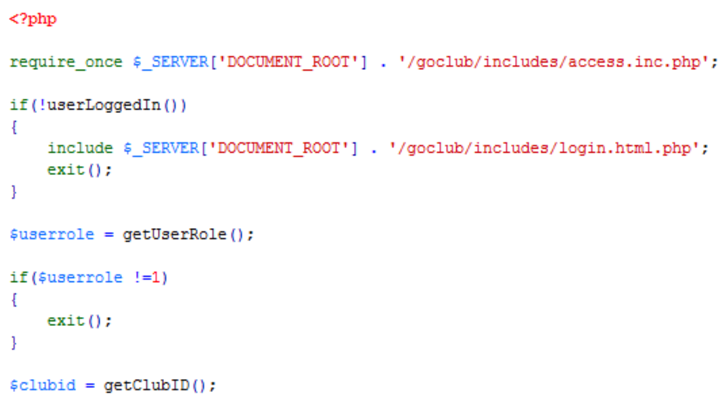
The important components that make up the website are detailed below.

4.3.1 Structure

The system is developed so that the view of a page (what is displayed) is separated from the main logic. This is done by containing the view in *.html.php* files and the logic in *.php.* 'Include' is used to link to the required *.html.php* file at the appropriate point to display what is required. This means there is never any HTML in the logic files of the website. This is done as it simplifies the code for the reader and also makes maintenance easier.

4.3.2 Login System

The website uses a login/registration system so that users can log in and access their own private pages. Each user has a row in the user table storing details about their account. The access management functions are held by an *access.inc.php* file, which gets included on all pages that require a user to be logged in, its methods are called in the page's code.



```php
<?php
require_once $_SERVER['DOCUMENT_ROOT'] . '/goclub/includes/access.inc.php';

if(!userLoggedIn())
{
    include $_SERVER['DOCUMENT_ROOT'] . '/goclub/includes/login.html.php';
    exit();
}

$userrole = getUserRole();

if($userrole !=1)
{
    exit();
}

$clubid = getClubID();
```

Figure 4.11 – Access Management

Fig. 4.11 shows the access checks that are executed on every page requiring a user to be logged in. The main methods are described below.

userLoggedIn() - This method handles the authentication of users. It can be broken down into a number of different segments. Firstly it can be used to log in users. The function checks the user table of the database and counts the number of rows that have the typed email and password, if this returns larger than 0, the user has got the credentials of an account correct, and session variables are set so that the user does not have to authenticate themselves on every page they visit. Another segment checks whether the user wishes to logout; all session variables are then unset. Finally the credentials of a user that has the *$_SESSION['loggedIn']* variable set are checked. This segment of code runs once a user has been logged in and has navigated to a new page.

getUserRole() – This takes $_SESSION['email'] variable and returns the user's role so it can be checked the user is of the right type to be on a certain page.

getClubID() – As all users access the same pages, this function gets the club ID a user is associated with, so only data on that club is displayed.

Passwords are stored in the user table using the PHP MD5 function. MD5 is a function for one way hashing. Using this, passwords are stored as an encrypted value. This is done for additional security so that those with access to the database cannot view user's passwords. MD5 hashes can be reversed and websites exist that will do this[32]. To prevent this, random data can be appended onto passwords before they are hashed; this is called 'salt'. Adding 'salt' is a common technique in cryptography [33] to improve the complexity of an encryption. When a user registers, the 'salt' is added before the result is hashed and stored to the database. When the user attempts to log in, the same 'salt' is added before the result is again hashed and compared with the values in the database to check that it matches.

4.3.3 Email Activation

On registration, the system will automatically generate and send a unique link to the email address supplied. Inside the email is an automatically generated link with the user's email address and a random 32-bit number, which has also been stored to the database. Below shows an example link:

http://www.rossphillips.eu/goclub/verify.php?email=ross.phillips@live.com&hash=db8e1af0cb3aca1ae2d0018624204529

Clicking the link will run the *verify.php* page, that will check whether a row exists in the user table with this email and random number. If this is the case, the email has been activated and the value in the active column is updated from 0 to 1. On log in, it is checked whether a user has activated their email and access is only granted if this is the case. [34]

4.3.4 Image CAPTCHA

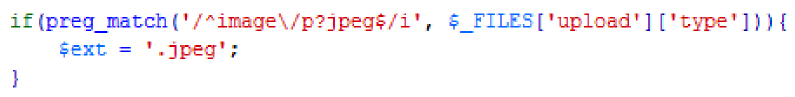
The website uses an Image CAPTCHA system to verify users interacting with certain parts of the website are humans[35]. The user has to type the numbers that are present.

The system works by generating a random number and storing it into a PHP session variable. PHP session variables allow information to be stored between HTTP requests. The value is taken from the session variable and stored into an image. Random lines are drawn over the image to increase the complexity of deciphering the text.

PHP's *Imagejpeg* function is called which outputs the resulting image to the browser. When the user submits the form they have to type the values that they see. The system then checks that the session variable matches the appropriate form field value. Every time the page is reloaded a new random number is generated, changing the value that must be typed.

4.3.5 Image Uploading

Users are able to upload a profile picture of their club to display on the application. Images can be uploaded by a simple HTML form. On the forms submission, details of the uploaded file get stored into a $_FILES variable. The upload is checked to confirm the file uploaded is an image file (in this case, only .jpeg files are allowed), shown in Fig 4.12.



```
if(preg_match('/^image\/p?jpeg$/i', $_FILES['upload']['type'])){
    $ext = '.jpeg';
}
```

Figure 4.12 – File Type Validation

Checks are done to ensure the file does not exceed 2MB. A filename is generated by appending the user's IP address and the time of upload. The filename is not taken from the filename of the uploaded file as Android applications have difficultly displaying files with spaces in their filename (it is a possibility user uploaded files have spaces in their name).

The image is then cropped to ensure it is square. This is to ensure all pictures are the same shape so they do not cause problems on the application. The file is then saved to a location in the file store. If the user already has a profile picture, this picture will be overwritten. If required, the database is updated, with a file location being stored in the *imageloc* column of the club table.

4.3.6 Graphs

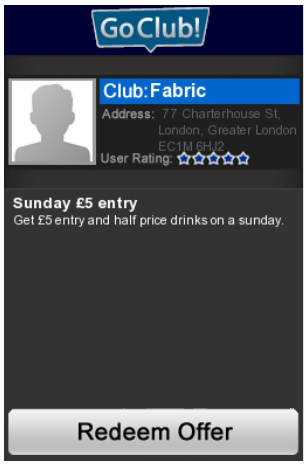
Statistics showing the success of adverts are visually represented as data on graphs. These graphs use the JpGraph library [36]; This is a PHP library using object oriented programming to create graphs. Graphs can be generated with data from the database and also customised. A graph is made by executing: *$graph = new Graph(X,Y)*. Then further parameters are set by using the $graph variable. For example, months on the axis are set by:

$graph->xaxis->SetTickLabels(array('Jan','Feb','Mar'…

Data retrieved from the database, which is stored in the variable *$datay1*, is converted to a set of bars using *BarPlot($datay1)*, which is then added to the graph. Finally, the graph is output on screen.

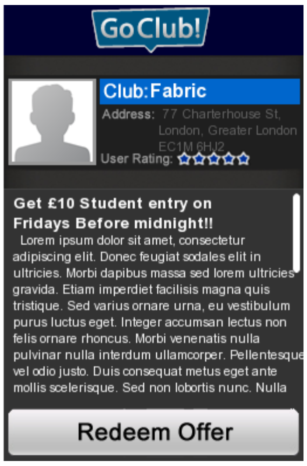
4.3.7 Advert Preview

Previews of adverts provide users an indication of what their advert looks like. Previews are generated using a static image as the background. The background differs depending on the length of the text in the description of the advert, as shown below in Fig. 4.13.1 and Fig. 4.13.2. If the description does not fit onto one screen, the text is cut and a scroll bar is present to show the user could scroll in the application. Text is wrapped, using the *wordwrap* function, so that it does not run off the side of the screen. Note how in Fig. 4.13.2, the title has wrapped meaning it takes up two rows and not one, the algorithm moves the description down a further row so that the description does not get positioned on the same line as the title.



Club:Fabric
Address: 77 Charterhouse St, London, Greater London EC1M 6HJ2
User Rating:

Sunday £5 entry
Get £5 entry and half price drinks on a sunday.



Club:Fabric
Address: 77 Charterhouse St, London, Greater London EC1M 6HJ2
User Rating:

Get £10 Student entry on Fridays Before midnight!!
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec feugiat sodales elit in ultricies. Morbi dapibus massa sed lorem ultricies gravida. Etiam imperdiet facilisis magna quis tristique. Sed varius ornare urna, eu vestibulum purus luctus eget. Integer accumsan lectus non felis ornare rhoncus. Morbi venenatis nulla pulvinar nulla interdum ullamcorper. Pellentesque vel odio justo. Duis consequat metus eget ante mollis scelerisque. Sed non lobortis nunc. Nulla

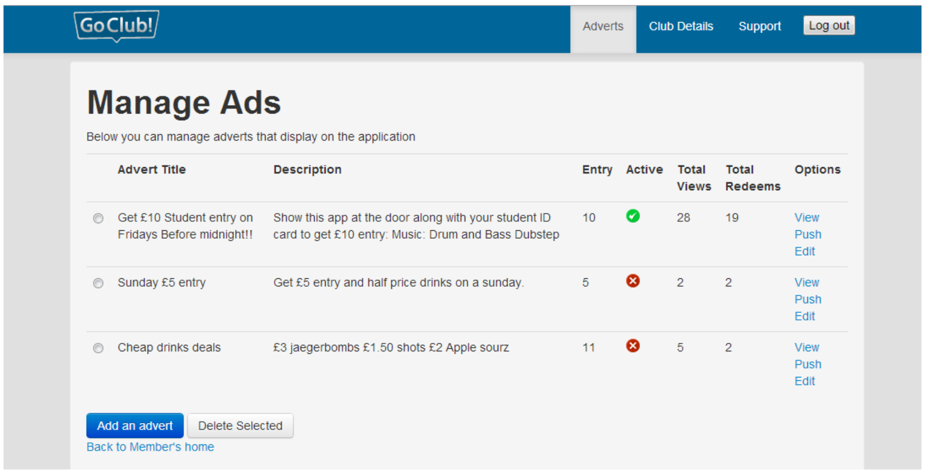
Figure 4.13.1 Advert without scroll bar - Figure 4.13.2 - Advert with scroll bar

4.3.8 Advert Management

An advert dashboard (seen in Fig. 4.14) allows clubs to manage adverts. From here, users can click to add a new advert; filling out the resulting form will add a new row in the advert table of the database. This is done using the following SQL query.

```
$sql = "INSERT INTO advert SET title = '$title', entry = '$entry', description =
'$description', previewloc= '$filename', clubid = '$clubid'";
```

Users can also edit and view individual adverts (for more user interactions see *Appendix G).*



Manage Ads

Below you can manage adverts that display on the application

	Advert Title	Description	Entry	Active	Total Views	Total Redeems	Options
○	Get £10 Student entry on Fridays Before midnight!!	Show this app at the door along with your student ID card to get £10 entry: Music: Drum and Bass Dubstep	10	✓	28	19	View Push Edit
○	Sunday £5 entry	Get £5 entry and half price drinks on a sunday.	5	✗	2	2	View Push Edit
○	Cheap drinks deals	£3 jaegerbombs £1.50 shots £2 Apple sourz	11	✗	5	2	View Push Edit

Add an advert Delete Selected

Back to Member's home

Figure 4.14 – Advert Management Table

4.3.9 Admin Account

A user with an admin account can log in and access different functionality to a club account. A mechanism is in place in the database where each user has a role attribute. This is an integer, which is set as 1 for a club and 2 for an admin. On submission of the login form, this is checked and admins are redirected to their area of the website. On all account pages, the user's role is checked to verify they are of the correct type to access the certain page. Admins are able to approve club registrations. Clubs have to be approved before they can post adverts. This stops anyone signing up and providing inappropriate or unwanted content to the users of the application. Admins can also moderate flagged reviews, removing them if they believe they are inappropriate.

4.3.10 Website Database

4.3.10.1 Structure

Website Dependency Analysis

- All tables are in **First Normal Form (1NF)** as the intersection between rows and columns contain one and only one value.
- All the tables have single-attribute primary keys, except the received table. These tables are automatically in **Second normal form (2NF).** The received table is also in 2NF as there are no partial dependencies.
- All tables, except GCM_user, are in **third normal form (3NF).** See the discussion below regarding the GCM_user table.

User table

ID	Email	Password	ClubID{FK}	Role	Approved	Hash	Active

Club table

ClubID	Name	Description	Location (address)	Latitude	Longitude	Imageloc

Rating table

ClubID {FK}	RatingValue	RatingAmount	RatingAverage

- Rating attributes could be placed in the club table but is separated for clarity and to keep tables of a manageable size.

Advert table

AdvertID	clubID {FK}	Title	Description	Entry	Active	PreviewLoc

statistic table

AdvertID {FK}	[Jan-Dec]Views	[Jan-Dec]Redeems

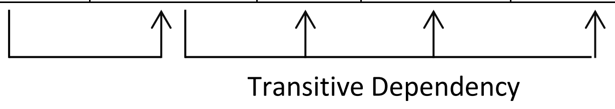
- Statistic attributes could be placed in the advert table but are separated for clarity and to keep tables of a manageable size.

Day table

ID	AdvertID{FK}	DayID

GCM_user table

GCMUserID	GCM_regID	Name	University	Created_at



- The GCM_user table is not in third normal form as there exists a transitive dependency. From the GCMUserID, you can look up the GCM_regID and from this value (as it is unique to each user), the name, university and created_at value. GCM_regID is a candidate key and if 3NF is desired in the future, converting GCM_regID to be the primary key and removing GCMUserID would make the table 3NF. GCM_regID was not chosen as the primary key (and the GCMUserID column introduced) because the GCM_regID is a very long character string supplied by the Google Cloud Servers. The ID has to be referred to throughout the system and for convenience shorter values are preferred. As each user can only have one row in the table, the table cannot be subject to update anomalies.

Review table

ID	ClubID {FK}	GCMUserID {FK}	Rating	Review	Date	Flag

Location table

GCMUserID {FK}	Lat	Longi	Flag	Timestamp

Recent use

ID	GCMUserID {FK}	AdvertID {FK}	Timestamp

notification table

ID	AdvertID {FK}	Timestamp

Received

UserID {FK}	**NotificationID** {FK}

4.3.10.2 Implementation

All interactions with the database where data is being written have been protected against SQL injection attacks[37]. This protection is implemented using PHP's *mysqli_real_escape_string()* variable on all user input values. This function escapes any characters with special meaning so that the raw un-executed characters are input into the database and not run as a SQL command.

4.3.11 User Interface design

The design of the website was aided by the twitter bootstrap template library, which was introduced and benefits given in *section 2.3.4*. Bootstrap is integrated into the website, by storing the provided .css file into the website's .css folder. This is then extended, by making a further .css and importing the bootstrap.css at the top. Further CSS declarations can be added to provide customization. The design follows the consistency design principle set out in human computer interaction guidelines for software. [38] All club pages have a blue navigation bar running across the top, with the GoClub logo (designed in Adobe Fireworks) in the top left. The internally consistent design means that users will soon become accustomed with the websites layout. The website is also externally consistent with many other websites that use the twitter bootstrap library as they follow a very similar design; meaning users will feel familiar with the design from the outset.

Chapter 5

Testing

5.1 Application

The different methods undertaken to test the application are detailed below.

5.1.1 Stress Testing – Monkey

As described in *section 2.3.5*, Monkey is a command line tool used to generate a pseudo random stream of events. The stream of events generated depends on the seed value given to the command line. As a result, in the case of Monkey, test cases are not specifically selected; however a number of measures were used to ensure good coverage of the stress testing.

The Monkey was run five times on each different activity (in one run the activities can change, however some activities may be tested less than others if the tests were all started on the same activity). As each run generated 500 random events, this meant that 2500 events were generated starting on each activity. Once an error was found, it would be fixed and the monkey would be ran again with the same seed to check whether the same error occurred, as the same events would occur that originally caused the exception.

As can be seen in *Appendix D.1.1 – Monkey Stress Testing Results*, a total of four exceptions were found. An interesting example of one found was the second exception found. This NullPointerException was caused by the user clicking 'Search' too quickly once the application had started up. If stress testing was not used, this may not have been discovered. It was not discovered previously as when manually tested, some time would be spent selecting the search criteria before clicking 'Search'; this gave the database time to load. Now that the exception has been resolved, when the 'Search' button is clicked the application will check first whether the database has been populated, if it has not, the user will be told to wait.

5.1.2 Functional testing

Functional test cases were devised at the end of the project to ensure that all requirements had been met. Each test case outlines a repeatable run of events that were completed. Each test case tests either one or multiple functional requirements of the system. Coverage of this testing was good, with 100% of Must Have, Should Have and Could Have requirements (all implemented requirements) being tested by at least one test case.

A summary of the test cases and results can be seen in *Appendix D.1.2 – Manual Application Functional Testing*. Each test case has an expected outcome column, which details what the result of completing a certain test case should be. If the outcome of completing the test case is different to the expected, the final column *'Outcome, if different to expected'* is filled in. On the first run of the functional testing, 4 out of the 42 test cases did not exactly match the expected outcome. The table shows those tests that had unexpected outcomes (the full document can be seen on the CD).

One interesting case was TA13. This would be caused by moving onto the *ReviewActivity* and then quickly moving away from it. An Exception would be thrown by an *ASyncTask* started in this activity once it had completed its work as it had no activity element to update with its result. To resolve this, the *.cancel()* method was added to the *ASyncTask*; that cancels the *ASyncTask* when the activity is paused (when the activity is no longer in the foreground). This means after the *ASyncTask* has finished its processing, the *onPostExecute()* method (which required the UI element) does not run, meaning the exception would not be thrown.

Another interesting test case was TA24, which resulted in a NullPointerException being found. On investigation of the cause of this, it was determined that this is a known problem [39] with the current version of the Facebook SDK. As a result, a line of code had to be added to the Facebook SDK for this to be resolved.

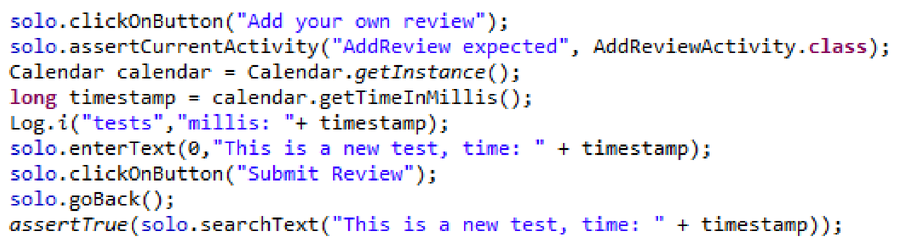
Test cases that resulted in unexpected outcomes were all fixed and run again. A 100% pass rate of all test cases was achieved after the required fixes.

5.1.3 Robotium

As stated in the introduction to Robotium in *section 2.3.7,* Robotium is a testing framework that drives user interface interactions with an application. It extends the functionality of the JUnit testing framework and uses the JUnit assert method to check whether certain conditions have been passed.

Robotium test cases were chosen to test the main user interactions with the application. User interaction is notably very restricted, which is a good thing. The main page allows for three dropdown boxes to be altered to change criteria, elsewhere in the application reviews can be added and a map interacted with, other than this, the application's do not take user's input and it is simply displaying information. Robotium struggles to test some functions which manual testing can do with ease. For example, it is very easy for someone to verify a correct route from location A to B has been mapped, however this is impossible using Robotium. As a result, Robotium was used in only the places that it was well suited. Test cases attempted to test places where user input was possible, a full list of the Robotium test cases completed can be seen in *Appendix D.1.3 – Robotium test cases.*

An interesting Robotium test case was testSubmitRatingReview; an excerpt of this can be seen below in Fig. 5.1. This test checked to see if reviews were successfully submitted and displayed among previously submitted reviews. It was realised that using static text in the review (the text says the same in each run) would possibly result in errors not being found. If a review with static text was submitted successfully (For example, "This is a review"), it would then be found and the test passed. However, as the review would remain (unless it is deleted), any future runs of the test will find the text "This is a review" (from the previous submissions) and pass the test, even if the submission of a new review was not successful. As a result, a method was developed so that each time the test ran a new review was submitted and checked for. This was done by submitting a review with the current timestamp and checking that that timestamp exists in the reviews.



```
solo.clickOnButton("Add your own review");
solo.assertCurrentActivity("AddReview expected", AddReviewActivity.class);
Calendar calendar = Calendar.getInstance();
long timestamp = calendar.getTimeInMillis();
Log.i("tests","millis: "+ timestamp);
solo.enterText(0,"This is a new test, time: " + timestamp);
solo.clickOnButton("Submit Review");
solo.goBack();
assertTrue(solo.searchText("This is a new test, time: " + timestamp));
```

Figure 5.1 - testSubmitRatingReview

The Robotium tests were developed and used throughout the project. They were re-run every time changes were made to ensure that no existing functionality had been broken. An example output from a run of the Robotium tests can be seen in the appendix D.1.3.

5.1.4 Device testing

The application has been tested on two different physical android devices:

Device: HTC Hero	**Device:** HTC Wildfire S
Operating System: 2.2	**Operating System:** 2.3.5

Using these two devices was useful as they have different screen sizes. As a result, the UI could be examined to see how it reacts to a change in the screen size. The main use cases were acted out on each device, and no problems were noted with the function or the user interface on either device.

Furthermore every activity in the application has been tested to see how it reacts to being interrupted by either a call or a text. The activity would be opened and then the situation simulated meaning the application would be left, after the call or text had been dealt with, the activity would be returned to. There were no problems found.

5.2 Website

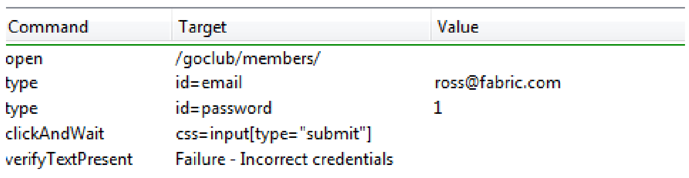
The different methods undertaken to test the website are detailed below.

5.2.1 Functional testing – Selenium

Selenium tests were developed during the production of the website. Every time new functionality was added, existing tests would be re-run to check that existing functions had not been affected negatively. Selenium focuses on testing user interactions with a system and checking the result. As a result, the test cases developed for this part of testing were all user interactions with the website forms. They checked that both successful inputs resulted in the desired output and incorrect inputs (for example, getting your password wrong) resulted in errors being handled correctly.

A full description of each of the test cases can be found in appendix D.2.1.

Fig. 5.2 gives an example of the test case *TestLoginWrongCredentials*. This test case was used to check that an incorrect password would result in the user not being logged in and the alert “Failure – Incorrect credentials” being shown.



Command	Target	Value
open	/goclub/members/	
type	id=email	ross@fabric.com
type	id=password	1
clickAndWait	css=input[type="submit"]	
verifyTextPresent	Failure - Incorrect credentials	

Figure 5.2 - TestLoginWrongCredentials

5.2.2 Functional testing

Manual functional testing of the website was conducted in the same way as done for the application (detailed in section 5.1.2 Functional testing). Test cases were chosen ensuring all functional requirements were tested; 100% of the website's functional requirements detailed in the requirements document were tested by at least one test case. A summary document can be seen in *Appendix D.2.2 – Manual Website Functional Testing.*

One interesting test case was TW5. This checked whether, during registration, a string that did not match the format of an email address would be accepted in the email address form field. This test initially failed as although client side protection against non-valid emails had been put in place, this did not perform the suitable checks required and an account with an invalid email address was allowed to register. To fix this, server-side checks (using regular expressions) were placed on the email field and all other fields that required checking.

5.2.3 Compatibility testing

It was of importance that the website displayed correctly when viewed by a wide range of browsers and operating systems, so not to limit the users that could access the website. Both automated and manual testing of the website were undertaken. These are detailed below.

5.2.3.1 Automated Testing

Using the free service provided by http://browsershots.org/, how the website displayed on a wide range of operating system and browser combinations was checked. The website display was tested over 150 times and fully worked on all 39 Firefox tests, all 23 chrome tests, all 50 windows tests and all 38 linux tests. The website did have small problems displaying on some older browser and operating system combinations, such as a combination of using the Arora (Version 0.11) web browser with the Debian operating system; however it is felt that this does not pose a problem to the majority of users so does not warrant changes being made.

The main disadvantage of Browsershots [40] is its lack of interactivity, you can check a single static page is displayed correctly, but not that interactions are handled correctly. Adobe BrowserLab [41] is an alternative tool that provides slightly increased interactivity testing (Links can be clicked). Another benefit of BrowserLab is that OSX and Safari can be tested. The website passed all the BrowserLab tests including displaying on IE and OSX.

Although BrowserLab allows links to be clicked, the testing of interactivity is still limited. For example, forms cannot be filled in so it is not possible to login; this is why manual testing was also undertaken.

5.2.3.2 Manual Testing

To conduct manual testing, the website's Use Cases were acted out on all the major browsers: Chrome, Firefox, IE and Safari. Generally results were good, as would have been expected considering the use of the Twitter Bootstrap framework which markets itself on its cross-browser compatibility.

One interesting problem found was that users on the Safari browser were unable to logout of the system. The code for the log out button is shown below in Fig. 5.3. Safari would execute the link re-direction, but not the included PHP code.

```
<div class="nav2">
      <a href="/goclub/members/"><?php include $_SERVER['DOCUMENT_ROOT'] . '/goclub/includes/logout.html.php'; ?></a>
   </div>
```

Figure 5.3 – Code causing problem

To overcome this, the code was modified to that shown in Fig. 5.4; the link was removed from its current location and placed as a header redirect after the PHP had executed. Logout now works successfully on all tested browsers.

```
<div class="nav2">
      <?php include $_SERVER['DOCUMENT_ROOT'] . '/goclub/includes/logout.html.php'; ?>
   </div>
```

Figure 5.4 – Modified code

5.3 Test summary

A wide variety of tests were used, some during development (e.g Robotium and Selenium) and some at the end of the project (e.g. Stress testing and Manual Functional testing). The automated testing methods that were used resulted in errors being found, which may not have been found if only manual testing had been undertaken.

The benefit of undertaking a wide variety of testing, and fixing the bugs found, is that there can be confidence that both the application and website now have met all the tests they have been subjected to. From testing, it can also be confirmed that the application meets all requirements set out at the start of this project in the requirements document.

Furthermore, the production of automated tests that can be re-run whenever desired, (through Monkey, Robotium and Selenium) is useful in case bugs are found once the application is in production. If a bug is found, the appropriate code can be modified to fix this bug, and tests re-run to check that the modifications have not caused problems elsewhere in the program. This gives confidence that the application will continue to function correctly.

Chapter 6

Conclusion and Evaluation

6.1 Summary of achievements

This project has achieved all the goals set out in the tables in section 1.5.2.1 and 1.5.2.2. There were twelve goals relating to the application and eleven goals relating to the website. A fully working and heavily tested Android application has been developed delivering all requirements detailed in the requirements documentation. This is backed up by a fully functional back-end web server, that is currently live, that also meets all its specific requirements.

6.2 Evaluation of project

6.2.1 The Problem/Challenges

As detailed in the Introduction, the application was developed to deal with a problem. The problem was that students did not know which clubs to go to and had little knowledge of a club's event, or how to get to the club. The students were also subjected to the underhanded tactics of promoters.

The application allows a number of adverts to be searched. Depending on the user's criteria, a user is able to find information on events and chose the event they know they will most enjoy; this helps them decide what club to go to. Providing both a map (with a route from the user's location) and an Augmented Reality view to identify a club from the outside, users now can be sure how to get to the desired club. Now that students know where the club is located, and that a special deal is waiting for them, there is no reason to have to talk to a promoter.

The Introduction chapter also outlined three challenges of the problem. The first challenge was that the application needs a way to ensure adverts are being honoured when users get to the club. The review system offered by the application allows users to comment on their experience with a club. When clubs do not honour their adverts, this may lead to users submitting negative reviews. Getting negative reviews will flag to GoClub and also its users that a club may not be honouring deals.

The second challenge was that current journey planners only directed you to nearby the club, maybe the street it is on, or for example*, "1 Leicester Square"* (an area that does not have visible numbers). Users may not be able to locate the club although in its proximity. The Augmented Reality view now allows a user to hold up their device's camera, and locate the club with the GoClub marker.

The third challenge was to integrate a mechanism for the future that would allow GoClub to record how many users had got to each club. This was so that GoClub could calculate the correct amount of commission to be paid by the club, assuming payment is paid per-customer that enters the club. Once the user claims they are at a club, the application checks the user's location and reports to the server that they are at the club if they are within 0.2km of the club's location.

The application was developed with the user and the problem in mind; a number of potential users were consulted throughout the project to gain students' opinions on functionality and the user interface of the design. It is felt that the application addresses the problem well.

6.2.2 Design and Implementation

Android applications are coded in Java. Java programs rely on classes, therefore the application is split into different classes. The classes represent different components of the application. Each Activity required needs an associated class, so the different Activities in the application dictate many of the classes required.

The application follows a number of Android's recommendations for implementing certain functionality. One of these is how the application obtains and uses the user's location. The first available location is used initially, and as more reliable and accurate locations become available these are then used. This is covered in more detail in *section 4.2.5.*

Throughout the development, there were often a range of different implementation approaches that could be followed and decisions had to be made on which approach to take. Efficiency, number of connections required to the backend server and time were all taken into account when deciding between approaches. One example of such a decision was in *section 4.2.5*, where the approach decided upon moved some processing to the backend server, meaning less data was sent to and processed on the actual device. This is useful as the application can be run on a variety of devices, with varying processing power. Another example of where a decision had to be made was how distances should be calculated for the sorting of adverts in a list (more details in *section 4.2.4).* It was decided that an as-the-crow-flies value should be used. This was used instead of connecting to an online route API and retrieving the distance for each in the list. Although as-the-crow-flies provides less accuracy, the values calculated will generally be relatively correct, so sorting between values can be done successfully. This method has the advantage of not requiring an internet connection, so results can be displayed without a connection and can be computed with much greater speed.

The application attempts to preserve the battery life of the device upon which it is running in a number of ways. One example of how this was done can be seen in *Section 4.2.7*, where a more efficient and less demanding-on-the-battery solution was used. Another way that the application saves battery is the starting and stopping of demanding processes (such as getting the location) on the starting and stopping of activities that hold the process. For example, when the map view is navigated away from, the location getter is paused and then resumed once the map is viewed again; this is so that the location is not got while it is not being used.

In terms of the User Interface design, as detailed in *section 4.2.10,* the application follows a number of Android design guidelines and patterns, especially related to the design of icons and customized buttons. In terms of the website's user interface, the use of a CSS/HTML framework, Twitter Bootstrap, is recommended over producing the CSS for the website from scratch. This is because the framework deals with many compatibility issues resulting in a compatible design. As a result, a wide range of users using varying combinations of operating systems and browsers can access the website successfully. This was confirmed in section 5.2.3.1 where the website UI was tested over 150 times and was confirmed to work on all commonly used browser/operating system combinations.

It is noticeable that code written at the end of the project was more coherent and well-structured than code written at the start. This was because the project was a learning experience and an iterative approach was aimed for, meaning development of the system was started at a very early stage. As further knowledge of Android programming was gained, it was occasionally the case that previously written code was returned to and time was spent altering the code. Alterations were often implemented to either improve efficiency or functionality. An example of where functionality was improved was the *isNetworkOnline* method. This was originally developed to check for WiFi connection; it was not realised it did not check for mobile internet connection. After further knowledge of Android's *ConnectivityManager* class was gained (the class that network checks can be performed on), it was realised that the method was only checking for WiFi connection and as a result, the method was altered to check for both WiFi and mobile internet.

Although the website separated the view (stored in *.html.php* files) from any logic (stored in *.php* or *.inc.php* files), it is understood that the design should have further imposed a separation of logic into either a model (manages data) or controller (controls interactions between model and view).

A complete separation of code into Model, View and Controller is called the Model-View-Controller[42] (MVC) design pattern. Using the MVC, minimises the repetition of code. For example, instead of undertaking security checks every time an "INSERT" SQL statement is called, calls to a method in the model can be done to execute inserts and inside the method security checks are done. This means the security check code only has to be written once and not multiple times. The MVC pattern was not originally followed as the book (detailed in *Section 2.1.5*) that was used to learn PHP from did not cover these topics. Furthermore, there are a range of PHP frameworks (such as http://cakephp.org/) that could have been used to enforce the MVC design pattern and usually provide pre-built functions for tedious tasks. CakePHP [43], like other PHP frameworks, offers built-in security and protection against common attacks, including SQL injection protection and XSS prevention. Although the GoClub system attempts to prevent against certain attacks, it is likely that a professionally coded framework will provide better protection.

6.2.3 Implementation Technologies

There were a number of decisions to be made throughout the project as to which technologies to use. Many technologies to use were pre-determined from the offset of the project. For example, an Android application must be written in Java. For Facebook integration, the Facebook SDK must be used. Other technologies were chosen because they had the best fit with what was already being used, for example, the Google Directions API was used (over other Direction API's from Microsoft and MapQuest, for example) to retrieve information for overlaying on the MapView. This was because the MapView was supplied by Google, so Google Directions API is the recommended route provider. The biggest decision that was made was which Augmented Reality SDK to use. There are a wide range of Augmented Reality SDKs for Android so each had to be investigated. Wikitude was chosen because it focused on location-based Augmented Reality, whereas other Augmented Reality SDKs focused on image recognition. Furthermore, it was felt that the Wikitude documentation was the best of what was available. A further discussion of why Wikitude was chosen can be seen in *section 2.3.1.*

For the webserver, PHP was chosen as it provided the desired database interactivity; this was due to the ability to embed SQL statements. Furthermore, Database driven websites were new to me and PHP is easy to learn, enabling development of the system to start early. PHP and MySQL are both supported by almost all web hosting services, so the choice of host is therefore not limited.

6.3 Future developments

A number of future developments are listed below:

- If the application were to be launched, the system should be able to accept payments from clubs, so that revenue could be generated from the application. Clubs would have a number of different payment plans to promote their adverts on the application; including paying to list adverts, paying per-impression, paying per-notification-pushed or paying per-customer that enters their club using GoClub. Payment would be accepted through the website and would be integrated to be a part of the logged-in member's pages.
  To accept payment, the website's security would have to be improved. This would include transitioning the website to work with an HTTPS connection.
- The backend server is currently hosted on a simple web hosting server. This server is shared between a number of customers of the hosting company; each customer is competing for bandwidth and the website can be slow to respond. This solution does not offer scalability and further resources cannot be allocated if usage increases dramatically meaning the server will get increasingly slow with an increase in usage. A solution to this and future development to the project would be to transition the backend onto a cloud service, such as *Amazon EC2*. A cloud service will allocate further servers to the application if usage increases and this would provide a more scalable solution that is ready to handle variations in usage.
- The application could be developed to use speech recognition to allow users to provide voice commands to navigate the application. This would be useful especially when a user is travelling to a club and may prefer to give voice commands instead of using their fingers for convenience as they can remain looking at where they are walking. In winter the demand for voice commands would increase further as people may be wearing gloves making interactions with touch screens difficult. Android provides a number of classes that aid the integration of speech recognition in applications.
- The application's value to a user relies heavily on user submitted content (adverts placed by clubs). If there are no adverts placed, there would be no use to the application; students are likely to attempt to find adverts once and if no adverts can be found, forget about or even give the application negative user reviews. To overcome the problem of having periods of low advert listings (for example, when the application has just launched), the system could scrape club ticketing websites, such as FatSoma[44], to retrieve details of current events.
- At the moment, the application is based around getting deals when paying on the door at clubs. Many clubs offer discounts for buying tickets in advance. A mechanism to buy tickets through the application could be set up. GoClub will receive the commission on any sale.

6.4 Conclusions

I feel this project has given me the opportunity to learn valuable new technologies and skills that will benefit me in my future career. There are many technologies that were totally new to me at the start of this project that I have now learnt and experienced, and I will use the knowledge gained in the future. Firstly, I have learnt to develop Android Applications; gaining knowledge of the Android SDK (and the classes it provides) and Android design patterns. Secondly, I have learnt how to integrate third party SDKs (Wikitude SDK for Augmented Reality, Facebook SDK) into a software product; something that I had not done before. Thirdly, I have been made aware of the benefits of using frameworks, such as Twitter Bootstrap that was used for the website layout. Furthermore, for the project to remain on track and a successful outcome to be reached, a great deal of organisation was required to balance putting the time in for this project and commitments to other pieces of final year work. The project has enhanced my organisational skills. I have also never been involved in managing a project of this scale; I have learnt to remain motivated throughout a project's duration, and also to set myself smaller (weekly) goals that when combined set out a timeline of when work should be completed for a successful end result.

I am very pleased with the outcome of the project and am proud of what I have produced. I intend to launch the application onto the Google Play store after the exam period.

Appendix A

Requirements Document

Application

Functional Requirements

ID	Description	MoSCoW
Category: Retrieving and storing adverts		
FRA1.1	The application should connect to an online web server to retrieve new adverts every time the application is opened.	Must Have
FRA1.2	The application should store these adverts on an internal database to allow quick access.	Should Have
FRA1.3	The application should allow access to adverts currently stored on its internal database while a network connection is not present (while the phone is offline).	Should Have
FRA1.4	The application should connect to an online database to retrieve club information and review data.	Must Have
FRA1.5	The application should remove any adverts from the internal database if they have been removed by the promoter on the website.	Must Have
Category: Displaying advert listings		
FRA2.1	The application should display the GoClub logo on its main page.	Should Have
FRA2.2	The application should allow the user to filter adverts by entry price.	Should Have
FRA2.3	The application should allow the user to filter adverts by night available.	Should Have
FRA2.4	The application should allow the user to filter by distance from the club.	Must Have
FRA2.5	The application should sort adverts by the location of the user; displaying the adverts from nearest to furthest from the user.	Should Have
FRA2.6	The application should display the distance to each club from the user's location.	Should Have
FRA2.7	The application should allow the user to click on an advert to view more details.	Must Have
FRA2.8	The application should display a blank search results box if no results are found.	Should Have
Category: Advert Details Page		
FRA3.1	The application should display details of the club; including its name, address and star rating.	Must Have
FRA3.2	The application should display the club's profile picture, if one has been uploaded.	Should Have
FRA3.3	The application should display a default picture if a profile picture has not been uploaded.	Should Have
FRA3.4	The application should display the title and description of the advert.	Must Have
FRA3.5	The application should display the days an advert is valid.	Must Have
FRA3.6	The application should display the entry price.	Should Have
FRA3.7	The application should allow the user to click "view reviews/ratings" to view user reviews.	Should Have
FRA3.8	The application should allow the user to click redeem to view a map to the club.	Should Have
FRA3.9	The application should only allow the redeem button to be clicked if an internet connection is present.	Should Have
Category: Review System		
FRA4.1	The application should display the average star rating of a certain club.	Must Have
FRA4.2	The application should display all the user reviews of a certain club.	Must Have
FRA4.3	The application should display the star rating given to a club by an individual next to their review.	Should Have
FRA4.4	The application should allow a user to give their own review of the club.	Should Have

FRA4.5	The application should allow a user to give their own star rating of the club.	Should Have
FRA4.6	The application should report back to the user whether their review/rating submission was a success or failure.	Should Have
Category: Mapping System		
FRA5.1	The application should provide a map pinpointing the clubs' location with the GoClub logo as a marker.	Should Have
FRA5.2	The application should provide a route from the user's location to the club.	Should Have
FRA5.3	The application should calculate the best on-foot route.	Should Have
FRA5.4	The application should not recalculate the route on a change in orientation of the screen.	Should Have
FRA5.5	The application should calculate the user's location when the "claim now" button is clicked.	Should Have
FRA5.6	The application should report to the website that a user has reached the club, if the user is within 0.2km from the club when the update button is pressed.	Should Have
Category: Connecting to Facebook		
FRA6.1	The application should allow the user to auto fill the registration inputs (when the app is first downloaded) by retrieving the appropriate data from Facebook.	Should Have
FRA6.2	The application should post a message to a user's Facebook stating that they are going to a certain club.	Must Have
FRA6.3	The application should ask the user's permission before making any post to Facebook.	Should Have
FRA6.4	The application should report to the user once it has successfully posted to Facebook.	Should Have
FRA6.5	The Facebook post should include a picture with the GoClub logo to increase brand awareness and visibility of post.	Could Have
FRA6.6	The Facebook post should be in the present tense: "[User's name] is going to.." while the event is happening.	Should Have
FRA6.7	The Facebook post should expire and change to the past tense ("[User's name] went to..") after the event.	Could Have
FRA6.8	The Facebook post should include the clubs name as a variable which can change depending on the club: "[User's name] is going to [Club Name]".	Should Have
FRA6.9	The Facebook post should allow users to click on the clubs' name and this will direct them to a custom page for that club, detailing that the selected club is using GoClub to promote offers; this will provide a further link to download the application.	Could Have
Category: Reporting Statistics		
FRA7.1	The application should report to the server when an advert is viewed.	Could Have
FRA7.2	The application should report to the server when an advert is redeemed.	Could Have
Category: Push Notifications		
FRA8.1	The application should be able to receive 'pushed' adverts from the website as notifications.	Must Have
FRA8.2	The application should be able to receive notifications even when the application is not open.	Should Have
FRA8.3	The application should allow a notification to be clicked on; this will open up the application and display the clicked on notification.	Should Have
FRA8.4	The application should limit the number of notifications a user can receive in one night to 3.	Should Have
FRA8.5	The application should only receive push notifications from clubs within 1km of the user's current location.	Must Have
FRA8.6	The application should have a custom "GoClub" logo as the notification logo.	Should Have
Category: Augmented Reality		
FRA9.1	The application should provide an Augmented Reality view.	Should Have
FRA9.2	The application should display markers showing the position of clubs over the images being displayed on the screen from the devices camera.	Must Have
FRA9.3	The application should only display markers over clubs that are within 1 kilometre of the user.	Could Have

FRA9.4	The application should allow the user to click on a marker placed on the screen and view currently available offers at the selected club.	Should Have
FRA9.5	The application should allow the user to click on a marker placed on the screen and view reviews of the selected club.	Should Have
Category: Speech Recognition		
FRA10.1	The application should allow the user to filter the initial advert listings by speech, instead of selecting the search criteria by hand.	Would Have
FRA10.2	The application should allow the user to perform basic navigation through the application by spoken command.	Would Have

Non - Functional Requirements

Category: Scalability		
NFRA1.1	The application must be scalable and ready to handle wide variations in usage and user numbers.	Should Have
Category: User Interface / Compatibility		
NFRA2.1	The application should work on a range of Android mobile devices with different screen sizes and resolutions.	Should Have
NFRA2.2	The application should conform to Android's design guidelines; to allow for a future inclusion on the android marketplace if desired.	Should Have
NFRA2.3	The application should have a professionally made user interface.	Should Have
NFRA2.4	The user interface should be consistent with the current GoClub colour scheme (dark blue and dark grey).	Should Have
Category: Performance		
NFRA3.1	The application should take no more than 5 seconds to load on any occasion.	Should Have
NFRA3.2	The user interface should report to the user if there is a delay moving from one activity to another.	Should Have
Category: Usability		
NFRA4.1	The application should not become unresponsive, if there is a delay in response from the backend webserver.	Should Have
NFRA4.2	The application should gracefully handle a loss of internet connection or no internet connection at all.	Should Have
NFRA4.3	The application should not become unresponsive in the case of lengthy processes, such as downloading and rendering images.	Should Have
NFRA4.4	The end user should be able to navigate through the application to retrieve directions to a selected club in less than 30 seconds.	Should Have
NFRA4.5	The end user should be able to navigate through the application to view reviews of a club in less than 30 seconds.	Should Have
NFRA4.6	The application should allow users (whatever their skill level) to download and install the application with ease.	Should Have
Category: API use		
NFRA5.1	The Augmented Reality should plug into the Wikitude Augmented Reality API.	Should Have
NFRA5.2	The Route Calculations to clubs should be done via connecting to the Google Directions API.	Should Have

Website

Functional Requirements

ID	Description	MoSCoW
Category: Student Front end		
FRW1.1	The website should include a student homepage, which promotes the application to target users.	Should Have
FRW1.2	The website should provide details on how a student would start using the app.	Should Have
FRW1.3	The website should provide a way to download the application.	Must Have
FRW1.4	The website should provide a way for promoters to navigate to the promoter sections of the website.	Should Have
Category: Promoters		
FRW2.1	The website should provide a homepage for club management/promoters. This should explain the benefits of promoting with GoClub.	Must Have
FRW2.2	The website should provide a clear way to log in or register.	Must Have
Category: Promoters > Accounts > Registration system		
FRW3.1	The Registration system should allow un-registered promoters to register.	Must Have
FRW3.2	The Registration system should collect the following details on a user: Email address, password, club name, description of club, address of club, clubs longitude and latitude.	Must Have
FRW3.3	The Registration system should check that an input email address is not registered to an already existing account.	Must Have
FRW3.4	The Registration system should accept input for the password field twice to ensure it is correctly typed.	Should Have
FRW3.5	The Registration system should check whether it has received a correctly formatted email address.	Should Have
FRW3.6	The Registration system should enforce that passwords are a suitable level of complexity. Passwords should contain at least one upper case, one lower case and one number.	Should Have
FRW3.7	The Registration system should enforce that all registration fields are mandatory.	Should Have
FRW3.8	The Registration system should email the user on a successful, registration so they can activate their account by clicking on a personalised activation link.	Should Have
FRW3.9	The Registration system should repopulate the registration form with the user-input values, if the form is not submitted successfully and the page is reloaded.	Should Have
FRW3.10	The Registration system should display an error message if the registration form is not completed successfully; detailing the error and how it can be resolved.	Must Have
FRW3.11	The Registration system should display a message instructing the user to check the provided email account for an activation link if the form is submitted successfully.	Must Have
FRW3.12	The Registration system should use an image CAPTCHA system to check that registrations are not automated by computers.	Must Have
FRW3.13	The Registration system should allow the user to pin point their clubs' location on a map.	Must Have
Category: Promoters > Accounts > Login system		
FRW4.1	The Login System should allow promoters to login, given they supply the correct email address and password.	Must Have
FRW4.2	The Login System should not allow access to accounts that have not been email activated.	Must Have
FRW4.3	The Login System should not allow access to accounts that have not been approved by the website administrator.	Must Have
FRW4.4	The website should provide promoters a clear way to logout.	Must Have
Category: Promoters > Club Details > General		
FRW5.1	The website should allow promoters to edit previously-stored details on their club.	Must Have

Category: Promoters > Club Details > Image upload system		
FRW6.1	The Upload System should allow promoters to upload a profile picture of their club, to be displayed on the app.	Must Have
FRW6.2	The Upload System should limit this upload to a certain file size.	Must Have
FRW6.3	The Upload System should limit this upload to a certain file type.	Should Have
FRW6.4	The Upload System should crop and resize images; so they are suitable for the application.	Should Have
Category: Promoters > Managing Adverts > Advert Dashboard		
FRW7.1	The Advert Dashboard should allow promoters to see all their currently running adverts.	Must Have
FRW7.2	The Advert Dashboard should allow promoters to view whether a certain advert is active or inactive.	Could Have
FRW7.3	The Advert Dashboard should display some basic statistics about an advert: impressions (views) and conversions.	Could Have
FRW7.4	The Advert Dashboard should provide access to further functionality: Viewing an individual advert, pushing an advert, editing an advert, adding an advert.	Should Have
FRW7.5	The Advert Dashboard should allow promoters to delete selected adverts.	Should Have
Category: Promoters > Managing Adverts > Adding an advert		
FRW8.1	The website should allow promoters to specify the days of the week that the advertised offer will be valid on.	Should Have
FRW8.2	The website should allow promoters to specify the details/description of an offer.	Must Have
FRW8.3	The website should allow promoters to specify the cost of entry when redeeming the certain offer.	Must Have
FRW8.4	The website should enforce that the entry field only accepts numerical values	Should Have
Category: Promoters > Managing Adverts > Individual Advert page		
FRW9.1	The website should display an approximate preview of what the advert looks like on the app.	Should Have
FRW9.2	The website should display the full title and description of the advert.	Must Have
Category: Promoters > Managing Adverts > Individual Advert page > Statistics		
FRW10.1	The website should report how many views an individual advert has received.	Should Have
FRW10.2	The website should break down overall views and redeems into monthly data to give an overview of the monthly usage.	Should Have
FRW10.3	The website should report how many users actually reached the club, in the last week of usage.	Should Have
FRW10.4	The website should graphically display through graphs the monthly views/redeems and also the usage over the last week.	Should Have
Category: Promoters > Managing Adverts > Editing adverts		
FRW11.1	The website should allow all details of an advert to be edited.	Should Have
FRW11.2	The website should allow promoters to pause (set to inactive) an advert; meaning it will not be displayed on the application.	Could Have
FRW11.3	The website should allow promoters to restart (set to active) a paused advert.	Could Have
Category: Promoters > Managing Adverts > Pushing adverts as a notification		
FRW12.1	The website should provide a way for clubs to push an advert as a notification to the user’s device.	Must Have
FRW12.2	The website should provide a way to target specific users by university, when pushing an advert.	Should Have
Category: Promoters > Reviews		
FRW13.1	The website should allow a promoter to view all the reviews that have been made about their club.	Should Have
FRW13.2	The website should allow a promoter to flag individual reviews for the admin to then review and if required, take further action.	Should Have
Category: Promoters > Support		
FRW14.1	The website should provide a way for clubs to contact the website administrators.	Should Have

Category: Admin		
FRW15.1	The website should provide a dashboard for the administrator.	Must Have
FRW15.2	The admin dashboard should flag whether there are any pending administrator tasks to be completed.	Should Have
FRW15.3	The website should provide information on new club registrations.	Should Have
FRW15.4	The website should allow an administrator to approve or deny new club registrations.	Should Have
FRW15.5	The website should, on approval, create a publicly accessible page advertising that the club is using GoClub. This is relevant to FRA6.9	Should Have
FRW15.6	The website should display to the administrator all the flagged club reviews that need to be moderated.	Should Have
FRW15.7	The website should allow an admin to either remove a flag from a comment (resulting in it being sent to apps when a new review request is made) or delete it permanently.	Should Have

Non - Functional Requirements

Category: Scalability		
NFRW1.1	The website must be scalable and ready to handle wide variations in usage and user numbers.	Should Have
Category: Performance		
NFRW2.1	The website should respond to all requests and load the required information in under a second.	Should Have
NFRW2.2	The website should be able to handle at least 100 concurrent users without a considerable loss of performance.	Should Have
Category: Compatibility		
NFRW3.1	The website should be compatible with all major browsers.	Should Have
NFRW3.2	The website should be compatible with all major operating systems.	Should Have
Category: Usability		
NFRW4.1	The promoter should be able to navigate through the website to add an advert in less than a minute.	Should Have
NFRW4.2	The promoter should be able to navigate through the website to view the usage statistics in less than a minute.	Should Have

Appendix B

Use Case Document

Application Actors

Student – This actor represents a user of the application.

Web Server – This actor represents the backend web server that supports the application, providing it data and functionality.

Facebook Server – This actor represents the Facebook SDK. It is used when the user connects to Facebook.

Website Actors

Unregistered User – This actor represents a user of the website that has not registered a user account.

Promoter – This actor represents someone who has signed up to the website as a promoter of a club.

Admin – This actor represents an authorised user who is in charge of running the website.

User – An abstraction of Promoter and Admin, used for simplicity, to describe a user of the website. It is used to group Promoters and Admins in Use Cases that they share.

Student – This actor represents a student visiting the website to download the application.

Application – Use Case Description

USE CASE	**StudentRegistersWithApplication**
ID	AUC1.1
BRIEF DESCRIPTION	A Student registers their details with the application.
PRIMARY ACTORS	Student
SECONDARY ACTORS	Web Server, Facebook Server
PRECONDITIONS	The registration flow has not previously been completed.
MAIN FLOW	The use case starts when the Student opens the application for the first time. 1) The Application displays a registration form. 2) The Student has the option to connect via Facebook (and auto-fill the form with details from there). 2.1) If the Student connects via Facebook: <<include(ConnectWithFacebook)>>. The Application retrieves the Students' name from their Facebook. 2.2) If the Student does not connect via Facebook, the Student fills out their name manually. 3) The Student enters their university, and clicks submit. 4) The Application registers them and displays a success notification.
POST CONDITIONS	The Student has registered with the application.

ALTERNATIVE FLOW/S	NoDataProvided

USE CASE	**StudentRegistersWithApplication:** NoDataProvided
ID	AUC1.2
BRIEF DESCRIPTION	A Student does not complete all fields when completing the registration process.
PRIMARY ACTORS	Application
SECONDARY ACTORS	Student
PRECONDITIONS	The registration flow has not previously been completed.
MAIN FLOW	The alternative flow starts after stage 3 of the main flow. 1) The Application alerts the student that they have not filled in all form fields. 2) The Student fills in the remaining fields, and clicks submit. 3) The Application registers them and displays a success notification.
POST CONDITIONS	The Student has registered with the application.
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentSelectsClubViaAugmentedReality**
ID	AUC2
BRIEF DESCRIPTION	A Student clicks on a marker positioned over a club in Augmented Reality (AR) view.
PRIMARY ACTORS	Student
SECONDARY ACTORS	None
PRECONDITIONS	The Application is running on a device capable of supporting Augmented Reality.
MAIN FLOW	The use case starts when the Student is in Augmented Reality mode. 1) The Student clicks on a marker. 2) The Application displays a pop up with details about this marker. 3) The Student clicks for more information. 4) The Application loads a new page and provides the option to either view reviews of the club or view current offers. <<Student views club offers>>
POST CONDITIONS	A Club has been selected.
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentViewsDealsFromSelectedClub**
ID	AUC3.1
BRIEF DESCRIPTION	A Student views all the deals available tonight at a certain club.
PRIMARY ACTORS	Student
SECONDARY ACTORS	None
PRECONDITIONS	A club has been selected through the Augmented Reality view.
MAIN FLOW	**Segment Student views club offers** 1. The System displays the offers available at the club tonight.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	NoDeals

USE CASE	**StudentViewsDealsFromSelectedClub:** NoDeals
ID	AUC3.2
BRIEF DESCRIPTION	There are no deals from the selected club to display.
PRIMARY ACTORS	Application
SECONDARY ACTORS	Student
PRECONDITIONS	A Student has attempted to view deals available at a certain club. There are no deals listed.
MAIN FLOW	The alternative flow starts after stage 1 of the main flow. 1) The Application displays the search results page with no results. 2) The Student navigates back to Augmented Reality view.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentSearchesDeals**
ID	AUC4.1
BRIEF DESCRIPTION	A Student searches deals based on input search criteria.
PRIMARY ACTORS	Student
SECONDARY ACTORS	None
PRECONDITIONS	None
MAIN FLOW	The use case starts when the Student is on the homepage of the application. 1) The Application displays a search form, allowing the Student to search by various criteria. 2) The Student selects the criteria they wish to search by, and clicks submit. 3) The Application returns the results matching the criteria searched on.
POST CONDITIONS	A list of adverts is displayed.
ALTERNATIVE FLOW/S	NoDeals

USE CASE	**StudentSearchesDeals:** NoDeals
ID	AUC4.2
BRIEF DESCRIPTION	There are no adverts matching the Students search criteria.
PRIMARY ACTORS	Application
SECONDARY ACTORS	Student
PRECONDITIONS	The Student has executed a search that has returned no results.
MAIN FLOW	The alternative flow starts after stage 2 of the main flow. 1) The Application displays the search results page with no results. 2) The Student navigates back. 3) The Application displays the search criteria page. 4) The Student alters their search criteria, and submits. 5) The Application returns the results matching the criteria searched on.
POST CONDITIONS	A list of adverts is displayed.
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentViewsAnOffer**
ID	AUC5
BRIEF DESCRIPTION	A Student views detailed information about a certain deal.
PRIMARY ACTORS	Student
SECONDARY ACTORS	None
PRECONDITIONS	An advert has been selected.
MAIN FLOW	The use case starts when the Student clicks on a certain offer from search results. 1) The Application displays full details of the offer and details on the club including name, address, rating and a picture.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentViewsReviews**
ID	AUC6
BRIEF DESCRIPTION	A Student views reviews of a certain club.
PRIMARY ACTORS	Student
SECONDARY ACTORS	None
PRECONDITIONS	None
MAIN FLOW	The use case starts when the Student clicks “View Reviews/Offers” for a certain club. 1) The system displays a list of the current reviews.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentPostsToFacebook**
ID	AUC7.1
BRIEF DESCRIPTION	The Application posts a message to Facebook stating the Student is going to a certain club.
PRIMARY ACTORS	Student
SECONDARY ACTORS	Facebook Server
PRECONDITIONS	None
MAIN FLOW	The use case starts when a Student clicks “redeem offer” from an individual offer page. 1) The Application displays a pop up asking for permission to post to Facebook. 2) The Student provides the permission. 3) If the student hasn’t connected with Facebook previously: <<include(ConnectWithFacebook)>>. 4) The Application posts to Facebook and reports a success back to the Student.
POST CONDITIONS	A post has been made on behalf of the Student to their Facebook page.
ALTERNATIVE FLOW/S	NoPermissionGranted

USE CASE	**StudentPostsToFacebook:** NoPermissionGranted
ID	AUC7.2
BRIEF DESCRIPTION	The appropriate permission is not granted by the Student.
PRIMARY ACTORS	Application
SECONDARY ACTORS	None
PRECONDITIONS	None
MAIN FLOW	The alternative flow starts after stage 1 of the main flow. 1) The Student does not provide permission.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentGetsRouteToClub**
ID	AUC8
BRIEF DESCRIPTION	A Student views a route from their current location to a selected club.
PRIMARY ACTORS	Student
SECONDARY ACTORS	None
PRECONDITIONS	An internet connection is present.
MAIN FLOW	The use case starts when a Student clicks “redeem offer” from an individual offer page. 1) The Application displays a route from the Students location to the club.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentClaimsOffer**
ID	AUC9
BRIEF DESCRIPTION	A Student claims a selected offer at a certain club.
PRIMARY ACTORS	Student
SECONDARY ACTORS	Web Server
PRECONDITIONS	An internet connection is present. The Student is at the club.
MAIN FLOW	The use case starts when a Student is on the redeem offer page. 1) The Student clicks “Claim Offer” once they are at the club. 2) The Application displays a pop-up for the club door staff to examine; this displays the details of the stated offer. 3) The Application reports to the webserver that the student has reached the club.
POST CONDITIONS	The statistics on number of Students to reach the club have been modified.
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentSubmitsClubReview**
ID	AUC10
BRIEF DESCRIPTION	A Student submits a review of a certain club.
PRIMARY ACTORS	Student
SECONDARY ACTORS	Web Server
PRECONDITIONS	None
MAIN FLOW	The use case starts when a user clicks “add review” 1) The Application displays the add review page. 2) The Student selects what star rating they give the club. 3) The Student inputs a review and clicks submit. 4) The Application submits the review to the Web Server and reports back that the review has been submitted successfully.
POST CONDITIONS	A review has been submitted.
ALTERNATIVE FLOW/S	N/A

USE CASE	**StudentReceivesPushedAdvert**
ID	AUC11
BRIEF DESCRIPTION	A Student receives an advert as a push notification to their phone.
PRIMARY ACTORS	Web Server
SECONDARY ACTORS	Student
PRECONDITIONS	An internet connection is present. The student is in close proximity to a club.
MAIN FLOW	The use case starts when the Web Server has a notification to push. 1) The Application generates a notification to appear on the phone. 2) The Student views the notification and clicks it, opening the app. 3) The Application displays a page displaying the details of that offer.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**ConnectWithFacebook**
ID	AUC12
BRIEF DESCRIPTION	A Student provides authentication credentials and connects with Facebook.
PRIMARY ACTORS	Student
SECONDARY ACTORS	Facebook Server
PRECONDITIONS	The Student has a Facebook account.
MAIN FLOW	The use case starts when the Student provides their Facebook authentication details. 1) The Student clicks submit. 2) The system authenticates the user.
POST CONDITIONS	The Student has connected with Facebook.
ALTERNATIVE FLOW/S	N/A

Website – Use Case Descriptions

The following Use Cases have been included in this appendix. To conserve space, some use cases have been omitted, these are listed at the end of this appendix.

USE CASE	**UnregisteredUserRegisters**
ID	WUC1.1
BRIEF DESCRIPTION	An Unregistered User registers with the website.
PRIMARY ACTORS	Unregistered User
SECONDARY ACTORS	Website
PRECONDITIONS	None
MAIN FLOW	The use case starts when the Unregistered User clicks “Register Now” on the log-in page. 1) The Website displays the registration form. 2) The Unregistered User fills in details about themselves and the club. 3) The Unregistered User pin points the clubs location on a map. 4) The Unregistered User completes the image CAPTCHA. 5) The Unregistered User clicks submit. 6) The Website reloads the page with an alert instructing the Unregistered User that an account has been made and they must verify the email address provided.
POST CONDITIONS	An account has been registered. It must now be verified by the user and approved by the administrator.
ALTERNATIVE FLOW/S	MandatoryDetailsNotFilledIn; EmailAddressRegistered; ImageCaptchaWrong

USE CASE	**UnregisteredUserRegisters:** MandatoryDetailsNotFilledIn
ID	WUC1.2
BRIEF DESCRIPTION	Registration fails as not all mandatory fields are filled in.
PRIMARY ACTORS	Website
SECONDARY ACTORS	Unregistered User
PRECONDITIONS	An Unregistered User has submitted the registration form with input values missing.
MAIN FLOW	The alternative flow starts at stage 6 of the main flow. 1) The Website reloads the form (repopulating submitted form elements) with an alert detailing that certain required input fields were submitted without a value. 2) The user fills in the missing values and clicks submit.
POST CONDITIONS	An account has been registered. It must now be verified by the user and approved by the administrator.
ALTERNATIVE FLOW/S	N/A

USE CASE	**UnregisteredUserRegisters:** EmailAddressRegistered
ID	WUC1.3
BRIEF DESCRIPTION	Registration fails as the email provided has already been registered.
PRIMARY ACTORS	Website
SECONDARY ACTORS	Unregistered User
PRECONDITIONS	An Unregistered User has submitted the registration form with an email that has already been registered.
MAIN FLOW	The alternative flow starts at stage 6 of the main flow. 1) The Website reloads the form (repopulating submitted form elements) with an alert detailing the email address supplied is already in use. 2) The user changes the email supplied and clicks submit.
POST CONDITIONS	An account has been registered. It must now be verified by the user and approved by the administrator.
ALTERNATIVE FLOW/S	N/A

USE CASE	**UnregisteredUserRegisters:** ImageCaptchaWrong
ID	WUC1.4
BRIEF DESCRIPTION	Registration fails as the image CAPTCHA value has been wrongly typed.
PRIMARY ACTORS	Website
SECONDARY ACTORS	Unregistered User
PRECONDITIONS	The Image CAPTCHA value has been wrongly submitted.
MAIN FLOW	The alternative flow starts at stage 6 of the main flow. 1) The Website reloads the form (repopulating submitted form elements) with an alert detailing the image CAPTCHA typed value was incorrect. 2) The user submits a new correct value and clicks submit.
POST CONDITIONS	An account has been registered. It must now be verified by the user and approved by the administrator.
ALTERNATIVE FLOW/S	N/A

USE CASE	**RegisteredUserLogsIn**
ID	WUC2.1
BRIEF DESCRIPTION	A User provides credentials and logs in to their user account.
PRIMARY ACTORS	User
SECONDARY ACTORS	Website
PRECONDITIONS	The User has an account to log into.
MAIN FLOW	The use case starts when the User selects the “log-In” button which is present on every page in the club section of the website. 1) The Website displays the login form. 2) The User provides a username and password, and clicks submit. 3) The Website checks the credentials provided. 4) The Website logs the user into their account.
POST CONDITIONS	The User gains access to their user account.
ALTERNATIVE FLOW/S	LogInDisallowed, CredentialsWrong

USE CASE	**RegisteredUserLogsIn:** LogInDisallowed
ID	WUC2.2
BRIEF DESCRIPTION	The login process fails and the user sees a message as to why.
PRIMARY ACTORS	Website
SECONDARY ACTORS	User
PRECONDITIONS	The user has registered an account. The account has not been activated (via email) or not been approved by the admin.
MAIN FLOW	The alternate flow starts after stage 3 of main flow. 1) The Website displays an alert detailing the reason that log-in was unsuccessful.
POST CONDITIONS	The user is not logged into an account and is left at the login page to attempt to log in again (restarts Use Case: RegisteredUserLogsIn).
ALTERNATIVE FLOW/S	N/A

USE CASE	**RegisteredUserLogsIn:** CredentialsWrong
ID	WUC2.3
BRIEF DESCRIPTION	The login process fails as the wrong credentials have been provided.
PRIMARY ACTORS	Website
SECONDARY ACTORS	User
PRECONDITIONS	The user has registered an account.
MAIN FLOW	The alternate flow starts after stage 3 of main flow. 1) The Website displays an alert detailing that the wrong credentials had been

	provided.
POST CONDITIONS	The user is not logged into an account and is left at the login page to attempt to log in again (restarts Use Case: RegisteredUserLogsIn).
ALTERNATIVE FLOW/S	N/A

USE CASE	**PromoterEditsClubDetails**
ID	WUC3.1
BRIEF DESCRIPTION	A Promoter edits stored details on their club.
PRIMARY ACTORS	Promoter
SECONDARY ACTORS	None
PRECONDITIONS	The promoter is successfully logged in.
MAIN FLOW	The use case starts when the Promoter clicks “Edit Club Details” from the member’s home page. 1) The Website displays the editable details about a club. 2) The Promoter alters the fields they desire to change. 3) The Promoter clicks submit. 4) The Website saves the changes.
POST CONDITIONS	The saved details on the club have been altered.
ALTERNATIVE FLOW/S	DetailsBlank

USE CASE	**PromoterUploadsClubPhoto**
ID	WUC4.1
BRIEF DESCRIPTION	A Promoter uploads a photo of a club.
PRIMARY ACTORS	Promoter
SECONDARY ACTORS	None
PRECONDITIONS	The promoter is successfully logged in.
MAIN FLOW	The use case starts when the Promoter clicks “Upload a Picture of your club” from the member’s home page. 1) The Website displays the image upload form. 2) The Promoter clicks Choose File. 3) The Website launches a file browser. 4) The Promoter browses their files until a suitable one is found and selected. 5) The Website closes the file browser. 6) The Promoter clicks submit. 7) The Website receives the file.
POST CONDITIONS	A new profile picture of a club has been uploaded.
ALTERNATIVE FLOW/S	InvalidFileType

USE CASE	**PromoterUploadsClubPhoto:** InvalidFileType
ID	WUC4.2
BRIEF DESCRIPTION	A file of invalid type has been uploaded.
PRIMARY ACTORS	Website
SECONDARY ACTORS	None
PRECONDITIONS	None
MAIN FLOW	The alternative flow starts after stage 6 of the main flow. 1) The Website displays an alert stating the file types that can be uploaded.
POST CONDITIONS	None
ALTERNATIVE FLOW/S	N/A

USE CASE	**PromoterFlagsReviews**
ID	WUC6

BRIEF DESCRIPTION	A Promoter flags a review submitted about their club.
PRIMARY ACTORS	Promoter
SECONDARY ACTORS	None
PRECONDITIONS	A review has been placed about the Promoter's club. The promoter is successfully logged in.
MAIN FLOW	**Segment Review deemed as inappropriate** 1. The Promoter clicks on flag review. 2. The System flags the review for moderation, and displays a new page stating the review has been successfully flagged.
POST CONDITIONS	A comment has been flagged for moderation.
ALTERNATIVE FLOW/S	N/A

USE CASE	**PromoterAddsAnAdvert**
ID	WUC9.1
BRIEF DESCRIPTION	A Promoter adds a new advert to the Website.
PRIMARY ACTORS	Promoter
SECONDARY ACTORS	None
PRECONDITIONS	The promoter is successfully logged in.
MAIN FLOW	The use case starts when the Promoter is on the Advert management table. 1) The Promoter clicks "Add an Advert". 2) The Website displays the 'Add Advert' input form. 3) The Promoter fills in the required details. 4) The Promoter clicks "submit". 5) The Website stores the advert and reports back a successful submission to the Promoter.
POST CONDITIONS	A new advert has been stored in the Website.
ALTERNATIVE FLOW/S	AllFieldsNotFilled

USE CASE	**PromoterEditsAdvertDetails**
ID	WUC11.1
BRIEF DESCRIPTION	A Promoter edits advert details
PRIMARY ACTORS	Promoter
SECONDARY ACTORS	None
PRECONDITIONS	An advert has already been input by the promoter. The promoter is successfully logged in.
MAIN FLOW	The use case starts when the Promoter is on the Advert management table. 1) The Promoter clicks "Edit" 2) The Website displays the "Edit Advert" form populated with the current values. 3) The Promoter alters the required information. 4) If Promoter wishes to alter 'active' status: Promoter changes active status of advert. 4) The Promoter submits the form. 5) The Website stores the advert and reports back a successful submission to the Promoter.
POST CONDITIONS	The adverts details have been edited. The active status of the advert may have changed.
ALTERNATIVE FLOW/S	AllFieldsNotFilled

USE CASE	**PromoterEditsAdvertDetails:** AllFieldsNotFilled
ID	WUC11.2
BRIEF DESCRIPTION	Submission is not successful as all fields have not been filled in.
PRIMARY ACTORS	Website

SECONDARY ACTORS	Promoter
PRECONDITIONS	An advert has already been input by the promoter. The promoter is successfully logged in.
MAIN FLOW	The alternative flow starts after stage 4 of the main flow. 1) The Website displays an alert that not all required fields have been filled in. 2) The Promoter fills in the remaining empty fields, and submits the form. 3) The Website stores the advert and reports back a successful submission to the Promoter.
POST CONDITIONS	The adverts details have been edited.
ALTERNATIVE FLOW/S	N/A

USE CASE	**PromoterPushesAdvertAsNotification**
ID	WUC12
BRIEF DESCRIPTION	A Promoter initiates a push of an advert to targeted user's phones.
PRIMARY ACTORS	Promoter
SECONDARY ACTORS	Application
PRECONDITIONS	An advert has already been input by the promoter. The promoter is successfully logged in.
MAIN FLOW	The use case starts when the Promoter is on the Advert management table and clicks "Push". 1) The Website displays a summary of the advert and targeted information. 2) The Promoter selects how they want to target users. 3) The Promoter clicks "push". 4) The Website pushes the selected advert to the targeted users.
POST CONDITIONS	An Advert has been pushed to the target user's devices.
ALTERNATIVE FLOW/S	N/A

USE CASE	**AdminReviewsNewRegistration**
ID	WUC13
BRIEF DESCRIPTION	An Admin reviews a new account registration.
PRIMARY ACTORS	Admin
SECONDARY ACTORS	None
PRECONDITIONS	The Admin is successfully logged in. There are registrations to be reviewed.
MAIN FLOW	The use case starts when the Admin is on their dashboard. 1) The Admin clicks "Accounts Pending Approval". 2) The Website displays all the new accounts that are pending approval. 3) The Admin views the details of a new account. 4.1) If the Admin wishes to approve an account: Admin clicks 'Approve'. 4.1) If the Admin wishes to deny an account: Admin clicks 'Deny'. 5) The system removes the account from the 'Accounts Pending Approval list'.
POST CONDITIONS	An account has been reviewed.
ALTERNATIVE FLOW/S	N/A

To conserve space, the following website Use Case descriptions were omitted from the final documentation:

- WUC3.2: PromoterEditsClubDetails: *DetailsBlank*
- WUC5: PromoterViewsReviews
- WUC7: PromoterContactsSupport
- WUC8: PromoterDeletesAnAdvert
- WUC9.1: PromoterAddsAnAdvert: *AllFieldsNotFilled*
- WUC10: PromoterViewsAdvertDetails
- WUC14: AdminReviewsFlaggedComment
- WUC15: StudentDownloadsApplication

Appendix C

System Overview/Class Listings

1 Application

Class Name	Description
AddReviewActivity	Allows the user to submit a review/rating of a certain club to the server.
AdvertCursorAdapter	Formats data it receives into the appropriate UI elements so that a ListView (that is held in the AdvertFrag class) is displayed properly.
AdvertFrag	Holds the ListView that displays adverts. It sets up the ListView; to do this it initiates a Loader that returns a cursor that holds data matching the user's criteria. This is then passed to the AdvertCursorAdapter, so it can be displayed properly as a list.
AdvertListings	Provides a list of adverts that match the input criteria (input in the GoClubHome activity). It contains the AdvertFrag class which holds the adverts.
AdvertListProvider	A content provider that allows Android Loaders to query the advert table of the database.
AdvertView	Displays details of an individual advert that the user has selected from the AdvertListings. It also allows users to post a message to Facebook stating that they are visiting a certain club.
ARBrowserActivity	Sets up and displays the Augmented Reality view. It provides the Wikitude SDK with the data that is required and also handles what to do when items in the Augmented Reality view are clicked.
DBHelper	Manages the database. It provides methods to either create or upgrade the database.
GCMIntentService	Manages events related to the notification capabilities of the application. It provides methods to deal with newly received notifications as well as dealing with successful registration.
GoClubApplication	Stores globally accessible variables and methods, such as the methods that are used throughout the system to connect to the back-end server.
GoClubHome	Main page of the application. All functionality can be reached from here. It allows the user to conduct a search for adverts (providing their criteria to its dropdown boxes) or to start the Augmented Reality View.
GoClubItemizedOverlay	Custom overlay for the MapView. It allows a marker to be placed on screen and provides a method to execute when the marker is pressed.
LocationGiver	Service that checks the user's location, and if it has changed, reports it to the back-end webserver. This service is used so that the server knows where the user is and can send notifications depending on the user's location. The service schedules itself to run every 15 minutes between 6pm and 2am.
PoiBean (stands for 'Point of interest' bean)	Data is passed from the application to the Augmented Reality view as a PoiBean. This class contains data on each marker for the Augmented Reality view to display; this includes, name, description, clubid and location. A list of JSON representations of PoiBeans are passed as to the Augmented Reality view.
PoiDetailActivity	Viewed by the user when they have clicked for more information on a certain club when in the Augmented Reality view. It allows the user to either navigate to the ReviewActivity or AdvertListings.
RedeemActivity	Provides the map with generated route to the selected club. It also checks whether the user has 'reached' the club.

RegisterActivity	Displays the registration form. It also allows connections to Facebook to retrieve data to auto populate the form.
RegisterationProcess	Handles registration of the device with Google Cloud Messaging (GCM) and the back-end server. It is aided by the ServerUtilities class (see below).
ReviewActivity	Provides a list of reviews submitted about the selected club. It contains the ReviewFrag class which holds the reviews.
ReviewCursorAdapter	Formats data it receives into the appropriate UI elements so that a ListView (that is held in the ReviewFrag class) is displayed properly.
ReviewFrag	Holds the ListView that displays reviews. It sets up the ListView; to do this it initiates a Loader that returns a cursor that holds data matching the user's criteria. This is then passed to the ReviewCursorAdapter, so it can be displayed properly as a list.
ReviewListProvider	Content provider that allows Android Loaders to query the review table of the database.
RouteOverlay	A class that allows a straight line to be drawn from one point to another, and then displayed over the MapView. This is used so that joining the straight lines generated form a route for the user.
ServerUtilities	A static class that helps integrate the application with Google Cloud Messaging. Provides method to register the device with the back-end server.
SplashScreen	Specified as the class to open when the application is launched. It determines whether the registration process has been completed or not, directing to the home screen (GoClubHome) or the registration screen (RegisterActivity) respectively.
UpdateDB	A service that manages the updating of the database. It retrieves data from the webserver and stores this data to the applications database. Executed every time the application is opened onto the home screen (GoClubHome).

2 Website Scripts

Name	**Description** - *All methods return results as JSON arrays.*
GCM/Register	Used as part of the registration process that registers a phone to allow for notifications via the Google Cloud Messaging (GCM) service. Takes the provided registration values (user name, university and GCM Reg ID) and stores these to the *gcm_users* table.
GenerateAdvertListing	Queries the database and returns details of all active adverts to the application.
GenerateClubDB	Queries the database and returns details of all clubs to the application.
GenerateClubReview	Queries the database and returns all un-flagged reviews about a certain club.
GenerateClubRating	Queries the database and returns a clubs' average rating.
GetValidOn	Queries the database and returns details of what day/s each advert is valid on.
SetLocation	Used as part of the Notification functionality. Takes the user's location and initiates the delivery of notifications for clubs that are nearby the user.
SubmitClubReview	Allows review submissions made on the application to be saved. Receives POST variables from user submitted data and stores these to the review table.
SubmitStatistic	Allows the application to report to the server when certain user events occur. These include when a certain advert is viewed, redeemed and when the user reaches the club claiming a certain deal.

Appendix D

Testing Reports/Results

1 Application testing

1.1 Monkey Stress Testing Results

1.2 Manual Application Functional Testing

1.3 Robotium Test cases

2 Website testing

2.1 Selenium Functional Testing

2.2 Manual Website Functional Testing

1.1 Monkey Stress Testing Results

Each run injected 500 different events and halted if an exception is produced from its actions. The Monkey runner was executed 5 times on each activity, so 2500 random events where generated for each starting activity. Although Monkey generates random events, it starts with a seed and if this is the same, the random events produced are the same, so for each run the seed was changed.

Below details errors found. Each row displays a different exception. If one of the test runs generates an exception, it is rerun (by supplying the same seed number) after a fix has been implemented. If the re-run is successful and no exception is found (so the problem has been fixed), this is noted in the final column.

Test started on:	Seed	Event /500	Exception	Cause	Fix	Fixed Rerun (Y/N)
GoClub Home	0	88	RuntimeException: Unable to start activity SimpleARBrowserActivity	Requesting the SimpleARBrowserActivity while the database is loading will cause an unhandled exception to be thrown.	Clicking to search, now checks that the database has fully loaded before Augmented Reality can be opened, telling the user to wait if it is not loaded.	Y
GoClub Home	1	100	NullPointerException	Clicking to search the database before it had populated (on the case that it was initially empty) resulted in an exception. This was never realised during manual testing as the short delay it took a user to fill the form, gave the database time to load.	Clicking to search, now checks that the database has fully loaded before a search can be conducted, telling the user to wait if it is not loaded.	Y
Registration Activity	0	397	BadTokenException	The Monkey is unable to move back to the registration pages after registration is complete. However, this is desired for the user as registration should not be accessible once complete.	None required.	N/A
Advert Listings	0	154	illegalStateException: Cannot start AsyncTask: the task is already running.	Requesting the RedeemActivity(Map activity) and then moving back to the AdvertView with speed will throw an exception, as the onResume method attempts to execute a task that is already running.	Added an additional check using getStatus() to check the status of the AsyncTask, only if it has finished is it called again.	Y

1.2 Manual Application Functional Testing

Note: Only the test cases that had an **unexpected outcome** have been included here. The full Manual Application Testing Document can be found on the attached CD.

Test ID	Test Description	Req/s tested	Expected	Outcome, if different to expected.
Category: Advert Details Page				
TA11	1) Click on the advert titled *'Totally New Advert Title'*	FRA2.7 FRA3.1 FRA3.2 FRA3.4 FRA3.5 FRA3.6	1) The club details are: Name: *Test Club Name/ Test Club Name2* Address: *Test Address, Test City, PostCode2* 2) Rating value has been loaded. 3) An image is loaded. 4) The details entered previously are shown. 5) An entry price should be shown.	Image not loading. On inspection, if an image has a space in its file name android cannot load it successfully. The way images are stored has now been changed so that filenames are generated numbers and so do not have possible gaps in their filename.
TA13	1) Navigate back to the dedicated page for '*Totally New Advert Title'*. 2) Click 'View reviews/ratings'.	FRA3.7	Previously submitted reviews/ratings (submitted in website functional testing) should be displayed.	Navigating to the reviews/ratings activity and then moving back instantly would cause an exception. This was as the asynctask would return and not have anywhere to notify. Now when the activity is moved out of, the asynctask is cancelled.
Category: Review System				
TA17	1) Go back to 'View Reviews/Ratings'.	FRA4.2 FRA4.3	1) Individual reviews of the club should be shown. 2) Individual star ratings should be shown. 3) Names of the user who submitted the review/rating should be shown.	Names were not being shown. This was a large-scale problem caused by a change in the package name (for the Wikitude Augmented Reality to work). Permissions had not been updated and user ID's were always null meaning the name of the user could not be given.
Category: Connecting to Facebook				
TA24	1) Go into the phones settings and wipe all data associated with the GoClub application. *(Settings → Applications → Manage → GoClub → Clear Data)* 2) Re-open the application, and view the registration page. 3) Click connect with Facebook. 4) Supply the wrong information. 5) Click to go back.	FRA6.1	No details should be filled in the name field, allowing the user to fill it manually.	Caused a RuntimeException: from a NullPointerException. This was a noted problem with the Facebook SDK. A fix was given here: http://bit.ly/15e3Zul

1.3 Robotium test cases

1. Creating the environment for the test cases

To set up the Robotium tests, all adverts have been set to inactive except:

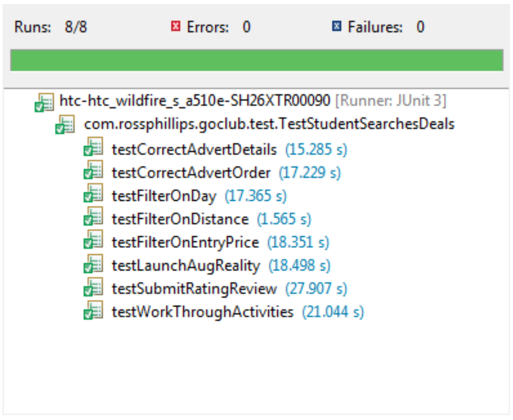
Advert by club with name: Test Club Name2
Title: Totally New Advert Title
Description: Totally New Advert Description
Entry: 3
Valid on: Tuesday

Advert by club with name: Another Club Name
Title: Another Great Advert Title
Description: Another Great Advert Description
Entry: 12
Valid on: Wednesday

2. Test cases

Test Case	Description
testFilterOnEntryPrice	This test checks that filtering search results by the entry price of an advert is successful.
testFilterOnDay	This test checks that filtering search results by the day an advert is valid on is successful.
testFilterOnDistance	This test checks that filtering search results by the distance an advert is from the user is successful.
testCorrectAdvertOrder	This test checks that adverts are displayed in the search results in the correct order.
testCorrectAdvertDetails	This test checks that the correct details of an advert are displayed.
testSubmitRatingReview	This test checks that reviews are submitted successfully.
testWorkThroughActivities	This test checks that all activities can be transitioned to successfully.
testLaunchAugReality	This test checks that augmented reality can be started successfully.

3. Test Results



```
Runs:  8/8          Errors:  0          Failures:  0

htc-htc_wildfire_s_a510e-SH26XTR00090 [Runner: JUnit 3]
  com.rossphillips.goclub.test.TestStudentSearchesDeals
    testCorrectAdvertDetails (15.285 s)
    testCorrectAdvertOrder (17.229 s)
    testFilterOnDay (17.365 s)
    testFilterOnDistance (1.565 s)
    testFilterOnEntryPrice (18.351 s)
    testLaunchAugReality (18.498 s)
    testSubmitRatingReview (27.907 s)
    testWorkThroughActivities (21.044 s)
```

2.1 Selenium Functional Testing

Test Case	Description
TestLoginClubCorrect	This checks that the login system is working. A club should be logged in when providing the correct credentials.
TestLoginAdminCorrect	This checks that the login system is working. An admin should be logged in when providing the correct credentials.
TestLoginWrongCredentials	This checks that errors in the login system are handled correctly. The login system should display an error when provided the wrong credentials.
TestLoginNotActivated	This checks that errors in the login system are handled correctly. The login system should display an error when a non-activated account attempts to login.
TestLoginNotApproved	This checks that errors in the login system are handled correctly. The login system should display an error when provided the wrong credentials.
TestLoginNoEmail	This checks that errors in the login system are handled correctly. The login system should display an error when no email is provided.
TestLoginNoPassword	This checks that errors in the login system are handled correctly. The login system should display an error when no password is provided.
TestEditClubNoDetails	This checks that errors when editing a clubs details are handled correctly. The system should display an error when there are missing fields.
TestEditClubSuccess	This checks that successfully editing club details results in the changes being stored successfully.
TestUploadInvalidFile	This checks that errors are handled correctly by the image upload system. The system should display an error when the file is of the wrong type.
TestUploadNoFile	This checks that errors are handled correctly by the image upload system. The system should display an error when the file is missing.
TestUploadSuccess	This checks the image upload system works correctly. The system should display that the image has been uploaded successfully.
TestAddNoDetails	This checks that errors are handled correctly when adding a new advert. The system should display an error when there are form fields missing values.
TestAddSuccess	This checks that adding an advert successfully will result in it being saved correctly.
TestAdvertView	This checks the correct information about an advert is displayed on the advert page.
TestEditMissingDetail	This checks that errors are handled correctly when editing an adverts details. The system should display an error when there are form fields missing values.
TestEditSuccess	This checks that editing an advert successfully will result in it being saved correctly.
TestPush	This checks that pushing an advert is handled corrected.
TestDeleteNoneSelected	This checks that errors are handled correctly when attempting to delete an advert. The system should notify the user that no image was selected.

2.2 Manual Website Functional Testing

Note: Only the test cases that produced **unexpected outcomes** have been included. In total 47 test cases were undertaken. The full document can be viewed on the CD.

Test ID	Test Description	Req/s tested	Expected	Outcome, if different to expected.
TW5	Attempt to register another account, using the email address *"ross.phillips@live".*	FRW3.5	The form should not submit when button pressed and a pop-up should appear instructing the user to enter a valid email address.	Although, there was client-side email validation, this is a weak solution: the example email was allowed to be registered. Server side validation has now been added.
Category: Uploading Images				
TW20	Attempt to upload a .jpeg under 2MB	FRW6.1	The system should report the file had been uploaded. Navigating back to the member's homepage should see the new picture.	The image on the member's home, doesn't always update straight away as a cached image may be shown. <META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE"> has been added inside the head tag of the page, however browsers vary on whether they take account of this.
TW25	1) Load up the application. 2) View the newly added advert. 3) Refresh: http://rossphillips.eu/goclub/members/manageads/	FRW7.3	"Total Views" has incremented by 1.	This was a problem with the application, a previous alteration to the code (from the Monkey testing) had resulted in some unreachable code. Now fixed and tested.
Category: Individual Advert page/ Statistics – *For these tests, ensure the device is <0.2km from the club Test Club Name*				
TW31	1) Open the application again, and work through the process to "view", "redeem" and then "claim" an the advert titled *Test Advert.* 2) On the website, Click "View" for advert with title *Test Advert.*	FRW10.1 FRW10.3 FRW9.2	Total Views should be 1. Total Redeems should be 1. Got to Club should be 1 (as the club's location was placed within 0.2km of your location in TW2) Advert preview should be displayed appropriately.	Problem with app submitting userID's meant statistics were not reaching the website. Fixed on the app.

Appendix E

Systems Manual

1. Accessing the code

The final project code (as well as all previous revisions) is stored on two BitBucket repositories.
One repository holds the **application** code:
https://bitbucket.org/rossphillips/comp-3091-project-android-app
One repository holds the **back-end server** code:
https://bitbucket.org/rossphillips/comp3091-project-website

To gain access to the repositories, permission has to be granted. To be granted permission, email: rossphillips25@live.co.uk

2. Setting up the code

2.1 Setting up application code

The steps below can be followed to set up the application code:

1. Import the project into the Eclipse IDE by selecting Import "*Existing Android Code into Workspace".*
2. Ensure that all dependencies have been successfully imported.
2.1. In the package explorer, click 'Android Dependencies'. The following .jar files should be displayed: 1) Facebook.jar 2) GCM.jar 3) Javadoc.jar 4) Android-support-v4.jar 5) Wikitudesdk.jar
2.2. Package explorer > 'Referenced Libraries'. Should contain Maps.jar
If any of these are not displayed, they will have to be downloaded from the internet and added to the project.
3. Right click on the project then click *Run/As Android Application*.
4. Launch the application on either a device or emulator.

The application should now run successfully.

2.2 Setting up Website code

The steps below can be followed to set up the website code:

1. Firstly, you will need a webhost that supports PHP/MySQL. The current provider is HostGator (http://www.hostgator.com/)
2. In PHPMyAdmin, enter the contents of the file *GoClubDBBuild.sql* as an SQL statement. This will build the database for you.
3. Create a new user that can access the database.
4. Edit the connect.inc.php (*goclub/includes/connect.inc.php*) file to include the *username* and *password* of the newly created user.
5. Use an FTP program to upload all the files to the *public_html* folder of the webserver.

The website is now set up. Make any further changes in a text editor of your choice and use the FTP program to upload them to the webserver.

Appendix F

Application User Manual

1. Setting up the application

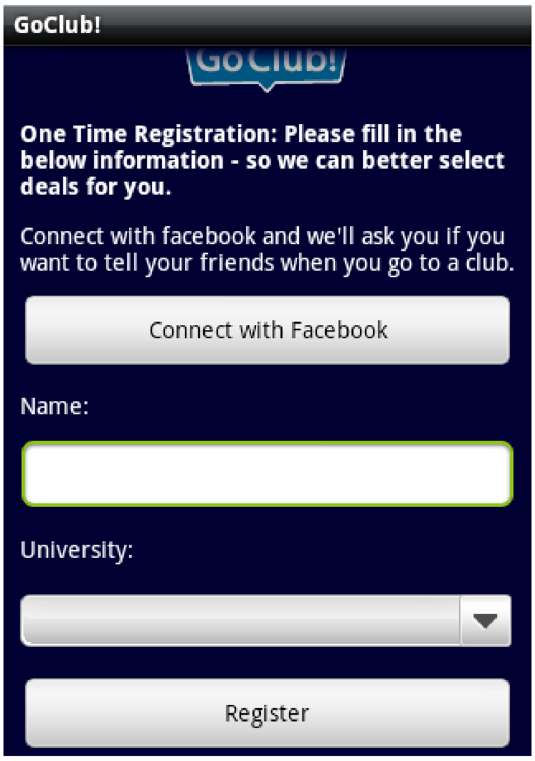
1.1 Downloading and installing the application.

1) On your Android device, go into the settings and check "Allow Unknown Sources".
Depending on the device, the location of this may vary, but it is likely to be similar to:
Settings / Applications / Unknown Sources
2) Navigate in your device's web browser to http://rossphillips.eu/goclub/
3) Click the Download button and save the .APK file.
4) Click the newly downloaded item in your download list.
5) Accept the privileges and click install.

The application has been installed.

2. Key Tasks

2.1 Registering your application


GoClub!

One Time Registration: Please fill in the below information - so we can better select deals for you.

Connect with facebook and we'll ask you if you want to tell your friends when you go to a club.

Figure 1.1 Registration form

Completing the registration process will allow notifications of special local deals to be pushed to your phone; these will appear in the notification bar and clicking them will open the application to view more information.

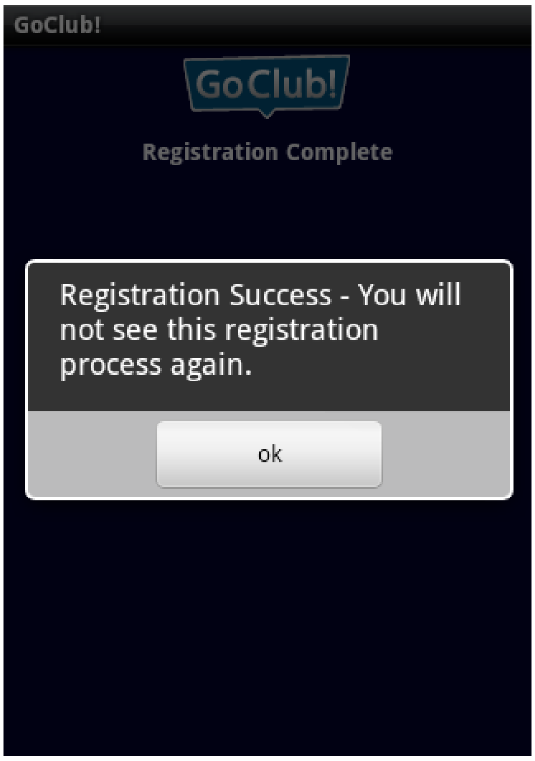
Note: Notifications can be turned off (See Key Task 2.8: Turning off notifications).

The registration process will only show the first time the application is run.

Steps:

1) Enter your full name or select 'Connect With Facebook'.

1.1) If 'Connect With Facebook' is selected follow the authentication steps 4.2 and 4.3 detailed in key task 2.4. On a successful connection, your name will be filled into the form field.


GoClub!

Registration Complete

Registration Success - You will not see this registration process again.

Figure 1.2 Registration Success

2) Select the 'University' dropdown box and select your university from the list.

3) Click 'Register'.

4) The application will report back on the status of your registration; If the registration fails, details are provided explaining the reason.

5) Click 'Ok'

You have now registered your application. The registration process will not show again.

2.2 Selecting search criteria

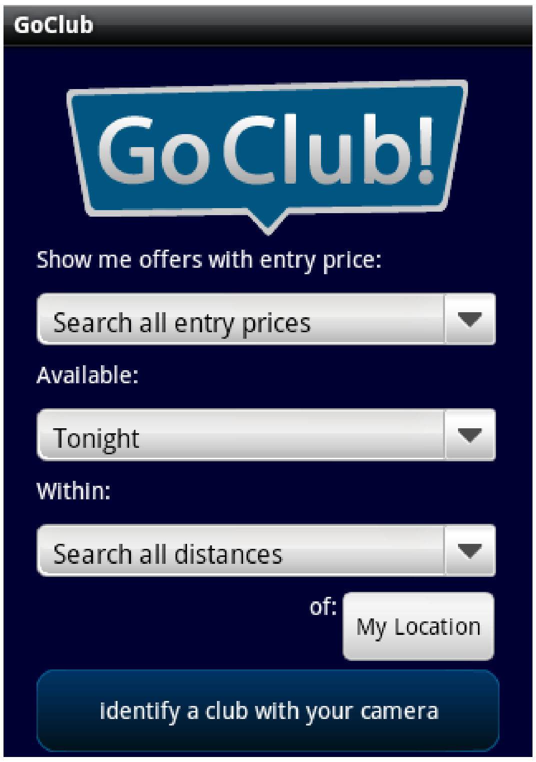


Figure 2.1 Main Page

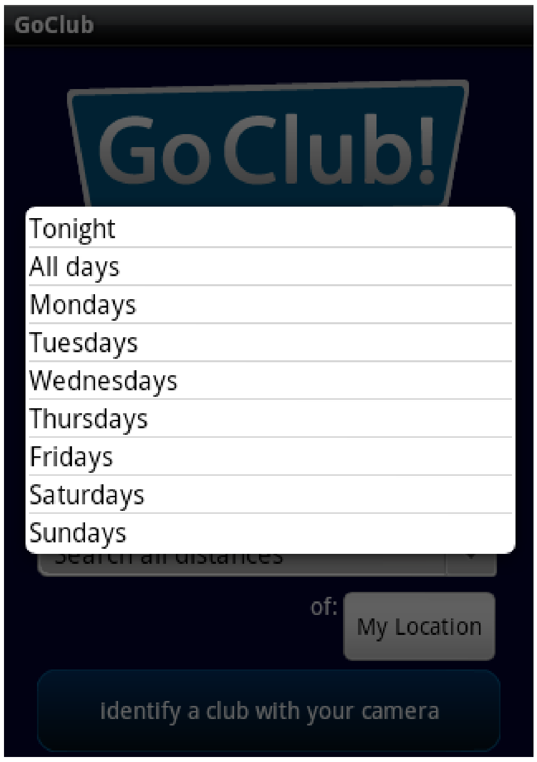


Figure 2.2 Drop down list

Figure 2.1 shows the main screen that is present every time the application is opened (now that registration has been completed). From here, all the functionality that the application provides can be accessed.

Below details how to search for different adverts.

Steps:

1) The page allows the option to search on **price**, **day** of event, and **distance** from your location. Each dropdown can be used to customize your search criteria.

2) Entry prices can be searched by clicking the first drop down box. The default selected value will search all entry prices.

3) Day of event can be searched by clicking the second drop down box; the default selected value will be 'tonight'. Figure 2.2 shows the options available when this drop down is selected.

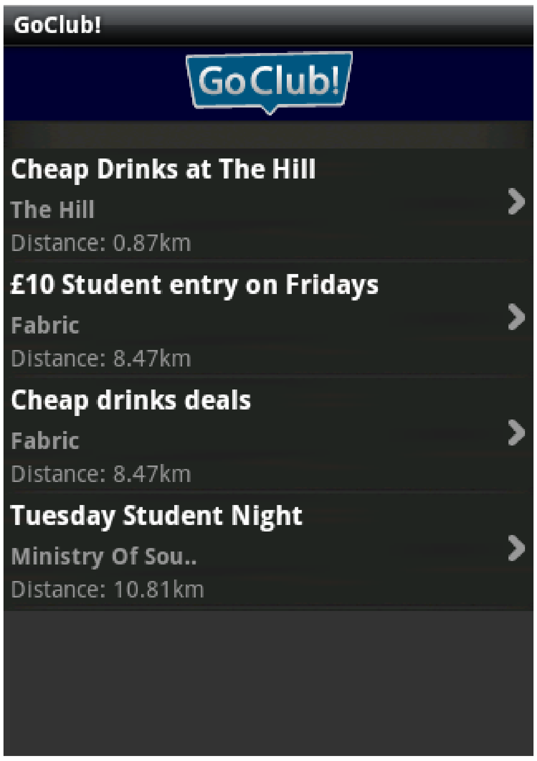
As well as searching for deals tonight, you can search for deals available on all days or pick a specific day and search for deals for that day only.

4) In figure 2.1, we can see the third dropdown box allows distances to be filtered. You can select with what proximity from your location to search.

5) Click 'My Location' to search.

6) Figure 2.3 shows the search results.

P.T.O


GoClub!

Cheap Drinks at The Hill
The Hill
Distance: 0.87km

£10 Student entry on Fridays
Fabric
Distance: 8.47km

Cheap drinks deals
Fabric
Distance: 8.47km

Tuesday Student Night
Ministry Of Sou..
Distance: 10.81km

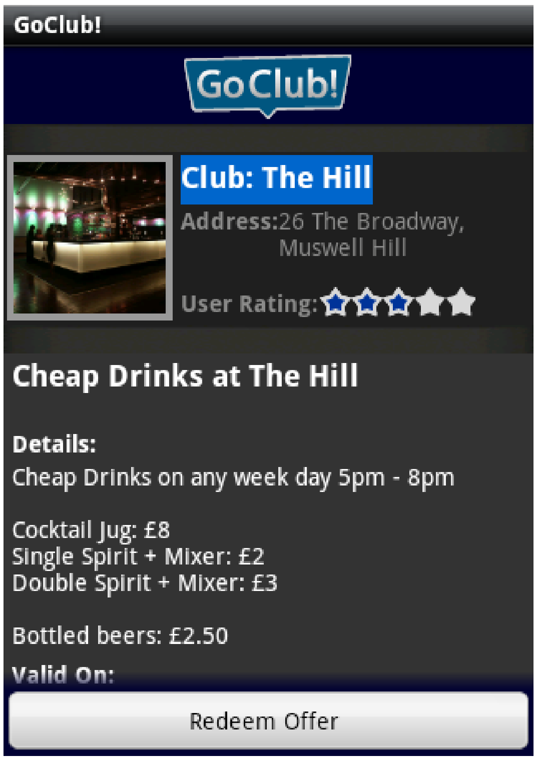
Figure 2.3 Search Results

6.1) The search results are sorted so that adverts for nearer clubs are shown first with the clubs getting further away.

The search results display the title of the advert, the club posting the advert and the distance from the user's location.

You have now searched for deals matching your desired criteria.

2.3 Viewing advert details and going to a club


GoClub!

Club: The Hill
Address:26 The Broadway, Muswell Hill
User Rating:

Cheap Drinks at The Hill

Details:
Cheap Drinks on any week day 5pm - 8pm

Cocktail Jug: £8
Single Spirit + Mixer: £2
Double Spirit + Mixer: £3

Bottled beers: £2.50
Valid On:

Redeem Offer

Figure 3.1 Advert Details

Below details the process of viewing more details of an advert and going to the club.

Steps:

1) The advert details page shows a number of details on a selected advert, including:

- The club name and address.

- The club's user rating.

- The club's profile picture.

- The advert title.

- Details of the advert

- Days the offer is valid on

- Entry price

2) Once you have decided that you want to go to a

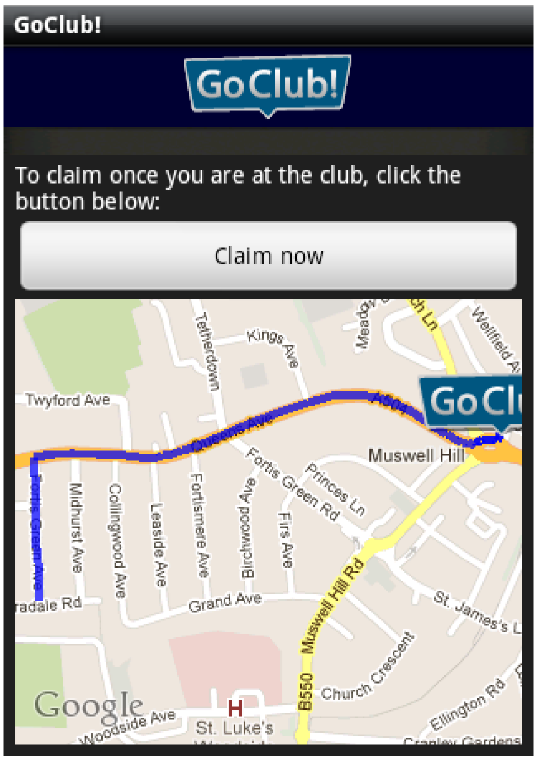


Figure 3.2 Map to club

certain club click "Redeem Offer"

3) You will be asked whether you want to connect to Facebook or not, in this case, click 'No' (Connecting to Facebook will be covered later).

4) An interactive map will show with a generated route from your current location to the club.

4.1) Clicking the clubs marker will display the address of the club, in case you are getting a taxi.

5) Once you are at the club, click "Claim now", the details of the offer that you are claiming will display on screen for the cashier to read.

You have now viewed, redeemed and claimed an offer.

2.4 Posting to Facebook

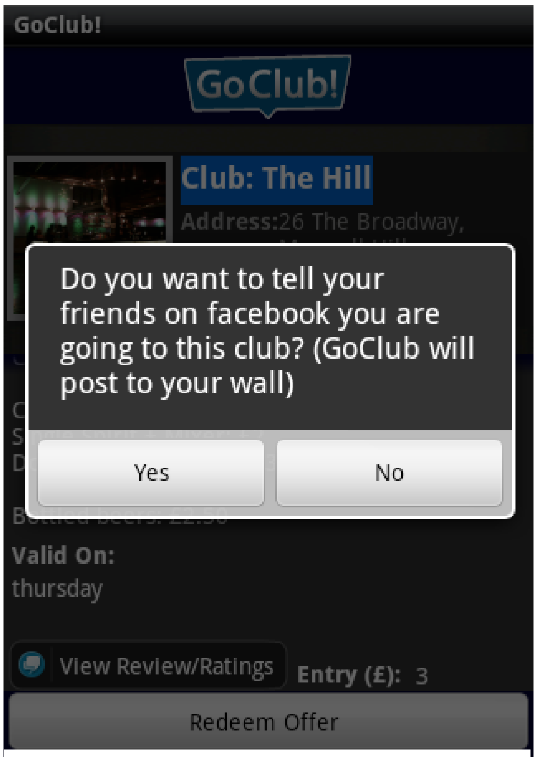

Do you want to tell your friends on facebook you are going to this club? (GoClub will post to your wall)

Figure 4.1 Permission request

The application allows you to post a message to your Facebook feed stating you are going to a certain club.

Steps:

1) Once you have decided you want to go to a club, click Redeem (like in key task 2.3). When the alert in Figure 4.1 is shown, click 'Yes' to posting to Facebook.

P.T.O

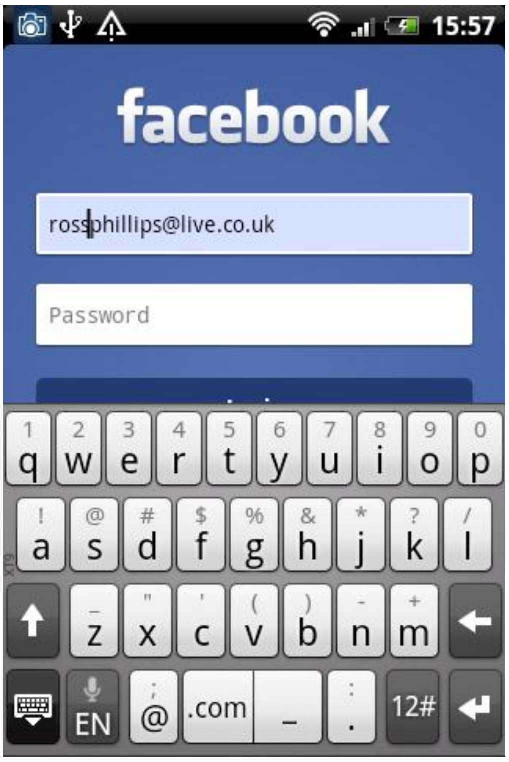
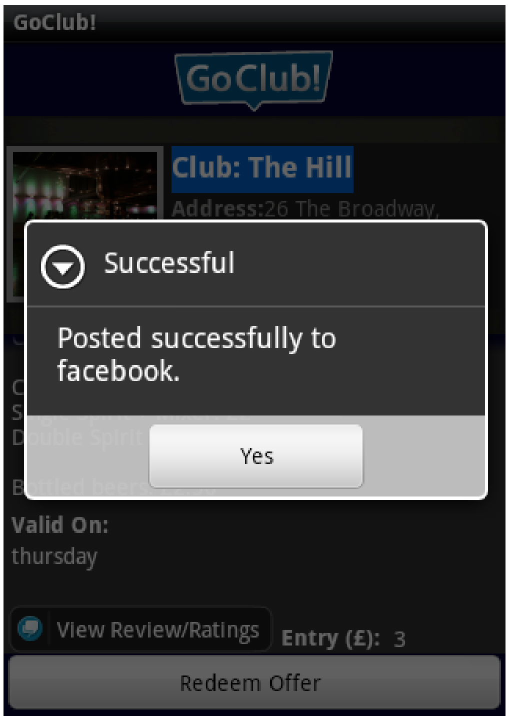


Figure 4.2 Facebook Authentication



Successful

Posted successfully to facebook.

Yes

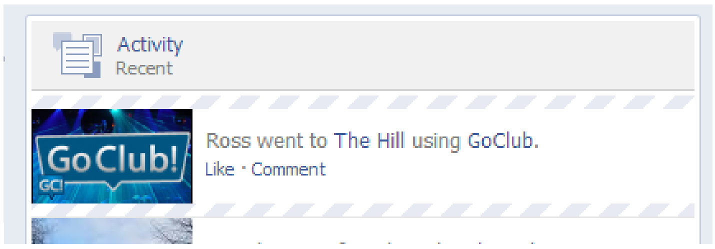
Figure 4.3 Success confirmation

2) If your mobile device is not already linked with Facebook, the authentication screen shown in Figure 4.2 will display; provide your login details.

3) Facebook will check the credentials. Assuming these are correct, GoClub will attempt to post the message to your Facebook page.

4) If the post is made successfully, you will see the alert in figure 4.3.

A post following the below format has been made to your Facebook page.



Ross went to The Hill using GoClub.

Figure 4.4 Facebook Post

2.5 Viewing reviews of a club

The application allows you to view reviews and ratings submitted by other users about a certain club.

Below details the process to do so:

Steps

1) When you are viewing an individual advert, click the "View Reviews/Ratings" button shown in figure 5.1

Note: this may require scrolling to the bottom of an advert if the advert cannot fit onto the page without scrolling.

2) A reviews page similar to that in figure 5.2 will show. This displays:

- The overall club rating (an average of all the individual ratings).

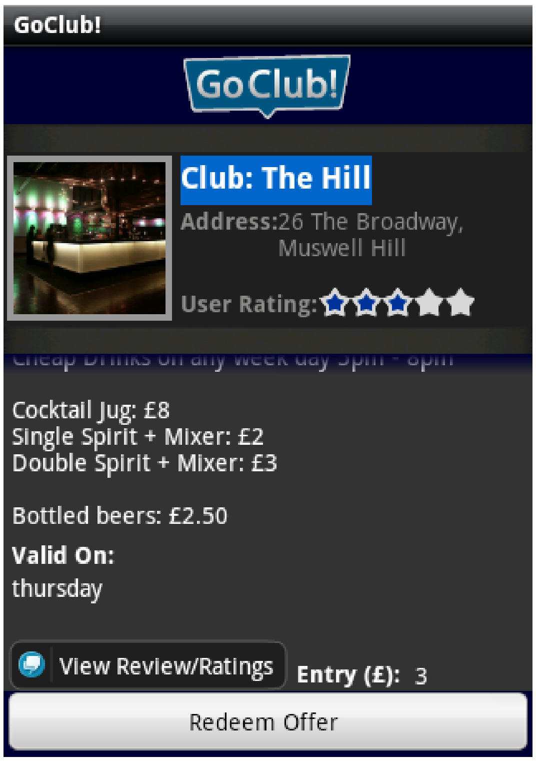
- Individual reviews, including:

  - Individual ratings

  - The user who submitted the review.

  - The date the review was submitted.

You have now viewed the reviews of a club.



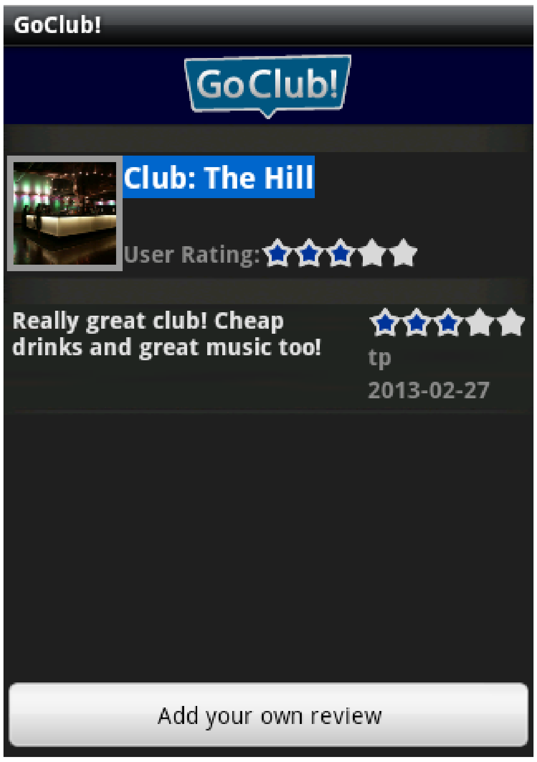
Cheap Drinks on any week day 5pm - 8pm

Cocktail Jug: £8
Single Spirit + Mixer: £2
Double Spirit + Mixer: £3

Bottled beers: £2.50

Valid On:
thursday

Figure 5.1 Scrolling down an advert



Really great club! Cheap drinks and great music too!
tp
2013-02-27

Figure 5.2 Reviews page

2.6 Adding a review of a club

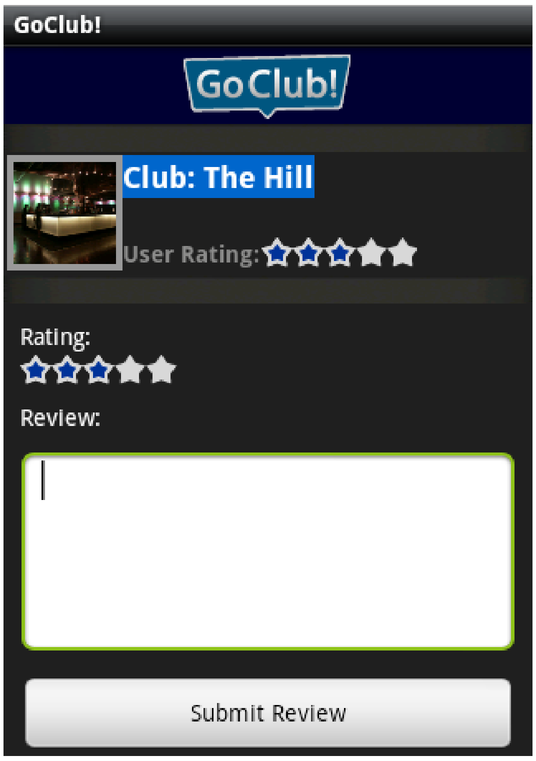


Figure 6.1 Adding a review

The application allows you to add your own review and rating of a certain club.

Steps

1) When viewing the reviews of a club, click "Add your own review" (shown in Figure 5.2).

2) The page in Figure 6.1 will display.

3) Select a rating, by touching the number of stars you want to award the club.

4) Input your review.

5) Click 'Submit Review'

6) The application will report back whether the submission was a success or failure.

You have now submitted a review.

2.7 Identifying a club using your camera

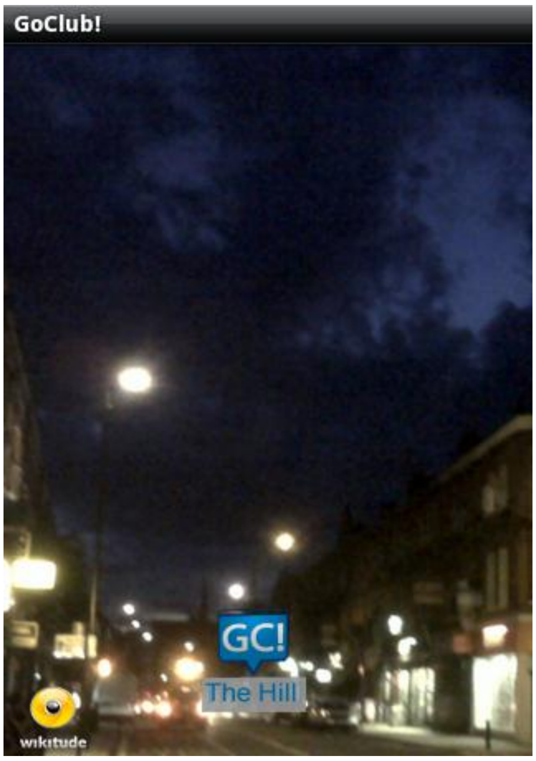


Figure 7.1 Augmented Reality

The application allows the user to identify the location of near-by clubs using your camera view.

Steps

1) Click 'Identify a club with your camera' from the home page (shown in figure 2.1).

2) The application will check your devices technical capabilities and display a message if your device cannot handle this function of the app.

3) Assuming your phone can handle this function, the device will start displaying images retrieved from the camera.

4) Clubs within 1 kilometre from your location will have a marker placed over their location. This allows you to direct yourself to a certain club.

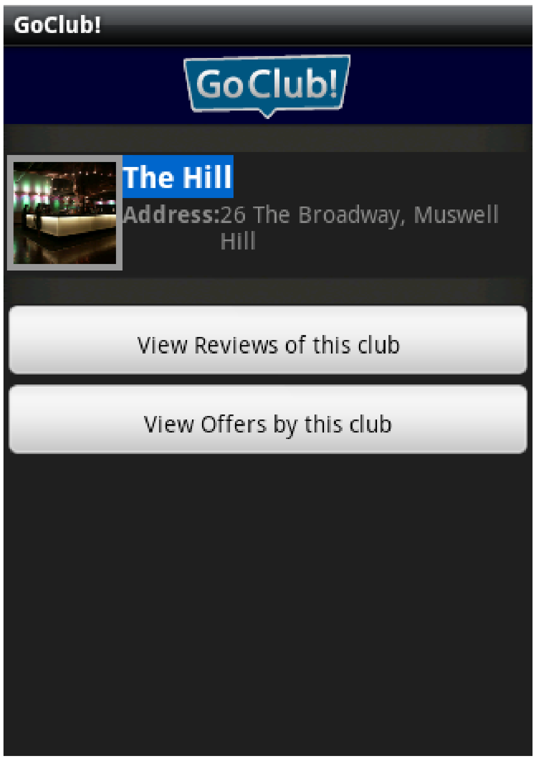

GoClub!
GoClub!
The Hill
Address:26 The Broadway, Muswell Hill
View Reviews of this club
View Offers by this club

Figure 7.2 Further Club details

5) Click on a marker and then 'more information' will display on the screen showed in figure 7.2.

6) From here you can view the reviews of the club (see 2.5 for more details on viewing reviews.

You can also view the offers available at the selected club tonight.

You have now identified a club using your camera.

2.8 Turning off notifications

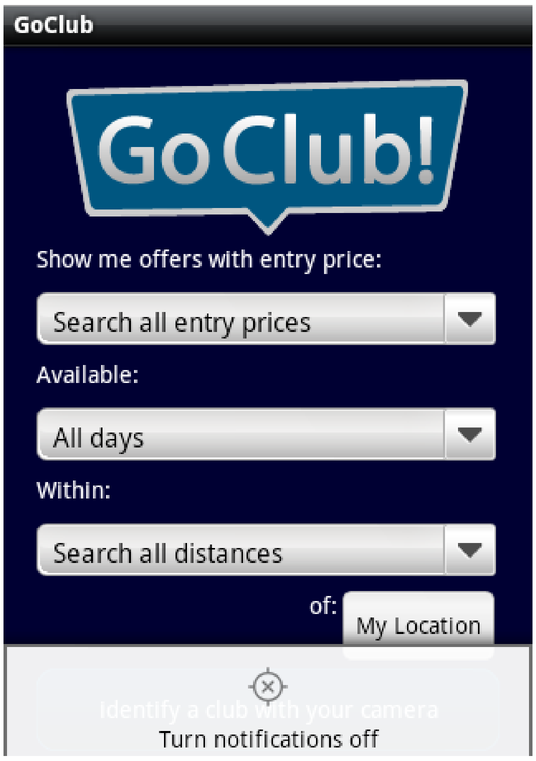


Figure 8.1 Notifications menu

Occasionally, your phone may receive notifications about extra special deals that are available in a close proximity to your location.

If you would rather not receive this service, you are able to opt out of notifications.

Steps:

1) Open up the application.

2) Press the menu key on your Android device.

3) A 'Turn notifications off' button will display (as shown in figure 8.1).

4) Click this button.

You have now turned notifications off.

Note: To turn notifications back on, follow the same process, the button will display 'Turn notifications on'.

Appendix G

Website User Manual

Website User Manual

1. Club Promoters:

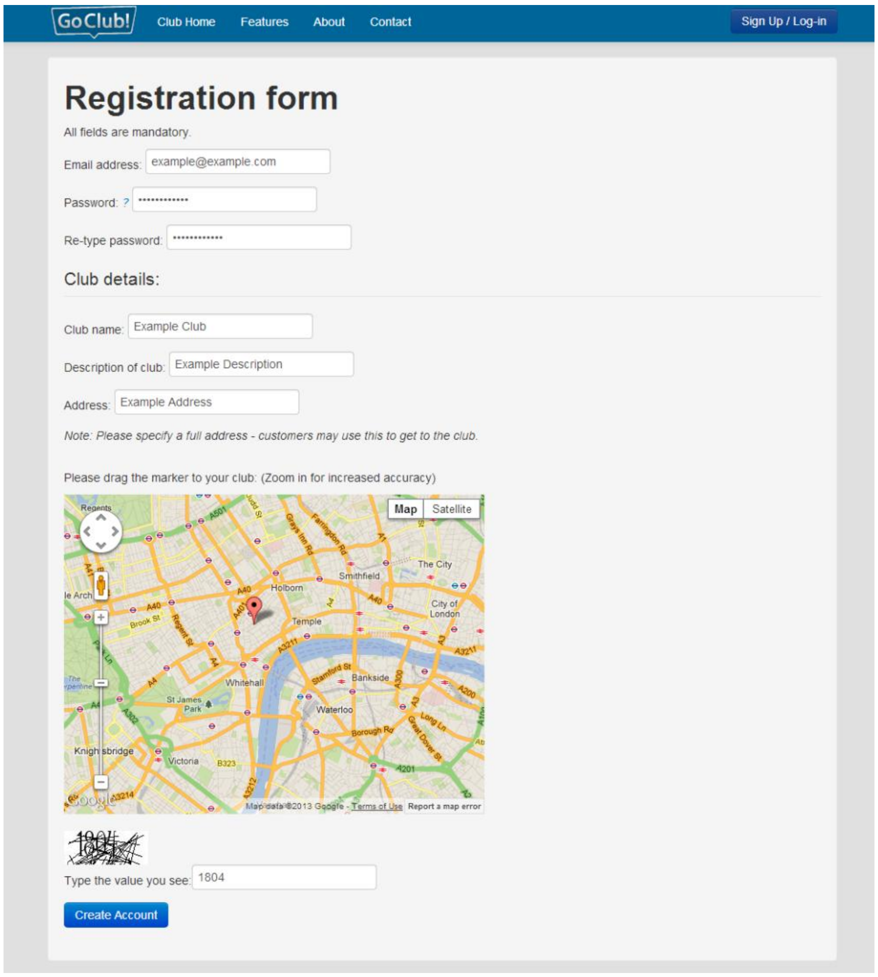
1.1 Registering an Account

The website provides an easily accessable way to manage your promotions through the GoClub system. From your account, you can manage your clubs stored profile and also manage adverts that you place on the application. To gain an account, you must go through the registration process. This is described below.

1) Go to http://rossphillips.eu/goclub/register/
2) The registration form in Figure 1.1 will be displayed.
3) Enter your details in all the fields of the form (all fields are mandatory). Notable points:

- Email Address: This must be an email using the clubs' website address.
- Password: This must be between 8 and 16 characters and contain at least 1 upper case, 1 lower case and 1 number.
- Club address: This will be displayed on the application, so please ensure it is accurate enough for people to find your club with it.
- Map marker: Drag the maps marker to the location of your club. This is how the
- application knows where to generate routes to. Ensure this is accurate.
- Image verification: Type the numbers that you can see in the image. This is used to verify you are not a computer.

4) On clicking submit, the form will alert you to the result of your registration. If an error in the values submitted causes the registration to fail, you will be alerted to the problem.



Registration form

All fields are mandatory.

Email address: example@example.com

Password: ? ••••••••••••

Re-type password: ••••••••••••

Club details:

Club name: Example Club

Description of club: Example Description

Address: Example Address

Note: Please specify a full address - customers may use this to get to the club.

Please drag the marker to your club: (Zoom in for increased accuracy)

Type the value you see: 1804

Create Account

Figure 1.1 Website Registration form

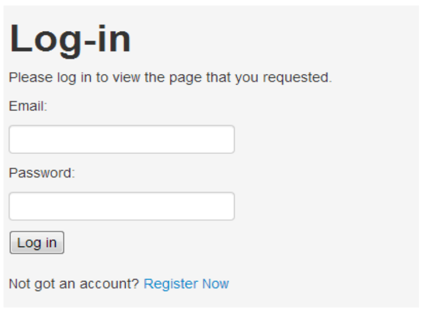
5) A successful submission will send an email to the email address supplied to ensure it was given correctly. Login to your email account and click the validation link on the email.

6) Once this has been clicked, you must now await approval of your new registration by the website administrator before being able to login.

You have successfully registered an account.

1.2 Logging in

Once an account has been registered and approved, you can then login following the steps below:



Log-in

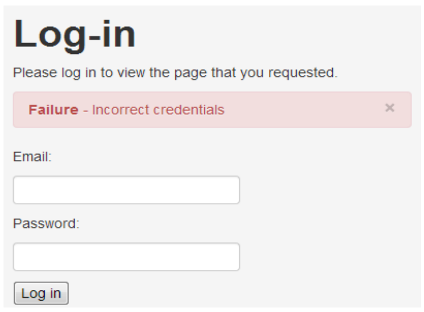
Please log in to view the page that you requested.

Email:

Password:

Log in

Not got an account? Register Now



Log-in

Please log in to view the page that you requested.

Failure - Incorrect credentials ×

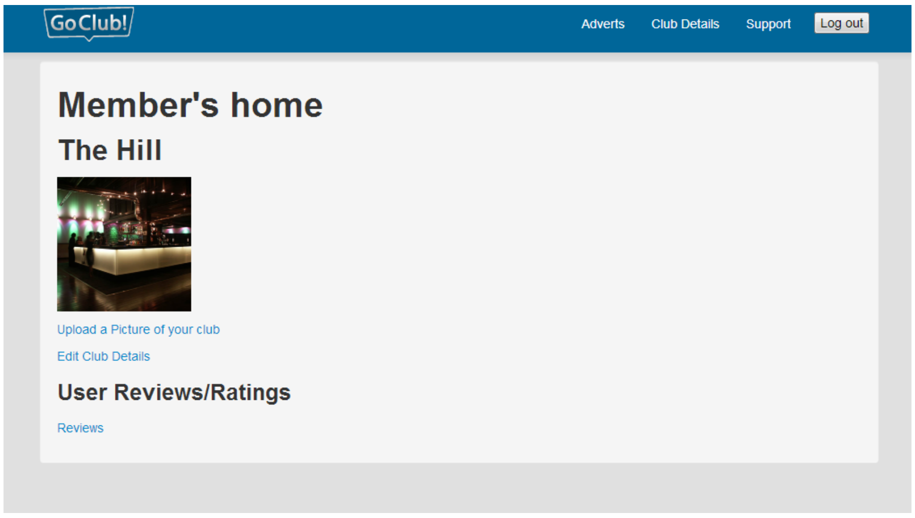
Email:

Password:

Log in

Figure 2.1 The Login Form - Figure 2.2 Failed login

1) Go to http://rossphillips.eu/goclub/members/
2) The login form in Figure 2.1 will be displayed.
3) Type your credentials supplied in the registration process.
3.1) Incorrect credentials will result in error alert (shown in Figure 2.2).
3.2) Correct credentials will log you into your account and display the member's dashboard as shown in Figure 2.3.



Member's home

The Hill

User Reviews/Ratings

Figure 2.3 Member's dashboard

The Member's dashboard provides access to the rest of the website's functionality. From here you can: 1) Upload a profile picture 2) Edit your club details 3) Manage reviews of your club
4) Manage adverts 5)Contact support.

The member's dashboard also displays your clubs' profile picture.

1.3 Uploading a profile picture

The profile picture is the image displayed along with your clubs' adverts when viewed on the application. If no image has been uploaded, the default (shown in Figure 3.1) is displayed.

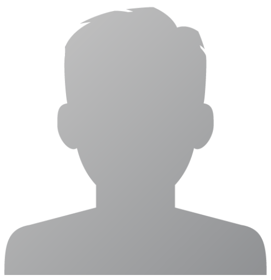
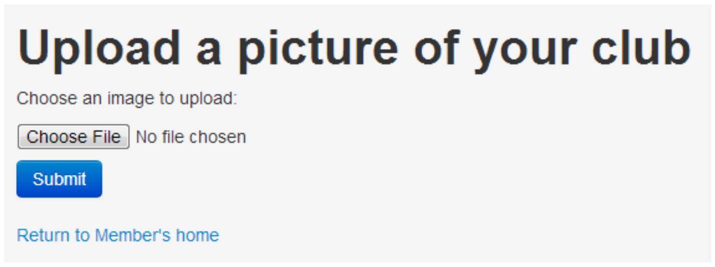


Figure 3.1 Default image

Below details the steps to upload a new profile picture.

Steps:
1) On the member's dashboard, click "Upload a picture of your club".
2) The image upload form, shown in figure 3.2, will be displayed.

3) Press 'choose file' and browse your computers' files to find one to upload. Make sure the chosen image meets the following criteria: 1) The file must be a .jpeg file. 2) The file must be under 2MB.



Upload a picture of your club

Choose an image to upload:

Figure 3.2 Image upload form

4) The website will state the file has been uploaded successfully.
You have now uploaded a new profile picture.

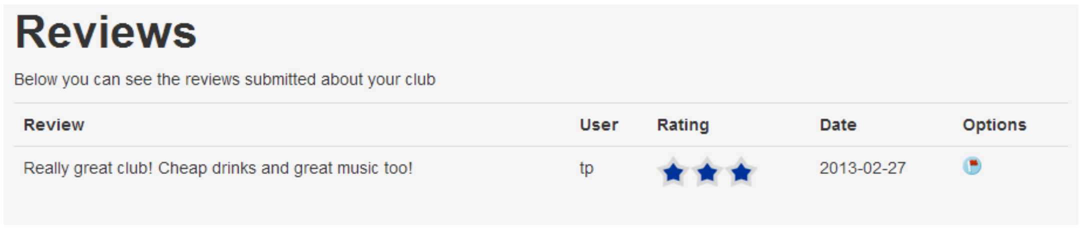
1.4. Handling reviews of your club

In the application students can place reviews about clubs. You can manage the reviews of your club from your account. It is possible to flag a review if you believe it is not suitable for distribution, it will then not be distributed to the application while it is pending moderation by an admin.

Below details the process of flagging a review.

Steps:

1) Click 'Reviews' from the member's dashboard.
2) The reviews submitted about your club will be displayed (shown in Figure 4.1).
3) Under the options column a flag icon is displayed; click this.
You have now flagged a review.



Reviews

Below you can see the reviews submitted about your club

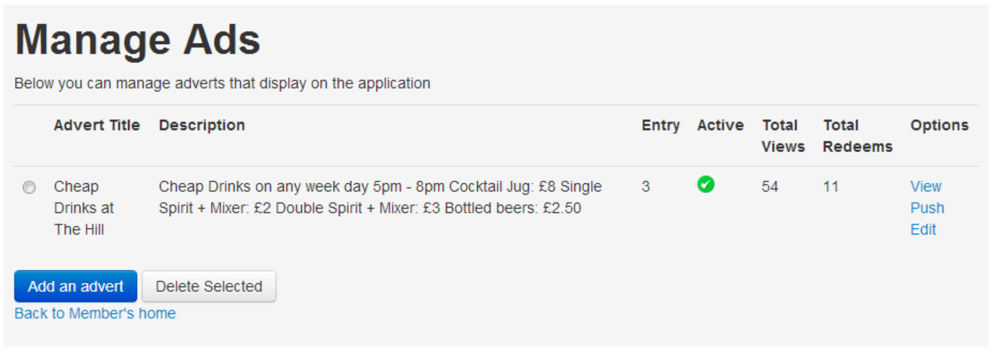
Review	User	Rating	Date	Options
Really great club! Cheap drinks and great music too!	tp	★★★	2013-02-27	⚑

Figure 4.1 Review list

1.5 Managing adverts

The website provides a way to manage adverts placed onto the application. This is done through the advert management dashboard (Shown in Figure 5.1). This dashboard gives an overview of the adverts placed by your club. Each advert is shown with information; including the title, description, entry price. Basic statistics are shown about the advert and also the 'Active' column indicates whether an advert is currently active or inactive. An active advert is one that is distributed to applications whereas an inactive one is not. Setting an advert to inactive removes the advert from the app, but keeps its details stored on your website, so it can be re-activated in the future.

The advert management dashboard provides access to the following features: 1) Adding a new advert 2) Viewing an advert 3) Editing an existing advert 4) Pushing an advert



Manage Ads

Below you can manage adverts that display on the application

	Advert Title	Description	Entry	Active	Total Views	Total Redeems	Options
○	Cheap Drinks at The Hill	Cheap Drinks on any week day 5pm - 8pm Cocktail Jug: £8 Single Spirit + Mixer: £2 Double Spirit + Mixer: £3 Bottled beers: £2.50	3	✓	54	11	View Push Edit

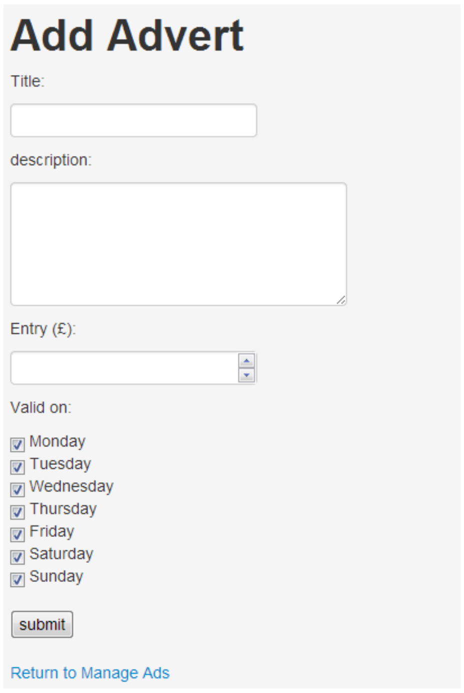
Figure 5.1 Advert Dashboard

1.6 Adding a new advert

Steps:

1) Click ‘Add Advert’ from the advert dashboard.

2) The website will display a form to allow details of your advert to be input (shown in Figure 6.1).



Add Advert

Title:

description:

Entry (£):

Valid on:

- [✓] Monday
- [✓] Tuesday
- [✓] Wednesday
- [✓] Thursday
- [✓] Friday
- [✓] Saturday
- [✓] Sunday

Figure 6.1 New Advert Form

3) Fill in the advert and description.

4) Select what days the advert is valid on.

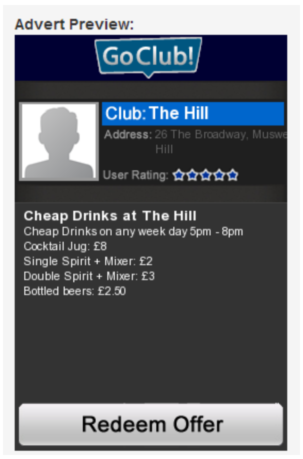
5) On pressing submit, a message is shown if the advert is submitted successfully.

1.7 Viewing an advert

Individual adverts can be viewed, by clicking ‘View’ next to an advert on the advert dashboard. Viewing an advert displays a range of information about it, including:

- The advert details: title, description, entry and days the advert is valid on.
- A preview of what the advert looks like on the application (shown in Figure 7.1)

- Statistics of the adverts success on the application: including views, redeems and the number of students that arrived at your club due to a particular advert. Example statistics and the associated graph are shown in Figure 7.2



Advert Preview:

GoClub!

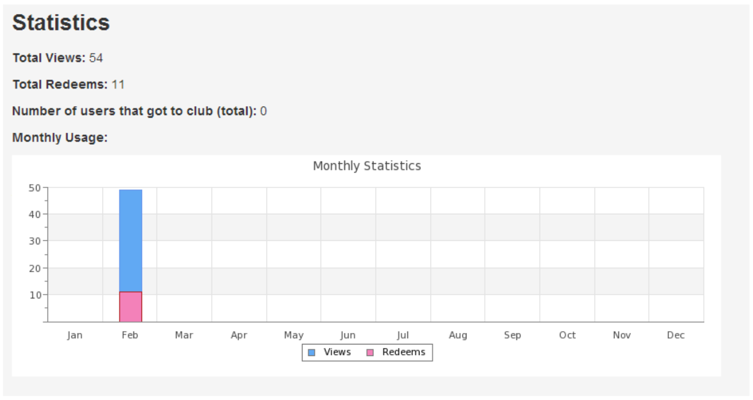
Club: The Hill

Address: 26 The Broadway, Muswe Hill

User Rating: ☆☆☆☆☆

Cheap Drinks at The Hill
Cheap Drinks on any week day 5pm - 8pm
Cocktail Jug: £8
Single Spirit + Mixer: £2
Double Spirit + Mixer: £3
Bottled beers: £2.50

Figure 7.1 Advert Preview



Statistics

Total Views: 54

Total Redeems: 11

Number of users that got to club (total): 0

Monthly Usage:

Figure 7.2 Advert Statistics

1.8 Contacting support

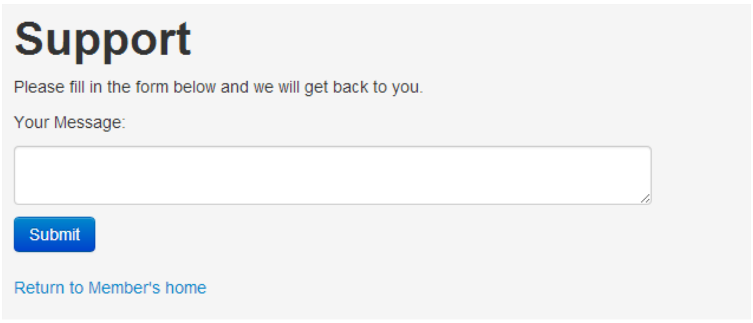
Support can be contacted in the case of a problem or query.

Steps:

1) Click the ‘Support’ link in the navigation bar of all logged in pages.

2) The form displayed in figure 8.1 is displayed.

3) Typing your message and press submit. This will email the website administrator.



Support

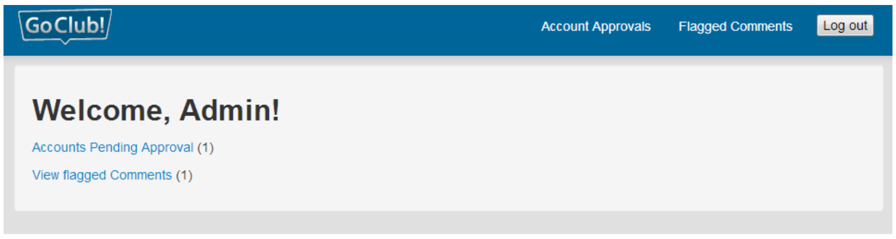
Please fill in the form below and we will get back to you.

Your Message:

Figure 8.1 Support

2. Website Administration:

Logging into an administrator's account displays a different dashboard. From this dashboard the administrator can: **1)** Approve pending registrations **2)** Moderate flagged comments. If there are pending tasks to be completed, the number of tasks to complete is given next to that task. This is shown in fig. 9.1. For example, there is one new account pending approval.



Welcome, Admin!

Accounts Pending Approval (1)

View flagged Comments (1)

Figure 9.1 Admin dashboard.

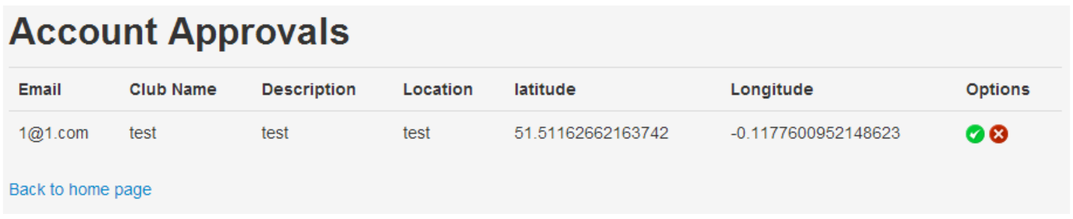
2.1 Approving accounts

Administrators can approve or deny membership to new accounts.

Steps:

1) From the admin dashboard, click 'Accounts Pending Approval'.

2) A list of accounts pending approval will be displayed (shown in Figure 9.2)



Account Approvals

Email	Club Name	Description	Location	latitude	Longitude	Options
1@1.com	test	test	test	51.51162662163742	-0.1177600952148623	✓ ✗

Back to home page

Figure 9.2 Account Approvals

3) The admin can view account information and then in the options click approve or deny.

A pending account has been dealt with.

2.2 Moderating comments

Administrators can moderate flagged comments.

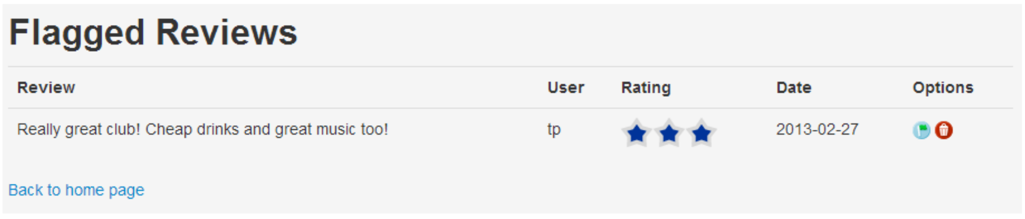
Steps:

1) From the admin dashboard, click 'View Flagged Comments'.

2) A list of flagged comments is displayed (shown in Figure 10.1)

3) The admin can view the comment information and then in the options either delete the comment or un-flag it.

A flagged comment has been dealt with.



Flagged Reviews

Review	User	Rating	Date	Options
Really great club! Cheap drinks and great music too!	tp	★★★	2013-02-27	● ●

Back to home page

Figure 10.1 Flagged Reviews

Appendix H

Project Plan

Name: Ross Phillips

Project title: GoClub – Mobile Phone application for nightclubs to promote special offers to students.

Supervisor's name: Graham Roberts

External supervisor's name: N/A

Aims and objectives

Aim: To develop an Android application, which will operate as an advertising platform for nightclubs to advertise discounts and special offers to students, via their mobile phones. Students can search (with a range of criteria) through adverts placed by local clubs, based on the student's current location, so the deals of nearby clubs can be found. Once the student has chosen which club to go to the app will give them directions to the club.

This will incorporate an online advert management portal, where club representatives can add, edit and delete offer listings for their club. The application will retrieve adverts relevant to the individual user, displaying them to the user via the apps interface or notification that appears on the users phone.

Objectives:

Application: Basic objectives

1. The application should connect up to an online database to retrieve adverts.
2. The application should store these adverts on an internal database for quick access.
3. The application should filter and display a list of adverts matching the user's input criteria.
4. The application should allow the user to click on an advert to view more details.
5. The application should provide a map with directions to the club from the user's location.
6. The application should display pictures of the clubs.

Application: Advanced objectives

7. The application should filter ads by the location of the user.
8. The application should allow users to view ratings of the clubs.
9. The application should allow users to input ratings of the clubs.
10. The application should allow the user to press redeem and will explain how to claim an offer.
11. The application should post to Facebook (if given permission to) when people choose to redeem a certain offer.
12. The application should have a way to track the success of adverts reporting these to the clubs through the website.

13. The application should be able to push special adverts directly to the user as a notification (for when the app is not open).

Website: Basic objectives

1. The website should allow clubs to log in to their own advert management dashboards.
2. The website should allow clubs to add, edit and delete adverts.
3. The website should allow clubs to upload a picture of their club.
4. The website should allow clubs to edit details stored about their club.
5. The website should look professional enough that industry would be prepared to use it.

Website: Advanced objectives

6. The website should provide a live preview of what adverts will look like on the app as they are input.
7. The website should report stats on advert performance.
8. The website should allow advert statistics to be displayed in graphs.

Deliverables:

- A design specification of the android application and website systems.
    - Requirements.
    - Use Cases.
- A fully working and tested android application.
- A fully working website for advert management.
    - Fully tested using the Selenium tool.

Work Plan:

Week beginning Monday 12th November

- **Application:** Have the application displaying a list of adverts matching the users input (App objective 3).
- **Application:** Allow the user to click on an advert to display it in more detail (App objective 4).
- **Application:** Implement a way to get pictures of the clubs advertising onto the users phone (App objective 6).
- **Website:** Have a live preview of what adverts will look like on the app. (Website objective 7).

Week beginning Monday 19th November

- **Application:** Find a way to access the user's location (App objective 5 and 7).
- **Application:** Look into plugging into Map API's. (App objective 5)
- **Website:** Start design process using CSS to make the website presentable (Website objective 5).

Week beginning Monday 26th November

- **Application:** The application should filter adverts automatically on the user's location. (Objective 7).
- **Website:** Continue implementing CSS to make the website presentable (Website objective 5).

Week beginning Monday 3rd December

- **Application:** Have a Map API plugged into that also uses the user's location (App objective 5)
- **Website:** Continue implementing CSS to make the website presentable (Website objective 5).

Week beginning Monday 10th December

- **Application:** Start work on allowing users to select a star rating of a club, having this stored to a server (Application objective 9).
- **Website:** Finish CSS work (Website objective 5).

Week beginning Monday 17th December

- **Application:** Fully implement star rating of clubs. (Application objective 9).
- **Application:** Display the average rating of a club when the user requests more info on an advert. (Application objective 8)
- **Application:** Implemented the app pushing certain adverts to the user's phone as notifications. (Application objective 12).

Week beginning Monday 24th December

- **Application:** Complete the redeem activity (App objective 10).
  Application: Look into ways of recording the success of offer uptake as statistics. (App objective 12).
- **Application:** Look into posting statuses to Facebook from the user of the apps account. (App objective 10)
- **Interim Report:** Work begins on interim report.

Week beginning Monday 31th December

- **Application:** Complete implementation of recording advert statistics. (App objective 12).
- **Application:** Complete integration of Facebook with app. (App objective 10).
- **Interim Report:** Work continues on interim report.

Week beginning Monday 7th January

- **Website:** Have the website reporting the stats that the app records. (App objective 7)
- **Website:** Start implementing display of advert statistics on graphs. (Website objective 8)
- **Interim Report:** Interim report finished.

Weeks through Monday 14th January – Monday 4th February

- System testing and evaluation.

Weeks through Monday 4th February – Monday 21st March

- Work on final report.

Interim report

Project title: GoClub – Mobile application for nightclubs to promote special offers to students.

Name: Ross Phillips
Supervisor's name: Graham Roberts
External supervisor's name: N/A

Progress made to date:

Below shows the objectives outlined in my project plan and the progress that I have made on them.

Application Objectives

#	Objective	done (%)	Comments/Details *(Where objective not at 100% complete).*
Basic Objects			
1	The application should connect up to an online database to retrieve adverts.	100%	
2	The application should store these adverts on an internal database for quick access.	100%	
3	The application should filter and display a list of adverts matching the user's input criteria.	100%	
4	The application should allow the user to click on an advert to view more details.	100%	
5	The application should provide a map with directions to the club from the user's location.	80%	Implemented fully however: • Slightly buggy (Exceptions thrown once in a while). • Map resets on device orientation change causing excess processing.
6	The application should display pictures of the clubs.	100%	
Advanced Objectives			
7	The application should filter ads by the location of the user.	90%	• Distances calculated and adverts are sorted on distance, with nearer adverts ordered first. • Need to now filter adverts so only adverts within a certain distance are shown.
8	The application should allow users to view ratings of the clubs.	100%	
9	The application should allow users to input ratings of the clubs.	100%	
10	The application should allow the	80%	• Need to review the layout and the

	user to press redeem and will explain how to claim an offer.		content of the redeem activity.
11	The application should post to Facebook (if given permission to) when people choose to redeem a certain offer.	50%	• User can connect to their own Facebook and pull personal data to fill forms such as their name. • Posting to Facebook mechanism is set up server side (on my own server and the correct Facebook settings are there), however I am having trouble getting the app to send the right request.
12	The application should have a way to track the success of adverts reporting these to the clubs through the website.	95%	• Progress tracking measures to see when the user is 0.2km away from the club and posts to the server. The current solution is using a static field for the location, this needs to be changed to get the user's actual location.
13	The application should be able to push special adverts directly to the user as a notification (for when the app is not open).	80%	• Done, but not location-based.

Website Objectives

#	Objective	done (%)	Comments/Details
Basic Objects			
1	The website should allow clubs to log in to their own advert management dashboards.	100%	
2	The website should allow clubs to add, edit and delete adverts.	100%	
3	The website should allow clubs to upload a picture of their club.	100%	
4	The website should allow clubs to edit details stored about their club.	95%	• All working, the word 'location' just needs to be changed to 'address' throughout system – website and app.
5	The website should look professional enough that industry would be prepared to use it.	90%	• Uses Twitter bootstrap framework for cross browser compatibility and professional CSS layout. • I noticed a problem on internet explorer with how the pages display, so this needs to be looked at.
Advanced Objectives			
6	The website should provide a live preview of what adverts will look like on the app as they are input.	100%	
7	The website should report stats on advert performance.	100%	
8	The website should allow advert	100%	

	statistics to be displayed in graphs.		

Work Plan:

Week beginning Monday 21st January

- **Application Objective 5:** Complete Mapping Interface/remove bugs.
- **Application Objective 7:** Allow the user to filter adverts based on distance from club.
- **Application Objective 10:** Review and implement fully the redeem activity.
- **Application Objective 12:** Make success-tracking based on the actual user's location.
- **Application Objective 13:** Make notifications location based.
- **Website Objective 4:** Alter 'location' to be 'address' throughout website and application.

Week beginning Monday 28thJanuary

- **Website Objective 5:** Check for website compatibility with internet explorer.
- Consider and implement scalability mechanisms for the app and website.
- Bug fixing.
- Consider one final large development to the application.

Week beginning Monday 4th February

- Finish implementing scalability steps.
- Implement one final large development to the application.

Week beginning Monday 11th February

- Finish implementing one final large development to the application.
- System testing and evaluation.

Week beginning Monday 18th February

- System testing and evaluation.
- Commenting of code.

Weeks through Monday 25th February – Monday 25st March

- **Final Report:** Work on final report.

Appendix I

Code Listings

Notable classes from the system are below. This is **not a complete listing** and what is included is a small subset of the entire system's code. The complete code can be found on the attached CD. On the CD, the application contains .class files for the logic, as well as the other project resources such as the .xml files for the user interface design, AndroidManifest file, and drawables (graphics) used in the project.

LocationGiver

LocationGiver demonstrates how the location of the device is retrieved and supplied to the webserver. It also demonstrates how the Android AlarmManager is utilised to schedule the service to run every 15 minutes.

```
package com.rossphillips.goclub;
/* A service that gets the user's location and reports it to the webserver. It schedules itself to run every 15 minutes between 6pm and 2am. */
public class LocationGiver extends IntentService{
	LocationManager locationManager;
	private Location currentLocation;
	private boolean locationChanged;
	public LocationGiver() {
		super(LocationGiver.class.getSimpleName());
	}

	@Override
	protected void onHandleIntent(Intent intent) {
		SharedPreferences settings = getSharedPreferences("GoClub", 0);
		Boolean notificationstatus = settings.getBoolean("notifications",true);
		if(!notificationstatus){
		} else {
			locationManager = (LocationManager) this.getSystemService(LOCATION_SERVICE);
			locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,0,0,locationListener);
		}
	}

	private void reportLocation(){
		String userid = ((GoClubApplication) getApplication()).getUserId();
		if(locationChanged && userid != null){
			try {
				Location networkLocation = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
```

```
                                if(((GoClubApplication) getApplication()).isNetworkOnline()){
                                        String webaddress = "http://rossphillips.eu/goclub/setlocation.php?gcmid="+ ((GoClubApplication)
getApplication()).getUserId() + "&lat=" + networkLocation.getLatitude() + "&long=" + networkLocation.getLongitude() +"&key=xY1OGCB";
                                        ((GoClubApplication) getApplication()).connectToWebAddress(webaddress);
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
                scheduleNextUpdate();
        }

        LocationListener locationListener = new LocationListener(){
                public void onLocationChanged(Location location){
                        if(currentLocation == null){
                                currentLocation = location;
                                locationChanged = true;
                                reportLocation();
                        }
                        else if(currentLocation.getLatitude() == location.getLatitude() && currentLocation.getLatitude() ==
location.getLongitude()){
                                locationChanged = false;
                                Log.i("LG","Location the same");
                                locationManager.removeUpdates(locationListener);
                        }else {
                                currentLocation = location;
                                locationChanged = true;
                                reportLocation();
                        }
                        if(locationChanged){
                                locationManager.removeUpdates(locationListener);
                        }
                }
        }

        private void scheduleNextUpdate(){
                        Intent locationIntent = new Intent(this,this.getClass());
                        PendingIntent pendingIntent = PendingIntent.getService(this,0,locationIntent,PendingIntent.FLAG_UPDATE_CURRENT);
                        long timeNowMillis = System.currentTimeMillis();
                        long updateTimeMillis = timeNowMillis + 900000;
                        Time updateTime = new Time();
                        updateTime.set(updateTimeMillis);

                        if(updateTime.hour<18 && updateTime.hour >= 2){
                                while(updateTime.hour<18 && updateTime.hour >= 2){
                                        updateTime.hour++;
                                }
                                updateTimeMillis = updateTime.toMillis(false);
                        }
                        AlarmManager alarmManager = (AlarmManager) this.getSystemService(Context.ALARM_SERVICE);
                        alarmManager.set(AlarmManager.RTC, updateTimeMillis, pendingIntent);
                }
}
```

RedeemActivity

RedeemActivity demonstrates the methods involved in retrieving and generating a route for the user and then drawing this onto the Android MapView.

```
package com.rossphillips.goclub;

/* RedeemActivity displays a map with a route from the user's location to the selected club. The selected club is indicated on the map by a
GoClub marker. The Activity also checks the user's location and reports to the server if they are nearby the chosen club. */
public class RedeemActivity extends MapActivity implements LocationListener, OnClickListener{

        MapView map;
        MapController mapController;
        LocationManager locationManager;
        Geocoder geocoder;
        TextView redeemText;
        DbHelper dbHelper;
        Double latitude = null;
        Double longitude = null;
        DirectionGetter dg;
        ArrayList<String> route;
        String clubname;
        String locationtext;
        String adverttitle;
        Long advertid;

    /*The following method adapted from (Percentage changed:80%): Laurent Tonon, L.T. (28/04/2011) LocationActivity [Computer Program] Available:
http://marakana.com/s/post/311/tutorial_android_location_service_example?&lang=en_us&output=json&session-id=3ac36367b846ff486a910b884141eac5 */

        @Override
    public void onCreate(Bundle savedInstanceState) {
             super.onCreate(savedInstanceState);
             setContentView(R.layout.activity_redeem);
             Bundle extras = getIntent().getExtras();
             Integer id = extras.getInt("id");
             advertid = extras.getLong("advertid");
                Button claimButton;
             adverttitle = extras.getString("adverttitle");
             redeemText = (TextView)this.findViewById(R.id.howToRedeem);
             redeemText.setText("To claim once you are at the club, click the button below:");
             claimButton = (Button) findViewById(R.id.button1);
             claimButton.setOnClickListener(this);
             dbHelper = new DbHelper(this);
             map = (MapView)this.findViewById(R.id.mapview);
             map.setBuiltInZoomControls(true);
```

```
            route = new ArrayList<String>();
            mapController = map.getController();
            mapController.setZoom(16);

            locationManager = (LocationManager)this.getSystemService(LOCATION_SERVICE);

            geocoder = new Geocoder(this);
            Location location = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
            getLatLong(id);
            if(location != null){
                    this.onLocationChanged(location);
            }
        }

        public void onClick(View v) {
                if(v.getId() == R.id.button1){
                        AlertDialog.Builder alertbox= new AlertDialog.Builder(this);
                        if(adverttitle != null){
                                alertbox.setMessage("GoClub Offer: " + adverttitle);
                        } else {
                                alertbox.setMessage("GoClub Offer: error getting advert details");
                        }
                        alertbox.setNeutralButton("OK", new DialogInterface.OnClickListener() {

                        @Override
                    public void onClick(DialogInterface dialog, int which) {
                                //do nothing.
                        }
                        });
                        alertbox.show();
                        Location location = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
                        if(latitude != null && longitude != null && location != null){
                                if(CheckDistanceToClub(location.getLatitude(), location.getLongitude(), latitude, longitude)){
                                        if(((GoClubApplication) getApplication()).isNetworkOnline()){
                                                String statsiteaddress = "http://rossphillips.eu/goclub/submitstatistic.php?advertid=" + advertid +
"&type=gotTo&userid=" + ((GoClubApplication) getApplication()).getUserId();
                                                try {
                                                        ((GoClubApplication)getApplication()).connectToWebAddress(statsiteaddress);
                                                } catch (IOException e) {
                                                        e.printStackTrace();
                                                }
                                        }
                                }
                        }

                }
        }

        private class DirectionGetter extends AsyncTask<Double,Void,Void>{
```

```
        @Override
                protected Void doInBackground(Double...params) {
                JSONArray jsonRoute = buildaddress(params[0],params[1],params[2],params[3]);
                if(jsonRoute != null){
                        for (int i = 0; i < jsonRoute.length(); i++) {
                                JSONObject row;
                                try {
                                        row = jsonRoute.getJSONObject(i);
                                        route.add(row.getString("poly"));
                                } catch(Exception e){
                                        Log.i("GOCLUB",e.getMessage());
                                }
                        }
                }
                return null;
        }
        @Override
        protected void onPostExecute(Void param){
                if(!route.isEmpty()){
                        drawPath(route,0xff0000ff,clubname,locationtext);
                        map.invalidate();
                } else {
                        Toast.makeText(getApplicationContext(),"Back-end webserver failed to provide routing information",
Toast.LENGTH_SHORT).show();
                }
                }
        }

        private void drawPath(ArrayList<String> points, int colour,String clubname,String clubdescription) {
        Drawable drawable = this.getResources().getDrawable(R.drawable.goclubsmall);
           List<Overlay> overlays = map.getOverlays();
           GeoPoint gp1 = null;
           GeoPoint gp2= null;
           List<GeoPoint> l1 = null;
           for (int i = 0; i < points.size(); i++) {
                   l1 = decodePoly(points.get(i));
                   if(l1.size()>0){
                           drawPathWithPoly(l1,0xff0000ff);
                   }
           }

           GoClubItemizedOverlay itemizedoverlay = new GoClubItemizedOverlay(drawable,this);
           OverlayItem overlayitem = new OverlayItem(l1.get(l1.size()-1),clubname,clubdescription);
           itemizedoverlay.addOverlay(overlayitem);
            overlays.add(itemizedoverlay);
        }

        private void drawPathWithPoly(List<GeoPoint> geoPoints, int color) {
               List<Overlay> overlays = map.getOverlays();
        for (int point = 1; point < geoPoints.size(); point++) {
               overlays.add(new RouteOverlay((GeoPoint)geoPoints.get(point - 1), (GeoPoint)geoPoints.get(point), color));
        }
}

/*  The following method from:
 *  Jeffrey Sambells, J.S. (27/05/2010) decodePoly [Computer Program]
 * Available: http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java
 * (Accessed 20/03/2013)*/
private List<GeoPoint> decodePoly(String encoded) {
    List<GeoPoint> poly = new ArrayList<GeoPoint>();
    int index = 0, len = encoded.length();
    int lat = 0, lng = 0;
    while (index < len) {
        int b, shift = 0, result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lat += dlat;
        shift = 0;
        result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lng += dlng;
        GeoPoint p = new GeoPoint((int) (((double) lat / 1E5) * 1E6),
             (int) (((double) lng / 1E5) * 1E6));
        poly.add(p);
    }
    return poly;
}

public JSONArray buildaddress(Double startlatarg, Double startlongarg, Double endlatarg, Double endlongarg){
    String startUrl = "http://rossphillips.eu/goclub/generateroutepoly.php?";
    String startLat = "startlat=";
    String startLong = "&startlong=";
    String endLat = "&endlat=";
    String endLong = "&endlong=";

    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(startUrl);
    stringBuilder.append(startLat);
```

```
        stringBuilder.append(startlatarg);
        stringBuilder.append(startLong);
        stringBuilder.append(startlongarg);
        stringBuilder.append(endLat);
        stringBuilder.append(endlatarg);
        stringBuilder.append(endLong);
        stringBuilder.append(endlongarg);
        return ((GoClubApplication) getApplication()).connectTo(stringBuilder.toString());
    }

    private void  getLatLong(Integer id){
        Cursor cursor = null;

        SQLiteDatabase db = dbHelper.getReadableDatabase();
        try{
                cursor = db.query(DbHelper.CTABLE, new String[]
{DbHelper.C_CLUBNAME,DbHelper.C_LOCATION,DbHelper.C_LAT,DbHelper.C_LONG},DbHelper.C_CLUBID +" = " + id, null, null, null, null, null);
                cursor.moveToNext();
                clubname = cursor.getString(cursor.getColumnIndex(DbHelper.C_CLUBNAME));
                locationtext = cursor.getString(cursor.getColumnIndex(DbHelper.C_LOCATION));
                latitude = cursor.getDouble(cursor.getColumnIndex(DbHelper.C_LAT));
                longitude = cursor.getDouble(cursor.getColumnIndex(DbHelper.C_LONG));
        } catch(Exception e){
                Log.i("GOCLUB",e.getMessage());
        } finally {
                cursor.close();
                db.close();
            }
        }

    protected void onResume(){
        super.onResume();
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,30000,50,this);
    }

    protected void onPause(){
        super.onPause();
        locationManager.removeUpdates(this);
    }

    public void onLocationChanged(Location location) {
        int locationlatitude = (int)(location.getLatitude() * 1000000);
        int locationlongitude = (int)(location.getLongitude() * 1000000);
        GeoPoint point = new GeoPoint(locationlatitude, locationlongitude);
        mapController.animateTo(point);
        dg = new DirectionGetter();
        dg.execute(location.getLatitude(),location.getLongitude(),(Double)latitude,(Double)longitude);
    }
    public boolean CheckDistanceToClub(double startLat, double startLong, Double endLat, Double endLong){
        int R = 6371;
        double distanceLatitude = Math.toRadians(endLat - startLat);
        double distanceLongitude  = Math.toRadians(endLong - startLong);
        double startLatRadian = Math.toRadians(startLat);
        double endLatRadian = Math.toRadians(endLat);
        double a = Math.sin(distanceLatitude/2) * Math.sin(distanceLatitude/2) + Math.sin(distanceLongitude/2) * Math.sin(distanceLongitude/2) *
Math.cos(startLatRadian) * Math.cos(endLatRadian);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        double d = R * c;
        if(d < 0.2){
                return true;
        } else {
                return false;
        }
    }
    …
}
```

ReviewCursorAdapter

Review cursor adapter gives an example of how a cursorAdapter is structured.

```
package com.rossphillips.goclub;
/* ReviewCursorAdapter is a custom CursorAdapter for the review ListView. It specifies how data received in a cursor should be displayed as a row
in ListView.  */
public class ReviewCursorAdapter extends SimpleCursorAdapter{
        private Context context;
        private int layout;
        public ReviewCursorAdapter(Context context, int layout, Cursor c, String[] from, int[] to, int flags) {
                super(context, layout, c, from, to, flags);
                this.context = context;
                this.layout = layout;
        }

        public View getView(int position, View convertView, ViewGroup parent){
                View view = super.getView(position, convertView, parent);
                return view;
        }

        public void bindView(View v, Context context, Cursor c) {
                int idCol = c.getColumnIndex(DbHelper.REVIEW_ID);
                int clubIdCol = c.getColumnIndex(DbHelper.REVIEW_CLUBID);
                int usernameCol = c.getColumnIndex(DbHelper.REVIEW_USERNAME);
                int reviewCol = c.getColumnIndex(DbHelper.REVIEW_REVIEW);
```

```
            int ratingCol = c.getColumnIndex(DbHelper.REVIEW_RATING);
            int dateCol = c.getColumnIndex(DbHelper.REVIEW_DATE);

            String reviewid= c.getString(idCol);
            String ClubID = c.getString(clubIdCol);
            String username = c.getString(usernameCol);
            String review = c.getString(reviewCol);
            String rating = c.getString(ratingCol);
            String date = c.getString(dateCol);

            TextView reviewidtext = (TextView) v.findViewById(R.id.reviewid);
            TextView clubidtext = (TextView) v.findViewById(R.id.clubid);
            TextView usernametext = (TextView) v.findViewById(R.id.username);
            TextView reviewtext = (TextView) v.findViewById(R.id.review);
            TextView ratingtext = (TextView) v.findViewById(R.id.rating);
            TextView datetext = (TextView) v.findViewById(R.id.date);

            if(reviewidtext != null){
                   reviewidtext.setText(reviewid);
            }
            if(clubidtext != null){
                   clubidtext.setText(ClubID);
            }
            if(usernametext != null){
                   usernametext.setText(username);
            }
            if(reviewtext != null){
                   reviewtext.setText(review);
            }
            if(ratingtext != null){
                   ratingtext.setText(rating);
            }
            if(datetext != null){
                   datetext.setText(date);
            }
            RatingBar reviewrb;
            reviewrb = (RatingBar) v.findViewById(R.id.ratingBar);
            reviewrb.setIsIndicator(true);
            reviewrb.setRating(Integer.parseInt(rating));
     }
}
```

ARBrowserActivity

ARBrowserActivity sets up and manages the Augmented Reality SDKs interaction with the application.

```
package com.rossphillips.goclub;

/* ARBrowserActivity holds the augmented reality view. It configures the view setting variables and passes the view data to display.
 * It handles events that are passed back to the application through the urlWasInvoked method.*/
public class ARBrowserActivity extends Activity implements ArchitectUrlListener, LocationListener{
       private static final String TAG = ARBrowserActivity.class.getSimpleName();
       private final static double  TEST_LATITUDE =  51.588568;
       private final static double  TEST_LONGITUDE = -0.153702;
       private final static float    TEST_ALTITUDE = 150;
       private String apiKey = "cTDU+mVwJjNV1QHKaOjYhgt4AfVUNVTgb540n35qGOJFem3kB2/T4wbvP7g+r8G4x8bT3T4UC4nT2MtY”
       Cursor cursor;
       private ArchitectView architectView;
       private LocationManager locManager;
       private Location loc;
       DbHelper dbHelper;
       private List<PoiBean> poiBeanList;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.getWindow().setFlags( WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN );
        if(!ArchitectView.isDeviceSupported(this)){
               Toast.makeText(this, "minimum requirements not fulfilled", Toast.LENGTH_LONG).show();
               this.finish();
               return;
        }
        dbHelper = new DbHelper(this);
        setContentView(R.layout.main);
        this.setVolumeControlStream( AudioManager.STREAM_MUSIC );
        this.architectView = (ArchitectView) this.findViewById(R.id.architectView);
        architectView.onCreate(apiKey);
        locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        locManager.requestLocationUpdates( LocationManager.NETWORK_PROVIDER, 0, 0, this);
     }

    @Override
```

```
protected void onPostCreate(Bundle savedInstanceState) {
    super.onPostCreate(savedInstanceState);
    if(this.architectView != null)
            this.architectView.onPostCreate();
    this.architectView.setCullingDistance(1000);
    this.architectView.registerUrlListener(this);
    try {
            loadSampleWorld();
    } catch (IOException e) {
            e.printStackTrace();
            }
}

@Override
protected void onResume() {
    super.onResume();
     this.architectView.onResume();
    loc = locManager.getLastKnownLocation(LOCATION_SERVICE);
    if(loc!= null){
            this.architectView.setLocation(loc.getLatitude(),loc.getLongitude(), TEST_ALTITUDE,1f);
    }
}

@Override
protected void onPause() {
    super.onPause();
    if(this.architectView != null)
            this.architectView.onPause();
    locManager.removeUpdates(this);
}

@Override
protected void onDestroy() {
    super.onDestroy();

    if(this.architectView != null)
            this.architectView.onDestroy();
}

@Override
public void onLowMemory() {
    super.onLowMemory();
    if(this.architectView != null)
```

```
              this.architectView.onLowMemory();
    }

    @Override
    public boolean urlWasInvoked(String url) {
       List<NameValuePair> queryParams = URLEncodedUtils.parse(URI.create(url), "UTF-8");
       String id = "";
       for(NameValuePair pair : queryParams){
                     if(pair.getName().equals("id")){
                            id = pair.getValue();
                     }
       }
       PoiBean bean = poiBeanList.get(Integer.parseInt(id));
       Intent intent = new Intent(this, PoiDetailActivity.class);
       intent.putExtra("POI_ID", bean.getClubId());
       intent.putExtra("POI_NAME", bean.getName());
       intent.putExtra("POI_DESC", bean.getDescription());
       this.startActivity(intent);
       return true;
    }

    private void loadSampleWorld() throws IOException {
       this.architectView.load("arlayout.html");
       JSONArray array = new JSONArray();
       poiBeanList = new ArrayList<PoiBean>();
       SQLiteDatabase db = dbHelper.getReadableDatabase();
       try {
              cursor = db.query(DbHelper.CTABLE, new String[]
{DbHelper.C_CLUBID,DbHelper.C_CLUBNAME,DbHelper.C_LOCATION,DbHelper.C_LAT,DbHelper.C_LONG},null, null, null, null, null, null);
              if(cursor == null){
              }
              if(cursor.getCount() == 0){
              }
              else {
                     cursor.moveToFirst();
                     int i = 0;
                     do{
                            int clubid = cursor.getInt(cursor.getColumnIndex(DbHelper.C_CLUBID));
                            String clubname = cursor.getString(cursor.getColumnIndex(DbHelper.C_CLUBNAME));
                            String location = cursor.getString(cursor.getColumnIndex(DbHelper.C_LOCATION));
                            Double lati = cursor.getDouble(cursor.getColumnIndex(DbHelper.C_LAT));
                            Double longi = cursor.getDouble(cursor.getColumnIndex(DbHelper.C_LONG));
                            PoiBean bean = new PoiBean(""+i,clubname,location,1,lati, longi,TEST_ALTITUDE,clubid);
```

```
                                array.put(bean.toJSONObject());
                                poiBeanList.add(bean);
                                i++;
                        } while(cursor.moveToNext());
                }
                this.architectView.callJavascript("newData(" + array.toString() + ");");
        } catch (JSONException e) {
                e.printStackTrace();
        }
    }

@Override
public void onLocationChanged(Location loc) {
        if(this.architectView != null)
                this.architectView.setLocation((float)(loc.getLatitude()), (float)(loc.getLongitude()), loc.getAccuracy());
}
```

GoClubHome

GoClubHome is the homepage of the application. It demonstrates how UI elements are connected to by the class and how interaction with these elements can be managed (for example, the onClick method).

```
/* GoClubHome is the Activity that launches every times, except the first time, the application is opened. It allows the user to select criteria
to search Adverts and also provides access to the Augmented Reality view. It starts up the LocationGiver and UpdateDB services. */
public class GoClubHome extends Activity implements OnClickListener, OnItemSelectedListener {
        private Button goToAdvertsButton,augmentedRealityButton;
        Spinner entryPriceSpinner;
        Spinner daySpinner;
        Spinner distanceSpinner;
        int entryPriceChosen = 0;
        int dayChosen = 0;
        int distanceChosen = 0;
        String[] entryPrice = { "Search all entry prices","£0-4.99","£5-9.99","£10-14.99","£15-19.99","£20+"};
        String[] daySpinnerValues = { "Tonight","All days","Mondays","Tuesdays","Wednesdays","Thursdays","Fridays","Saturdays","Sundays"};
        String[] distanceValues = { "Search all distances","<1km","<2km","<3km","<5km","<10km"};
        Intent locationIntent;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_go_club);
    goToAdvertsButton = (Button) findViewById(R.id.goToAdvertsButton);
    augmentedRealityButton = (Button) findViewById(R.id.augmentedRealityButton);
    entryPriceSpinner = (Spinner) findViewById(R.id.entryPriceSpinner);
    daySpinner = (Spinner) findViewById(R.id.daySpinner);
    distanceSpinner = (Spinner) findViewById(R.id.distanceSpinner);

    ArrayAdapter<String> entryAdapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item,entryPrice);
    ArrayAdapter<String> dayAdapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item,daySpinnerValues);
    ArrayAdapter<String> distanceAdapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item,distanceValues);

    entryPriceSpinner.setAdapter(entryAdapter);
    daySpinner.setAdapter(dayAdapter);
    distanceSpinner.setAdapter(distanceAdapter);

    entryPriceSpinner.setOnItemSelectedListener(this);
    daySpinner.setOnItemSelectedListener(this);
    distanceSpinner.setOnItemSelectedListener(this);
    goToAdvertsButton.setOnClickListener(this);
    augmentedRealityButton.setOnClickListener(this);

    startService(new Intent(this,UpdateDB.class));

    locationIntent = new Intent(this,LocationGiver.class);
    SharedPreferences settings = getSharedPreferences("GoClub", 0);
   Boolean notificationstatus = settings.getBoolean("notifications",true);
   if(notificationstatus){
          startService(locationIntent);
   }
    ((GoClubApplication) getApplication()).startGettingLocation();
}

private final BroadcastReceiver mHandleMessageReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
          String newMessage = intent.getExtras().getString("message");
   }
};

public void onClick(View v) {
          if(v.getId() == R.id.goToAdvertsButton){
                 if(((GoClubApplication) getApplication()).getDBfull()){
                        Intent launchAdverts = new Intent(GoClubHome.this, AdvertListings.class);
                        launchAdverts.putExtra("entry",entryPriceChosen);
                        launchAdverts.putExtra("day",dayChosen);
                        launchAdverts.putExtra("distance",distanceChosen);
                        launchAdverts.putExtra("clubid",0);
                        startActivity(launchAdverts);
                } else {
                        final AlertDialog.Builder alertbox = new AlertDialog.Builder(this);
                        alertbox.setMessage("Please wait a few seconds for the database to load then try again");
                        alertbox.setNeutralButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface arg0, int arg1) {
                                }
                        });
                        alertbox.show();
                }
        } else if(v.getId() == R.id.augmentedRealityButton){
                if(((GoClubApplication) getApplication()).getDBfull()){
                        Intent launchAugReal = new Intent(GoClubHome.this, ARBrowserActivity.class);
                        startActivity(launchAugReal);
                } else {
                        final AlertDialog.Builder alertbox = new AlertDialog.Builder(this);
                        alertbox.setMessage("Please wait a few seconds for the database to load then try again");
                        alertbox.setNeutralButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface arg0, int arg1) {
                                }
                        });
                        alertbox.show();
                }
        }
    }

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    SharedPreferences settings = getSharedPreferences("GoClub", 0);
    Boolean notificationstatus = settings.getBoolean("notifications",true);
    if(notificationstatus){
        menu.findItem(R.id.notificationstatus).setIcon(R.drawable.ic_notification_off);
        menu.findItem(R.id.notificationstatus).setTitle("Turn notifications off");
    } else {            menu.findItem(R.id.notificationstatus).setIcon(R.drawable.ic_notification);
        menu.findItem(R.id.notificationstatus).setTitle("Turn notifications on");
    }
    return super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_go_club, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
                case R.id.notificationstatus:
                        SharedPreferences settings = getSharedPreferences("GoClub", 0);
                        Boolean notificationstatus = settings.getBoolean("notifications",true);
                        SharedPreferences.Editor editor = settings.edit();
                        if(notificationstatus){
                                editor.putBoolean("notifications", false);
                                stopService(locationIntent);
                        } else {
                                editor.putBoolean("notifications", true);
                         startService(locationIntent);
                        }
                        editor.commit();
                        default:
                        return super.onOptionsItemSelected(item);
        }
    }
        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2,
                        long arg3) {
                if(arg0.getId() == R.id.entryPriceSpinner){
                        entryPriceChosen = entryPriceSpinner.getSelectedItemPosition();
                } else if(arg0.getId() == R.id.daySpinner){
                        dayChosen = daySpinner.getSelectedItemPosition();
                } else if(arg0.getId() == R.id.distanceSpinner){
                        distanceChosen = distanceSpinner.getSelectedItemPosition();
                }
        }
}
```

Webserver code:

GenerateAdvertListing

GenerateAdvertListing demonstrates how information is retrieved from the database and returned to the requesting application as a JSON Array.

```php
<?php
/* Generates the current advertlistings and returns these to the requesting application as JSON.*/

include $_SERVER['DOCUMENT_ROOT'] . '/goclub/includes/connect.inc.php';
$select = 'SELECT advertid, title, description, clubid, entry';
$from = ' FROM advert';
$where = ' WHERE active = 1';
if(isset($_GET['clubname'])){
        $clubname = mysqli_real_escape_string($connection,$_GET['clubname']);
        if($clubname != '') {
                $sql = "SELECT clubid FROM club WHERE clubname = '$clubname'";
                $result = mysqli_query($connection, $sql);
                if(!$result){
                        echo 'error1';
                }
                $row = mysqli_fetch_array($result);
                $clubid = $row['clubid'];
                $where .= " AND clubid = '$clubid'";
        }
}
$result = mysqli_query($connection, $select . $from . $where);
if(!$result){
        echo 'error';
        exit();
}
$adverttitles = array();
while($row = mysqli_fetch_array($result)){
        $adverttitles[] = array("advertid" => $row['advertid'], "title" => $row['title'], "description" => $row['description'], "clubid" =>
$row['clubid'], "entry" => $row['entry']);
}
if(count($adverttitles) > 0){
        $jsoncode = json_encode($adverttitles);
        echo $jsoncode;
}else{
        echo "No adverts";
}
Return;
?>
```

Appendix J

References

[1] Marko Gargenta, M.G.2011.Learning Android: Building Applications for the Android Market. Sebastopol, CA: O'Reilly Media.

[2] Android, 2012. Android Developers [Online] Available at: http://developer.android.com/develop/index.html
[3] Bucky Roberts, 2010. Android Application Development. [Online] Available at: http://thenewboston.org/list.php?cat=6

[4] Diego Torres Milano, D.T.M. 2011. Android Application Testing Guide. Birmingham: Packt Publishing Ltd.
[5] Kevin Yank, K.Y.2009.Build Your Own Database Driven Website Using PHP & MySQL. 4th Edition. Collingwood, Australia: SitePoint Pty. Ltd.
[6] Stack Overflow, 2013. Android Questions. [Online] Available at: http://stackoverflow.com/questions/tagged/android

[7] App Agency Limited, 2011. 'London Clubs App by Kpopteam is currently free for your iPhone and iPod Touch'. Apps-Free, [Online] 10/04/2011. Available at: http://www.apps-free.co.uk/2011/04/10/london-clubs-app-by-kpopteam-is-currently-free-for-your-iphone-and-ipod-touch/
[8] Apple, 2011. 'iTunes Preview: London Clubs'. [Online] Available at: https://itunes.apple.com/gb/app/london-clubs/id372580063?mt=8
[9] Socio News, 2012. 'Love your nights out? There's an app for that!'. [Online] Available at: http://lovesocio.com/News/154-love-your-nights-out-theres-an-app-for-that

[10] James Cadle; Debra Paul; Paul Turner, JC; DP; PT. 2010. Business Analysis Techniques. Chippenham, UK: British Informatics Society Limited. Chapter 6; Section 1.
[11] Jan L. Harrington, J.L.H. 2009. Relational Database Design and Implementation. 3rd Edition. Burlington, MA: Morgan Kaufmann. Chapter 4.1.
[12] Anujarosha, 2012. Handling HTTP POST method in Android. [Online] Available at: http://anujarosha.wordpress.com/2012/01/27/handling-http-post-method-in-android/
[13] Android Snippets, 2011. Get the content from a HttpResponse (or any InputStream) as a String. [Online] Available at: http://www.androidsnippets.com/get-the-content-from-a-httpresponse-or-any-inputstream-as-a-string
[14] Thomas M. Connolly; Carolyn E. Begg, TMC; CEB.2 004. Database Systems. 4th Edition. New Jersey: Pearson Education. Chapter 13.
[15] SQLite,2013. About SQLite. [Online] Available at: http://www.sqlite.org/about.html
[16] Android Developers, 2013. Storage Options: Using Databases. [Online] Available at: http://developer.android.com/guide/topics/data/data-storage.html#db
[17] Android Developers, 2013. Loaders. [Online] Available at: http://developer.android.com/guide/components/loaders.html
[18] Android Developers, 2013. Cursor. [Online] Available at: http://developer.android.com/reference/android/database/Cursor.html
[19] Lars Vogel, 2010. Android SQLite Database and ContentProvider - Tutorial. [Online] Updated: 2013. Available at: http://www.vogella.com/articles/AndroidSQLite/article.html

[20] Movable Type Scripts, 2013. Calculate distance, bearing and more between Latitude/Longitude points [Online] Available at: http://www.movable-type.co.uk/scripts/latlong.html
[21] Android Developers, 2013. Location Strategies. [Online] Available at: http://developer.android.com/guide/topics/location/strategies.html
[22] PHP.net, 2013. Client URL Library. [Online] Updated: 15/03/2013. Available at: http://php.net/manual/en/book.curl.php

[23] Google Developers, 2013. Encoded Polyline Algorithm Format. [Online] Available at: https://developers.google.com/maps/documentation/utilities/polylinealgorithm
[24] Jeffrey Sambells, 2010. Decoding Polylines from Google Maps Direction API with Java. [Online] Available at: http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java
[25] Tobias Knell, 2010. Routing / Driving directions on Android – Part 2: Draw the route. [Online] Available at: http://blog.synyx.de/2010/06/routing-driving-directions-on-android-%E2%80%93-part-2-draw-the-route/
[26] Android Developers, 2013. Google Cloud Messaging for Android. [Online] Available at: http://developer.android.com/google/gcm/index.html

[27] Ravi Tamada, 2012. Android Push Notifications using Google Cloud Messaging (GCM), PHP and MySQL. [Online] Available at: http://www.androidhive.info/2012/10/android-push-notifications-using-google-cloud-messaging-gcm-php-and-mysql/
[28] Shane Conder & Lauren Darcey, 2011. Augmented Reality: Getting Started on Android. [Online] Available at: http://mobile.tutsplus.com/tutorials/android/android_augmented-reality/
[29] Wikitude, 2013. Wikitude SDK Documentation for Android. [pdf] Wikitude. Available at: Wikitude SDK Download
[30] Android Developers, 2013. Iconography. [Online] Available at: http://developer.android.com/design/style/iconography.html
[31] Android Developers, 2013. Styles and Themes. [Online] Available at: http://developer.android.com/guide/topics/ui/themes.html

[32] Marcel Oelke, 2008. 'MD5();'. [Online] Available at: http://md5.rednoize.com/
[33] Kevin Yank, K.Y.2009.Build Your Own Database Driven Website Using PHP & MySQL. 4th Edition. Collingwood, Australia: SitePoint Pty. Ltd. Page 281
[34] Philo Hermans, 2009. How to Implement Email Verification for New Members. [Online] Available at: http://net.tutsplus.com/tutorials/php/how-to-implement-email-verification-for-new-members/ [20/03/2013].

[35] Carnegie Mellon University, 2010. CAPTCHA: Telling Humans and Computers Apart Automatically. [Online] Available at: http://www.captcha.net/
[36] Asial Corporation, 2013. jpGraph – Most powerful PHP-driven charts. [Online] Available at: http://jpgraph.net/
[37] Bryan Sullivan; Vincent Liu, BS; VL. 2011. Web Application Security. Berkshire: McGraw-Hill. Chapter 7.
[38] https://moodle.ucl.ac.uk/mod/resource/view.php?id=1026474
[39] Nick Assendelft, 2012. 'NullPointerException on pressing back on native facebook login'. [Online] Available at: https://developers.facebook.com/bugs/120539814778972
[40] BrowserShots, 2013. [Online] Available at: http://browsershots.org/
[41] Adobe BrowserLabs, 2013. [Online] Available at: https://browserlab.adobe.com/
[42] Kai Chan, K,C. 2012. Choose the right PHP framework, .NET Magazine, [online] Available at: http://www.netmagazine.com/features/choose-right-php-framework
[43] Cake Software Foundation, 2012. Security. [Online] Available at: http://book.cakephp.org/2.0/en/core-libraries/components/security-component.html
[44] FatSoma, 2013. [Online] Available at: http://www.fatsoma.com/