**Mobile Application for the University’s Marketing Department**

Final Report for CS39440 Major Project

*Author*: Mark Lewis ([mal60@aber.ac.uk](mailto:mal60@aber.ac.uk))

*Supervisor*: Dr. Bernie Tiddeman (bpt@aber.ac.uk)

6th March 2016

Version 1.5 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (G401)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

**Declaration of originality**

In signing below, I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work I understand and agree to abide by the University’s regulations governing these issues.

Name …………………………………………

Date ……………………………………………

**Consent to share this work**

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

Name …………………………………………

Date ……………………………………………

**Acknowledgements**

I am grateful to…

I’d like to thank…

**Abstract**

My project is to develop a mobile application for Aberystwyth University’s Marketing Department. The application is aimed at prospective undergraduate and postgraduate students, advertising Academic Information, Student Life and Open Days at the University. The application needs to be able to work on devices when offline and should be finished to a professional standard in order to be used by the Marketing Department.

The app is developed for Android Devices, written in Java using the Android Studio SDK.

**Contents**

Contents

[1. Background, Analysis & Process 7](#_Toc448744686)

[1.1. Background 7](#_Toc448744687)

[1.2. Analysis 8](#_Toc448744688)

[1.2.1. Deciding topics areas for the app 8](#_Toc448744689)

[1.2.2. Professional finish and feel 9](#_Toc448744690)

[1.2.3. Storage of data offline 10](#_Toc448744691)

[1.2.4. Longevity of the app 10](#_Toc448744692)

[1.2.5. App platform 11](#_Toc448744693)

[1.3. Process 12](#_Toc448744694)

[2. Design 12](#_Toc448744695)

[2.1. Overall Architecture 13](#_Toc448744696)

[2.1.1. Data Storage 13](#_Toc448744697)

[2.2. Detailed Design 14](#_Toc448744698)

[2.2.1. Class – AppData 14](#_Toc448744699)

[2.2.2. Class – MainMenu 15](#_Toc448744700)

[2.2.3. Classes Containing a Single Layout Resource 16](#_Toc448744701)

[2.2.4. Classes Containing Multiple Layout Resources - Accommodation 16](#_Toc448744702)

[2.2.5. Classes Containing Multiple Layout Resources – Getting Here 17](#_Toc448744703)

[2.2.6. Classes Containing Multiple Layout Resources – Undergraduate/Postgraduate Courses 17](#_Toc448744704)

[2.3. User Interface Design 19](#_Toc448744705)

[2.3.1. Menu Design 19](#_Toc448744706)

[2.3.2. Single Page topic areas 20](#_Toc448744707)

[2.3.3. Pages Containing Large Amounts of Content 21](#_Toc448744708)

[2.4. Jsoup 22](#_Toc448744709)

[3. Implementation 24](#_Toc448744710)

[3.1. Features 24](#_Toc448744711)

[3.1.1. Filling the App with Content 24](#_Toc448744712)

[3.1.2. Offline Storage of Data 25](#_Toc448744713)

[3.1.3. Updating the Apps Content 27](#_Toc448744714)

[3.2. User Interface 27](#_Toc448744715)

[3.2.1. Single Page Topic Areas 28](#_Toc448744716)

[3.2.2. Pages Containing Large Amounts of Content 28](#_Toc448744717)

[3.2.3. Menu/Home Page 29](#_Toc448744718)

[4. Testing 31](#_Toc448744719)

[4.1. Overall Approach to Testing 31](#_Toc448744720)

[4.2. Integration Testing 31](#_Toc448744721)

[4.2.1. Problems Discovered During Testing 31](#_Toc448744722)

[4.3. Stress Testing 32](#_Toc448744723)

[4.3.1. Testing Environments 32](#_Toc448744724)

[4.4. User Testing 32](#_Toc448744725)

[4.4.1. Questionnaire Results Analysis 33](#_Toc448744726)

[5. Critical Evaluation 34](#_Toc448744727)

[5.1. Overall Reflection 34](#_Toc448744728)

[5.2. Tools Used During the Project 34](#_Toc448744729)

[5.2.1. Version Control System 34](#_Toc448744730)

[5.2.2. Fluid UI 35](#_Toc448744731)

[5.2.3. Android Studio 35](#_Toc448744732)

[5.3. Methodology 36](#_Toc448744733)

[5.4. Positives Outcomes 36](#_Toc448744734)

[5.5. Starting the Project Again 36](#_Toc448744735)

[5.6. Future Work 36](#_Toc448744736)

[6. Appendices 37](#_Toc448744737)

[A. Third-Party Code and Libraries 37](#_Toc448744738)

[B. Ethics Submission 38](#_Toc448744739)

[C. Code Samples 40](#_Toc448744740)

[D. Similar Apps used by other institutions 43](#_Toc448744741)

[E. Test Table 44](#_Toc448744742)

[F. Usability Questionnaire 47](#_Toc448744743)

[7. Annotated Bibliography 49](#_Toc448744744)

# Background, Analysis & Process

## Background

Prior to starting this project I had no previous experience working with the Android SDK, therefore one of my first tasks was to familiarise myself with it. I began by visiting the site for Android developers [1], where I was able to find tutorials on many aspects of Android development as well as tips for new developers on getting started.

Although I had no previous experience of developing an Android application (app), I chose this project, as it was an area of the course that I was keen to learn more about. I have studied Java during my undergraduate modules and when I learnt that Android apps are written in Java, I thought Android app development would be an area where I could develop the skills I initially gained from studying Java. I also like the design aspect of system development, and enjoy producing an interactive app at the end of the process.

The first tutorial I followed walks you through creating a basic app in Android Studio; the official IDE for Android. The tutorial includes setting up Android Studio, creating an app with a simple user interface and starting another activity. The tutorial guides you through the basic functions of Android Studio, and gives you a base from which you can start developing your own app.

This Android developer’s tutorial introduced me to the basic functions and structures of Android, where I learnt how to add items such as buttons and textboxes using xml as well as how to link the layout xml files to java classes. Another lesson in the tutorial was to move between Android activities, a core function needed when developing apps.

As this was my first time using Android Studio it gave me a chance to use the UI designer, a feature of the IDE which would be heavily used to design and implement layouts in my app. Creating layouts can be done using the device preview screen or programmatically using xml code. Initially I found the xml code confusing as it is something I have only briefly used previously, and so, I focused my attention on designing pages using the device preview screen.

There was a two week period between the start of the project and my first meeting with the Marketing Department, meaning there was a brief time where I had no specification or list of requirements for my app. I decided to use this time to conduct research into apps published by other higher education institutes in the United Kingdom.

I noticed that many universities had published apps, a large proportion of which were aimed at current students, with only a small number of universities also targeting potential students. This meant I only had a limited number of other apps available to provide me with an idea of suitable content, layout and functions which would be the most desirable in my own app.

I spent some time studying these apps in more detail, taking notes about the topics they covered, any libraries they were using and also the layout of content itself. All this information helped me build a picture of the type of data and layout I could potentially include in my app.

I also studied the webpages on the Aberystwyth University website to view the range of content which could be covered in the app. Many of the webpages had a natural flow between each pages content; I noted that the pages which were heavily advertised and promoted were the pages which would be most appropriate for the app, as this is the important or key information aimed at recruiting students.

I then combined the information from both sources, about competitor’s apps and the current content available on the University’s website, in order to start building a requirements list to present to the Marketing Department in our first meeting.

After my initial meeting with the Marketing Department I found that their ideas were very similar to what I had been working towards. They were keen to find out what their competitors were doing and if there was a general theme, they wanted to match it.

## Analysis

From the background research I learnt that many other institutes were using their prospective student apps in two ways; either to promote courses and student life at their University or to view publications such as magazines and prospectuses. From meetings with the Marketing Department I knew they wanted me to develop an app similar to the first style as they had been in talks with an external company to develop a separate system on which they could distribute publications.

### Deciding topics areas for the app

I knew my main task was to decide which topics I would address in the app. I was able to use the knowledge of competitor’s apps and an understanding of the available content on the University’s website gained during my background research, to start putting together some ideas.

Due to the University’s website containing a large amount of content aimed at prospective students, the number of topics and possible combinations available was similarly very large. This meant it was important not to over fill the app with too much content and instead select key information which would be the most informative and useful for the users of the app.

A final list of ten topics was decided to be covered in the app. Listed here;

* Undergraduate Courses
* Postgraduate Courses
* Scholarships and Funding
* Study Abroad
* Accommodation
* Open Days
* Virtual Open Days
* Getting Here

These topics were then grouped into three categories; Academic Information, Student Life and Open Days.

The list of topics was generated by comparing the content available on competitor’s apps with the information available from the Aberystwyth University website, to identify and understand the key areas of information. Once the list had been generated it was approved by the Marketing Department. This was an important aspect of the requirements gathering phase, as it meant the app would address the topics and provide the overall content that the customer wanted.

A full list of features used in other institutes apps can be found in appendix D.

There were other topic areas which could have been covered in the app that did not make the final list. One idea was to try and base the app around the University’s Online Virtual Open Day [2]. The Online Virtual Open Day gives potential students a taste of the University without having to physically travel to Aberystwyth. One feature of the Online Virtual Open Day which I looked at implementing in the app was the ability to chat with lecturers and support staff via an instant chat function.

I knew that the University used an instant messaging service called LibraryH3lp [3] which ran on the XMPP chat protocol. There was no API or support for building an Android App on their developer’s site so I set out to find a tutorial which I could follow to see if I could get a prototype working.

I found a tutorial on TutorialsFace [4] which I started to follow before the final list of requirements was decided. I did not manage to create a working prototype of the instant chat facility before it was ruled out as a feature of the App, but it did further my understanding of using Android Studio.

Another content area which was not included in the final design was information about the different service departments within the University. The University has several departments which offer support to students with non-academic issues such as Student Support, Information Services, Campus Services and many others. I had a discussion with the Marketing Department as to whether we should include content on these departments, as I thought it may be beneficial for educating users about other aspects of the University, but it was decided that it was not needed and would not be crucial information for recruiting students.

### Professional finish and feel

If the end product is of a high quality then it could be adopted by the Marketing Department and published on the App Store. This meant another aspect was to create an app that is professional enough to represent the University.

This means the app needs to have an appearance which is aesthetically pleasing as well as being efficient and easy to use. There are many interface design options available when creating an app; therefore it was important to select a style which could display the large amount of content in a readable manner, as the content will be displayed on a variety of devices with different screen sizes.

The interface also needs to be consistent throughout the app. This means pages which have a similar amount of content or structure need to use the same layout. This will help to give the app a more professional look as well as improving the user’s ability to navigate through the app.

When deciding on layout types it is important to choose styles which can be re-used and shared on pages with similar structures. If styles are picked which are too specific then the app could end up containing multiple layout types which will damage the consistency and appearance.

Another aspect of design which needs to be taken into consideration is the colour scheme. In order to look professional and represent the University the app needs to adopt the University’s colour scheme, for this reason the app implements yellow, navy and orange. The full list of colors I can use within the app are in Figure 1.

An app is not of a high quality just because of its appearance, it also needs to behave as the user expects in order to fall into the category of a professional app. The final app needs to be robust and reliable, so not prone to crashing; this can be achieved by putting in methods to ensure proper error checking and handling.

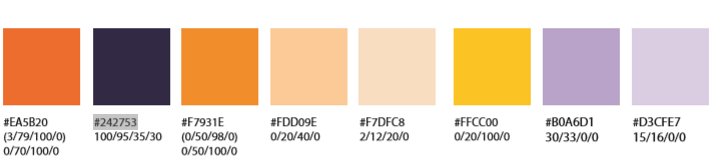


Figure . Colours used in the University colour scheme

### Storage of data offline

Another aspect of the app the Marketing Department were keen to see was being able to access the content at any time, making the app similar to a prospectus in your pocket which you could refer to wherever you are.

Therefore, the app needs to download and save content for each activity of the app somewhere on the device and then be able to load content back in when needed. There are multiple options which can be chosen for this feature and this created a new task of choosing the option which would be most beneficial to the app.

The amount of data which is within the app is quite large and so storing it on the device was chosen to reduce the load time between activities as well as improving the likelihood of an individual using the app. If the user is going to have to switch on their mobile data to use the app it may discourage them from doing so, as data charges can be relatively expensive.

### Longevity of the app

In order for the app to be reusable each year it is important that there is some way in which the content on the app can be updated. If the app needs to be re-written each year to contain the latest information then it has been poorly developed. For example it would be a bad idea to hard code the content into the app on creation as it could not be updated at a later date without re-building the whole app.

This means another task was to find a way to keep the content of the app up-to-date. There are multiple ways in which this can be achieved: using a content management system is one option. The University already has a content management system for its website, if it is able to work with an Android app then it would be a suitable option; if not then one would have to be created specifically for the app.

An alternative is to find a way to scrape data from websites and use it within the app; this requires a small amount of research into Java libraries. If there is the possibility of using a library to do this then it would be extremely helpful as it would save time by not having to write my own methods to do so.

I discovered that the University’s content management system did not work with Android apps, ruling out the option of using it to populate the app with content. I then turned my attention to whether or not I could create my own content management system or find an alternative. I looked into other content management systems online and found none which were suitable for my project.

There was the possibility of creating my own content management system. It would need a web interface in order for the Marketing Department to input the content and a database to store the content which they had uploaded. The data could then be read straight from the database and into my Android app. One drawback to this method was that the topics which were going to be covered in the app already had content available on the University’s website. This meant that the data would simply be duplicated by copying it straight from the site into the content management system.

So, I looked for a way of extracting data from the website and using it within my app. I found a Java library called Jsoup [5] which is a HTML parser and it looked as if it was able to fit my needs. You are able to get content from a given URL or from inputting a file, something which I would need in order to edit the data if the device was online, and also to load content which had been saved for offline use.

Once a webpage is parsed as a document you are able to manipulate the data by selecting HTML attributes, elements and text. After the content you require has been selected you can then output it as either plain text or wrapped with its existing HTML code, this means it could be displayed in either Android Text Views or Web Views.

### App platform

Developing mobile apps for multiple operating systems can be done in different ways. One way is to build an app which is cross-platform compatible. This usually involves building the app in HTML and using a program which will wrap code around your app in order to display it on each different operating system. Another alternative is to develop the app so it is native to one operating system and repeat this process for each operating system you want to target.

In an ideal situation the app would be useable on all mobile operating systems. Therefore I looked into the possibility of using PhoneGap [5], a program which allows you to build cross-platform apps using HTML, CSS and JavaScript. I did some research into PhoneGap and found that it had mixed reviews online, an article on Roadfire Software’s [6] website debated the pros and cons of using PhoneGap versus developing a native app.

The article suggests there is a noticeable drop in overall performance and many user interface bugs to overcome when developing an app using PhoneGap. There is also some design issues due to mobile differences, for example Android phones have a back button whereas iOS does not, and this is something that would have to be addressed when building the app.

Another reason why I decided to develop an app native to Android is because I had experience using Java and had already found several Java libraries which I intended to use. I felt I was more likely to produce an app of higher quality due to previous experience in developing Java than I would by starting from scratch with PhoneGap.

I also had a discussion with my Major Project Supervisor Bernie about which option may be the best. We decided that it was best to target a native Android App and if the finished product was good enough to be used then the Marketing Department could find another individual to build the app in iOS too.

A final factor which influenced choosing Android was its larger market share than any other mobile operating system available. According to IDC, at the second quarter of 2015 82.8% of mobile units shipped were Android devices [8]. This should result in the app having a larger target audience than choosing an alternative operating system.

## Process

The life cycle model I have used for this project is based around feature driven development as it seemed like a perfect fit for both this project and myself. An important aspect of feature driven develop is the features; gathering them and planning how you will implement them. Features are aspects of the system which are important to the customer, they are used to gather requirements and plan the design.

Initially there were no requirements set for this project and I had to meet with the Marketing Department in order to find what they wanted from the app. This helped me to work towards building a feature list and also developing an overall model of the system.

Once a feature list has been created, you can then start the next phase of planning by feature. Traditionally in feature driven development you assign features as classes to programmers in your development team, as this is an individual project the ownership of all the features belongs to me.

During the design by feature phase a design package is created for all features. The lead programmer meets with each programmer to create sequence diagrams for their features. Once all features have been assessed then the lead programmer selects a group of features to be developed over the coming iteration. These features are then developed and promoted to the main build after they have passed unit tests.

For my project this is the stage where I have decided to make an alteration to feature driven development. Instead of working in iterations I decided to work through my features in a progressive order. I tried to start off with features which were necessary in order for other features to be implemented. For example implementing the offline storage of content is a simpler task if you have already implemented the content, as you have a better understanding of how the content can be stored.

Feature driven development has a strong emphasis on project tracking. Features are split into milestones; there are six milestones per feature. This helps you to track the progress of the feature you are currently working on as well as the overall progress of the project. I felt this aspect of feature driven development allowed me to continue to be motivated during the project.

# Design

The first stage of design on a feature driven development project is to design by feature. As I already had a feature list and had chosen to work through the list sequentially, each feature’s design phase happened separately rather than as a group of features which were to be implemented over the coming iteration. This allowed the design to be much more dynamic as I was able to learn from mistakes made during the design of earlier feature’s and use similar designs for resembling features.

Implementing features one after another also meant that many of the original designs were no longer adopted. In most cases a design would be made which would allow the feature to be built into the app, then when it came to adding new features which relied on prior features, the old design would no longer be viable. This process meant it was important not to become too engrossed in creating concrete designs as they would likely change.

One aspect of design which could have been overlooked on this project was the user interface. There were no requirements set for the appearance of the app as the Marketing Department had given me full freedom to create a layout which I believed worked best for the task. For this reason it is easy to forget that the user interface is also a feature as it was not an aspect of the product that was stipulated by the customer.

Generally each feature is assigned its own class, depending on the complexity of the feature the amount of methods it contains will vary. Some features are considered as activities in an Android system, this means the user is able to interact with them and as a result they require their own layout file. Where activities are very much alike it is possible to re-use layout files in order to save time and space, but there may be special occasions where similar activities require different layouts.

## Overall Architecture

Each mobile app will have some form of home screen, depending on the apps function this may be used once or multiple times. I knew that my app would have multiple pages displaying content and would therefore require a main screen which could be used to link all of the pages together.

From the menu page there needs to be a way to access the detailed content on the ten topic areas which have been selected during the requirements gathering process. Each topic area will have its own page within the app, while some topics may require multiple pages; this is something which will be discussed further in the detailed design section of this document.

Before the user can view the content on the app it must first be downloaded and stored on the device, to do this I planned to implement a splash screen as they are commonly used in apps and games whilst they load. I felt that if executed correctly, the splash screen would give the app a professional feel and finish. Alternatively the app could have loaded the menu screen immediately, but the user would be unable to view any content as it would not have been saved on the device yet.

When the app is launched it should first check if there is data saved on the device. If there is no data downloaded on the device, if then the device must be online in order to fetch content. Depending on the result of these checks different methods need to run; download content, proceed to menu or display error for no internet connection.

Once the data has been successfully downloaded the user may proceed to the menu screen, where they are able to view the subject areas within the app. The design of the user interface can be make or break for a mobile app, in the user interface design section of this document I will outline the decisions I made in terms of design as well as justification for the mentioned decisions.

On selecting a content area from the menu the user should be taken to a section about that topic. Depending on the topic the section may be a singular page within the app, or it may contain its own menu where the user can select to view more specific data. For example the undergraduate and postgraduate courses are likely to broken down into departments and individual course.

### Data Storage

The data used to populate the pages within the app needs to be stored within a data structure. There are five possible storage options listed on the Android Developer’s website [9], I used this information in order to choose the storage method which I felt was most appropriate for the amount of files which would need storing.

Two of the options were easily dismissed straight away; shared preferences and network connection. Shared preferences are generally used to store user choices such as ringtones which are stored as objects on the device. These objects can then be loaded into the app to set it up to the users liking. As the data which needs to be stored on the phone is consistent across all user’s instances of the app this storage method would not be required.

Network connection involves the device being online in order to connect to a web-based service to store and load the data. Due to the requirement for offline storage within this app, network connection was removed as a potential option.

That left three conceivable routes; the first was a SQLite database on the device which is private to the app. A SQLite database would be useful if the app required large amounts of normalised data, however the data needed could be stored as string values and would not desire a complex data structure.

Therefore I was left to choose between storing the data on the devices internal or external storage. Ultimately I chose to use the phones internal storage instead of any external storage such as SD cards the device may have. One reason for choosing internal storage is because not all devices have external storage, and users don’t always use external storage even if their device supports it. Secondly the data that needs to be stored should not be too large in capacity and should be able to be stored on the device. The final feature was that data stored internally is only accessible by the app which creates it, unlike data stored externally which is available to any person or app. This was not a major factor when choosing between the two storage options as the data stored is not sensitive, but it does stop the user changing or deleting files which may cause the app to become unstable.

## Detailed Design

This section contains added detail on the app’s design, where appropriate classes will be discussed individually, but in some cases classes were very similar and therefore do not require individual deliberation.

### Class – AppData

This is the first class to run when the app is launched; the plan for this class is to download data onto the device. An initial check needs to be done in order to test the internet connection on the device; from there the data can be downloaded.

To stop data being downloaded each time the app is launched and connected to the internet a further check needs to be done; regardless of whether there is data already on the device. This check can be combined with the check for internet connection as both will return a Boolean answer. There are three potential answer combinations which will require action; data downloaded true, internet connection true and downloaded data false, finally internet connection false and data downloaded false. We can ignore the status of the internet connection at all times when data downloaded equals true, as the device does not need to be online to download data if it already exists.

This can be one method which is run in AppData’s onCreate after the layout file has been linked and any elements initiated.

If there is no data downloaded as well as no internet connection then an error message will be displayed. There is no method required to do this, as a single line of code can produce a message on the screen to tell the user to enable a connection.

When there is no data downloaded but the device is connected to the internet the class should run a method to download content. This is likely to be the most complex method within the app as it needs to connect to each page, manipulate the data and finally save it on the device. As this method is going to use the device’s network it must be run as part of a thread or AsyncTask.

This method will retrieve each page of the apps content and save it as a separate file for the corresponding page. These methods will require a URL for the webpage they need to connect to, then using methods from the Jsoup library, content can be selected in order to be saved.

I would like each method to download only the content required for the app in order to minimise the amount of storage the app will use. This should also speed up the process of loading data into the app as there will not be any data manipulation executed as pages are loaded.

Each method for generating the app’s content will need to save the data. For each method whose output is in the same format they can make use of the same save method. This method will require an input variable in order to name the files.

If the content is outputted in different formats then multiple save methods may be required. For example a different save method for saving strings and saving arrays.

### Class – MainMenu

This class runs the code behind the menu’s user interface. Regardless of the layout of the final menu screen this class has one main function, to provide navigation to the apps content. Methods will be required for each page the menu links to, these will be called when items or buttons are selected in order to start the next activity.

From the menu screen I also wanted the user to be able to re-download the apps content. This means that if the University webpages that were used to generate the content get updated, then users are still able to download the latest content onto their phone. This method could also be used if there is any corruption of data within the app, by replacing the current corrupted content.

In order to download the latest data, the files which are currently on the device must first be deleted. After the files have been deleted there needs to be a way in which the app can download the new content. To do this another method could be implemented within the menu class. But at its current state the app is the same as if it has just been installed on the device, therefore we can re-use the code which runs on start up within the AppData class to also generate content. This means instead of creating a new method an intent can be called to start the AppData class again.

The final method used within this app is to override Android’s handling of pressing the back button. Ordinarily if the back button is pressed then the app will move to the current activity’s parent activity (which is declared in the Android Manifesto) if there is no parent activity then it will rewind to the last used activity. As the menu is the apps main page there is no need for a parent activity and also no need to move back to the splash screen, for these reasons the back button can be disabled.

### Classes Containing a Single Layout Resource

Within the app five of the topic areas only require one page to display their content; Study Abroad, Students’ Union, Sports & Societies, Open Days and Virtual Open Days. This means these five classes all share similar structure and methods to avoid repetition of descriptions they have been categorised together.

Present in all activities within an Android app is an onCreate method and these five activities are no different. The onCreate method sets the activity’s layout by linking the appropriate XML file and initialising any interactive elements. Next the content needs to be loaded onto the screen, to do this the data must first be loaded from a file into a class variable, this requires a separate method. There will also need to be some error handling when calling the loading method in the event of a ‘file not found’ exception.

The method to load the content from a file is the same for these five classes, it can also be re-used in other classes in the app where the data is to be loaded in as a string. In order to load the file, the method will require the targeted filename as an input, this could either be taken as a variable as it is run or coded into each class’s version of the app.

### Classes Containing Multiple Layout Resources - Accommodation

The remaining five topic areas all have a larger amount of content and therefore are unable to be displayed on a single page, resulting in each topic area requiring multiple classes.

The first topic area is for Accommodation. An appropriate design for this section would be to first have a list of available accommodation which when clicked, directs the user to a page with details about the selected accommodation block.

For each specific accommodation block on the University Accommodation website [10] there are up to eight sections of content. Displaying all eight sections would potentially be too much content to view within a small mobile device and therefore I decided to reduce the number of sections included. I chose to exclude the following sections; What’s in your shared access, Location, Gallery and Tour. There were multiple reasons for not including these sections, for example the location and tour sections included map walkthroughs and were going past the level of detail required in the app. This left me with the following core sections; Overview, Facilities & Services, What’s in your room and Fees.

The first class I would require would be for the list of accommodation. This class needs a list of accommodation blocks, as well as the content for each block; these can be stored as arrays. It is possible to re-use the load from file method which is also present in the five classes containing single layout resources.

The list of accommodation blocks needs to be clickable so a second method is needed to handle the items being clicked. This method must start the accommodation detail activity, passing it the correct content for the chosen accommodation block. This can be done by putting the accommodation block’s content in to an intent and then starting the activity.

A second class is required to display the content specific to each accommodation block. This class will be in control of receiving the pages content, splitting it up into each section and arranging it. The data which has been sent from the intent in the first class will need to be stored as a class variable for use later on; there is no need for a separate method to do this.

There will be a method required in order to separate the received data into the four prescribed sections. This method will make use of Jsoup library functions, parsing string data into a document, selecting the needed content and finally returning four separate outputs. These outputs can be displayed immediately as part of the method, though it is a better idea to store them as class variables and then assign them to Android views; this should hopefully reduce data loss when switching between sections.

The exact layout of the accommodation details will be discussed further in the user interface design section of this document. Depending on the chosen layout style there may well be need for a class for each of the four sections.

### Classes Containing Multiple Layout Resources – Getting Here

This class’ task is to provide users with information on how to get to Aberystwyth and its University campuses. The four webpages which this section of the app are generated from contain information on travelling to Aberystwyth University via car, bus, train and from further afield.

Unlike most pages on the University website there was a lack of consistency in the layout of content within the webpages, this meant code would not be re-usable in order to generate content from these pages. I also felt that the content for getting to Aberystwyth via bus was too precise as it was limited to the buses within Aberystwyth and then linked to the local bus company’s site. Additionally I felt the content for traveling from other countries was irrelevant as it gave users a list of airports and directed them to the nearest train station with a link to National Rail’s website to check train times; which they would likely have already done themselves.

This left me with two remaining sections; car and train. The section on travelling by car appeared relevant and it gave postcodes for each campus within Aberystwyth, it also mentions the noticeable road signage to look out for while en route. The train content is a little vague but nevertheless it is a popular method of travelling to Aberystwyth and should be included.

The class would not require great amounts of complexity its main operation will be to load the data for each section into class variables and display them within a view. Therefore this class will only need the default onCreate activity and the load from file method which has been used throughout the other classes within the app.

### Classes Containing Multiple Layout Resources – Undergraduate/Postgraduate Courses

I have grouped together the final two classes as they both have the same structure, the only variant between the two is the area of webpage they generate content from. These classes will list the courses available to prospective undergraduate and postgraduate students.

On the website where the app’s content originates, the courses are listed first by department, then once a department has been selected a full list of courses for that department is shown. I plan to adopt the same structure within my app as I think it provides a simple way to find courses within the user’s subject of choice. The way the website differentiates between undergraduate and postgraduate courses is by using tabs which can be selected to change the list. When the tab is selected a different div element is shown on screen, I will need to make use of these separate div elements when fetching the data using Jsoup.

The first class required is for retrieving a list of departments. In order to display the departments as a list, each department will need to be selected individually and placed in an array. To do this the div containing the list of department needs to be selected, then all the links *(<a> tags*) can be selected; this will output the link’s text as well as any attributes it may be storing. Therefore, for each link selected, the corresponding text must be stored in an array and the links destination must be stored in a second array in order to find the next page in sequence. This process could be done when the app is first started and all the content is downloaded.

The first task for the class is to load in the lists and display departments on screen. If the data has already been stored in arrays then a new load from file method will need to be used as we are no longer loading in a string of data. Again, like other classes, a check needs to be run on the loaded data to catch any exceptions. Once successfully loaded the list can be displayed on the screen within an Android list view. This is probably the most appropriate view to display the content in as the layout of the list will be neatly displayed and the list view will also be clickable.

This leads on to the final method required in the first class, selecting a department. When a department is clicked it should then move on to the next screen; which will display a list of all courses provided by that department. To do this a method will be needed which finds the position of the department in the list and passes that position onto the next class, this can all be done using an intent much like in the accommodation class.

A second class is now required for displaying all courses within a chosen department. This class will need to receive information about which department has been selected, this should be done during the class’ onCreate method.

Now the class knows which department has been selected it must load in a file containing a list of courses corresponding to the correct department. It is likely that the list of courses will be generated in a similar manner to the list of departments, resulting in the courses being stored as an array. This means that the method used to load in the departments can also be used to load in the list. The list will be displayed using a list view in order to keep consistency and familiarity within the code.

Finally this class will require a method that can handle items in the list being selected. This method will be roughly similar to the one used when selecting a department.

The final class needed to view course information will be used to display material on a single course. Similar to the two classes previous this class will need to know which course it should be showing, this once more will be received from the intent sent from the preceding class.

Once the course is known, the class can then load in the corresponding file. The content is expected to be saved in string form and so the method used to load in the data will be the same as the load method most commonly used throughout the app.

The data from which the content for each course derives from seems to have an inconsistent number of sections depending on which course is selected. For this reason the most common sections will be displayed within the app, rather than each section available. By doing this I hope to avoid some courses looking sparsely populated compared to others as well as maintaining uniform content throughout. There appears to be five regular sections used in a large proportion of the course’s webpages; Course Overview, Course Content, Course Teaching, Employability and Student Views, and I plan to implement all five sections within the app.

## User Interface Design

Now that the topic areas have been selected and the content for each topic decided, decisions need to be made in regards to the user interface. For some sections of the app the interface will be simple and intuitive, for instances where topics only have one page of content. But for others there is the task of cleanly and effectively displaying large amounts of data in a way that is easily readable and navigable.

As the app is being developed for the Marketing Department and could potentially be representing Aberystwyth University it is vital that the app matches the house style adopted by the University. I sort out some information relating to the house style adopted by the University and received a copy of The University Brand Manual [11] from the Marketing Department. The document contains vast amounts of information which I would need to understand in order to match the house style.

There are details on using the Aberystwyth University logo and crest and a minimum size restriction which I could not exceed. The document also contained the hex values for colours in the colour scheme; these were useful as I could set the colours as defaults within my apps colour xml file. As the document was produced in 2013 I did have to get an updated version of some of the colours hex values from the Design and Print Department within the University.

Each activity within the app needs to have an xml layout file in order for the activity to be displayed on screen. For some activities the content would be similar and therefore the layout file could be shared. This will improve the consistency in the app’s layout and should give the user confidence that they understand how to navigate the app. The same principles should be adopted when using elements within a layout, such as buttons and list views, again to maintain consistency.

One design feature I knew I wanted across all pages of the app was a toolbar containing the page’s title. The toolbar would be best suited to using the dark navy colour with white text, as using navy as the background colour for each layout would result in an excessively dark screen. For certain pages the title can be hardcoded, but there will be pages where the title is unknown and must be set during runtime, for example when viewing specific course details.

The sections below contain more detail on individual aspects of the app. For some of the sections I have included mock up designs created using a free online user interface design tool called Fluid UI [12].

### Menu Design

As previously mentioned the menu screen plays an important role in the app by tying together all the content, thus the design of the menu needs to be easy to use and suitable for the style of the app.

One of my first ideas was to use a library for the menu page, I looked at a few options online at Android Arsenal [12] as I thought I was more likely to get a professional looking menu screen using a third party library. I found a few libraries which were based around Androids Navigation Drawer, where a menu bar appears when the user swipes across the screen or selects the menu button in the upper left of the screen. This menu style is commonly used within Android apps and it became a front runner from the start. There were also libraries featuring tiled views, where the screen is split up into rows and columns, with each field of the table linking to content.

Initially it was hard to choose a design for the menu as I was unsure how the content would all come together. So I started working towards implementing a Navigation Drawer as it seemed to be the most logical route to take and I needed a menu to start linking in other activities within the app. I found a tutorial on Code Theory’s website [13] which I used alongside the Navigation Drawers developers page on the Android site in order to create the menu.

Within a Navigation Drawer menu there are two aspects which need to be created; firstly the menu which swipes in and out, and also the main screen fragment. I decided to fill the main screen with introductory content taken from the Prospective Students welcome page as the content was relevant to this section of the app.

The navigation bar by default is laid out with an image at the top, followed by the menu items listed below. This layout suited what I wanted from the menu as it would allow me to place the Aberystwyth University logo at the top adding a brand to the app. The menu items can also be grouped together and given headings in order to place related topics together, I planned to make use of this feature as I had already grouped my topic areas together during the requirements gathering stage.

Another possible customisation to the Navigation Drawer is to change its background and text colours. This will allow me to adopt the University branding by changing the background colour to the bright yellow, as used on University publications and the text and toolbar to the secondary navy colour.

### Single Page topic areas

As previously mentioned, there are five topics within the app that can display their content on a single page. It is highly likely that the topics will all share the same layout file and so the design of all five can be discussed as one.

The first feature of these designs is the toolbar which will be present throughout the app, containing the activity’s title as well as a navigation button. By including a navigation button I hope to make it easier for users to move through the app’s pages on larger devices when the default back button may be away from the natural hand position.

The next area of the design to focus on is the area for displaying content. Depending on how the data is stored an Android Text View or Web View will be used to present the content. There are two options for the colour scheme here; yellow background with navy text or white background with black text. Below in Figure 2 are two previews of the potential designs.

My main concern on having the background colour as yellow for the whole screen is that it may look too garish, therefore I decided to use the white background design as a backup just in case the colourful yellow option is too overwhelming when implemented on the device.

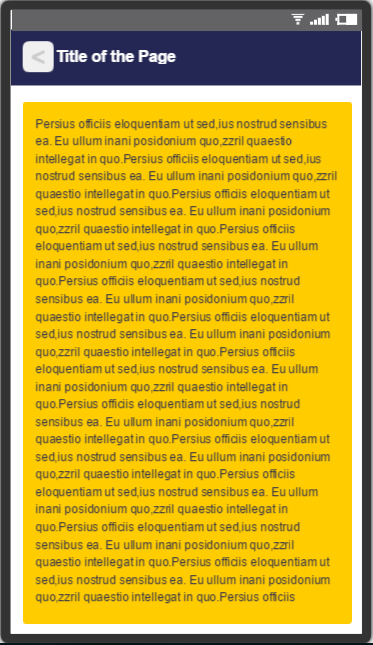
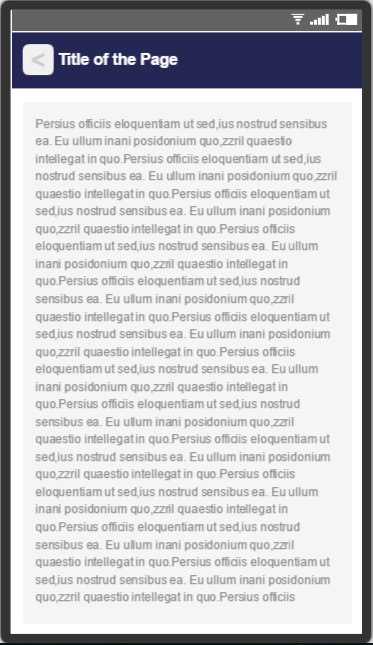


Figure . Designs for single page topics

### Pages Containing Large Amounts of Content

One of the major design issues for mobile app development is displaying large amounts of content on a small screen. However it is likely that the app will be displayed on an array of devices with varying screen sizes, so this also needs to be taken into consideration. Within my app there are a few activities that contain multiple sections of content, for these activities I will need to implement a layout which will smoothly manage the transition between sections without overcrowding the screen.

I wanted to pick a design which could be applied to all topics within the app consisting of multiple content sections in order to provide clean and consistent finish. One way to do this was to have a separate page and layout for each section, though this idea seemed clunky and would involve too much exploration by the user to find content. By dismissing this first idea I knew that I would like to display all the sections within one page.

I created two possible designs, one using drop down sections and the other using a tab bar along the top of the page. The designs are shown in Figure 3 and were created using Fluid UI and modelled around displaying course details; they helped give me a feel for how they would look if implemented within the app.

I preferred the appearance of the tabbed design, as it allowed each section to have an equal amount of screen available. Whereas in the drop down design the sections looked as if they were getting in each other’s way and taking up valuable space on the screen.

## Jsoup

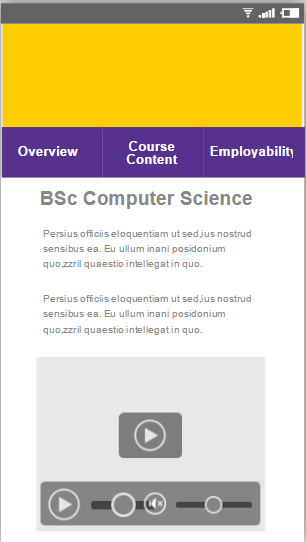
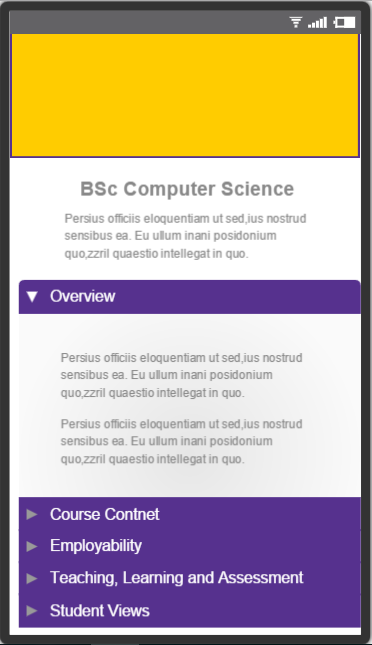


Figure . Potential designs for pages with a large amount of content

To generate content for the app I planned to use the Jsoup library. In order to re-use the app each year it would need to be updateable with the latest content, this meant that I would have to carefully select content in a way which would give the app the longest lifespan.

Jsoup is able to parse html from a given URL as well as from a document loaded in as a string. This means that a webpage can be manipulated and content generated at two separate stages in the apps lifecycle. I had the choice of whether to generate the content when the app is first launched and connected to the internet or when each page is displayed.

Downloading data to be stored on the device will take time and require the user to wait until the process has run. This would be an ideal point at which to generate the app’s content as it will reduce load time when manipulating content before it is displayed and would also reduce the time taken to save the content as it would only be the essential content being saved; not whole webpages.

A disadvantage to setting up the content on the apps original start up is that some data may be missed if the structure of the webpage was to change. By saving the webpage used to generate content the likelihood that targeted content will be missed is reduced. But the downside to this method is an increase in the size of files being stored on the device.

Another factor which will influence the time taken to generate content is the complexity of manipulation needed to scrape off relevant text from the webpage. For some topic areas the whole of the webpage’s body will be relevant, resulting in a simpler generating method. For others specific elements of the HTML will need to be selected and then operations performed on them to gather the required content. For these cases it may be best to create the data during the longer downloading method on start up.

There is potential for Jsoup to fail in its task of populating the app with content. If there is a major change to the structure of the University’s webpages and div ids are changed then the implemented methods will return null data. Another occasion when there may be null data present, is if the webpages specified for each topic area either change or are deleted; if this happens the app will not be able to connect to the site in order to generate any data.

Unfortunately, if any of the above does occur then there is little that can be done. I can put in place error checks to inform the user or there may be a way to revert back to the last working version of the app. But what Jsoup does offer is an easy way to maintain the app as the content is automatically generated. This means that if the Marketing Department update content on topic’s webpages then the changes should be naturally mirrored within the app. Using Jsoup means there will only be one content base, rather than multiple like if a content management system was used. This means that the app will be simpler for the Marketing Department to maintain in the future.

# Implementation

During the implementation phase I adhered to the methodology I had planned to use throughout the project. I worked towards adding each feature to the final build one at a time; I believe this helped me to develop a better understanding of the app and the relationships between each feature.

The methodology forced me at times to change my designs, as I learnt better ways to implement my plans and encountered problems which could not be addressed by my original designs. I think this was important as changing my designs and having a dynamic approach allowed me to practise many aspects of feature driven development, as well as agile software development.

One principle of agile methodology that I believe to be vital is to regularly produce working software, as this gives you an indication of how much the project has progressed, and also provides motivation as you can see the result of your work. By working through the feature list the app systematically came together and I was able to produce working software both at the end of each completed feature, and also once smaller tasks which made up a feature were finished.

I wanted to include functional content within the app as soon as possible and I worked towards including pages for topic areas before I created a comprehensive user interface. My thoughts were that the user interface side of the app could be fine-tuned once all of the content was incorporated, so I first created a menu screen based around the Navigation Drawer in order to add and test the content.

## Features

Within this section I will highlight any classes which may be of interest due to deviations from the original designs or because of the implementation of methods which are key to the app. I will try to describe features in the chronological order to which they were implemented with the aim of showing the way in which my methodology affected how the app came together.

### Filling the App with Content

After a basic menu had been created I started working towards developing my list of topics. Firstly, I started working on implementing the undergraduate course section of the app, as most of the code would be re-usable for the postgraduate course section with the addition of minor changes. This section would also be one of the largest and so I wanted to tackle this first.

I noticed that even though I was using feature driven development to include each feature within the app one at a time, I could also break up features into more manageable tasks. For example the undergraduate course feature was broken down into each topic; I then worked through each topic one at a time until they were all implemented. Some topics took up a larger percentage of time than others due to having a more complex class structure. However I found that as I became more familiar with the proposed layout types and Jsoup that the last few topic areas were added to the build much quicker.

As I was developing by feature I prioritised getting the content working in the app without initially worrying too much about the offline storage, this meant the device had to be online in order to view content. The app primarily worked this way as I was connecting directly to the University website in order to quickly show the results of the methods I had written to scrape data from the web. Once I had the correct output for each topic area I could then focus on the offline use.

For the topic areas whose content was able to be displayed on a single page I was unsure when designing the classes whether to use text or Web Views to display the information. When it came to creating these pages I found that Jsoup has two functions which could be used to generate content; one outputs just the text from selected elements, the other includes the html code it is contained within. I chose to keep the outer html on the data and load it into a Web View. The Web View would offer assistance with laying out the content as well as keeping original text styles such as headings within the view. Another advantage of using a Web View was its ability to contain links, as some content such as the Students Union’s page contains links to external sites. This means if a user of the app wants to find out more they are able to follow the links which will be opened within their browser.

Generating the content needed to be done off the main UI thread, therefore I had to use an AsyncTask class within the topics main class. An AsyncTask has three main methods which are run at different stages; pre-execute, on execute (called doInBackground) and post execute. I used the doInBackground method to parse in the webpage, manipulate the data and store it as a class variable. The variable could then be passed to the post execute method and used to display the content on the screen.

I did not encounter any major problems using Jsoup’s functions to display content within the app. But there were occasions when the data I received was not the same as the data I had expected which meant that I had to debug the code. The most common issue I experienced was when selecting text, Jsoup has a select function which will select all the chosen elements within a page or highlighted section. I found that text which was a heading or had been given emphasis on a webpage was missed due to it not being contained within a p tag. An alternative was to select everything within the highlighted section, but often this would lead to including content which was not necessary. Fortunately there is a Jsoup method which will find and remove all given elements from the output; this was used to combat the problem.

### Offline Storage of Data

The app was now filled with content when the device was online, allowing me to move on to the next feature; using the app offline. In my design I had planned to have a class which was responsible for connecting to the required webpages, selecting content and saving it onto the device. Before this class was created, I wanted to get a working save and load function and check that these were working as expected when the device was offline. Therefore I decided to test some methods on a small section of content on the menu page.

I found methods for both reading and writing strings to and from a file from an answer by a user called R9J on a stack overflow question [15]. These methods were used initially to get the offline storage to work for the content on the home screen and were also used later on to store all of the content within the app. The only change made from the original methods was to set the file name for each topic area; I have provided the two methods in Appendix C.

After getting the load and save methods working I started to implement a check which ran on the apps start-up; ‘is the device online?’, and ‘does the device need to download data?’ As there was not yet a method for downloading all the apps content I could only check whether the test page had been downloaded. There was also no splash screen yet so I used toasts to pop up on the screen to display the outcome of the checks. I found a method on Stack Overflow [16] from a user called Tim, which checks whether the device is online; I combined the method with one of my own which checked if a file existed. The method found on Stack Overflow uses a Connectivity Manager which means you need to allow the app permission to view the network state of the device; this is declared in the Android Manifesto for the project. The final method is listed in Appendix C.

I could then start work on the new class, AppData, which was responsible for downloading the app’s content. Again, I worked through each topic individually, refactoring code used in the topics class to generate content over to a new method within the AppData class. I also had to give each topic area the load method in order to store the pages content within a class variable.

Once each topic had been set up to store content on the device I was able to move the method which checks the status of the devices internet connection, and downloaded data into the AppData class. In order to check whether there was any content on the device the method would first need to check whether a file exists. The first time I copied the method across, I set it to check whether the first topic area from the download method was on the device. Later I changed this to the last topic area, in case there was a problem downloading after the first file had been created. It was changed one last time to check a file exists which would be generated once all topic areas had been successfully downloaded, eliminating the likelihood of missing content.



Figure . Example splash screen

To complete the class I needed to add a layout file, this would be the apps splash screen. The splash screen is a full screen layout, with the Aberystwyth University logo and app title. There is also a Text View placed at the bottom of the screen, the text displayed will vary on the status of the app and device. If there is no internet connection then a message is displayed telling the user to enable a connection and restart the app, if there is a connection but no data downloaded then it will inform the user that it is downloading data. If there is already data on the device then the app will move straight onto the menu activity. The final layout of the splash screen is shown in Figure 4.

The biggest change that occurred when implementing this feature, involved how the content was stored on the device. I had planned to have multiple save and load methods depending on whether the content was to be stored as a string or as an array. When it came to implementing the offline storage of data I realised that it would be much easier to store all topic areas as strings, and then manipulate the content as needed within each class.

One area of this feature that I had underestimated was gathering data for all courses on the University website. When the app was only working when the device was online this could be done by storing one array containing the course title and another with the URL to the corresponding course. When a course was selected it would send the appropriate URL in an intent to the next class. The content detail class could then use the URL to connect to the correct webpage.

I had to create multiple methods within the AppData class in order to download the webpage for all courses. Using Jsoup functionality the first method creates an array containing the URLs of each departments list of courses before calling the next method. The second method loops through each departments list, selecting the URL linking to the detailed webpage for each course and storing these in another array. A final method is called which saves all pages in the second array to a file, naming them based on the URL.

### Updating the Apps Content

This feature could only be implemented once the app was almost finished as it relied on content already being stored on the app. For this reason it was the final feature to be built into the app.

Implementing this feature could be done by using a single method and so it was relatively easy to do. The logic of the code written was the same as planned during the design phase; deleting each content file and then launching the AppData class. This seemed the most straight forward approach to updating the content.

The main issue implementing this feature was establishing how it would be executed; automatically by the app or manually by the user. Updating the content automatically could be done in two ways; time passed since the last update or depending on if changes had been made to the website. If the content was updated after a specified time period then the app would need to know when the content was last downloaded, and a suitable time period would need to be chosen. My concern with this approach was that an update would be executed unnecessarily even if the webpages had not been changed, resulting in a negative impact on the app as it would appear to be doing useless updates. Updating the content based on whether the website had changed would involve a more complex version of updates than if the update was based on time. Periodically the app would need to cross check the content on the device with content on the web, a high percentage of the time these checks would find no difference. Therefore I decided not to use this technique and allow the user to choose when the content is updated.

After it was decided that the content would be updated manually I had another choice of where to place the update option; either as an item in the menu along with the topics or within its own options menu. I kept the feature separate and added a drop down menu within the menu page’s toolbar, as this is the most common place you would find an options menu within an Android app. This meant that my app would have a familiar layout. Having a separate option menu also gave me an advantage if I was to find any other options which were required, as they too could be placed in this list.

## User Interface

Once all of the chosen features had been implemented within the app it was time to refine the user interface. Whilst building features into the app I included the layouts chosen for them in the design phase, but did not develop the layouts to a high quality at this time. This allowed me to see if the layouts were correct for the amount of content and structure of the information. If the layouts were not effective I was then able to revert to another layout without wasting resources.

As each layout was not fully finished for each feature, there was a small section of the implementation phase where I focused solely on the layout files associated with each activity. This phase consisted of small tasks which would contribute to the professional feel of the app; changing text styles, optimising the layout of Android elements, implementing the University colour scheme and finishing the home screen/main menu.

In the sections below I will go into further detail about the final designs.

### Single Page Topic Areas

The simplest layout file used was for the topic areas whose content could be displayed on a single screen. The layout consisted of a navy toolbar at the top of the screen, with the title of the topic in white text. The title of each topic is set programmatically during the activities onCreate method allowing the layout file to be reused across multiple topics.



Figure . An example single screen topic

The remainder of the screen was used to display content within an Android Web View. During the design phase I was unsure whether to set the Web Views background to yellow or leave it as the default white. I experimented with both on a device and decided to use the white background design as it was less overwhelming on the screen.

The final adjustment to the layout was to add padding around the Web View. This improves the readability of the text, by adding a grey border around the outskirts of the page. The result of the customisation is shown in Figure 5, where an example of the Study Abroad page is shown.

### Pages Containing Large Amounts of Content

When considering the layout for the pages that contained multiple sections and large amounts of content, I chose to implement a tabbed design. The implemented design uses an Android Tab Host Widget with individual tabs which include layouts for each section within the Tab Host.

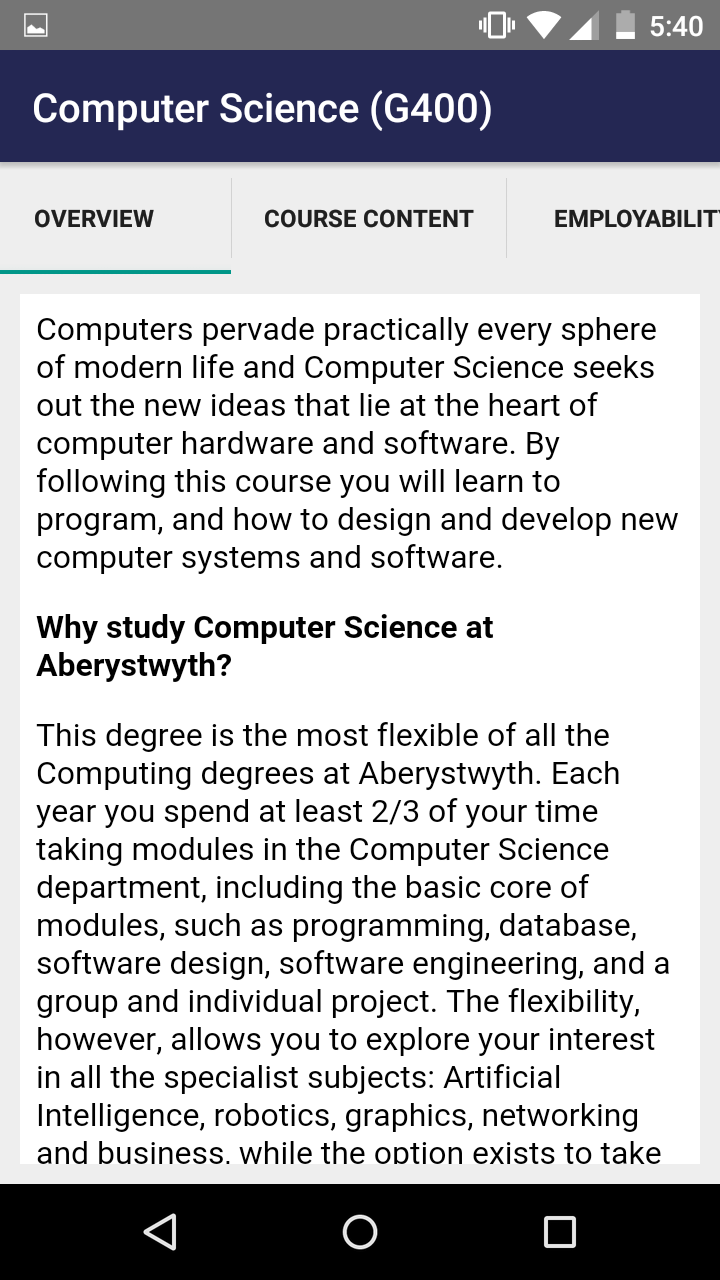


Figure . Tabbed layout used for undergraduate courses

The layout uses a navy toolbar with white text at the top of the page, keeping consistency with other pages within the app. The title is set programmatically for pages with large amounts of content as the title is changeable depending on the users input. Below the toolbar is the Android Tab Host, containing multiple tabs as appropriate. In cases where the number of tabs exceeds the width of the screen, the Tab Host scrolls horizontally to display them all.

Each tab within the Tab Host can use a separate layout file, but within this app each tab was able to share the same layout due to the content being displayed identically. The layout file used for each tab contains only an Android Web View with padding attributes to match the same style as Web Views used in other areas of the app.

Using this design did result in a change to the class structure first proposed for the app. Originally I planned to have one class for each topic area, even if it contained multiple sections. But when it came to implementing the tabbed design I had to create new classes for each section of a topic. This change occurred as tabs were not saving their content correctly, resulting in empty tabs when switching between sections.

Figure 6 shows how the tabbed layout looks for viewing an Undergraduate course.

### Menu/Home Page

I had originally planned to use an Android Navigation Drawer to display the apps menu and act as a home screen. I did initially implement the planned design, but found that once I had built in all the proposed features the structure was not suitable for a Navigation Drawer. I think the designed menu would have been more appropriate if there was regularly changing content which could be displayed on the home screen. But the final structure of content and topics was fixed and therefore I had to find a home screen design which would be more suitable for this style.

I chose a design which used buttons for each topic area, loosely based around the tile design used in Microsoft Windows operating systems. As topics had been grouped into categories I was also able to incorporate this into my design. I learnt the techniques required to build the menu from an answer by Voicu on Stack Overflow [17]; using Linear Layouts nested within each other to split the screen into rows and columns. This technique was combined with weighting the width of the buttons to give chosen topics a larger button which stretched to the width of the screen. The first category, Academic Information contained four topic areas; I chose to have two larger buttons for the Undergraduate Courses and Postgraduate Courses, as they were both the largest in terms of content and importance. This left two remaining topics which could share a line within the layout. The other two categories contained odd numbers of topics and so one topic was chosen from each to have a larger button, this ensured there were no gaps within the design.

The final layout of the home screen included a navy toolbar along the top of the app as used within each activity of the app. This toolbar is slightly different from the others as it has its own drop down menu on the right hand side, which includes the option for re-downloading the app’s content. Within the rest of the screen is the Aberystwyth University logo and the menu buttons.

To incorporate the University’s house style the background of the screen is set to yellow and the buttons use the combination of navy and white for their background and text. In other areas of the app’s design using yellow as a background colour was discouraged as it was too overwhelming, but it looks effective on the menu due to the quantity of navy neutralising the yellow and the fact that less text is being displayed.

Each button within the menu is linked to a method which creates an intent and starts a chosen activity. These methods are assigned to the buttons onClick attribute. The final design is shown in Figure 7; two screenshots are required as the menu does not fit on a mobile screen and so users must scroll vertically to view the full menu.

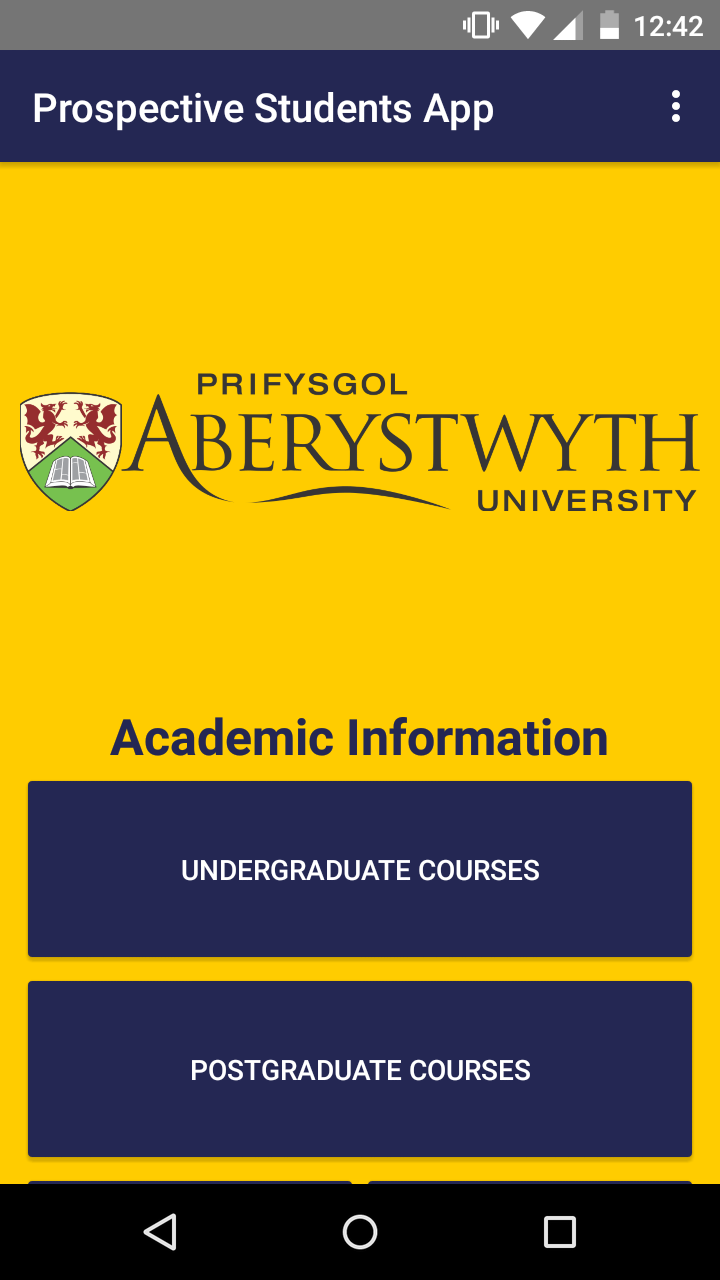


Figure . Menu screen showing all topics

# Testing

## Overall Approach to Testing

One requirement for this project was to create an app with a finish which is professional enough for it to represent the University. This could be achieved partly through building a robust app, meaning the testing must be rigorous in order to catch any bugs.

As I was following a methodology which was based around feature driven development it meant that each feature within the app underwent testing before it was promoted to the final build, this process allowed for comprehensive integration testing. The integration testing was executed on a mobile device, then once all features had been implemented a planned to run the stress testing on both mobile and tablet.

Once the app had been finished, I was able to stress test the application. The methodology I used meant that I did not require automated code tests. Due to the structure and function of the app it was better suited to being tested physically on the device rather than on an emulator.

As the app could potentially be used by the Marketing Department it is important that the app is tested on real users. To do this I created a usability questionnaire which I ask a handful of volunteers to fill in, this added another dimension of testing for the app.

## Integration Testing

In order for each feature to be promoted to the final build it had to pass a series of tests. These tests ensured that the feature being developed worked correctly as an individual component of the app. During this first phase I would check that the feature functioned how I expected; for example a topic included the correct content. I would also check that any clickable elements, such as buttons, within the app were working.

Next I would check that the new feature had not effected any other areas of the app which had already passed integration testing. For example it was important to check that I had not accidentally overwritten any files when adding new topics, or lost any content when implementing offline storage of data.

Once these series of checks had been completed I was able to promote the feature to the final build and commit my code to GitHub.

### Problems Discovered During Testing

I found one major problem during my testing phase when I came to testing the application on a tablet, it was caused by the method I had used to check whether the device was online. When the app was ran on a tablet it would immediately crash, so I checked the error logs on Android Studio and found that the Connectivity Manager was throwing an exception.

The way the method checked if the device was online was to first find the Wi-Fi connection and check whether it was connected, then to find the mobile network to see if it was connected. Due to the tablet I was using not having a mobile network the Connectivity Manager was returning null, if I had used a tablet which had a mobile network I would not have found the error.

Therefore I had to change the way in which the app checked whether the device was online so that it worked for both mobile and tablet devices. I found a method on Stack Overflow [18] from a user called QLag which I used to rectify the problem. The new method checked whether the Connectivity Manager was not equal to null before it checked the status of the mobile network.

I had to change the logic of the startup checks so that the app first checked whether there was data downloaded on the device, then if not it would check the status of the internet connection. I have included these two methods in Appendix C.

## Stress Testing

Once the app was at a stage where I felt it was finished I was able to undertake a series of Stress Tests, this is where I tested the app’s components to ensure it worked as expected. Though this stage of testing does not guaranty to find all bugs within the app it does increase the likelihood of doing so.

I undertook the stress testing on both a mobile and tablet device as it is likely that the app would be used on an array of devices. I produced a test table to record the outcomes of the tests, this is available in Appendix E.

### Testing Environments

Table 1 shows the specifications of the devices which I tested the application on.

Table . Testing Device Specifications

|  |  |  |
| --- | --- | --- |
| Testing Devices | | |
| Device | Mobile | Tablet |
| Make | Motorola | Asus |
| Model | Moto G | Nexus 7 |
| Android Version | 5.1 | 6.0.1 |

## User Testing

I undertook a usability questionnaire as part of my testing, in order to gather feedback on the app’s appearance and function. Ideally I would have liked to present the questionnaire to a group of prospective student as they are the target audience for the app, this would result in the most informative feedback. Unfortunately I didn’t have a group of prospective students to whom I could present the questionnaire and so had to use a different demographic in my testing.

The group of participants who filled in my questionnaire were all current students at Aberystwyth University, though they were not my target audience they were the next best alternative. The participants have recently been prospective students for the University and will have a knowledge of what is wanted in a prospective student’s app. Furthermore they also have a knowledge about Aberystwyth University and can then offer advice on content which may have been missed within the app.

Appendix F contains a copy of the questionnaire.

### Questionnaire Results Analysis

# Critical Evaluation

## Overall Reflection

Overall the project went much better than I was expecting. I wouldn’t consider myself a strong programmer and I had no previous experience developing an Android app, for these reasons I thought I would struggle with this project. In order to successfully complete this project I tried to do all the basic processes well, taking time to complete each phase of the process in order to produce a quality output.

The requirements phase of this project was a vital aspect as when I was given the project there were limited indications of what was required. Therefore I had to do my own research as well as meeting with the Marketing Department in order to understand what was required from the app.

I think the final list of requirements included in the app were the correct choice. During the requirements gathering phase there was the opportunity to add more features and content within the app, but I think it was important to limit the amount of requirements in order to avoid overfilling the app and making unrealistic targets for the time period. Therefore I had to select requirements which I though bought the most value to the app and its users.

My design changed and adapted throughout the project due to the methodology that I used. The final design wasn’t too dissimilar from what I had planned but I did have to respond to a few changes throughout the project which my methodology allowed me to do with ease.

I think the final design was appropriate for the app created. Due to the rigidness of the app’s structure there was a limited choice of design options for both the app’s architecture and layout. The final design which I chose balanced simplicity of use for the user as well as encouraging longevity of the app for the Marketing Department.

I choose multiple forms of testing which I think were most appropriate for the app. Gathering user feedback is a vital aspect of app development as many users will view the app differently, giving me more rounded information. The app needed to be tested on real users as it has the potential to be used be real users.Another aspect of testing which was important was stress testing, this was done once the app was complete to minimise bugs being include in the finished application.

## Tools Used During the Project

### Version Control System

Throughout my project I used GitHub to back up and version control my work. I created two repositories on GitHub which were used for different aspects of the project; one to store my code, and the other for documentation (including weekly blogs). The reason for using a VCS (Version Control System) was so I could access my project off multiple machines as well as always having a backup if I was too loose the code on my main computer.

Android Studio and GitHub are compatible and you can make use of all of GitHub’s functionality from within Android Studio. I was able to easily promote finished features to the final build as soon as they had been tested by simply clicking a button, allowing a smooth transition period between each feature’s implementation.

Another feature I made use of was being able to revert back to the last commit of code. I had to do this on several occasions as experimenting with new features sometimes broke parts of the app.

Though I did use GitHub for this project I did not always follow the standard procedures for version controlling software and there were a few features I did not make use of. This was because I only had a basic knowledge of using GitHub; pulling, committing and pushing. If I had done more reading on using GitHub during the background of the project I would have been able to use the system more effectively.

I have included a graph generated from the repository storing my code which shows the amount of additions and deletions throughout the project; Figure 8.

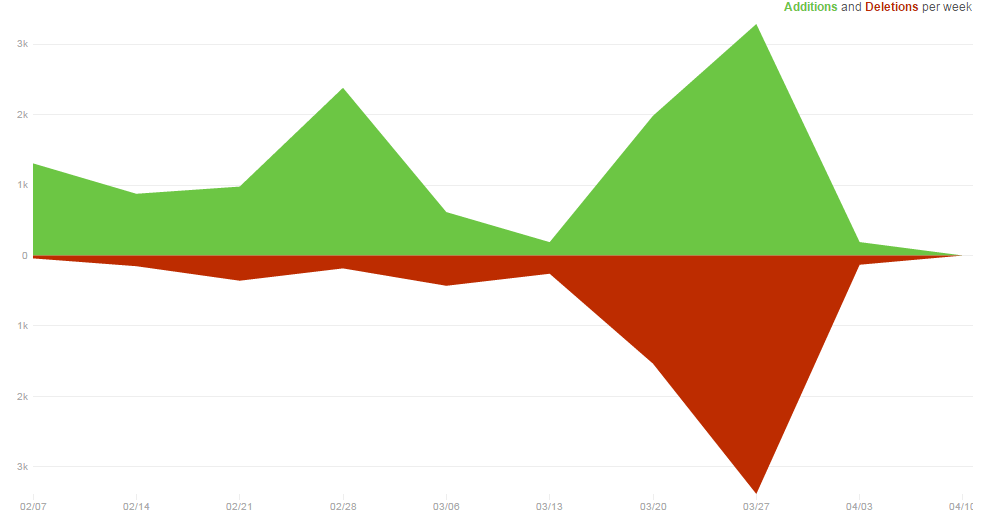


Figure . Additions and deletions per week on my code repository.

### Fluid UI

Though only used briefly, this free web based design tool helped me to create and compare potential designs. I was able to quickly produce mock ups which allowed me to evaluate a designs potential without having to produce the designs within the app.

I did find that Fluid UI was very limited and therefore I was only able to produce simple mock ups. If I wanted to produce layouts which were more complex such as a menu deign, I had to do this within my IDE which was more time consuming. Though it did allow me to develop a better understanding for XML.

### Android Studio

Android Studio was the obvious choice of IDE for this project as it is the official IDE for Android development. Alternatively I could have used another IDE with an Android plugin, but these would not include all the features of Android Studio.

Android Studio assists you when building specialised activities, by including required methods and variables within classes automatically. It also brings up suggested methods related to both Java and Android when you are programming classes and methods; giving suggestions depending on the situation you are in and what you have used previously.

As previously mentioned Android Studio integrates really easily with GitHub, allowing me to make full use of GitHub’s features from within the IDE.

One aspect of Android Studio which I didn’t make use of was running the app on an emulator. This would have allowed me to test the app on a variety of devices without having to install the app on the physical devices. Unfortunately I could not get the emulator running on my computer and had to make do with a mobile and tablet device for testing.

## Methodology

Before I started this project I decided to follow a methodology which was similar to Feature Driven Development. I used many of the core aspects of Feature Driven Development throughout the project; build a feature list, design by feature and implement by feature. I also adapted my testing so it was first performed on each feature before testing the whole app.

Another change I made to feature driven development was to remove iterations, as I don’t feel they offer any value to a single developer project. Instead I decided to work through my feature list sequentially, breaking up larger features into small manageable tasks.

Working through the project this way allowed me to track progress and manage my time accordingly. By having a clear plan of what was left within the project I was able to keep motivated as I could see the progress I had made.

## Positives Outcomes

The Android app produced at the end of the project successfully incorporated all the requirements chosen during the analysis phases. I was able to include all ten topics worth of content within the app and this content was stored locally so the app could be used even if the device was offline.

## Starting the Project Again

## Future Work

# Appendices

* 1. Third-Party Code and Libraries

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

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

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

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

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

* 1. Ethics Submission

**Ethics Application Number: 4296**

Undergraduate or PG Taught

**Your aber.ac.uk email address**

mal60@aber.ac.uk

**Full Name**

Mark Lewis

**Please enter the name of the person responsible for reviewing your assessment.**

Reyer Zwiggelaar

**Please enter the aber.ac.uk email address of the person responsible for reviewing your application**

rrz@aber.ac.uk

**Supervisor or Institute Director of Research Department**

cs

**Module code (Only enter if you have been asked to do so)**

CS39440

**Proposed Study Title**

MMP Mobile Application for the University

**Proposed Start Date**

28/1/16

**Proposed Completion Date**

4/5/16

**Are you conducting a quantitative or qualitative research project?**

Qualitative

**Does your research require external ethical approval under the Health Research Authority?**

No

**Does your research involve animals?**

No

**Are you completing this form for your own research?**

Yes

**Does your research involve human participants?**

Yes

**Institute**

IMPACS

**Please provide a brief summary of your project (150 word max)**

Creating an android application for the Marketing Department within the University. The app is aimed at

prospective students.

**I can confirm that the study does not involve vulnerable participants including participants under the**

**age of 18, those with learning/communication or associated difficulties or those that are otherwise**

**unable to provide informed consent?**

Yes

**I can confirm that the participants will not be asked to take part in the study without their consent or**

**knowledge at the time and participants will be fully informed of the purpose of the research (including**

**what data will be gathered and how it shall be used during and after the study). Participants will also be**

**given time to consider whether they wish to take part in the study and be given the right to withdraw at**

**any given time.**

Yes

**I can confirm that there is no risk that the nature of the research topic might lead to disclosures from**

**the participant concerning their own involvement in illegal activities or other activities that represent a**

**risk to themselves or others (e.g. sexual activity, drug use or professional misconduct). Should a**

**disclosure be made, you should be aware of your responsibilities and boundaries as a researcher and be**

**aware of whom to contact should the need arise (i.e. your supervisor).**

Yes

**I can confirm that the study will not induce stress, anxiety, lead to humiliation or cause harm or any**

**other negative consequences beyond the risks encountered in the participant’s day-to-day lives.**

Yes

**Please include any further relevant information for this section here:**

The participants will be asked to complete a survey on the app as part of the testing phase

**Where appropriate, do you have consent for the publication, reproduction or use of any unpublished**

**material?**

Not applicable

**Will appropriate measures be put in place for the secure and confidential storage of data?**

Yes

**Does the research pose more than minimal and predictable risk to the researcher?**

No

**Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth**

**Office advise against travel to?**

No

**Please include any further relevant information for this section here:**

**If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check.**

**Tick to confirm that you will ensure you comply with this requirement should you identify that you**

**require one.**

Yes

**Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and**

**that you will inform your department should the proposal significantly change.**

Yes

**Please include any further relevant information for this section here:**

* 1. Code Samples

**Read string from file**

private String readFromFile() {

String ret = "";

try {

InputStream inputStream = openFileInput("filename");

if (inputStream != null) {

InputStreamReader inputStreamReader = new InputStreamReader(inputStream);

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String receiveString = "";

StringBuilder stringBuilder = new StringBuilder();

while ((receiveString = bufferedReader.readLine()) != null) {

stringBuilder.append(receiveString);

}

inputStream.close();

ret = stringBuilder.toString();

}

} catch (FileNotFoundException e) {

Log.e("login activity", "File not found: " + e.toString());

} catch (IOException e) {

Log.e("login activity", "Can not read file: " + e.toString());

}

return ret;

}

**Write to file**

private void writeToFile(String result, String title) throws FileNotFoundException {

File file = new File(title + ".html");

FileOutputStream fos = openFileOutput(String.valueOf(file), MODE\_PRIVATE);

PrintWriter pw = new PrintWriter(new BufferedWriter(

new OutputStreamWriter(fos)));

pw.print(result);

pw.close();

}

}

**Check whether the device is online and whether it has data downloaded on it (Version 1)**

public void checkStatus() {

ConnectivityManager connec = (ConnectivityManager) getSystemService(Context.CONNECTIVITY\_SERVICE);

if (this.hasDataDownloaded() == false && connec != null && (

(connec.getNetworkInfo(ConnectivityManager.TYPE\_WIFI).getState() == NetworkInfo.State.DISCONNECTED) ||

(connec.getNetworkInfo(ConnectivityManager.TYPE\_MOBILE).getState() == NetworkInfo.State.DISCONNECTED))) {

//no data on device, no internet connection

textView.setText("No internet connection, unable to download content. Please enable a connection and restart the app.");

if (this.hasDataDownloaded() == false && connec != null && (

(connec.getNetworkInfo(ConnectivityManager.TYPE\_WIFI).getState() == NetworkInfo.State.CONNECTED) ||

(connec.getNetworkInfo(ConnectivityManager.TYPE\_MOBILE).getState() == NetworkInfo.State.CONNECTED))) {

//no data on device, device online

textView.setText("Downloading data...");

new JsoupAsyncTask().execute();

}

} else if (this.hasDataDownloaded() == true) {

//device has data

textView.setText("Loading..");

Intent intent = new Intent(this, MainMenu.class);

startActivity(intent);

}

}

**Check Device Has Data Downloaded**

public void hasDataDownloaded() {

File f = new File("/data/data/mark.jsouptestapp/files/downloadComplete.html");

if (f.exists()) {

textView.setText("Loading..");

Intent intent = new Intent(this, MainMenu.class);

startActivity(intent);

} else {

this.checkStatus();

}

}

**Check Device is Online (Version 2)**

public void checkStatus() {

if (this.getNetworkState(this) || this.getWifiState(this)) {

textView.setText("Downloading Data...");

new JsoupAsyncTask().execute();

} else {

textView.setText("Device must be online to download the latest data. Please enable a connection and restart the app.");

}

}

public static boolean getNetworkState(Context pContext) {

ConnectivityManager connect = null;

connect = (ConnectivityManager) pContext.getSystemService(pContext.CONNECTIVITY\_SERVICE);

if (connect != null) {

NetworkInfo result = connect.getNetworkInfo(ConnectivityManager.TYPE\_MOBILE);

if (result != null && result.isConnectedOrConnecting()) {

return true;

} else {

return false;

}

} else

return false;

}

public static boolean getWifiState(Context context) {

ConnectivityManager connect = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY\_SERVICE);

if (connect.getNetworkInfo(ConnectivityManager.TYPE\_WIFI).isConnectedOrConnecting()) {

return true;

} else {

return false;

}

}

* 1. Similar Apps used by other institutions

**Teeside University**

* Downloadable PDFs

**Swansea University**

* Downloadable PDFs
* Menu with links to website (not in the app)

**Bristol**

* Doesn’t work

**University of South Wales**

* Featured news articles
* View subjects/courses
* Link to website to book open day

**Essex University**

* About Essex University
* About Essex
* Open days
* Courses
* Campus maps
* Directions to get there

**Greenwich University**

* About Greenwich University
* Student Life-Accommodation, SU, Sports and Social clubs
* Studying at Greenwich-department/courses
* Campus Maps

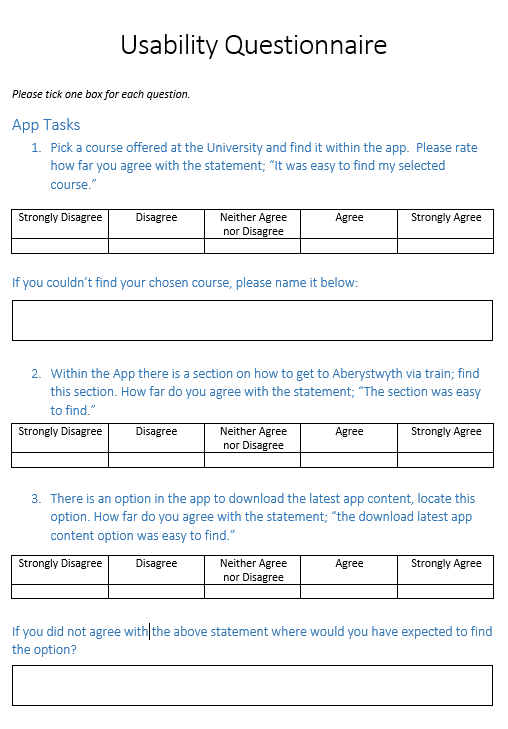
**Stirling University**

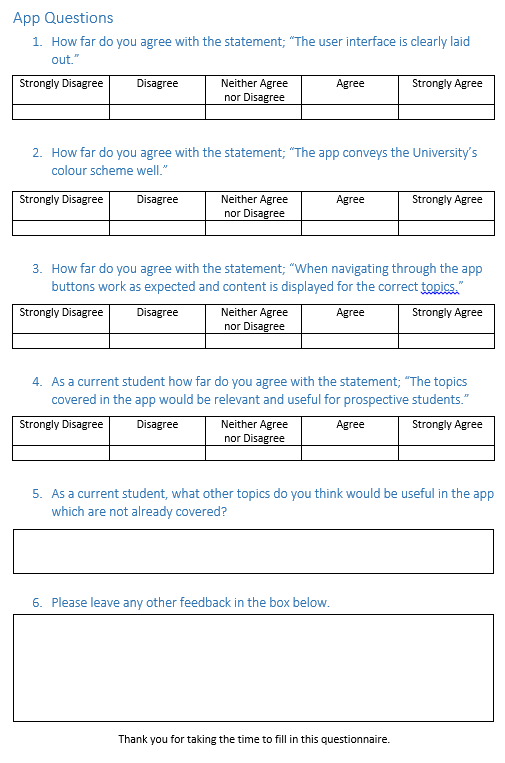
* Stirling University Magazine PDF download
* Courses
* Maps
* Scholarships and funding
* Find us
* Open days

**Keele University**

* What Keele has to offer?
* Courses
* Visit/open days
* Maps
* Key contacts
* Social Media
  1. Test Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case | Test | Device Environment | Input | Expected Output | Mobile Pass/Fail | Tablet Pass/Fail |
| 0.1 | First time launch | Device offline, no content on device |  | Error message on screen asking user to enable a connection an restart app. | Pass | Pass |
| 0.2 | First time launch | Device online, no content on device |  | App should start downloading content. | Pass | Pass |
| 1.1 | Non-first time launch | Device offline, content on device |  | App should show menu screen | Pass | Pass |
| 1.2 | Non-first time launch | Device online, content on device |  | App should show menu screen | Pass | Pass |
| 1.1 | Undergraduate courses button | Main Menu | Click undergraduate courses button | List of departments should be shown | Pass | Pass |
| 1.2 | Undergraduate departments list view | Undergraduate departments page | Click a department | List of courses should be shown | Pass | Pass |
| 1.3 | Undergraduate course list view | Department courses page | Click a course | Course detail page shown, with course title in toolbar. 5 tabs present, first tab selected. | Pass | Pass |
| 1.4 | Undergraduate course content tab | Course details page displayed on screen | Click course content tab | Content displayed on screen | Pass | Pass |
| 1.5 | Undergraduate employability tab | Course details page displayed on screen | Click employability tab | Content displayed on screen | Pass | Pass |
| 1.6 | Undergraduate teaching tab | Course details page displayed on screen | Click teaching tab | Content displayed on screen | Pass | Pass |
| 1.7 | Undergraduate student views tab | Course details page displayed on screen | Click student views tab | Content displayed on screen | Pass | Pass |
| 1.8 | Content saved when switching between tabs | Course details page displayed on screen | Click through tabs 1-5, then back through tabs 5-1 | Content should be displayed on screen for each tab. | Pass | Pass |
| 2.1 | Postgraduate courses button | Main Menu | Click on Postgraduate courses button | List of departments should be shown | Pass | Pass |
| 2.2 | Postgraduate departments list view | Postgraduate departments page | Click a department | List of courses should be shown | Pass | Pass |
| 2.3 | Postgraduate course list view | Department’s courses page. | Click a course | Course detail page shown, with course title in toolbar. 5 tabs present, first tab selected. | Pass | Pass |
| 2.4 | Postgraduate further information tab | Course details page displayed on screen | Click further information tab | Content displayed on screen | Pass | Pass |
| 2.5 | Content saved when switching between tabs | Course details page displayed on screen | Click between key facts and further information tabs. | Content should be displayed on screen for each tab. | Pass | Pass |
| 3.1 | Scholarships button | Main Menu | Click scholarships button | Scholarships and funding page displayed | Pass | Pass |
| 3.2 | UK & EU students button | Scholarships page | Click UK & EU students button | Display a list of scholarships | Pass | Pass |
| 3.3 | Students outside of the EU button | Main Menu | Click Students outside of the EU button | Displays a list of scholarships | Pass | Pass |
| 4.1 | Accommodation button | Main Menu | Click accommodation button | Displays list of accommodation | Pass | Pass |
| 4.2 | Accommodation list view | List of halls of residence page | Click a halls of residence | Displays the accommodation title in toolbar. Displays 4 tabs. | Pass | Pass |
| 4.3 | Accommodation facilities & services tab | Halls of residence details page | Click facilities & services tab | Displays content | Pass | Pass |
| 4.4 | Accommodation what’s in your room tab | Halls of residence details page | Click what’s in your room tab | Displays content | Pass | Pass |
| 4.5 | Accommodation fees tab | Halls of residence details page | Click fees tab | Displays content | Pass | Pass |
| 4.6 | Content saved when switching between tabs | Accommodation details page displayed on screen | Click between tabs 1-4. | Content should be displayed on screen for each tab. | Pass | Pass |
| 5 | Students union button | Main Menu | Click students union button | Students union content displayed on screen | Pass | Pass |
| 6 | Sports & society’s button | Main Menu | Click sports & society’s button | Sports & Society’s content displayed | Pass | Pass |
| 7 | Open Days’ button | Main Menu | Click open days’ button | Open days’ content displayed | Pass | Pass |
| 8 | Virtual open days’ button | Main Menu | Click virtual open days’ button | Virtual open days’ content displayed | Pass | Pass |
| 9.1 | Getting here’s button | Main Menu | Click getting here button | Getting here page displayed. With 2 tabs. First tab’s content displayed. | Pass | Pass |
| 9.2 | Train tab | Getting here page | Click train tab | Content on train travel displayed | Pass | Pass |
| 10.1 | Options menu button | Main Menu | Click options button | Options list displayed | Pass | Pass |
| 10.2 | Download latest data button | Device offline | Click download latest data button | Error message on screen asking user to enable a connection and restart app. | Pass | Pass |
| 10.3 | Download latest data button | Device online | Click download latest data button | Splash screen displayed, informing user data is downloading | Pass | Pass |
| 11 | Minimising and resuming app | Tested on each page in the app. | Minimise app and resume it | Page should show as it did before being minimised. | Pass | Pass |
| 12 | Android Back button | All pages | Click button in android options at the bottom of the screen | Pages parent should be shown.  Menu shouldn’t allow use of back button. | Pass | Pass |
| 13.1 | Deleting app data from options | Device offline | Load app | Error message should be displayed telling user to enable a connection and restart the app. | Pass | Pass |
| 13.2 | Deleting app data from options | Device online | Load app | App should start downloading data. | Pass | Pass |

* 1. Usability Questionnaire

****

# Annotated Bibliography

|  |  |
| --- | --- |
| [1] | Google, “Android Developers,” [Online]. Available: http://developer.android.com/index.html. [Accessed March 2016]. |
| [2] | A. University, “Online Virtual Open Day,” 2016. [Online]. Available: https://www.aber.ac.uk/en/undergrad/visit-us/virtual-open-day/. [Accessed March 2016]. |
| [3] | Nub Games Inc, “LibraryH3lp Homepage,” 2016. [Online]. Available: https://libraryh3lp.com/. [Accessed Febuary 2016]. |
| [4] | Laxman, “Building your own android chat messenger app similar to Whatsapp using XMPP,” 2015. [Online]. Available: http://www.tutorialsface.com/2015/08/building-your-own-android-chat-messenger-app-similar-to-whatsapp-using-xmpp-smack-4-1-api-from-scratch-part-1/. [Accessed Febuary 2016]. |
| [5] | J. Hedley, “Jsoup Homepage,” 2015. [Online]. Available: http://jsoup.org/. [Accessed January 2016]. |
| [6] | Adobe Systems Inc., “PhoneGap Home Page,” 2016. [Online]. Available: http://phonegap.com/. [Accessed January 2016]. |
| [7] | Roadfire Software , “When to use PhoneGap versus developing a native iOS app,” 2015. [Online]. Available: http://roadfiresoftware.com/2014/04/when-to-use-phonegap-versus-developing-a-native-ios-app/. [Accessed January 2016]. |
| [8] | IDC Research Inc., “Smartphone OS Market Share, 2015 Q2,” August 2015. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed January 2016]. |
| [9] | Google, “Storage Options,” 2016. [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html. [Accessed Febuary 2016]. |
| [10] | A. U. A. Department, “Types of Accommodation,” 2016. [Online]. Available: https://www.aber.ac.uk/en/accommodation/types-accommodation/. [Accessed Feburary 2016]. |
| [11] | Aberystwyth University, “The University Brand Manual 2013-2014,” 2013-2014. [Online]. Available: https://www.aber.ac.uk/en/media/departmental/dwpps/brandidentity/university-brand-manual-2013-14.pdf. [Accessed Febuary 2016]. |
| [12] | Fluid, “Fluid UI Homepage,” 2016. [Online]. Available: https://www.fluidui.com/. [Accessed Febuary 2016]. |
| [13] | Android Arsenal, “Android Arsenal Homepage,” 2016. [Online]. Available: http://android-arsenal.com/. [Accessed Febuary 2016]. |
| [14] | R. -. C. Theory, “Creating Android Sliding Sidebar (Hamburger) Menu with Navigation Drawer Icon,” 1 Febuary 2015. [Online]. Available: http://codetheory.in/android-navigation-drawer/. [Accessed Febuary 2016]. |
| [15] | R9J, “How To Read/Write String From A File In Android,” January 2013. [Online]. Available: http://stackoverflow.com/questions/14376807/how-to-read-write-string-from-a-file-in-android. [Accessed Febuary 2016]. |
| [16] | Tim, “How can i check whether an android device is connected to the web?,” May 2010. [Online]. Available: http://stackoverflow.com/questions/2789612/how-can-i-check-whether-an-android-device-is-connected-to-the-web. [Accessed Febuary 2016]. |
| [17] | Voicu, “How to create layout with 6 buttons like windows tiles,” May 2013. [Online]. Available: http://stackoverflow.com/questions/16466914/how-to-create-layout-with-6-buttons-like-windows-tiles. [Accessed March 2016]. |
| [18] | QLag, “Android check network connectivity on some tablets crash,” September 2012. [Online]. Available: http://stackoverflow.com/questions/12376247/android-check-network-connectivity-on-some-tablets-crash. [Accessed April 2016]. |