An iOS application to check river levels for kayakers

Final Report for CS39440 Major Project

Author: Ieuan Tudur Peace (itp9@aber.ac.uk)
Supervisor: Dr. N. A. Snooke (nst@aber.ac.uk)

21 April 2013
Version 1.5 (Release)

This report is submitted as partial fulfilment of a BEng degree in Software Engineering (G600)

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

Declaration of originality

In signing below, I confirm that:

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

Signature ……. (Your 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.

Signature ……. (Your Name)

Date ……

Acknowledgements

I am grateful to Dr Neal Snook for being my supervisor and guiding me through the different tasks that was needed to complete this project. I would also like to thank Prof Chris Price and Neil Taylor for running the iOS development course that I attended.

I'd like to thank Thomas Salu (tas9) who is a iOS developer that has given me advice and help throughout the project.

Also I would like to thank Barbara Jones from student support and Sue Peace for proofreading my dissertation.

Finally I'd like to thank my family and friends for supporting me throughout the project and for helping me with the User Testing.

Abstract

To create a mobile application that assists kayakers in selecting a suitable river. The application will be developed for iOS devices taking advantage of the features on a device that runs iOS.

When the application is finished a user will be able to select a river, see its current river level and information about the river. Using the frameworks and technology provided by Apple Inc on their iOS devices the user will view their current location and get directions to the river. Third party frameworks will be used to allow users to comment on each river and view the comments of other users.

An Agile approach will be taken for working on this project. This includes feature driven development and following principles from the Agile Manifesto.

Contents

1. Background & Objectives

Kayakers need knowledge of river levels to make an informed decision of whether it is worth traveling to a river and the location of that river. In the UK almost all rivers do not have enough water to kayak down unless it rains and the rivers will stay in condition for 6 to 12 hours after it stops raining. For a kayaker this makes planning a trip to the river when it is at the correct level difficult. If the kayaker is not in the area close to the river then he will not know what is happening regarding rain and the current level of the river.

Online, there are detailed current weather forecasts for the UK. River level data is available through the Environment Agency's website, as well as flood warnings. A kayaker who has not done much kayaking in a specific region of the country may not know how the rain affects the river levels there. This project was initialized with the idea of calibrating that data. On the web there are already websites that calibrate the data from the Environment Agency to get a user friendly river level.

Welsh Rivers; www.welsh-rivers.co.uk is a simple website. It has a list of rivers showing whether they are empty, low, medium, high or in spate. It has a description and basic map with the locations of the rivers.

Rainchasers; www.rainchasers.co.uk is a user friendly calibrated river level website. There is a detailed map showing the rivers as annotations and displaying their levels. It also has a table of rivers listed in order of their levels from highest to lowest. This site also has a page for each section of river with the river level, grade and a short description. However it lacks the powerful features that a mobile application can bring, such as the directions from the user's current location

www.paddleguide.org/levels This is a website which is designed for mobile devices. It is very basic showing the levels of the rivers and, if requested, what the levels are calibrated for being low, medium and high. This site does not show any other information to help a user to decide what to paddle.

Around Me is an application where the user can find information about businesses and services around them. When they have selected a business or service the application will provide directions from the user's current location. It is available on Apple's App Store. An example of its use would be a user wanting to find the closest public house. The user selects the type of service needed from a table. In this example, a public house, then a list of all the closest public house is displayed in which there is a button to show a map. On the map there are annotations with the public houses displayed. From the list or the map it is possible to get the directions.

The purpose of Around Me is not the same as the objectives application of this project. The technology used and the design of the application is similar to the technology in this project. The application will be using the same frameworks for the maps as well as a similar navigation system.

The overall objective of the project was to create an iOS application to assist kayakers in the selection of a suitable river to kayak. It gives them directions to the river and forecasts about what the river levels are expected do in the next three days. This is a list of the objectives for the project:

- List of rivers, with the river levels displayed as either: low, medium, high.
- Detailed page per river, with a description, river level and a map showing the location
- Directions to the river
- Comments so users can alert other users of dangers for each river
- Forecasts for what rivers will rise in the next three days

This project was to create a powerful application that kayakers can use to help them go to and from rivers without the disappointment of arriving and finding them too low or high.

An application designed for iOS devices does have its constraints and limitations. The good rivers for kayaking in the UK are often in remote locations which do not have good phone signal and no internet signal. However there is a push by the government to extend the area that is covered by 3G.

2. Development Process

This section describes the methodology and the tools used to develop the application. The section is split into two subsections, one discussing the methodology used and the other explaining what tools were used and why.

2.1. Methodology

An Agile methodology was chosen as the best development process for this project. The reasons for this were its flexibility and how it creates a sustainable development environment. There is no need to create detailed technical plan or designs hence leaving more time for development. Also Agile allows changes in the requirements late on in the development stage.

Another methodology that was considered was Waterfall; it was not chosen because of its long design process. Waterfall is not a very flexible methodology as it does not allow a change in requirements over the course of development. With Waterfall software development happens late on in the life cycle, leaving less leeway for development.

This project was different from the usual types of projects that use Agile and the type of project that Agile was designed for. As it was a one man project there was no need for Scrum meetings or daily meetings with business people. It was possible to use some of the principles like constant refactoring and working software over comprehensive documentation.

Principles of the Agile Manifesto used during this project were:

- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." [1]
- "Working software is the primary measure of progress." [1]
- "Continuous attention to technical excellence and good design enhances agility." [1]
- "Simplicity – the art of maximizing the amount of work not done--is essential." [1]
- "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." [1]

Out of the twelve principles it was these five that were most relevant to the project. There were some core principles associated with Agile that were not used. Test driven development was not used nor were iterations. This is due to the nature of the project involved working with new technology.

User stories were used in the development process because they helped in showing what needed to be done.

During developing the application it was easy to stray away from some of the Agile principles and take a 'hacking' approach to developing the work. This means the project was straying away from the Agile principle of "Continuous attention to technical excellence and good design enhances agility". One reason for straying away from Agile methodology was because of the need to learn and use technology. Another reason that the project strayed away from the Agile principles was a lack of experience using it as a methodology. The key principle which was followed was "Working software over comprehensive documentation". [1] This meant that priority could be given to the development of the application.

2.2. Implementation tools

Developing an iOS application limits what type of IDE and language can be used. This is quite understandable as the application has to run on a specific operating system, iOS 6. This section lists the types of technologies that were used during the development of this project.

2.2.1. Xcode

Xcode is the only IDE available for developing iOS applications. It is provided free by Apple. It has all the native software development kits (SDK's) for iOS preinstalled. Also it has the features for developing the user interface (UI) with the Interface Builder.

2.2.2. Objective-C

iOS, and most development for Apple, uses Objective-C. It is an object oriented language that is quite simple to learn. Objective -C is merely an extension of standard ANSI C. Objective-C was originally developed by the Stepstone Company in the 80s. After this it was licensed by NeXT which was acquired by Apple Inc. Apple used Objective-C to develop their operating systems in the 90s. Objective-C was naturally the language that Apple mobile devices were going to inherit.

When developing an iOS application there was only one choice of language; Objective-C. Online there are many different tutorials as well as the comprehensive documentation from Apple, easing development. [2]

2.2.3. Software development kit's

Apple provides many Frameworks and Libraries; these are all available through Xcode. The application used frameworks by having a map in the application, a user interface as well as libraries to handle SQL queries.

The application has had to use a non Apple framework. This was used in conjunction with the Parse technology.

2.2.4. Parse

Parse is a 'Cloud' that provides a space to store data. It means there is no need to have a server for an application. It has a framework and is

compatible with iOS. If there are less than 1 million pushes a month then it is a free service as well as 1GB of free storage. The application in its development stages will not reach anywhere near this amount of data pushes or need more than 1GB of storage.

2.2.5.Java and Jsoup

The application had to have a 'scraper' to get the river levels from the Environment Agency's website. Using Java it was possible to make a backend web service that scrapes data on given rivers and makes a Jason file which is stored online on the Aberystwyth University webspace. To do this it is using a Java library Jsoup. It is a library which can scrape HTML and specific tags from a webpage then put them in an array. Jsoup is licensed under the MIT license.

2.2.6.GitHub

The project needed to have version control system for development. There are many different version control system that are readily available free. GitHub and SVN were considered. GitHub was chosen due to its ease to set up on a Mac OS X, its ease to use, look at old code and see the changes in-between versions of the product. An account is required to use GitHub; this is free for students.

2.2.7.forecast.io

This is a weather forecasting website which provides data to developers. The first 1000 API requests a day are free. It provides a detailed forecast for any location in a JavaScript Object Notation (JSON) format. They are other free forecasting API available but from research this looked like the best. It calibrates 16 weather forecasting sources such as the Met Office Datapoint API and Worldwide METAR (Meteorological Terminal Aviation Routine Weather Report or Meteorological Aerodrome Report) weather reports.

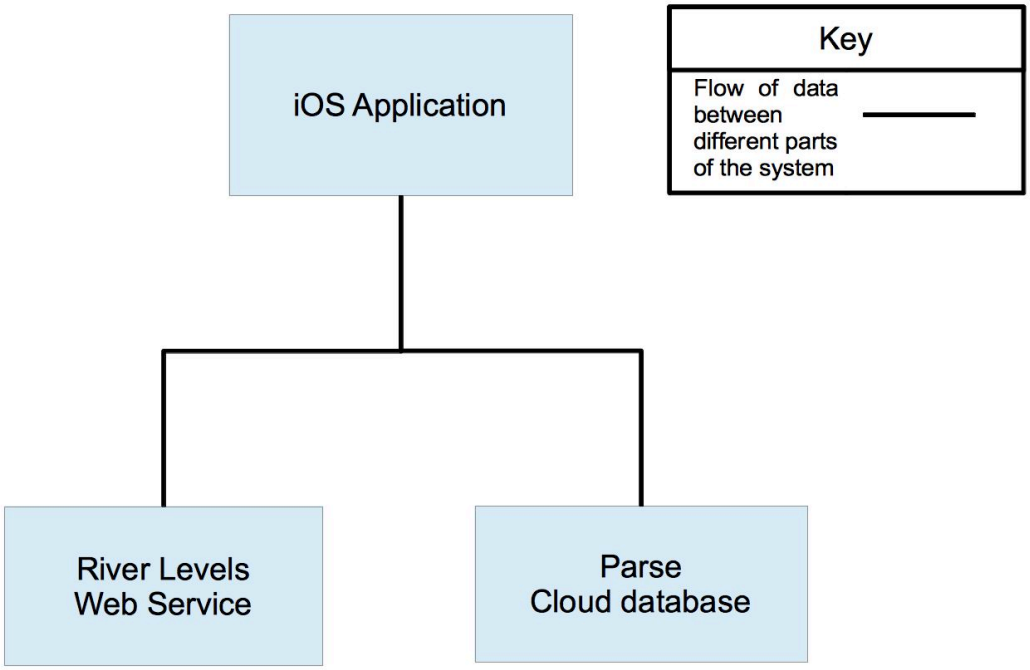
3. Design

This project was implemented on an Agile approach. There was not a large upfront design but it did stray away from the Agile methodology. The reason for this was because an application on iOS had not been made before or an application of this scale.

This section has four sub sections describing the different aspects the architecture of the application, data storage, in-depth detailed design and the user interface design.

3.1. Overall Architecture

Initially the application was not going to rely on any backend program or web service. This design was based on an ideal scenario where the application would receive the data directly from the Environment Agency. It is not an ideal situation and not having a backend web service would have made it far more complicated.



The image above shows the overall architecture of the application. Below is a description of what they do and why they were chosen:

3.1.1. The iOS application

iOS was chosen at the beginning of the project as the platform the application was going to be designed for. Android was considered, but it is quite difficult to build an application that is compatible with all Android devices. The decision to make the application compatible with all devices which run iOS 6 was based on the knowledge that most iOS users run iOS 6. [3] A key feature of the application was to give the user directions from

their current location. This is only possible if the device has GPS or connected to Wifi.

The iOS application is the front end and is what the end user will interact with and use to judge the program. This is the core of the program; it provides all the features for the user.

The application is going to store the data about the rivers internally and when the device is offline the data can be accessed. As stated previously the users (kayakers) will often be in locations where there will be no connection to the internet. The disadvantage of this is when there are updates to the data stored on the device. In this situation an update of the application is the only way to change the data stored on the device.

Core Data was considered as it is the data management framework provided by Apple. It is based on the Model View Controller pattern. SQLite was chosen for the data management in the application. Core Data was too complicated for this project because all the data is stored in one table and it would have taken longer to understand how to use Core Data compared to SQLite. Time was better spent learning technologies that can be used to implement functionality. [4]

3.1.2.River Levels Web Service

This is the 'scraper' and it was added later in the design stage due to the Environment Agency's charges for river level data. The web service runs separately to the iOS application. The only connection to the iOS application would be the transfer of data. The river levels data is stored in a JSON file.

To make this web service Java or Python were considered. For both of these there are many third party libraries available for free and almost all are open source.

Python has a large range of third party libraries and frameworks available for development. 'Scrapy' was a reason why Python was considered to develop the web service. It is one of the most powerful web scrapers and is fast at scraping HTML. [5] Python was not chosen due to a lack of experience developing with it.

Java has a third party library, Jsoup which is an HTML parser. It can scrape HTML from a given URL which is the core functionality of the web service. Jsoup was the easiest library to implement and due to having experience programming with Java this is why it was chosen to build the web service.

3.1.3.Parse

The use of Parse technology was only added in the later stages of development. This part of the application could have been stored in a JSON file stored online or using the Facebook SDK.

Having the comments stored in a JSON file would have been the safest option to store data as the application would not be dependent on third party technology. If the company running Parse was to go bankrupt then

the application would not work. Using JSON would have complicated the code as well as the backend data management.

Using Facebook's SDK the application could have integrated with some of its features. The advantage of using Facebook SDK means that a user could log on with their Facebook account and the comments would be visible, not only on the application, but also on Facebook. A user could 'check in' to a river.

Parse controls the backend data management and it also can connect the application to social networks. Parse cuts the development work due to its SDK. At the back end all that is needed is to add the relevant classes.

3.2. Data storage Design

XML files are easily read by iOS applications. Apple have provides a library which can parse the files and there are other libraries available. There are two sort of libraries; a SAX parser or a DOM parser. SAX parsers notify your code as it traverses the XML tree. A DOM parser stores the entire file in memory and then you can query it and is the simplest to use. TBXML library has been designed as a fast, lightweight DOM parser which only reads XML. Other options are the NSXMLParser which comes with iOS SDK or XMLTouch which is a DOM parser which is read only and supports XPath. The NSXMLParser is slower than the TBXML due to having features such as the ability to write to a file as well as being more complex to use than a DOM parser. If the application used XML files then it would use the XMLTouch library. It is simpler to develop with compared to TVXML due to it supporting Xpath and the files do not contain that much data so speed will not be an issue. [6]

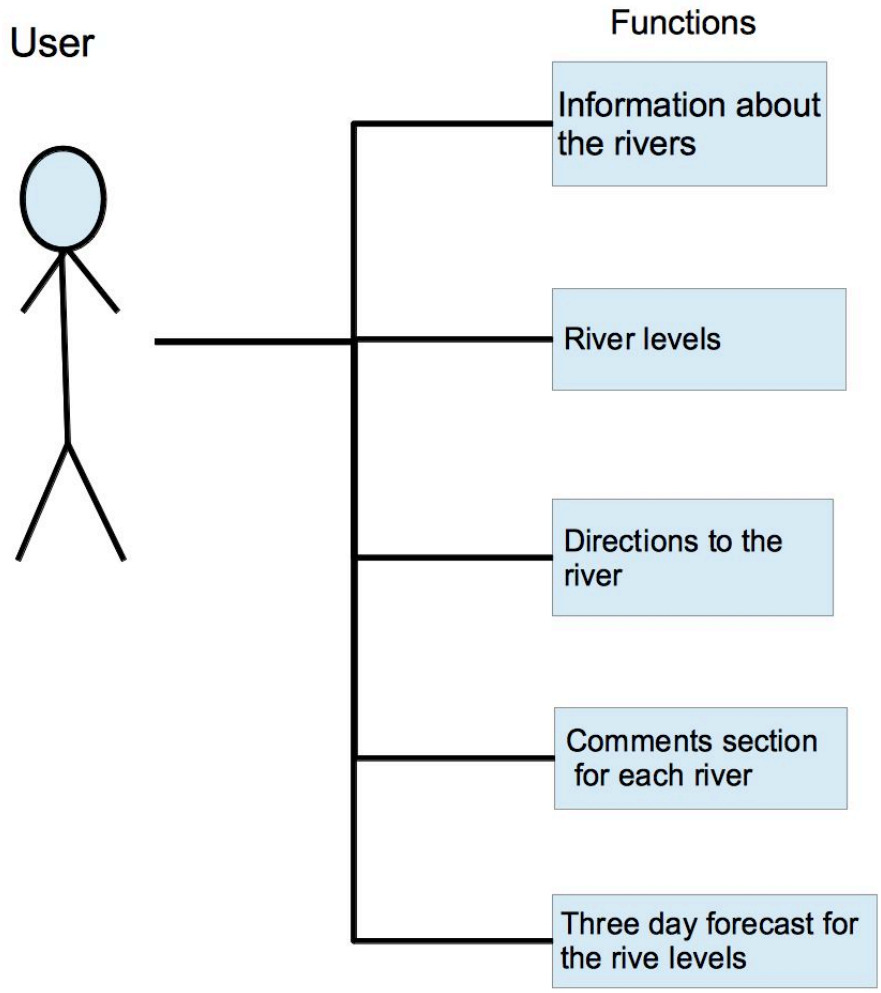
JSON is a lightweight data-interchange format. It can store data as numbers, strings, booleans, arrays or objects. JSON uses less memory and is faster to read than XML files. Apple provides a class named NSJSONSerialization which converts JSON files into objects. Using JSON files would let the application transfer data in the simplest way possible. The application will be transferring data, not marked up text. The data file will not need to be extendable. As it will not be receiving any of the data from an external system it will be safe not to have to validate the data type which is something that JSON does not provide but XML does provide type validation. It could save all the data in a JSON file rather than have a database but a database would be more flexible and simpler to maintain. [7]

IOS provides two different libraries which can extracted data from a database. CoreData is the highest level data model framework which is built on the Model View Controller architecture. The other library is SQLite which is simple and easy to use. It treats databases as a flat file therefore you do not have to have a web service running. The application will be using SQLite to accesses data as it will not have a large complex database. SQLite is more compact compared to CoreData making it more suitable for this application. If the database needed to be used on another platform, using SQLite would be more compatible. [8]

3.3. Detailed Design

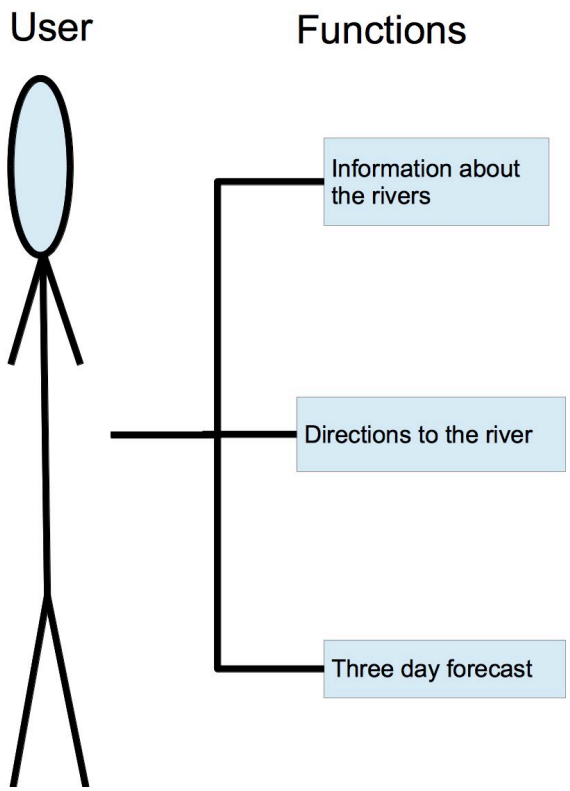
Designing the iOS application was the largest section of the design. As the project was built using Agile principles the design changed during development. The design of the flow of the program did not have any radical change throughout the development. However implementation of the design radically changed through development.

When designing the application case diagrams were used and were beneficial to figure out the logic behind the flow of data.

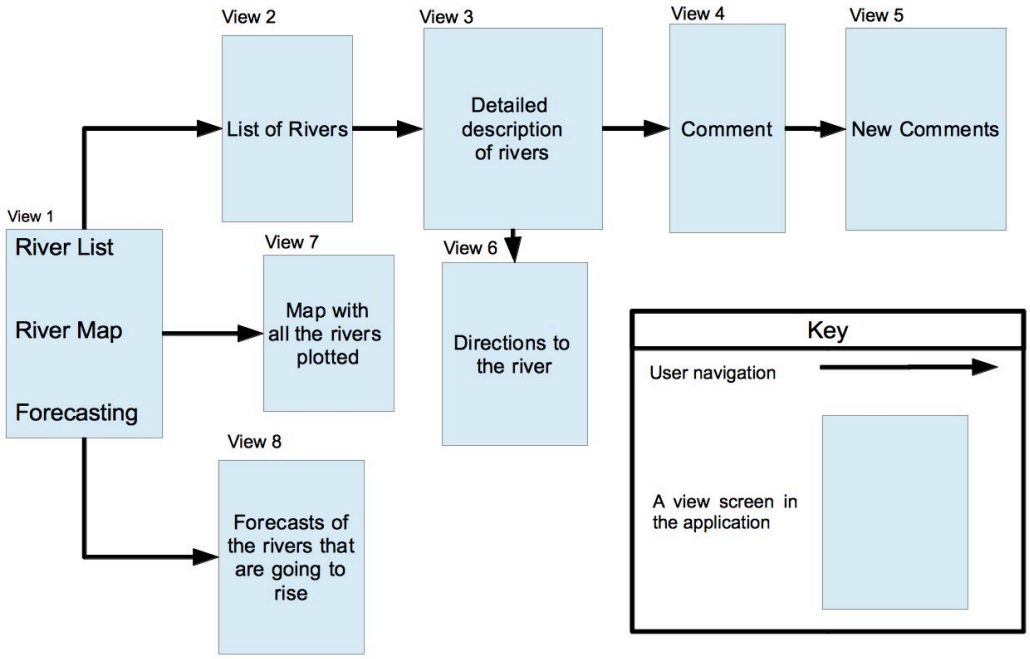


All the interactions that the user has with the application are shown in this use case diagram. Some of the use cases in the diagram above can be put under the same category. If the flow of the application was the same as the use case diagram above the flow would be cluttered. The diagram above shows all the general behaviour of the application.

Some of the use cases in the diagram above can be linked meaning that the use cases can be narrowed down.



This use case diagram is a simpler version of the first diagram. It shows the three use cases that the user wants from the application. As there is only one type of user this is the use case which the application is based on. The first version design of the application was taken from this.



The most important design of the application was to have a logic flow between views. Views are a single screen in an application. Above is the basic storyboard, showing all the views. It also shows the basic flow which

the user follows. The user should be able to navigate back and forth through the views. In the navigation bar there is a navigation button taking them back.

View 1: A simple static table with all three of the options that the user can choose.

View 2: A dynamic list of rivers taken from a database, each cell has the river name and the section of the river's name. The user can choose a river and it then takes them to a view with more details.

View 3: This view has a more detailed description of the river. It displays the river name, section name, current river levels that come from the backend web service. It also displays a map with the user's current location and river's location plotted as an annotation. There is also a button taking the user to the comment section and another to a view that has the directions to the river.

View 4: Another view with a table containing all the comments about a section of the river. Each cell has only one comment. The comments are stored using Parse.

View 5: A text box in which the user can write a comment about a river. It creates an object and sends the comment as an object to Parse.

View 6: The view shows the user's current location and the get on river location on a map. On the map there is a route from the user's current location to the river get on location. When the user travels to the river it gives them turn by turn directions.

View 7: This is a view containing a map with the user's current location as well as annotations plotting all the rivers. When a user chooses an annotation it shows the name, section and grade of the river.

View 8: Every day this view contains a list of all the rivers which are predicted to rise. Also a three day text forecast that is calibrated with the current river levels to estimate the river levels for the next three days.

3.3.1.More Detail Design of the Forecasting

Predicting if a river will rise or fall is difficult as to do this requires the river level and the weather forecast for the area around the river. The level the river will rise to depends on the height of the river, the amount of rain forecast and ground saturation. The data needed to predict how much the river is going rise is individual for each river, some rivers rise faster than others.

To make the algorithm which is going to forecast the river level a minimum amount of rain is needed to make it rise to a level which is possible to kayak from any river level. For example 25 mm of rain will make most rivers rise to a level which is possible to kayak. Then a level of the river needs to be established where 15 mm of rain will bring it to a level which it is possible to kayak. Some rivers will rise into condition from 15 mm of rain. After this each river will need to have a level in which it will come into condition from 10 mm of rain and 5 mm of rain.

There are two ways of creating this forecasting model. The first would be to create a backend web service which posts the rivers which are predicted to rise into an JSON file. The advantages of this would be less processing needed on the device making the application work quicker. The other option is to make the algorithm run in the iOS application.

Logically the algorithm is quite simple. It compares the forecast for the next 24 hours with the current river level using the data needed for the rivers to rise.

The weather forecast could be 'scraped' from a weather forecasting website or use a weather forecasting API. Using a 'scraper' would be bad practice, especially as there are free APIs for weather forecasts.

Forcast.io is a free API which provides the weather forecasts' data in a JSON format. The first 1000 requests per day are free which is suitable for this project. [9] To use this API the application would need to send the API key, which is available once registered, and the latitude and longitude of the location for which the forecast is needed. This returns a JSON file with the forecasted precipitation which can be used to compare with the river level and calculate what the river levels will do in the next 24 hours.

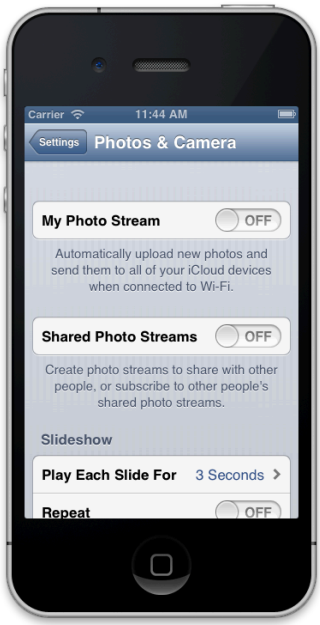
3.4. User Interface Design

In Xcode there is the Interface builder that supports Storyboard. It is a drag and drop environment. The elements in the Graphical User Interface (GUI) are connected to the code. Using Storyboard allows the developer to maximize their time developing functionality and not worry about the GUI. These tools help with making changes to the GUI.

Apple have guidelines for designing the user interface. It is important for applications to follow these guidelines. The user interface is one of the most important aspects of the application. A poor user interface could confuse a user leading to them not using the application.

The UIKit framework that is provided by Apple includes user interface elements. Almost all applications use elements that are provided by this framework therefore it is important to make the elements behave as they do in other applications. If the application's user interface did not follow the norm then users would get confused and frustrated. [10]

3.4.1.Example User Interface



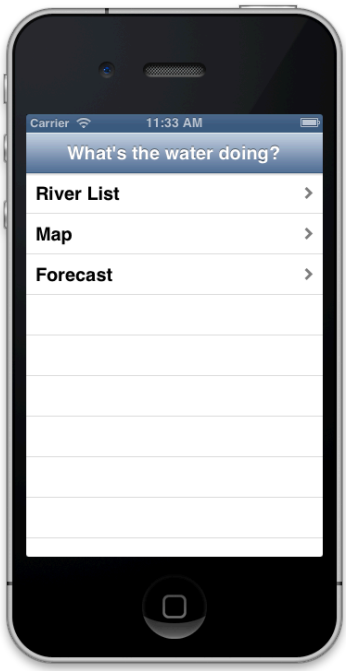
The screenshot above is a view from the settings application that is on iOS devices. It is a good example of a clear design with all the elements working as the user expects.

Importantly the layout follows convention. At the top there is a navigation bar with a navigation item that goes to the previous page. This is at the top of the application and always remains in place regardless of any scrolling.

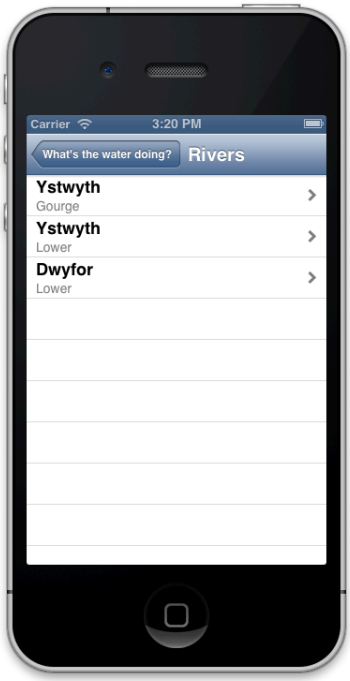
There are two types of elements in this view. The view is made up of tables and cells. In some of the views there is a switch. Normally switches are used to turn something on or off as in this application. A user would know how to use a switch making it an effective way to change a boolean variable through the user interface. The 'Play Each Slide For' of the cells is linked to another view. This is apparent to the user due to the arrow.

3.4.2.Detailed User Interface Design

This section will include an example layout of some views as well as a description of the logic behind the layout.

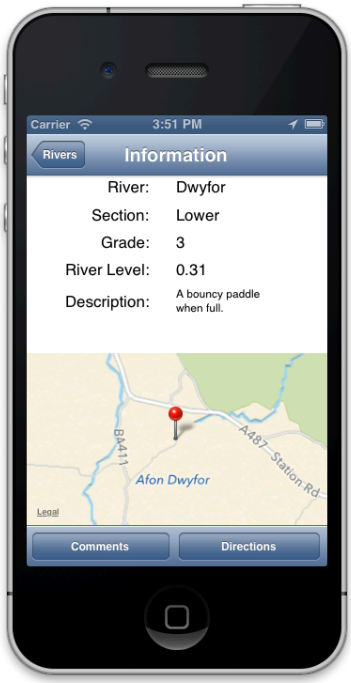


The first view gives the user a choice. It does this by following the convention of navigation in applications. There are three static cells in a table each linked to separate views. The view changes when the user presses on one of the cells. Originally it was planned to have three buttons for navigation but the table cells give the user a better feel of the logical flow of the application.



Above, is an example design of the view which is linked by the 'River Level' cell in the first view. It contains a dynamic list of rivers that are pulled from a database. Each cell has a title and a subtitle. In the title part of the cell is the river name and in the subtitle the name of the river section. Designing the layout of this page follows the convention that the rivers are all grouped in a table as they are related. At the top of the

display is the navigation bar which allows the user to go back to the previous page.



This is the view which is linked to each river cell in the 'List of Rivers' view. This is designed to be a simple view with one purpose to give the user useful information about the section of river. Above the mock-up shows the information in a logical manner as well as the map with an annotation of the location to get on the river.

Navigation from this view is obvious because at the top left corner there is a navigation item which takes the user to the previous views. On the bottom is a bar with two buttons. One of these takes the user to the comment section for that river and the other gives the user directions to the river, from their current location. Buttons are better in this situation rather than cells, especially for directions. A cell would imply that the next view is going to have a table.

3.4.3.Orientation of the application

iOS applications can orient the device in a upright or sidewise position. If the application was going to be compatible in a landscape orientation then it would be important to factor this in with the design. Advantages of having the application compatible with the landscape view would be showing the map in a different angle. The difficulty in making the application compatible in a landscape view is that every screen needs to be designed with the landscape view in mind. If the screens are not designed with landscape in mind then the application will not be user friendly. As there is no great advantage in having the application in a landscape view, changing the orientation of the application will be disabled.

3.4.4.User Interface on different iOS applications

As an iOS application is available on the App Store for any iOS device it was important to consider how it would look on different devices. The device where there would need to be a change in the way the application looks is the iPad. An iPad's screen is 9.7 inches diagonally [11] compared to an iPhone 5's of 4 inches diagonally [12]. The difference in screen size is large, over double, so building an application for an iPhone will not look the same on an iPad.

This application is going to be targeted for iPhones. The application will be used by kayakers who are on the move with their iPhone but not an iPad. Also having 3G on an iPad is optional and without 3G it would be difficult to use the application on the move. Time would be better spent on developing features rather than an iPad version of the application.

4. Implementation and Issues

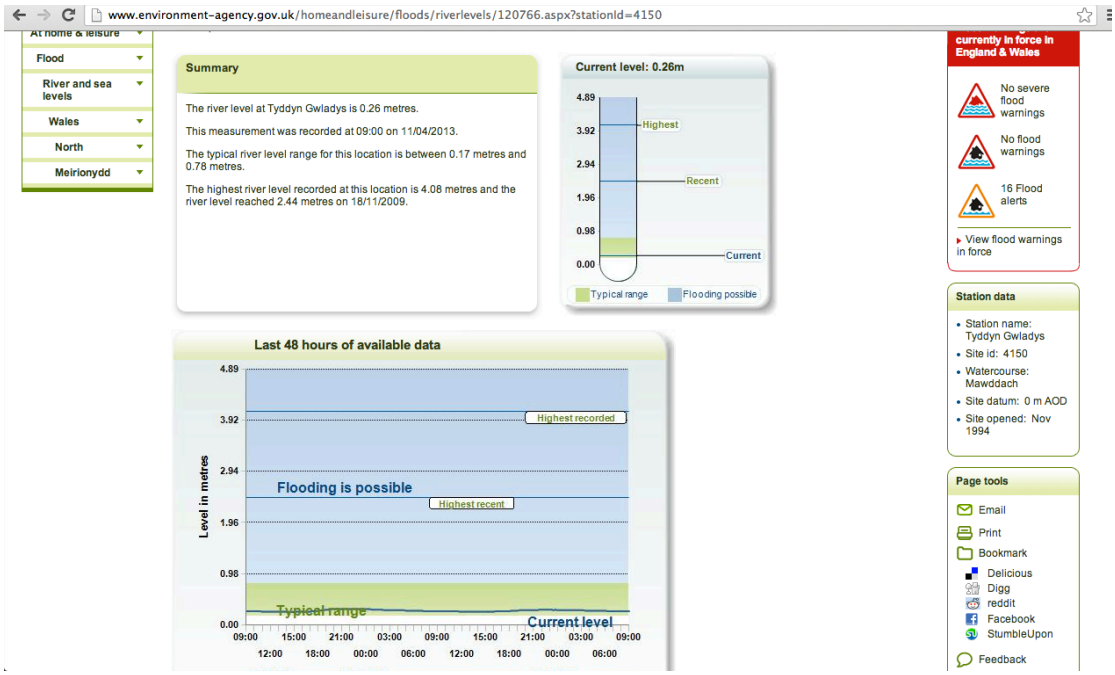
Within the different parts of this section is an explanation of how the features of the application were implemented and any problems or issues that occurred during implementation.

4.1. Implementation of Features

This subsection has nine subtitles which contain a detailed explanation of the implementation of the major features.

4.1.1. River levels

Presenting the user with up to date river levels is the most important functionality of the application. Unfortunately there were some problems obtaining the river levels data directly from the Environment Agency *(see the Problems and Issues section 4.2.2 for a detailed explanation)*. The first part of the application which was built was a simple scraper. It was built using Java and the Jsoup library. If it had been a commercial application it would have taken the data directly from the Environment Agency hence a considerable amount of time did not need to be spent on the development of the scraper.



Above is an example page of the river levels on the Environment Agency's website. The river level is in the Summary box and is in the <div.plain_text> tag which the Jsoup library extracts and puts in an array. Then the element in the array, which contains the river level, at the time of reading, and the name of the river, is written into a JSON file. The JSON file is stored on the Aberystwyth University user's space. This is a link to the JSON file online: http://users.aber.ac.uk/itp9/riverlevels/riverlevelsfile.json

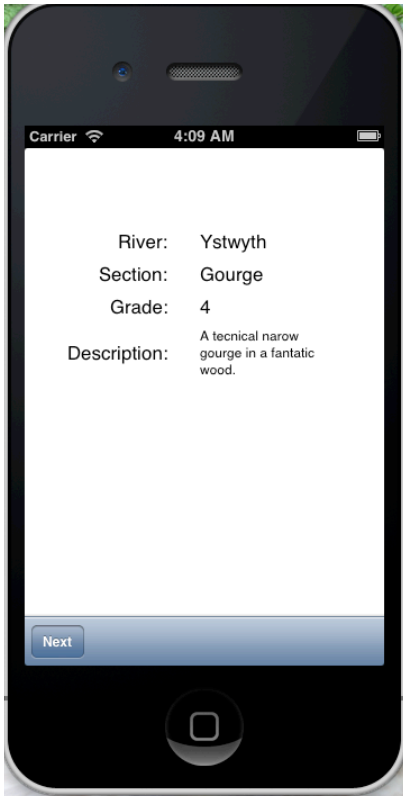
Once the river levels were in a file, which was easily accessible, then the development of the iOS application could begin. As the project was

following Agile development methodology the first objective was to have the application display a river level. Working functionality is an important part of Agile. To do this there was a view with a text label and a view controller connected to the view. The view controller has code which extracts the river level from the JSON file and puts it in the text label. At this point the application could only show one river level.

4.1.2.Displaying information for each river

In the application each river has a page with a description, the name of the river, name of the section, the level and its location. All this information is stored in a SQLite database which is stored internally in the application. To extract the data from the database a group, named Model, with two classes was created: RiverLevels and MyRiverLevels. These are both database access objects (DAO). By executing SQL statements in the code they assign the results in to an array. [13]

Using these classes the database could be accessed from any view controller needed in the application. Once the application could read data from the database it needed to display in a view.



Above is a screen shot of the original view that displays the information about each river. Each page has read the information from the database using the Model class. Using the next button it changes to the next record of a river in the database.

4.1.3.Rivers displayed in a list

An important part of navigation in the application is to have a list of the rivers. Making lists for an iOS application commonly uses a table view.

The difficulty in making a list of rivers was the combination of obtaining the data from the database and putting it into a table view.

The first part of implementing this requirement was to read the data from the database. Then the data, which was to be shown in each of the cells, was put in a two dimensionally array. After this the table was populated with the river name as the title and the section name as the subtitle. To populate the table it needed to be a dynamic table. A dynamic table will change depending on how many elements are inserted into it.

There was some difficulty in linking the river list and the view displaying the river details. To do this each view needs to be linked by a cell. In the code of Table View Controller there is a method prepareForSegue. When the user chooses a cell it runs this method and passes on the ID of the object to the view controller of the description view. The description view uses the ID to find the correct river before displaying it.

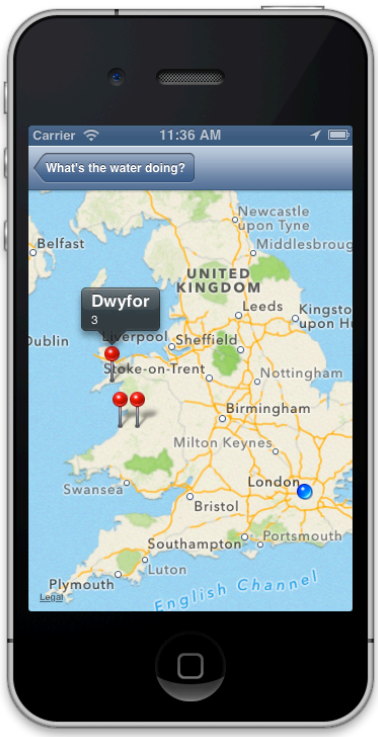
4.1.4.Displaying the rivers on a map

Using storyboard it is quite easy to have a map displayed in a view. To get the map displaying any single location it needs coordinates. Apple provide a reliable framework called MapKit; it provides all the classes needed for using maps in iOS. An annotation takes the coordinates and displays the location using a pin. It is possible to display information when the user taps on the pin in a bubble.

The coordinates were already being read from the database. The map view method uses the co-ordinates to create an annotation on the map. It is also programmed to zoom and centre in on the annotation.

Above is an example of the map in operation and displaying the location of the get on of the lower section of the Ystwyth River.

As well as having the rivers shown on a map in their individual description view, the application also includes another map showing all the rivers. In the applications development is similar whether showing one river or multiple rivers on the map. A user can choose an annotation which then displays data about the river, its name, grade and the section name.



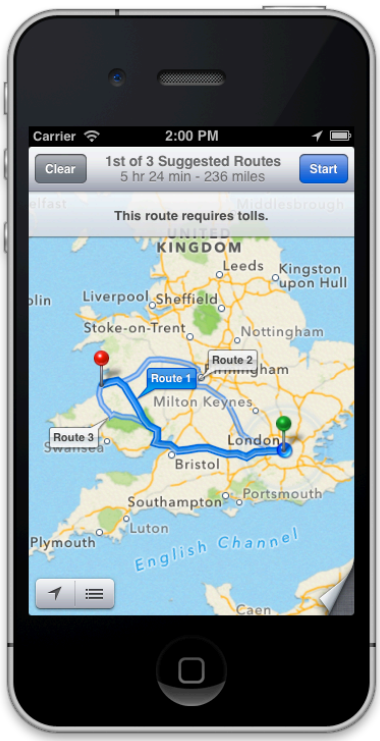
Above is an image of the application showing three rivers which have been read from the database. The map also shows the user's location. To do this, the application uses Apple's Core Location framework. This framework provides the classes that use the device Wifi and GPS to determine the device's location.

To show multiple annotations on the map, it required an extra class. RiverLocation class plots the annotation on the map. It is called by the MapViewController class. When it is called it plots the annotation with the river name and grade in a 'bubble'

4.1.5.Directions

Originally it was decided to have the directions in the application. After researching the area it was apparent that using Apple's Maps application would provide a better and quicker solution. If the directions were to be in the application it would need to be coded from scratch.

IOS 6 has made it easier to open the Maps application.Once MapKit was understood it was relatively easy to get the direction part of the application to work. In the view, as well as the information about each river, there is a button for directions. It runs a method which gets the river's coordinates and launches a MapKit method that opens the 'Maps' application.



Above is an image of the Maps application opening from the application. It has the location of the river as a destination and the current location of the user. As the image shows, Maps is a very comprehensive application because it plots multiple routes with and without tolls and has turn by turn directions.

4.1.6.Comments

Comments were implemented in a different manner than expected. To begin with there was the expectation to create a backend web service that could store the information securely in a database. The use of Parse was then recommended by Prof Chris Price. Using Parse instead of building a backend database/web service allowed more time to be spent on development.

The first step of implementing the comments section was setting up Parse. This was simply done by setting up a free account which included one million API requests and one million push notifications to the device per month. The application in its development stage was not going to need this number of notification pushes or API requests. After setting up an account it was then possible to make an application in Parse. A class was created in the Parse application to store the comments.

The iOS application had to write the comments to the parse application. To do this the application had to receive the comments. A view was created for making a new comment and within the view there needed to be a Text View. A Text View is a box in which a user can type in the comment using the onscreen key pad.

Above is an image of the new comment view. The comment is sent to Parse when the submit button is pushed. To send an object to Parse the application needs to be configured to work with Parse. Doing this was straightforward as Parse has a step by step guide for setting up an application on their website. [14] To use Parse the application has to use Parse's third party framework named 'Parse'. Once the program has been set up to deal with Parse it is quite easy to send the object to the Parse class using their 'PFObject'. Not only did the method have to send the comment string in the object but it had to have a river section ID, so each comment could be associated to the correct river section.

4.1.7.Displaying comments in a table

As the comments are being stored on Parse which had been set up there were no extra modifications needed on Parse to extract the data. All that was needed to display the comments was to have the application query the Parse class.

To put the comments into a table the data had to be queried. The first part was to get the river ID from the previous view; to do this there was a Segue method [15]. Parse provides a query class PFQuery. Using this class's methods then the application can query for the correct objects. The application queries for all the objects in the Comment class on Parse with the riverID. Then it puts them in an array which is used by the tableView method and displayed in a table format.

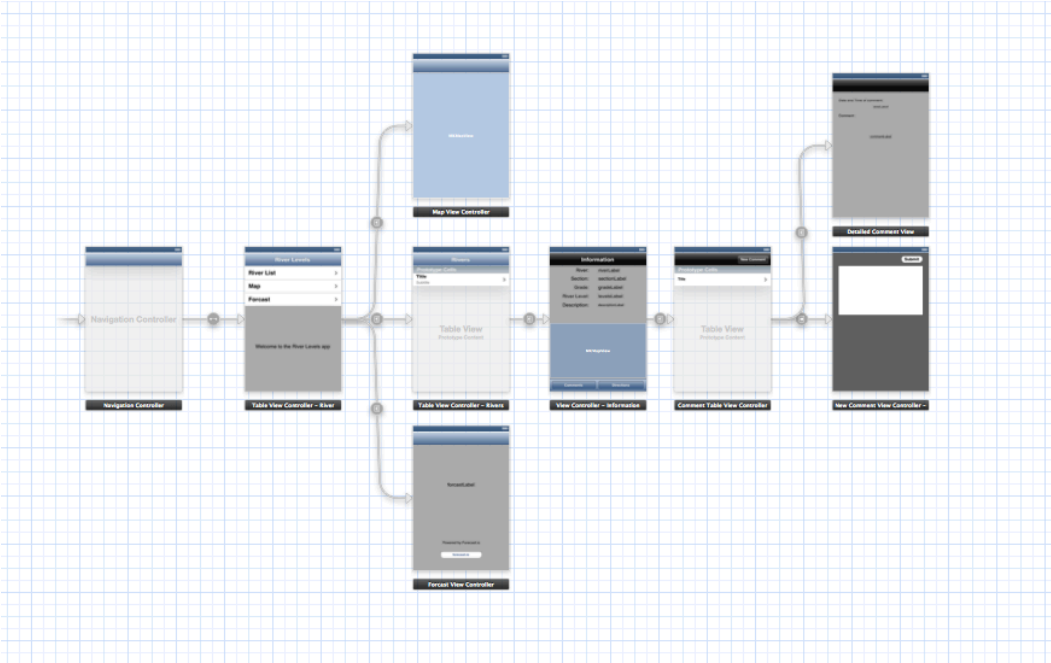
The issue that arose while implementing the comment table was putting the comments in cells.

An additional functionality was implemented in linking the comment to an additional view which displays the whole comment and the date and time that it was submitted. This was done after a recommendation during user test. Implementing this required passing the comment's Parse object to

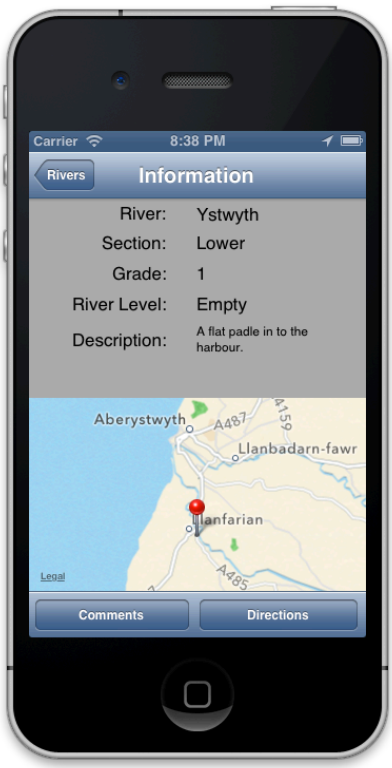
the view that displays the whole comment then querying the object for the comment text and the createdAt field. Both are displayed in a text label.

4.1.8.Layout and flow of the views in the application

The most important part of the implementation was fitting all the views together in a logical layout. Getting an application to behave like the user expects is key.



This is the final layout of the application's views. It shows how the navigation flow of the application has evolved since the design stage. Connecting the views created some bugs and errors. The first new view will be a static table view that is the opening screen of the application. The first cell will be a link to the view with a list of rivers. Each river in the list will be linked to a view with more details in them.



Above is the detailed view of each river and at the bottom there is a toolbar with two buttons. The Comments button is linked to the associated view and Directions is linked to the method that calls the Maps application.

The second cell, in the first view of the application, is linked to the map view. Having a link to a map, which shows an over view of all the rivers, is a useful feature as the user can see all the rivers and their location, giving them the perspective of how far they would have to travel to a river.

The errors that were created while making the flow of the views in the application were often logical errors or errors that came from a lack of experience in iOS. By using the debugger in Xcode all the bugs, errors and logic errors were found and fixed.

4.1.9.Forecasting which rivers are to rise

Unfortunately this was only partially developed. Individual rivers do not have a forecast to whether they will rise or not. Developing the forecasting model was left until the end because it is not a core functionality.

As time was short the forecasting model was developed within the application rather than the original back end development. The forecasting model was developed for a single river, the Dwyfor in North Wales.

A view was created to display the forecast. Within the view is a button which is linked to the forecast.io website as is requested by their Terms of Use. [16] Also there is a text label which displays the forecast if the river level will rise in the next 24 hours.

The forecasting view has a view controller named ForcastViewController. Within this class is the code which reads the JSON file provided by forecast.io and gets the data about the amount of rain around the river in the next 24 hours. The data is compared to the amount of rain which will make the river rise compared to the current river level. Then the results of the comparison are displayed in a text label.

4.2. Problems and Issues

This subsection, and subtitles within, discusses any problems or issues that arose during the implementation stages.

4.2.1. Maps and directions

In iOS there is only one map framework that can be used which is MapKit. This has classes for map view, plotting annotations and directions.

The directions were first intended to be in the application, but MapKit has a class which passes the coordinates to Apple's Maps application which in turn gives the user directions from their current location. Having the direction in the application would cut confusion for the user as they might not have realized that they had left the application when they were in the Maps application. Advantages of using Maps is its powerful features. It has the ability to plot several routes such as the fastest, avoiding tolls and avoiding highways. It also has turn by turn navigation which is a useful feature as the device can be used in a similar way as sat-navigation.

Changing the way the application implemented the directions affected the design, but it did not take more time to develop. If it had been decided to have the directions in-app then it would have taken more time to develop.

4.2.2. Receiving river level data

In an ideal situation the application would not have to have a backend web service that gets the river level data. This is possible as the Environment Agency do provide the data in an XML file, but there is a charge of up to £500 to connect to their server as well as a £1500 annual service charge. On top of this the application would need a license for which the minimum payment is £1875 plus VAT. If the application is sold there would also be royalties to pay. See Appendix C for the correspondence from the Environment Agency.

Considering this is an academic project it would be unreasonable to pay this amount of money for the data. The solution was to build a web service that directly 'scrapes' the data from the Environment Agency's website. The scraper is only run when needed minimizing the amount of traffic on the Environment Agency's website. To get the current river levels it will need to be set to run every two hours.

Having to make a scraper used valuable time and focus from development of the application. The advantage of doing this was being able to decide what type of format the application would use. JSON was preferred as it works better with NSDictionary and NSArray in Objective-C.

If the application was being developed for a business then the scraper would be unnecessary as the data would be bought from the Environment Agency. Releasing the application as it is would also bring legal issues as there is no license in possession to redistribute the data.

4.2.3.Data stored in the Cloud

Unfortunately it was later on in the project that information about Parse technology was found. Fortunately it could be used for the comments section of the application. It would have been ideal for the storage of data for the rivers instead of having them in a database which is stored on the device. Also it could have been used for storing the river levels data.

Refactoring the application so it used Parse for all its data management unfortunately would have taken too long at the end of the development stage. The advantage of having all the data stored in the 'Cloud' is the ease of changing and updating any information.

Once the application is on the App Store any changes that are made to the information about the rivers, for example a description needs to be changed, then the application will have to release an update. With Parse all that would need to be done is change the data in the object.

Using Parse in the application changed the development. Originally the intention was to use a JSON file to store all the comments. The advantage of Parse is its security and its ease of use.

It ended up taking longer than anticipated to make the comment section in Parse. When estimating development of the comment section previous experience using JSON was factored in whereas there was no previous experience using Parse.

4.2.4.IOS development

Over the course of the whole project iOS development was the biggest factor in slowing development. At the start of developing the application there was no previous experience in programming with Objective-C and developing iOS applications.

Thankfully, iOS is a large and growing sector of the mobile market. It is very well documented and Apple have comprehensive documentation of Objective-C and iOS. This documentation was helpful while developing these technologies. With all the major development topics there are tutorials online. At the beginning of the academic year there was a iOS development course which gave the basic skills later used to develop the application.

Throughout working on this project there was an underestimation of the amount of time it would take to develop aspects of the application. Simple code which it was known how to implement in other programming languages created problems and errors. Inexperience led to badly engineered code at times which needed refactoring.

Developing for iOS is different to other development previously done in and out of university. The code is similar to other objective oriented

languages which have been experienced. Java and C++ have the same concepts as Objective-C, as well as a similar syntax.

The application did not complete all the functionality that the project set out to achieve. A lack of experience developing iOS applications was what delayed development.

This was not a fundamental problem to the project. Developing the fundamental parts of the application took longer than anticipated. Getting data from a database and putting it into a table view is something that was expected to be a simple task. Unfortunately it took significantly longer than expected. Once experience was gained working with table views it made working with them the next time a lot quicker. This happened throughout development but by the end of the project it was a lot quicker.

5. Testing

Testing the application happened throughout the development stage. As the project was using Agile as its methodology it should have used test driven development. The project had an informal testing method throughout development and at the end of development there was user testing.

When testing iOS applications it is important to consider what version of iOS is running on the device. Another concern is what type of device the application targeted is. The newer versions of iPhones have higher resolution displays or an iPad has a larger display. Different devices and versions of iOS could change how the application runs and the layout of the user interface. A user interface designed for an iPhone or iPod Touch would not look right on an iPad.

Xcode comes with the iOS simulator which is useful for testing the application. The simulator was good enough for testing the application during development. It can be customized to different devices, for example iPad or iPhone 5. The application was tested on the iOS 6 simulator. One disadvantage of using the simulator is that it does not have the user's current location. Current location of the user can be inputed into the simulator. The disadvantage of using the simulator is that it does not take into account the hardware on an iPhone or similar device. This was taken into consideration throughout testing. Also memory leaks can occur with the simulator but not be picked up until the application is tested on a device.

The following sections discuss the different types of testing considered and an explanation of the testing that happened through the project.

5.1. Unit Testing and Integration Testing

Apple provide a platform for Unit Testing. OCUnit testing is the framework that is provided for testing in Xcode. OCUnit is made by Apple and is easy to set up in an Xcode project. Considering the amount of time that was available for testing it was decided that it was not worth the advantages it would bring.

When developing iOS there is not so much need for unit testing as the application is quite simple in many ways it runs on a device with a small screen and a small backend system. It uses one language and there are many frameworks that are provided, such as MapKit, that are tested and just needed to be plugged into the application. These frameworks have been tested and created just for iOS, all that needs to be done is to plug them in. Also it is not possible to see if the application works as intended without running it on a device. [17] [18]

Agile encourages test driven development. For the iOS application there was no need to spend time designing unit tests. This allowed more time to work on integration testing. No application or program can be created without any tests so integration tests were the most suited for this project. When each feature was developed it would be tested to make sure that it worked as specified. As each feature was developed it would

be tested to each feature that was developed previously. This makes sure that no new code breaks anything that was already developed. Testing in this way is integrated testing and regression testing. Testing the development in this way might not have followed Agile principles.

5.2. Stress Testing

There was no need to stress test the application as it can only be used by one user on a device. The only possible parts of the application that need to be stress tested would be the Parse classes.

5.3. User Testing

Having feedback from real users was be beneficial to the project. After developing each feature a user would be asked to test the application on an iOS device. To do this this a developer license was needed to install the application on a mobile device.

User testing was important for the project. The application is designed to be used by human users. It is important that it was tested by a range of people to see if it is user friendly and works how the user expects. User testing was implemented through test tables. A tester would have the application installed onto their iOS device. As the application was still in production it would not be possible to release the application on to Apple's App Store. Once the application was on a user's device then the tester used the test tables.

As well as testing the application with different users it was important to test the application on different iOS devices and different generations of iOS operating system when possible. If the application was released it would be available on the App Store. Any generation of iOS devices have access to the App Store so it was important for the application to be compatible with as many of them as possible.

Unfortunately the timescale of the project did not allow for prolonged user testing. The application was mainly tested on an iPod touch and an iPad. The application was targeted for an iPhone running iOS 6. The application would not work on a device running iOS 5 as it does not support Apple Maps.

User testing had two phases. The first was carried out after development had finished then the results were used to fix any failed tests and recommendations would be used for further development.

The second phase was after any fixes that were raised from the first phase of testing for further development. This was more of a test to make sure that any of the fixes did not break something else accidentally.

5.4. Testing on different devices

The application was designed to run on an iPhone that runs iOS 6. As the Apple App Store is available for all generations of iOS devices the application was tested on an iPhone 5, iPod Touch and an iPad. Each device was tested using the same tests.

Unfortunately there was no access to a device that did not run iOS 6 or below. As the application was targeted for iPhone the results of running it on the iPad's larger screen were unsatisfactory. The layout of each screen looked wrong because the application is targeted for a smaller iPhone screen. When its run as an iPad application the user interfaces is stretched out of proportion as the image below shows.

On an iPhone the application looked and worked as expected. The layout of the application worked fine, and above is an image of the application targeted for iPad and being run on the Xcode simulator

As is apparent in this image the application does not work the same way for an iPad. The problem is with the layout on an iPad views with tables do not look as good on an iPhone. The application was built for an iPhone it was to be expected that it would not be as user friendly on an iPad.

5.5. Results

Integration testing happened during development and the results were dealt with soon as an issue was found. As integration testing happened during development there was no strict process of logging any failed test. A failed test was noted and then the next task was to fix the problem that caused that test to fail. No further development would happen until the problem was fixed. This may not be the standard way of testing but it suited this project due to its small size, one developer and its timescale.

When a large functionality was created users were asked to test the application. They used the test tables for this. Also users were asked for any recommendations that could be used in the application.

In sections 5.6.1 and 5.6.2 there is an explanation of the two phases of user testing and the results.

5.5.1. Results for phase 1 of User Testing

Once the project was completed the users were asked to test it using the test tables. In the first phase of user testing there were only two failed

tests. The reason for this was because of thorough testing during development.

Test ID	Test	Expected Results	Pass/Fail
8	A new comment is viewed in the comments view.	When a user creates a new comment it gets automatically loaded into the table view when the comment is submitted.	Fail
13	Annotations have a 'bubble' that shows information about a river.	A users chooses an annotation by tapping it and a 'bubble' appears with the grade and river name.	Fail

Above are the two tests from the test table which failed. Having users to test the application brought the advantage that users could give the following recommendations:

- That the new comment view needed to close once a new comment had been submitted.
- Each comment has an individual screen with the full comment.
- Every comment displays the date it was created.
- That the first view in the application has a title.

Each one of these recommendations was taken on board. Time was spent to change the application and to implement the recommendations.

5.5.2.Results for phase 2 of User Testing

The second phase of user testing happened as soon as the development fixing the failed tests of user testing 1 was completed. In this phase of testing none of the tests failed. Users had more recommendations for the application which were:

- From the Map view a user could see more details about each river in a separate view.
- In the Map view there is an option for directions to the river from the user's current location.
- A user can change the type of map that is being shown in the map view and the detailed page view; the map could be a hybrid or a satellite image as well as a road map.

Unfortunately there was not enough time to implement any of the recommendations for phase two of user testing. Future versions of the application will possibly use the recommendations as extra functionality.

6. Evaluation

The evaluation has been cut into subsections discussing requirements, design, tools used and whether the application meets the needs of a user. Future work and what would be done differently are also part of the evaluation.

6.1. Requirements

The main objective of the project was to create an iOS application that assists kayakers in the selection of suitable rivers. Achieving this objective required breaking down the process for selecting a suitable river into requirements that aid kayakers. All the factors a kayaker would need to consider and do before getting to a river were listed. First the kayaker would choose a river which has a suitable grade and river level. Then check the forecast to see if the river level will rise or fall depending on the amount of rain predicted. It can be dangerous if there are floods whilst paddling a river. After choosing a river then the kayaker will need directions there.

It was important to choose an effective method of identifying the requirements. Setting the requirements required separating each part of the process that goes into choosing a river. Each part of the process involved choosing a river had to be set as a requirement. This is an effective way of identifying the requirements. There were some exceptional situations that do not normally occur when choosing a river. Having a comment is a suitable extra requirement as it gives the kayaker more information about the river before having to make a choice. All the requirements were correctly identified and there were no unnecessary features.

6.2. Design

Design was not a major part of the project due to Agile being the choice of methodology. One of the design difficulties was lack of experience in working with iOS and developing a mobile application. The success of the design and any decisions that were associated were mixed.

The general flow of the application was the first part of the design. It provided a guidance for what needed to be made and then for building the storyboard for the application. Having an outline of the application's structure helped build the application in a logical and tidy manner.

An important feature that was implemented was having the database with information about the river, internal to the application. Otherwise it would have been difficult to update or add rivers to the database. If the application was released then it would be difficult to update or add rivers to the application database. The database, with information about the rivers, shoulFd be stored in the 'cloud' and cashed on the device when it has signal. When developing the design, this weakness should have been noticed, changed and re-factored in the application.

One of the main issues with the application and its design is the information about the river levels. In the first outline of the application it

was assumed that it would obtain the river levels data directly from the Environment Agency. Once it was apparent that this was not possible then the design evolved to take this into account. This led to the application needing to read the data from a JSON file, a fault in design that unfortunately was implemented into the application. It should have taken into consideration the use of Parse as the backend 'cloud' storage and stored the data there which would have simplified the architecture of the application.

Whilst designing some integral mistakes were made which weakened the application. As it was developed using Agile the mistakes in design had not caused the application to crash due to following the principle 'Working software over comprehensive documentation'. A mixture of inexperience in developing a mobile application and not refactoring the design led to the mistakes.

6.3. Tools used

When making an iOS application it is only possible to use technology provided by Apple for the majority of development. The only IDE that is used to develop iOS is Xcode. Tools used for the iOS part of the project were suitable due to being designed specifically to develop iOS and the lack of other choice.

For the 'scraper' Java was used and a third party framework Jsoup. Both these choices were the right choice and proved to be successful. The main reason for this was previous experience using Java. The other possible and perhaps more suitable tool to use would have been Python. However, without any experience using it development could have been at a slow pace.

Storing data in the 'Cloud' using Parse was effective. The database used in the iOS application, SQLite, was not the ideal tool to use. As mentioned elsewhere to change any data in the SQLite database an update of the application would be needed. Parse was the tool that should have been used in this situation. As it provides flexibility to the application.

GitHub was used for version control. The use of version control proved to be valuable. GitHub was easy to set up on the Mac and it was easy to use. Through development changes of code were not submitted regularly but were only submitted when a major functionality was done. As a development tool it was perfect but it should have been used more frequently.

Developing an iOS application does not require the developer to think about what tools to use because, for the majority of the time, all the tools are provided by Apple. The non iOS parts of the application needed some thought about what to use. All the tools for this project, except for SQLite, were well chosen.

6.4. Meeting needs of the user

It is difficult to know how well the application has met the needs of the user. The only way to know this would be by a full release of the application to the public to discover whether the users like it.

In my opinion the requirements for the project were what a user would want. The application has not managed to implement all the requirements although core requirements for the project have been completed and so have some of the extra functions. A user can make an informed decision as to which river they wanted to paddle and it will assist them in getting there.

Sadly some parts of the application were lacking in development. The comment section could have been expanded. The main reason to use the comments is to warn other users of potential dangers on a river. Originally the application was going to be able to let the user take photos of the danger and include it in a comment with a description and date.

Another requirement that could have been implemented into the application was alerting the users to which rivers are rising within a 50 mile radius of them. There was not enough time to develop this aspect. It would have been an extra but it was not a core requirement. Any keen kayaker will be checking the river levels and forecasts regularly as a matter of course.

Parts of the interface of the application could have been changed. The rivers list could have had a filter so the user would be able to choose a certain grade or rivers in their proximity. Doing this would have made an application which users could personalize to their own needs. Unfortunately doing this was too big a task to complete in the time scale available.

This application has met my needs as a kayaker. It gives me information on which rivers have enough water in them and information such as river grade and a description of the river. It also helps by giving directions to the river.

6.5. What I would do differently

The project, in my opinion, was a success; the application will prove to be a useful tool for kayakers but there are many changes that I would make if the opportunity came to start it again. No part of the development life cycle was perfect.

Starting again, the biggest changes would be regarding the testing. Even though the project used Agile methodology, in which testing is very important, there was a lack of testing. Test driven development could have been beneficial and would have speeded up development because more focus would have been in what is needed from the code. Another problem was when I would re-factor code it would break code not directly related to what was being re-factored. This wasted a lot of time.

The design section required some changes but the biggest problem was spending too much time designing rather than developing. In the

beginning I spent time mapping out the flow of the application. Whilst developing I had problems and changed the flow but did not re-factor the design. This led to confusion and meant that the design was quickly out of date. What I would do if I had the chance to start again would be to have a minimalistic design.

From the start I would have used test driven development which would have simplified the development and the code. My lack of experience was a key factor in the slow development of the iOS application. With hindsight I should have spent more time practicing making applications using iOS. After attending a brief iOS course I started developing the application, instead I should have made small applications that use the different technologies involved in the project.

If I had the chance to work on the project again I would use iterations to manage the development time. Each iteration would require set features to be developed and then tested before moving on to the next iteration. Feature driven development was used as a development method, but the use of iterations would have given a better overview of what needed to be done and by when. Also using iterations could give more of a chance for testing after each iteration and better time management.

6.6. Future work

The application is quite comprehensive in the requirements of a kayaker. I do think that there is a need for a second more complex application. As the application is now it is not possible for the user to personalize it. Also it would be sensible to have the application running on Android and possibly as a mobile web application.

Extra functionality that would be in a second version:

- Make it compatible on Android and as a mobile web application.
- The user could put a filter on the list of rivers so it shows rivers close to their location or rivers of a serrate grade.
- A photo could be linked to a comment, a useful tool to show users of a new danger on a river.
- An alarm is triggered if a river reaches a certain level, the user has control of which rivers and level they want to be notified. To do this the application would need to use Apples Push Notification service.

A problem that could arise in the second version is over complication of the application. Applications that are a chore to use do not get used. So the additional functionality needs to be integrated into the application’s smooth style.

Another idea which would engage the user would be integrating the application with Facebook. The comment section could be visible on Facebook. The comments show up on the user’s Facebook news feed and this will engage the users and make the application more interactive.

6.7. Summery

The project was set with what I thought were realistic aims. The finished product does meet the needs of a user. It is a success as it could be used by a kayaker to determine what river is at the correct water level as well as other information about the river. Unfortunately the application did not reach all the projects aims but. In the end the application did not have the detailed forecasting for what river levels would rise.

In hindsight the whole approach to creating the application was naive. Inexperience in producing such an application and the technology involved proved to slow development considerably. Some parts of the project, such as making tables, proved to be far more time consuming to develop when the data had to be retrieved from a database or Parse. Also I underestimated the amount of time that would be spent on developing the map and direction functionality.

A principle that I intended to keep throughout working on this project was good time management. At the beginning of the project I would try to work throughout the day, 9 until 5, like a working day. Unfortunately when developing I would not keep to this principle, often working late. This led to periods of the project that were not well time managed. Looking back, these periods of development were not as efficient as they should have been if I had adhered to the time set out for development each day.

By working on this project I have gained valuable experience as a developer. Not only have I learnt how to program with Objective-C I have learnt how to develop iOS applications and the technologies involved.

Appendices

A. Apples Frameworks

During the development of the applications several of Apple's frameworks were used that are used for iOS development.

UIKit Framework: http://developer.apple.com/library/ios/#documentation/uikit/reference/UIKit_Framework/_index.html

This framework is used for creating and managing the UI within the application. The application uses several of the classes that are in this framework.

MapKit Framework: https://developer.apple.com/library/ios/#documentation/MapKit/Reference/MapKit_Framework_Reference/_index.html

MapKit is used in the views which have maps. Its classes are used to plot data and directions.

CoreLocation: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CoreLocation_Framework/_index.html

The application requires the user's location. This framework makes use of the technology within the device to determine the user's location.

B.Third-Party Frameworks and Libraries

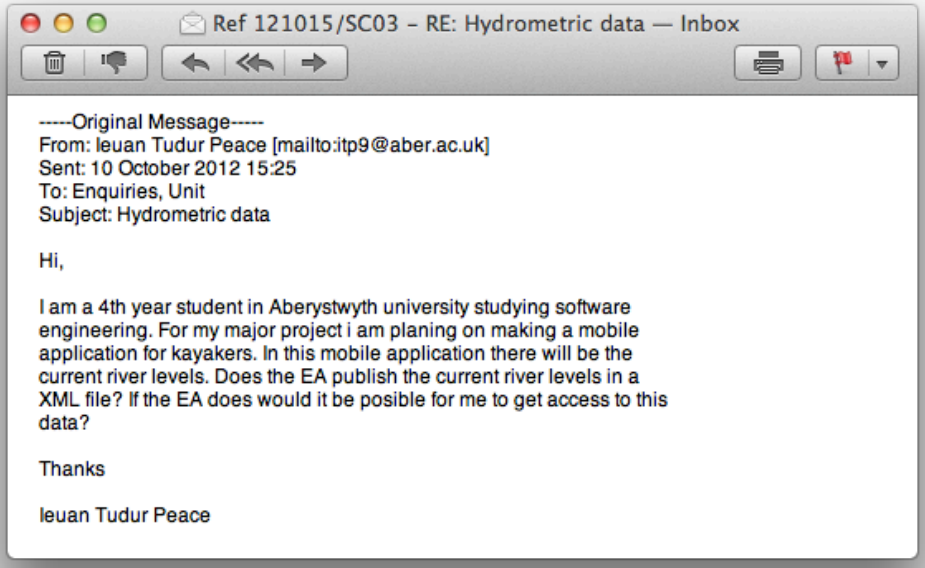
Parse Framework: https://www.parse.com/apps/quickstart

This framework was used to access the data which is stored in the Parse cloud. To use this framework the application needed to use some of Apple's frameworks such as; Social.framework, Account.framework and Security.framework.

Jsoup Library: http://jsoup.org/

This was used in the 'Scraper' to read the data from the Environment Agency's website. It is a Java library which is licensed under the MIT license.

C.Correspondence with the Environment Agency



Ref 121015/SC03 - RE: Hydrometric data — Inbox

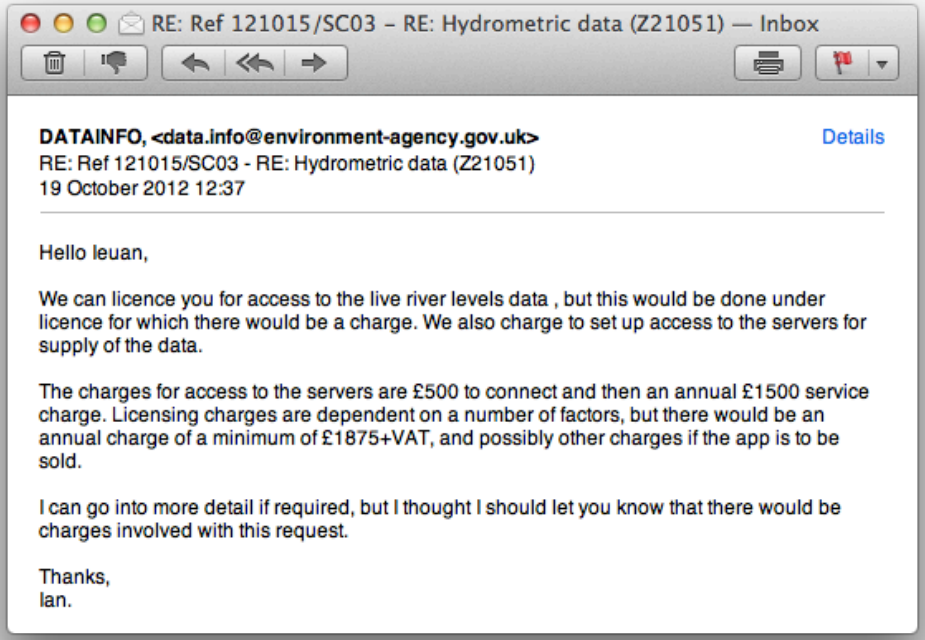
-----Original Message-----
From: Ieuan Tudur Peace [mailto:itp9@aber.ac.uk]
Sent: 10 October 2012 15:25
To: Enquiries, Unit
Subject: Hydrometric data

Hi,

I am a 4th year student in Aberystwyth university studying software engineering. For my major project i am planing on making a mobile application for kayakers. In this mobile application there will be the current river levels. Does the EA publish the current river levels in a XML file? If the EA does would it be posible for me to get access to this data?

Thanks

Ieuan Tudur Peace



RE: Ref 121015/SC03 - RE: Hydrometric data (Z21051) — Inbox

DATAINFO, <data.info@environment-agency.gov.uk> Details
RE: Ref 121015/SC03 - RE: Hydrometric data (Z21051)
19 October 2012 12:37

Hello Ieuan,

We can licence you for access to the live river levels data , but this would be done under licence for which there would be a charge. We also charge to set up access to the servers for supply of the data.

The charges for access to the servers are £500 to connect and then an annual £1500 service charge. Licensing charges are dependent on a number of factors, but there would be an annual charge of a minimum of £1875+VAT, and possibly other charges if the app is to be sold.

I can go into more detail if required, but I thought I should let you know that there would be charges involved with this request.

Thanks,
Ian.

D.Test Tables

Test ID	Test	Expected Results	Pass/Fail
1	Open the application on an iOS device.	The application successfully launches in the device showing the first view.	c
2	The first view is linked to the river list view.	The application changes from the first view to the river list view.	
3	In the river list view is a table containing rivers	A table containing rivers is in the view, each has a subtitle.	
4	Each river in the table is linked to a view with more details about the river	When a river is chosen from a list it changes screen to a more detailed view.	
5	The detailed view displays data from the database	The view shows the river name, river section name, description, level and grade of the river. As well as a map with an annotation showing the get on of the river	
6	Comment button links the description view to the comment view	The comment button links the application to the comment view	
7	A list of comments are shown in a table view all are retrieved from the Parse cloud	A table containing rivers is in the view all have been retrieved from Parse	
8	Displays the detailed view with the comments.	Each comment has an individual view with the date and time the comment was submitted and the comment	
9	It is possible to submit a comment to Parse	A comment is successfully submitted to Parse.	
10	A new comment is viewed in the comments view.	When a user creates a new comment it gets automatically loaded into the table view when the comment is submitted.	
11	The map view displays the user's location	It displays the user's location on the map in the map view	

12	An annotation shows where the rivers are situated on the map	In the map view on the map is an annotation showing the locations of the rivers	
13	Annotations have a ‘bubble’ that shows information about a river.	A user chooses an annotation by tapping it and a ‘bubble’ appears with the grade and river name.	
14	The button for directions to the river takes the user to the maps application.	It takes the user to the map application.	
15	The maps application shows the user the directions to the river from their current location.	It shows the directions to the river.	
16	The forecasting view shows the river forecast for a single river.	It shows the forecasted river level for the river Dwyfor.	

Annotated Bibliography

[1] Agile Alliance, (2001), *Agile Manifesto*. [Online] Access from: http://agilemanifesto.org/iso/en/manifesto.html [Accessed: 2 April 2013]

This was used as a source of information about the Agile principles.

[2] Apple Inc, (2010), *Why Objective C?* [Online] Access from: https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/OOP_ObjC/Articles/ooWhy.html [Accessed: 5 April 2013]

Documentation about Objective-C, why it is used and its history.

[3] CAMPELL, M (2013), Apple's iOS 6 now accounts for 83% of all iOS-based traffic in North America. [Online] Access from: http://appleinsider.com/articles/13/02/13/apples-ios-6-now-accounts-for-83-of-all-ios-based-traffic-in-north-america [Accessed: 27 March 2013]

A blog explaining why the majority of iOS users run iOS 6 and the data they have used to reach this conclusion.

[4] Apple Inc, (2013), *Core Data Tutorials for iOS*. [Online] Access from: http://developer.apple.com/library/ios/#documentation/DataManagement/Conceptual/iPhoneCoreData01/Introduction/Introduction.htm [Accessed: 5 April 2013]

Documentation and tutorials on how to use Core Data for iOS.

[5] Scrapy, (2013), *An overview Scrapy*. [Online] Access from: http://scrapy.org/ [Accessed: 10 April 2013]

A summary of what Scrapy is and the advantages of using it.

[6] Raywenderlich, (2010), *How to chose the best xml parser for your iPhone project.* [Online] Access from: http://www.raywenderlich.com/553/how-to-chose-the-best-xml-parser-for-your-iphone-project [Accessed: 15 January 2013]

A blog post discussing the advantages and disadvantages of several different XML parsers for iOS.

[7] JSON, (2013), *JSON: The Fat Free Alternative to XML.* [Online] Access from: http://www.json.org/xml.html/ [Accessed: 15 January 2013]

Discusses why JSON is better than XML for transferring data from a file.

[8] Maniac Dev, (2009), *Iphone Sqlite vs. Core Data – Which to choose?.* [Online] Access from: http://www.json.org/xml.html/ http://maniacdev.com/2009/09/iphone-sqlite-vs-core-data-%E2%80%93-which-to-choose/ [Accessed: 15 January 2013]

Explains why and when to use SQLite or CoreData.

[9] The Dark Sky Company, (2013), *Forecast for Developers.* [Online] Access from: https://developer.forecast.io/ [Accessed: 15 April 2013]

Provides information for developers wanting to develop using the service provided by forecast.io.

[10] Apple Inc, (2012), *iOS Human Interface Guidelines.* [Online] Access from: https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html#//apple_ref/doc/uid/TP40006556-CH1-SW1 [Accessed: 8 April 2013]

Documentation that provides the guidelines creating a user interface for iOS.

[11] Apple Inc, (2013), *iPad specification.* [Online] Access from: http://www.apple.com/uk/ipad/specs/ [Accessed: 10 April 2013]

A page with the iPad specifications.

[12] Apple Inc, (2013), *iPhone specification.* [Online] Access from: http://www.apple.com/uk/iphone/specs.html [Accessed: 10 April 2013]

A page with the iPhone specifications.

[13] LANGUEDOC, K (2013), *Tutorial on creating an iOS SQLite application.* [Online] Access from: http://klanguedoc.hubpages.com/hub/Tutorial-on-Creating-an-IOS-5-SQLite-Database-Application-IOS-5-SQLite [Accessed: 12 April 2013]

A tutorial on how to create an application in iOS that gets data from an SQLite tutorial.

[14] Parse, (2013), *iOS/OS X guide and documentation for Parse.* [Online] Access from: https://www.parse.com/docs/ios_guide [Accessed: 12 April 2013]

The complete documentation and guides on how to use Parse in an iOS or OS X application.

[15] Apple Inc, (2013), *UIStoryboardSegue class documentation.* [Online] Access from: http://developer.apple.com/library/ios/#documentation/uikit/reference/UIStoryboardSegue_Class/Reference/Reference.html [Accessed: 10 April 2013]

Documentation for using the Segue class UIStoryboardSegue.

[16] The Dark Sky Company, (2013), Forecast.io: *Terms of Use.* [Online] Access from: https://developer.forecast.io/terms_of_use.txt [Accessed: 20 April 2013]

The Terms of Use for using the data supplied by forcast.io.

[17] MERGULHAO, S (2012), *Its not about the tests.* [Online] Access from: http://agilewarrior.wordpress.com/2012/10/06/its-not-about-the-unit-tests/ [Accessed: 17 April 2013]

A blog post explaining why there is less of a need for unit testing when developing an iOS application.

[18] GALLAGHER, M (2010), *Quality control in application development without unit testing* [Online] Access from: http://www.cocoawithlove.com/2010/01/high-quality-in-software-development.html [Accessed: 17 April 2013]

Explaining how an iOS application can be created without using unit testing.