Final Report

ANDROID 'TIDES' APPLICATION

Carlton Sandhu

Supervisor: Ralph Martin
Moderator: Martin Caminada

Module CM3203 One Semester Project (40 Credits)

Abstract

In completing of this project I have produced a fully location-aware android application that can not only display tidal results but has an implementable way of estimating tides as a vast solution to existing applications that lack functionality. There are several ways, designed to fulfil User needs that tidal ports are able to be selected and their respective data retrieved. This already offers more functionality than pre-existing applications.

However the real investigation was into providing an acceptable estimation of Tidal Data at an uncharted location using existing data. Challenges of which included which data is selected, what criteria influences the tides and how best to approximate the data. This assignment looked to provide a usable, feasible solution to the problem and look to improve on existing methods, where available.

Acknowledgements

I would like to sincerely thank Professor Ralph Martin for his continuing support, patience and insight he has provided in my project and for providing fruitful criticism and invaluable suggestions through the duration. Without Professor Martin, I wouldn't have learned half as much from this venture and I hope the project reflects his efforts in doing so.

Contents

1 - Introduction

1.1 – Aims and Goals

The overall goal of this project are to create a comprehensible, readable and efficient Android Application to channel a solution which will allow users to swiftly find tidal predictions on the android platform. There are several ways that can determine which station the tidal data is sourced from but the main aim of this project is to derive it automatically through the user's location. Upon port selection the user shall have all relevant data displayed including and up until a certain date range providing a good degree of accuracy and correctness in retrieving these results to display in an understandable manner.

There are very few tidal applications on the Playstore, all few of these offer very basic tidal information and I believe there is room for improvement in the manner these applications present and manipulate tidal data. The client application will be written for the Android platform solely.

Furthermore with this application a big Goal is to be able to predict a tide in between stations for a more accurate result offered to users which, as my preliminary research suggests, would be a completely unique feature offering very useful extra functionality.

1.2 – Project importance

The prediction service this assignment is going to offer, I believe fulfils a need for more coverage of tidal data on the UK coastline. Despite offering 695 official locations along the coast, there are actually many significantly large gaps between stations. This is not ideal for people who live near these uncovered sites. Perhaps one of the uncovered sites is a fishing haven where information about the tide would be paramount to locals and visitors. My preliminary research into the matter suggests that there are several tidal applications, none of which provide this added (or much other) functionality which performs such a service which I aim to provide. Thus I feel there is an actual need to bridge the gap.

1.3 – Audience and potential beneficiaries

The audience for my project assumes that they can read tidal data in tabular and/or graphical format, yet assumes no technical-aptitude or even moderately complex geographical domain knowledge. My project is only available on the android platform and will require a working network connection. I will also only be developing the application in English for obvious reasons (see the scope below for more information).

In terms of actual audience, the target market is very wide encircling anybody of which tidal data can be useful to. This includes a multitude of casual and professional enthusiasts ranging from fisherman and casual sailors to professional fields of studies such as marine biologists. We can ensure that the data source we are using (Admiralty Total Tide) is the United Kingdom Hydrographic Office standard. Admiralty TT is 'The world's most comprehensive tidal prediction' and 'fast, accurate tidal height predictions'. Partnered with the fact that it is fully automated, which reduces chance of user error, the accuracy and integrity of the data can be guaranteed throughout.

The target audience has no age restriction as such but will generally the kind of activities tidal data is used for assume a more mature audience from about 18 upwards. The general tone and layout of the program is going to be formal as the main purpose of the application is informative as opposed entertainment since the general purpose is meant to be informative.

1.3 – Scope of the project and outcomes

The scope of this project limits the functionality of the features to locations within the United Kingdom and ports that are available for selection will be geographically located within the UK. The primary objective of the project is to provide extensive tidal information and provide current data. The project needs to be able to do the following by the end:

- Allow user free selection of any port within the UK via a map
- Allow user selection of a port by way of using list of results (categorised by some criteria to enable easier navigation)
- Use the user's location to be able to determine the closest (potentially a list of the closest) station to their current/last known location
- Display tidal data from the selected port in tabular format and in graphical format

To solve the problem that somebody quite far from a tidal station couldn't get local enough tidal data if there is a lack of a port, this application will have some advanced goals.

- Be able to select the nearest body of water to the users location to enable a location where a prediction can be made
- Be able to give a reasonable estimation of Relative Tide Times / Tide height between two stations given a point in the water

Given these goals I am going to make them measurable in that all retrieval of tidal data for existing ports is to be obtained and processed in real time (dependent upon internet connection). Furthermore interpolation of tides between two stations must be calculated and rendered within a feasible amount of time, ensuring not to inconvenience the user.

Realistically regarding project scope and the time I have to develop it, a more intense focus will be on the functionality as opposed to implementing and designing a massively user friendly application. Due to the audience and beneficiaries there is less need for a fancy GUI and more need for a formal, functional UI so this is not really an issue.

1.4 Approach

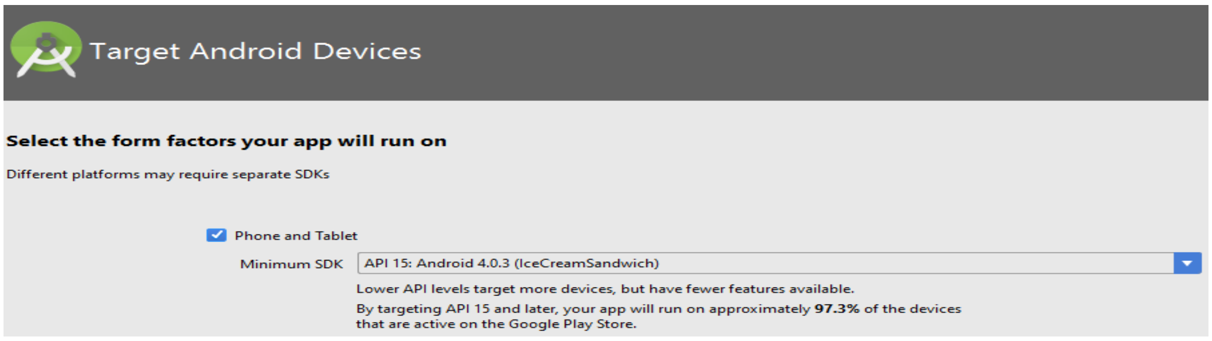
For my application's development I've chose to adhere to the Waterfall model. This software development cycle best suits my needs as I have pre-identified goals, I can be invested completely in one stage at any given time ensuring quality work and the project is relatively short. Furthermore I have familiarity with this procedural development. I have chosen Waterfall over other methods such as Agile as the project will not be repeatedly changing and it's important with an assignment like this there is adequate planning. I also considered the Iterative method but disregarded it due to the fact that the project isn't that big, requirements are not always present up front which may cause design issues later on down the line and it's a rather costly model in the fact that rough products are created at every iteration.

1.5 Assumptions

In developing this project there are a few assumptions I am obeying:

- As aforementioned the end-product will be a fully functional prototype so there will be more focus on meeting the functional requirements rather than having a visually appealing application.

- The project's locational scope is limited to the UK as research as we can use reliable data from the official UK standard and there is limited port information elsewhere in the world available to me.
- Assume that there is an active internet connection (Via Data or via Wi-Fi) on the user's android smartphone otherwise application will not be able to function correctly
- Assume user's Android SDK is API 15: Android 4.0.3 (IceCreamSandwich) or above. This is based on Android's recommendation to not program to anything before this as over 97% of Android users are running an API from or later than this. See picture below.


Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☑ Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available.
By targeting API 15 and later, your app will run on approximately **97.3%** of the devices that are active on the Google Play Store.

Figure 1 – Recommended device settings from Android

2. Background

2.1 Problem Identification and Context

As it stands UK tidal data is currently available on several well-known places on the web (BBC, UKtidetimes, and Admiralty etc.) yet not one of these large providers have an application available on the android market. Furthemore (See Below) if you lived in Happisburgh your two nearest stations, Cromer and Winterton-On-Sea, are both roughly 17 miles away. So someone who would need tidal data relevant to Happisburgh has to pick between these stations and use their tidal data to estimate for their location. This could vary hugely between those stations therefore this application hopes to provide an 'Interpolation' technique to create data from existing data.

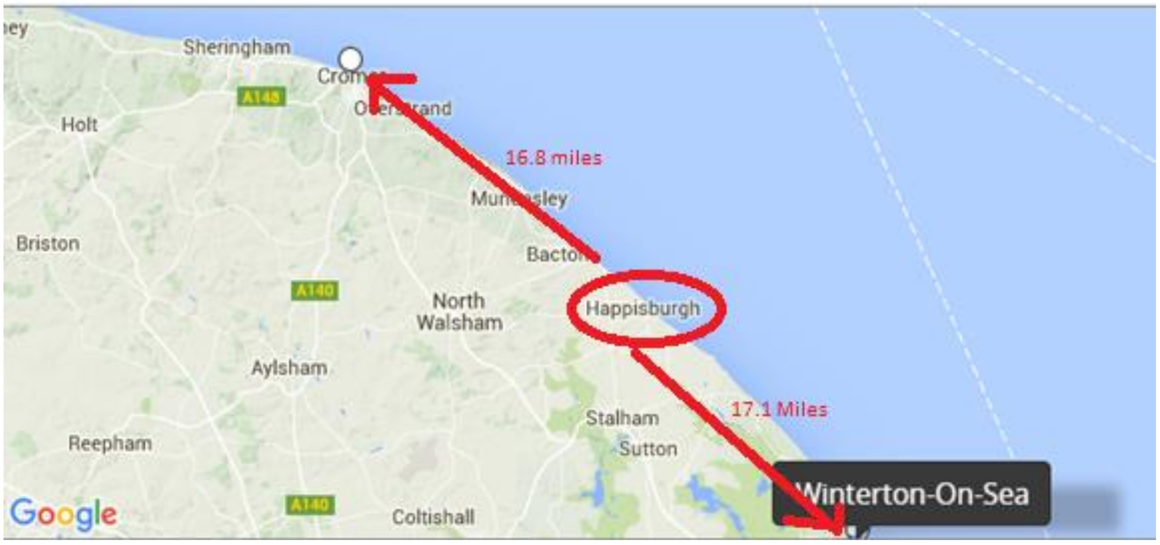


Figure 2 - Happisburgh on google maps with distance between the closest two stations

This project's aim is to provide a more complete tidal coverage of the UK to provide people with a suitable estimation of tide given their location.

2.2 Coastal Theory and Interpolation

The only complete way to observe the change in water levels is to actually have a huge backlog of data from a constant point. From knowing the data, knowing the orbit of the Earth, Moon and Sun (which are exactly known) means tides can be predicted in the future at the given point with great accuracy. To make predictions of the tides height and times other factors such as lunar phases and the periodic rotation of the Earth are factored in and as shown in the diagram below produced trustworthy results. R4

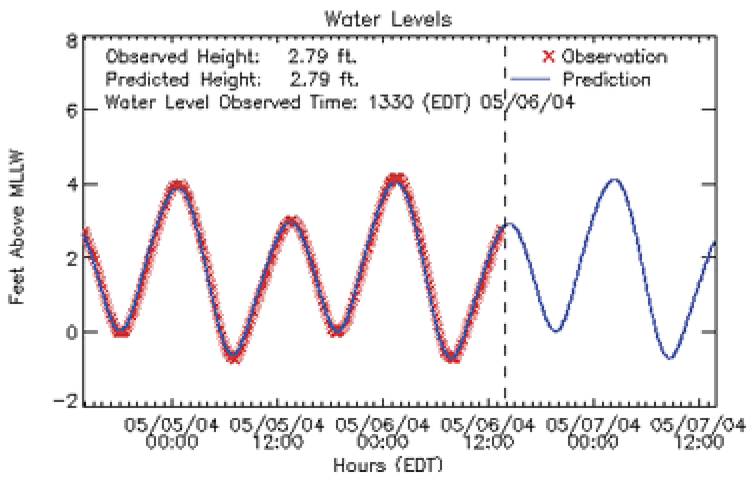


Figure 3 - Predicted water level vs Observed Water Level

However accurate prediction is, trying to interpret and interpolate between known stations can be problematic. If the known stations were in the middle of the ocean it would not be a problem but because I am looking to predict tides on the UK, the tide can be heavily dependent upon the shape of the coastline. Therefore I am seeking to provide a reasonable estimation with a certain degree of accuracy.

2.3 Constraints to the approach

There are over 600 stations in the UK I have access to be able to retrieve tidal data. As aforementioned there's no scientifically verified, pre-existing, deterministic algorithm that I can utilise in conjunction within my project to predict tides between these stations. This means I'm going to have to determine a suitable geographical solution that can be automated and performed within an Android application and smart-phone's capability..

The application also requires the user to have an internet connection to be able to retrieve results. The application also needs full permission and location services to be active when accessing location, to be able to provide some of the intended features for the application.

Finally time could become a factor with needing to complete all parts of the report/project to an acceptable standard in my opinion. Partnered with the fact I'm learning Android Programming, explore several solutions to a problem (which will require some geographical research as well). Due to not having access to any backlog of data for every single point on the UK coast there's unfortunately no way I can feasibly gather data within the time frame. Not to mention the fact that I have lack of access to tools or resources to be able to gather the data anyway.

2.4 Exploring existing solutions

There are several third-party applications that provide tidal information. Upon first inspection of the two most popular of these applications it was immediately clear that they were lacking features which I think could greatly enhance the application. The two applications I examined were 'anyTide' and 'UK Tides'. Both of these applications take your last known location / your current location and display the port(s) nearest to you (In the case of anyTide nearby Ports are shown on a map). They were both lacking the selection of a port of anything other than your current location however, so for example if you were going to travel to Newport but were currently situated in Cardiff, you wouldn't be able to obtain the information on the UK Tides application since you weren't actually at Newport. I want my application to offer several ways of selecting the port they want as I found not only both of their solutions to be inadequate in terms of choices, but far-from intuitive as to what each button performs which task on either of these apps.

AnyTide has a limited map selection of under 200 ports which can only be selected by Map. Besides the search button which is a pretty good feature to search for a port (though mostly doesn't return a result) they are confusing and have no hints as to what each of them do.

UK Tides doesn't even provide an alternative means for searching besides using your current location. However it does provide a good feature to enable you to save your favourite ports for later use. Furthermore if you have recent data it saves it rather than having to re-scrape the data from the web which saves battery usage and data.

As suggested by my preliminary research, none of these have a solution to intermediate estimation between tide stations. Besides the displayed Graphs, they both offer a very simplistic view. The saving grace of these is that when a port's data is actually obtained the initial date the data is very easy to understand.

2.5 Methods and Tools for Solution

Whilst doing preliminary research into potential methods to aid in the project, there were a couple of incredibly useful Java modules and interpolation methods. The first was a Google API which is incredibly powerful tool in locational and geo-locational utility services. Secondly a lightweight web-scraping module which is perfect for the limited processing power of a smart phone. Lastly I've looked into producing some techniques involving triangulation. Partner that with investigation into the tidal interpolation method that exists to see if a feasible, computational resolution can be made.

The first priority I had was to find and learn Android Programming, as well as find a good IDE to be able to collate everything in a package as well as have good IDE functionality. Fortunately, Android not only provides an official IDE with more extensive functionality and support than any other Android IDE., but also they provide an online course from which it only took a few days to become familiar with almost everything I needed. The Android Studio IDE for android development is a Java IDE like Eclipse or Netbeans, both of which I am familiar with and are good for Java development and have some Android support, but provides a vast amount more maintenance for resources, automatic generation of necessary XML files and in-built libraries to help with designing the UI. Furthermore it provides an emulator on which you can test your application as well as providing support for managing all imports/external APIs with broad library management.

Google Maps API allows developers to integrate google maps into their project which is essential for this project. With the need for location functions such as plotting points on a map, retrieving location and a user being able to select a point where tidal data can be estimated, this API provides functionality that encompasses methods to be able to achieve this. Google offer this as a free for non-commercial use license with limited requests (1000 per day). This shouldn't be a problem as my application won't be available publicly. Google Maps API was chosen in place of the in-built Android location modules. Google's location API is recommended by Android themselves therefore I chose this over the deprecated Android Location module.

There are many android-compatible web-scrapers, using a Web-View widget on Android could even perform the necessary data acquisition. I chose Jsoup because it doesn't need to waste time and battery power to display all the elements as it just parses the HTML as a document. Furthermore its compatibility with Android Development was a big factor for inclusion. HTMLUnit was considered also for its added javascript-parse functionality but ultimately I didn't need to process javascript so was unnecessary.

Lastly I explored tidal data and prediction and found that there is a way tidal data can be estimated. However upon further studying found out that the only way the complex interpolation algorithm can be done is if you have a pre-existing backlog of data of at least 3 months at a certain point. Considering the time frame of the project, lack of measuring instruments and resources this is not an option I could carry on pursuing. However, using computed versions of trigonometry/triangulation to estimate distances between ports and the angle of incident, I can achieve a better estimation of tides. There are several Java libraries like the Path library or Math to help with triangular formulae.

3. Specification

3.1 Application/Program Specification

The outcome of this project is to have a fully functional android application that provides at least all of the features, in improved form, that existing applications provide. The application is going to produce tidal information such as height, time and whether the tide is high/low in real-time along with a comprehensive list of UK tidal stations. The application also aims to provide several ways of being able to select which station to retrieve data from. Lastly, the application hopes to be able to provide a good way of estimating tidal data between known stations through algorithmic interpolation of pre-existing data given a user-defined location.

Given below are a list of essential and desirable requirements for the application side of the project.

Essential Requirements

1. Application should offer choice to display a map with UK tide ports charted from which user can select the tide they wish by clicking on the marker
2. Application should offer choice to display a list of the UK tide ports by name
3. List of tides should be categorised geographically by region (I.e. Southwest, Northeast, Scotland etc.)
4. Application should offer choice to display a some of the closest tides to the user's last known location
5. Application should display the Tide Height, Tide Time and Tide Type in a tabular format after a port is selected, plus the name of the port that was selected
6. Application should plot said Data in requirement 5 in graphical format
7. Application should provide a way to estimate tidal data by the user selecting the 2 nearest stations
8. The application should have a complete list of UK Tide Times concurrent with tidetimes.org.uk

Justification:

Requirements 1, 2 and 4 are quintessential to the project as it addresses the problem that pre-existing solutions have in that they don't offer many ways to select a port. These 3 requirements cover pretty much every way there is to select a port and will do so intuitively. Requirement 3 I have selected the regional categories as opposed to selection by County which would produce too many results and categorisation by country which is too general. Regional classification is well-known and widely used within the UK so I've elected to choose this grouping. Requirement 5 denotes all the information that is possible to retrieve for the user regarding tides and is the core functionality. The web pages that display tidal data and all the applications I researched provide a graphical plot so I will too as it would be what a user might come to expect. Requirement 7 is essentially what makes the project unique. Given a map and an estimation of the user's location (which the user can alter) it should find the nearest two stations and offer a prediction of the tides between the locations. Lastly requirement 8 is just a case of keeping an up-to-date list available to the application.

Desirable Requirements (Time-permitting requirements that would enhance the application)

1. Application could store results along with the date the result was retrieved in the devices memory

2. Application could retrieve results from 6 days in advance as well as the current day
3. Application could provide a favourites feature to store their favourite ports

Justification:

For requirement 1 this reduces the usage of the internet and save battery life by storing the results and the date in which it was obtained. If a user would try to access the data on the same given date it could be swiftly retrieved without the need of an internet connection thus increasing functionality and reducing the project's dependency on the internet. Requirement 2 extends the generic convention of applications and websites tending to show a week's worth of data in advance. Lastly, for faster retrieval and increased user-satisfaction, a 'favourites' option where a user could maintain a list of ports they'd like to swiftly and frequently access.

3.2 User Interface Specification

As mentioned earlier there is going to be less concentration on building a user-friendly UI since it is a formal, informative application. However the UI within this application still has to have a good base-level to provide an intuitive and attractive interface. Therefore I've decided to include the following requirements to help satisfy some design principles as recommended by android.

UI Requirements

1. The Main Menu (Home Page) will be accessible from every section of the application
2. Keep all phrases and textboxes as brief as possible without sacrificing the meaning of the text
3. Conform to design trends set by pre-existing applications
4. Keep layout/colour scheme formal
5. Have error messages displayed in 'toasts'
6. Have System-status visible to the user at all time by having title atop of the page depending on section of the system

Justification:

Requirement 1 ensures the user can comfortably 'resort' to the main menu, where all main modules will be accessed from, to ensure easy navigation. Requirement 2 comes from a recommendation from Android that states for brevity; 'people are likely to skip sentences if they're long'. The application's prior intent is to provide information and the target audience aren't seeking a colourful, animated and cluttered interface. Despite taking extra care to ensure as much error prevention as possible (more desirable by heuristic and usability standards), unpreventable error cases need to be displayed attractively to show the user the nature of the error. R7 Android 'toast' widgets are a good tool to do this. Lastly, another heuristic principle of having the status of the system known to the user will be adhered to by displaying the name of the section the user will be in atop of the page.

* Note: I will generally try to adhere to all the 'good' Nielsen heuristic principles as referenced to try and produce a good-quality application. R12

4. Design

4.1 Android Modules

4.1.1 HCI

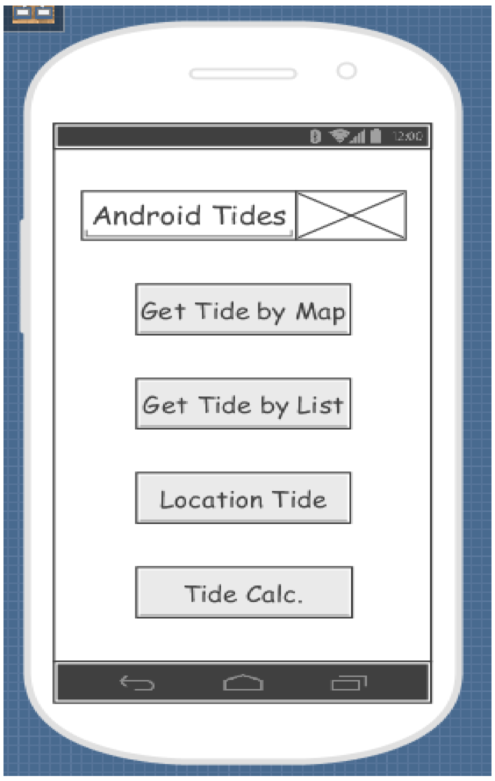


Figure 4 – HCI – Main Menu

Main Menu (left)

From the main screen every major section of the selection process within the application is available via the four buttons. The name of the application along with a tidal symbol image is displayed on the top in a text view. The button 'Get Tide by Map' will access the Map Selection screen, the button 'Get Tide by List' will access the List Selection screen, the 'Location Tide' button will access the Location Selection screen and finally the 'Tide Calc.' will open the Specific Tide Location screen.

As you can see from the design, the menu is very minimal in terms of design. The menu is essentially a bridge to more complex parts of the project and by having large buttons displayed in a list it follows the norm, it's what users come to expect from a menu. It's very clear which state the system is in and this page is easily recognisable which is essential as the user is likely to navigate back to this screen frequently.

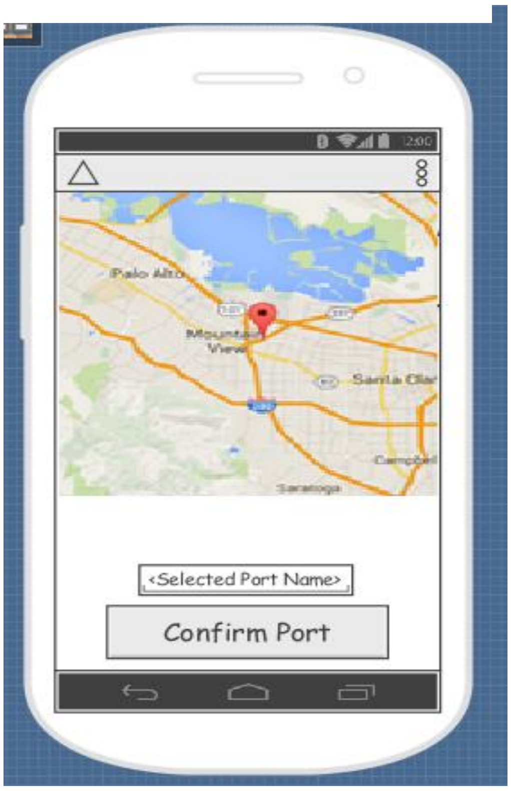


Figure 5 – HCI – Map Selection

Map Selection (left)

As Figure 5 below shows a map, provided from the google Maps API, which will take up most of the screen. The user selects a port by clicking on a marker within the map, which then displays the name of the tide. The button to confirm the port at the bottom will not work unless a port is selected. If the user tries to click 'Confirm Port' without selecting a port via one of the pre-plotted port markers on the map, a brief message will be displayed informing the user that they need to select a port to continue. This adheres to one of Nielsen's principles of error prevention as well as visibility of system status.

Once a port is chosen you are able to select Confirm Port and it will then navigate you to the Tide Data screen.

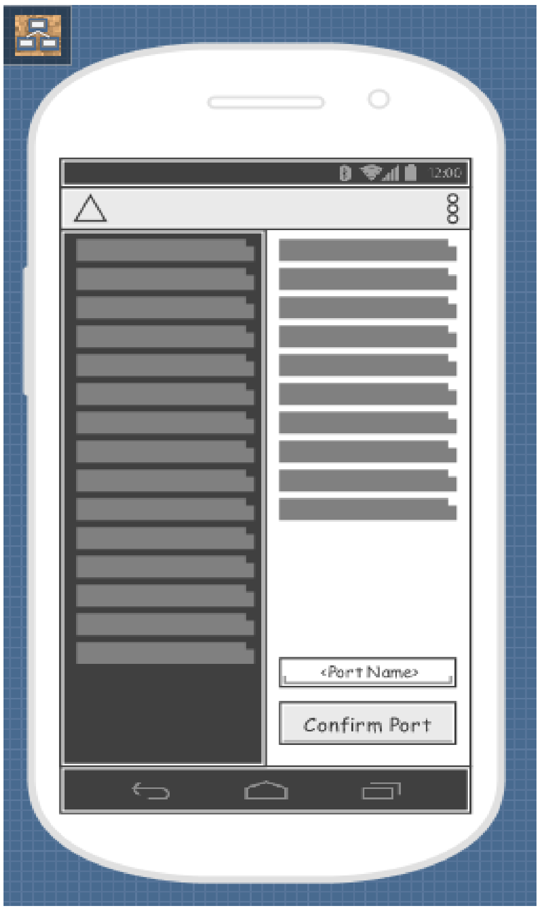


Figure 6 – HCI – List Selection

List Selection (left)

The dark grey list view on the left-hand side of the screen will display the regions of the UK. After one of these is selected, the list view on the right-hand side of the screen will display all ports that are contained within that region. After selecting a port from the right-hand side of the list it will be displayed in the text box below it. If the user tries to press 'Confirm Port' without selecting a port from the right-hand side, a message will be displayed informing the user that they need to select a port to continue. This adheres to one of Nielsen's principles of error prevention as well as visibility of system status.

The lists are categorised into regions as a recognisable match between the real world and the system and consistency. They are organised alphabetically also, all of this reduces time taken to locate a tide and improves the overall efficiency of use

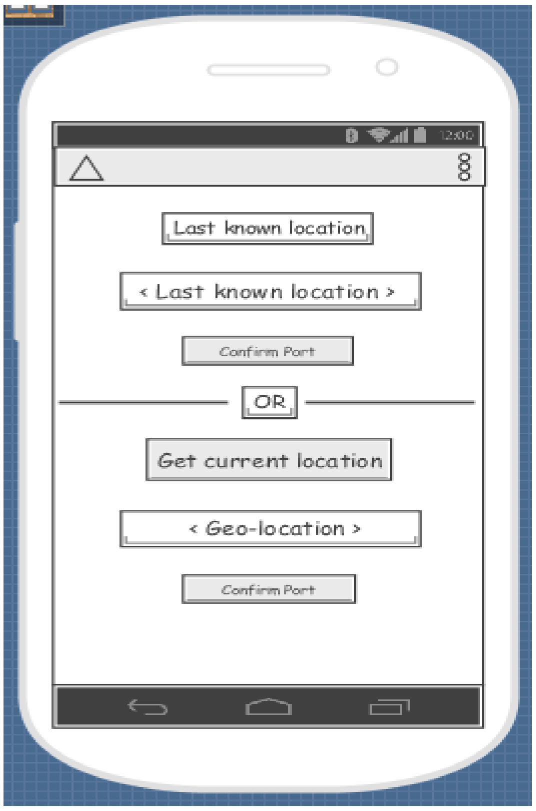
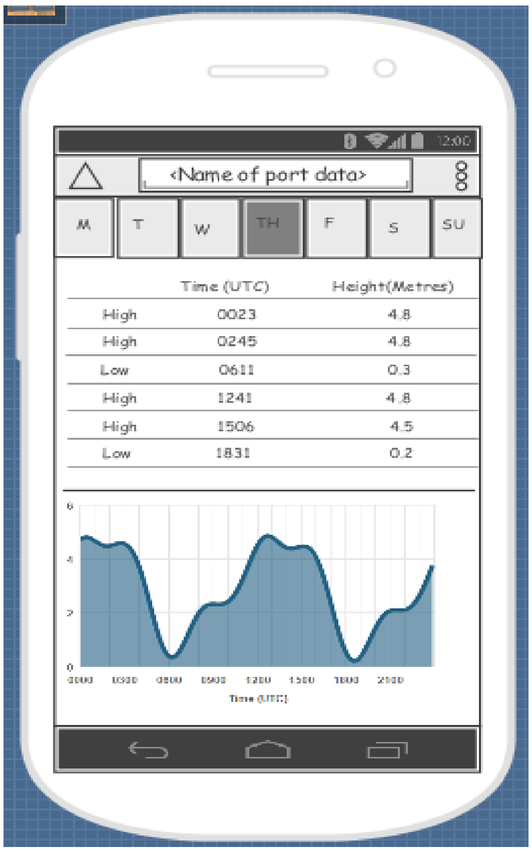


Figure 7 – HCI – Location Selection

Location Selection (left)

On entering this 'activity' the application will retrieve the last known location and display it in the <Last known location> box shown below. Then a list of the closest ports to the user will be displayed. If the last-known location is incorrect, by pressing the 'Get current location' button the application will attempt to listen for location updates. The newly updated list of closest ports will be displayed following this. Once a port is selected it's displayed in a text box. The button to confirm the port at the bottom will not work unless a port is selected.



	Time (UTC)	Height(Metres)
High	0023	4.8
High	0245	4.8
Low	0611	0.3
High	1241	4.8
High	1506	4.5
Low	1831	0.2

Figure 8 – HCI – Tide Data

Tide Data

This activity is where the tidal data is displayed. The port that the user selected is displayed on top of the application. There are 7 days below this, the day is highlighted when the application is displaying that given day's data. Below that is the tidal type, tidal time and tidal height displayed in a table. At the bottom of the page is a labelled and detailed graph based on the ports data, displayed to the user.

Users are able to switch between days by swiping left/right. I've chosen this method to switch between the days as it's the most intuitive for a smart-phone user and the day being highlighted gives good system-state visibility for the user.

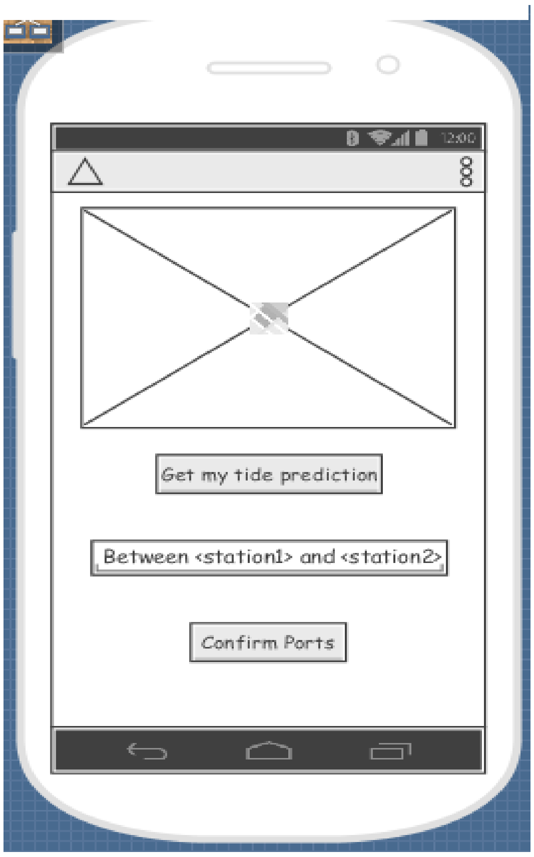


Figure 9 – HCI – Specific Tide Location

Specific Tide Location

This activity uses google maps and a custom marker, which can be dragged to required location, and then selects and highlights the two nearest ports. The activity will display the names of these two ports as well as emphasising them on the map. The visual response is vital to the user to see what is going on in this activity and whether the selection is correct, hence the live updates from the map which adheres to visibility of system status in the heuristic principles.

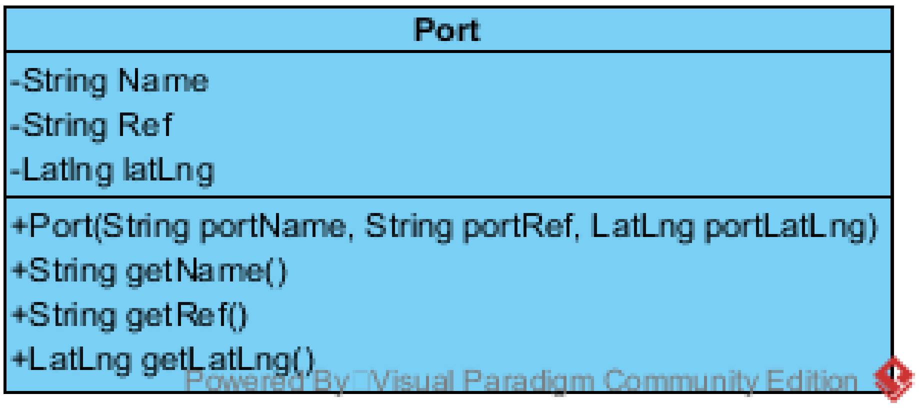
After this, the button to confirm selection will populate results on the Tide Data page and displayed in the same fashion as any other selection keeping consistency and standards.

4.1.2 Data types

Below are a couple of Data Types I've made due to frequent access within my application. The datatypes provide a means of combining Port and Tidal data which is quintessential to the application and are implemented like maps where corresponding data are stored in the same index of different arrays.

Port Record

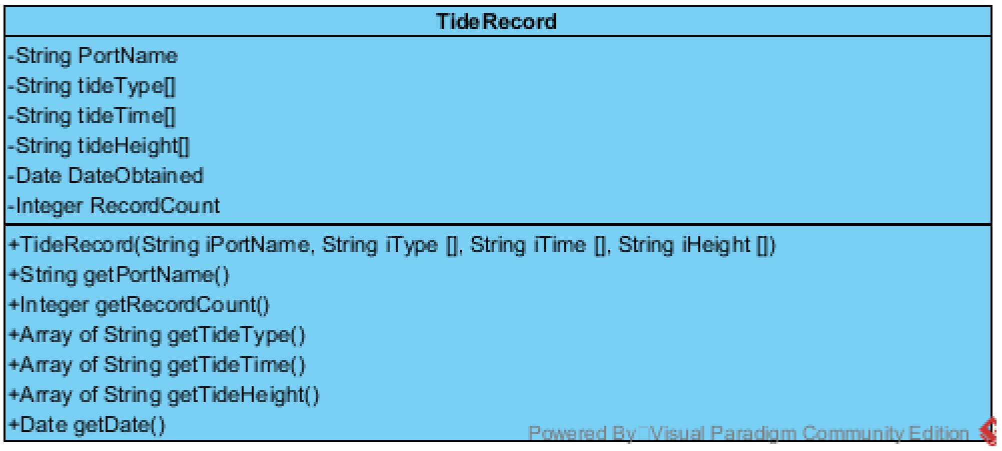
This is a java class to combine necessary data the application will need regarding a port. The name of the port and the hyperlink to the page where data can be obtained about the port are stored as a string. The Latitude and Longitude I could have stored in two long variables, but google provides a LatLng class for added functionality and less conversion is needed when plotting map points this way so I've elected that. Each variable can be accessed with its appropriate getter method.


Port
-String Name
-String Ref
-Latlng latLng
+Port(String portName, String portRef, LatLng portLatLng)
+String getName()
+String getRef()
+LatLng getLatLng()
Powered By Visual Paradigm Community Edition

- String Name: Name of the port
- String Ref: Hyperlink to the port's data page
- LatLng latLng: Holds the latitude and longitude where the port is located
- Port(): Constructer which sets the Port's Name, Ref and LatLng
- getName(): returns Name variable
- getRef(): returns Ref variable
- getLatLng(): returns LatLng variable

Tide Record

The 'TideRecord' class stores the Tide types (High/Low), the tide heights in metres and the tide times, all as string arrays. The class stores one day's worth of records, therefore the three String arrays will all be of the same length. The length of these is stored as an integer in 'RecordCount' as it will be frequently used and is less costly (in terms of time and memory) than constantly checking the size of one of the arrays. Lastly, when the TideRecord is instantiated the date is retrieved and stored. This is for potential use of storing records on the phone so that current data won't have to be re-obtained from the internet by checking if you already have today's records. Again, once a day of TideRecord is set, it cannot be changed but each variable can be accessed with its appropriate getter method. I considered using an 'enum' type for the Tide Type since the values are constant (either High or 'Low) but keeping it as a string allows for more functionality and easier manipulation of data.



TideRecord
-String PortName -String tideType[] -String tideTime[] -String tideHeight[] -Date DateObtained -Integer RecordCount
+TideRecord(String iPortName, String iType [], String iTime [], String iHeight []) +String getPortName() +Integer getRecordCount() +Array of String getTideType() +Array of String getTideTime() +Array of String getTideHeight() +Date getDate()

Powered By Visual Paradigm Community Edition

- String PortName: Name of the port where record was retrieved from
- String tideType[]: Array of the day's tide type (either 'High' or 'Low')
- String tideTime[]: Array of the day's tide time (24 hour format)
- String tideHeight[]: Array of the day's tide height (in metres)
- Date DateObtained: Hold's the date in which record was retrieved
- Integer RecordCount: Hold's the number of tuples in the record
- TideRecord(): Constructor which sets the three string arrays and the name of the Port

4.1.3 Back-Stack model

The diagram below depicts how each activity interacts when an item is added to the back stack. When the user presses the *Back* button on Android the foreground activity is destroyed and the activity previously that called it continues.

As shown below Main Menu is the original parent, from here only a selection task can be selected. All of these selection screens can only lead to displaying tidal data. The selection activities include all aforementioned screens of selection including Map selection, selection from list of Ports and selection based on the users given location. When navigating back (as shown in the last step of this back-stack model) the activity is destroyed and it would revert to the previous activity.

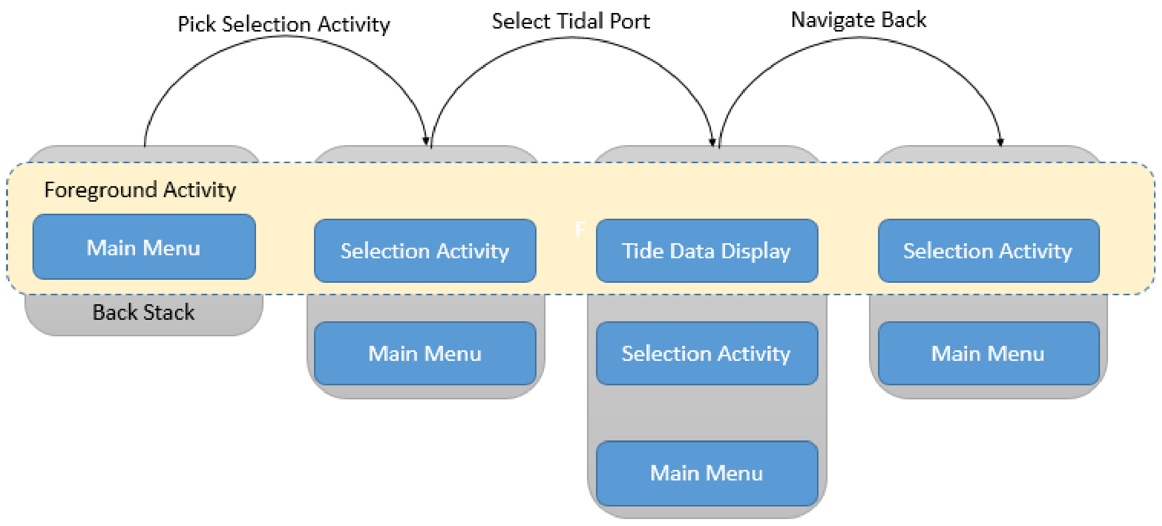


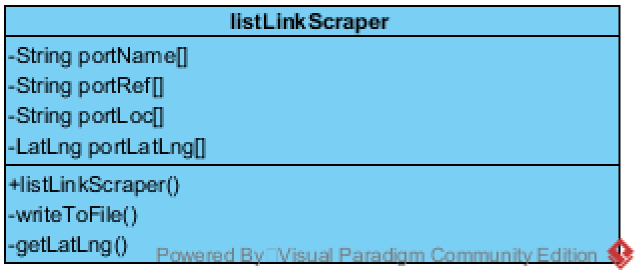
Figure 10 – Android Back-Stack model

4.2 External Modules

4.2.1 Module Design (Static Architecture)

listLinkScraper

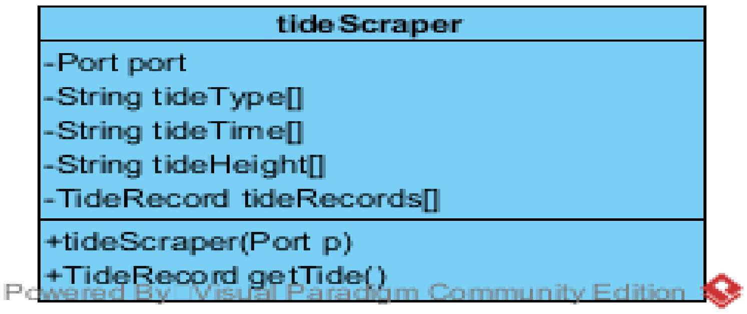
The listLinkScraper module essentially just obtains a list of every single port from tidetimes.org.uk along with the hyperlink and Latitude and Longitude describing location of the port. Then the module writes it to a .txt file which can be read into an array of 'Port' datatypes as described above in 4.1.2. All results are retrieved as String, the portLoc string array which stores location strings are all then converted into LatLng by the getLatLng function, which are then stored in portLatLng.



- String PortName[]: Array of all the Port Names
- String portRef[]: Array of all the hyperlinks of the Ports
- String portLoc[]: Array of the Ports latitude / longitude as a String
- String portLatLng[]: Array of the Ports location after it has been converted to LatLng
- listLinkScraper(): Constructor which populates all the private variables with the results from https://www.tidetimes.org.uk/all which is a complete list of the tidal prediction locations
- writeToFile(): Method to output all of the corresponding portName, portRef and portLatLng to a text file
- getLatLng(): Converts all results from portLoc array into LatLng variables

tideScraper

This module produces one weeks worth of tidal data stored in an instantiation of the class 'TideRecord' . This is obtained by calling getTide after the tideScraper has been instantiated. To construct this class it takes a 'Port' data type which provides the hyperlink needed to web-scrape the results. The TideRecord array will always be size 7 as I'm only providing data a week in advance (Which is the normal for tidal prediction applications). The tideScraper will predominately be used in the Android Module 'Tide Data' to retrieve the results via this module then display them.



- Port port: Holds the port from which the dater will be gathered
- String tideType[]: Array of the day's tide type (either 'High' or 'Low')

- String tideTime[]: Array of the day's tide time (24 hour format)
- String tideHeight[]: Array of the day's tide height (in metres)
- TideRecord tideRecords[]: Array of TideRecord which is always size 7 for a week's results

---

- tideScraper(): Constructor which sets the port variables
- TideRecord getTide(): Retrieves the data for the next 7 days of the selected port

4.2.2 Tide Calculation Algorithm

Since there is no pre-existing algorithm to determine tidal data between two locations, there are a few ways I considered designing one. Firstly my original idea was to take an average of the data at either point e.g. if location 1 had 10 metre tide height and location 2 had a 5 metre tide height then the output of the user-defined location would be 7.5. However I quickly dismissed this as its far too inaccurate and overall a cheap, poor solution.

Secondly it made more sense to split the difference based on distance from one port to another. Essentially if you consider the 3 ports as a triangle, known ports and the location of a port without data in between them. As the diagram below shows, if you take the angle at the unknown port and dissect it in half, then produce a line that goes through this until it intersects the line between the two known ports. If you consider that line almost as if a number line where the points range from one number to another, the intersection point produces a far more accurate estimation of the average tide/height between the two points. The diagram below illustrates this.

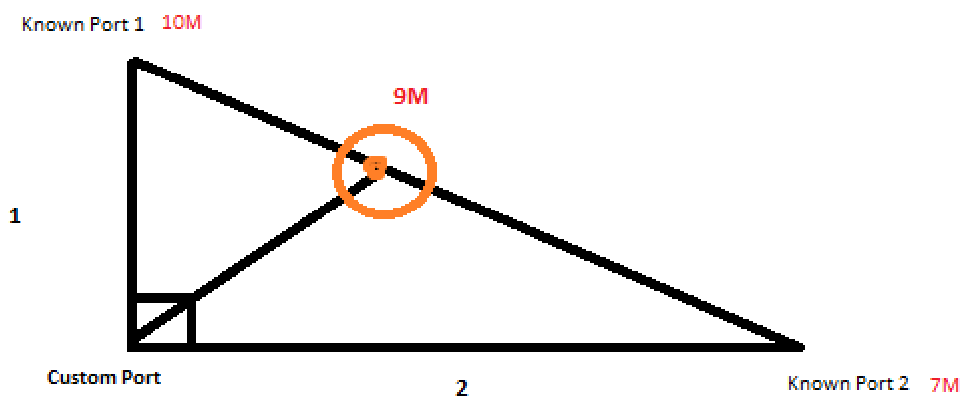


Figure 11 – Relative Distance Triangle Theory

Therefore as my second option I considered plotting the Latitude/Longitude of the ports as X,Y coordinates on a graph and using a visual computing technique to dissect the angle, draw the diagram and ultimately get the ratio between the two values of the known ports. However this technique would require visual computing knowledge, would probably be too computationally intensive for an android application and there are no Android-compatible Java libraries to perform this.

Fortunately I endeavoured to discover that there was a purely mathematical solution which essentially finds the point of intersection and what the value of this is. I have the following pseudo-code which is a rough template and a visual diagram of a triangle to aid in labelling and understanding of code.

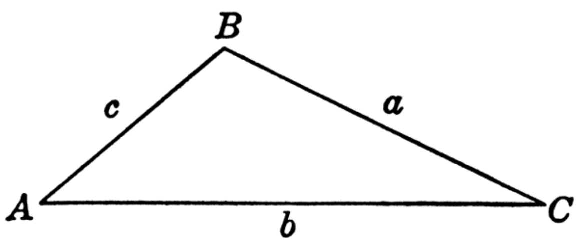


Figure 12 – Algorithm template triangle

For the following pseudocode, assume upperport is a known port located at point A, lowerport is located is a known port at point C and the customport is at point B.

```
Calculate Side a
Calculate side b
Calculate side c
tideHeightA = A.getHeight
tideHeightC = C.getHeight
HeightDifference = Max(tideHeightA,tideHeightC) - Min(tideHeightA,tideHeightC)
UnitOfHeight = HeightDifference (side a + side c)
if Max(tideHeightA,tideHeightC) = tideHeightA
    tideHeightB = tideHeightA - (side c * UnitOfHeight)
else
    tideheightB = tideHeightA + (side c * UnitOfHeight)
TimeDifference = Max(tideTimeA,tideTimeC) - Min(tideHTimeA,tideTimeC)
UnitOfTime = TimeDifference (side a + side c)
if Max(tideHeightA,tideHeightC) = tideHeightA
    tideHeightB = tideHeightA - (side c * UnitOfHeight)
else
    tideheightB = tideHeightA + (side c * UnitOfHeight)
Add tideTypeA, tideheightB, tidetimeB to B
```

So what this algorithm does is first work out each side of the triangle. Then it retrieves the height/time values at each port for a certain day. After that if you divide the difference of those values by side A and side C you have a unit of increment along the line B. Then you determine whether port A's value is the largest or smallest value. If it's the largest you need to take away the unit of increment multiplied by side C. If not the largest, then you add the increment times side C. This way you get a more accurate average of the value by taking relative distance to the closest ports into account. This is an improvement on the second method as it eradicates the need for visual computing and will therefore take a lot less time to compute. The diagram below highlights the point of increments marked along the line and the one that would be selected.

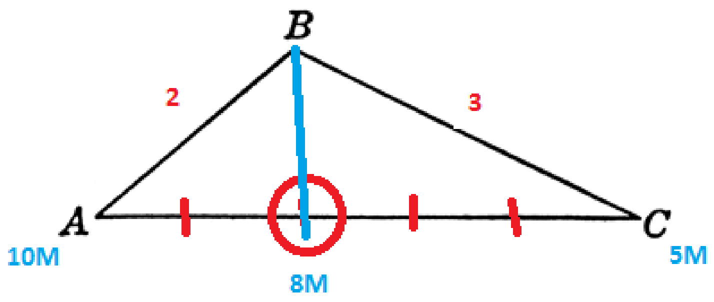


Figure 13 – Triangle Unit measurement to calculate relative distance

As you can see from the blue line of intersection above, it is what we would expect. If the height at Port A was 10M at a given time and the height was 5M, as the point of intersection on Port the value would be 8 since the ratio is 2:3. The incremental unit would be 1 as height Difference (5M) divided by side C plus side A (5). Then because A is the largest value of height, you take away side C * incremental unit, thus getting 8M.

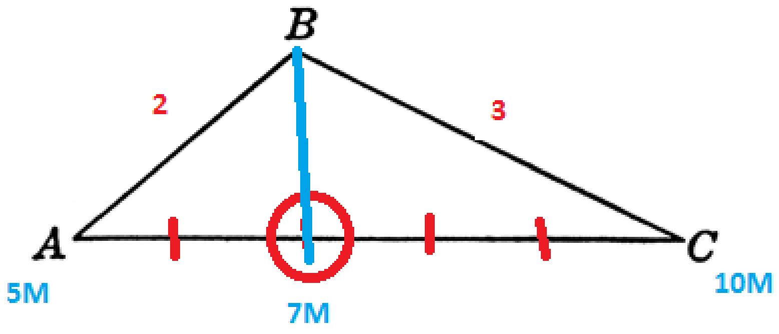
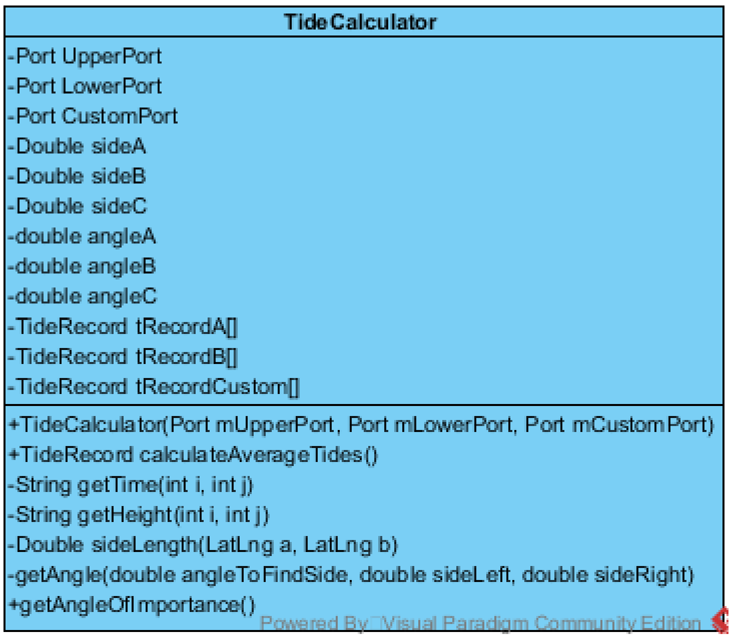


Figure 14 – Reversed example of Triangle Unit Measurement to calculate relative distance

The diagram above shows the same correct result from the algorithm, however this time port A has the smallest value of height. Therefore side C * Unit of increment is taken away, therefore making the resulting value 7M.

tideCalculator

Tide Calculator is essentially the module which handles the prediction of tidal data between two ports. It takes 3 ports into the constructor, upperport and lowerport are the two closest existing ports either side of the customport. Since the algorithm (as detailed in a later section) uses a triangle between the locations of these 3 ports, an explanation of how it works is detailed in the next section



```
TideCalculator
-Port UpperPort
-Port LowerPort
-Port CustomPort
-Double sideA
-Double sideB
-Double sideC
-double angleA
-double angleB
-double angleC
-TideRecord tRecordA[]
-TideRecord tRecordB[]
-TideRecord tRecordCustom[]
+TideCalculator(Port mUpperPort, Port mLowerPort, Port mCustomPort)
+TideRecord calculateAverageTides()
-String getTime(int i, int j)
-String getHeight(int i, int j)
-Double sideLength(LatLng a, LatLng b)
-getAngle(double angleToFindSide, double sideLeft, double sideRight)
+getAngleOfImportance()
```

- Port UpperPort: One of the ports for which data is known, considered the upper bound
- Port LowerPort: One of the ports for which data is known, considered the lower bound
- Port CustomPort: Port for which data is unknown and is to be calculated by this class
- Double sideA: Length of side A which is the opposite side of UpperPort's location

- Double sideB: Length of side B which is the opposite side of the CustomPort's location
- Double sideC: Length of side C which is the opposite side of LowertPort's location
- Double angle A: Value in degrees of the angle where side B and C meet at UpperPort
- Double angle B: Value in degrees of the angle where side A and C meet at CustomPort
- Double angle C: Value in degrees of the angle where side A and B meet at LowerPort
- TideRecord[] tRecordA: 1 Weeks' worth of data scraped from Upper Port
- TideRecord[] tRecordB: 1 Weeks' worth of data scraped from Lower Port
- TideRecord[] tRecordCustom[]: 1 Weeks' worth of estimated data will be stored here

---

- TideCalculator(): Constructor which sets the port variables to their corresponding place in the triangle, calculates their side length and their angles
- TideRecord[] calculateAverageTides: Method that performs the estimation algorithm and returns the result in a TideRecord array with one weeks' worth of estimation results
- String getTime(): Estimates the time given the index i, which determines the day position in the records and at j which determines the tuple position in the records and returns value as a String
- String getHeight(): Estimates the height given the index i, which determines the day position in the records and at j which determines the tuple position in the records and returns value as a String
- Double sideLength(): Determines the sideLength of a triangle given two coordinates
- Double getAngle(): Determines the angle of a point in a triangle given two coordinates
- Double getAngleOfImportance(): Returns the angle at the custom port, this method is needed as it's the way I'll be categorising results from tests.

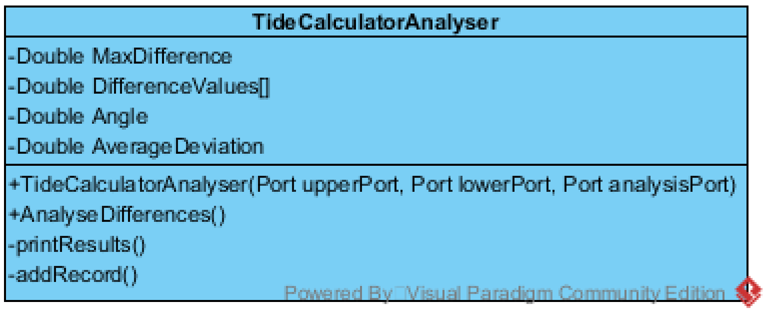
4.2.3 Tide Calculation Analysis Algorithm

As stated by the National Tidal and Sea Level Facility, tides are not the same around the entire coast of Britain due to 'Shape of coastline and bathymetry (water depth)'. 'When tidal wave crosses onto shelf seas the wave's speed decreases and wave is refracted by the local bathymetry.' R8

So assuming port B, the location at which the user wanted to find the tidal data, was either very far away from the pre-existing ports or had a very steep angle at the joining lines from the existing port this prediction is quite likely to be a larger way off. Theoretically, the closer the angle at B is to 180° the closer it is to lying on the line from A to C which stands to reason the prediction is likely to be more reliable. This is why there is a need to calculate and provide the erroneous bounds associate with predicting the tides.

To find the error bounds I'll perform the Tide Calculation Algorithm, passing into it 3 ports as usual. However I'll pass a known port, with retrievable records, as the custom port. This way I can compare the estimated data with actual data to determine accuracy. As aforementioned the angle at which the User-defined location meets the two known-ports immensely affects the precision of estimated results. To allow consideration and adjustment for this I was going to categorise the results into disjoint, contiguous buckets of numbers from 180 downwards with intervals of 5. The number to determine what bucket the results are stored in would have been derived from the angle at the bespoke port. This could be done by using the public 'getAngleOfImportance' method in the Tide Calculator Module (Section 4.2.3).

I've elected for the results to be stored along with the angle, a record consisting of a maximum difference value, a minimum distance value and an average deviation. Therefore as an improvement on the bucket, each grouping will be a record self-containing all results added into it. E.g. it will keep an array of the difference between height/time values from the expected ones and upon having a new value added will determine if it is the new 'range'. Below is the record structure I plan to implement for use in this algorithm.


TideCalculatorAnalyser
-Double MaxDifference
-Double DifferenceValues[]
-Double Angle
-Double AverageDeviation
+TideCalculatorAnalyser(Port upperPort, Port lowerPort, Port analysisPort)
+AnalyseDifferences()
-printResults()
-addRecord()
Powered By Visual Paradigm Community Edition

TideCalculatorAnalyser will then print the results which I will keep a record of in Excel. This way I can determine legitimacy of the algorithm's precision. The 3 ports that are passed into the constructor have their tide records retrieved from the internet. Then the 'analysisPort' will have records estimated by the Tide Calculator and compared against the actual results from the third port as it will be an existing port.

4.3 Considerations

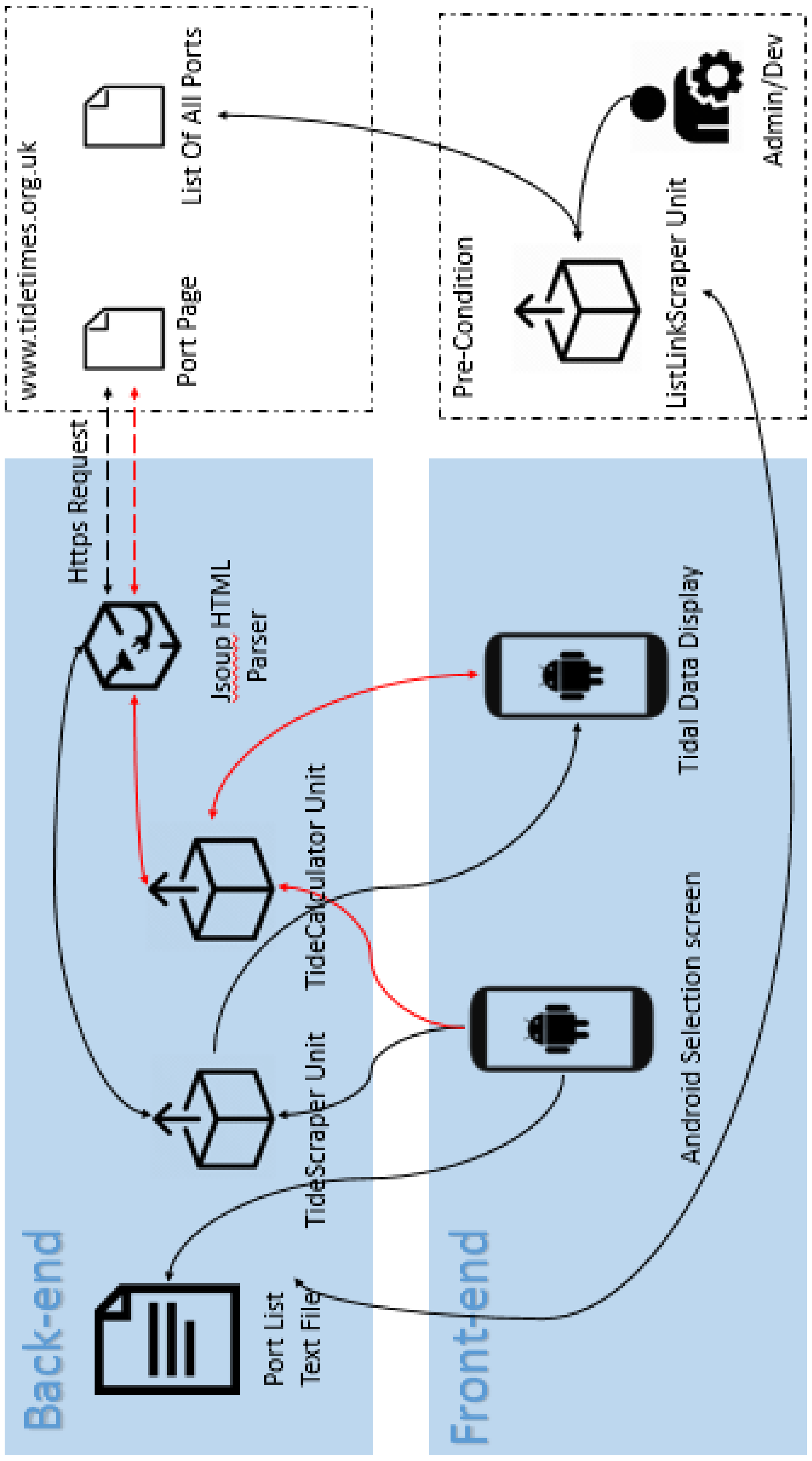
4.3.1 Data acquisition

To acquire the relevant tidal data on the application, I deliberated of having a server perform the web-scraping and necessary formatting of data for retrieval. The mobile app would then connect to the server to get the weeks' data via SSH. For example I was going to create a JavaScript web-scraper to gather a weeks' worth of results on the schools Apache server (given permission). This would then parse the data and store it in a way that it could be easily retrieved I.E. an SQL database. This had many benefits such as reducing the amount of data the users would have to retrieve and therefore also help to conserve battery life; these are key considerations for this application. Plus I wasn't sure whether Android had the capability to even perform web-scraping within an application. However after some research I found that due to the ever-increasing capabilities of computational power of Android phones and amount of Wi-Fi coverage partnered with the minor difference it would make in Data Usage that Web-scraping a page and parsing the data in a self-contained module would be sufficient.

4.3.2 Visual computation of Coastline

As mentioned in the Background and research section, tides in the UK are heavily influenced by the shape of the coastline. Further research suggested that the height of the coast (altitude) as well as type of feature (Notches, headlands, bays etc.). Before I discovered the complexity and number of unobtainable, unknown variables I considered a visual computing approach. Unfortunately due to accessible variables and time constraints this solution is unfeasible for me to undertake, although I do think it's an interesting branch to explore given time. This path would also require a geographical/geological expertise to work in cohesion.

4.4 Static Architecture



4.5 Data Flow

4.5.1 Project Data Flow

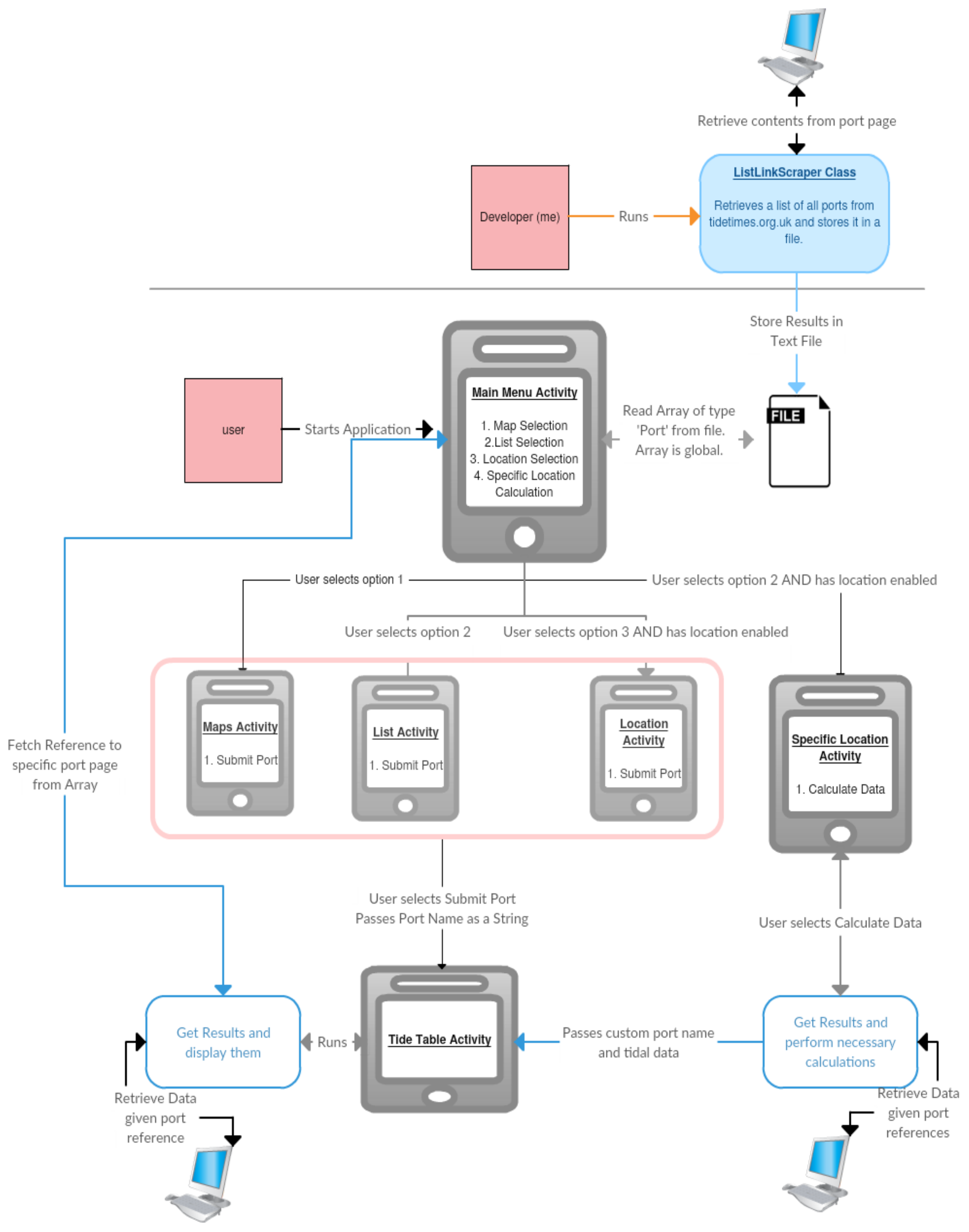

Retrieve contents from port page
ListLinkScraper Class
Retrieves a list of all ports from tidetimes.org.uk and stores it in a file.
Developer (me)
Runs
Store Results in Text File
FILE
user
Starts Application
Main Menu Activity
1. Map Selection
2.List Selection
3. Location Selection
4. Specific Location Calculation
Read Array of type 'Port' from file. Array is global.
User selects option 1
User selects option 2 AND has location enabled
User selects option 2
User selects option 3 AND has location enabled
Maps Activity
1. Submit Port
List Activity
1. Submit Port
Location Activity
1. Submit Port
Specific Location Activity
1. Calculate Data
Fetch Reference to specific port page from Array
User selects Submit Port Passes Port Name as a String
User selects Calculate Data
Get Results and display them
Runs
Tide Table Activity
Passes custom port name and tidal data
Get Results and perform necessary calculations
Retrieve Data given port reference
Retrieve Data given port references

Figure 15 – Android and End User Data Flow Diagram

4.5.2 Tide Calculation Data Flow

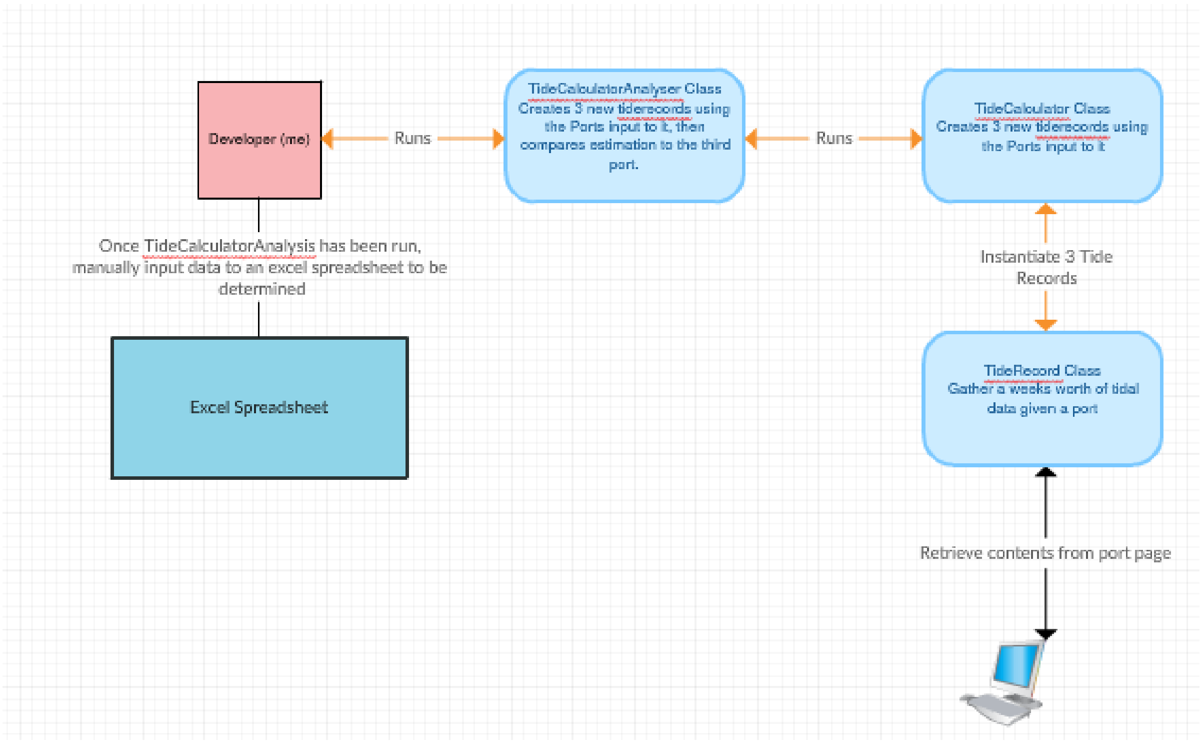
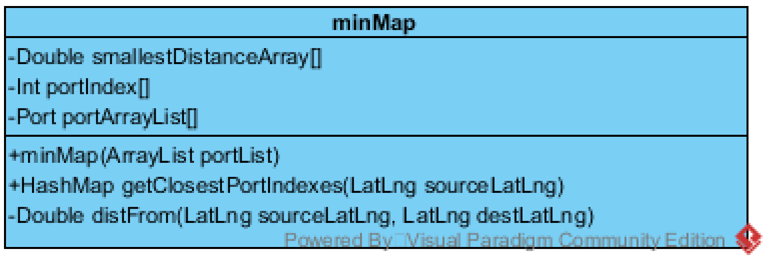

Developer (me)
Runs
TideCalculatorAnalyser Class Creates 3 new tiderecords using the Ports input to it, then compares estimation to the third port.
Runs
TideCalculator Class Creates 3 new tiderecords using the Ports input to it
Once TideCalculatorAnalysis has been run, manually input data to an excel spreadsheet to be determined
Instantiate 3 Tide Records
Excel Spreadsheet
TideRecord Class Gather a weeks worth of tidal data given a port
Retrieve contents from port page

Figure 16 – Tide Calculation Data Flow Diagram

5 – Implementation

5.1 minMap

When designing my modules and necessary algorithms, I had an unforeseen problem that I would need to be able to calculate a list of closest ports to a given location. The given location in question would be supplied by user-location obtained through the Google Location APIs or a selected marker on the Google Maps API. Hence I created a minMap module with the following UML design.


minMap
-Double smallestDistanceArray[]
-Int portIndex[]
-Port portArrayList[]
+minMap(ArrayList portList)
+HashMap getClosestPortIndexes(LatLng sourceLatLng)
-Double distFrom(LatLng sourceLatLng, LatLng destLatLng)
Powered By Visual Paradigm Community Edition

- Double[] smallestDistanceArray: Array of size 6 which holds the values of the smallest distances in metres, in ascending order (smallest distance first)
- Double[] portIndex: Array of size 6 which holds the index of the closest ports (Closest first). Position in array corresponds to the smallestDistanceArray
- Port[] portArrayList: ArrayList of type Port which contains a copy of the ports from the main program
- minMap(ArrayList portList): Constructor which sets the portArrayList as a copy of portList
- getClosestPortIndexes(LatLng sourceLatLng): Returns a hashmap with 6 entries of the 6 closest port Indexes (as key) and their distance from SourceLatLng (as value).

- Double distFrom(): calculates and returns the distance (in metres) from sourceLatLng to destLatLng

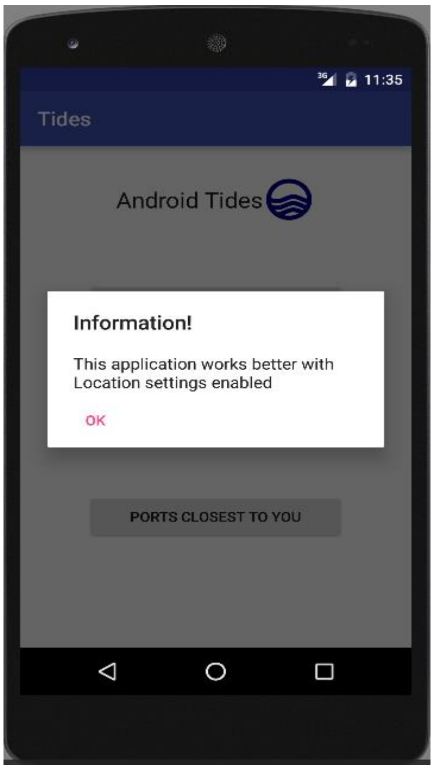
minMap is instantiated by passing an arrayList of the ports from the main project into its constructor. To retrieve a HashMap of the 6 closest locations getClosestPortIndexes is called. I chose a Hashmap because being able to retrieve the port through the Index as well as supplying a user of the relative distance is far more useful.

distFrom, due to a lack of software documentation, was originally difficult to implement a distance determining algorithm. Firstly I would calculate straight-line distance using Pythagorean Theorem which produced correct results in that it found the nearest ports but not any kind of unit for distance. Secondly, I found a formula that took into account the Earth's Radius and was an official formula for calculating distances between points on a map [r13]. This worked perfectly but after deeper delving into the Location class discovered a 'distanceTo' between 2 locations performed exactly the same function so instead of *reinventing the wheel* as it were I settled on using this.

5.2 Screen by screen walkthrough (Android Implementation)

*Note: an 'Activity' in android is essentially a new window with a single focused content view.

5.2.1 Main Menu



This application works better with Location settings enabled

Figure 17 – Main Menu Activity Screen

When the user loads the application, the Main Menu in its initial state displays an informative dialog, just once throughout the lifetime of the program thanks to a Boolean variable, to the user to remind them that the application functions better with locational settings enabled. The Main Menu, as the main parent activity means that while the application is running the contents of this activity will still exist in a device's memory. To overcome the need of continuously having to repopulate a list of the ports (which every activity will need access to) or passing this list between activities, the complete list of known ports is held in a globally visible variable:

```
public static ArrayList<Port> portArray;
```

Main Menu, when created, reads the 'portList.txt' file which was created by the listLinkScraper. Main Menu takes the contents of this text file and converts each line into a port Data Structure. Since the port is only read from, it's also safe to access from other activities as follows:

```
MainMenu.portArray
```

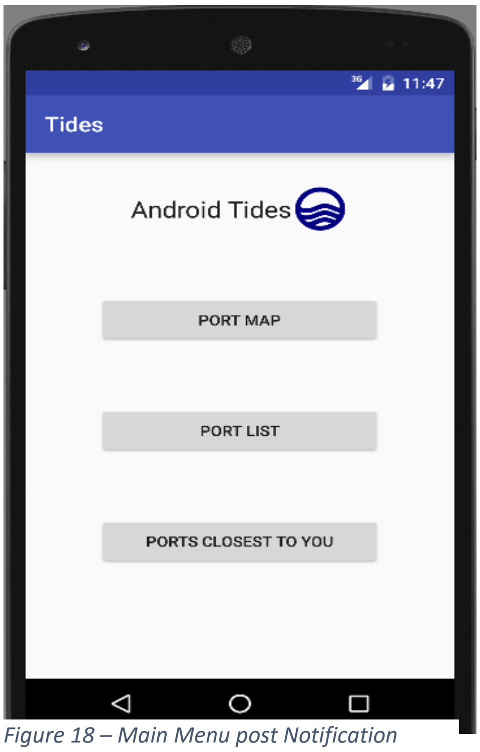


Figure 18 – Main Menu post Notification Dismissal

The dialog is dismissed by a simple click event displaying the actual bulk of the Main Menu.

There are now 3 buttons instead of 4 as originally planned in the HCI Design section. Since the proposed 'specific tide calculator' and 'map selection' modules both utilised google maps, feedback I received suggested I combine them. Therefore the 'Port Map' loads an activity which pools those two modules together.

The button's create a new intent and begin a new activity. In doing so the activities created as a result end up being children of the Main Menu thus keeping good user-control and consistency in the path you follow into the application. This also ensures that the back-stack model as discussed in the design stage is adhered to which is what the user would typically expect from a mobile application when navigating backwards and forwards between activities.

5.2.2 Port List

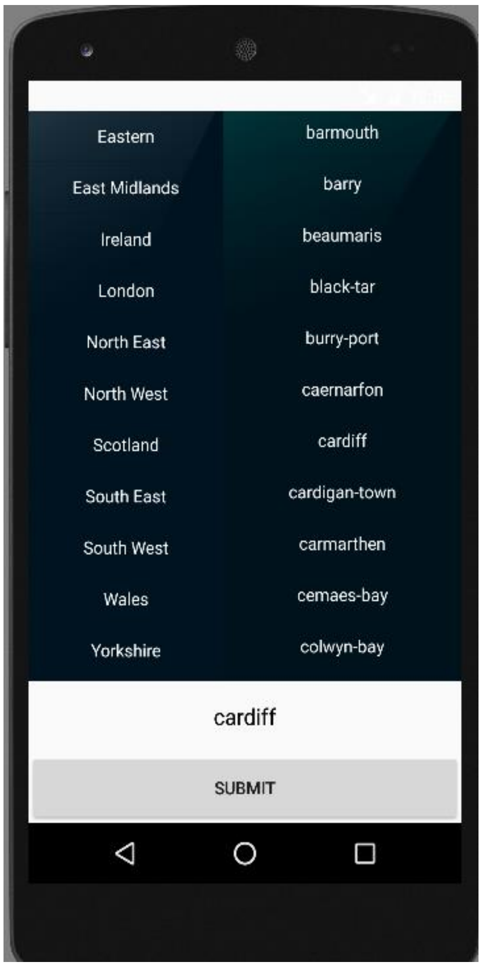


Figure 19 – List Selection Activity Screen

After deciding to categorise the Ports based on their geographical location and regional situation, I had to find a method to do this. I considered adding an extra variable to the *Port* record which would hold the Region as a string value. However I quickly rejected this as it would mean there would be a lot of duplicate and redundant data needlessly being stored. Secondly I found out that the Google Maps API has a Polygon class which physically draws a polygon on the map. It has a method which even checks if a LatLng point sits within the shape. However I would need to initialise a map which takes a little while to do and was wholly less efficient than the method I decided on.

The method I settled on is essentially the same method as the Google Maps Polygonal method, except there is no need to load the unnecessarily large and computationally expensive map. I went on google maps in browser and made a polygonal overlay according to a regional map [R14]. From here I was able to extract the coordinates X and Y positions, which I put in constant variables within the List Activity. After this I searched the internet for a mathematical way to determine whether a 2D point was contained within a polygon given its bounds. After a while of searching I found one and implemented it as follows:

```
private boolean contains(double x, double y, double xarr[], double yarr[]) {
    boolean oddNodes = false;
    int j = xarr.length - 1;
    for (int i = 0; i < xarr.length; i++) {
        if (yarr[i] < y && yarr[j] >= y || yarr[j] < y && yarr[i] >= y) {
            if (xarr[i] + (y - yarr[i]) / (yarr[j] - yarr[i]) * (xarr[j] - xarr[i])
< x) {
                oddNodes = !oddNodes;
            }
        }
        j = i;
    }
    return oddNodes;
}
```

The variable x denotes the latitude position of the Port being checked, the variable Y denotes the longitude position of the Port being checked and the xarr and yarr are the values of the polygons vertices. Below is an example of the xarr and yarr value for London to give an idea of how the polygon is mathematically described. Each side of the polygon is compared to the longitude coordinate of the point. Then a list of nodes is produced where each node crosses the Y threshold. If there are an odd number of nodes either side of the given point then it is inside the polygon.

```
private final double irelandX[] = {55.7518494, 54.4317129, 53.26521290000001,
51.1517861, 51.6589266, 54.4955675, 55.3915936, 55.7518494};
private final double irelandY[] = {-7.9101569000000005, -10.2612318, -
11.447753899999999, -10.0415046, -6.328125700000001, -5.0976569, -6.0424805, -
7.9101569000000005};
```

All X and Y arrays must being and end with the same value to ensure that the polygon is closed otherwise this causes huge problems. The first value in X array would correspond with the first value in the Y array as Latitude and Longitude of the apexes respectively. The contains method is called and the regions are assigned as follows using a simple if-else implementation of a case statement:

```
Thread popRegionThread = new Thread() {
    @Override
    public void run() {
        for (int i = 0; i < MainMenu.portArray.size(); i++) {
            double x = MainMenu.portArray.get(i).getLatLng().latitude;
            double y = MainMenu.portArray.get(i).getLatLng().longitude;
            if (contains(x, y, northEastX, northEastY)) {
                regionArray[i] = "North East";
            } else if (contains(x, y, northWestX, northWestY)) {
                regionArray[i] = "North West";
            } else if (contains(x, y, southWestX, southWestY)) {
                    regionArray[i] = "South West";
                } else if (contains(x, y, southEastX, southEastY)) {
                    regionArray[i] = "South East";
                } else if (contains(x, y, londonX, londonY)) {
                    regionArray[i] = "London";
                } else if (contains(x, y, yorkshireX, yorkshireY)) {
                    regionArray[i] = "Yorkshire";
                } else if (contains(x, y, westMidlandsX, westMidlandsY)) {
                    regionArray[i] = "West Midlands";
                } else if (contains(x, y, scotlandX, scotlandY)) {
                    regionArray[i] = "Scotland";
                } else if (contains(x, y, irelandX, irelandY)) {
                    regionArray[i] = "Ireland";
                } else if (contains(x, y, easternX, easternY)) {
                    regionArray[i] = "Eastern";
                } else if (contains(x, y, eastMidlandsX, eastMidlandsY)) {
                    regionArray[i] = "East Midlands";
                } else if (contains(x, y, walesX, walesY)) {
                    regionArray[i] = "Wales";
                } else regionArray[i] = "Other";
            }
        }
    };
```

5.2.3 Location List

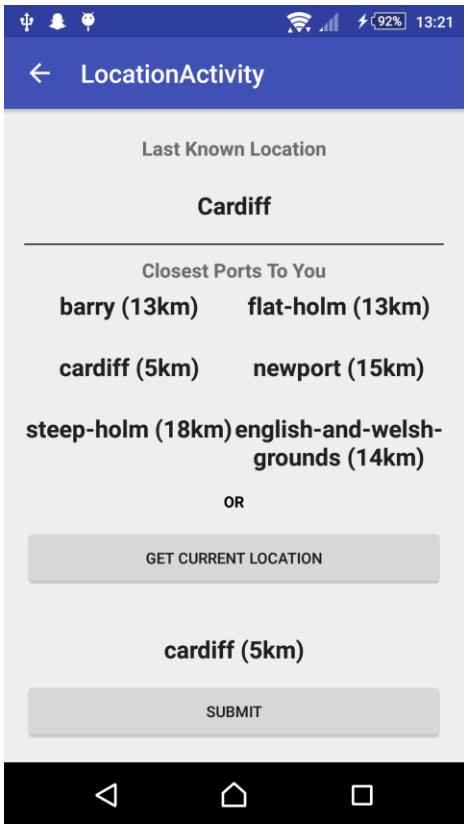


Figure 20 – Location Selection Activity Screen

Location List was originally intended to just select the closest port to the user, as is the case in the pre-existing applications I researched. However, it seemed more appealing to offer a small list of the closest ports so the user has some input and interactivity with the matter.

When this activity is created, an attempt is made to connect to the google location services. From this a last known location value is retrieved and displayed at a *locale* level. I.E. If I was in Cathays, Cardiff the location would only display the city or Area the user was in (as image left shows). To achieve this you have to convert between several Google Data structures to get the functionality. The Geocoder module provided by google allows retrieval of an address, from which you can obtain postcode, locale, country etc.

The reason I display the last known location on this screen is because if it's wrong I.E. you're in Swansea but your last known location is Cardiff, then you can press the Button labelled *Get Current Location*. This will refresh the location and get the users whereabouts. There are 2 accuracies Google offer with locational services FINE location or COARSE location. FINE location will retrieve a more pinpoint location but use more resources, more battery power and potentially more internet power to do so. COARSE provides a, still very reasonable, more vague location estimation which is more than sufficient enough for this application, since it is highly, highly unlikely to affect the list of ports retrieved and saves on unnecessary resource usage. The way the location is retrieved is based on a broadcast system which when a location is found, is broadcast back to the current intent (application) and displays a loading dialogue in the meantime. The application achieves this by using the *PendingIntent* method of location retrieval since the app is granted the same permissions and identity as the rest of the application. Specifying permission to access a user's location is legally obligatory so it is made known and stored in the application permissions, thus the need for its child screen to know this has been allowed.

5.2.4 Map Selection & Specific Tide Calculator

Originally, retrieving a port via map selection and getting a tide at a custom point were separate activities that both employed google Maps. However after a constructive discussion with my supervisor we realised that the modules were executed in such similar ways it would be constructive to merge the modules. This way it keeps the aesthetic to a minimal design as is recommended by one of Nielsen's heuristics.

Another factor in deciding to combine the activities is that I've discovered a more user-friendly way to allow them to select a custom port by allowing them to place and move their own marker. Wherever this marker is dropped, it highlights the closest ports giving real-time feedback and enhanced visibility of the system status.

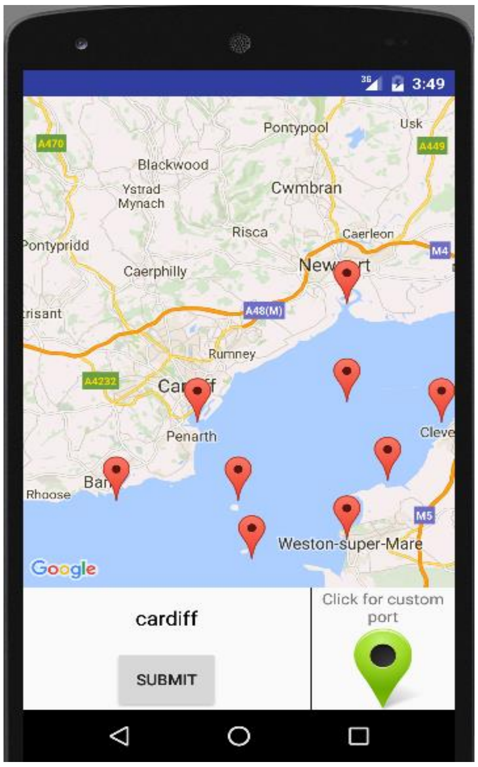


Figure 21 – Map Selection/ Tide Calculator Activity Screen

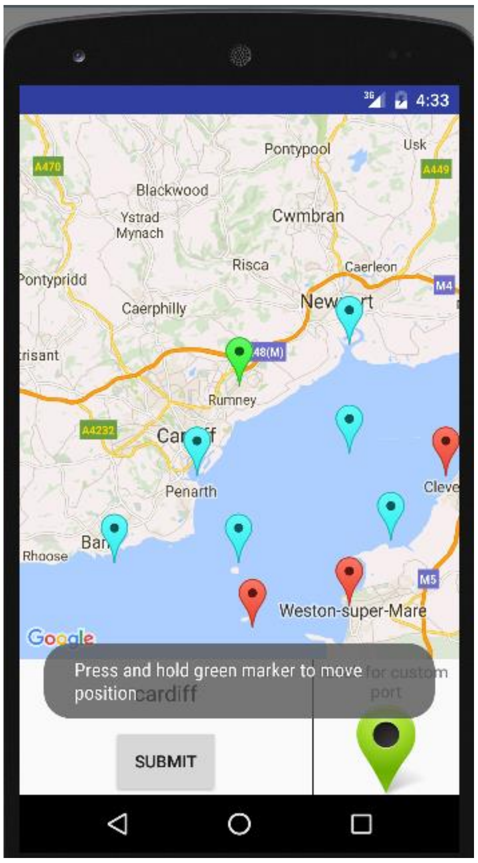


Figure 21 – Map Selection/Tide Calculator Activity Screen with Custom Marker

When the google maps API is created, the map is populated with markers plotting each port in the Port array. Users should intuitively know that by tapping one of these markers they select a port, and in doing so the name of the port they last clicked will be displayed below the map.

Each marker is dynamically created, but to be able to manipulate them and change them later on a HashMap <String, Marker> holding a reference to each marker is held in memory where the Key is the name of the port. The hashmap is necessary as highlighting specific markers is something that will be performed as shown by the next state when a custom port is to be selected.

After clicking the green marker in the bottom-right hand corner, a marker is dynamically created and placed in the centre of the position on the map. A hashMap of the closest ports is returned from a minMap class. Code in this module then finds those ports and highlights them (blue). By highlighting them it gives user confidence that the application is working and better visibility of the system status. Also a toast is displayed to provide help to the user as the way you move the marker might not be that instinctive.

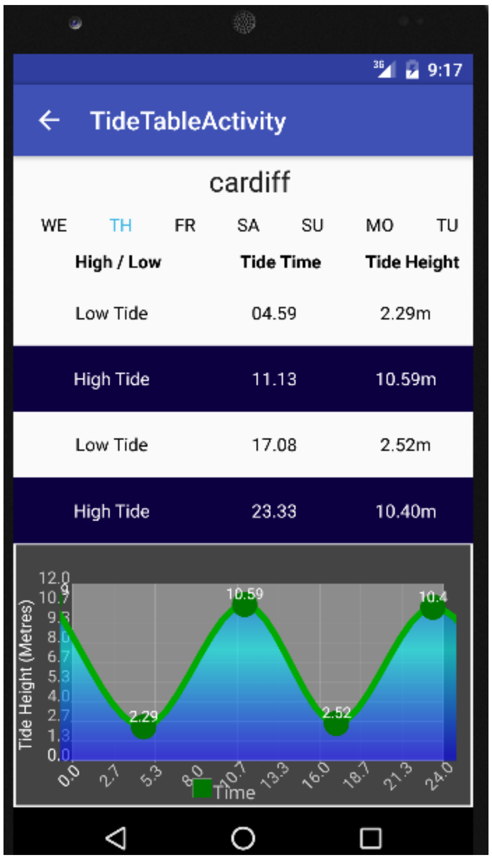
Upon holding and dragging the marker, the old blue markers where the closest ports were highlighted are reverted back to normal marker colour. Wherever the new marker ends up the new 6 closest will be emphasised. The 6 closest ports to the user's custom marker are always held in memory as they will be utilised later on in the program, if the user has chosen to submit their custom port.

Since after selecting a custom port, to get an estimation you need 2 ports either side to create data. As cited previously, it makes sense that if you draw a triangle between the 3 ports and obtain the angle at the bespoke location; the closer that angle is to 180° then the closer it is to lying on a perfect line and therefore the more accurate the estimation. Hence to select the data I have created the following algorithm to iterate through all possible pairs of ports plus the custom port to select the Optimal ports to achieve the best results:

```
private int[] bestGuessPorts(Port customPort){
    int Ports[] = new int[2];
    int upperPort = 0;
    int lowerPort = 0;
    Double angle = Double.MIN_VALUE;
    for (Integer key : hMapClosest.keySet()){
        for (Integer secondKey: hMapClosest.keySet()){
            if (key != secondKey){
                TideCalculator t = new
TideCalculator(MainMenu.portArray.get(key),MainMenu.portArray.get(secondKey),custom
Port);
                if (t.getAngleOfImportance() > angle){
                    upperPort = key;
                    lowerPort = secondKey;
                    angle = t.getAngleOfImportance();
                    Log.e("angle",Double.toString(angle));
                }
            }
        }
    }
    Ports[0] = upperPort;
    Ports[1] = lowerPort;
    return Ports;
}
```

This algorithm performs exactly that, after iterating through every possible permutation of the closest 6 ports (excluding duplicate pairings as a matrix would) to find the 'fittest' pairing. It will choose the largest angle, the closer to 180° the better and return the indexes of the Port. This method is called from the submit button method which is the same in all selection activities, except this one will also send the two ports to the tide-data display activity. That activity will know how to handle a custom port and its two *bound ports* as opposed to one that already exists.

5.2.5 Tidal Data Display



High / Low	Tide Time	Tide Height
Low Tide	04.59	2.29m
High Tide	11.13	10.59m
Low Tide	17.08	2.52m
High Tide	23.33	10.40m

Figure 22 – Tide Data Activity Screen

Tide selection is passed in either a Port Name as a string, or a Custom port plus the indexes of the 2 most optimal ports as calculated from the Maps/Custom location screen. If a Port Name has been passed in then this activity finds the corresponding name in the Port Array and retrieves the port data so the data can be scraped and displayed.

If a custom port was selected by the user, it retrieves the two ports that have been selected by the two indexes provided from the previous maps activity. From this a TideCalculator class is instantiated and a weeks' worth of tidal data is estimated via the method. No matter which way the tide records are retrieved, a loading box is displayed throughout the duration as accessing internet records could take time and it's important to give feedback to the user.

To switch between the days' (in which the current day is always highlighted in blue) table of data and days graph, I made an *onTouch* event listener which detects when the layout is pressed and when the layout is un-pressed. By detecting the location on where the user has touched and where the location of the touch is released, I was able to implement detecting a significant swipe from the user. Using this and being able to detect the direction of the swipe from positive and negative X values; the way the user switches from day to day is by swiping in the direction they wish. This creates a bridge between the real-world and the application and is intuitive, both key features of Nielsen's heuristics and essential to creating a desirable informative application.

Lastly I used a free external API called AndroidPlot to create the graphs for each day. The time values are put into an array of X values and the corresponding height values are put into an array of Y values which are then mapped onto the graph. The graph then retrieves an extra result either side of the current day's data so a line of best-fit can be drawn across the points; a feature provided by all tidal data providers which allows a user to get an approximation of the tide at a given time.

General Implementation

Throughout the android implementation there are coding sections that would need slightly more computing power to complete than generic functions. Since Android's main thread generally handles the creation and employment of the User Interface objects, I've moved all intensive or large chunks of work into background threads. The threads can be run while the UI is being updated allowing for much better performance without halting the flow of the program. In using Asynchronous threads if a section of code needs to be executed and potentially could consume time, before updating the UI, I've displayed a loading screen informing the user of this while the program waits for execution to end .

5.3 TideCalculator

The TideCalculator class performs the key, unique component of the project. It is able to provide an educated approximation of tides between stations based on several fitness criteria. Firstly, using a method that determines the angle of importance (angle at the selected custom location) the truest 2 ports out of the 6 closest ports are selected based on how close they are to 180° (A 180° angle would mean the port was on an exact straight line therefore producing the best guess). After 2 ports are selected, an algorithm as described in some detail in the Design phase using relative distances, Pythagoras theorem and a form of triangulation manages to produce comparative values of the tide time and tide height. However there was an unforeseen problem in my design, I assumed that all ports had the same amount of tidal records. Unfortunately they don't so I had to add some code:

```
public TideRecord[] calculateAverageTides () {
    tideScraper tScraperA = new tideScraper(UpperPort);
    tRecordA = tScraperA.getTide();
    tideScraper tScraperB = new tideScraper(LowerPort);
    tRecordB = tScraperB.getTide();
    tRecordCustom = new TideRecord[7];
    for (int i = 0; i < 7; i++) {
        ArrayList<String> aLTideType = new ArrayList<>();
        ArrayList<String> aLTideTime = new ArrayList<>();
        ArrayList<String> aLTideHeight = new ArrayList<>();
        if (tRecordA[i].getRecordCount() == tRecordB[i].getRecordCount()){
            for (int j = 0; j < tRecordA[i].getRecordCount(); j++) {
                if
(tRecordA[i].getTideType()[j].equals(tRecordB[i].getTideType()[j])) {
                    String sHeight = getHeight(i, j);
                    String sTime = getTime(i, j);
                    aLTideType.add(tRecordA[i].getTideType()[j]);
                    aLTideHeight.add(sHeight);
                    aLTideTime.add(sTime);
                }
            }
        }else {
            TideRecord smallerRecord;
            TideRecord largerRecord;
            SimpleDateFormat tideTimeParser = new SimpleDateFormat("HH.mm");
            Date dRecordSmall = new Date();
            Date dRecordLarge = new Date();
            if (Math.min(tRecordA[i].getRecordCount(),tRecordB[i].getRecordCount())
== tRecordA[i].getRecordCount()){
                smallerRecord = tRecordA[i];
                largerRecord = tRecordB[i];
            } else{
                smallerRecord = tRecordB[i];
                largerRecord = tRecordA[i];
            }
            for (int j = 0;j<smallerRecord.getRecordCount();j++){
                for (int q = 0;q<largerRecord.getRecordCount();q++){
                    try {
                        dRecordSmall =
tideTimeParser.parse(smallerRecord.getTideTime()[j]);
                        dRecordLarge =
tideTimeParser.parse(largerRecord.getTideTime()[q]);
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }
                    if (Math.max(dRecordSmall.getTime(),dRecordLarge.getTime()) -
Math.min(dRecordSmall.getTime(),dRecordLarge.getTime()) <= 40*60*1000){
                        if (smallerRecord == tRecordA[i]) {
                            String sHeight = getHeight(i, j,q);
                            String sTime = getTime(i, j,q);
                            aLTideType.add(tRecordA[i].getTideType()[j]);
                            aLTideHeight.add(sHeight);
                            aLTideTime.add(sTime);
                        } else{
                            String sHeight = getHeight(i, q,j);
                            String sTime = getTime(i, q,j);
                            aLTideType.add(tRecordA[i].getTideType()[q]);
                            aLTideHeight.add(sHeight);
                            aLTideTime.add(sTime);
                        }
                    }
                }
            }
        }
        String[] sATideType = aLTideType.toArray(new String[aLTideType.size()]);
        String[] sATideHeight = aLTideHeight.toArray(new
String[aLTideHeight.size()]);
        String[] sATideTime = aLTideTime.toArray(new String[aLTideTime.size()]);
        tRecordCustom[i] = new
TideRecord(CustomPort.getName(),sATideType,sATideTime,sATideHeight);
    }
    return tRecordCustom;
}
```

To combat this, the code highlighted in blue determined the port with the smaller amount of records for that day. It then compares all of its records on that day with the records from the other port on the same day by time. If the time is within 40 minutes of each other (as highlighted in yellow) then those tidal records correspond and an estimation can be gathered from them. To provide methods for *getTime* and *getHeight* where I needed to change the index to match different records, I created duplicate methods with a 3rd variable for specific index of the 2 port records. This will always produce as many records for that given day as the smallest amount of records. Before I implemented this method, since the first technique only performed approximation if the amount of records were the same – if the records differed it would return no results and show a blank table and produce an error in the project. This way ensures there are no errors and that all possible records that can be obtained, are.

6 Results and Evaluation

6.1 Testing

Given the project's nature I am unable to really test correctness via the method of unit-testing. I will be testing my application vs. the specification as well as the HCI and every component within the application to see that everything works as intended. My application should not produce any errors or erroneous results at any point during its lifetime. As a side-note, extensive testing has taken place throughout when adding any method to my application.

My external modules such as the tide calculation, web scraping modules and data structures however can be unit tested – incredibly useful for finding out that the implementation works as envisioned. They can be unit tested since I can feed in values and I know what the expected results that the implementation should return.

Furthermore, my Tide Calculation needs to be tested for degree of accuracy and the results need to be analysed and critiqued. This algorithm is completely bespoke and was designed from research into the subject area. Results will be categorised by buckets of their values from which a range and average to verify validity.

6.1.1 Implementation vs. Specification Testing and Evaluation (Android Modules)

To test whether my application has met the criteria it was designed to meet, I will consider the original specification to the essential requirements and search for the specific part in my employment that fulfils them. Some activities in the Android application may meet several of my original requirements so I'll combine them. The reason I'm testing every component within the application and the data flow between activities is because an error in an Android application is very unprofessional and could cause all sections that should work to fail. If there are any fails or partial test failures they will be addressed in the evaluation section that follows.

Requirements testing

*For tests proving the following, see Appendix A – Android Requirement Testing

Requirement 1: Application should offer choice to display a map with UK tide ports charted from which user can select the tide they wish by clicking on the marker

Requirement 8: Application should provide a way to estimate tidal data by the user selecting the 2 nearest stations

The Maps activity in android not only displays every single Port as a clickable marker, it allows the user to drop and drag a custom marker from which the 2 fittest ports are automatically selected. After the user has submitted it provides an estimation at the custom marker's location thus meeting both of these requirements. The components testable in the Maps screen are the markers, the custom marker button which should plot a draggable custom marker at the center of the screen and the port submission button.

The markers which are automatically populated from the list of ports when any clicked always updates the text to display the name of the port at that marker thus correctly working. In terms of testing, when the text from this is selected and passed into the activity to retrieve the port it never returns (and will never return) an error as it was retrieved from the Port list in the first place. Thus giving confidence of complete error prevention in selecting a known port from a marker.

The custom marker is draggable, upon clicking it the text displays 'Custom Marker' so the user knows that this marker has been selected. Furthermore as added functionality, clicking the button that creates the custom marker if the marker has already been created centers the Map on wherever the custom marker is situated. The custom marker was easily testable as wherever it was dragged it highlighted the 6 nearest ports (see appendix B) which was its function.

Lastly the port Submission button has 3 possible states before it has been clicked. Firstly, if no port has been selected it shouldn't go to the next screen. Testing this confirms that not only does it not navigate to the Tide Data screen (as no port has been selected) a small help *Toast* is displayed informing the user that no Port has been selected. Secondly if an established Port is selected it should pass the name of that port to the next Activity. Lastly if Custom port is selected it should calculate the two ports with the steepest angle and pass the indexes of these to the next activity.

Selecting a known port such as Cardiff and passing it to the next activity works correctly as the data from Cardiff's port is displayed within the Tide data screen. Selecting a custom port between Cardiff and Barry as shown in appendix B on the screen and within the Logs proves that this works correctly.

Requirement 2: Application should offer choice to display a list of the UK tide ports by name

Requirement 3: List of tides should be categorised geographically by region (I.e. Southwest, Northeast, Scotland etc.)

The List selection activity displays all of the regions of the UK. After a region is selected, a list of all the tide ports within that region are displayed. Both of these are in the form of scrollable, vertical lists. This activity at its core was designed to fulfil both of these requirements, and does so efficiently with absolutely no delay in populating each list. This module has been extensively tested both throughout development and after development. Every single port is displayed and is contained in one of the regions, and every region contains at least one value so these requirements are extensively met.

The lists, despite the amount of data contained in them, are completely responsive due to the 'recycling' feature which is why I chose them. The list component I implemented dynamically stores/retrieves the data via an adapter so only the number of values on screen (plus one above and one below which are unseen) are loaded into memory and displayed. This saves the UI having to load all values into memory, something that's implausible with Smart Phones and has been taken into consideration.

The only component that needs testing after both vertical lists is the submit button, which when no port is selected tells the user to select a port in an informative *toast*. Once a port has been chosen, it passes the name of that port into the next activity as intended.

Requirement 4: Application should offer choice to display some of the closest tides to the user's last known location

Since this was the core feature of every application I discovered in my preliminary research, and a useful one at that, this was essential to the program. Hence it was covered in its own screen, the location selection. It shows the closest 6 locations as well as their relative distances. Rigorous testing with the location enabled on the Android phone proved that it populated the results based on the devices last known location very swiftly. Also refreshing the location with the button produced results quick enough that when the loading screen was displayed, the sole purpose of which was to inform the user location was being retrieved as it can take a long time, was never displayed for more than a third of a second.

However testing this module with the location turned off in Android firstly produces no results for last known location when the activity is created since it can't access the locational services. If you then try to populate the page with the closest ports via the Button to refresh location, an error is displayed. The application recovers from this error but this screen fails its test when the Location is disabled in the Android settings. Upon opening the application a dismissable Dialog informs the user that "This application works better with Location settings enabled" in regards to this.

Requirement 5: Application should display the Tide Height, Tide Time and Tide Type in a tabular format after a port is selected, plus the name of the port that was selected

Requirement 6: Application should plot said Data in requirement 5 in graphical format

Desirable Requirement 2: Application could retrieve results from 6 days in advance as well as the current day 2

These 3 requirements all involve the displaying of tidal data. Ergo it makes sense that these are all implemented within the *Tide Table* activity. As shown in appendix B the tidal data is not only displayed in a tabular format, but also in a graph below it for each day. Furthermore one of my desirable requirements of being able to display a week's worth of data has been met. I decided this was a desirable requirement with actually quite a lot of importance due to most other applications and website displaying this data. There aren't really any components that are testable within this module since it's mostly a display of information, although you have to swipe left and right between day's data.

On swiping right, the data should be of the next day and upon swiping left the data should be the previous day. If it is the first day or last day then you can't swipe left and right respectively since they are the beginning and end. After switching the day via swipe, data is changed as well as highlighting the current day which the records correspond to. Looking at appendix B you can see the behaviour is as expected from the swipe motion.

Requirement 8: The application should have a complete list of UK Tide Times concurrent with tidetimes.org.uk

As of the current date, 29/04/2016, I have a complete list of Tidal Stations concurrent with UK Tide Times list. I can ensure that I have every single UK Port available form them as the *listLinkScraper* module which automatically scrapes a list of every available port they have. This is done from a page listing their Ports from a web page entitled 'All Tide Prediction Locations' consequently giving me confidence in the fulfilment of Requirement 8.

Desirable Requirement 1: Application could store results along with the date the result was retrieved in the devices memory

Unfortunately due to time constraints I was unable to begin thinking about implementing this feature.

Desirable Requirement 3: Application could provide a favourites feature to store their favourite ports 3

Unfortunately due to time constraints I was unable to begin thinking about implementing this feature.

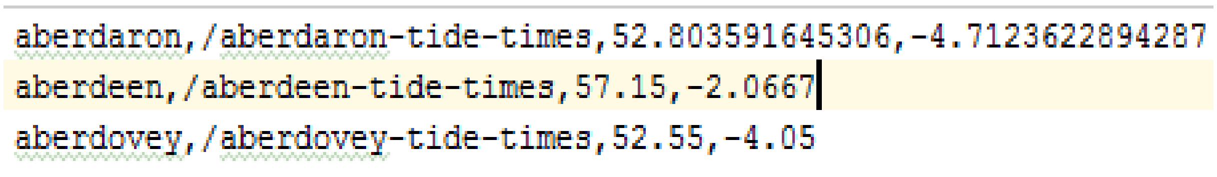
General Evaluation
Overall I've managed to meet all of my essential requirements, however not quite perfectly. As mentioned there was a slight flaw in requirement 4 where the activity does not behave as it should in the event the Android Location setting isn't enabled. Despite displaying a message when the application begins advising the user to enable location settings, there is very minimal error handling when on the Location screen in the event that it is disabled. Critiquing this case, I must say this is a relatively large weakness within an otherwise structurally sound and error-free system. With location settings enabled, all parts of the system are Strong and provide no errors anywhere throughout the system. It was intended that the case of having location settings disabled would be handled by the program but I found difficulty in finding a reasonable method of checking whether the setting was allowed/ denying access until they were turned on.

Despite meeting all my crucial requirements, there are 2 optional ones I unfortunately didn't have time to implement. Firstly, desirable requirement 1 in which all data retrieved was stored along with a date. Using this date and the name of the port the application could check if it exists in memory before attempting to web-scrape it. The nature of which was to save battery-power, processing power and potential data-usage by temporarily storing the results – a technique similar to caching. Not being able to implement this feature does not really impact my application though so it's not a weakness as such.

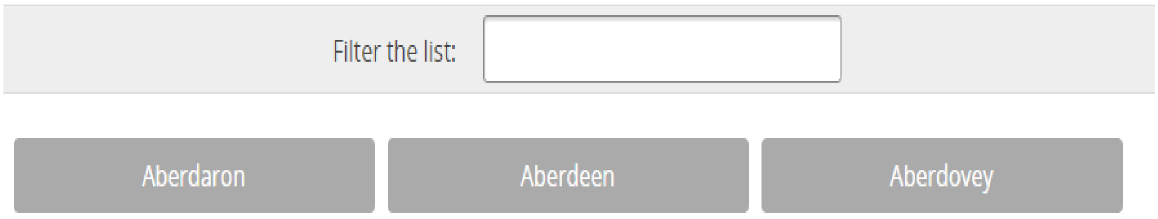
Desirable requirement 3 to add a favourites was also not implemented due to time constraints. Having favourites is quite useful in my type of application to allow a user ease of navigation to what they want. I thought about implementing it, I would only need to save the port index to a text file within the android device which could be retrieved and populated on screen in a list view. The main reason I didn't implement it was to ensure that all essential requirements were working correctly first as a priority. Again I don't think this has much impact on the final product and therefore not convinced it's much of a weakness.

6.1.2 Does implementation perform what its intended to do (External Module Unit Testing)
As important as the specifications testing, without the external modules producing the precise result I anticipate it to the application would be useless. Fortunately the external modules are able to be unit tested. The major benefit and strengths of this testing is that given we know the results that should be produced as an outcome given input data. The unit testing is definitive which means I can determine whether the module I've created works or doesn't work. It's not subjective so either the test passes or fails which can prove to be more useful in determining the success of the deliverables within this venture.

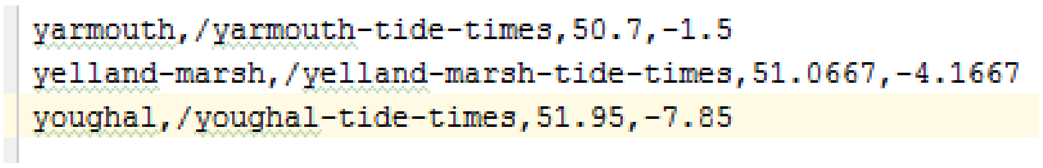
ListlinkScraper
The listLinkScraper module was intended to scrape the Name, hyperlink and latitude/longitude of all ports from tidetimes.org.uk and store these in a text file for use in the project. I can prove this works because the amount of lines in the text file is synonymous with the number of ports that the website claims to provide. Furthermore comparing the start/end of the text file and the website list below proves that all results have been gathered and formatted rightly.


```
aberdaron,/aberdaron-tide-times,52.803591645306,-4.7123622894287
aberdeen,/aberdeen-tide-times,57.15,-2.0667
aberdovey,/aberdovey-tide-times,52.55,-4.05
```

First 3 results in the text file

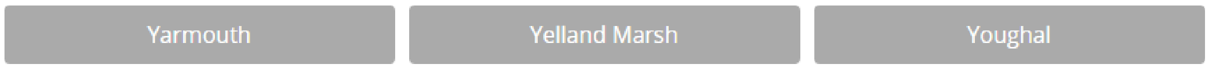


First 3 results from the ‘all Tide Prediction Location’ page



```
yarmouth,/yarmouth-tide-times,50.7,-1.5
yelland-marsh,/yelland-marsh-tide-times,51.0667,-4.1667
youghal,/youghal-tide-times,51.95,-7.85
```

Last 3 results in the text file



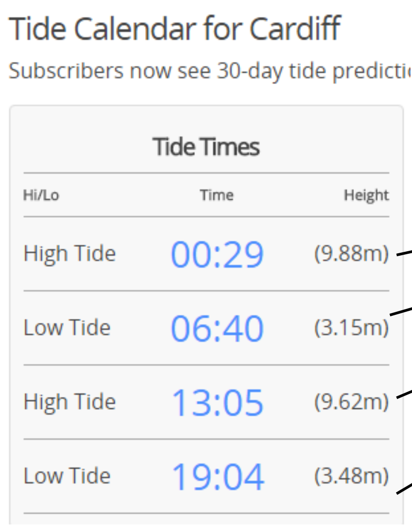
Last 3 results from the ‘all Tide Prediction Location’ page

tideScraper

tideScraper is a module that returns a weeks’ worth of tide records from tidetimes.org.uk given a Port data structure. Within the port data structure, the port’s name, hyperlink to the port’s page (which is used by tideScraper to access and retrieve results) and the latitude/longitude location of the port. For example if I provide the port with Cardiff as the name, the appropriate link to the Cardiff page and its latitude/longitude values I should get the same results. As shown below, I executed a method passing Cardiff’s port details in and am printing the results to the Console. From here it’ll be easy to compare them with the websites Data. If they match, then tideScraper performs its task correctly.

Testing Method (In the TideScraper class)

```
public static void main(String[] args) {
    Port cardiff = new Port("cardiff","/cardiff-tide-times",new LatLng(51.45,-
3.1667));
    tideScraper t = new tideScraper(cardiff);
    TideRecord[] tRecord = t.getTide();
    for (int i = 0;i < tRecord.length;i++){
        for (int j = 0;j<tRecord[i].getRecordCount();j++){
            System.out.println(tRecord[i].getTideType()[j] + " " +
tRecord[i].getTideTime()[j] + " " + tRecord[i].getTideHeight()[j]);
        }
    }
}
```



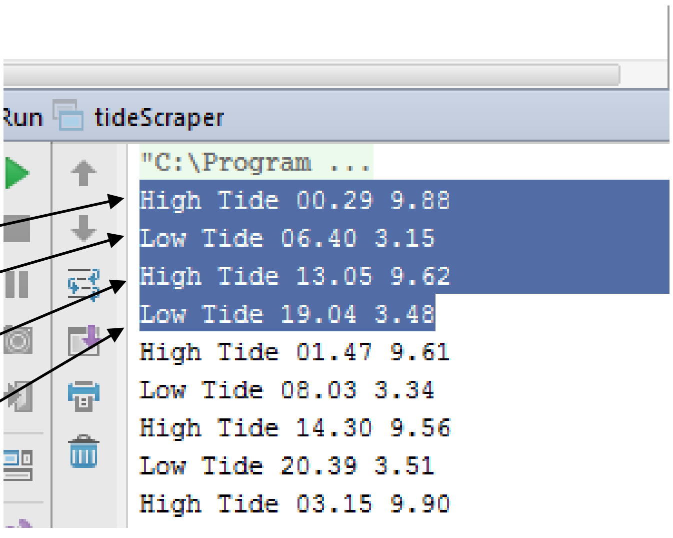
Tide Calendar for Cardiff

Subscribers now see 30-day tide predicti

Tide Times

Hi/Lo	Time	Height
High Tide	00:29	(9.88m)
Low Tide	06:40	(3.15m)
High Tide	13:05	(9.62m)
Low Tide	19:04	(3.48m)

Cardiff's tidal data for 30/04/2016



```
"C:\Program ...
High Tide 00.29 9.88
Low Tide 06.40 3.15
High Tide 13.05 9.62
Low Tide 19.04 3.48
High Tide 01.47 9.61
Low Tide 08.03 3.34
High Tide 14.30 9.56
Low Tide 20.39 3.51
High Tide 03.15 9.90
```

Console output of the Testing Mthod for 30/04/2016 – 06/05/2016

As you can see from the above diagrams, the retrieved results match that of the actual results we expected to receive. I also checked the data for every day between 30/04/2016 and 06/05/2016, all existing records on the website are taken correctly therefore I can say with absolute confidence that tideScraper executes as proposed.

TideCalculator

TideCalculator is the critical module for educated estimation of tides between two ports. In terms of testing, if I supply it 2 known ports I.E. Cardiff and Barry and supply the third port as a custom Port with its location set roughly halfway between them (at the international Sports Village Cardiff). For testing there are results we know for sure; the amount of results should total 26 due to the amount of tidal records per day each of those ports have that can be estimated. Given any tide time/tide height on a given day, the corresponding value in the custom port has to lie somewhere between the value of the time/height of Cardiff and Barry at the same record. I.E. Assuming the tide time was 18:00 at Barry with a height of 8M and the tide time was 18:30 at Cardiff with a height of 9M then the time, T would be 18:00 <=T <=18:30 and height, H would be 8M <= H <= 9M. The easiest way to compare this is check whether this formula is satisfied in an excel table (see below).

Table 1 – Testing Tide Calculator times

Tide Type	Barry Tide Times	Estimated Tide Time	Cardiff Tide Times	
High	01:37	01:46	01:47	TRUE
Low	07:46	08:01	08:03	TRUE
High	14:17	14:28	14:30	TRUE
Low	20:22	20:37	20:39	TRUE
High	02:58	03:13	03:15	TRUE
Low	09:12	09:29	09:32	TRUE
High	15:38	15:51	15:53	TRUE
Low	21:49	22:07	22:10	TRUE
High	04:14	04:27	04:29	TRUE
Low	10:26	10:45	10:48	TRUE
High	16:49	17:01	17:03	TRUE
Low	22:58	23:18	23:21	TRUE
High	05:20	05:31	05:33	TRUE

Low	11:31	11:51	11:54	TRUE
High	17:49	18:01	18:03	TRUE
Low	23:59	N/A	00:22	FALSE
High	06:18	06:29	06:30	TRUE
Low	12:30	12:50	12:53	TRUE
High	18:43	18:54	18:56	TRUE
Low	00:57	01:15	01:18	TRUE
High	07:11	07:22	07:23	TRUE
Low	13:25	13:43	13:46	TRUE
High	19:34	19:45	19:46	TRUE
Low	01:50	02:08	02:11	TRUE
High	08:02	08:12	08:13	TRUE
Low	14:15	14:33	14:36	TRUE
High	20:22	20:32	20:33	TRUE

Table 2 – Testing Tide Calculator Heights

Tide Type	Barry Tide Height	Estimated Tide Height	Cardiff Tide Height	
High	9.17	9.54	9.61	TRUE
Low	3.2	3.32	3.34	TRUE
High	9.14	9.49	9.56	TRUE
Low	3.39	3.49	3.51	TRUE
High	9.42	9.82	9.9	TRUE
Low	2.89	2.97	2.99	TRUE
High	9.57	9.99	10.07	TRUE
Low	2.88	2.91	2.92	TRUE
High	10.04	10.52	10.61	TRUE
Low	2.32	2.3	2.3	TRUE
High	10.29	10.8	10.89	TRUE
Low	2.21	2.13	2.12	TRUE
High	10.8	11.34	11.44	TRUE
Low	1.7	1.6	1.58	TRUE
High	11.07	11.62	11.72	TRUE
Low	1.58	N/A	1.41	FALSE
High	11.53	12.06	12.16	TRUE
Low	1.16	1.01	0.98	TRUE
High	11.76	12.3	12.4	TRUE
Low	1.06	0.88	0.85	TRUE
High	12.11	12.62	12.71	TRUE
Low	0.75	0.56	0.53	TRUE
High	12.27	12.78	12.88	TRUE
Low	0.7	0.51	0.47	TRUE
High	12.49	12.95	13.04	TRUE
Low	0.51	0.31	0.27	TRUE
High	12.54	13.03	13.12	TRUE

Before discussing the results further, the red highlighted rows above in each table shows a record that hasn't been estimated hence the N/A value. This is due to the way my algorithm compares and stores Tidal Records. Records are held in an array of 7 *tideRecord*s, each corresponding to a day. Despite having a statement which checks whether the records are within 40 minutes of each other (so that they match up), because they are technically on different days they aren't checked within the loops. This is a feature I could improve to yield extra results as this is a case that is likely to occur relatively frequently. This is why I only expected 26 results produced as a result of the *tideCalculator* despite Cardiff and Barry having 27 records in that period. This is something I could look to resolve in the future by allowing the lower bound of one day to be checked with the upper bound with the next day and vice versa as a final refinement in fixing this.

Besides the accounted for records, every single estimation meets the requirements for a correct estimation. All results fall within the bounds as expected and shown by the True values generated by the boolean testing method in excel. The number of records is also as expected.

Not only do the estimated results fall within the bounds, you can see with the height/times that they are always a lot closer to the Cardiff results. This is due to the distance/triangulation part of the algorithm and further validates the correctness of this algorithm. This is because the Custom Port we input is considerably closer to Cardiff.

Testing Method:

```
public static void main(String[] args) {
    Port upper = new Port("cardiff","/cardiff-tide-times",new LatLng(51.45,-
3.1667));
    Port lower = new Port("barry","/barry-tide-times",new LatLng(51.3833,-3.2667));
    Port custom = new Port("custom","",new LatLng(51.449211,-3.186057));
    int recordCount = 0;
    TideCalculator t = new TideCalculator(upper,lower,custom);
    TideRecord[] tRecord = t.calculateAverageTides();
    for (int i = 0;i < tRecord.length;i++){
            for (int j = 0;j<tRecord[i].getRecordCount();j++){
                System.out.println(tRecord[i].getTideType()[j] + " " +
tRecord[i].getTideTime()[j] + " " + tRecord[i].getTideHeight()[j]);
                recordCount ++;
            }
        }
    System.out.println(recordCount);
    }
```

Excel formula for verification to check whether estimated results are within the correct bounds:

=IF(AND(D4>=MIN(C4,E4),D4<=MAX(C4,E4)),TRUE,FALSE)

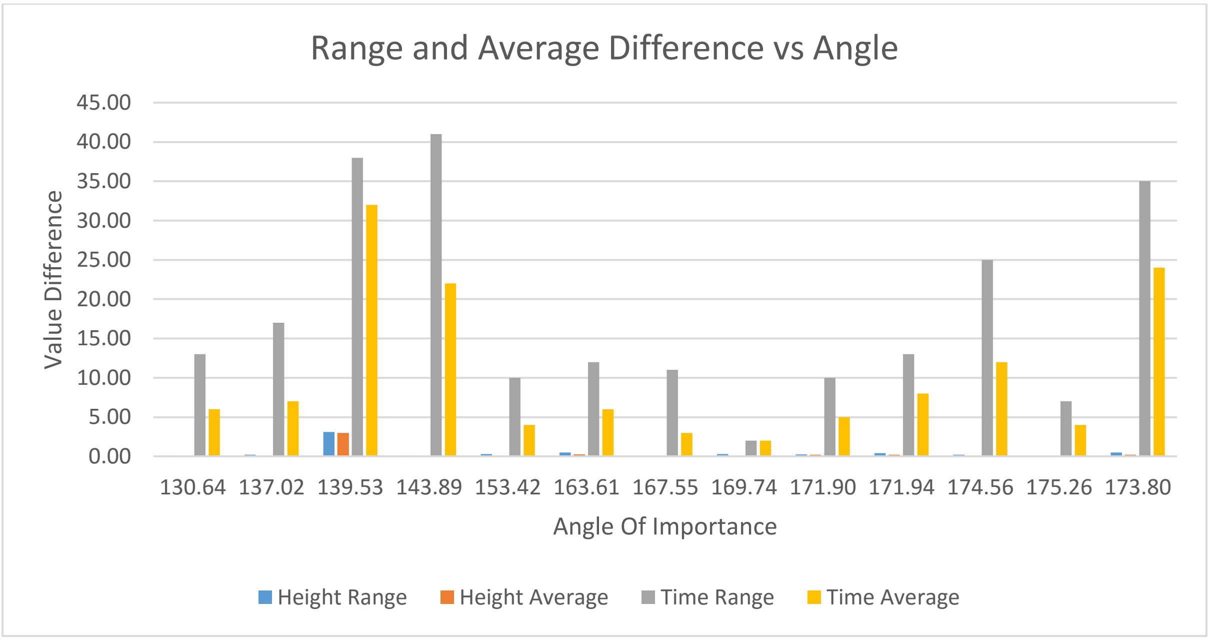
6.2 Tide Calculator Performance

It has been discussed and speculated within this project that the angle meets the other two ports is probably the most deterministic factor just behind absolute distance between the locations. This section's purpose is to produce results which compare 3 existing ports and aim to equate them with the estimated results. I.e. the third port passed into the *tideCalculator* module actually exisst. The records are taken from it and compared with the approximation created. The entire purpose of this section is to gauge the accuracy and proficiency of the algorithm. From this I'm looking to find any patterns or correlations which can demonstrate any strengths/weaknesses, from which I can build-on in future work.

For this test I am be manually inputting close-distance triplets of Ports and running *tideCalculatorAnalyzer*. That module keeps a record of the difference between the actual value and estimated value of every record. It also holds the value of the highest time/height difference (the range of each), the angle at which the fake Custom port meets the other two and the average deviation of height/time. Estimations are never going to be entirely accurate and considering there are too many unobtainable variables which are unfeasible to require and factor in the test is looking to quantify what degree of accuracy the estimation provides.

I'm going to store my results from this test in an excel table hoping to group values by the angle at the port being tested. Not only is this be more readable, it'll be easier to draw conclusions from partnered with the fact that I could potentially generate a graph giving a visual representation of the results to aid in the explanation of their meaning. If the estimations don't deviate too far from the actual values then the algorithm can be considered robust to a degree, the extent at which the results differentiate can be made clear to the user if using the estimator.

Graph 1 – Range and Average difference vs Angle



The graph above was produced from 13 tests of the *tideCalculatorAnalyzer*. It's interesting to note that contradictory to the original hypothesis, the Angle at which the custom port meets the other two ports doesn't seem to have much of an influence. I reasoned that as the angle approaches 180° it was more likely to lie on a consistent line and therefore have a better approximation of time/height. As you can see there appears to be no strong correlation of any kind when it comes to the average time and range of time. I think there's no strong correlation with the steeper angles due to the coastal shape playing such a large factor.

Something else to note is that the average Height and the height range is consistently considerably very close to the actual result. Besides the one value at 139.53 degrees which you could potentially consider an outlier, never exceeds half a metres difference from the genuine value. The actual average height disparity is only 0.35 metres and the average range at only 0.47 metres. This is quite a precise estimation and therefore in terms of predicting a tide's height I would consider the algorithm a strong success.

However as you can see with the Time Range and Average Time, the estimation is considerably further off. The average time difference prediction is about 10 minutes from the true result. The graph actually shows that the time can fluctuate heavily, seemingly with no dependence on the angle. With the tide times I noticed that when picking the ports, the relative distance between the 3 ports were a larger distance apart on ports with a bigger time disparity. After concluding that the time estimation is a slight weakness of the algorithm (albeit much of a weakness since on average the time is correct within 10 minutes), I think the more important and deterministic factor is the distance between the 3 ports which could be up for further investigation.

Since the point of this test was a verification of accuracy, the test itself has been a success. From the results we can produce a literal definition of the degree of accuracy. I can give the user ratification that the data is likely to be correct within those bounds, the height is likely to be within 0.35 metres of accuracy and the time is likely to be within 10 minutes correctly across all results generated from tide calculator. The test's other purpose was to try and draw a causal link between the angle of incident and the ports. This test does create call for a different, new test however since I managed to speculate that distance has more of an influence on the tide time than anything else. The table of results is included in Appendix A:

6.3 Evaluation / Conclusions

6.3.1 Project achievements

Overall as a project, I believe I've accomplished what I set out to achieve. First and foremost I believe my application is an improvement on the pre-existing tidal applications. Not only does it provide multiple ways of selection, which is a general problem in the leading applications, but also implements a unique functionality of being able to estimation a tide at any given point. My project met all essential requirements, but doesn't have some of the niche desirable features I wanted to implement had I had time due to ensuring there was a robust, complete system at the end.

The android application was set out to be informative, intuitive and offer the user complete freedom in selecting a Tidal Port. By offering the end-user multiple ways they can use their preferred methods to display the information. All aspects of the application function correctly with location based settings on, thus I'm pleased that no discernible errors exist throughout all the testing. As mentioned, having locational settings disabled causes one screen of the application to behave incorrectly. This is a weakness in the system that can be fairly easily dealt with, method of which I will discuss in the Future Work section. Besides this, the app achieves everything it set out to do from the specification.

The Performance of all my algorithms in modules such as *TideCalculator* and *TideCalculatorAnalyser* along with all the web-scraping modules, background-threads in the android activities and location retrieving are all done in a feasible time, often real-time. While performing tidal estimation or tidal scraping, which is often dependent upon internet speed, since the amount of data has been taken care to be reduced results returned are almost always immediate. As a precaution, any of these that might be more computationally intensive than the rest of the program a loading-screen with information is displayed to the user adhering to visibility of system status, importance of which is stressed in Nielsen's heuristics. Thorough testing on mobile with WiFi and Mobile Data has proved that the project demonstrates good efficiency in algorithmic performance and therefore can be considered a strength of the project.

Evaluating the android application is mostly a triumph from a technological viewpoint, besides the one minor glitch which I plan to fix, there is a second question to be answered: is it practical to develop a solution to estimating tidal data between two ports? To answer this I have to look at my solution. From the tests, I can say with some confidence that my solution in selecting the strongest ports seemed to be very effective at predicting height with very low error range/average error bound. However the slight downfall of the algorithm seemed to be that the accuracy of the time wasn't influenced by the Angle at which the port to be estimated met the other two. Considering I noticed that larger time disparities were created with Ports that were a larger distance from each other, it's not unreasonable to investigate that further. The modules, which were able to be tested in binary (either they worked or they didn't) all passed in achieving what I intended which is some consideration.

In light of all the unobtainable variables, backlogs of data or lack of concrete prediction algorithm I don't think it's practical to ever derive solution that provides an exact guess. As stated in my background research the only way to confidently predict tidal data is to have months upon months of data prior and derive lunar/ tidal data from this. This would require costly professional equipment, constant maintenance and was infeasible to even consider for a project of this magnitude. Hence the solution looked to be an approximation, never a substitution for actual data. I believe with some success the algorithm meets this requirement. So long as the user knows the room for potential error in both Time and Height (which aren't very far from the actual results) then this provides a means to that end. To relay this information to the user I could plot the error bounds on the graph with the estimated data or display a note informing them before they use the function.

6.3.2 Successfulness of design

During the lifetime of this project, between the Design and implementation some minor things were changed due to the addition of unforeseen difficulties or immediate improvement on the existing solution. Yet the project never deviated from the template. I.E. all classes still interact in the same way with each other and fulfil the purpose expected from them, but not necessarily in the way I proposed. Adhering to the design for the most part provided me with a clear view of how parts of the system interacted with each other. By specifying deliverables the success of the design could not only be based on correctness of code, but also had time goals I could strive to meet.

An example of the only major change I've made from the design phase, is the combination of two of my Android activities: *Map Selection* and *Tide Calculator*. Originally they were separate activities, both employing google maps API as the foundation. After some discussion with a peer I decided to combine them which I now believe to be a better option than the original design. Aberration from the design in my project was always intended to benefit the application. The reason I haven't changed from the template I put forward in the design section is because it works. Referring to the Data Flow Diagram in the Design section, if my module design was to change drastically I wouldn't have much idea of how data flowed throughout the system.

6.3.3 What worked well/less well

Going into this project with limited knowledge in both the geographical domain and no prior knowledge of Android programming, this project would not have succeeded but for the sequential style of the Waterfall system cycle. Given a project of this size and nature, with set deliverables and known requirements this system allowed me to get the most out of the project.

Due to the geographical element, lack of readily accessible resources on the topic and difficulty of the subject, my computational knowledge despite research could only get me so far on the estimation. This put a bit of a limitation on the project. In retrospect I could have sought advice or first-hand knowledge of tides from a Geographical, or more specifically, tidal expert to help give me insight to a deep problem.

Since all the methods are performed almost instantaneous, the time complexity of the algorithms were never a negative issue at any point. Considering the operations on data generally spanned all possible ports in the UK I would say this was a strong point in the project. Not once was there a need for further optimisation due to space/time complexities which is something very critical to consider when programming for a phone with limited space and processing capabilities.

Given more time and potentially more data from Ports outside the UK I'd like to carry out more tests. The triplets of ports I selected are all representative of ports likely to be chosen by the user/ the algorithms, perhaps creating some sort of test program to test every combination of triplets within a certain distance could be something to look to do. Essentially more rigorous testing partnered with more quantity of results to get a more comprehensive approximation. Furthermore, since the estimation was intended to be an improvement on just getting average results between 2 ports, it would work better.

The work plan I designed in the initial plan had specific, set deliverables for the end of each week and fortunately the goals were realistic and were achieved on time throughout every stage. I think this method worked well as it gave a gauge to whether I was on track or not. Having a time-pressure situation forces you not only to complete it, but given any spare time before the next stage, improve it.

Lastly I would also have liked a larger test-space given more time, a larger test space would hopefully reinforce my results and instil confidence in the range/average range of accuracy which can be relayed to the end-users.

6.3.4 What I learnt from the project

From this project I learnt that it is possible to create and provide a reasonably good projection of estimated tidal data. In light of not being able to ever achieve exact results without months' worth of backlogs of data for a specific point. Considering I managed to achieve a prediction of height within an average of 0.37M and an average time disparity of 10 minutes despite there allegedly being no existing geographical algorithm at all to extrapolate this data I learnt that it's possible to do so with some confidence.

I also gained a lot a lot of insight into Android Programming, starting to use some of the more advanced techniques and complicated routines to achieve desired effects. I feel like I expanded my knowledge of OOP as a whole. By taking Androids course and applying what I learnt in each lesson to my application straight away, I was able to grasp Android confidently in less than a week. Safe to say I still have much to learn of their advanced techniques, but I have used many of the components to create this application as well as experience on working on a good range of API libraries.

I've also learnt to produce something with a professional, quality finish to it. Unlike previous projects or coursework tasks in previous years I feel as though I've fashioned (with a couple of tweaks) an end product with competence is of a high standard. I also believe this is due to meticulous planning when it comes to the Specification/Design stage of the cycle. I didn't have to make any changes that altered the Data Flow of the project, at most a difference between the Design and Implementation would be the addition of methods to achieve the outcomes set by the Design. I discovered the importance of this in the duration of the Android 'Tides' application.

Due to not being able to implement 2 of the optional requirements which would have only slightly enhanced my Application, I have also come to agree that as in 'Hofstadter's Law': everything takes longer than you think.

7 Future Work

There are multiple sections of this project that could be built-upon with some effort in the future. First and foremost I would hugely recommend implementing a way that forces the user to have location settings switched on before accessing the 'Location Selection' screen. Currently, it behaves slightly unexpectedly with the locational settings disabled on Android which could is a fair weakness in the context of the project.

There are 2 key sections where future work could prove useful and yield a higher quality Application, or in the case of tidal estimation, higher quality research. Beginning with the Android app itself, there are several features I'd have liked to implement had I had time and how I might go about doing so:

- A favourites feature where a list of user ports could be efficiently stored and retrieved
    - Store the integer indexes of the ports user favourites in a text file for retrieval. Absolutely minimal on Memory Cost and maximum efficiency as the indexes are easily fetched from ArrayList given Index.
- Application could store results along with the date the result was retrieved in the devices memory
    - Program already obtains the date that the tidal records were retrieved, all that needs to be done is to write the contents of the record to the device memory which can be checked if existing data for that port at that given date exists. If so, no need to re-scrape results through web saving data and battery.
    - Results also need to be recycled after they are outdated so every day an automated internal Clean-up would need to be performed.
- Upgrade the User Interface to be more visually appealing
    - Currently priority is responsivity which has been achieved, application is also intended to have a formal tone, but might benefit from a good, consistent colour scheme/ custom UI to increase user satisfaction.
- Consider setting up a central server to scrape results and give formatted data to user
    - By moving the web-scraping/data formatting to a central server, the application would have to retrieve about 95% less data because they wouldn't have to retrieve the whole page, they could just send a request to the server and only be sent the key data post-format.
    - This method could mean quicker reclamation of results if implemented correctly and reducing data usage/battery power is always a key consideration on Android.

The second section which probably has a larger scope for investigation and would be more beneficial to the project would be looking at developing the method I've created for tide estimation. Currently it is based on relative distance, triangulation between ports and manages to determine what bounds the tide can fall in. Being able to boast a more precise tidal approximation would be hugely profitable in the context of the assignment. Hence I propose the following to consider for future work:

- Further research into correlation between absolute distance and time difference
    - While testing the estimation results and how they differe from actual results, the height estimation was highly accurate being very close to the real result. Time didn't seem to depend much on the angle at the port, but I did notice that the length between ports did therefore further investigation would be beneficial.
- Consider having a community-based project where people could collect and collate tidal data at a point, from which predictions could be derived
    - Think about implementing an experimental method where a user, locally only, could gather the data for months at a given point from which the actual Tide Estimation, from which the actual formula for predicting tides could be used.

    - Since the only concrete, confident way to predict the tides is with months of backlog data, this project could be a user-driven way of potential estimation at a certain spot using their own observed data.
- Exploration of a visual computation technique to detect the shape of coastline to be used in combination with other techniques could be explored
    - Google Maps also provides a way to detect altitude which could be used in combination of Latitude/longitude. By looking at visual computation techniques to detect patterns on the UK coast and detect coastal features such as inlets and coves, a study could produce some results which really aid in the prediction process.

Aside from the things that could be implemented or researched to consider for the future, there are things such as API Licenses or a license to be able to use the tidal data for commercial purposes to consider. Currently I'm using a free license for Google Maps in which I get an allocated amount of accesses. To allow unlimited access/commercial usage in the future a license would have to be purchased. Tidetimes.org.uk would also have to be approached about using their data in a marketable realm.

Last of all, it will be important to constantly consider that uktidetimes.org.uk may change their website, the way in which their data is stored or even add/discontinue ports. In the name of consistency and keeping true to Requirement 8: "The application should have a complete list of UK Tide Times concurrent with tidetimes.org.uk" continuous checking of all changes made to the website will need to be monitored. If changes are made, the web-scraping modules will need to be altered accordingly. Fortunately, if any ports are added there is already a module, *ListLinkScraper*, which if re-run by anyone in charge of application maintenance will re-populate the Port list with the current list as given on their website. This unit was designed to keep that in mind and reduce the complexity of the task at hand.

8 Further Reflections

From my study, I've been able to conclude that my initial method of tidal estimation is accurate enough to be practically feasible. I've also set a good base ground for further research into the matter and with access to more Data, or by using a different technique in conjunction with my foundation, there is real scope for an improvement on the solution. I have learnt to appreciate the quantity of work it takes to output a worthy project. Furthermore, I've learnt a great deal throughout the process I.e. OOP, geographical knowledge and the incredibly useful/powerful tools Android has to offer etc. I can hopefully continue to challenge myself in all these areas to improve the application.

I also believe I've created an application more desirable than alternative tidal applications by offering methods of tidal selection that users would find more intuitive. Where other applications limit the user's method to locational selection of tides which severely limits users looking for data not nearby them, this application excels at that. Referring back to my initial Plan, the aim of the project was to 'design, develop and create an Android application to allow user to attain tide predictions at different locations'. My end product does exactly this and more which I am happy about. The features which those applications contain and mine does not are all realistically achievable given a little further development in the future.

9 Bibliography

ACM, C. of the (1998) *10 Heuristics for user interface design: Article by Jakob Nielsen*. Available at: https://www.nngroup.com/articles/ten-usability-heuristics/

Stack overflow (2016) Available at: http://stackoverflow.com

Calculate distance in meters when you know longitude and latitude in java (2016) Available at: http://stackoverflow.com/questions/837872/calculate-distance-in-meters-when-you-know-longitude-and-latitude-in-java

(No Date) Available at: http://www.naldic.org.uk/assets/ukmap.gif

Determining whether A point is inside A complex Polygon (1998) Available at: http://alienryderflex.com/polygon/

Android design principles (no date) Available at: http://developer.android.com/design/get-started/principles.html#simplify-my-life

Methods - Heuristic evaluation (2002) Available at: http://www.usabilityfirst.com/usability-methods/heuristic-evaluation/

About tides – tides: Questions and answers (2016) Available at: http://www.ntslf.org/about-tides/tides-faq

Web-scraping with java (no date) Available at: http://www.geog.leeds.ac.uk/courses/other/programming/practicals/general/web/scraping-intro/2.html

2016, C.C. (2016) *EasyTide - on-line tidal predictions from the UKHO (Easytide)*. Available at: http://www.ukho.gov.uk/Easytide

Ocean motion: Background: Observing and predicting tides (no date) Available at: http://oceanmotion.org/html/background/tides-observing.htm

Tide predicting machines - NOAA tides & currents (2013) Available at: https://tidesandcurrents.noaa.gov/predmach.html

HMKCode (2015) Available at: http://hmkcode.com/android-check-enable-location-service/

10 Appendices

Appendix A. Results Table

Appendix B. Android Requirement Testing