**FACULTY OF SCIENCE AND**

**ENGINEERING SEMESTER 2, 2018**

**IAB330: Mobile App Development**

**Assignment 3: App Prototype**

**Due Date: Friday, 2ndNov 2018****, 11:59 pm**

**Assignment submission as a team through Blackboard**

**Weight: 50%**

You must sign below. By signing this form, you agree to the following:

We declare that all of the work submitted for this assignment is our own

original work except for material that is explicitly referenced and for which we have permission, or which is freely available (and also referenced)

The assignment shall be conducted in a team of 3-4 students, each team member must sign, as it is a formal agreement that represents that everyone is contributing to the whole assignment.

|  |  |  |
| --- | --- | --- |
|  | **Team Member Detail** |  |
| **Student Number** | **Student Name** | **Signature** |
| N9656286 | Dean McHugh | Dean mchugh |
| N9967273 | Donghyeon Kim | Donghyeon Kim |
| N9720316 | Patric Marchant | Patric Marchant |
| N9460314 | Jade Kelley-Reskic | Jade Kelley |

**Task 1: Final Prototype**

https://github.com/deanmchugh/IAB330\_assignment

**Task 2: Report**

## User Stories

As discussed in Assignment 2, the features for our Minimum Value Product (MVP) were not particularly ambitious, and we more so set out to polish core functionality and features described by the project stakeholders, this turned out to be positive, as in terms of the features we set out to accomplish, we finished almost all of them which related to the MVP. The core MVP functionality implemented is as follows: being able to record data such as points, lines, and polygons and store the data using the device’s storage (should work with no internet connection), then exporting the plotted data when an internet connection is present, via an email containing the geoJSON file. After plotting, the user is able to see the plotted data in the “collected data” section of the application, providing a tabular format of the data, which can be sorted via point, line, or polygon. The user is able to fill provided metadata input fields for each type of data, providing information about what the data actually describes through fields such as a title, and metadata fields such as a float, integer and string field. The user is also able to import plotted data from other users through loading a geoJSON file in the app, overwriting the users previously plotted points.

The MVP features we did not complete were both to do with the “collected data” page of the application where user’s would be given a tabular view of the plotted data, however, due to time constraints we couldn't fix the issues we were having with the page and showing each points information. We also planned on being able to tap on the individual data in this section of the app to view its information in more depth, however, this feature was implemented for clicking on the logged data on the map, which shows its metadata such as title, float, string, and int fields.

We also did end up implementing some features that were not on the list of MVP features, but were decided upon as they added functionality that was deemed helpful/nice to have by our team. The first feature that was added that wasn't actually necessary was showing the plotted data on the map page of the application, this was decided upon to implement, as it provides users with some “feedback” on the screen to show their data visually plotted on the map screen of the application, as in data plotting applications, without showing the data on the map, having a map in the first place is pretty much useless. Our team also did end up implementing functionality (called a watcher) to provide users with a way to test their GPS location accuracy, which was decided upon as it would be useful to users to make sure they are plotting accurate locations.

## User Interface

|  |  |
| --- | --- |
|  | Map Page  This page is the primary page of the application. The function of this page is to present the map with the collected locations displayed.  The plus and minus button at the bottom right of the map zooms the area displayed closer or further away respectively.  The ADD button at the top right is used to open a pop-up window with the options of collecting a point, line or polygon location.  The burger button at the top left is used to access the navigation page to move from one view to another gaining access to the other functionality of the application. |
|  | Side Drawer  This drawer provides the main navigation for the application. The drawer can be accessed from all other pages by pressing the burger button at the top left. The functionality if this drawer is to slide in from the left and occupy ¾ of the screen and present 4 buttons. The buttons use a stack layout to flow down the drawer.  This setup was done with a master detail page. The drawer is the master aspect and the pages are the detail.  The MAP button will navigate back to the main page and display the map with collected locations.  The COLLECTED DATA button will navigate to the list view of collected locations.  The SETTINGS button will navigate to the setting options for the applications (not implemented). |
|  | Collected Data Page  This page uses a tabbed navigation design to navigate through the different types of collected locations. This choice was used to have the all the locations shown or a specific type all available on the same page. The locations will be displayed in a list flowing down the page.  The hamburger button will bring up the side drawer for navigation to other pages.  The ALL tab will display a list of all points collected.  The POINT tab will only display the point type of locations collected.  The LINE tab will only display the line type of locations collected.  The POLYGON tab will only display the polygon type of locations collected. |
|  | Import and Export Data Page  This page uses a stack layout to display both buttons and labels to explain and initiate the functionality of the page. This page is used to import or export the locations data.  The GEOJSON button will generate a pop-up window with the locations data displayed to check the information before exporting.  The ENTER EMAIL line is used to enter an email for the location data to be sent to.  The SEND button will initiate the export functionality using the email entered in the line above.  The IMPORT button will take the sent location information and import it into the application to be displayed on the map and collected data page. |

## Intended Software Architecture

## 

## Above was the proposed architecture from report 1, which was intended to be implemented in our final prototype.

## 

## 

## 

## 

## However, during the development of our project, we have leaned towards the below UML diagram of

## As seen in the revised MVVM software architecture UML. Users will use the application for collecting geo-location data using GPS. In the Students (Users) Table, students will have their current device’s IMEI as the unique identifier for the ownership of the dataset, they will then enter data information such as, data name, metadata about the data and its automatically gathered location data (Latitude, Longitude and Altitude) using the “Watcher” button which refreshes the location data upon click.

## The model table represents collection of classes which explains and manages the business logic and the data models, which are known as the .CS files.

## The view table identifies and configures the UI display to the user, it will get the business logic (or the model) and interpret it into a user friendly interface, written in the .XAML file.

## The view-model table is responsible for handling methods, commands and other classes that affects the state of the “View” table. It will manipulate the “Model” and “View” separately using the “View-Model” in between.

## The main reason, that we have implemented our application in the MVVM software architecture is, firstly, it utilizes the “Observer Design Pattern”, where it clearly separates the logic between the UI and application process and it is most commonly used in situations where data binding is in use.

## As the “View-model” class manipulates the information processed and delivered through the “Model” class, then passing it through to the “View” for the users to see. As one of the MVP features was being able to display user collected data into a tabular list format (part of the required MVP feature). Using MVVM, the application would accept the data given by the user and process / store from the “Model” table and it would be given to the “View-Model” where it is manipulated and updated before passing through to the “View” table for the users to see, meaning both “View” AND the “Model” is affected by the user input.

## 

## Testing and Quality Assurance Strategy

The UI was manually tested. This was done by listing the required functionality of each button. Each button was clicked multiple times and the resulting action documented. Each time a button did the correct action the button was marked as operational. If the button did not perform the correct action the code connected to the event listener was analyzed. Each member of the team participated in this testing to ensure nothing was missed. Each member focused primarily on the functionality that they implemented to ensure absolute understanding of what the function was meant to do. The following table illustrates the compiled manual testing completed by each member which records the features being tested and the actions required to produce a result.

**Manual Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test of Specific Functionality** | **Test Setup and actions** | **Expected Result** | **Result** |
| Map is displayed and functional | Run the app, and observe and ensure the map responds to user interaction | Map is displayed when app is emulated and can be manipulated by user (ie. Zoom in/out and scroll to different locations) | Pass |
| Display point, line and polygon on map | Use test values that is to be displayed on the map. Test values used were in the vicinity of QUT campus. | Test point, line and polygon locations (longitude and latitude) appears on the main map | Pass |
| Side draw menu | When emulating the app, interact with the side draw menu | Side draw menu expands to show buttons which takes the user to the corresponding pages. Side draw menu button should be accessible from all pages | Pass |
| Add point location data in geoJSON format | From the main map, click the add button in the top right and choose point from the pop up. Fill out the data form and capture the coordinates. A test page was added to display the collected data to ensure it’s in the required format | Point data is added to the FeatureCollection in the correct format. | Pass |
| Add line location data in geoJSON format | From the main map, click the add button in the top right and choose point from the pop up. Fill out the data form and capture the coordinates. A test page was added to display the collected data to ensure it’s in the required format | Line data is added to the FeatureCollection in the correct format. | Pass |
| Add polygon location data in geoJSON format | From the main map, click the add button in the top right and choose point from the pop up. Fill out the data form and capture the coordinates. A test page was added to display the collected data to ensure it’s in the required format | Polygon data is added to the FeatureCollection in the correct format. | Pass |
| Export data via email | Collect location data and navigate to the Import/Export page. Input the desired email to be exported to and press the send button. | An email from the Geo App is received with an attachment of the location data collected. | Pass |
| Import data to the app | Download a geoJson file to the emulator. Navigate to the Import/Export page and press the import button. This will take the user to the emulator file system where the file to be imported is selected. | The imported geoJSON file is added to the collected data on the app | Partially Implement. Data is replaced by imported data rather than appending it. |

## Reflection on Learning

The semester was broken down into easily digestible pieces for ease of learning. As a team we focused of a separate area of the application to ensure in depth understanding of the functionality required to complete the MVP. We believe this lead to a smoother operating application as the different aspects or slotted together closer to the end to create a complete application.

One of the issues our team faced when beginning the project was git (moreso the merge command), as none of us had used git extensively, and thus had trouble using the merge command and merging multiple students code. After some time, we learnt how to better use the command, as well as work with git and its entirety.