

IAB330 - Mobile App Design

Assignment 3: App Prototype

Due Date: Friday, 2nd Nov 2018, 11:59 pm

Submission Method: Blackboard

Submission Cover Sheet Declaration

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).
- We agree that QUT may archive this assignment for an indefinite period of time, and use it in the future for educational purposes including, but not limited to: as an example of previous work; as the basis for assignments, lectures or tutorials; for comparison when scanning for plagiarism, etc.
- We agree to indemnify QUT and hold it blameless if copyright infringements are found in this work and the copyright owner takes action against QUT that is not covered by the normal terms of Educational Use.

Chosen Project Name: GeoAware		
Team Member Details		
Student Number	Student Name	Signature
n9751696	George Delosa	G.D.
n9446826	Ari Luangamath	A.L.
n9966471	Reilly MacKenzie-Cree	R.M-C.
n9879757	Faiyaz Samiul Haque	F.S.H.

Marking Guide:

Github Repository:

<https://github.com/Ariit0/mobile-geo-application>

Visual Studio

- Community 2017 Version 15.7.5

Microsoft .NET Framework

- Version 4.7.03056

Nuget Packages:

- GeoJSON.Net - 1.1.70
- NETStandard.Library - 2.0.3
- Newtonsoft.Json - 12.0.1-beta1
- NUnit - 3.11.0
- NUnitTestAdapter - 2.1.1
- PCLStorage - 1.0.2
- Plugin.CurrentActivity - 2.1.0.4
- Plugin.Permissions - 4.0.1-beta
- Plugin.Share - 7.1.1
- Rg.Plugins.Popup - 1.1.4.168
- Xamarin.Android.Support.Design - 28.0.0-preview5
- Xamarin.Android.Support.v4 - 28.0.0-preview5
- Xamarin.Android.Support.v7.AppCompat - 28.0.0-preview5
- Xamarin.Android.Support.v7.CardView - 28.0.0-preview5
- Xamarin.Android.Support.v7.MediaRouter - 28.0.0-preview5
- Xamarin.Essentials - 0.1.1.0-preview
- Xamarin.Forms - 3.4.0.925479-pre1
- Xamarin.Forms.Maps - 3.4.0.925479-pre1
- Xamarin.Plugin.FilePicker - 2.0.121
- Xamarin.UITest - 2.2.6.1906-dev

NOTE: When building/deploying select the Release configuration. (This is mainly for the UI testing to work.)

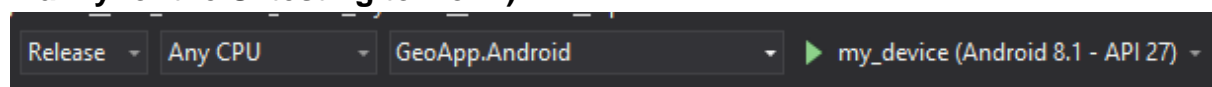


Table of Contents

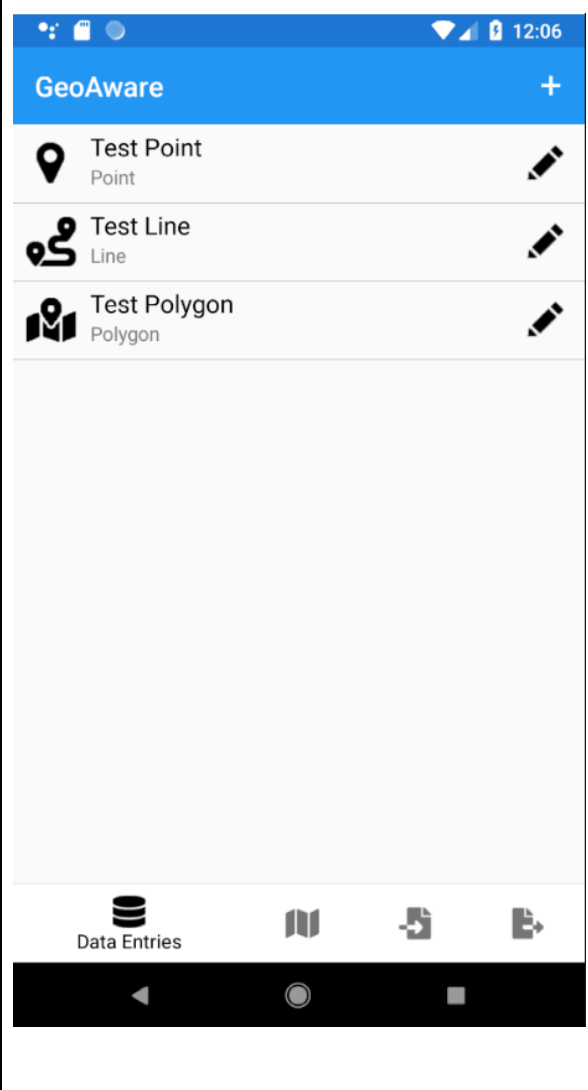
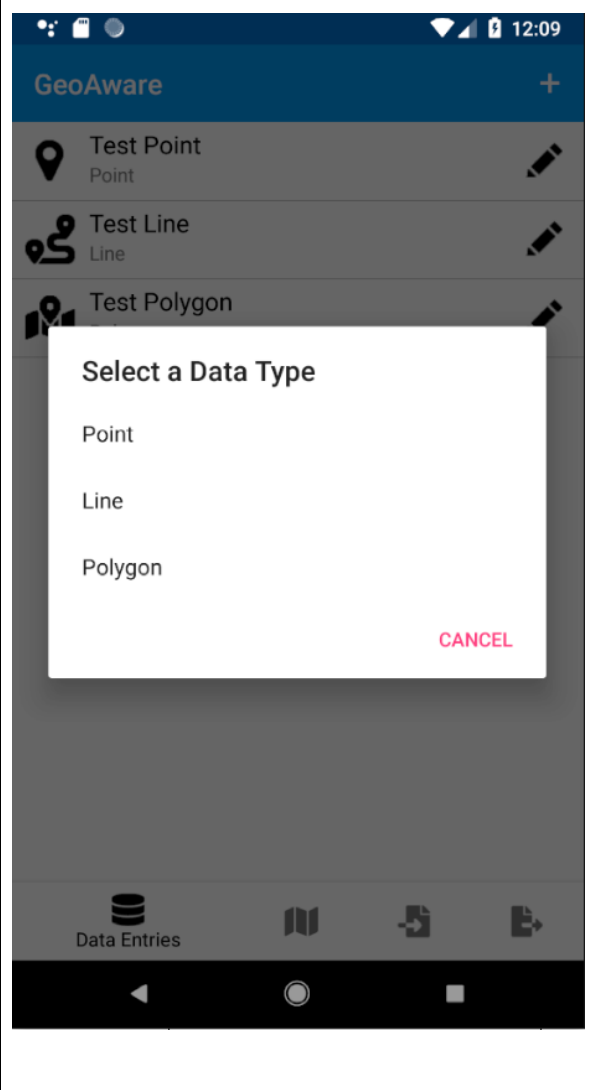
Contents

1	User Stories	1
2	User Interface	1
3	Software Architecture.....	5
4	Testing and Quality Assurance Strategy	7
5	Reflection on Learning	8

1 User Stories

User Story	Acceptance Criteria	MVP Version?	T-Shirt Size estimation	Priority	Implementation Status
Submit Form for Data Collection	The user is able to fill out a form with several different fields to describe their data.	Yes	M	High	Complete
Store Data Offline	Data is recorded and saved locally. Data can still be accessed/modified without an internet connection.	Yes	S	High	Complete
Record Appropriate Metadata	Upon recording a data point, the required metadata is automatically added. This metadata includes device or user identifier, time and user notes.	Yes	S	Medium	Complete
Share Data	Locally saved data can be exported to a formatted GeoJSON file. This file can then be shared by email/dropbox link. Duplicate location data is merged into one (may require prompt). Functions to edit the merged data and fix inaccuracies.	Yes	L	Medium	Complete
Record Different Types of Positional Data	The user can choose either point, line or polygon when recording positional information. The data types are made up of data points representing longitude, latitude and altitude as XYZ coordinates using GPS	Yes	L	High	Complete
Map Display	The app presents a live map page. The user's position and data entries are marked on the map display.	No	M	Medium	Partially Complete

2 User Interface

	
<p>The first tab functions as both the landing screen/home screen for the app and as the list of all data points a user has recorded. From here, the header presents the user with a '+' (add) button that allows the creation of a new point. Each data entry is also a button, and tapping on one will bring up its current information. Alternatively, tapping on the edit button to the right will bring up the editing page for the appropriate item.</p>	<p>When tapping the '+' button at the top right of the page, this popup appears and prompts the user to select the type of data being added. Tapping an option will take the user to the corresponding 'add data' page.</p>

New Point

Name
Date

Name of item
11/2/2018

Geolocation Data

Latitude
Longitude
Altitude

0
0
0

USE CURRENT LOCATION

Metadata

ADD FIELD

View Point

Name
Date

Test Point
11/2/2018

Geolocation Data

Latitude
Longitude
Altitude

37.421998...
-122.084
42


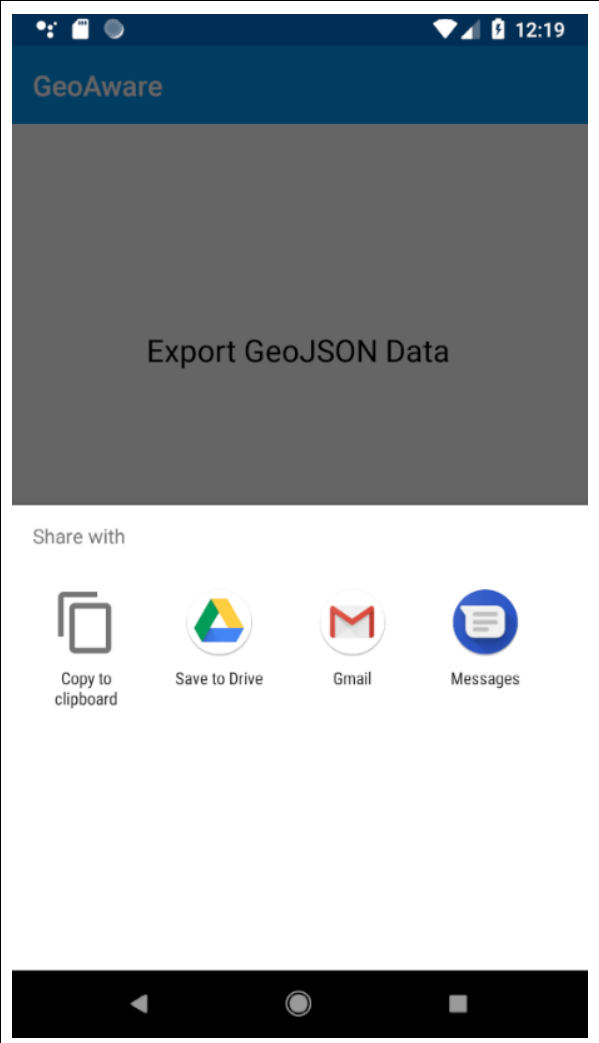
Metadata

Description
This is my test point!

This page contains all the forms needed to add a data point. It is similar for point, line and polygon, with the only difference being that a different number of geolocational coordinates are required for a valid type of entry (valid in terms of being representable in geoJSON).

The view data page shows all information stored about the selected data point. It also allows the point to be deleted by tapping the bin icon at the top right of the page.

	
<p>The second tab displays a native map view to the user. This map can be used to visualize the location of the user, and the locations of their saved data points. The user is able to drag, zoom and rotate the map freely, as well as tap on any data pin to view its name and coordinates. This screenshot shows the iOS version of the maps, as the Android version is currently not functional.</p>	<p>The third tab presents the user with a simple import page. With only a label and a button, as it stand the page is rather blank. However, the functionality required is there. Hitting the import button will allow a user to import their own txt file containing GeoJSON for it to be added to their data list.</p>

 <p>GeoAware</p> <p>Export GeoJSON Data</p> <p>EXPORT</p> <p>Export</p>	 <p>GeoAware</p> <p>Export GeoJSON Data</p> <p>Share with</p> <p>Copy to clipboard</p> <p>Save to Drive</p> <p>Gmail</p> <p>Messages</p>
<p>Tab four displays a very similar page to tab three, however, this time its function is to allow the user to export their saved data. Upon pressing the 'Export' button, they are presented with their devices native share sheet.</p>	<p>This share sheet allows the user to export/send their geojson to any app of their choosing, including saving it directly to the device. Tapping any option will result in their GeoJSON data being fed through to any system process or third-party app selected.</p>

3 Software Architecture

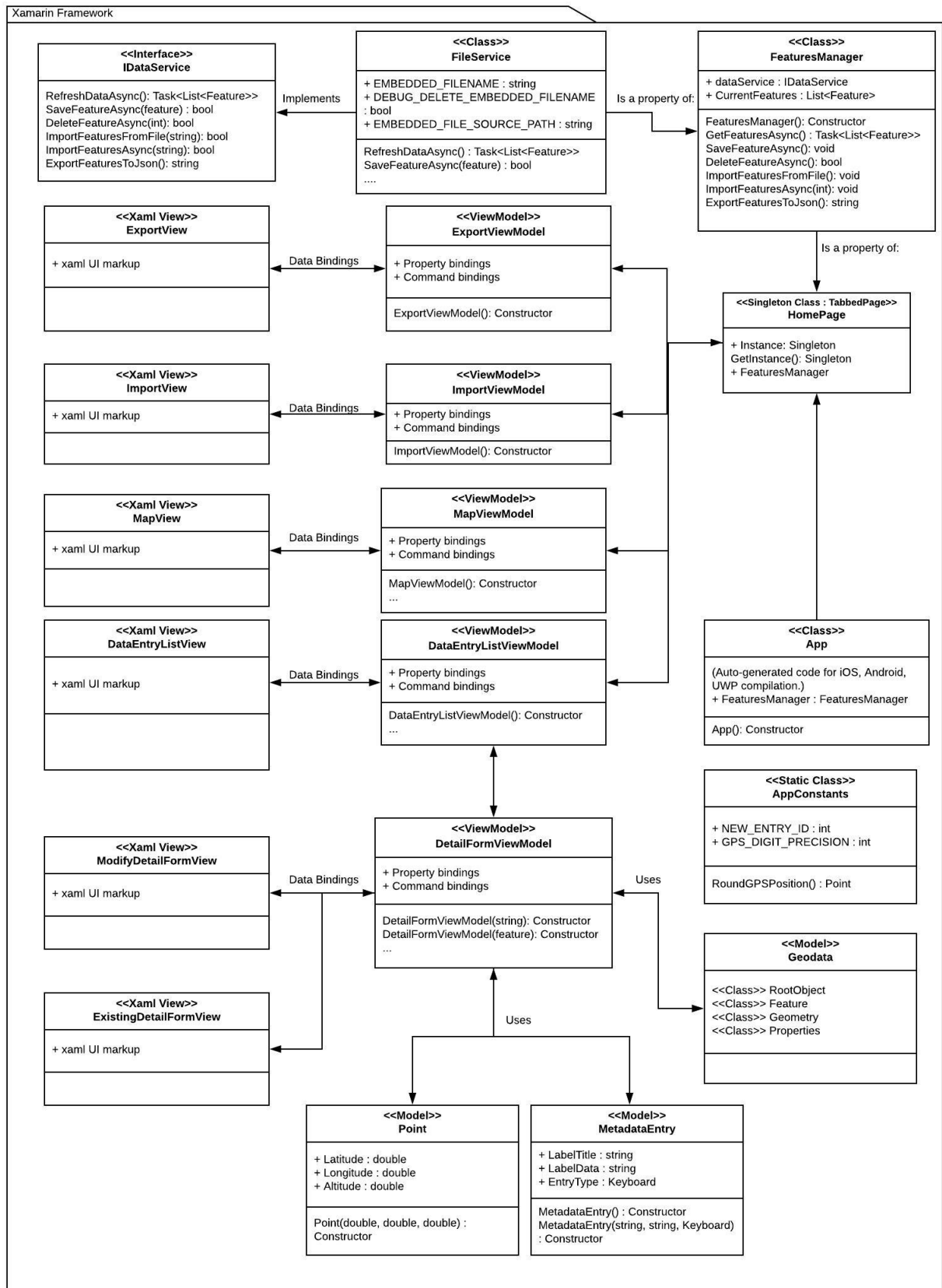


Figure - Final System Architecture UML Diagram

Our final system architecture closely resembles the proposed architecture from the previous report. This was because we were able to complete most of the functionality of the application successfully, as well as continuing to adhere to the MVVM pattern that was apparent in the proposed architecture. When compared to the proposed UML diagram, there is a clear increase in complexity as many components of the system were missed. These components include the file service system shown at the top of the diagram, as well as the model classes at the bottom. The google maps interface is now missing from the final version, as we had found that the native Xamarin maps served as a simpler and more elegant solution for our needs.

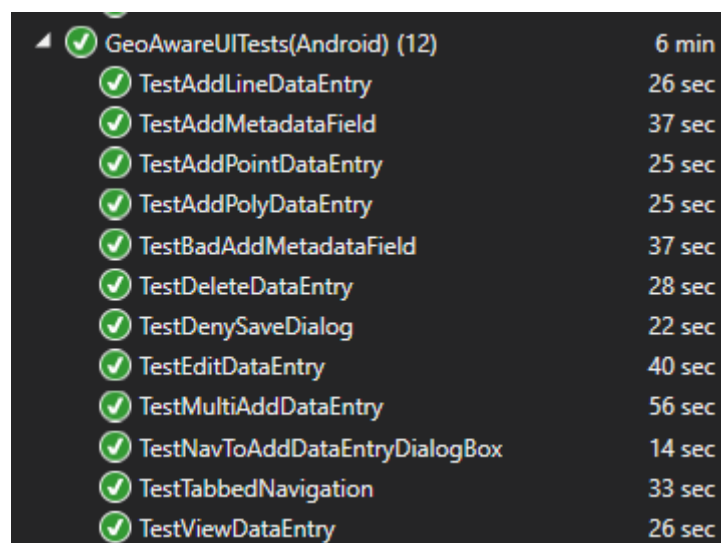
As a whole, the diagram describes the app's functionality and components in greater detail than the proposed version, due to our findings and practical development that led to a clearer view of the architecture that we were working with.

4 Testing and Quality Assurance Strategy

Our project uses the UITests that Xamarin.Forms provide. These tests allowed us a simple way to automate typical user interaction with the app's pages in order to create UI-based tests, with the help of the REPL system. REPL allows interaction with the application which helps generate UI test commands to be run. These tests comprise the black-box tests that we originally outlined in the previous report. For white-box tests, they cover the functionality of each component in the viewmodels of our app and ensure that they produce expected results.

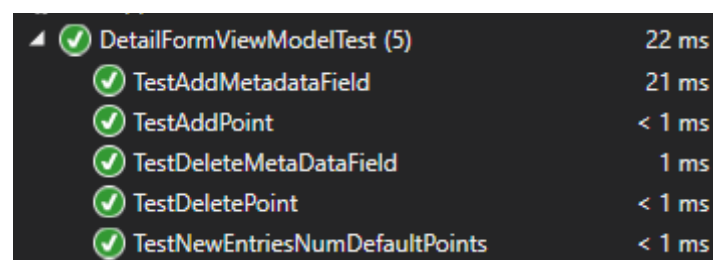
In particular, for UI testing, simulating user interaction was done by obtaining the position of the xaml element and using those coordinates as a reference for the automation to tap on. This ensures that the automation will work for any device/emulator UI scaling.

The following tests were covered for UI testing:



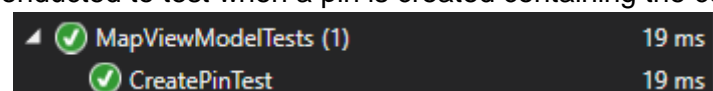
GeoAwareUITests(Android) (12)	6 min
TestAddLineDataEntry	26 sec
TestAddMetadataField	37 sec
TestAddPointDataEntry	25 sec
TestAddPolyDataEntry	25 sec
TestBadAddMetadataField	37 sec
TestDeleteDataEntry	28 sec
TestDenySaveDialog	22 sec
TestEditDataEntry	40 sec
TestMultiAddDataEntry	56 sec
TestNavToAddDataEntryDialogBox	14 sec
TestTabbedNavigation	33 sec
TestViewDataEntry	26 sec

Unit testing was also done on the DetailFormViewModel, specifically testing for the view model logic encompassing Metadata entries and Geolocation points. All tests conducted for the DetailFormViewModel have succeeded:



DetailFormViewModelTest (5)	22 ms
TestAddMetadataField	21 ms
TestAddPoint	< 1 ms
TestDeleteMetaDataField	1 ms
TestDeletePoint	< 1 ms
TestNewEntriesNumDefaultPoints	< 1 ms

Map testing was conducted to test when a pin is created containing the correct information.



MapViewModelTests (1)	19 ms
CreatePinTest	19 ms

5 Reflection on Learning

Meeting MVVM Standards

Our team ran into problems when adhering to MVVM as the design pattern we originally decided upon before development. Sometimes it was the case where a difficult problem was solved easier by implementing MVC, only to be tediously converted into MVVM afterwards. While it allowed us to get features into a working state quicker, it was less than ideal to fix since it usually required refactoring more code than was anticipated. Going forward, it would be wise to spend time into researching MVVM frameworks with Xamarin such as MVVMCross or MVVMLight, as it may have alleviated the workload spent on adhering/converting code to MVVM.

Design / Appearance

While the initial UI for the application was drafted around Android, during development it became apparent that it would be better to use a UI that caters more to the familiarities of the majority of its users on the iOS platform. This decision to switch UI design patterns mid-development meant switching from a MasterDetailView to a TabbedPageView, a process which delayed production as it meant changing the entire base view of the app. This is obviously not ideal, so in the future, UI should be more carefully considered when prototyping. UI should have also been more of a focus in our group discussions as it can be very subjective, and agreeing upon a design sooner would have benefited us earlier on in development.

Time Management

The MVP for this project was met in time for presentation, which as a bonus gave the team a substantial amount of time in the following week to simply work on bug fixing, code quality and testing. This process did not go without its issues however, as development presented problem after problem. These problems could often lead to an extended amount of development time to find solutions. An example of this was the difficulty reading in and writing JSON files in the correct GeoJSON format. This task ended up taking far more time than expected and set the entire team back in terms of development schedule as it was a vital task for most app functions. This issue, like many others was overcome, with extra work being put in for the following days, allowing development pace to be back on track.

Although completion of the MVP was believed to be on track before the presentation date, the discovery of a few bugs the night before and morning of the presentation lead to some panic and a rush to quickly work out the issues. With better time management that left more room for thorough testing, these issues would have likely been discovered and fixed well in advance of the presentation.

Distribution of Work

While the distribution of work on the project did not adhere to the original assigned workloads, it was in part due to the reliance of some features on others, and so it made more practical sense for some team members to work on features other than what they were assigned. The original assigned workloads also did not take into account the fact that only one team member had access to a Mac computer, and as such was the only person who

could work on iOS. This caused a complete shift in workloads, with the Mac user focusing on the iOS specific design/import features and the others focusing more on feature development.

Achieving Non-MVP Tasks

No Non-MVP tasks were able to be fully implemented before the development deadline. This was due to the refactoring, polishing and testing taking up a majority of time near the end of the development cycle, leaving little room for starting new features. Tasks such as adding an app sign-in and user verification system and form creation templates were simply impossible to start work on this late in development.

A component of the map display task was implemented in time, however it is not complete. The map only works on iOS, as the Google Maps SDK required the use of an account with billing set up. In addition while the code is there for the maps to render custom shapes, not enough time remained to fix the issues encountered when getting the stored data to automatically render. As it stands, the only feature available of this task is the rendering of pins on the iOS map for all data types.

Semester Reflection

As a team, working with Xamarin in this semester has given us first-hand experience on using a shared code library for developing mobile applications, and the benefits it provides when compared against writing individual codebases for each platform. The project we have created has proven to be very rewarding for our understanding of mobile app development.