QDI MOBILE FINAL REPORT

May 26, 2019

Drew Wilken
Leo Molitor
Page Bennett
Wenxuan Li

Contents

1 QDI Mobile Product

The QDI Mobile App is an iOS application that can be used by Minnesota Geological Survey geologists to collect sample data. This data then gets exported so it can be imported in to Minnesota Geological Survey's QDI database. The application is designed to be user friendly in a way that makes it easy for the geologists to collect data from the field and export it from their Apple product. Once the geologist is done with a session of collecting data in the field and are ready to enter their data in to QDI, they select all of the datapoints they would like to enter and export them to a file. This is a .zip file that contains the image data for each of the samples, and a JSON file with the rest of the data for each of the samples. Once this file is on a computer that has ArcMap on it, the geologist clicks the Import button on the desktop version of QDI, and the sample data that was collected by them in the app is then entered into QDI. The app itself has the ability to samples, edit sample data, and delete samples from the app.

In our separate document that includes our reference documentation, we have made sure to include any references, diagrams, or documents that will be helpful to MGS employees who are trying to understand how the app works or how to use it. These resources include, but are not limited to, a public repository hosting our API, an Xcode story board describing the different view controllers and how how they're connected, and links to our style guide and framework documentation.

2 Functional Requirements

2.1 Use Cases

Functional Requirements

Use Cases

Critical Features

Name: Collecting and exporting Mobile QDI App sample data

Identifier: Version 1

Description: This scenario will simulate a user opening the app, collecting NEW data, and exporting this data.

Actor: The actor of this scenario is a geologist in the field familiar with QDI.

Preconditions: Before this scenario the geologist has an iPhone with the QDI Mobile app installed

Postconditions: After this scenario, the data that the geologist collected in the app will have been exported, and therefore ready to be imported in the desktop version of QDI.

Scenario:

1. The geologist opens the Mobile QDI App, and types in their name.
2. The geologist adds new sample data by selecting the "+" button in the upper right corner and entering information in to all the required data fields.
3. The geologist clicks the save button to save the data they just collected as a datapoint.**[Alt Course A]**
4. The geologist now goes to the "Data" page by selecting it from the tool bar of the map page to export the data that has been collected.
5. The geologist selects what datapoints they want to export by tapping them, taps "Export Selected" and selects how they want to export the data (e.g. email).
6. This exports the data into a .zip file with the sample data images and the rest of the sample data in JSON form.

Alt Course A: Add more datapoints

A.4 The geologist repeats steps 2 and 3 with new data each time, until the geologist does not need to record any more data.

A.5 The use case continues from step 4.

Name: Editing datapoints that have already been entered in to the app and exporting the sample data

Identifier: Version 2

Description: This scenario will simulate a user opening the app, editing data from datapoints that have already been collected in the app, and exporting this data.

Actor: The actor of this scenario is a geologist in the field familiar with QDI.

Preconditions: Before this scenario the geologist has an iPhone with the QDI Mobile app installed. The app is already open and signed in to by the geologist, and the they have data from multiple data points entered. They are starting on the page that shows the map.

Postconditions: After this scenario, the data that the geologist edited in the app will have been exported, and therefore ready to be imported in the desktop version of QDI.

Scenario:

1. The geologist chooses to edit data by selecting the "Data" page at the bottom of the screen. This displays all of the previous datapoints collected. **[Alt Course A]**
2. The geologist selects the datapoint that they would like to edit the data of by clicking the ⓘ icon on that sample.
3. The geologist edits the data for this sample.

4. The geologist clicks the save button, saving the changes just made. **[Alt Course B]**
5. The geologist now navigates back to the data view page to export the data that has been collected.
6. The geologist selects what datapoints they want to export by tapping them, taps "Export Selected" and selects how they want to export the data (e.g. email).
7. This exports the data into a .zip file with the sample data images and the rest of the sample data in JSON form.

Alt Course A: Select datapoint by selecting pin on map

A.1 Instead of going to the Edit Data page, the geologist selects a pin on the map. Each pin is a datapoint whose GPS coordinate is represented by the location of the pin on the map.

A.2 The pin then shows a preview of the data in this datapoint.

A.3 The geologist confirms this is the datapoint that they want to edit and selects it.

A.4 The use case continues from step 3.

Alt Course B: Edit more datapoints

B.5 The geologist repeats steps 1-4 until the geologist does not need to edit any more data.

B.6 The use case continues from step 5.

Name: Deleting samples that have already been entered in to the app.

Identifier: Version 3

Description: This scenario will simulate a user deleting sample data from the QDI Mobile App.

Actor: The actor of this scenario is a geologist in the field familiar with QDI.

Preconditions: Before this scenario the geologist has an iPhone with the QDI Mobile app installed. The app is already open and signed in to by the geologist, and the they have data from multiple data points entered. They are starting on the page that shows the map.

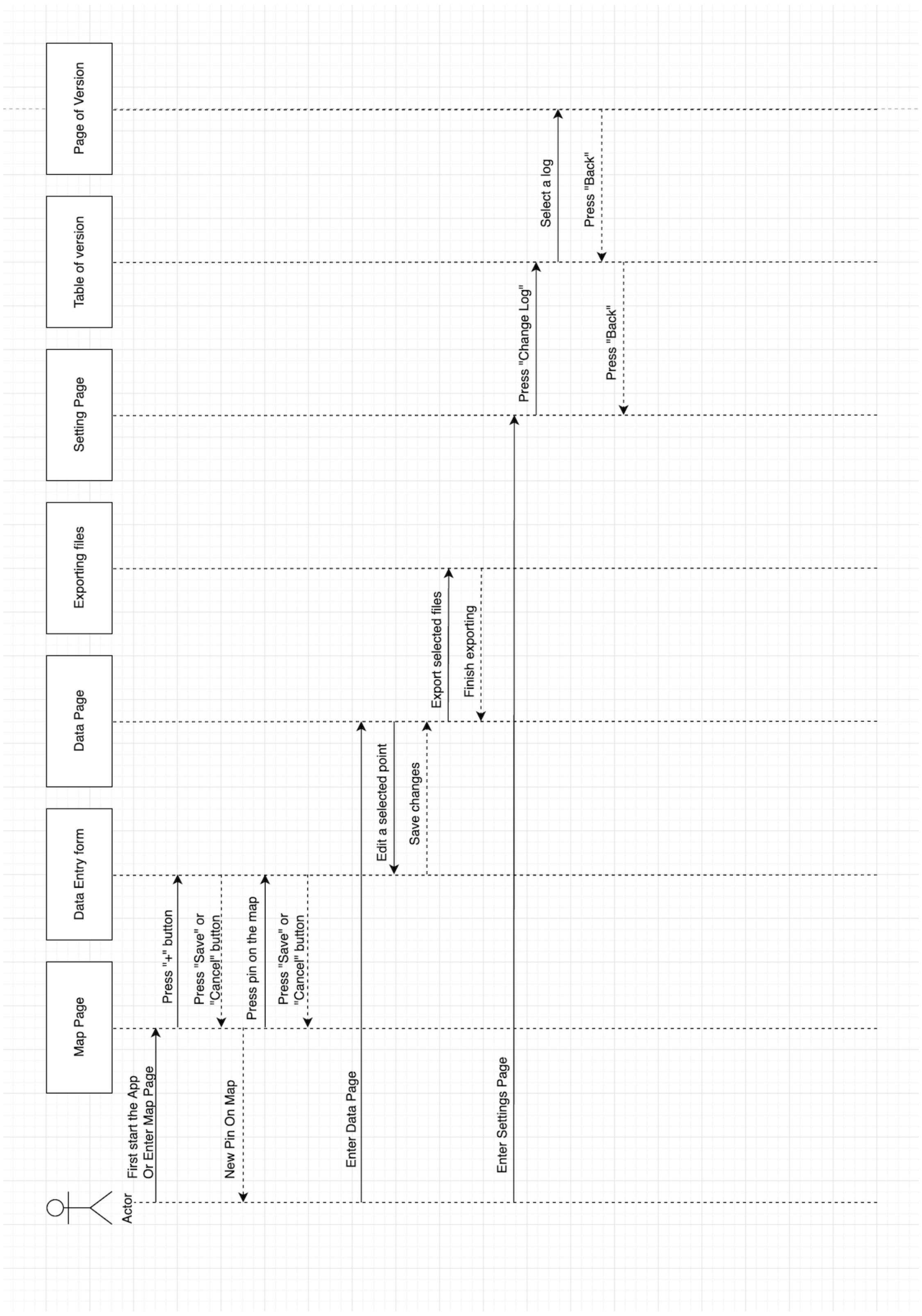
Postconditions: After this scenario, the geologist will have deleted 1 or more previously selected samples from the app. **Scenario:**

1. The geologist chooses to view the sample data that has already been recorded by selecting the "Data" page at the bottom of the screen. This displays all of the previous datapoints collected.
2. The geologist selects "Edit" in the top right corner of the page.
3. The geologist deletes the samples they want to get rid of by clicking the red icon on the desired sample and then selecting "Delete".

2.2 System Sequence Diagram

*Diagram on next page

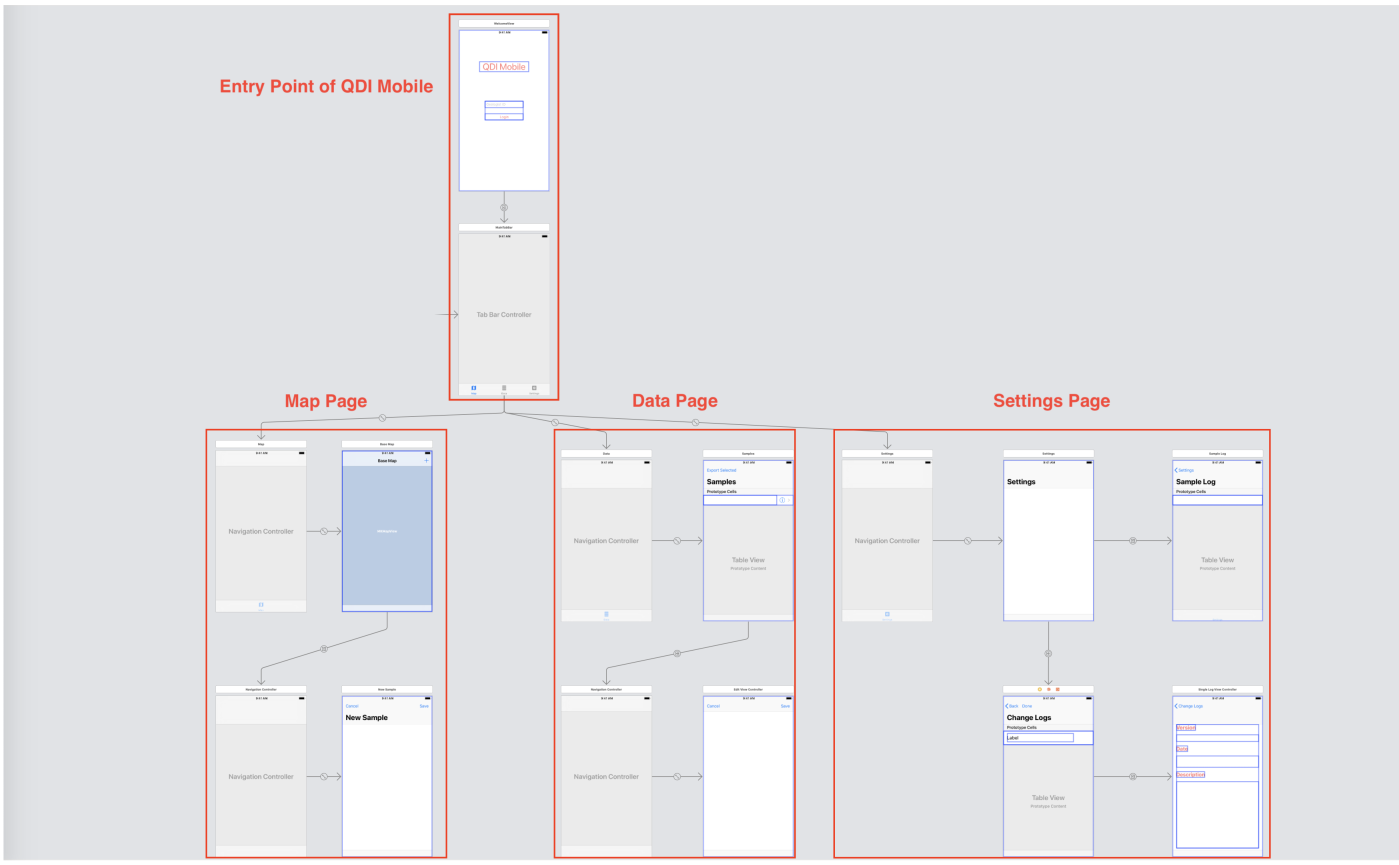
Description This System sequence diagram claims the navigation between pages in our application. There are 3 main pages in the application, users could accept each of them by pressing the corresponding buttons on the navigation bar which locates at the bottom of the application. The diagram also show how users' behaviors could manipulate data-points in the local storage.


Actor
Map Page
Data Entry form
Data Page
Exporting files
Setting Page
Table of version
Page of Version
First start the App
Or Enter Map Page
Press "+" button
Press "Save" or
"Cancel" button
New Pin On Map
Press pin on the map
Press "Save" or
"Cancel" button
Enter Data Page
Edit a selected point
Save changes
Export selected files
Finish exporting
Enter Settings Page
Press "Change Log"
Select a log
Press "Back"
Press "Back"

3 Nonfunctional Requirements

- Accessibility
  - We wanted the app to be accessible to MGS geologists that would use it. To do this, we made the app available through TestFlight so that the users can install versions of the app that have not yet been published to the app store. This will also help in the future if further development is done on these apps, enabling MGS to test new added features in the app.
- Simple layout
  - We also wanted the app to be as simple as possible. We designed the layout of the app so that anyone would figure it out in a minimal amount a time. We knew simplicity would be key in our design because many of the potential users of the app would be used to doing this work on pencil paper. We wanted to make the transition as smooth as possible by not adding any features that were too complicated.
- Datapoint pin selection
  - A feature we implemented was the ability to select a pin on the map, and bring up the sample for that location. The pins populate on the map when a sample is added at that location. When a pin is selected on the map, it shows a preview of the sample at the location. If this preview is selected, it brings up the Edit Data page for that sample so that changes can be made to the data if need be.

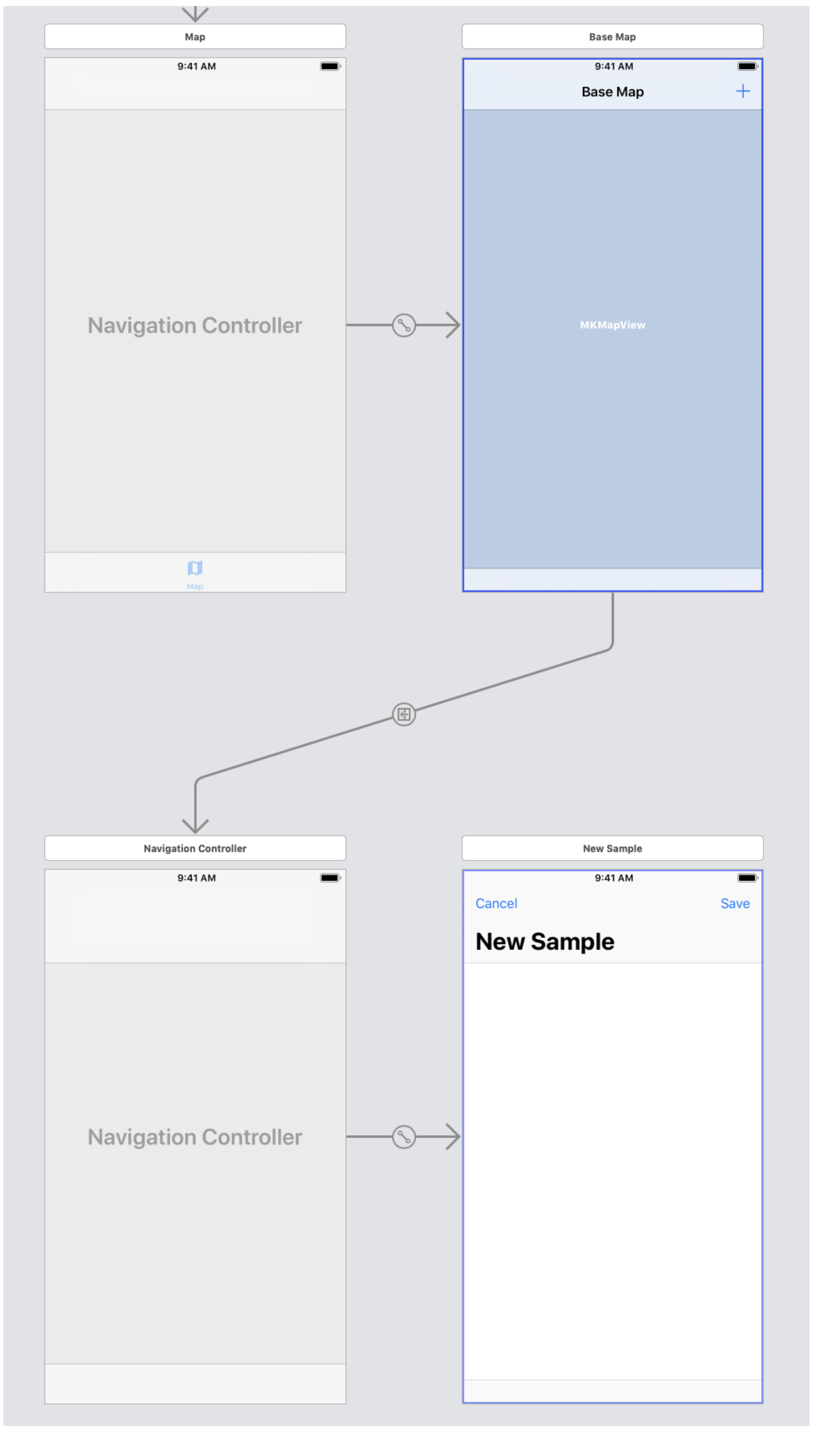
4 Swift Storyboard



This is an overview diagram for UIViews in our application. As description above, there are 3 main pages in the application, **Map page**, **Data page**, and **Settings page**; descriptions of these pages are shown below. There is also a log-in page which works as the entry point of the application. It will set the text users entered as the default geologist name for new data-points. All behaviors in log-in page will be handled by *GeologistLoginViewController.swift*.

In IOS development, bar controllers and navigation controllers are designed for allowing users to navigate among pages. There usually have simple functionality for entering and exiting pages.

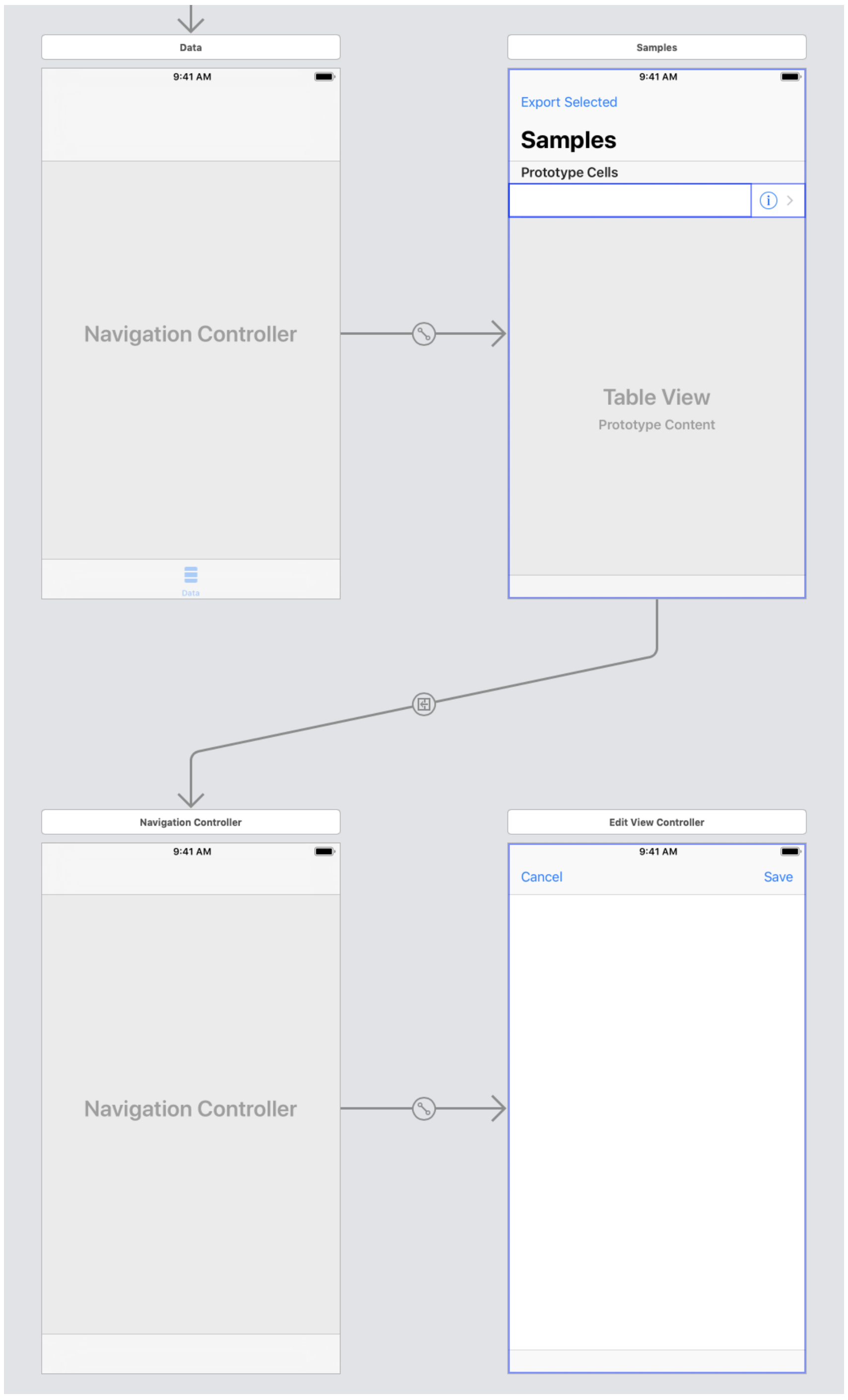
4.1 Map Page



This diagram shows all pages in the **Map** page. The Navigation Controller at top-left handles the event that users press map button on the bottom bar or after users log-in. The MKMapView at top-right is a basic map provided by Apple; it allows users to interact with a dynamic map and provides the ability for showing new data-points as pins. By pressing these pins, the user could check or edit the data of specific data-point. All behaviors in Map page will be handled by *MapViewController.swift*.

The Navigation Controller at bottom-left handles the event that users press "+" button at Map page, which will generate a data entry form for a new data-point. This data entry form will be generated in the New Sample page at bottom-right of diagram above. This form includes all the data users need to enter for a new data-point. All behaviors in New Sample page will be handled by *AddDataViewController.swift*, which is a subclass of the FormViewController from Eureka framework.

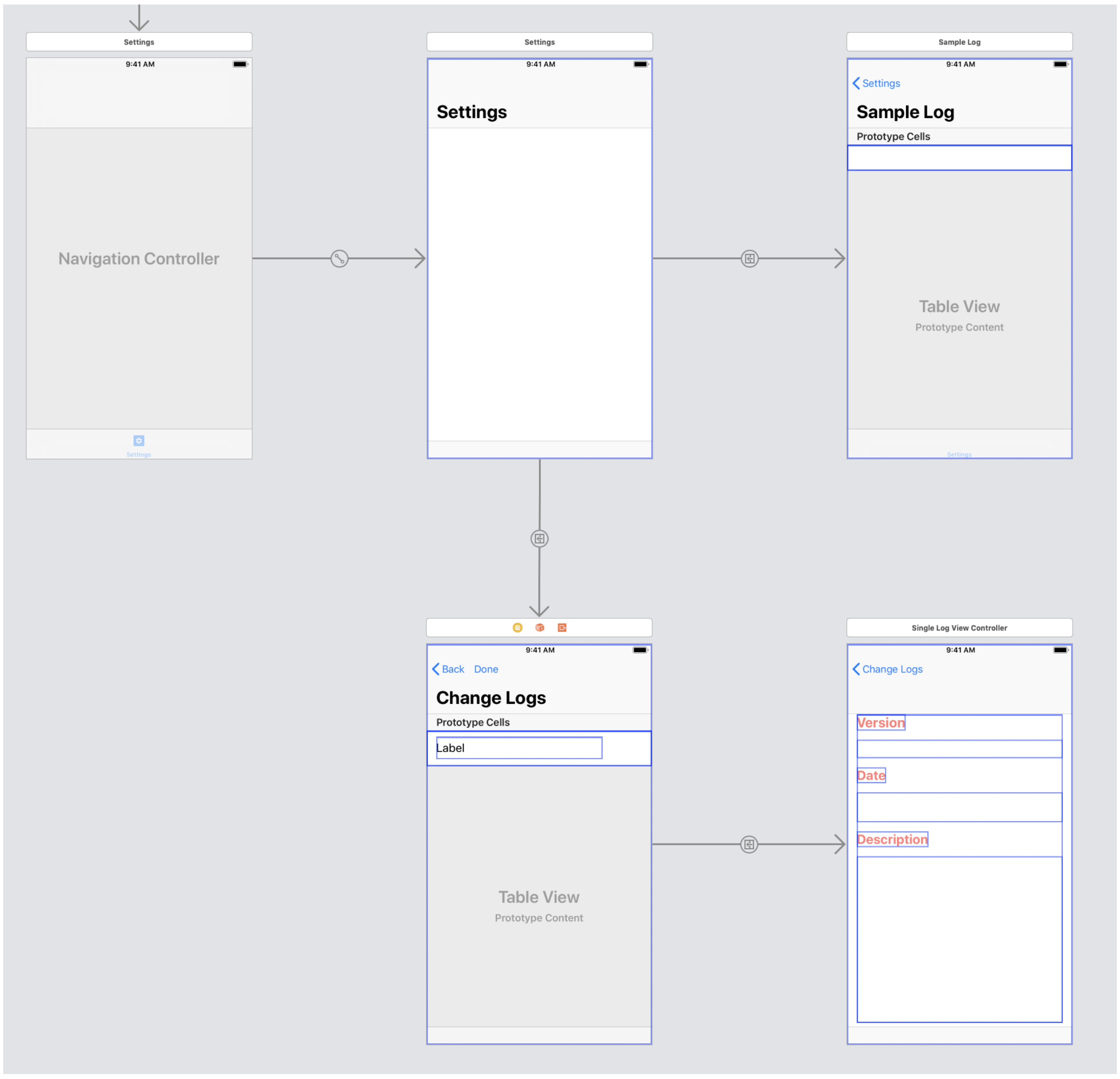
4.2 Data Page



This diagram shows all pages in the **Data** page. The Navigation controller at top-left handles the event that users press data button on the bottom bar. The Table view at top-right is a table for showing all local data-points as a list. The **Export Selected** botton at top-left of the page includes a functionality that exporting all selected data-points as a zip file. There is another button at the top-right of the page which is a **Edit** button (it does not show on this diagram but will appear when users enter into the Data page). By pressing this button, users are allowed to delete samples in the table. In the table, each data-point has a information button at the most-right; by pressing this button, users could get into the form for each data-point to check or edit data. All behaviors in Data page will be handled by *DataViewController.swift*.

The Navigation Controller at bottom-left handles the event that users press information buttons of specific data-point. Then, the application will generate a form for this specific data-point with the entered data in the Sample page at bottom-right of the diagram above. Users could check and edit all fields of the data-points. There will be two buttons on the top of the Sample page, **Cancel** and **Save**. By pressing Cancel button, the application will dismiss all the changes on the form. By pressing Save Button, the application will store all the changes to the Core Data. All behaviors in this Sample page will be handled by *EditViewController.swift*, which is a subclass of the FormViewController from Eureka framework.

4.3 Settings Page



This diagram shows all pages in the **Setttings** page. The Navigation controller at top-left handles the event that users press settings button on the bottom bar. The Settings page at top-middle of the diagram is showing all categories of settings of this application. There are Geologist Name, Sample Log and Change Logs. Users could change the Geologist Name here; the default content of this is the text users entered when they open the application. All behaviors in this Settings page will be handled by *SettingsViewController*, which is a subclass of the FormViewController from Eureka framework.

The Sample Log page at the top-right should contain all the historical samples as a archive of this application. This feature is not completed because there are many other tasks have more priority.

The Change Log page at the bottom-middle is a table page for users to check the version changes of this application. By pressing each version, the application will show the details, including version name, posted date for version, and version description, in the Single Change Log page at the bottom-right of the diagram. All behaviors in the Change Logs table view is will be handled by *ChangeLogTableViewController.swift*. All behavior in the Single Change Log page will be handled by *SingleLogViewController.swift*.

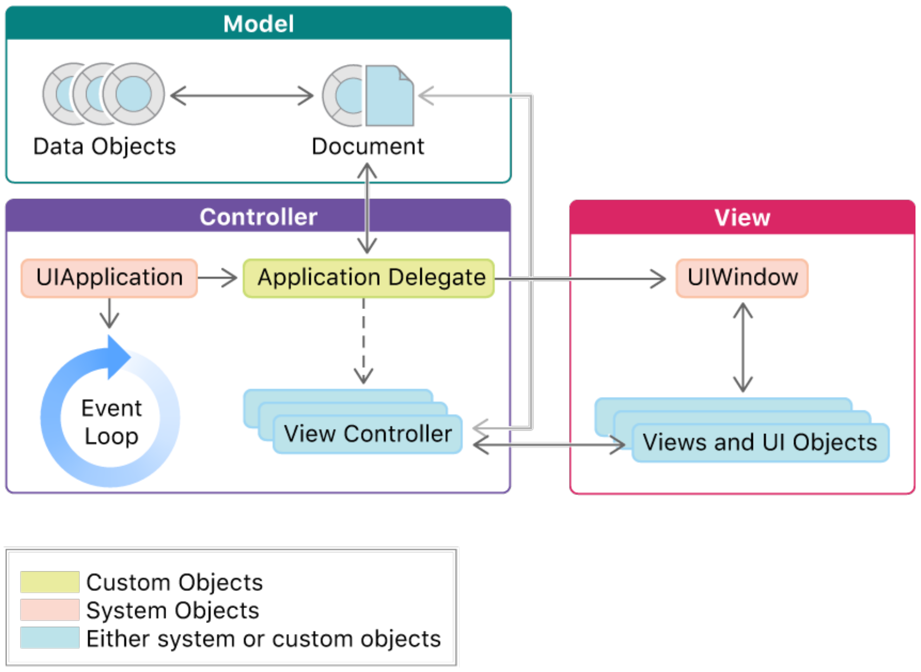
5 Implementation

5.1 Style Guide

Swift official design guideline:

- `https://swift.org/documentation/api-design-guidelines/`

5.2 MVC Model for IOS Development



IOS application is designed to develop in the pattern of Model-View-Controller. In our project, we implemented the Model part by using the Core Data, which is the local storage for IOS devices. Every time a new data-point is created, the corresponding view controller generates an object for this data-point and store it in the Core data.

In the development environment in Xcode and Swift, view and controller are bound together. Each UIView in the application is required a UIViewController to have the ability to listen to the users' activities (events). **Swift storyboard** section below will explain the details of implementation in our project.

In our project, we used the Eureka framework to construct lists and forms for the application. Specifically, we extend the *FormViewController* in the Eureka framework for constructing data entry form which is used for adding, checking and editing new data-point.

5.3 Algorithms, Data Structures, and Design Patterns

In order to make sure our project is easy to understand and maintain in the future, we intentionally avoided using needlessly complex design patterns and algorithms in our app. We tried to stay as close to the standard Swift/IOS design patterns as possible. This meant we made heavy use of the MVC model (Model-View-Controller). Our main data model objects are the DataPoint and GeologicalMaterial objects in CoreData. These hold all of he information for each sample. We used UIView objects to display information to the user. These are UI elements such as tables, buttons, text, etc. We then used ViewController objects to facilitate navigation between views, listen for events, and manipulate data.

To ensure fault tolerance, we made sure that all lines of code that could throw exceptions were wrapped in appropriate do-catch statements. We made sure that these statements print meaningful error messages to the console to enable easy debugging for future development. We also took steps to ensure that whenever a delete, edit, or export function is called, all appropriate data is removed from the device. For example, when a sample is deleted, the images tied to that sample are deleted(only the copies, not the original images). Then the sample is removed from CoreData, and all content on the DataView page is refreshed. This ensures that there are no bugs where old data is conflicting with new data. It also means that any unnecessary data, such as deleted images, are fully removed from the device to avoid using any unnecessary storage.

5.4 Data Storage

The data storage for QDI Mobile utilizes two different storage techniques provided by Apple. **User Defaults** and **Core Data**.

5.4.1 User Defaults

According to Apple, User Defaults are an interface to the user's defaults database, where you store key-value pairs persistently across launches of an app. In this case, we use it to store basic user settings that need to be persistent through all launches. For example, the **geologist name**. User Defaults were used to store this. For future implementation, it is recommended to use User Defaults for any configurations or settings that you want to store for the user.

5.4.2 Core Data

Core Data is Apple's main persistent store technique in iOS and Cocoa Touch, as it provides the ability to persist or cache data. As Apple states, it is *not a database*, but rather a *framework* that uses a SQLite database to store and retrieve structured data. It abstracts the direct database communication to make transactions quick and easy. Xcode also includes an easy-to-use interface for modeling your data. For QDI Mobile, we used Core Data to store all the samples that a user inputs.

To model your data in Core Data, you create different **entities**, which are basically tables. Each entity has **attributes** associated with them, which are basically columns. Different entities can also have different types of relationships between them. For instance, a *one-to-many* or *one-to-one* relationship. In code, to store objects in Core Data, you would create these entities and populate their attributes before writing to disk. Each entry you store would be similar to a row in a database.

It is easy to think of this more so in an object-oriented sense. You create *objects* and populate their *attributes*. Each *row* is like an instance of an object.

For QDI Mobile, we had two separate entities: **DataPoint** and **GeologicalMaterial**. Below are the attributes and relationships associated with each.

DataPoint:

- county

- dataSource
- elevation
- elevation
- geologist
- imgPath - the path to the core image
- img2Path - the path to the exposure image
- inputSource
- latitude
- longitude
- locationMethod
- method
- section
- siteName
- source
- subSection
- timeStamp - the timestamp when the sample was originally recorded
- townRange
- type
- **Relationship:** Geological Material

GeologicalMaterial:

- cal
- color

- depositType
- fieldDrillerDescription
- to
- from
- moisture
- munsell
- munsell2
- name

Database transfers are done in three scenarios. When the user **creates**, **updates**, or **deletes** an entry. When the user creates a new entry, a new object is created to be added to the Core Data store. When a user edits an entry, the object is edited and the changes are written to disk. When the user deletes an entry, this object is deleted from the Core Data store.

More information can be found on these topics on the Apple Developer documentation site:

- Core Data: `https://developer.apple.com/documentation/coredata`
- User Defaults: `https://developer.apple.com/documentation/foundation/userdefaults`
- A nice tutorial for Core Data: https://www.raywenderlich.com/7569-getting-started-...

5.5 Testing and Verification

Our repository requires at least one person to accept a pull request created by someone who has made a change in the code and wants to make those changes in the code base. Before we get to this point, we test any changes we make by running the simulator in Xcode which allows you to run the app in a simulated version of almost any Apple device. We also test it by plugging in one of our iPhones and testing it like that on our physical phone. At the end of every sprint, we publish a new version of the app to test flight, so that MGS employees can install the new version of the app and test it themselves.

6 Project Management

6.1 Overview

Our project was completed over the course of the spring 2019 academic semester. Work on this project was divided into two week sprints, with each sprint having it's own set of development goals. Each sprint, one member of the team was assigned as the group manager, responsible for coordinating with other teams, setting goals for the sprint, and ensuring all team members had the resources they needed. This group manager changed week to week. At the end of each sprint we aimed to have a new deliverable, which was typically a new release of our app on TestFlight. Below is a development timeline, listing the members, manager, and important milestones achieved for each sprint. There were five sprints in total, and we have also included a description of what was achieved after the end of sprint five. For a more details about what was achieved in a specific sprint, you can reference the individual sprint reports included in the appendices of this report.

6.2 Sprints

6.2.1 Sprint 1

- **Goals -** There were three goals for this sprint. We wanted to compile a requirements document for the project. We wanted to create a wireframe diagram for how we envisioned the app. Finally, we wanted to get the basic app set up with the outlines and navigation.
- **Members**
  - Page Bennett (Manager)
  - Zachary Glasgow
  - Drew Wilken
- **Deliverables/Milestones**
  - Completed requirements document
  - Completed app wire-frame diagram
  - Set up basic app with basic navigation between screens
  - Released version 0.1

6.2.2 Sprint 2

- **Goals -** This sprint we wanted to get core data set up in our project, allowing us to save datapoint objects to the phone's local storage. We also wanted to be able to fetch that data and display it in the edit data screen.
- **Members**
  - Zachary Glasgow (manager)
  - Page Bennett
  - Drew Wilken
- **Deliverables/Milestones**
  - Set up core data in our project
  - Added ability to save data points to local storage
  - Released version 0.2
  - Incomplete - Ability to fetch and display core data objects

6.2.3 Sprint 3

- **Goals -** We had three main goals for this sprint. We wanted to be able to view and edit data points and their data. We also wanted pins on the map to show the locations of existing data points. Finally, we wanted to add a changelog page within the app.
- **Members**
  - Wenxuan Li (Manager)
  - Page Bennett
  - Zachary Glasgow
  - Drew Wilken
- **Deliverables/Milestones**
  - Added ability to view and edit data points
  - Added pins to map page showing data point locations
  - Added changelog page to app

  - Released version 0.3
  - Created imtermediate report

6.2.4 Sprint 4

- **Goals -** Our primary goal for this sprint was adding the ability to add images to data points. This meant adding the functionality to attach or take photos in the app, as well as adding the infrastructure to save images, reference them, and delete them. We also wanted to determine how we would be getting the datapoints to the QDI Database.
- **Members**
  - Drew Wilken (Manager)
  - Page Bennett
  - Wenxuan Li
  - Leo Molitor
- **Deliverables/Milestones**
  - Added the ability to add photos to a datapoint
  - Added ability to save and delete photos on disc
  - Incomplete - Did not determine how we would be getting data to the QDI database.
  - Released version 0.4

6.2.5 Sprint 5

- **Goals -** The end goal of this sprint was to accomplish 3 main things: have our app communicate with the database, have it validate input data, and add a feature that allows the user to select a pin and have its sample show in a pop-up.
- **Members**
  - Leo Molitor (Manager)
  - Page Bennett
  - Drew Wilken

  - Wenxuan Li

- **Deliverables/Milestones**
  - Added the ability to export datapoints as a zip file containing a JSON file and image files
  - Added Munsell color selection
  - Incomplete - Data validation, as other teams would be validating data we sent them
  - Incomplete - Showing data when a pin on the map was selected

6.2.6 Post Sprint 5

There were several tasks accomplished between the end of the final official sprint and the writing of this report.

- Added ability to tap pin and see information about the data point
- Finalized the data entry page
- Finished commenting all methods, including in-depth comments for all custom methods using Jazzy markup.
- Released version 0.5

7 Reflections

7.1 Technical Challenges

We had a few technical challenges as a group, but not many. Firstly, we didn't initially have any computers to develop the application on. Apple requires you to own a Mac of some sort to develop any iOS applications, so only one group member was able to work on the app, because they had a MacBook. We had to go about working through ITS at the university to get Xcode installed on 3 of the few Macs they have on campus. We did this and got the software installed in the library, so another technical challenge was that if we wanted to work on the code and didn't have any other Apple products to develop it on, we had to be in the library on these computers - not forgetting to mention that anyone could use these computers, so we had to make sure no one was.

Apple also requires you to jump through a lot of hoops and security checks to test your applications. You need to be a registered Apple Developer, too, which costs a hundred dollars per year. Not only this, but you had to have an iPhone to test it on if you wanted to try it on an actual device, not a simulator.

Aside from these technical challenges, only one group member was somewhat versed in iOS development, so the rest of the group members had to catch up and learn the ropes on the whole Apple and Cocoa Touch environment, let alone learn a new language - Swift.

There was also the challenge of finding the right medium to transfer the data to the actual QDI database - the most essential part of the project. Since the whole ArcGIS ecosystem is very complicated, it was hard to do so and also to figure out how to do so through network. We settled on doing so with a flat file, as this alleviated a lot of the stress and allowed a transfer of data that was easy to do, especially without network connection. This method also turned out to be the best for the field and offline use, and the lightest weight of any of them.

7.2 Retrospective Changes

If we were to have the chance to do this project over again, it'd be beneficial for us to get to know the ArcGIS ecosystem and frameworks better and to better research how the QDI and Visual Basic programs work. As the mobile team, we were very separated from the rest of the teams, as was our software, so it was hard for us to jump in when it was time to think about data transfer and understand the best way to get the data into the database. So, if we had the chance to do it over again, this is what we would spend more time on.

7.3 Continued Changes

If we had the chance to continue working on this project, we would do just as it says above - to research the ArcGIS ecosystem more and figure out other ways to transfer data more effectively.

We would also take a look at the Collector App by Esri and note the features which are included in this. Since this is the ideal application state, it'd be useful to take some notes from Esri. This could include more customization on the UI of the map, ability to add different layers, to better visualize the data, and maybe most importantly *see existing datapoints from the QDI database on the map*, as this was a feature that the MGS team voiced as being very important.

8 References

1. Apple Developer. *Xcode,* https://developer.apple.com/xcode/
2. Jazzy. *Documentation Tool for Swift,* https://github.com/realm/jazzy
3. Apple Developer. *About App Development with UIKit,* https://developer.apple.com/documentation/uikit/about_app_development_with_uikit
4. Eureka. *Elegant iOS form builder in Swift,* https://eurekacommunity.github.io/
5. Swift. *API Design Guideline,* https://swift.org/documentation/api-design-guidelines/
6. ZIPFoundation. *a library to create, read and modify ZIP archive files,* https://github.com/weichsel/ZIPFoundation
7. CocoaPods. *Dependency manager for Swift,* https://cocoapods.org/

Appendices

A Sprint Report 1

This is the mobile team's report for the first sprint.

Sprint Report

Mobile Team

University of St. Thomas
CISC 480

March 17, 2019

End Goal of Sprint

We wanted to have three main deliverables this sprint. First, we wanted to create a requirements document containing what we believed to be the main features that needed to be in the final app. Second, we wanted to create a wireframe diagram of what we envisioned the app to look like. Finally, we wanted to get the app started, and get it functioning with all of the pages, navigation between them, and inputs/text fields placed.

Deliverable Status

- Requirements document: Complete
- Wireframe diagram: Complete
- Basic app: Semi-complete - The basic app is complete, however we ran into issues uploading the release to the App Store in order to share it via TestFlight. We are currently working through this issue.

Tasks for this Sprint

- Determine if the "Collector" app would work with the existing database/system: Complete
- Create the requirements document: Complete
- Create app wireframe: Complete
- Ensure library computers are set up with xcode and that they work with Github: Complete
- Create blank pages with navigation between them: Complete
- Add map to the maps page: Complete
- Add input fields to the data entry page: Complete
- Add log-in screen with geologist named saved as user default: Complete
- Set up core data in preparation for saving data points: Complete

- Create documentation for different releases(changelog): Complete
- Add app to test flight: In Progress

Team Member Review

Task Breakdown

- Drew Wilken
  - Create blank pages with navigation between them - Drew created the skeleton of our app, adding in all of the pages and the navigation buttons between them based off our wire-frame diagram.
  - Add input fields to the data entry page - Drew added all of the input fields needed to enter new data into the app and made it look nice
  - Add basic map to maps page - Added a basic map using the using the built-in ios map to the maps page of the app
  - Add log-in screen with geologist named saved as user default - Drew added functionality to enter a geologist's name on the login page. The name then autofills into all relevant data fields in the app, can be edited in settings, and saves across sessions.
  - Add app in test flight: Drew attempted to upload version 0.1 of our app to the app store to be distributed through TestFlight, but ran into issues part-way through the process which he is still working to resolve.
- Zachary Glasgow
  - Create the requirements document - Wrote up a draft of the requirements document based off emails we received from MGS. After all team members had provided input, he edited it and proofed it.
  - Create app wireframe - Created the app wireframe diagram with input from the rest of the team.

  - Create documentation for different releases(changelog) - Created a writeup for the intitial release of our app (version 0.1) explaining the current state of the app for our changelog

- Page Bennett
  - Set up core data in preparation for saving data points: Researched Core Data library for swift, created code which will allow for saving of new data points in the ios local storage.
  - Manager activities - Facilitating meetings, maintaining task board, communicating with other teams, organizing standups, etc.

- Group Tasks
  - Determine if the "Collector" app would work with the existing database/system - All members spent time researching the ArcGIS Collector app to determine what functionality it had, if it would work with the current setup of the QDI database, and if we could utilize any parts of the ArcGIS mobile SDK for our own app. We determined it would not work with the current setup, and did not find anything particularly useful in the SDK.
  - Ensure library computers are set up with Xcode and that they work with Github

Member Conduct

- Drew Wilken
  - Overall share of work completed: 45%
  - Drew came in as the only member of our team with any experience using Swift or doing any app development for iOS. As a result, Drew was able to complete much more of the development work than me(Page) or Zachary. He did a good job reporting his progress on the different tasks he was completing. He was also a valuable source of information for the rest of the team, explaining how he completed each task and linking us to tutorials he thought might be helpful to us. Did a good job following chain of command.

- Zachary Glasgow
  - Overall share of work completed: 30%
  - Zach came into this project with no experience with using Swift or Xcode. He did a good job on the non-coding tasks he was assigned. He spent a lot of time going over the work Drew completed and learning from it, as well as going through Swift and iOS development tutorials. Did a good job following chain of command.
- Page Bennett
  - Overall share of work completed: 25%
  - I came into this project with no experience with Swift or app development. I think I did a good job organizing the sprint and making sure everyone was getting things done. I also spent considerable time going over what Drew completed and spending time on Swift tutorials.

Goals for Next Sprint

We want to accomplish:

- Save new data point entries to local storage with unique identifiers
- Display saved data entries on the data points page
- Allow for the editing of saved data points
- Add the changelog to the settings page
- Determine how we will be getting data entries from our app to the database
- Ability to convert data entries to a format that can be sent to the database.

Documentation

- Requirements document - Attached
- Wireframe diagram - Attached
- Initial release writeup for changelog
  - Initial Release (version 0.1 - 15 Mar 19) 3 seperate pages of the application that are navigable via a navigation bar at the bottom of the screen. The initial page a user will start on is the sign in page. This is where the user will enter their geologist ID which will then populate in forms that take that ID as input. Once signed in, the user is brought to the Map page where they can press the "+" button in the top right hand corner of the screen. If they press that button the user is brought to a page of Data Entry text boxes that contain specific data related to whatever new sample a user wishes to add. The settings page has two fields, one of which is the geologistID which is autofilled upon login and can be changed whenever necessary. The other field is a sample log which is not functional yet but will house a record of each sample that has been sent to QDI once the app is up and running.

Mobile team

Requirements

Basic and Core Features Description

Basic data collection tool that you can bring to the field where you can create new data-points and upload said datapoints to the QDI database.

The app will include a map which you can click on the location of the map and add a point to that location for a new sample, or press a button titled ‘record new sample’ which you can populate the location field yourself.

All datapoints will be stored locally until they are sent off to the database or exported to CSV; until that point is uploaded or exported, you may edit or delete it.

When entering data for the sample, the *information to be collected* points listed below will be available to enter.

There will be multiple ‘tabs’ within the app:

- The map with local points that haven’t been sent to database / exported yet
- A list page that has a table of local points that haven’t been sent to database, here you can delete or edit them manually
- Settings page where any configurations for the app can be figured out, whatever they are

There will be a log page that will show all data points collected on the device, or at least the relateIDs of each one.

Potential Advanced Features

- Ability to see existing data points in an area and directly manipulate points that are already existing in the QDI database
- Upload pictures of the sample
- Ability to interface with external elevation/GPS hardware
- Interface with elevation map to determine elevation from coordinates

There will be a sign-in page for the geologist, so that the data entry can populate the person who took the sample.

Features

- Basemap
- Add location of a point/sample
- Able to add information about the sample

- Store and locate pictures of the sample

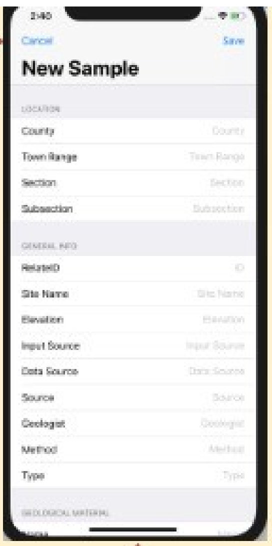
Information to be collected

- RelateID (assigned QDI Number)
- Location (County, Town Range, Section, Subsection)
- Site Name
- Elevation
- Location Method
- Input Source
- Data Source
- Source
- Geologist
- Method
- Type
- Geological Material (seperate tab)
    - Name (this is our sample same)
    - Field/Driller Description (personal notes about each sample)
    - From (depth of sample)
    - To (depth of sample)
    - Color
    - Deposit Type (drop down with one of the following options: Alluvial, Bedrock, Colluvium, Complex, Deltaic, Eolian, Fill, Fluvial, Ice Contac, Ice-walled lake plain, Lacustrine, Lag, Loess, Outwash, Peat, Residuum, Saprolite, Till, Till Uncertain, Top Soil)
    - Moisture (drop down with one of the following options SATURATED, MOIST,DRY, WET
    - Munsell (drop down with complete list of color numbers found in Munsell Color Book)
    - Munsell 2 (same as above)
    - Cal (drop down with yes/no)

Notes

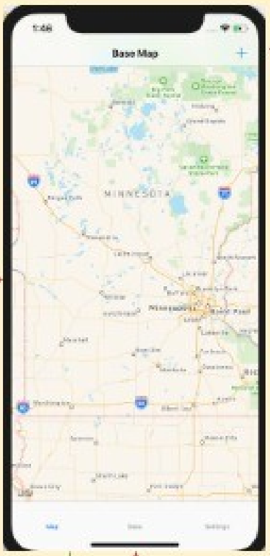
- Collector does not work with the current database. The reason I brought it up is, we would like a QDI app to look pretty much identical to the Collector app but is able to interact with our Database. Your idea of a nice collection tool where we could enter data in and send it off to the database from the field is exactly what we are looking for.

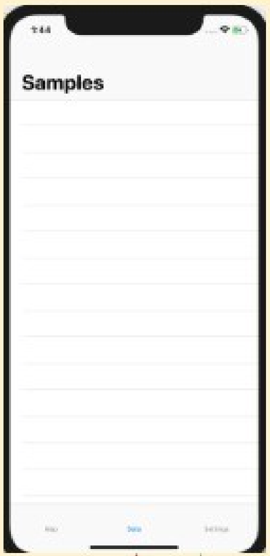
- Information about each sample- this information currently shows up in the QDI interface with drop downs. I've attached a screen shot of how this currently looks for the geologist as they are entering data
- Geological Material-there is a tab labeled Geological Material in the current Geological Material, this is where the geologist enters information specific to each sample they take. I've attached a screen shot of what this tab looks like and the information we want to include for each sample. We need a field for all the information that is currently in the Geological Material Tab but I've included a list of the field we would like to fill out while doing our fieldwork.

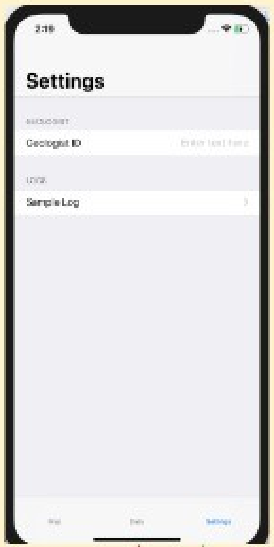


This page will come up when either the map or the plus button are clicked. This is where a user will enter the necessary info to start a data point. (excludes geological data)

QDI Mobile Application







This is the base map for this application. Will show current location and allow user to add data points to the map.

The Samples page will show all of the local data points that have not been exported to the QDI database. Each sample is clickable, which brings the user to a page where they are able to edit the necessary geological information for a point.

The setting page will have fields for general information a frequent user could pre-populate. (ie geologists name)

B Sprint Report 2

This is the mobile team's report for the second sprint.

Sprint Report

Mobile Application Team

University of St. Thomas
CISC 480

April 3, 2019

End Goal of Sprint

We wanted to get the Core Data (Apple's form of local storage on Apple devices) working on the application. If time permitted we wanted to get the Core Data fetched and printed to the Samples screen, so that the user could see the previous samples the have added. Note: We were permitted by Dr. Myre to have a lighter sprint due to the fact that we are only able to develop on Apple Devices and have limited access to those devices over spring break.

Deliverable Status

Semi-complete - We have the Core Data up and running but we have not been able to fetch the data and print it to the Samples screen on the application.

Tasks for this Sprint

- Core Data Created: Complete
- Fetching of Core Data: Incomplete, because we have not been able to figure out how to print from the Core Data database.

Team Member Review

Task Breakdown

- Page Bennett, Drew Wilken, Zach Glasgow
  - Figure out format in which data is stored within Apple local storage
  - Take data entered by user in the Add Datapoint page and have it sent to the devices local storage
  - Access data that has been stored in local storage and store the individual data points in objects
  - Display the objects on our Samples page for review by the user

Member Conduct

- Page Bennett
  - Very Proactive and worked hard over break while Drew and Zach were not able to
  - Work Done: 65%
- Drew Wilken
  - Figured out the format that our data would be stored in Core Data
  - Work Done: 20%
- Zach Glasgow
  - Researched Core data to determine how to store our data and how to fetch and print to our Samples page
  - Work Done: 15%

Goals for Next Sprint

We want to accomplish:

- Get the Core Data printed to the Samples page
  - This entails fetching our data each time it is stored and printing it to the Samples page. This should update each time a sample is add to our storage.
- Make Samples on sample page clickable and able to be reviewed in detail (not edited)
  - In the future we want the samples to be able to be edited and reviewed before being sent to the database at QDI, so our first step will be turning these objects into viewable forms.

C Sprint Report 3

This is the mobile team's report for the third sprint.

Sprint Report

Mobile Team

University of St. Thomas
CISC 480

April 15, 2019

End Goal of Sprint

We wanted to have three main deliverables this sprint. First we wanted to create CRUD (created, read, upload and delete) operations for data points. Second, we wanted to add coordinates for new data points and use it to show data points as pins on the map. Finally, we wanted to create a page in setting for displaying the versions' information of our application.

Deliverable Status

- Data points CRUD operations: Complete
- Pins on the map: Complete
- Change Logs Page: Complete

Tasks for this Sprint

- Major Task 1: Add editing and deleting operations to data points
- Major Task 2: Display data points as pins on map
- Major Task 3: Add Change Logs Page to display updates for the application
- Major Task 4: Prepare intermediate demo and report.

Team Member Review

Task Breakdown

- Drew Wilken, Zach Glasgow
  - Work on editing operations on data points
  - Figure out displaying data points as pins on the map.
  - Introduce project and information about Swift and Xcode to new member in the group

- Page Bennett
  - Work on editing and deleting operations on
  - Write most pre code review document
- Vincent Li
  - Work on Change Logs page
  - Write the intermediate report.

Member Conduct

- Drew Wilken
  - Overall share of work complete: 35%
  - Implemented editing operation on data points
  - Corrected and update the data fields in Core data
  - Created the feature of displaying data points as pins on the map
- Page Bennett
  - Overall share of work complete: 20%
  - Implemented deleting operation and help with editing operation
  - Worked on pre code review document
- Zach Glasgow
  - Overall share of work complete: 20%
  - helped with implementing editing operation on data points
  - Researched the solution of displaying data points as pins
- Vincent Li
  - Overall share of work complete: 25%
  - Created the Change Logs page
  - Wrote the intermediate report
  - Worked on pre code review document

Goals for Next Sprint

We want to accomplish:

- Build the connection with Database and receive details for data points.
  - Our idea is to store coordinates for data points in the Database locally in the application, so that the user in MGS could see data points as pins on the map.
  - The final feature we want to implement is that the users could get details for any pins from the Database by pressing pins on the map.
- Optimizing the data entry form
  - The data entry form now is not complete yet. We need to add more field and check for validation.
  - We also want to add the feature of storing picture for data points.

D Sprint Report 4

This is the mobile team's report for the fourth sprint.

Sprint Report

Mobile Team

University of St. Thomas
CISC 480

May 1, 2019

End Goal of Sprint

We wanted to focus this sprint primarily on two things: Firstly, image capability. We wanted to add the capability for the device to store images with the samples and for each entry to have an image associated with it. Secondly, we wanted to talk with the other teams and collaborate on communication with the database and how we can send our sample data to the actual QDI database.

Deliverable Status

Semi-complete - We have completed the image capability for the samples. It is now possible to input a photo with each sample the user is entering, and also change the photo when editing the entry. With database communication, we didn't get a deliverable on it, as it is still a little unclear on the best way for us to send the data over. So, we have discussed it and are working out details but there are no deliverables on the actual database communication for this sprint.

Tasks for this Sprint

- Major Task 1: Add the capability for a sample to hold an image and for the user to add, change, and delete images associated.
- Major Task 2: Add the capability for images to persist, storing the filepath in the database with the image stored on disk
- Major Task 3: Work with database team and other teams on communication with database and figure out the best way for us to send sample data.

Team Member Review

Task Breakdown

- Drew Wilken

  - Image capability and persistent storing
  - Talk with other teams on database communication
  - Research ArcGIS SDK

- Page Bennett
  - Image deletion
  - Talk with other teams on database communication
  - Research ArcGIS SDK

- Vincent Li
  - Help with Image Capability
  - Research on ArcGIS SDK for Swift

- Leo Molitor
  - Review Swift and get comfortable with Cocoa Touch (Apple Framework)
  - Research on ArcGIS SDK

Member Conduct

- Drew Wilken
  - Percentage of work: 35%
  - Overall led the team on adding image capability and method of storing images
  - Did research on ArcGIS SDK for Swift and formulated ideas with team on communicating with Database.

- Page Bennett
  - Percentage of work: 35%
  - Developed the photo deleting feature that allows user to delete the photo associated with a record and also takes the image off the device once a user deletes a sample.

    - Did research on ArcGIS SDK for Swift and helped formulate ideas with team on communicating with Database.

- Vincent Li
    - Percentage of work: 15%
    - Did research on ArcGIS SDK for Swift and helped formulate ideas with team on communicating with Database.
    - Got the data validation code for VB team and begun implementing in Swift to validate the data the user is sending to the database.

- Leo Molitor
    - Percentage of work: 15%
    - Did research on ArcGIS SDK
    - Worked with database team to find out ways we could send data
    - Did research on Swift and got up to speed with Cocoa Touch and iOS development.

Goals for Next Sprint

We want to accomplish:

- Communicate with the database - this next sprint we want to go specifically for just getting some communication with the database and/or server. For example, a request or some sort of validation that we have a connection with the database.

- Data validation on the iOS app for when the user sends the data off to the server. For example, making sure there are no duplicate RelateIDs or anything.

- Click on pins on the map to reveal information about the sample(s) there - the user can click on the map pins that represent different points and a popup will show the data of that sample.

E Sprint Report 5

This is the mobile team's report for the fifth sprint.

Sprint Report

Mobile Team

University of St. Thomas
CISC 480

May 13, 2019

End Goal of Sprint

The end goal of this sprint was to accomplish 3 main things: have our app communicate with the database, have it validate input data, and add a feature that allows the user to select a pin and have its sample show in a pop-up.

Deliverable Status

Partially complete - We have a way to get our sample data from the app to the database, but not in the way we would have hoped. Instead of directly connecting to the database, the app creates a .zip file with the data from samples in JSON format, along with each sample's corresponding pictures. The .zip files are then sent to the database via the C# team or VB team's import button that they have been working on. As far as the input data validation goes, we ended up not doing much of this because we found out that the database team was already working on it, so that the data being sent to the database is usable. The pin pop-up feature did not get finished, as it took a back seat when we were trying to figure out how we were going to get the sample data to the database, since that was more important.

Tasks for this Sprint

- Major Task 1: Communicate with the database. We needed to find a way to get our samples' data into the database so it can be utilized by QDI.
- Major Task 2: Validate data that is being entered in the app so that it can be sent to the database.
- Major Task 3: Add a feature where a pin can be selected from the map and the samples from that location are displayed in a pop up.

Team Member Review

Task Breakdown

- Drew Wilken

    - Comment AddDataViewController
    - Add capability to add 2 types of images
    - Comment AddDataViewController

- Page Bennett
    - Commented DataViewController
    - Get app to make sample data into a format that can be exported to VB and C# teams

- Vincent Li
    - Comment rest of code
    - Create documentation outline
    - Work on data validation

- Leo Molitor
    - Comment MapViewController
    - Figure out how Munsell and Cal values should be represented, add it to app
    - Get familiarized with Swift and app frame work

Member Conduct

- Drew Wilken, work done: 30%
    - Found way to automatically create documentation and display it in html
    - Commented MapViewController
    - Commented AddDataViewController
    - Added capability to add 2 types of images
    - Commented AddDataViewController

- Page Bennett, work done: 30%
    - Commented DataViewController

  - Talked to C# team and VB team about how to get them the sample data
  - Added ability to export sample data and images

- Vincent Li, work done: 25%
  - Commented code
  - Created documentation outline
  - Worked on data validation

- Leo Molitor, work done: 15%
  - Communicated with Jenn about Munsell colors and images being taken in the field
  - Added Munsell colors to be able to be chosen in app
  - Got familiarized with Swift and app frame work

Goals for Next Sprint

We want to accomplish:

- Documentation
  - Mobile Team final report
  - Mobile Team documentation
  - Full class report, including Mobile Team documentation