QDI MOBILE DOCUMENTATION

May 26, 2019

Drew Wilken
Page Bennett
Leo Molitor
Vincent Li

Contents

1 Introduction

QDI Mobile application is a basic data collection tool on IOS platform that users can bring to the field where they can create new data-points and upload said data-points to the QDI database. Instead of using sketch paper for record, users could use QDI Mobile to record data at the field, and export new data-points to their computer and upload to the database.

There are 3 main pages in the application. In the Map page, users could check their new data-points on the map and create new data-point. In the Data page, users have more options to operate data-points in the local storage. In the setting page, users are allowed to change the name of geologist which will auto-fill in the new data-points.

During the development of the application, our group used Xcode 10.2 and Swift 5. They were both the newest version for IOS development at the time. Additionally, we use Eureka for building the data entry form.

2 Installation Documentation

There are 3 way to get the QDI Mobile application. The first one is **for development purpose**, which allows developers to run the newest modified code and run the application on the simulators in OSX. The second one is **for beta testing**; after developers release the beta of new versions, testers of the application could download the updates from TestFlight on their IOS devices. The last one is **for regular usage**, users could download the application from the App store on their IOS devices.

2.1 For development purpose

1. You need an OSX device and an Apple developer Account.
2. Download the newest version of Xcode. Link: Downloads for Apple Developers
3. Clone project from Github. Link: `https://github.com/ust-cisc480s19-myre/Mobile_Dev`
4. Double-click the file **Mobile_Dev/QDIMobile.xcworkspace** or right-click this file and open with Xcode.

2.2 For testing purpose (on IOS Device)

1. Go to itunes connect and sign in with the Apple development account.
2. Click the QDI Mobile application in the grid
3. Click **TestFlight** tab at the top of the page.
4. Click **External testing** on the left.
5. Add new testers by entering their first name, last name and email address.
6. New testers need to download TestFlight Application on their IOS device.
7. Check emails sent from TestFlight on their IOS device.
8. Click **"View in TestFlight"** in the email, and click download in the TestFlight application.

2.3 For using purpose (for mobile application users)

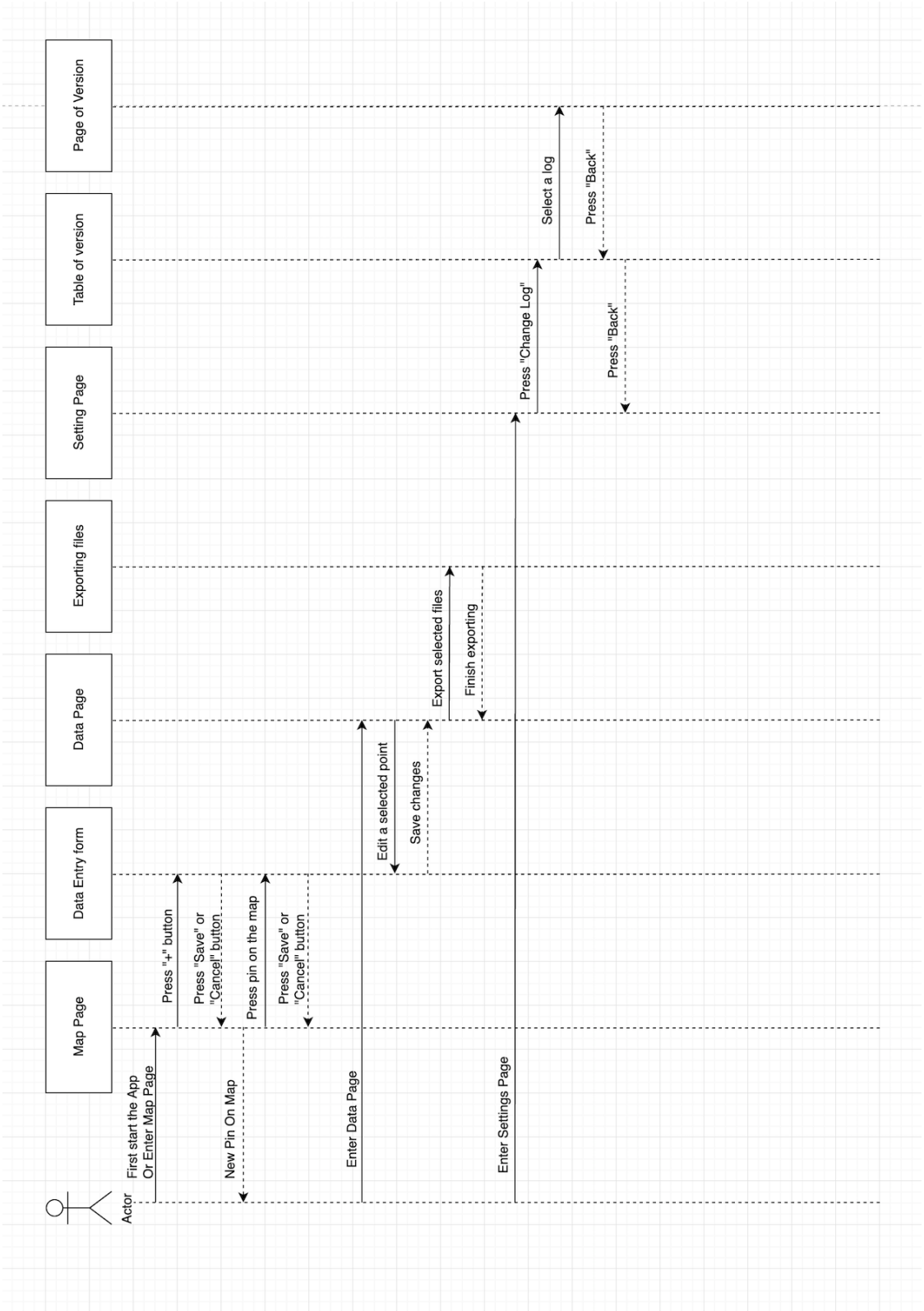
1. You need an IOS device to use the QDI mobile application.
2. Make sure the QDI Mobile Application is released to the public (on App store).
3. Open App Store in your IOS device and search **QDI Mobile**.
4. Click **Download**.

3 Use Case Diagrams

3.1 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"

3.2 Basic interface of the application

There are 3 main pages in the application, **Map**, **Data**, and **Settings**.

In **Map** page, users could add new data-points to the local storage by pressing **"+"** button at top-right, it will use the current coordinate of users by default and users are able to change it. On the other hand, users could check and edit all local data-points by pressing the pins on the Map. All those points are available in the Data page as well.

In **Data** page, users could check all data-points in local storage as a list. Users could check details of the point or edit data by pressing the information button of each data-points. By pressing **"Edit"** button at top-right, users could delete the data-points from the local storage. For exporting data-points as a zip file, users could select multiple data-points (selected point will turn to grey), and pressing **"Export selected"** at top-left. After the data-points are compressed into one single zip file, users could choose the approach to share the file.

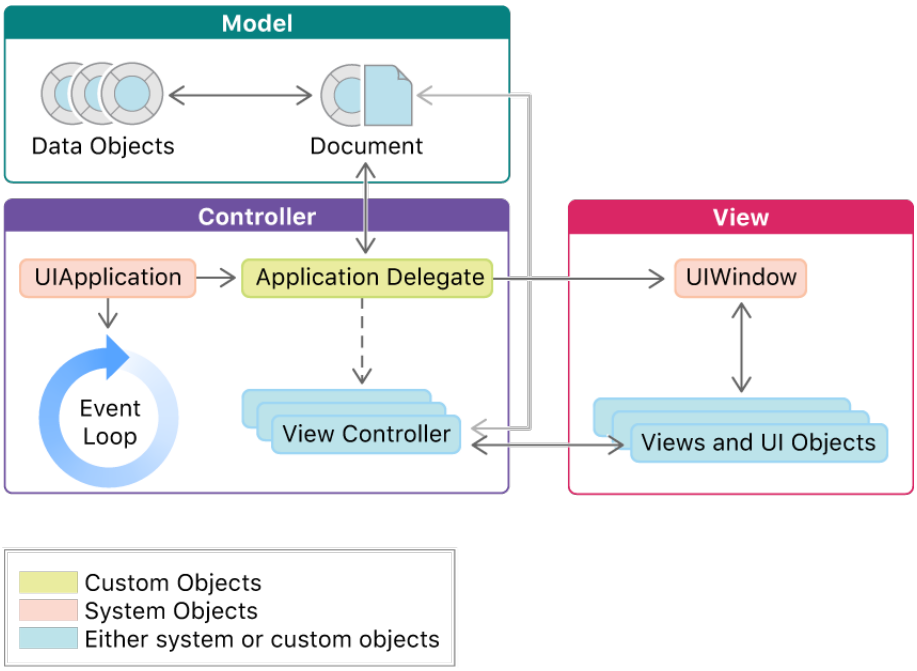
In **Setting** page, users could change the name of geologist (which was set in the log-in page, and will use as default when users are creating new data-points). In Change Log, users could check the updates for each version.

4 Api

QDI Mobile API Documentation

- `https://ust-cisc480s19-myre.github.io/MobileAPI/`

5 UML Diagram

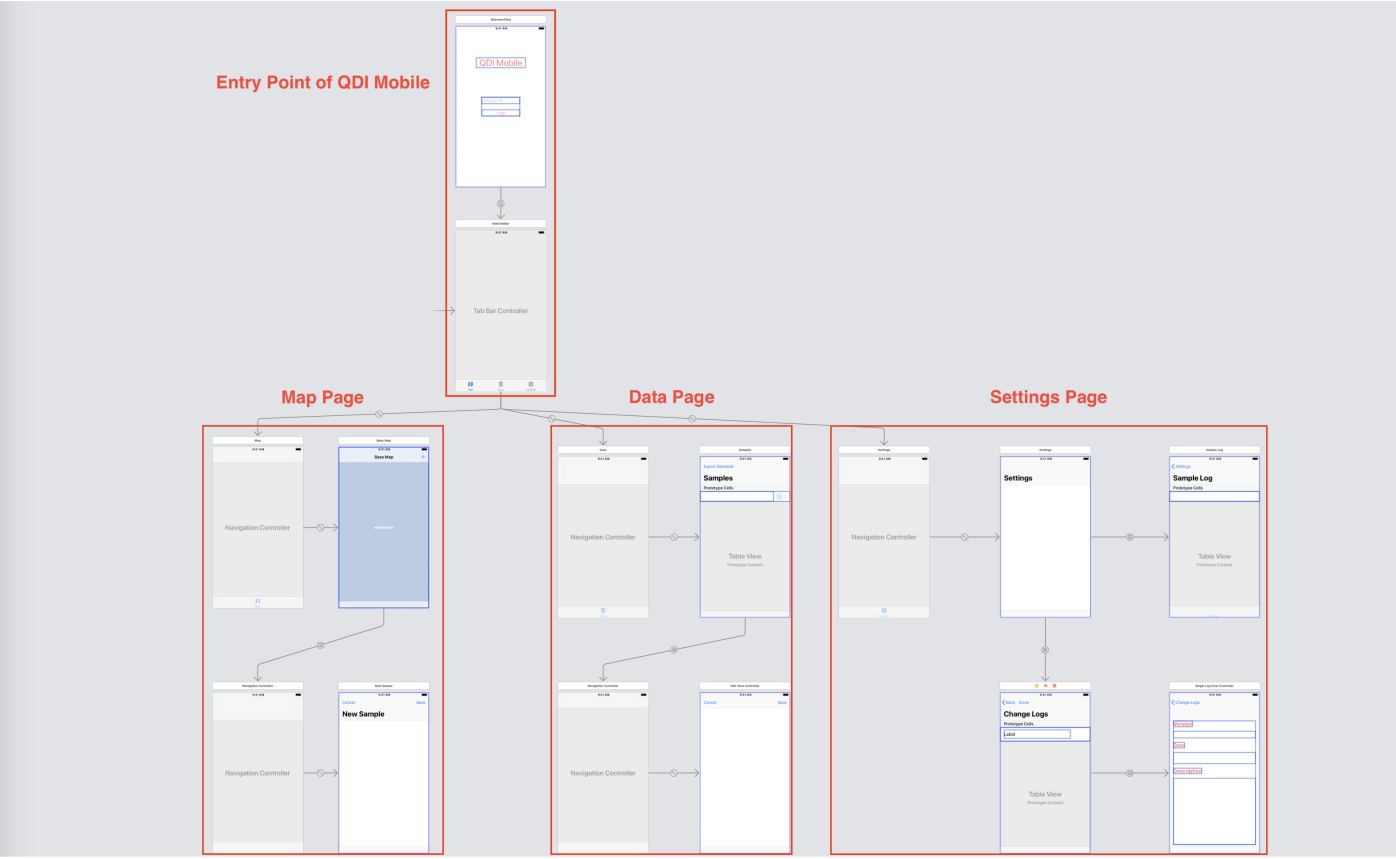


IOS application is designed to develop in the pattern of Model-View-Controller. In our project, we implemented the Model part by using the Core Data, which is the local storage for IOS devices. Every time a new data-point is created, the corresponding view controller generates an object for this data-point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

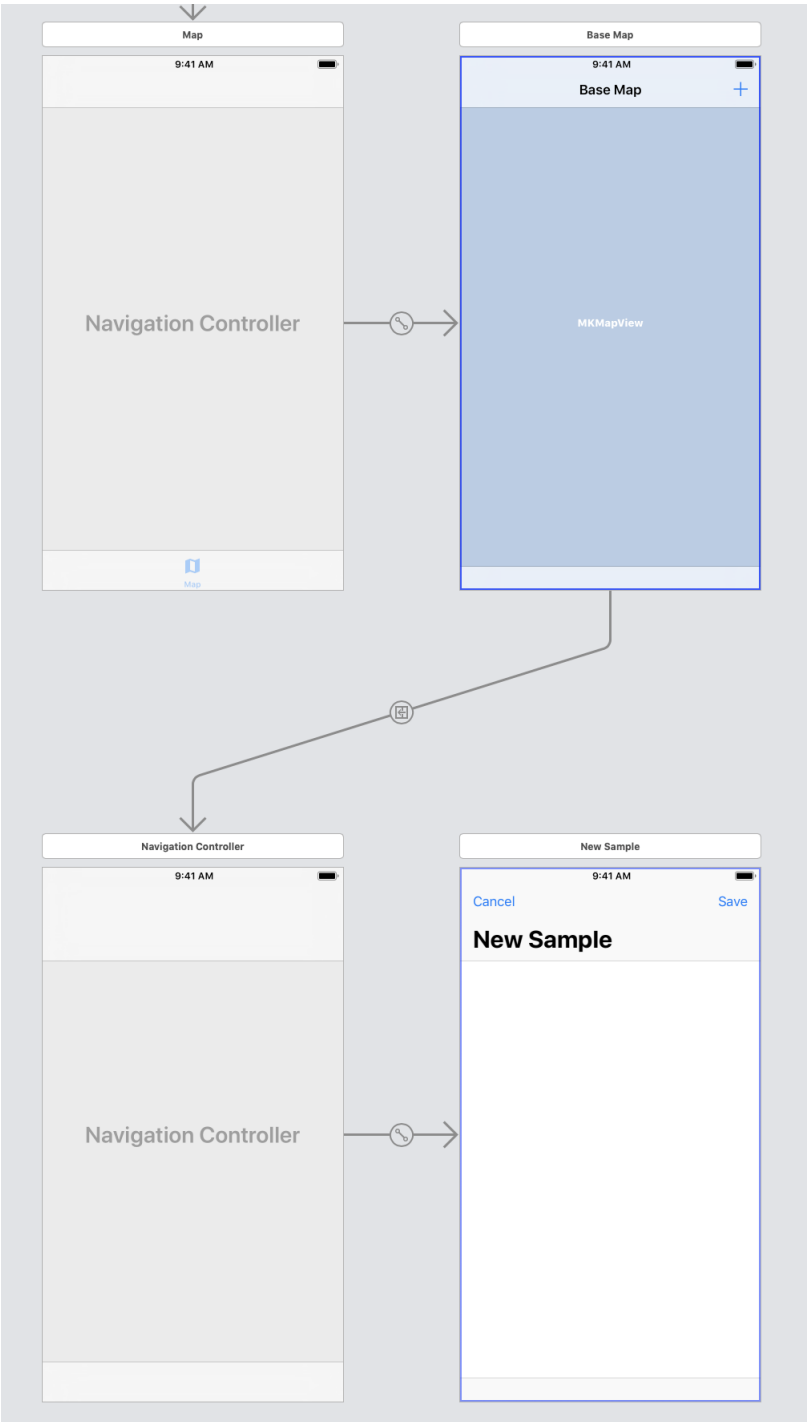
6 Swift storyboard



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.

6.1 Map Page

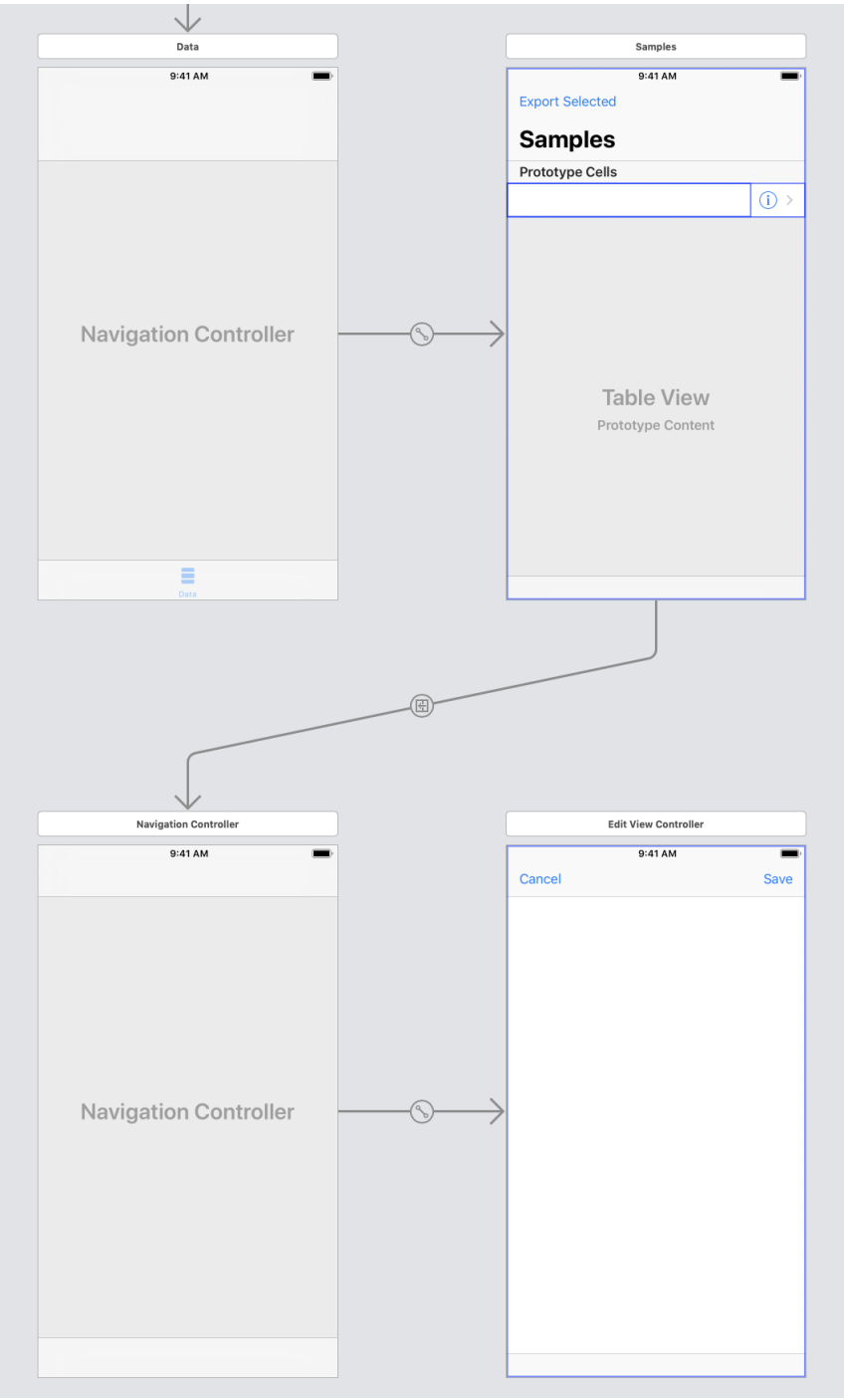


This diagram shows all pages in the **Map** page. The Navigation Controller at top-left handles the event that users press map button on the bottom bar or after users log-in. The MKMapView at top-right is a basic map provided by Apple; it allows users

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.

6.2 Data Page

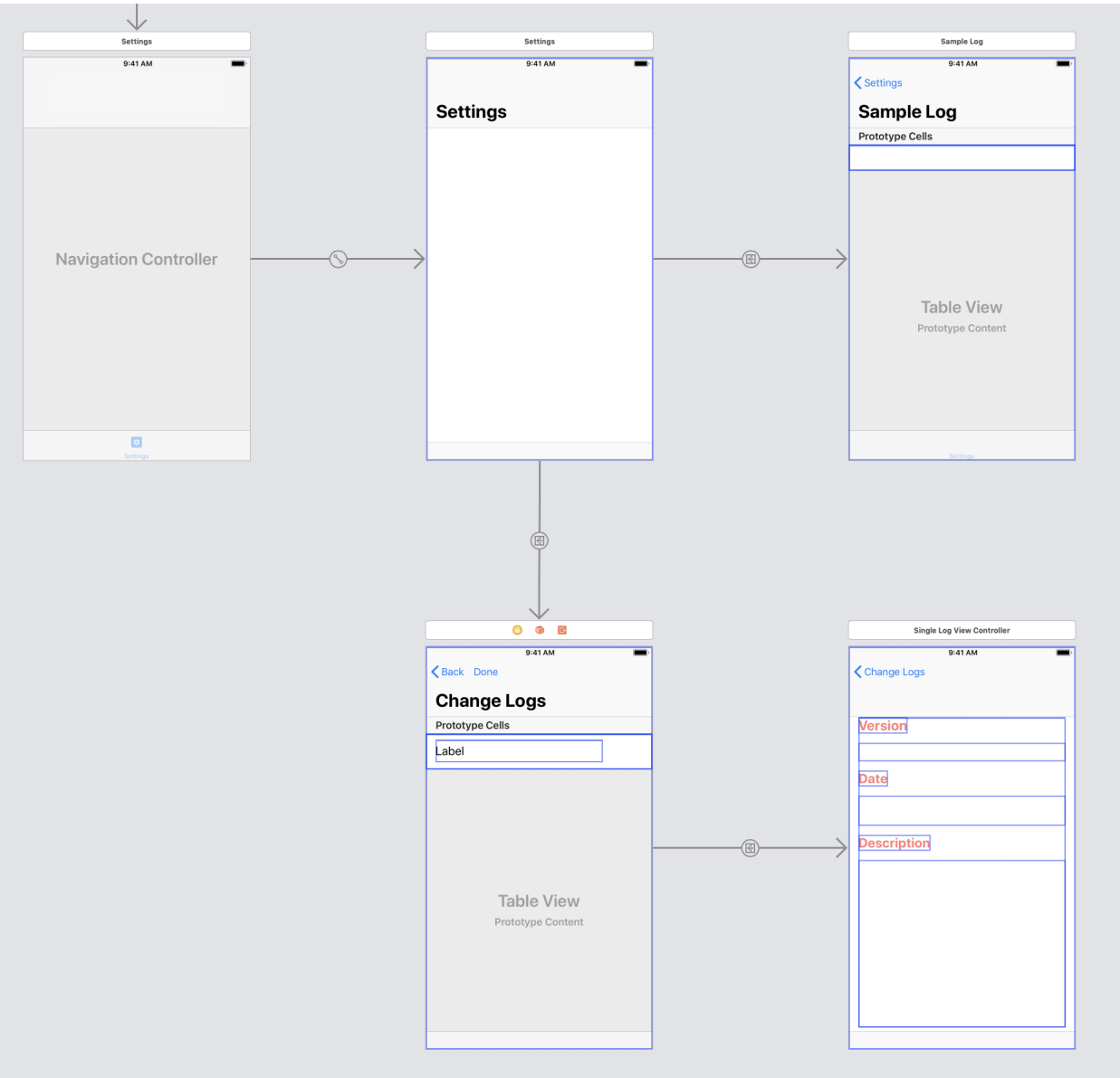


This diagram shows all pages in the **Data** page. The Navigation controller at top-left handles the event that users press data button on the bottom bar. The Table view at top-right is a table for showing all local data-points as a list. The **Export Selected** botton at top-left of the page includes 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.

6.3 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*.

References

1. Xcode from Apple. https://developer.apple.com/xcode/
2. Jazzy from Realm Inc. https://github.com/realm/jazzy
3. Eureka from XMARTLABS. https://eurekacommunity.github.io/
4. ZIPFoundation from Thomas Zoechling. https://github.com/weichsel/ZIPFoundation
5. CocoaPods from The CocoaPods Dev Team. https://cocoapods.org/
6. Apple Developer. *About App Development with UIKit,* https://developer.apple.com/documentation/uikit/about_app_development_with_uikit
7. Swift. *API Design Guideline,* https://swift.org/documentation/api-design-guidelines/