New documentation for iOS Lake Condition:

NTL Lake Condition on iOS is developed for the Center of Limnology to distribute its lake information collected by the buoys to the researchers and general public. It is funded by the NSF funded project “Next generation of buoy for Trout lake” led by Dr. Noah Lottig. The data is provided by the IT center in Center of Limnology, UW-Madison, where Dr. Corinna Gries as the data management director.

Lake Mendota, Trout Lake and Sparkling Lake have long been used for both academic research and public recreation, such as fishing and sailing. Center of Limnology, UW Madison has outpost on those lakes for collecting research data such as water and air temperature, water clarity through their buoys deployed on those lakes. There has already been a website built by the Center of Limnology, but due to its complexity, it’s hard to navigate to get essential by people through mobile. Thus, a phone app specific to this purpose can be useful to distribute those data to others quickly.

Currently, this app aims to help both researchers and general public to get the data just by glance. It only has a minimal user interaction, including swipe and tap. By the end of June, around 70 people have downloaded the app. I anticipate in the future, this app will keep this simple way of providing data for users, but get extended by adding other advanced function such as graph for user.

This specification is used to inform the developer who luckily works on this project of what has done in the past to facilitate his or her learning process. This specification will try to cover all of the important architect decisions made by previous developers, primarily myself. However, it’s also the first big software project I have made (when I start, I have only taken CS 302 and 367), so I might make some of the part really messy when I program. Thus, you need to prepare for some unexpected bug that maybe I don’t expect. Anyway, breaking the code can be a good start to make this program perfect.

We also have an Android app for the same purpose, which I will introduce in another specification. This iOS App is open source and developed under MIT license.

How do use the code:

1: Get a Mac, or a virtual machine for MacOS.

2: Install Xcode. The newest version is fine

2: Git clone this repository from Github

3: import the project NTLRealTimeConditioniOS.xcodeproj into the Xcode. You should be able to see the data

**I: Previous work:**

Basically, the whole iOS App is built from scratch by myself. However, one essential part in the data communication is built by another developer before, that is the web service that processes the data request by this app and gets the data from the limnology database. It’s essential to know that iOS App can’t directly connect to the database on the server, so it requires a webservice that this app can request data. This webservice, developed in JAVA, has been deployed on the limnology server. You will find the technical specification in another documentation.

The webAPI that returns the current lake information is: http://thalassa.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions

This lake information is communicated as JSON format, for more information: <http://www.w3schools.com/json/>

Lake Information Explanation:

Although Corinna will probably talk about it, I think it will be good for me to explain the data here, about the unit and about how I clean the data. The data is using the metric unit as default unit. I’m a Chinese so I’m quite used to that. I hope if you aren’t used to it, learn it! It makes pretty sense.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lake Info Name | Unit | Value Bound | Explanation | Comment |
| lakeId | N/A | N/A | The unique for the three lake | Only three strings are available right now: “ME” “SP””TR” |
| lakeName | N/A | N/A | The unique name for the three lakes | Only three strings are available right now: “Lake Mendota”, “Sparkling Lake”, “Trout Lake” |
| sampleDate | N/A | N/A | The timestamp when the sample is collected | Format in objc: @“yyyy-MM-dd’T’HH:mm:ss” |
| airTemp | Celecius | -10 ~ 50 | The air temperature on the lake | Corresponding Fararenhit Bound:  14 - 122 |
| windDir | Degree | 0 ~ 360 | The wind direction detected by the buyo at that location | I have a code inside to map each degree to a specific direction string |
| windSpeed | Meter per Second | 0 ~ 30 | The wind speed at that location | Corresponded to 0 ~ 67 miles per hour |
| waterTemp | Celecius | -5 ~ -50 | The current lake water temperate | Corresponding Faranrenhit Bound:  23 ~ 122 |
| thermoclineDepth | Meter | 0 ~ 50 | <https://en.wikipedia.org>  /wiki/Thermocline | Corresponding to 0 ~ 164 feet. Fisherman probably likes this data |
| SecchiEst | Meter | >-1 | Basically how deep people can see through the lake | Only apply to LAKE MENDOTA |
| SecchiEstTimeStamp | N/A | N/A | The timestamp when the secchi is collected, this data usually is not up to date, because it requires manual test. | Only apply to LAKE MENDOTA |
| PhycoMedian | I don’t know | IDK | IDK | Only apply to LAKE MENDOTA, and I use it for nothing |
| Wind Gust | Meter per Second | >windSpeed | It shows the sudden speed of the wind. | Only apply to LAKE MENDOTA right now. |

* Value Bound: that’s almost arbitrary as long as it makes sense. I uses this value bound only in this iOS App. Of course, the data can be null for some of the labels, so please make sure to check whether the data is null, or it might break the program. If the data is out of the bound, it will be stored as 999 in the database.

**II: Architecture design in a nutshell:**

* Language: Objective - C
* Platform: > iOS 7, iPhone, not on iPad.
* Development Tool: OS X (now called macOS), >iOS 7 SDK, Xcode 8 (This helps you to see whether the app works on iOS 10)
* Additional Dependency: Reachability.m (not a part of SDK, but Apple developed it), MessageUI.framework, SystemConfiguration.framework (both by Apple for reachability)
* Local Data Persistence: CoreData (a database framework developed by Apple)
* Code Style Guide: Read “Clean Code” by Robert C Martin (I know my code is far from clean)

**III: User Interaction Design:**

The user interaction includes two part: How people actually operates the app (User experience) and how people look at the app (User interface)

User Interface:

This main user interface has been modified by several times, and I hope if it’s possible, it’s possible to find a better way to redesign it. Before that, let me introduce some elements on the current user interface. The layout of the current user interface is in the “*JJWeatherViewController*.xib” file.

This app adopts a common way of fitting the interface to different screens, that is the “Auto Layout”(https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/). The layout for the interface is based on “Constraints”, a fixed distance between two UI element (such as “label” and “View”). Thus, even though the It can take a while to fully understand it (I probably don’t fully understand it), so practice it a lot.

As you probably know, the apple user interface is built in the small unit called “view”. On the base view, I have organized different views into 5 major parts. Those are The Lake Title, the temperature info, the wind info, the water info, the bottom tools. (from top to bottom).

The Lake Title has the lake name of this lake and the timestamp when this data sample is collected.

The temperature info, the wind info and the water info is basically design in a similar way. The left side is a picture that illustrates what’s this section is about (Temperature, Wind, Water). The right side is two blocks that show the actual data information. Each block is put in a view (defined in the iOS framework), and contains three smaller views, called “Prompt”, “Data”, “Unit”. In the actual view controller for this layout file, there are different fonts size for each small units.

In the bottom tools part, I have two main parts. The one with a lot of words on the above is used to show some current app information especially whether there is network connection or not. If the network works fine, there should be a line with small font size. If it’s not, there should be a red, big line “Network Connection Fail”. On the very bottom of this view, that’s the second part of the bottom tools. In that rectangular view, it has a “Set homepage” button, which allows the user to set the current lake page as the homepage, which by definition will be the first to show up every time the user starts the app. It also has a setting button that creates a menu of setting user can set. Right now, it supports changing the unit under which the lake information will be represented. User can also check the information about this app.

Be **cautious** about making any substantial change on any of the view. Right now, everything seems to be fine. However, due to the fragile nature of this system, it can be broken in the way you haven’t expected. Try to understand auto-layout system better before making change, and learn from actually break my view.

User Experience:

I’m that kind of person who really cares about how to use a fancy to provide a good user experience. Here, I will describe the using scheme that I choose to implement for this app. My priniciple is to make user spend less time in

Right now, there are only three lakes, which I guess is not that hard for users to remember. I think let users scroll between different lakes is a fast way to change from lake to lake without many steps. It’s like using the apple weather app in the phone. Due to the technical issue, I don’t know how to implement a scroll bar (You can see the scroll bar with lake names on Android App) on iOS, so I couldn’t implement one.

I guess that few people will need to change the lake really often because for example, if you are sailing for 3 to 4 hours on the Lake Mendota, you don’t have to know the lake information for Trout Lake. But it’s possible that you switch to another app and not use the lake app for a long time, which will lead to termination of the iOS App. Then after you reopen the app, you will see another lake and have to swipe to the lake you want. I think this isn’t the best solution for user experience because it can take extra steps to go to the place you want. Thus, I design a concept called “HomePage”, which stands for the page which will show up when the app is opened by users. In each lake information page, I include a “Set as homepage” button, which can set or unset this page as homepage. Thus, users can customize which information they want to see at a glance.

To keep the data most up-to-date (there is no reason for the sailor to get outdate data because the wind on the lake is really changeable), I implement the auto update for this app. That is, as long as the app is shown on the screen, it will update the data once a minute. For Lake Mendota, it can be really helpful because the information in the limnology gets updated every minute. However, for Sparkling Lake and Trout Lake, it might be a waste of data because the data gets updated every half an hour to an hour.

Another important feature is the change in different unit. Researchers actually prefer the SI unit, which includes meter, while general public is more used to British unit, such as feet and so on. I tried to make both of these two units with the corresponding data on the same screen, but since the screen size of iPhone is generally not large, thus the user interface can be quite crowded with two versions of data on it. Thus, I design that the users can change the unit in the setting menu.

For developer, I have an update view controller which will hold a place to present to user what feature has been added. That presentation will jump out every time the user just updates the data or the user first uses this app.

IV: Architecture Design:

Local Persistence Solution:

iOS Framework actually provides a great solution for saving different kind of data onto the local device. I use two techniques provided by the iOS Framework: **UserDefaults** for saving user preference and **CoreData** for storing the old lake data (This technique might be too complicated for such simple task).

NSUserDefaults is a class that helps keep track of the user’s setting even though the app gets closed. I use it right now to remember these important information: the unit user wants the data to display in, the app version user is using or has used, and whether this app is first used on this device. When you see my class called “*UserSetting*”, which wrapped the user default, you will more details about it.

Core Data is a nice, but not simple framework developed by Apple itself to store some data locally. It can actually achieve a lot if you understand how it works. I recommend to start by reading online tutorial and copy some codes for practice. In this application, the primarily implementation of the core data are in the *WeatherInfoDB* which regulates the reading, manipulation and storing of the data in CoreData.

!!!! Here are some important decisions of mine you definitely need to know: check the *weather*.*xdatamodel* file, in which I have specified which data fields the object “Weather” would need. Each data field name has the **same** name with one field I showed you above. This is essential for how class “*WeatherModifier*” works, or the whole program can go crash. I use dynamic assignment of properties in modifier to make the code easy for maintaining.

Network:

This app actually uses a very primitive way of getting data from the web service. Read the class *WeatherDataWebEntrance*, you will find that it just uses the NSURLSession and NSURLSessionDataTask to connect with web API. No other Framework (such as AFNetwork, which I never make work on my Mac) has been used.

Storyboard:

This is my first time developing an iOS App, so I’m not familiar every interesting framework Apple provides for the developer. Storyboard is one of them. I haven’t used it to develop the app.

**V: Brief Implementation Specification:**

After introducing all those high level concepts about what I want to achieve and what I use to achieve, I will introduce how I implement those ideas. In this section, a brief specification for different functions will be introduced to give you a good sense about which functions each class gets involved. A more detail implementation will not be articulated. It might be more efficient for you to read my in-line documentation to understand what I have done.

- RootViewController:

RootViewController is the highest level of controller in the whole program. It works other classes to execute multiple functions. UpdatePageViewController, LakePageViewController, WeatherModel, WeatherInfoDB are all initialized, and owned by RootViewController. It also implements the WeatherInfoDBDelegate protocal to decides what to do after the WeatherInfoDB updates the data. I made rootViewController almost the only owner of WeatherInfoDB, which means others can only make contact with db through it.

It also works with UserSetting to decide whether the update pages should pop up to inform users about the recent change

- UserSetting:

UserSetting is the only singleton I implement in the program. It can be accessed by any other component. Right now, three functions might involve this class: display data (by remembering user choice), homepage (by remembering user’s homepage) and updateView (by comparing the version of the app last time the user uses).

- Show lake data (Data refresh):

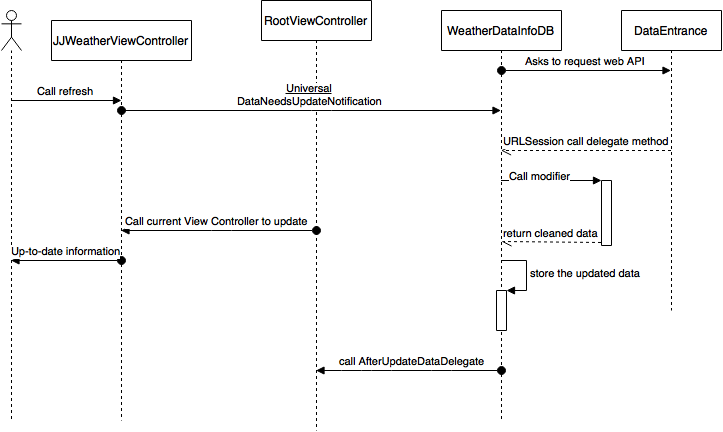
JJWeatherViewController: The controller that fills the label with data from database for its view based on user’s preference on unit. In JJWeatherViewController, there are two long methods to convert data into different units. It’s long, and probably you can find another way

WeatherDataInfoDB: Every time the app gets restarted (I mean after it’s) It receives a NSNotification called “DataNeedsUpdateNotification” to trigger the cycle of connecting web to get the latest data. It is also responsible for initializing the CoreData Database and keep the CoreData Database synchronized. It will then send message to its delegate when the data get updated.

WeatherDataWebEntrance: this class implements a protocal called WeatherDBDataEntranceProtocal which weatherDataInfoDB uses as an interface for getting data. In the WeatherDataWebEntrance, it has a URL Session which uses db as the delegate. After the URLSession receives the data, the WeatherInfoDB will automatically get the data through one of its delegate method. This WebEntrance also serves the function of checking whether the network is connected by using the Reachability written by Apple. It will then send out a “NetworkDidCheckNotication” every time the reachability has changed. Then RootViewController will catch the notification and then change the current view.

*WeatherModifier*: This class is a help class that assigns the data from the website to CoreData Model. Remember, it’s critical to set the field name in the model the name as the field in json object. Or a unexpected crash might occur.

RootViewController: It receives the message from WeatherDataInfoDB and then updates the current view controller.



* User change the unit the program displays in: Each *JJWeatherViewController* will invoke a *MenuViewController* when the user clicks on the setting button. The Menu View Controller will check the *UserSetting* for which unit user is currently using. The table cell for the currently used unit will have a check mark. If the user clicks on the unused unit, *UserSetting* will get updated and when the menu view controller gets dismissed, the menu will call the *JJWeatherViewController* to refresh the page.
* How the view controller works together to put the homepage onto the first page: I tried to manipulate the order when we got data from the CoreData. However, it doesn’t work out. It gets complicated if I want to specify the order. Thus, what I did is to manipulate data after three Weather object get fetched. By using UserSetting to store the first page id, I would manually exchange that id with the first index of the Weather objects. Good thing: It’s easy to change.

-Show update notification:

Update notification involves the collaboration among three classes: “*RootViewController*”, “*UserSetting*”, “*UpdateViewController*”. In the “*UserSetting*”, there exists a Boolean called justUpdated. If the current app version is not equal to the app version in user setting, justUpdated will be set YES. If the app is launched the first time on this device, it will also be set to YES. *RootViewController* will use this Bool in *UserSetting* to decide to launch the *UpdatePageViewController*, which has several “*UpdateViewController*” and one *letsStartViewController* as the end of the view controllers to leave the update view controller. *UpdateViewController* is a generic view controller with view that contains a picture and a place for paragraph. The text for the paragraph and image name will be specified in the *UpdateModel*, which also regulates the number of the update view controller.

These are primarily all I have got for the function,

**VI: Update:**

I think at some point, probably you would want to update the app. Thus, a brief introduction of how to update the app might be helpful for you to avoid some pitfall.

Before you update:

1: Make some substantial change on the app. Test it on the simulator provided by xcode to make sure that it doesn’t crush in some weird ways

2: Use an iPhone to test it, and also find others to use it before you are sure at least running feels good.

3: If you have major change in function or UI, add a screenshot and description to the Update model, which will help user to adapt your function or UI quickly next time they open the app

Update Essential:

1: Change the “Version” in the project into a bigger number

2: Change the number of “Bundle version string, short”in“Info.plist,”, Try to keep this number consistent with the version number in the first step

3: Contact Dave, “dave.schroeder@wisc.edu”, tell him that you want to update this app

4: Zip the whole project folder. Send it to Dave with some description about your feature and if necessarily, some screenshots so that Dave can update the description on App Store

5: Wait for Dave’s notification about whether the app gets approved by App Store.

After you update:

1: Download the app, and try to use it from the user perspective

**VII: Potential future update:**

1: I really want to add the feature that when people click on one prompt (probably a jargon to them), a small label will pop out to explains the jargon for the user.

2: I want to give people a more intuitive way of seeing the wind direction

3: Integrate the graph into this app.

End note:

I will be really happy if you can read this document and I would be happy to know that my app still works and gets updated☺ I hope maintaining this app doesn’t make you headache, and you can build awesome thing above it.