# Tshibuaya, Aimeric



## Chase Curriculum

| Date | Topic |
|---|---|
| Beginning - June 24th | Pick a code challenge, Swift Fundamentals |
| June 27th - July 1st | Design Patterns (MVC, MVVM & MVVM-C) |
| July 4th - July 8th | Hands on implementations. Applied patterns in real project and understand the differences. |
| July 11th - July 15th | Unit Test cases / Test Driven Development (TDD) |
| July 18th - July 22nd | Dependency Injection |
| July 25th - Aug 5th | Level up - code challenge |
| Aug 8th - Aug 12th | Tile Architecture |

## Universal Curriculum

| Level | Topics |
|---|---|
| Beginner | 1. Basis syntax<br>2. Application life cycle<br>3. ViewController life cycle<br>4. Storyboard vs Programming constraints<br>5. Navigation Stack<br>6. Delegation Pattern v.s. Local Notification Pattern<br>7. Get hands on with iOS UI elements (UITableView, UICollectionView, StackView, UIView, SplitView, Tab Bar, Picker, Slider, Switch... etc)<br>8. Get hands on with iOS SDK Frameworks 30+ (CoreLocation, CoreData, MapKit, MessageUI, PhotoUI, Network, Wallet...etc)<br>9. UI Animation / Transition Animation / Multi-Tasking<br>10. File Management |
| Intermediate | 1. Understand OOP(Object Oriented Programming) v.s POP(Protocol Oriented Programming)<br>2. Closure Pattern<br>3. Generic<br>4. Functional Programming<br>5. Push Notification (remote)<br>6. DispatchQueue (Multi-Threading)<br>7. Understand HTTP protocols, restful API (post, get, delete, put...etc), make your first service call<br>8. iOS Data Storage (NSUserDefault, Keychain, CoreData, iCloud) |
| Advance | 1. Reactive Programming (Publisher, Subscriber, Subject ...etc)<br>2. Understand TDD (Test driven development)<br>3. Understand Design Patterns (MVC, MVVM, MVVM-C, MVP, VIPER)<br>4. External / Remote Data Storage (GWS, Firebase, Realm ... etc)<br>5. Authentication (Login)<br>6. More... |

| | |
|---|---|
| Plus+ | 1. Algorithms<br>2. Data Structure<br>3. System Design |

# Activity Logs

| Date | Activities |
|---|---|
| July 12, 2022 | • Go through Aimeric's project and review the progress of his final MVVM-C refactoring.<br>• Modify Aimeric's project to demonstrat the expected MVVM-C refactoring results<br>• Upload and share the sample project with Aimeric. Sample Project: NYC High School with MVVM-C |
| July 1-8, 2022 | • Go through Aimeric's project and review the progress of MVVM-C refactoring. |
| June 28, 2022 | • In person Demo with Geff, Arundathi, Franqueli and other dev leads. |
| June 27, 2022 | • Review Aimeric's code refactoring progress and prepare for demo. |
| June 23, 2022 | • Resolve a git issue for Aimeric to push his commits. |
| June 21, 2022 | • Schedule a meeting for demo with Aimeric's first MVP.<br>• Follow up with Aimeric on the progress of refactoring.<br>• Start creating documents, materials for mThree devs to go thru. |
| June 17, 2022 | • Let Aimeric showcase his NYC School challenge project. (To understand his knowledge base)<br>• Review code with Aimeric and provide some feedbacks for him to dive in, improve and refactor.<br>• Upload Aimeric's NYC School challenge to his Github. |
| June 13, 2022 | • Setup a daily meeting with Aimeric to walk thru any questions he has. |
| June 09, 2022 | • Work with Aimeric to decide what code challenge he will take for training. (NYC School) |

# Code Review Logs

## June 27, 2022

| File Name | Feedbacks |
|---|---|
| SchoolListViewModel | 1. optional still needs to be applied |
| SchoolListViewController | 1. Indicator is not showing properly |

## June 17, 2022

| File Name | Feedbacks |
|---|---|

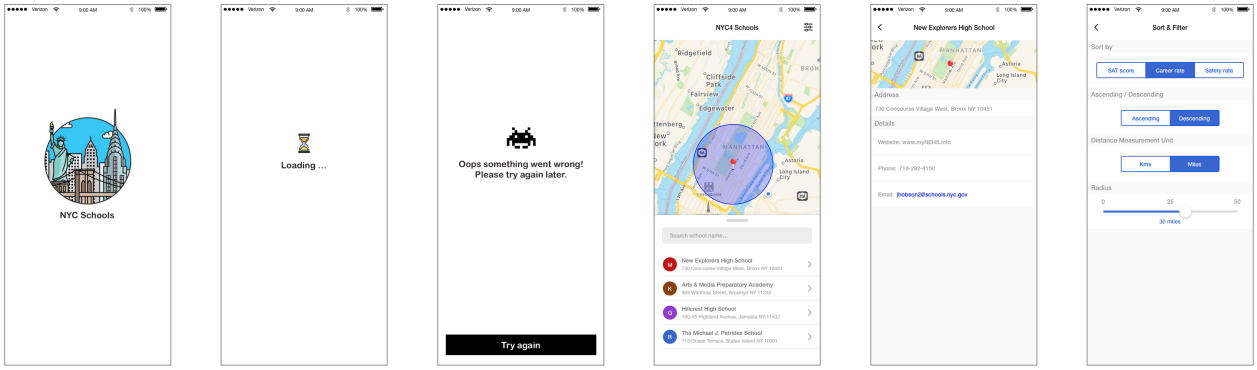
| | |
|---|---|
| SchoolList ViewModel | 1. Structs are preferable for ViewModels if they are copiable and mutation is not required because copying is way safer than having multiple references to the same instance as happens with classes. This is especially important when passing around a variable to many classes and/or in a multithreaded environment.<br>2. Each ViewModel should have its' corresponding protocol.<br>3. Access Control: https://docs.swift.org/swift-book/LanguageGuide/AccessControl.html<br>a. **Open** access and **public** access enable entities to be used within any source file from their defining module, and also in a source file from another module that imports the defining module. You typically use open or public access when specifying the public interface to a framework. The difference between open and public access is described below.<br>b. **Internal** access enables entities to be used within any source file from their defining module, but not in any source file outside of that module. You typically use internal access when defining an app's or a framework's internal structure.<br>c. **File-private** access restricts the use of an entity to its own defining source file. Use file-private access to hide the implementation details of a specific piece of functionality when those details are used within an entire file.<br>d. **Private** access restricts the use of an entity to the enclosing declaration, and to extensions of that declaration that are in the same file. Use private access to hide the implementation details of a specific piece of functionality when those details are used only within a single declaration.<br>4. Swift Coding Style Guide: Please see the References section in this Confluence page.<br>5. Guard is preferable than If let in most cases to reduce the closure & brackets hell.<br>6. Understand the Optional. (What is the difference between exclamation mark and question mark?)<br>7. Exit early concept |
| SchoolList ViewController | 1. Access Control<br>2. Update If let with guard.<br>3. Business logics should be moved to ViewModel.<br>4. When define a ViewModel in a ViewController, it should be defined with ViewModel's protocol instead of the ViewModel's type.<br>5. We normally don't use `UIAlertController` for loading. In general, the loading indicator will be displayed with `UIActivityIndicatorView`. Feel free to try it out.<br>6. Move the Utility functions / Common used extensions to separated files. |
| SchoolList DetailViewController | 1. Access Control<br>2. Each ViewController should have its own ViewModel |
| SchoolsDataManager | 1. How to make a service response more generic? Research and we can talk about it. |

# Sample Project

## NYC High School with MVVM-C


# Sample Design

## Overview




## Design Assets


# References

## Style Guide

### List of Guides

There are some well known style guides out on the Internet. Please see below for details.

- Raywenderlich - Swift Style Guide
- LinkedIn - Swift Style Guide
- Airbnb - Swift Style Guide
- Google - Swift Style Guide
- Swift.org - Swift Style Guide

## Tools

We are all humans and deliberately or not, we all make mistakes. Thus, it is not enough to document or verbally agree on *Swift coding conventions*. To minimized the number of mistakes, the rules must be enforced by an automated tool, known as *linter*.

- Linter SwiftLint