CODE QUALITY REPORT

Serdachny Hockey iOS Mobile App

Abstract

This document provides an overview to application code structures and pattern review referring to The fundamental Code Review Checklist reference

Prime Software

University of Regina

2020

VERSION HISTORY

Version	Description of Change	Author	Date
1	Document Creation	*McKenzie Busenius*	01/26/2020
2	Code Revision, adding functional comments of swift classes, functions	*Jiahao Li*	02/18/2020
2.1	Document complete, Code Revision, writing more code flow details for each file	*Jiahao Li*	04/02/2020

1. Development Environment

Our capstone project named Serdachny Hockey IOS mobile app is built in MacOS with XCode as our development tool with Swift 5 programming language. Our team took advantage of many organizational advantages within XCode. For example, XCode provides opportunities of files structures layers navigation as shown as **Figure 1**, development screen color themes, code character colors and intuitive tool icons. Swift 5 language provides great powerful libraries and good defined naming conventions of functions.

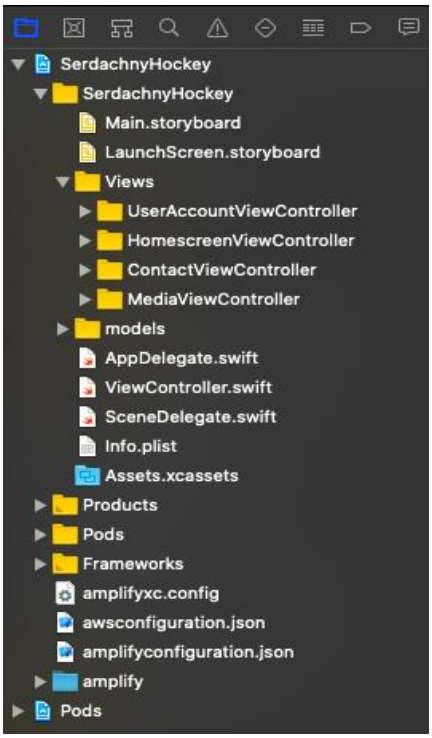


Figure 1. *XCode File Navigation Order*

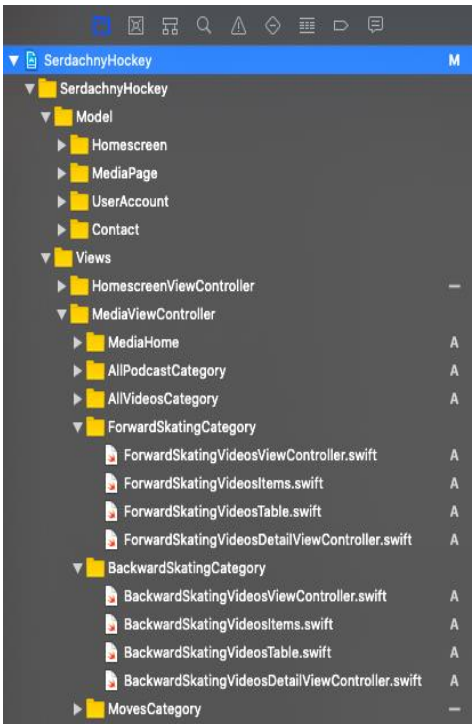


Figure 2. *Source Files and*

2. Code Formatting

It's important that our group focused on a well-developed application that f followed proper code formatting that should be done during or after building program code. As the programs get more complex, they get harder and harder to understand by others and even the personal who wrote the codes cannot understand and be able to re-read it.

Good layout has been proven time after time to make code more readable and the time it takes to do it nearly always repays itself many times over in the future. After reviewing all codes from the app, we ensured that each class file name has meaningful or

intuitive meaning because we are developing a marketing video hockey app that requires clear categorized sources file names. Therefore, my teammates are capable to modify, and even future developers can easily pick up, read and write. We have created appropriate folder location that contains related source files with proper ordering. For example, as shown as above **Figure 2**, based on the Media Page, Views navigation part , there are *ForwardSkatingCategory*, *BackwardSkatingCategory* folders as an example. Each folder represents one type of hockey skills. Expending one of them, we clearly see that it has consistent and relatable names and order. Inside *ForwardSkatingCategory* folder, there are *ForwardSkatingVideosViewController.swift* that is a controller function to display all contents in the screen, *ForwardSkatingVideosItems.swift* that is an instance class to initialize all variables/objects, *ForwardSkatingVideosTable.swift* that is a helper function to pass and update current video's variables being modified and lastly *ForwardSkatingVideosDetailViewController.swift* that is a sub controller function to pass data to anther screen for showing a video details when user taps one of video contents.

3. Architecture

MVC pattern is good at handling model, view, controller patterns during the development. There is also a strong reason to use MVC that we can have total separation in programming logic and the interface code. Therefore, it provides an effective way to design, implement, add, and debug/test functionalities. In this app, we are using MVC design pattern. The Serdachny app is all about video-based with marketing services that means it requires a lot of demands on querying, displaying processes. In the Model state, our backend configuration files which are connected to AWS S3 Bucket server side are in the model folder. In the View state, it was designed to contains storyboard where contains UI designs and have the parallel relationship with Controller state that contains manipulation files. Our team exactly followed all the way of MVC pattern to build, modify and test codes from the beginning to the end.

4. Best Coding Practices

The best coding practices is essential for developers or companies with the respect to benefits such that efficiency and productivity workflow can improve teams overall output and will result in costs reduction. When we reviewed all codes, we focused on questions regarding best coding practices such as how do we write appropriate proper comments about what we are doing with the code block? What is the complexity of logical blocks? Whether we use framework features instead of writing custom code? All these were design considerations. For instance, as shown as screenshots of part of codes below, we wrote engineering practical comment on each of the top sections for each telling what is expected functionality as wells as providing a team name which is shown as **Figure 3**. With the respect to framework modular, we used built-in functions via import Foundation or import *UIKit* from library. In the **Figure 4**, all *func tableview(…)* functions are required to deal with interaction between View state and Controller state, in

other words, way of storing and displaying contents. And the benefit is we can view these functions' definition and usage within the Xcode Swift 5 library or research online IOS Swift 5 reference manual.

```
/*

  Program Name: ForwardSkatingVideosViewController.swift
  Application Name: SerdachnyHockey
  Author: Jiahao Li

  Description:
              1. The purpose of this program is to store video data into array and display
                  into TableView
              2. Pass table cell data to the detail page
              3. Search Filter Function

  Copyright © 2020 Prime Software. All rights reserved.

*/
```

Figure 3. Top File Comments

```
extension ForwardSkatingVideosViewController: UITableViewDataSource, UITableViewDelegate {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if isFiltering() {
            return filteredForwardSkatingVideos.count
        }
        return forwardSkatingVideosItems.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
        UITableViewCell {

        let forwardSkatingVideosItem = forwardSkatingVideosItems[indexPath.row]
        let forwardSkatingVideosCell = tableView.dequeueReusableCell(withIdentifier:
            "forwardSkatingVideosCell") as! ForwardSkatingVideosTable
        let currentForwardSkatingVideo: ForwardSkatingVideosItems

        if isFiltering() {
            currentForwardSkatingVideo = filteredForwardSkatingVideos[indexPath.row]
        } else {
            currentForwardSkatingVideo = forwardSkatingVideosItem
        }

        forwardSkatingVideosCell.setForwardSkatingVideos(video: currentForwardSkatingVideo)

        return forwardSkatingVideosCell
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        let video = forwardSkatingVideosItems[indexPath.row]
        performSegue(withIdentifier: "forwardSkatingDetail", sender: video)
    }
}
```

Figure 4. func tableview() structure

5. Non Functional Requirements

1) Maintainability

Our team has three members and each one has own individual coding parts. Afterwards, we needed to do integration such that combining and testing features in regular intervals. So that requires code's readability, testability from anyone. It would save time to understand codes. From the code review perspective, we ensured that we used appropriate names for variables, functions and classes. Moreover, the code can be easily tested with existing refactoring classes.

2) Reusability

"Do not repeat yourself". We were trying to eliminate duplicate logic codes(classes, functions) via abstraction method in separate location and call it logically if necessary. It also can be reused when it is called. Recalling layout in our app, each video category folder contains a source file that initializes Swift UI components such as *UIImage*, *UILabel*, *UIText*.

3) Reliability

Based on the User Account page, we used AWS S3 Bucket authentication credentials for login, signup. We designed exception handling for each case such as login failure, signup failure and user invalid input. The purpose is to save user from failures.

4) Extensibility

We used abstraction method to reduce complexity of logical code as much as we can. It is easy to make minimal changes to existing code without breaking codes.

5) Security

Since we applied AWS S3 bucket authentication credentials method to process login, signup. The backend will handle/provide user data security, input data validation etc.

6) Scalability

The scalability standard of application is critical to deployment such that it can handle numbers processes of requesting or querying user data. However, based on our app testing, we just had added small sample contents/data because we realized that it depends on whether you purchase relative serveries from AWS bucket if you want to handle a large data.

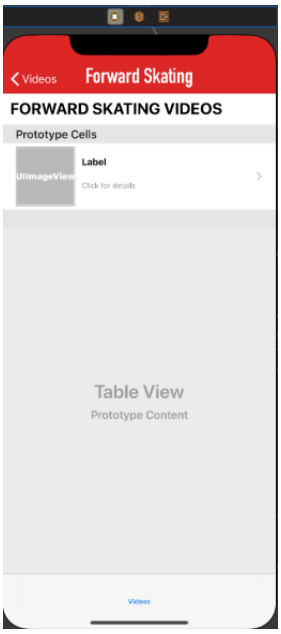
7) Usability

As we started on user interface design, there were Swift 5 language UI components design documentations that we could refer to. And we made a nice design layout for each part. The user interface should easily to be understood and used.

6. Object-Oriented Analysis and Design (OOAD) Principles

During app development, our team started with carefulness of writing simple and clean (single responsibility principle) for each function. As more logical things being added, we had tried to reduce complexity within a function (separation if necessary). With respect to open closed principle, new features can be easily added without modifying existing code. For instance, as shown as below, on the left, this is the

ForwardSkatingCategory of Media Page UI design in XCode. There is a TableView that contains Image View, Title Label, Description Label. If I want to add another feature to display what is the date of content creation like Date Label. I need to find and add another label icon from library. The last thing is to add another instance of Date Label into the class – *ForwardSkatingCategory().* Also it is also convenient for code debugging.




```swift
// initialize variables
class ForwardSkatingVideosItems {

    var image: UIImage
    var title: String
    var description: String
    var url: String

    init(image: UIImage, title: String, description: String, url: String) {
        self.image = image
        self.title = title
        self.description = description
        self.url = url
    }

}
```

Left **Right**

7. Tools for Code Reviews

Currently our team still check/review codes manually but we are planning to use kind of technologies/tools in the feature version(future MVPs).

8. Reference

Code Review Checklist. https://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews/