Spring 2018 iOS Training Program
iOS Mini Project 5

Overview

In this project you will be building on top of MDBSocials. You will refactor & improve your existing codebase with more industry best practices. You will also add additional features (all pertaining to notifications). Lastly, you will add some finishing touches that are important for any app that you build.

There are very few features/refactors in this project, but some of them are quite difficult for beginners to implement. This project will be time consuming so start early.

Once all Mini Project 5s have been submitted, we will evaluate who has the best project based on the following 3 criteria:

1. Usefulness
2. Design
3. Quality - Robustness, Speed, Bug Free, Code Quality

The best project will then be launched to either TestFlight or the AppStore - and we will get the whole club to use it!

Concepts Emphasized

Here are the new concepts emphasized in this week's project:

- NotificationCenter
- How to Make a REST API
- Push Notifications
- Logging

Useful Tips

We encourage you to proceed in the order of the deliverables are listed in this spec (i.e. complete Part I before Part II). Push Notifications can be tough to test & debug, so leave plenty of time for that. The REST API you have to build is very basic, so do not be intimidated by it. Look at the sample code that was posted from the demo. You should be able to copy a lot of it.

We also encourage you to complete all of this week's readings before beginning the project. They will help solidify your understanding of these topics and help with this project.

Note: Along with all of the functionality specified in this project, we expect your project to include all of the functionality specified in Mini-project 3 & 4. If you did not finish or had bugs in your Mini-project 3 & 4, fix those while working on this project.

Part I - Refactor & Codebase Improvements

SwiftyBeaver - Integrate the SwiftyBeaver cocoapod into your project. Look at the documentation and log wherever appropriate in your project. Logging is an important part of building any industry level codebase because it helps you easily track the user's actions, print the server's responses, and debug any issues that may arise. At the very least, you should log every response you receive from a network call, log all errors, and log when the user signs in or signs out.

NotificationCenter - This part is very open ended. We did an activity during the hackshop where we compared the usage of NotificationCenter & delegation. An important part of being a software engineer is figuring out which mechanisms to use when. That's why we're leaving this up to you. You have to find 1 part in your codebase to use NotificationCenter. If you're confused about when to use NotificationCenter, reference the articles in the Week 5 Resources & Readings.

Part II - REST API

For this part of the project, you will be making a REST API using NodeJS and deploying it to Heroku. Fear not, this is actually really easy. If you didn't catch everything that was going on during the live demo at the hackshop, just search on YouTube or Udemy for how to videos. There's plenty of great content already out there. For an overview of the steps involved in setting up your RESTAPI, please reference the ReadMe from the hackshop demo: https://github.com/MobileDevelopersOfBerkeley/meme-api. You are also encouraged to download Postman. It's a great tool to test the REST APIs you build.

Here's what you have to do. Take all of the functions in your FirebaseDatabaseClient.swift file and write that logic on your REST API. Then delete the FirebaseDatabaseClient.swift file in your iOS project and create a new file called RESTAPIClient.swift. In RESTAPIClient.swift, use Alamofire to call methods from your REST API to accomplish the exact same functionality you had earlier (ie retrieving posts, creating posts, retrieving a user, etc). And yes, the functions in your RESTAPIClient should be utilizing PromiseKit.

Part III - Push Notifications

This part will be the only new "feature" you are adding on to MDBSocials in this project. The goal here is to send a notification to all users when a new social is posted. The notification will be of the form: "<User> posted a new social called <Social Name>. It's happening this <Day> at <Time>!".

You will first want to read the “Introduction” & “About FCM Messages” sections of the FCM documentation: https://firebase.google.com/docs/cloud-messaging/. Once you have a basic understanding of the service, you’re ready to get started.

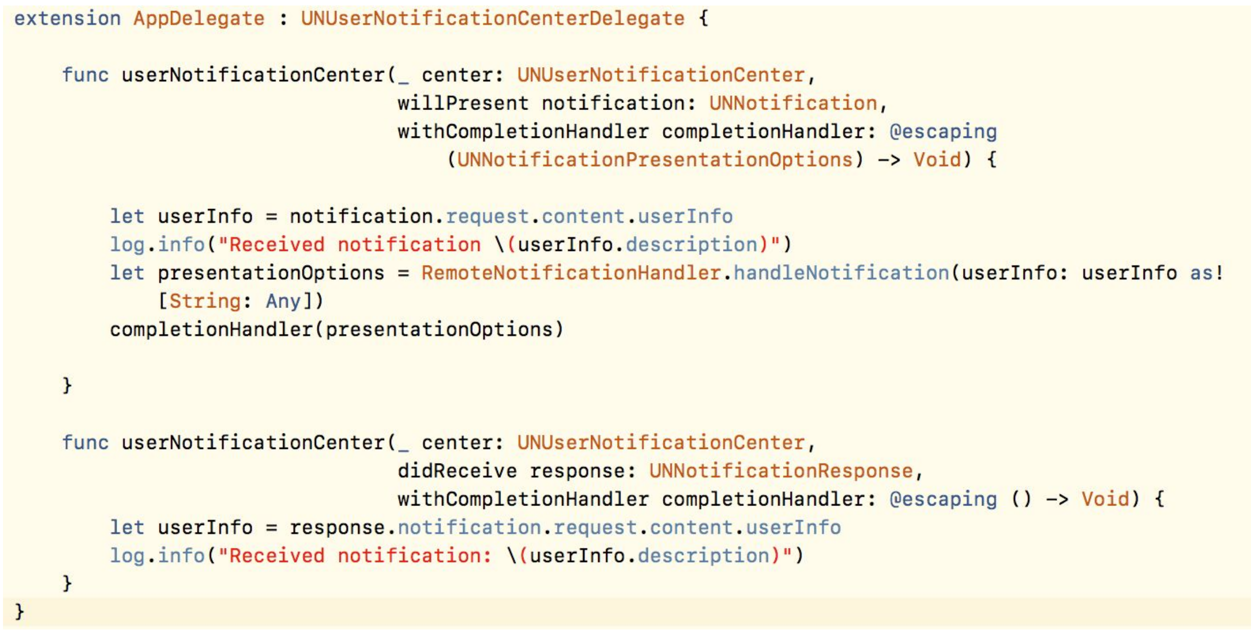
Follow the tutorials in the iOS part of the FCM documentation to:

1. Setup your iOS client with FCM
2. Configure APNS with FCM (you will need to get added to the MDB Apple Developer account for this)
3. Subscribe your user to an FCM topic named “NewSocial” when the user signs up on the app.

Next, follow the directions in the Admin part of the FCM documentation to:

1. Setup your REST API with FCM
2. Create a function that sends a notification to all users subscribed to the “NewSocial” topic.
3. Call the new function you just created from the function in your REST API that you use to create a new post.

Lastly, implement the appropriate delegate methods in your AppDelegate to handle the receipt of these notifications. Here’s an example from the hackshop:



```swift
extension AppDelegate : UNUserNotificationCenterDelegate {

    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                willPresent notification: UNNotification,
                                withCompletionHandler completionHandler: @escaping
                                    (UNNotificationPresentationOptions) -> Void) {

        let userInfo = notification.request.content.userInfo
        log.info("Received notification \(userInfo.description)")
        let presentationOptions = RemoteNotificationHandler.handleNotification(userInfo: userInfo as!
            [String: Any])
        completionHandler(presentationOptions)

    }

    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                didReceive response: UNNotificationResponse,
                                withCompletionHandler completionHandler: @escaping () -> Void) {
        let userInfo = response.notification.request.content.userInfo
        log.info("Received notification: \(userInfo.description)")
    }
}
```

To test part 3, do the following. When you create a new post, a notification should be sent to all users informing them that a new social is coming up. Yes, it should even send a notification to the user who posted it. This is just for the sake of simplicity. If you want to go the extra mile, you can write code to omit the poster from the list of users that receives the notification. But most likely, you will not have enough time to do so (so don’t worry about it).

Part IV - Polishing

Try out your app on a physical device to make sure that the UX is good. Implement any finishing touches you need to.

One example of some polishing you should do: If some of the text fields on your signup page get covered when the keyboard pops up, you should shift up the whole screen. Tapping anywhere on the screen outside of text fields should dismiss the keyboard. There’s really simple code on StackOverflow that you can plug and chug for small things like this.

Have some of your peers try out your app, and if they face any UX issues similar to the one described, go ahead and fix them!

Submission

Please create a new repository for this. Do not just use the same repo from project 3 or 4. You will submit this link in the google form we usually use:
https://goo.gl/forms/1TMFCwLokuG8aWV32.