ECS189E Homework 2

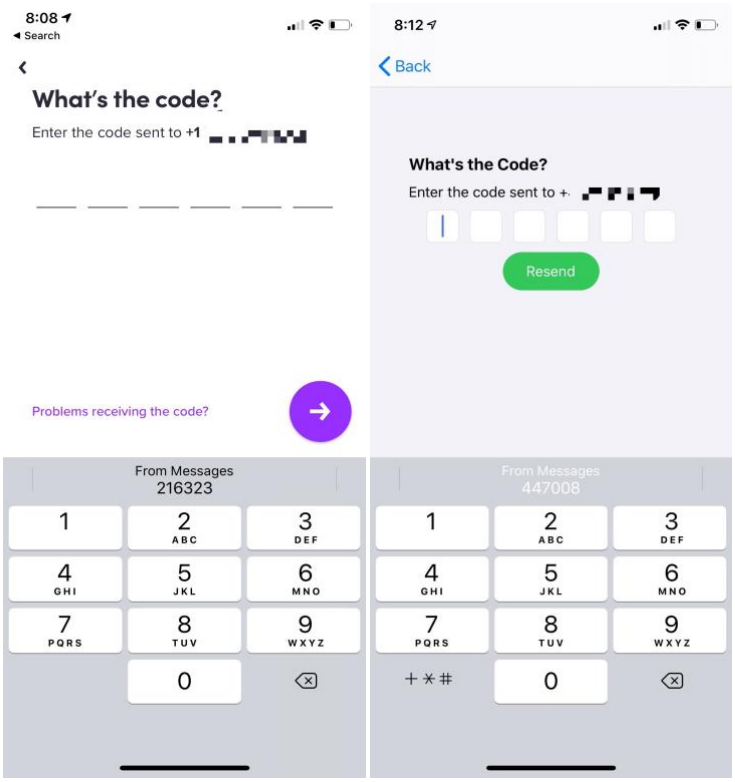
Overview

In the homework 2, you will complete the login part. Following the homework 1, your App will send out the verification code to the phone number that user entered. After that, you need to instruct users to enter the verification code that they received; verify if the code is correct.

Usage of Provide Code

Clone the homework 2 repository to your local device. Move or copy your code of homework 1 to this directory. Open your moved/copied project in XCode as you did when working on it for the last Homework. Drag the Api.swift that was in the base of the repository to be in the same folder as your other .swift files (ViewController.swift, SceneDelegate.swift, etc). You should be prompted with a popup about creating references, to your project, assess that it will be placed in the right place then press finish. The Api will now successfully be added to your project, you simply need to follow steps later in this PDF in order to use it in your code.

View of Verifying Code



The first image shows how the Lyft instruct users to enter the verification code. And the second image shows a previous project's example app.

For this homework, **You have to use a separated view for verification.** The purpose of this requirement is for you to have some experience of using Navigation Controller. For this verification view, just like the first view, you are required to include:

1. an instruction,
2. an OTP style text field,
3. a resend button that will resend the verification code,
4. and a label to present verification error.

Make sure you use a separate swift file for the new ViewController and give it a reasonable name (ex: VerificationViewController).

More about Verification

1. **It is required to** make the text field in the OTP style. In other words, six separate boxes or positions each containing one digit. **You'll do this by using six different text fields**. I know there are online tutorials that tells you how to do it with only one text field and several labels.

   Don't follow that tutorial because you'll need to know protocols and delegates for that method.

   - While the user is entering the code, the cursor should automatically go to next text field. If the code, after the 6th value, does not enter correctly, then all boxes should clear and the "cursor" should return to box one. You will probably notice that if you are mid-typing code, the backspace will not work correctly. Don't worry about this, you won't be able to effectively fix this until we learn about delegates. We aren't looking for "hacky" solutions, so stick with having the boxes all clear and restart if you enter the wrong code fully.
   - The user shouldn't be able to interact with any text fields to change which one is editing. If they just typed in the first box, the cursor should automatically move to the second box and so forth.
   - Once again, if you reach the end of the code and it is wrong, then all fields should be deleted and the user should be set to restart typing.
   - Hint: Life gets a lot easier if you keep an index count of which OTP UITextField you should be working in, and append all of your UITextFields to a list to iterate through such as: `var fields: [UITextField?] = []`

2. **Your app is also required to** auto verify the code once the user has typed in all the code. In other words, there shouldn't be any next button needed.
   - Present the **Home View** if the verification code is correct. You don't need to have anything in the Home View. An empty view controller is fine.
   - Show the error message if the verification code is incorrect, and restart to the first box.
3. The resend button should function as you expected: send the verification code again to the phone number you entered in your Login View.
4. Lastly, **your app should** only present the phone pad and **should have** the autofill functionality.
   - Hint: Check out the content type of the text field.

Navigation

Since we are dealing with multiple ViewControllers from now on, you need to use the NavigationController and "present" for your App. For this homework, the controllers to consider are the initial ViewController from the last homework (let's call it LoginView), the new VerificationView containing a one time code, and an empty HomeView that they enter if the code was successful. Keep in mind these are just our names for them that we will refer to in this document, they can be called whatever reasonable name you would like in your own homework.

Navigation Controller

You'll use the Navigation Controller to switch from the LoginView to the VerificationView, and later the VerificationView to the HomeView. Simply add the Navigation Controller as a UI Component, make it point to your initial LoginView, and then transition appropriately through the other View Controllers using code.

Present

You'll use present to transition from the Verification View to the the Home View. In other words, there shouldn't be a back button on the top left so that the user can go back to the Verification View. **Hiding the navigation bar or remove the back button on the navigation bar is not the same as "present". Make sure to find a way so that whatever HomeView that you choose cannot return to the VericationView.**

To clarify the view flow should be as follows once the app is up and running fully: The user should open the app to the **LoginView** where they enter their phone number. Then they should be shown the **VerificationView** to enter their one-time code, which, can be exited back to the **LoginView** if they choose. If they get the code correct, they are sent to the empty **HomeView**, and are unable to return to any other views at this time.

Send Verification Code and Verify Code

For this section, you will use the provided Api to send the verification code to the entered phone number, and verify the entered code. There is also a short note provided as instructions at the beginning of the file.

Usage

```
// Send Verification Code
Api.sendVerificationCode(phoneNumber: YourPhoneNumber) { response, error in
    // Handle the response and error here
}

// Verify Code
Api.verifyCode(phoneNumber: YourPhoneNumber, code: YourCode) { response, error in
  // Handle the response and error here
}
```

Test Usage

Since we have experienced issues with rate limits in the past when texting phones with this API (essentially meaning we can only send your phone a limited number of texts before it locks you out for 24 hours), we have also included two functions **Api.testSendVerificationCode(…)** and **Api.testVerifyCode(…)** that you can use in place

of the above functions in the early stages of testing your code. **Api.testSendVerificationCode(…)** will remember the phone number set in the **LoginView**, and also set a fake “verification code” of “123456”. You, as the tester developing this app, will then pretend you got texted that code, and move forward with a VerificationView where the user can enter “123456” into your OTP to get an automatic success.

The point of this is, once again, to limit the number of texts with real verification codes sent during the testing stages. Once you have a successful flow with **Api.testSendVerificationCode** and **Api.testVerifyCode**, you should proceed by switching to **Api.sendVerificationCode** and **Api.verifyCode** moving forward, in order to test the auto-fill and actual verification capacity of your app. This, hopefully, will provide a minimal change.

Possible Callbacks

In the case of success (verification code sent or code verified), the response should not be nil while the error should be nil. For the opposite (cannot sent the verification code cannot be verified), the response should be nil while the error should not be.

For this homework, you do not need to handle the response, however you will need to print out if there any errors.

Error

There are two class variables that you can use: `error.code` and `error.message` .

code	message
"invalid_phone_number"	"Your phone number is invalid"
"incorrect_code"	"Incorrect verification code"
"code_expired"	"Your code expired"

You do not have to use the message provided. You can also test for code and customized your message.

Coding Style

It is always very important to make sure that your code is easy to read and understand. The following points will be considered when your coding style is graded:

1. Reasonable names for ViewControllers, views and objects.
2. Reasonable names for files, functions and variables.
3. Necessary comments.
4. Reasonable order of the functions.
5. The use of optional (Question mark and exclamation mark).

Documentation

Write a README.md to briefly state what functions does the App have and how did you implement them. Imagine yourself working in a big company, and this document is for the people who will take over your work, or will become your partner and work on this App with you.

Remember to include your name and student ID in the document.

Grading

1. UI Design and Auto Layout (5')
2. Navigation (15')
    - There should be 3 View Controllers (LoginView, VerificationView, HomeView)
    - Users should be able to go between LoginView and VerificationView, but not move from the HomeView to any other View once they enter.
3. OTP Style Text Field and Resend Button Functionality (15')
    - OTP should be able to both manually enter the code **and** auto-complete the code by texting a phone number.
    - If an incorrect code is entered, all fields should empty and the “cursor” should restart at the first box.
4. Send and Verify One Time Code (10')
    - Only valid US Phone Numbers should move on to the VerificationView, and only entering a correct code should move onto the HomeView
    - Any incorrect values should have proper error displays as outlined in this document.
5. Coding Style and Documentation (5’)

Submission

Push your files to the repository and submit a link to that repository on Canvas by the due date.