E4C iOS App – Front-End Documentation

Running the code

1. Download the code from github: https://github.com/sc96/e4cApp
2. Download xCode https://developer.apple.com/xcode/
3. Download cocoapods https://cocoapods.org/
4. Open the terminal and CD into the e4cApp directory
5. Install the pod dependencies by typing in 'pod install' in terminal
6. Open 'e4cApp.xcworkspace' with xCode
7. Run the simulator. For best viewing use "Iphone SE"

Possible Errors:

1. Make sure xCode and cocoapods are up to date
2. Pod install should download 3 dependencies
3. Being able to run the simulator, but getting error connections means problems on the back-end, e.g, the server.

App Navigation

The RootViewController of the app is a TabBarViewController. The TabBarViewController has 4 tabs, each of the tab is a NavigationController. Each of the NavigationControllers has a rootViewController, specifically HomeViewController, NewsViewController, WebinarViewController, and the fourth one is either WelcomeViewController (if you're not logged in) or CommunityViewController (you're logged

in). Logging in or logging out appropriately replaces the fourth tab with the correct ViewController.

Using four different NavigationControllers allow the user to navigate through one tab, without affecting his/her place in another tab.

Transitions to a new ViewController can be accomplished through eiter pushing a new ViewController to the stack (Navigation Controller) or modally presenting one. In the latter case, transition back to the previous ViewController must be done manually.

Class models

The four models in the project are Article, Webinar, User, and Project. They each roughly represents the models used in the backend, but not all the information is stored in the front-end, (like account creation date, etc). Each of them conforms to the NSObject/NSCoding protocols so they can be saved in NSFileManager. The id field of each of the object is unique and is the same id (key) in the database.

Making Network Calls

To make network calls we use the WebService class. First we create the parameters, which is a [String: Any] type. We then create a request by using WebService.createMutableRequest. We pass in the URL (endpoint), the method type, (POST, GET, etc), and the parameters to create a request object. We then execute the request by using WebService.executeRequest. A response code of 200 is a success, and anything else means

there was an error (probably a server error). If the network call is a success, we get the data back in the form of a JSON, which we can then parse. We use Alamofire library (https://github.com/Alamofire/Alamofire) to make network calls easy, and SwiftyJSON library (https://github.com/SwiftyJSON/SwiftyJSON) for easy JSON parsing. Both of them should be installed in step 5 of **Running the Code**.

Data Persistence

We've implemented data persistence to allow for an offline mode. As mentioned before, all the models adhere to NSObject/NSCoding protocols so they can be archive. We save the current user to the file systems (UserController.sharedInstance.currentUser) whenever we make changes to it, so logging in, registering, and editing (via UserController functions). We also save current user when its filters and favorited articles/webinars are changed. Example code of saving a user to the file systems is in LoginViewController (lines 64-66). Example code of retrieving the user from the file system is in AppDelegate (lines 33-47). We only remove the current user from our archives when the user press log out in ProfileViewController (lines 62-73).

We use the same approach for saving news articles and webinars. Whenever a user favorite an article or webinar, we append the ID (of the article or webinar) to the user's favorited list (an array of Ints), and stored the content of the article (or webinar) to the phone's memory using the same NSFileManager approach. This way the user can access his favorites even with no connection. When a user removes a favorite, we go into the app's file systems and remove the file. We also remove the entry from the user's favorited list. Whenever we're

adding or removing favorites, we also save the current user to the app's file system since its fields has been changed (either favoritedWebinars or favoritedArticles).

Future Works/Bugs

1. Constraint the views. Currently UiViews are not all constrained, so they do not look right in all screen dimensions, besides the iPhone SE. This can be either done in Interface Builder or programmatically. Related, functionality should be added so that pressing anywhere on app dismiss the keyboard (add a tap gesture to the ViewControllers). There's also ways to push up the screen when the keyboard appears.
2. Get sectors/thumbnails pictures of the webinars/news articles. Currently, the app does not get the sectors or thumbnail pictures of the webinars/news articles because the back-end does not support it. There's a way to get the sector of a news article/webinars, but it requires an API call for each article, which is extremely ineffective. Optimally, sector information/thumbnail picture (a url string) should be included in the JSON returned from /getnewsforsectors and /getwebinarsforsectors
3. Currently the featured news and upcoming events in the home tab are hardcoded because there's no API call for it.
4. Projects: Add a Created Projects section to a User's profile. Allow for numerous contributions per project.
5. Commenting for articles/webinar/projects
6. Searching returns all matches. Maybe show 10 at a time, and when the scrollview scrolls to the bottom, it shows 10 more results (Kind of like Facebook's newsfeed)

7. HTML parsing algorithm to make the HTML string more beautiful and mobile friendly before rendering it in the WebView (may be quite difficult)
8. Landing ViewController for each sector? Currently pressing on a sector cell (on the home page) redirects the app to the News Tab with the filters array false for every sector except for the selected sector. Instead, we can maybe have a new ViewController that contains description about that sector (trend analytics,etc), like the current E4C website.

Final

Github Repo: https://github.com/sc96/e4cApp

If there any questions, feel free to contact me at: samcheng@princeton.edu