Mobile NAT

For this code challenge, you will be creating a simple weather application named “Umbrella”. The application will download both the current conditions and an hour by hour forecast from Weather Underground. This application is intended to only be released in the United States.

In this code challenge we will be paying particular attention to the following items:

- **Design fidelity:** Can you accurately and efficiently implement the design as specified in the art comps.
- **Functionality:** Does the application meet the technical requirements and work reliably?
- **Architecture:** How do you structure your application and its classes? Would the application be extensible? How do you encapsulate data parsing and access?
- **Coding practices and use of IDE:** How do you organize your files and groups? What practices do you adhere to make the code accessible and usable to other developers? How is source control used within the application?
- **Fit and finish:** Do you adhere to the platform’s recommended practices?

Getting Started

All assets to get started can be found at the git repository. Use the terminal or your favorite source control GUI to clone the appropriate repository using the credentials:

The checkout contains reference designs, design metrics, and classes to expedite development. Feel free to not use the classes that we provide, but they’re there to help you out.

Weather Underground’s API provides free access to their service for developers. You can sign up for an API key at http://www.wunderground.com/weather/api/

Functionality

Every time that the application becomes the foreground app, the application should fetch the weather. If the user has not entered a ZIP code previously, the application should automatically prompt the user for the ZIP code.

ZIP Code Entry

The user should be able to enter the ZIP code as prescribed by the designs. On Android, the user should also be able to switch between Imperial and Metric representation of the data. On iOS, the user should see the weather displayed in the phone’s current locale. If there is any error with the data, an alert should be presented with relevant information and the user should immediately taken to enter a ZIP code again.

Hourly Weather Display

The hourly weather has two main sections. The top section is the current conditions of the entered ZIP code. If the temperature is below 60˚, use the cool color as specified by the designs. If the temperature is 60˚ or above, use the warm color as specified by the designs.

The other section of the main weather display is the hourly forecast. The data from the API should be grouped by days. The highest temperature of the day should have an warm tint as specified by the designs. The lowest temperature of the day should have a cool tint as specified by the designs. If there is a tie for the high or low, just highlight the first occurrence. If there is an occurrence where the high and low are the same hour, do not use a tint color.

Included in the checkout is a class that will provide the remote URL string for the weather icon - don't use Weather Underground's icons; they're ugly.

Application Requirements

- The application should be built with the latest public SDK and can target the latest public release of the OS.
- **iOS:** Application can be written in either Swift or Objective-C.
- **Android:** Application can be written in either Kotlin or Java.

Change Log

This is a living document. All changes that occur with will be detailed below. Most of the changes should be clarifications. If this document changes while you are completing the code challenge, you may continue with your assumptions or adapt to what has changed. If you choose to not implement what has changed, please detail that in a `README.md` at the root of your project.

- **July 20, 2017:** Added Kotlin as an allowed language
- **March 16, 2017:** Removed the requirement to change between Metric and Imperial Units for iOS
- **October 13, 2015:** Updated the document to allow the candidate to use Objective-C or Swift
- **July 21, 2015:** Updated the document to change "check in" to "locally commit" when referring to version control.
- **February 5, 2015:** Updated the document to ask for regular version control commits.