PROJECT REPORT
CS 4768

Instructor:
Dr. Chen

Group Members:
Bradley Gavan 201208634

Amir Torkashvand 201347184

Table of Contents

Overview

For our project we create a weather app. When launched, the local weather information is loaded, showing the current temperature, the name of the city the user is in, the date, the minimum and maximum temperature for the day, humidity percentage, an image representing the current weather conditions and a table populated by cells each containing an image of the weather condition, the temperature, the name of the weather condition and the time. There is a cell for every three hours of the remainder of the day. By using a 'flicking' motion towards the right (measured by a combination of the devices acceleration in the x-axis, and rotation in the z-axis) following days can be seen. A 'flick' to the left will return the previous day's information. The forecast data in our app has information for five days. Using the text field at the top of the screen, the user can also search for cities and have the forecast returned for that city (as opposed to the local data).

When the app is first used, a request for location services is made, if not accepted then local data cannot be received automatically (the user may still search for their own city in the search bar). An internet connection is needed to use this app.

Weather data is taken from OpenWeatherMap (openweathermap.org). Various free and premium options are available through OpenWeatherMap, our app uses the "5 day/ 3 hour forecast" service. The 5 day forecast is available at any location or city and includes weather data every 3 hours over that 5 day span. The forecast is available in JSON, XML, or HTML format. The data has a large list of parameters as seen below. The forecast information can be accessed by city name, city id or geographic coordinates. For further information on OpenWeatherMap, the OpenWeatherMap API can be consulted.

Below are list of parameters recieced from a OpenWeatherMap data point:

Parameters:

- `location`
  - `location.name` City ID
  - `location.type` Prototype parameter
  - `location.country` Country code (GB, JP etc.)
  - `location.timezone` Prototype parameter
  - `location.location`
    - `location.location.altitude` City geo location, altitude above the sea level
    - `location.location.latitude` City geo location, latitude
    - `location.location.longitude` City geo location, longitude
    - `location.location.geobase` Prototype parameter
    - `location.location.geobaseid` Prototype parameter
- `meta`
  - `meta.lastupdate` Prototype parameter
  - `meta.calctime` Speed of data calculation
  - `meta.nextupdate` Prototype parameter
- `sun`
  - `sun.rise` Sunrise time
  - `sun.set` Sunset time
- `forecast`
  - `forecast.time`
    - `forecast.time.from` Beginning of the period of data forecasted
    - `forecast.time.to` End of the period of data forecasted
  - `forecast.symbol`
    - `forecast.symbol.number` Weather condition id
    - `forecast.symbol.name` Weather condition
    - `forecast.symbol.var` Weather icon id
  - `forecast.precipitation`
    - `forecast.precipitation.value` Precipitation volume for the last 3 hours, mm
    - `forecast.precipitation.unit` Period of measurements. Possilbe value is 1 hour, 3 hours.
    - `forecast.precipitation.type` Type of precipitation. Possible value is rain, snow.
  - `forecast.windDirection`
    - `forecast.windDirection.deg` Wind direction, degrees (meteorological)
    - `forecast.windDirection.code` Code of the wind direction. Possilbe value is WSW, N, S etc.
    - `forecast.windDirection.name` Full name of the wind direction.
  - `forecast.windSpeed`
    - `forecast.windSpeed.mps` Wind speed, mps
    - `forecast.windSpeed.name` Type of the wind
  - `forecast.temperature`
    - `forecast.temperature.unit` Unit of measurements. Possilbe valure is Celsius, Kelvin, Fahrenheit.
    - `forecast.temperature.value` Temperature
    - `forecast.temperature.min` Minimum temperature at the moment of calculation. This is deviation from 'temp' that is possible for large cities and megalopolises geographically expanded (use these parameter optionally).
    - `forecast.temperature.max` Maximum temperature at the moment of calculation. This is deviation from 'temp' that is possible for large cities and megalopolises geographically expanded (use these parameter optionally).

- `forecast.pressure`
  - `forecast.pressure.unit` hPa
  - `forecast.pressure.value` Pressure value
- `forecast.humidity`
  - `forecast.humidity.unit` %
  - `forecast.humidity.value` Humidity value
- `forecast.clouds`
  - `forecast.pressure.value` Name of the cloudiness
  - `forecast.pressure.all` Cloudiness
  - `forecast.pressure.unit` %
  -

Core Location & Map Kit

An instance of CLLocationManager (“locationManager”) is allocated and initialized at the beginning of our program in the “viewDidLoad” method. Following this, a call to method getUserLocation is called, which sets the delegate for CLLocationManager as well as sets various values such as, requests for authorization , desired accuracy, and tells the manager to start updating location. The didFailWithError method for CLLocationManger is also defined. If there is an issue with getting a user's location, a UIAlertAction iappears to inform the user.

- locationManager:didUpdateLocations:

This is called everything time the location manager receives an update. It gets the most recent location and creates an instance of CLLocation. The longitude and latitude of the location are used in the MapKit method reverseGeocodeLocation. What this does is gives us a MKPlaceMark of the city the user is in. The city named is derived from this and is updates the city name label of the interface.

- locationManager:didFailWithError:

This delegate method is called when the location manager is unable to get a location value. In our code, it catches any errors, which may include kCLErrorLocationUnknown, kCLErrorHeadingFailure and kCLErrorDenied. A UIAlertView is created to inform the user if an error has occurred in retrieving their location. At this point, the user can try to resolve this issue. They may need to simply wait and try again, or check that their location services are enabled. The user also have the option of simply searching for the city that they are in using the search bar at the top of the screen.

- loadDataFromLat:Lon:

This is a method we defined. It creates the URL needed for an NSURLSession. The URL has a general form of "http://api.openweathermap.org/data/2.5/forecast?lat=_&lon=_&units=metric&appid=_". The purpose of this method is to fill in the blanks with the variable information (latitude and longitude). The latitude and longitude are passed in as parameters, the key (appid) is defined in at the beginning of the program in the viewDidLoad method. The base URL and these variable informations are concatenated together to create the full NSURL needed for the OpenWeatherMap API call of the user's location. An example of a URL for St. John''s is "http://api.openweathermap.org/data/2.5/forecast?lat=47.574237&lon=-52.735337&units=metric&appid=006f2b6cd7724f9e3b60d7fdfd28de82". Finally, the method loadDataFromURL:url is called using the generated NSURL. Metric units are used

- loadDataFromCity:sender

This is a method we defined. This is a variation of loadDataFromLat:Lon: used to make a call to the OpenWeatherMap API based on the name of a location, instead of its coordinates. Though it does not make use of the CoreLocation or MapKit frameworks, it makes sense semantically to be included in this section. It is called in response to the pressing of the "Go" button on the interface. Similarly to loadDataFromLat:Lon:, a NSURL is constructed by concatenating the variable components to the OpenWeatherMap API call. The URL has a general form of "http://api.openweathermap.org/data/2.5/forecast?q=_&units=metric&appid=_". The purpose of this method is to fill in the blanks with the variable information (city name and appid). This variation of the OpenWeatherMap accepts a city name, this is taken from the cityTextField textfield (seach bar at top of screen).

The API key is also added (it is defined in the viewDidLoad method). These NSStrings are concatenated together to create the full NSURL needed for the OpenWeatherMap API call. An example of a URL for Halifax is "http://api.openweathermap.org/data/2.5/forecast?q=Halifax&appid=006f2b6cd7724f9e3b60d7fdfd28de82".

Finally, the method loadDataFromURL:url is called using the generated NSURL.

The method checks if the text field is empty, if so no action will be taken. Seemingly nonsensical city names can be entered and will return results. For example if "zzz" is given as the city name, the forecast of Blue Ridge, US is returned. This is determined from the OpenWeatherAPI itself. At this time we do not understand why this is or how it works.

Network

loadDataFromURL:

This is the method where we get all of our data. A NSURL is passed as a parameter and is the URL needed for the call to OpenWeatherMap to get the desired forecast. It comes from either loadDataFromCity or loadDataFromLat:Lon. A call to downloadDataFromURL (a method defined in the AppDelegate) is called to handle the downloading of information. An instance of NSURLSession is configured with the default settings and a NSURLSessionDataTask is created and is given the URL. Any errors here are logged. If the Data Task is successful, we check the HTTPStatusCode to ensure that everything went smoothly (this would be a HTTPStatusCode of 200). If there are no issues, we now have the data.

We now return to the main thread with our new forecast data. The data is returned as a dictionary and is in JSON format. We use the JSONSerialization class to create a NSMutableDictionary from the returned data. If there's no errors, we insure that data we received is complete and correct (the "cod" field == 200). At this point we can iterate through each of the data points (should be 40). Each member in the "list" element of the dictionary ("list" is where the data points are contained) is a dictionary containing many parameters. We use the dictionary that is each data point to initialize a WeatherInfo object. We end up with the 'forecastsArray' having a WeatherInfo object for each data point. A UIAlertController appears if there is an error.

CoreMotion

We decided to implement CoreMotion in our app as a way to navigate through the days of the forecast. Our forecast has up to 40 data points and we wanted to incorporate all of that data into our app. We knew that that was too much information to display on a single view. Our solution to that was to keep our app as a single view application but to load the different data (grouped by day) when a "flick" motion is registered. A "flick" to the right will show the next day's data, while a "flick" to the left will return to the previous day's data. This motion is registered by a combination of acceleration in the x-value and the gyroscope measurement of the device's yaw (rotation of the z-axis).

We had difficulty figuring out the right combination of measurements and values to differentiate between left and right and maintaining a threshold. The CMAttitude class offers solutions such as

rotation matrices and quaternions but those went beyond our scope of knowledge and understanding. We ended up finding our solution by a combination of trial and error and use of the DemoCoreMotion app from class, which helped us see the different acceleration/rotation values as various motions were performed. We created an instance of CMMotionManager to manage both the accelerometer and gyroscope. Both update intervals are set to '.1' of a second.

- UpdateAccelerationInfo:

This method is called every '.1' seconds (the accelerometer update interval). This method checks three things when it is called. The acceleration in the x-axis, rotation in the z-axis and the currentDay variable (an integer, from 1 to 5, representing which day's data is currently being shown). To go to the next day (increment currentDay by 1 and reload the weather data), currentDay must be less than 5, acceleration to the right must be greater than 1.5 and the yaw must be less than -5. To go to the previous day (decrement currentDay by 1 and reload the weather data), acceleration to the left must be greater than 1.5 (less than -1.5) and the yaw must be greater than 5.

WeatherInfo data structure (Object)

To encapsulate the information for each weather data point, we created a WeatherInfo object. The object has a list of properties and a method for setting them. The properties are initialized with "@property" and can therefore be easily accessed with dot notation. The WeatherInfo implementation files defines the method initWithDictionary which initializes all these values for an instance of WeatherInfo.

initWithDictionary

Initializes a WeatherInfo object with the following properties:

- Atmospherical pressure
- Minimum temperature
- Maximum temperature
- Temperature
- Wind direction in degrees
- Wind speed (m/s)
- Date
- Weather Description
- Icon name of weather condition
- Category of weather condition
- Humidity
- Sea level
- Expected rain in mm
- Expected snow in mm

seperateForcastByDateFromArray

This method is used once all the data for the five days are received. The WeatherInfo objects of for each day will be separated by date into NSMutableArrays, which is we called _forecastDays. We create a temporary WeatherInfo objecte and assign, one-by-one, as a reference to each WeatherInfo objects we have. Date objects are created which represent each of the five days of the forecast and that is what is used to differentiate the objects by day.

LoadTableDataFromArray:Day

In this method, the properties to be displayed on the interface are set. Arrays for each of the properies (such as temperature and humidity) are created and the respective values for each WeatherInfo

object (of a certain day) will be added to the correct array. The minimum and maximum temperature for a day is found, and the time is formatted in this method. The interface filelds (aside from the table) is set. Each weather condition has an ID and this ID is used to display the appropriate weather Icon (matched by ID) on the screen. For the main weather icon, either the weather description from the WeatherInfo object is selected (if we are using the first day) or the WeatherInfo object for noon is selected (if we are using days two through five).

Interface

Figure 1

As it is shown in figure 1, the main storyboard has a text field where the user can type in the name of the city and press “Go” and all the weather info (current and next four days) of that city will be shown. Also, the “Local Weather” button creates the forecast for the user's location (useful) if the user is viewing the weather of another city.

Also, on the main story board there are text fields for city, date, Min/Max temperature, humidity and UIImage view which presents the icon of the current weather condition (or weather condition at noon if it is not today's weather). On the bottom of the view we have a UITable which shows custom cells. There are only four styles avaiablefor UITable cells, and we felt that none of these met our needs, so we decided to create our own subclass of UITableViewCell. We made our own customized cell to be able to present more data in each cell (figure 2).

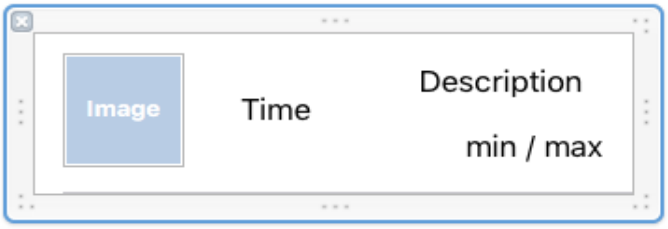

Figure 2

WeatherTableViewCell.h and WeatherTableViewCell.m have been created and they are the sub-class of UITableViewCell. In the .h file properties such as iconImage, time, minMaxTemp, weatherDescription exist and the UITable in the main storyboard uses this custom cell.

In terms of UIImages, each cell presents the data for the interval of 3 hours and the UIImage in the main story board is the same as the first image on the cell for the current day depending on the time the user launches the App. However, the icon for any other day other than the current day, is equal to the icon of 12pm of that specific day.

Permissions

In info.plist we have added a few fields regarding permissions. For example, we have added NSLocationWhenInUseUsageDescription which asks user if the application is authorised to use device location for loading data and if they select “Allow” the App will run as usual but if the user chooses “ Don’t Allow” the App will not find the user’s location but the user will be able to search any city and the data for that city will be loaded.

Regarding the security, App Transport Security (ATS) forces applications to use HTTPS for connecting to web but since our API does not use HTTPS we had to add “Allow Arbitrary Loads” to ATS to allow HTTP connections. Furthermore, with the graphical layout and use of tables in our App, we decided to support only the portrait orientation. In Supported interface orientations section we deleted the options for the landscape mode to make sure our App is only used in portrait. Lastly, since it is essential to use location we have added another item to required device capabilities which is “location-services”.