Assignment 2 –
Weather tracking app

ENG1003, Semester 1, 2016
Due: Sunday May 29th, 11:59PM (local time)
Worth: 19% of final mark

Aim

Many people use their mobile phones to look up weather conditions in their current location or to check the weather in other places of interest. Have you ever wondered how exactly these apps work? In this assignment you will build a location-aware app that utilises an online forecasting service to present to the user current and historic weather information for locations of interest.

What you are provided with

You are given skeleton code which can be found in the resources folder on the ENG1003 server:

https://eng1003.eng.monash.edu/resources/

This skeleton code demonstrates navigation between the required pages of the app, and shows some navigation controls using the MDL JavaScript framework.

We provide you with a description of the required functionality for the app. This assignment will build on (but is not limited to) your understanding of variables, arrays, control structures, objects, functions, sensor use, local storage, web service communication, and software design. For this assignment, you will extend the skeleton code we have given you, keeping the provided directory and file structure.

What you need to do

This assignment consists of several tasks, including programming, documentation, use of software tools. For the programming tasks, we suggest you work **together** on individual tasks and gradually build up the application functionality.

Getting started

1. Someone in your team needs to sign up for an account on developer.forecast.io, the web service you will use for weather data (see "Resources" section below). This is required to obtain an API key which is in turn needed to call the forecast.io JavaScript API.
2. Team members should have already have a GitHub account and should have joined their team Git repository (as per Week 7 prac class or using the classroom link provided via email).
3. One team member should create the necessary initial directory structure in your team GitHub repository by committing the provided skeleton code (see Skeleton code and also the "Submission" section below).

Required functionality

Launch page ("Locations List")

The app should launch to page that displays a list of saved locations that have been added by the user. This list will initially be empty. These user-specified locations should persist between app launches, that is, be stored in the browser's local storage. A "location" consists of a nickname, a latitude and a longitude.

Each entry in the list should display the location nickname. For each location the current day's low and high temperature and condition summary (as text or icon) should also be shown. When first loading the app this information might not be available. Hence, the page should show that the weather for each entry is loading until the required data is returned from the forecast API. You should use the `LocationWeatherCache` class to request weather information (see below).

This page should display a header bar at the top with the title "Locations". This header bar should have a "+" button on the far right, which when tapped opens the Add Location page.

Tapping on any entry in the Locations List should open the Location Weather page for that location.

Add Location page

This page should have a header bar at the top with the title "Add location". It should show two text fields. The first field should allow the user to type a location. The second should allow them

to enter a nickname for the location, e.g., “Home” or “Work”. Aside from the text fields, this page should display a map and button titled “Add location”.

The expected workflow for this page is that the user taps the location field and types in a location such as an address or suburb. As they enter characters, the app should utilise the Google Geocoding API (see “Resources” section below) to look up the GPS coordinates for that location and display the location on the map. The app should display the formatted address returned by the geocoding API as an annotation on the map. The behaviour of this page is intended to allow the user to confirm they have entered the desired location. Next, the user can optionally enter a nickname for the location. Then, if the user taps the “Add location” button the current GPS location should be added to the `LocationWeatherCache` object (the cache saved to localStorage) and the user returned to the Locations List page. If no nickname is specified by the user, the formatted address from the geocoding request should be used.

Note: If there is no valid GPS position for a given location entered by the user, then nothing should be added to the Locations List and an error can instead be shown to the user.

Location Weather page

The Location Weather page should show weather information for the selected location. The top of this page should show a header bar with the nickname for the active location. Below this the page should include the following:

- A map displaying the active location,
- The date that the displayed weather applies to,
- A date selection slider,
- A summary of the weather for the particular date, and
- A “Remove this location” button

The date should be displayed in the YYYY;-MM-DD format. We have provided a new method for the Date class called `simpleDateString()` which returns a String.

The slider should be able to be dragged to change the displayed date, and the displayed weather. The slider should have 30 positions, representing the current day and the previous 29 days. The slider position should initially be set to the far right, i.e., the current day. The change to the selected date should happen in response to the user dragging the slider control, rather than waiting until the user has let the tool go.

The map should be centred on the selected location and highlight it in some way.

The weather information displayed for the selected date and location should include:

- A human-readable text summary of current conditions, e.g., “Partly Cloudy”
- Minimum temperature in ℃
- Maximum temperature in ℃
- Current humidity in %

- Current wind speed in km/h
- *Anything else you want to show.*

Like the Locations List, this page might need to display weather information which is not yet available. In this case it should show that the weather information is loading and this information should be automatically filled in once the info has been returned by the forecast.io API. You should use the `LocationWeatherCache` class to request weather information (see below).

If the user taps the "Remove this location" button then location being viewed should be removed from the `LocationWeatherCache` (the cache saved to local storage) and the user should be returned to the Locations List page.

LocationWeatherCache class

This file `locationweatherCache.js` in the skeleton contains a `LocationWeatherCache` class. This class includes method names but the methods themselves still need to be filled in. This file should not be dependant on any of the rest of the code of the app. This JavaScript file will be included in each of the three web pages of the app.

The purpose of the `LocationWeatherCache` class is to provide a simple way to access weather information (current day or historic) for particular locations. This class should act as a cache for the forecast.io API. That is, when the app requests the weather for a particular location and date, the app should check the cache and return this information if it is stored locally by the class, otherwise it should request this data from the API, and then store the result in in the class and notify the app via a callback when it receives the information.

- This class should have a private attribute called locations, an array that stores a list of location objects. This array should initially be empty.
- Each location object should have a `nickname`, `latitude`, `longitude` and `forecasts` properties. The `forecasts` property should contain an object. This forecasts object should contain a property for each cached weather forecast. The property name should be a combination of the location and date, of the form "{lat},{lng},{date}", e.g., "46.0617697,12.078846800000065,2016-04-01T12:00:00" (the same format used for the forecast.io request). The property value should be the daily weather forecast object returned by forecast.io. Hint: the hasOwnProperty() method can be used to check if the forecasts object has weather for a particular location and date.
- `The LocationWeather` class instances should be persistent, i.e, they should be preserved between app launches. This means storing them to Local Storage when their data changes. Hence you need to implement a `toJSON()` method that writes out the private attributes of the class by returning a version of the class with only public attributes. You should also implement an `initialiseFromJSON()` method which sets the private attributes for a class instance, allowing the `LocalWeather` class instances to be recreated when the app is reopened.
- The cache class should include a `getWeatherAtIndexForDate()` method. This method takes three arguments: the index of the location, a JavaScript Date object

representing the requested date, and a callback function to be called when the weather is returned (see comments in the `locationWeatherCache.js file`).

- The time component of the date can be ignored. The app cares only about dates at daily granularity. We provide you a `Date.forecastDateString()` method which given a date returns a string formatted exactly for use with the forecast.io.
- This method should check if the weather data point for this date is already in the `forecasts` array. If so, it should call the callback function and pass the corresponding weather data as an argument.
- If the class does not have the weather data locally, it should call the forecast.io API using JSONP. When the request returns, the 'daily' weather data point stored be stored into the forecasts array and also returned to the callback function. Note, we don't want to use the hourly forecasts returned by the forecast.io API,
- By default, the forecast.io API returns a heap of data we don't need. It is just the data point in the `daily` property that we are interested in. Hence the `minutely`, `hourly` and currently data blocks should be ignored, which can be done by including the `exclude` field in the API call query string.
- The user may specify different callback functions to be invoked for different location weather requests. In order to cope with this possibility it is necessary for you to temporarily store these callback function references in the `callbacks` private attribute of the class. These are only useful until the query has completed, and don't need to be serialised to local storage.

The locationWeatherCache.js file should contain a loadLocations() function. When invoked, this function should create a single LocationWeatherCache class instance in a global variable called LocationWeatherCache. It should then check local storage for an existing cache object. If there is one it should initialise the locationWeatherCache object from the serialised version in local storage. This function should be called at the end of the locationWeatherCache.js file. Since this file is included in all pages of the app, this ensures that the cache will be loaded from local storage and always be available.

The `locationWeatherCache.js` file should contain a saveLocations() function. When invoked, this function should serialise the locationWeatherCache object to local storage.

Extension: Current Location

This functionality is optional. Without completing it the maximum code mark is 9/10

For the Locations List, the first entry in this list should be "Current location", followed by any other locations the user has added. The "Current Location" need not have a weather summary shown for it on the Locations List page.

When the user taps on the Current location list entry the app should open the View Weather page. In this case, the View Location page should show the text "Current location" as the title in

the header bar. Additionally the page should initialise the GeoLocation API in order to watch the user's position via the device GPS. As the app receives location updates from this API it should display the current position and location accuracy on the map. It should also display the weather at this location for the current day, updating as the GPS location changes.

General notes

Beyond providing the above functionality you should feel free further customise the layout, style and behaviour of the app.

The programming tasks together are worth 10% of your unit mark.

Contribution through use of tools

Each team member will be individually assessed on their use of appropriate tools for software development. You will use GitHub for managing revisions of the app source and handling commits by multiple team members. You will use Google Drive for document management. You will use Asana for team communication, project management and issue tracking.

The history of your contribution to Git, Asana and Google Drive will be individually considered. For each of these tools you will be given a score of "appropriate", "some" or "none" depending on your observed level of contribution. Students with less than the acceptable level of contribution will incur a penalty to their assignment mark. Note: it is not enough to just use these tools for some dummy actions just prior to submission—this will not be counted. It is expected that you will use these tools weekly throughout the term of the assignment.

You **must** give your demonstrator access to your team Asana workspace and Google Drive folder for Assignment 2.

Technical documentation

Your team should produce two short pieces of technical documentation for your app:

- Project Management Plan. This is internal documentation detailing:
    - Project aim and scope
    - Team members, and their responsibilities
    - How you are going to do the project, e.g., team meetings, minutes, handling communication of changes to the code
    - Any other information you would want to give to a new team member
- User Guide. This is external documentation detailing:
    - How to use the various features of the app, with screenshots
    - Any known bugs or limitations
- You may include other documentation if you feel it is relevant

The technical documentation tasks are together are worth 9% of your unit mark.

Testing the app

The main part of the app can be tested from a local director with Desktop Chrome. Testing the GPS functionality requires the app to be uploaded to a web server.

In order to test your app you should upload the source files for you app into your team directory on the ENG1003 server (http://eng1003.eng.monash.edu/) using an SFTP client. More information is available on the unit Moodle page. 2

Resources

- http://www.getmdl.io/
  (MDL: Material Design Lite documentation)
- https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/Using_geolocation
  (Mozilla Developer Network: Reading the GPS sensor)
- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Date
  (Mozilla Developer Network: Date documentation)
- https://developer.forecast.io/docs/v2
  (The Dark Sky Forecast AI documentation)
- https://developers.google.com/maps/documentation/javascript/tutorial
  (Google Maps JavaScript API)
- https://developers.google.com/maps/documentation/javascript/geocoding
  (Google Maps JavaScript API: Geocoding Service)

Submission

You must your submit your assignment with the folder structure and file naming scheme shown below:

```
─┬  assignment2/
 ├─┬  code/
 │ ├─┬  js/
 │ │ ├──  mainPage.js               (Main page - Locations List)
 │ │ ├──  addLocationPage.js        (Add Location page)
 │ │ ├──  viewLocationPage.js       (View Location page)
 │ │ └──  locationWeatherCache.js   (LocationWeatherCache class)
 │ ├─┬  css/
 │ │ └──  weatherapp.css
 │ ├─┬  images/
 │ │ └──  [...]
 │ ├──  addLocation.html         (Add Location page)
 │ ├──  viewLocation.html        (View Location page)
```

```
│  └──  index.html                (Main page - Locations List)
└──┬  docs/
   ├──  UserGuide.pdf
   ├──  ProjectManagementPlan.pdf
   └──  GroupAssignmentCoversheet.pdf
```

These should be zipped up with the team name as the filename, e.g., “Team014.zip”. The contents of the zip file submitted to Moodle must match the content on the “assignment2” folder in your team’s GitHub repository. The submission should be uploaded by the team leader by the assignment due date.

Each team member must individually complete the

- CATME peer assessment survey

as described in the “Peer Assessment” section below.

Your app will be assessed based on the version of the code you submit. We will upload it to the server and run it on the same type of phone as your team smartphone.

Late submissions

We do not accept late submissions without special consideration. Such special consideration applications should be made to the unit email address with a completed form and supporting documentation within two business days of the assignment deadline.
http://www.infotech.monash.edu.au/resources/student/equity/special-consideration.html

Plagiarism

Cheating and plagiarism are serious academic offenses at Monash University. Students must not share their team’s work with any student outside of their team. Students should consult the policies linked below for more information.
http://www.monash.edu.au/students/policies/academic-integrity.html
http://eng.monash.edu.au/current-students/cheating-and-plagiarism.html
Students involved in collusion or plagiarism will be subject to disciplinary penalties, which can include:

- The work not being assessed
- A zero grade for the unit
- Suspension from the University
- Exclusion from the University

Peer Assessment

You are expected to work together as a team on this assignment and contribute roughly equal amounts of work. Peer assessment will be conducted via the CATME online system. You will receive email reminders at the appropriate time.

Not completing the CATME peer assessment component may result in a score of zero for the assignment.

Do:

- Give your teammates accurate and honest feedback for improvement
- Leave a short comment at the end of the survey to justify your rating
- If there are issues/problems, raise them with your team early
- Contact your demonstrators if the problems cannot be solved amongst yourselves

Do NOT:

- Opt out of this process or give each person the same rating
- Make an agreement amongst your team to give the same range of mark

Marking criteria

This assignment consists of several component assessment items with the following associated marks (percentages of total marks for the unit):

- App code and functionality — 10% (group)
- Production of technical documentation — 9% (group)
- Use of appropriate tools — (used to calculate individual contribution factor)

Assessment criteria:

- Required functionality and correct behaviour of the produced app
- Quality of app source code, including code documentation and overall structure
- Clarity and quality of individual oral presentations
- Structure, appropriateness, and level of team-client presentation
- Participation and appropriate use of tools for team software development, including communication style
- Appropriateness of technical documentation, including structure, completeness and communication quality

You will be marked as a group, however your individual marks will be subject to peer review moderation based on CATME feedback and scaling factors.

Where to get help

You can ask questions about the assignment on the General Discussion Forum on the unit's Moodle page. This is the preferred venue for assignment clarification-type questions. You should check this forum (and the News forum) regularly, as the responses of the teaching staff are "official" and can constitute amendments or additions to the assignment specification. Before asking for a clarification, please look at the forum.