Homework 8: Weather Forecast Web App using Bootstrap, JQuery, and Facebook Mashup

1. Objectives

- Become familiar with the AJAX, JSON & XML technologies.
- Use a combination of HTML, CSS, DOM, PHP and JSON.
- Get hands-on experience in Amazon cloud computing (AWS)
- Get hands-on experience on how to use Bootstrap and jQuery UI to enhance the user experience
- Provide an interface to perform Weather forecast search and post details to Facebook.

2. Background

2.1 AJAX & JSON

AJAX (Asynchronous JavaScript + XML) incorporates several technologies:

- Standards-based presentation using XHTML and CSS;
- Dynamic display and interaction using the Document Object Model (DOM);
- Data interchange and manipulation using XML and XSLT;
- Asynchronous data retrieval using XMLHttpRequest;
- JavaScript binding everything together.

See the class slides at http://cs-server.usc.edu:45678/slides/ajax.pdf

JSON, short for JavaScript Object Notation, is a lightweight data interchange format. Its main application is in AJAX web application programming, where it serves as an alternative to the use of the XML format for data exchange between client and server. See the class slides at

http://cs-server.usc.edu:45678/slides/JSON1.pdf

2.2 Google Maps Geocoding API

Google Maps Geocoding API provides functionality for converting addresses to geographic coordinates. For more information, refer to the following link:

https://developers.google.com/maps/documentation/geocoding/intro

2.3 Forecast.io API

Forecast.io API provides detailed description of current weather, the next 24 hours as well as the forecast for upcoming days. You can refer to the API description on the following link:

https://developer.forecast.io/docs/v2

2.4 Bootstrap Library

Bootstrap is a free collection of tools for creating responsive websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. To learn more details about Bootstrap please refer to the lecture material on Responsive Web Design (RWD). See the class slides at:

http://cs-server.usc.edu:45678/slides/Responsive.pdf

and

http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

2.5 Amazon Web Services (AWS)

AWS is Amazon's implementation of cloud computing. Included in AWS is Amazon Elastic Compute Cloud (EC2), which delivers scalable, pay-as-you-go compute capacity in the cloud, and AWS Elastic Beanstalk, an even easier way to quickly deploy and manage applications in the AWS cloud. You simply upload your application, and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. Elastic Beanstalk is built using familiar software stacks such as the Apache HTTP Server, PHP, and Python, Passenger for Ruby, IIS 7.5 for .NET, and Apache Tomcat for Java.

The Amazon Web Services homepage is available at:

http://aws.amazon.com/

2.6 Facebook

Facebook provides developers with an API called the Facebook Platform. Facebook Connect is the next iteration of Platform, which provides a set of API's that enable Facebook members to log onto third-party websites, applications and mobile devices with their Facebook identity. While logged in, users can connect with friends via these media and post information and updates to their Facebook profile.

Below are a few links for Facebook Connect:

https://developers.facebook.com/
https://developers.facebook.com/docs/javascript

3. High Level Description

Similar to HW6, in this exercise you are asked to create a webpage that allows users to search for weather forecast information using the Forecast.io API and display the results on the same page below the form.

The difference being, in this homework you will create a PHP file to return a JSON formatted data to the front-end (making asynchronous AJAX calls) and the front-end will parse the JSON data and show it in a nicer-looking responsive UI (using Bootstrap).

A user will first open a page as shown below in Figure 1, where he/she can enter the location information such as Street address, City and State (mandatory – indicated by a red asterisk symbol after each title) and make a selection for the preferred degree unit or keep the default Fahrenheit and execute the search. The description of the Search Form is given in Section 4.1. Instructions on how to use the APIs are given in Section 5.

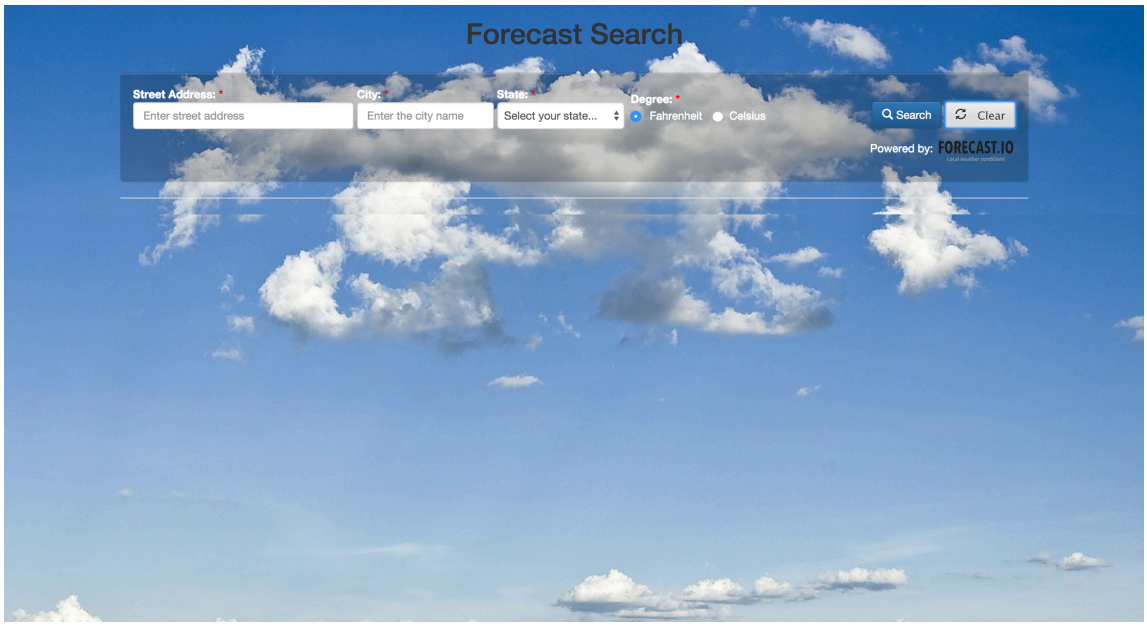

Figure 1: Initial Search Page

Once the user has provided data and clicks on the Search button, validation must be done to check that the entered data is valid. Form validation rules along with screenshots are given in Section 4.2.

Once the validation is successful, the jQuery function .ajax() is executed to start an asynchronous transaction with a PHP file (script) running on your AWS server, and passing the search form data as parameters of the transaction.

The PHP file you request is based on your HW#6. The difference is that this time the file does not need to display the data as HTML but instead will return the JSON data from the Forecast.io to your search webpage. The webpage must then use JavaScript to extract data from the JSON and display the results on the same webpage. Description of how to display the results is given in Section 6.

4. SEARCH FORM

4.1. Description

You **must replicate** the form displayed as in Figure 2 using a Bootstrap form. The form fields are the same as in your homework 6.

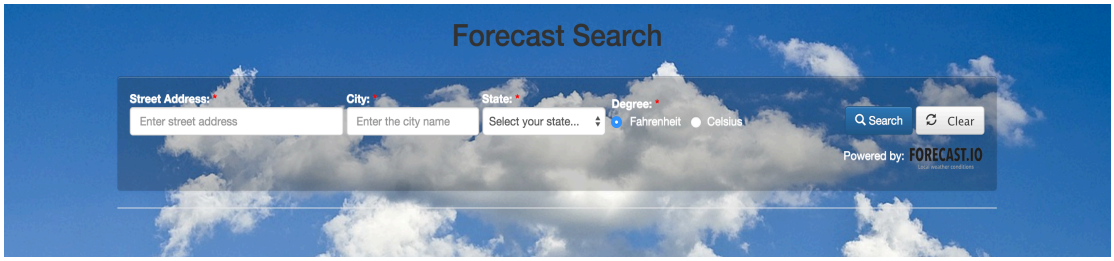

Figure 2: Initial Form

The search form has two buttons:

1. **SEARCH button:** On the button click validations must be performed. If validations are successful, then an AJAX asynchronous request is made to your web server (PHP file hosted on AWS), providing it with the form data that was entered. If validations fail, appropriate messages must be displayed under the appropriate text box, and an AJAX call should **NOT** be made with the invalid data.

2. **CLEAR button**: This button **must** clear the result area, all text fields, reset the temperature option to Fahrenheit and clear all validation errors if present. The clear operation is done using a JavaScript function.

Note: As we are making an asynchronous AJAX call in the process of generating results, the page isn't refreshed, hence you need not maintain state of the form fields explicitly as it was required in HW6.

The webpage has a background image that spreads through the entire visible browser window. The background image can be found at http://cs-server.usc.edu:45678/hw/hw8/images/bg.jpg. The form is displayed with a transparent background. The form should include a forecast.io logo disclaimer,

linking to http://forecast.io. The logo to be used can be found at http://cs-server.usc.edu:45678/hw/hw8/images/forecast_logo.png . The form should end with a white horizontal line to separate the form from the search results.

4.2. Validations

If the user did not enter a street address, city and/or state, then a message should be shown with appropriate text requesting the user to provide the missing information. Popups are not acceptable. The validations to be done along with the messages to be displayed are listed below.

Street Address – should not be empty or just be spaces. If it is, "Please enter the street address" should be displayed below the text field.

City – should not be empty or spaces. If it is, "Please enter the city" should be displayed below the text field.

State – should not be selected to "Select your state…". If it is, "Please select a state" should be displayed below the text field.

These error messages are displayed if any one or more validation cases are not satisfied and the user clicks on the search button. Further, once the user presses the Search button, if a user is deleting the content of a required field, and after deleting all characters from the required field it becomes empty, then the same message should be displayed. An example is shown in Figure 3.

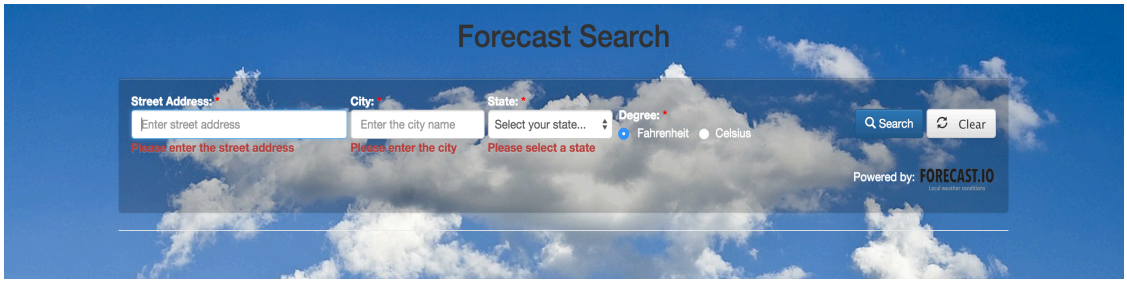

Figure 3: Form displaying validation errors

Responsive to Devices

You are supposed to make the search form responsive to devices. If the page is loading on a smart phone or a tablet, the form should display according to the width of the devices. One example is shown in Figure 4 below.

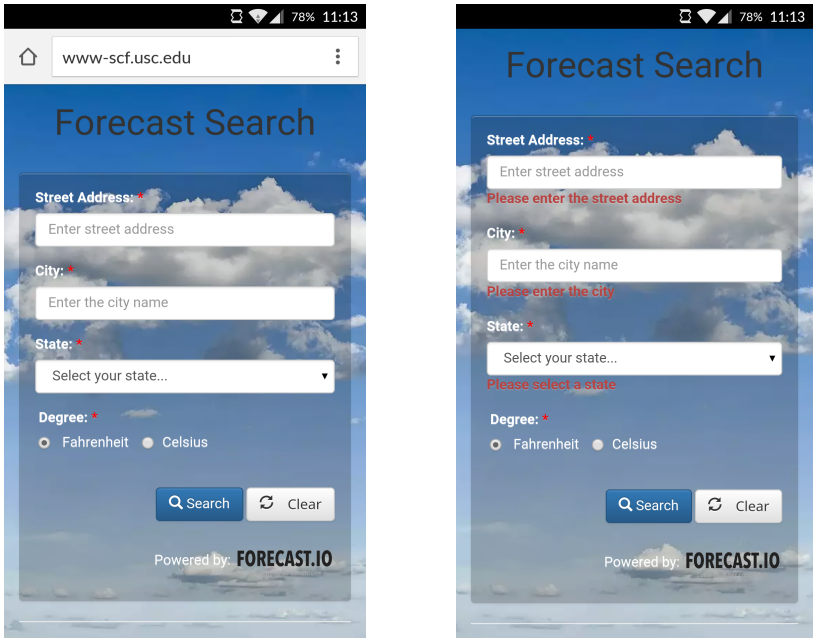

Figure 4: Initial form page and form with validation errors on smartphone

5. API Usage

The same Google Maps Geocoding API and Forecast.io API calls created for homework 6 can be used for homework 8 as well.

5.1. JSON data from Forecast.io

The following sample JSON data explains the section of the data to be used for displaying the results:

Current Weather Data:


```
{
   "latitude":34.028668,
   "longitude":-118.286325,
   "timezone":"America/Los_Angeles",
   "offset":-7,
   "currently":{
      "time":1445385373,
      "summary":"Clear",
      "icon":"clear-day",
      "nearestStormDistance":11,
      "nearestStormBearing":228,
      "precipIntensity":0,
      "precipProbability":0,
      "temperature":76.5,
      "apparentTemperature":76.5,
      "dewPoint":53.5,
```

```
      "humidity":0.45,
      "windSpeed":7.71,
      "windBearing":261,
      "visibility":9.93,
      "cloudCover":0.04,
      "pressure":1009.28,
      "ozone":298.55
   }
```


Next 24 hours Weather Data:
Below is the hourly data retrieved from Forecast.io. A single object in the data[] array maps to a single hour.


```
 "hourly":{
      "summary":"Clear throughout the day.",
      "icon":"clear-day",
      "data":[
         {
            "time":1445382000,
            "summary":"Clear",
            "icon":"clear-day",
            "precipIntensity":0,
            "precipProbability":0,
            "temperature":77.03,
            "apparentTemperature":77.03,
            "dewPoint":53.75,
            "humidity":0.45,
            "windSpeed":7.12,
            "windBearing":264,
            "visibility":8.91,
            "cloudCover":0.04,
            "pressure":1009.13,
            "ozone":298.97
         }….]
```


Next 7 days Weather data:
This is the daily weather data. Each object in the data[] array maps to a single day.


```
"daily":{
      "summary":"No precipitation throughout the week, with te
mperatures rising to 85°F on Tuesday.",
      "icon":"clear-day",
      "data":[
         {
            "time":1445324400,
            "summary":"Clear throughout the day.",
            "icon":"clear-day",
            "sunriseTime":1445349798,
```

```
        "sunsetTime":1445390092,
        "moonPhase":0.25,
        "precipIntensity":0.0001,
        "precipIntensityMax":0.0012,
        "precipIntensityMaxTime":1445320800,
        "precipProbability":0.02,
        "precipType":"rain",
        "temperatureMin":61.92,
        "temperatureMinTime":1445346000,
        "temperatureMax":77.2,
        "temperatureMaxTime":1445378400,
        "apparentTemperatureMin":61.92,
        "apparentTemperatureMinTime":1445346000,
        "apparentTemperatureMax":77.2,
        "apparentTemperatureMaxTime":1445378400,
        "dewPoint":54.71,
        "humidity":0.61,
        "windSpeed":1.89,
        "windBearing":303,
        "visibility":9.24,
        "cloudCover":0.07,
        "pressure":1010.65,
        "ozone":304.79
    }….]
```


5.2. OpenWeatherMap

OpenWeatherMap map is a service supplies libraries which can provide the current cloud map over a region. You need to use **OpenWeatherMap direct tile server** functionality to add the weather map layers to form the cloud map. OpenLayers.js is the library which is required to be included to create the cloud map.

The layers to be added to the basic **Open Street Map** are:

1. Clouds
2. Precipitation

For details for the library and its usage refer the OpenWeatherMap documentation at

http://openweathermap.org/hugemaps#how

Hints:

- You will need to set the latitude and longitude in the cloud map using the JSON data retrieved from Forcast.io
- Set the zoom value such that the city specified in the search form is properly displayed in the map.

- You can set the location and zoom level using the setCenter() method of OpenLayers.Map object.
- When adding the cloud and precipitation layers, make sure the map is visible by adjusting the opacity of these layers appropriately.

6. Results Display

The results should be displayed below the form as shown in Figure 5. You are supposed to display the results responsive to mobile devices. If the page is loading on a smart phone or a tablet, the display should be modified according to the width of the devices.

The display of the results is divided into three tabs, viz., **Right Now, Next 24 Hours, Next 7 Days**. The bootstrap tab color should be customized to match the current theme as shown in Figure 5. The detailed description of all the tabs is given in the following sections.

Note: If any of the field in any of the tabs is unavailable in the returned JSON data, you should display "N.A." instead.

6.1. Tab 1: Right Now

The entire tab 1 area is divided into two sections. You must use Bootstrap for the area to make it responsive for mobile devices. These two sections get stacked vertically on mobile screens, the Current Weather table should be displayed at the top followed by the Cloud Map section.

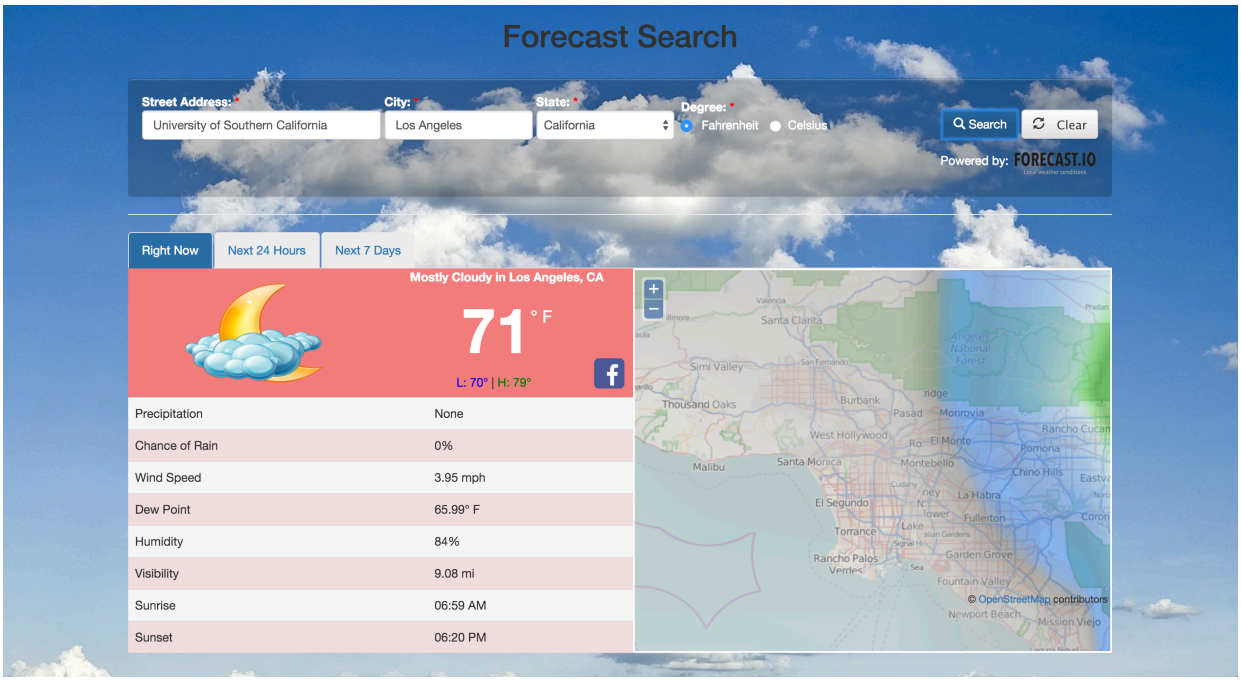

Figure 5: Results Display

6.1.1 Current Weather Table

To the left, the current weather data is displayed. The top part consists for two subsections with background color. The left subsection displays the icon image whereas the right subsection displays the "current" weather condition and location, current temperature and High/Low temperatures for the day. It also includes a Facebook button, the functionality of which is explained later in Section 6.4. The table below it displays additional weather data. The mapping of the weather data is shown in Section 6.1.3. If the page is loading on a smart phone, the information should be adjusted accordingly as shown in Figure 6 below.

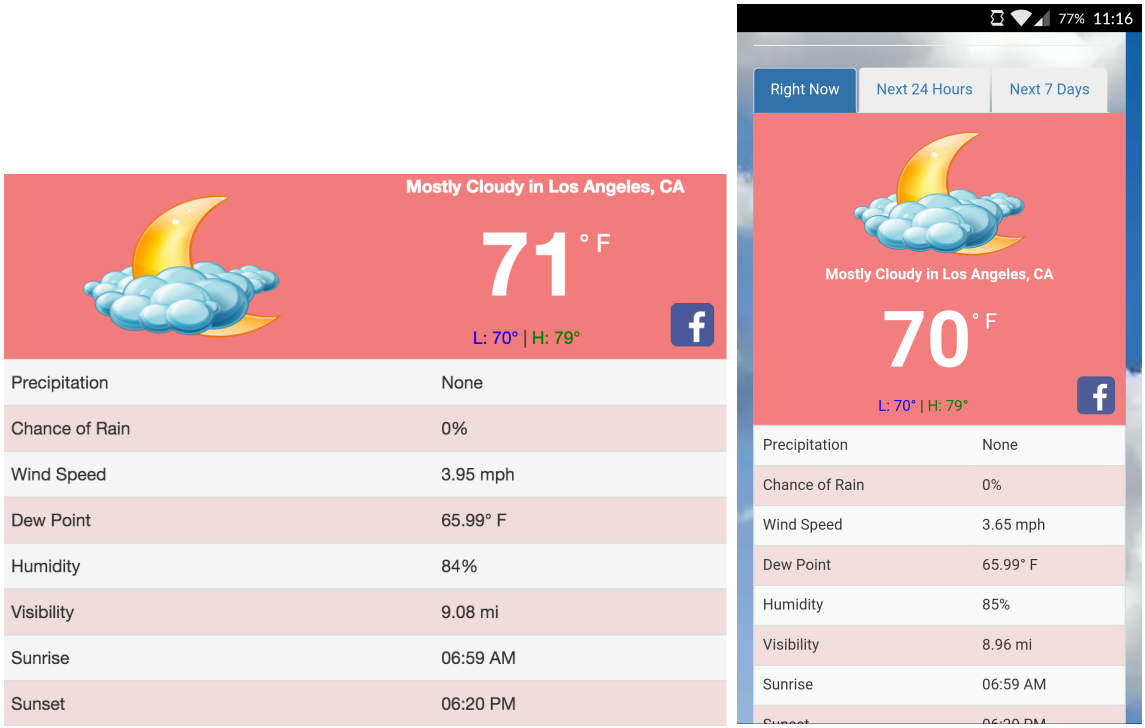

Figure 6: Current Weather table in Web and smartphone view

6.1.2 Cloud Map

To the right, the cloud map is displayed. The details on how to use the OpenWeather API for displaying the cloud map is explained in Section 5.2

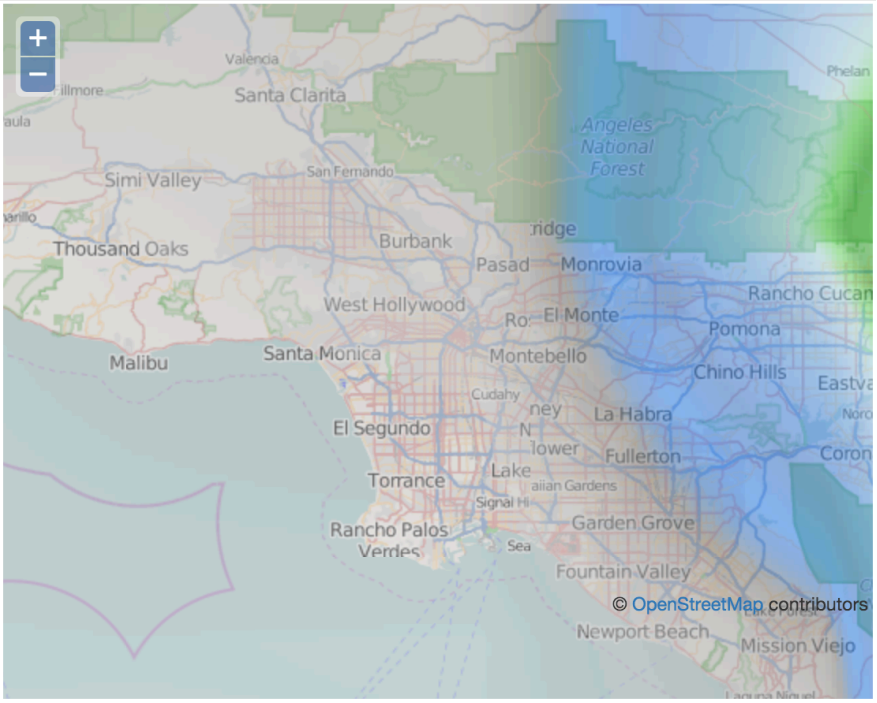

Figure 7: Cloud Map

6.1.3 Weather data mapping

Table Column	Data from results of Forecast.io API call
Top Left subsection on Tab 1	The displayed icon depends the value of *icon* in the *currently* object. • icon value = “clear-day” → display “clear.png” • icon value = “clear-night” → display clear_night.png” • icon value = “rain” → display “rain.png” • icon value = “snow” → display “snow.png” • icon value = “sleet” → display “sleet.png” • icon value = “wind” → display “wind.png” • icon value = “fog” → display “fog.png” • icon value = “cloudy” → display “cloudy.png” • icon value = “partly-cloudy-day” →

	display “cloud_day.png” ● icon value = “partly-cloudy-night” → display “cloud_night.png” The images are available at http://cs-server.usc.edu:45678/hw/hw8/images/. The Weather Icon must be displayed, and the alt text and title must be the text of the weather condition.
Top right subsection on Tab 1	The weather condition is the value of *summary* in the *currently* object. The location is formed using the form inputs City and State. The weather temperature is the value of *temperature* in the *currently* object. The temperature should be printed in integer format. The low temperature is obtained as *daily -> data[0] -> temperatureMax* The high temperature is obtained as *daily -> data[0] -> temperatureMin* The low and high temperature should be displayed in blue and green color respectively and in integer format. The right bottom corner contains Facebook icon, details of which are mentioned in Section 6.4
Precipitation	The value of *precipIntensity* in the *currently* object. ● 0 <= *precipIntensity* value < 0.002 → display “None” ● 0.002 <= *precipIntensity* value < 0.017 → display “Very Light” ● 0.017 <= *precipIntensity* value < 0.1 → display “Light” ● 0.1 <= *precipIntensity* value < 0.4 → display “Moderate” ● *precipIntensity* value >= 0.4 → display “Heavy”
Chance of Rain	The value of *precipProbability* in the *currently* object. You should multiply the value by 100 and display the integer value along with the percentage “%” character.

Wind Speed	The value of *windSpeed* in the *currently* object. The value should be limited to two decimals.
Dew Point	The value in degrees of *dewPoint* in the *currently* object. The value is limited to two decimals.
Humidity	The value of *humidity* in the *currently* object. You should display the value in percentage without decimals followed by "%" character.
Visibility	The value of *visibility* in the *currently* object. The value is limited to two decimals.
Sunrise	The value of *sunriseTime* is the first object of data array in the *daily* object. The value is a Unix timestamp so it needs to be converted to the "two-digits-hour:two-digits-minute AM/PM" format. The hour should be in 12-hour format. Example is 01:15 PM.
Sunset	The value of *sunsetTime* is the first object of data array in the *daily* object. The value is a Unix timestamp so it needs to be converted to the "two-digits-hour:two-digits-minute AM/PM" format. The hour should be in 12-hour format. Example is 01:15 PM.

You need to display proper units beside each value. The following unit mapping should be done according to the degree option (F or C) selected by the user.

Forecast metric	**Fahrenheit (units=en)**	**Celsius (units=si)**
temperature	°F	°C
windSpeed	mph	m/s
dewPoint	°F	°C
visibility	mi	km
pressure	mb	hPa

Note: pressure is not present in tab1 but is used in tab2 and tab3.

6.2. Tab 2: Next 24 Hours

This tab displays the weather information for the next 24 hours in a table format. The table has 5 columns, the details of which are described in the following table.

Table Column	Data from result of Forecast.io API call
Time	The value of *time* in the object of *data* array in the *hourly* object. The value is a Unix timestamp so it needs to be converted to the "two-digits-hour:two-digits-minute AM/PM" format. The hour should be in 12-hour format. Example is 10:00 AM.
Summary	The displayed icon depends the value of *icon* in the *data* array in the *hourly* object. The mapping of icon value to icon images is same as for tab1 given in Section 6.1.3.
Cloud Cover	The value of *cloudCover* in the *data* array in the *hourly* object. You should display the value in percentage without decimals followed by "%" character.
Temp	The value of *temperature* in the *data* array in the *hourly* object. The values are limited to two decimals by default. Note: The table header displays the unit of temperature.
View Details	A glyphicon symbol of + sign is displayed which on click displays a section below that row as shown in Figure 9. It should act as a toggle button meaning it will hide the display section below it if you click it again. The details of the view details section is given in the following table.

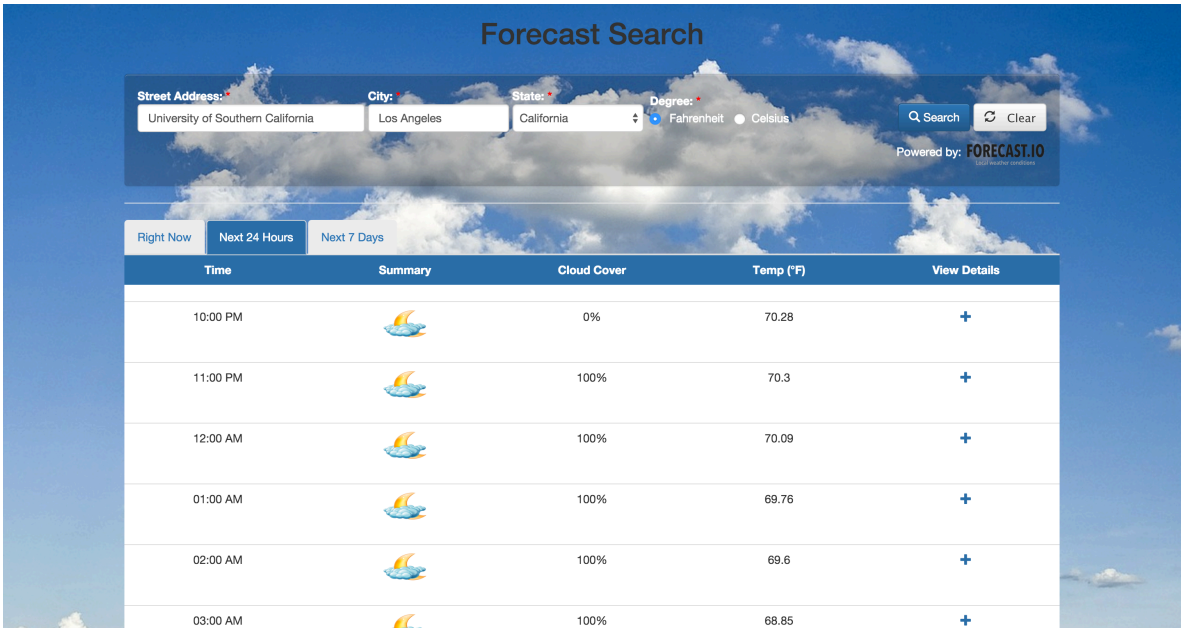

Figure 8: Tab2 result display

The view details table consists of 4 columns. This table should be responsive to mobile devices.

Table Column	Data from result of Forecast.io API call
Wind	The value of *windSpeed* in the *data* array in the *hourly* object.
Humidity	The value of *humidity* in the *data* array in the *hourly* object. You should display the value in percentage without decimals followed by "%" character.
Visibility	The value of *visibility* in the *data* array in the *hourly* object.
Pressure	The value of *pressure* in the *data* array in the *hourly* object.

Note: All the metric values are followed by the appropriate units. The units to be used are given in Section 6.1.3

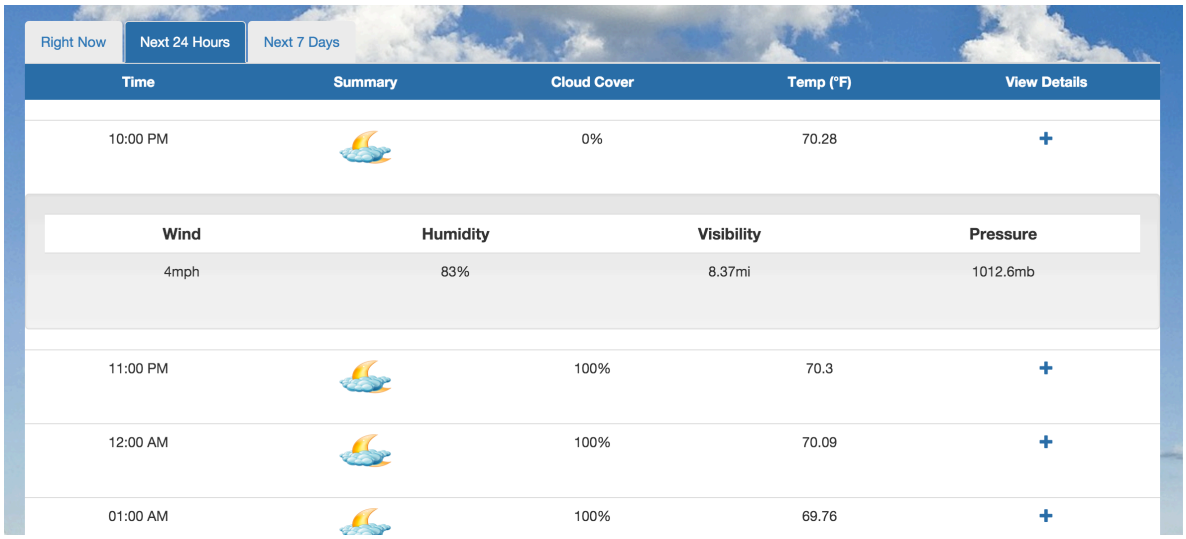

Figure 9: Tab 2 results showing the view details display

If the page is loading on a smart phone, the information should be adjusted accordingly as shown in Figure 10 below.

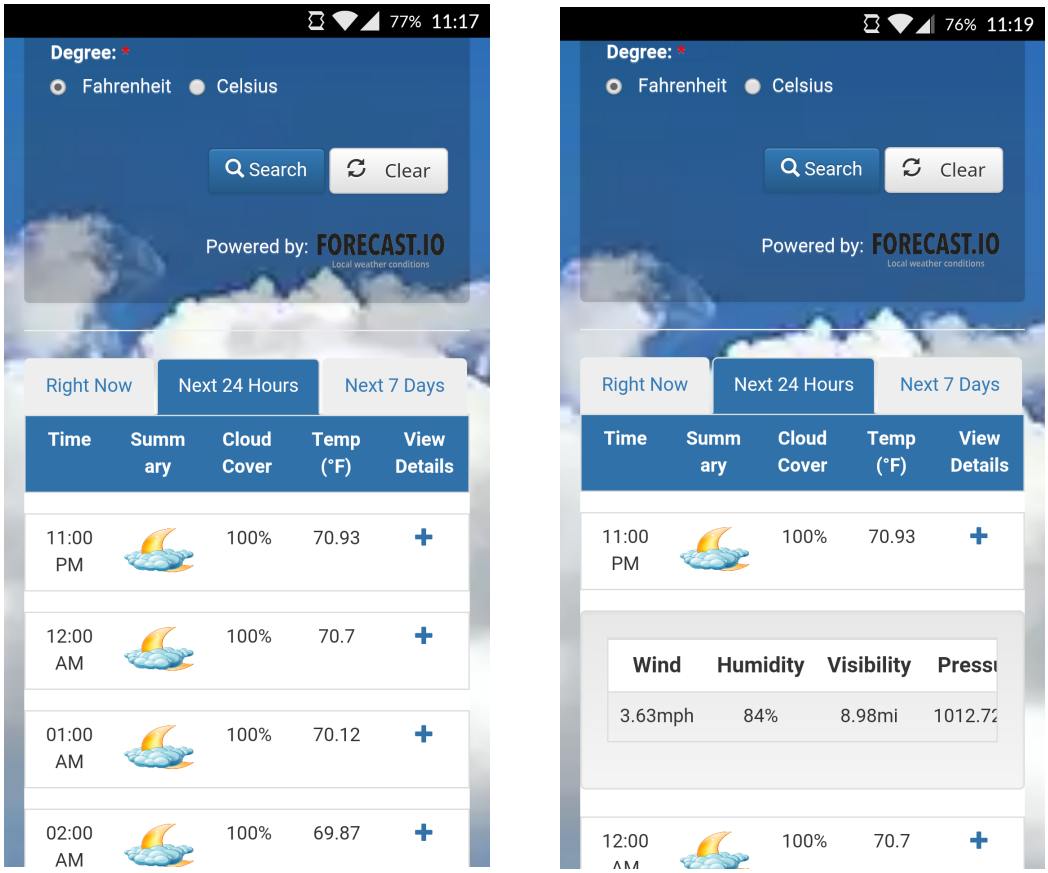

Figure 10: Tab 2 Smartphone view

6.3. Tab 3: Next 7 Days

This tab displays the forecast for the next 7 days in a column format as shown in Figure 11. The height of the columns should be fixed to accommodate all the details.

Note: The next 7 days starts from tomorrow and does not include today. So when fetching data from *daily* object, make sure to start from the second value of the array (index 1 and not index 0).

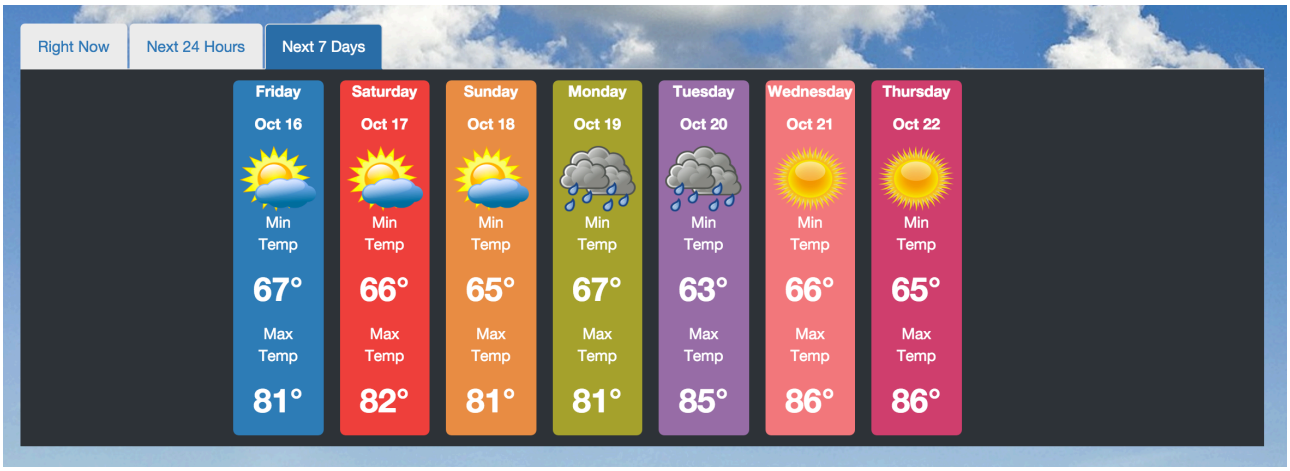

Figure 11: Tab 3 result display

The details for each of the columns is mentioned in the following table.

Data	**Data from results of Forecast.io API call**
Day	The value of Day is obtained using the value of *time* in the *data* array in the *daily* object. The value is a Unix timestamp so it needs to be converted to get the day. The day should be displayed in full and not short hand notation. Example: Wednesday (and not Wed)
Month Date	This value is obtained using the value of *time* in the *data* array in the *daily* object. The value is a Unix timestamp so it needs to be converted to get the month and date. The month should be displayed in short hand notation. Example: Nov 10
Icon image	The displayed icon depends the value of *icon* in the *data* array in the *daily* object. The mapping of icon value to icon images is same as for tab1 shown in Section 6.1.3. The icon images should be responsive.
Min Temp, Max Temp	The values of *temperatureMin* and *temperatureMax* respectively in the *data* array in the *daily* object. The temperature should be printed in integer format, followed by a degree (°) symbol.

All the 7 columns should respond to a click. If you click anywhere on a particular column, it should display a modal window as shown in Figure 12.

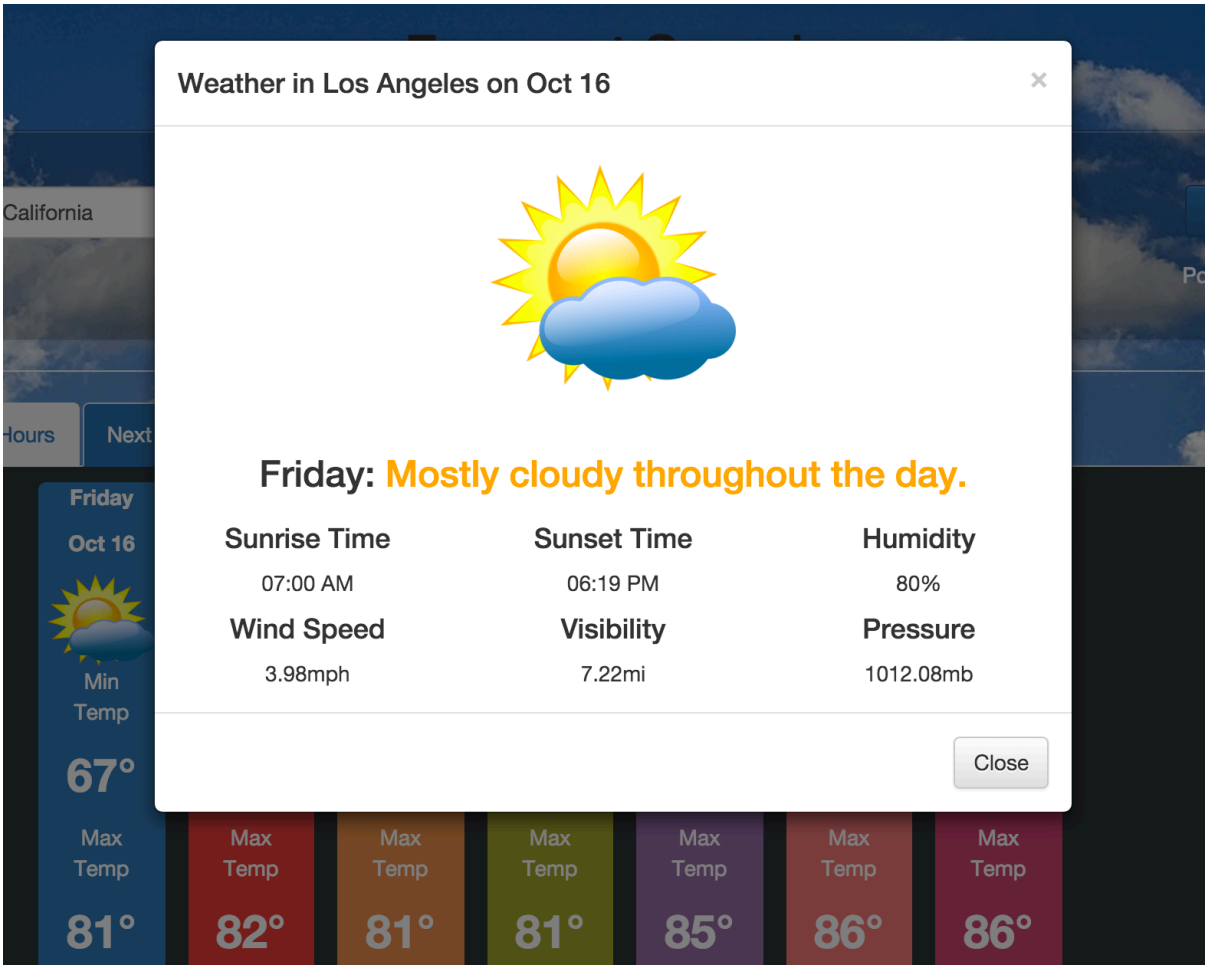

Figure 12: Tab3 Modal display

The details of the modal window are mentioned in the following table.

Data	**Data from result of Forecast.io API call**
Header	The header includes the city name obtained from the form input for City. It also includes Month Date of the particular column clicked.
Icon image	The icon image is the same icon image of the particular column clicked. This image should be aligned in center horizontally.
Summary line	It starts with the Day of the column clicked. It is followed by the value of *summary* in the *data* array in the *daily* object. The summary line has a greater font size than the rest of the font used in the modal window.
Sunrise Time, Sunset Time, Humidity	The values of *sunriseTime* and *sunsetTime* respectively in the *data* array in the *daily* object. These Unix timestamp values need to be converted to "two-digits-hour:two-digits-minute AM/PM" format. The value of *humidity* in the *data* array in the *daily* object, you should display the value in percentage

	without decimals followed by "%" character.
Wind Speed, Visibility, Pressure	The value of *windSpeed, visibility* and *pressure* respectively in the *data* array in the *daily* object. The values should be followed by the appropriate units.
Footer	The footer should only have one Close button to dismiss the window.

The tab 3 and modal window appear on a mobile device as shown in Figure 13. If the page is loading on a smart phone or a tablet, the tab 3 columns gets stacked up vertically. The background should be transparent in the mobile view with rounded edges. The three column table-like display for the details in the modal gets stacked vertically. An example is shown in Figure 13 below.

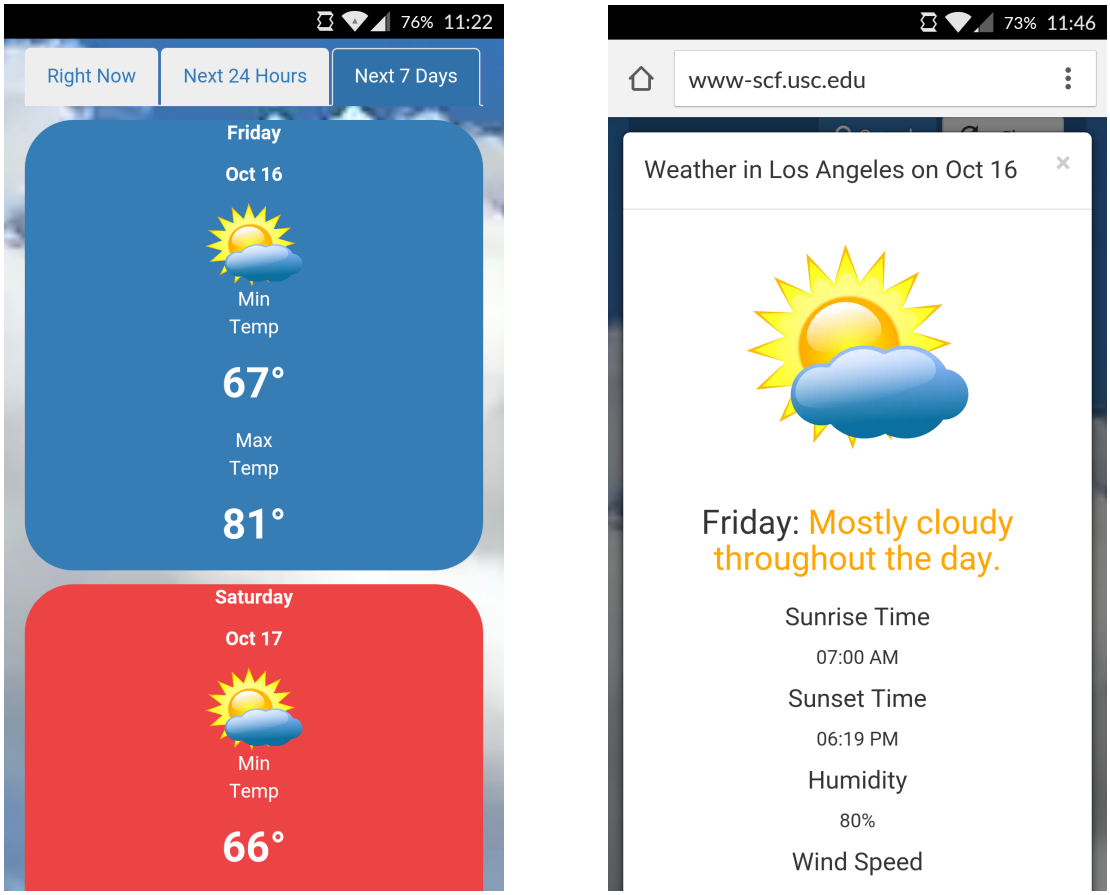

Figure 13: Tab3 Smartphone view

6.4. Facebook post

On click of the Facebook post button, a Facebook post must be done with the format shown in Figure 15.

In particular, when the button is pressed, the web application does the following:

- Displays a popup to authorize the user to Facebook (i.e. logs him/her in) using the application and user credentials if the user is not already logged in to Facebook;

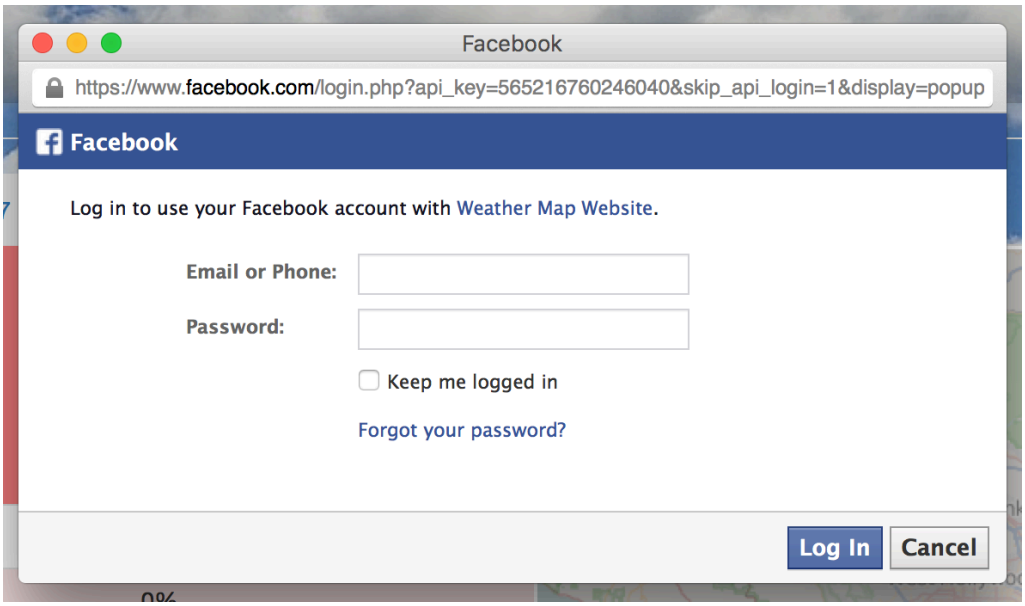

Figure 14: Facebook Login

- Posts an Update Status message to the feed;
- The above two steps can be performed using the Facebook Connect API, using the JavaScript SDK, which provides a rich set of client-side functionality for accessing Facebook's server-side API calls. It is documented at:

  https://developers.facebook.com/docs/reference/javascript/

Display the appropriate values for title, caption, description and icon image as shown in Figure 15. The hyperlink on the post should directly link to http://forecast.io/. The image for the Facebook button can be obtained from http://cs-server.usc.edu:45678/hw/hw8/images/fb_icon.png.

Figure 15: Post to Facebook

Once the post has been published, you should show an alert box informing the user of whether the post has been published successfully or not. For example:

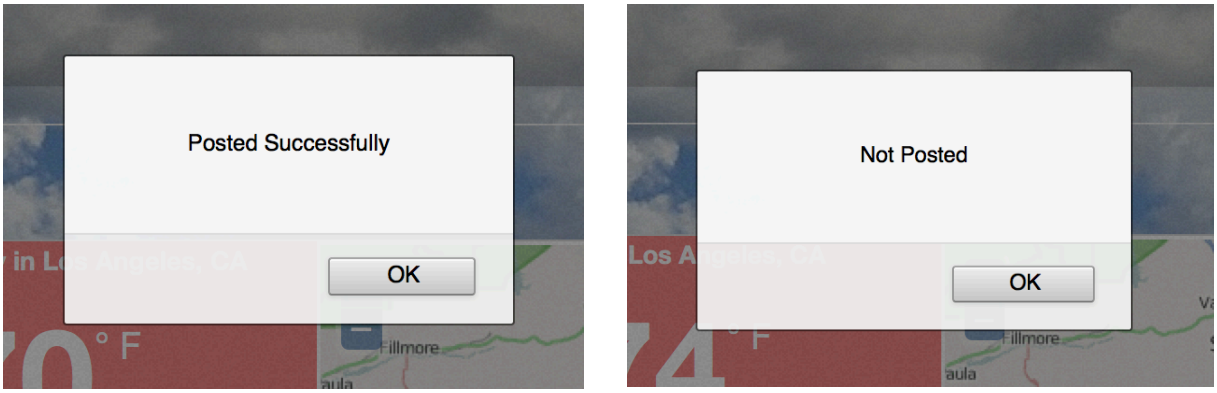

Figure 16: Post status Alert box

7. Implementation Hints

- **Get started with the Bootstrap Library**
  To know how to get started with Bootstrap, please refer to the link http://getbootstrap.com/getting-started/. You need to import the necessary CSS file and JS file provided by Bootstrap.

- **jQuery Validation**

You are required to use jQuery validation together with Bootstrap to validate the user input. Please refer to the links below for more details about the implementation of this functionality.

http://jqueryvalidation.org/
http://getbootstrap.com/css/#forms-control-validation

- **Bootstrap UI Components**
  Bootstrap provides a complete mechanism to make Web pages responsive to different mobile devices. In this exercise you will get hands-on experience with responsive design using the Bootstrap Grid System. You will at a minimum need to use Bootstrap Form, Tab, Modal, Collapse and Glyphicons to implement the required functionalities.

  Bootstrap Form http://getbootstrap.com/css/#forms
  Bootstrap Tabs http://getbootstrap.com/javascript/#tabs
  Bootstrap Modal http://getbootstrap.com/javascript/#modals
  Bootstrap Collapse http://getbootstrap.com/javascript/#collapse
  Bootstrap Glyphicons http://getbootstrap.com/components/#glyphicons

- **AWS**
  You should use the domain name of the AWS server you created in HW#7 to make the request. For example, if your AWS server domain is called default-environment-randomstring.elasticbeanstalk.com, and the user performs a GET request with parameter name="John Smith", then a query of the following type will be generated:

  http://default-environment-randomstring.elasticbeanstalk.com/?name= John%20 Smith

- **AJAX call**
  You can send the request to the PHP file by passing the URL to $.ajax(). You must use a GET method to request the resource since you are required to provide this link to your homework list to let graders check whether the PHP code is running on Amazon AWS. (Please refer to the grading guideline for details).

  The AJAX call:

```
$.ajax({
        url: 'URL you created in HW#7',
        // this is the parameter list
        data: { name: "John Smith",
                age: "23"
                },
        type: 'GET',
        success: function(output) {
                        // parse the data here
                },
```

```
            error: function(){
                }
        });
```

Files to Submit

On your course homework page, you should update the HW8 link to refer to your new initial web page. Also, Submit your files electronically to the csci571 account so that they can be graded and compared to all other students files.

IMPORTANT:

All discussions and explanations in Piazza related to this homework become part of the homework description. So please review all Piazza threads before finishing the assignment.

In your Facebook application settings, you should go to the "Status & Review" section and choose "Yes" for the question "***Do you want to make this app and all its live features available to the general public***?" as shown in Figure 17. If you answered "YES", anyone will be able to log in through her/his Facebook account and post via your web interface. Otherwise, the developer will be the only person who is able to use the Facebook functionality in the web page. If graders try to test the functionality of the Facebook button and they are not able to log in you will lose **all** points related to the Facebook component.

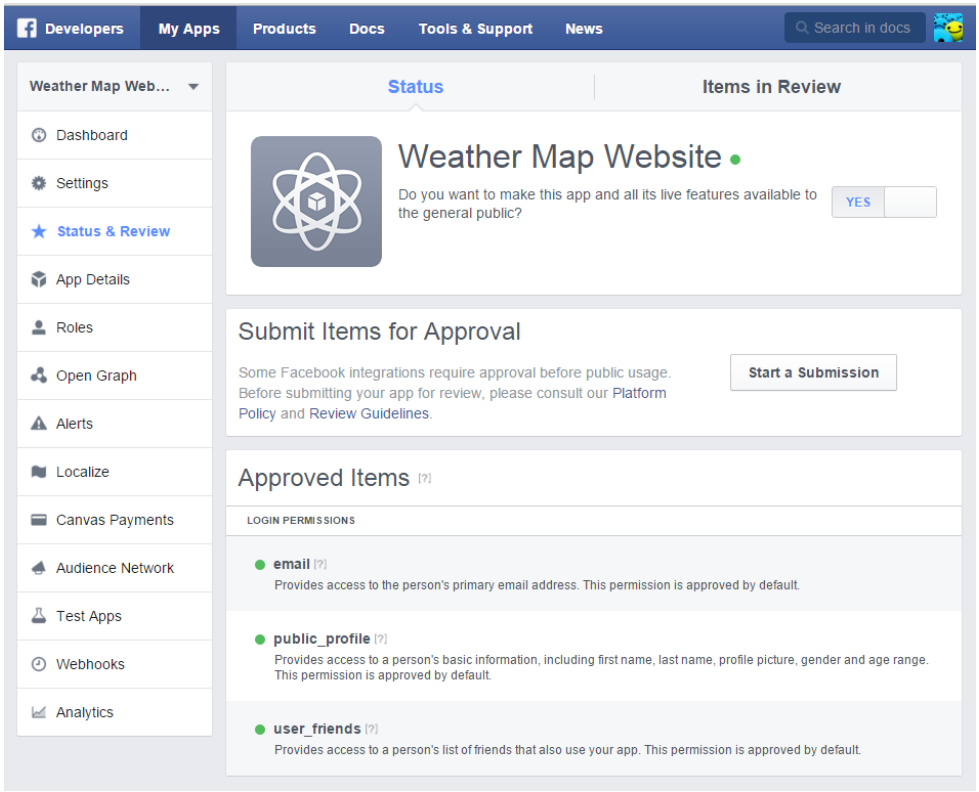

Figure 17: Status & Review of Facebook Application