Homework 8: Ajax, JSON, Responsive Design and Node.js

Weather Search

(AJAX/JSON/HTML5/Bootstrap/Angular/Node.js/Cloud Exercise)

1. Objectives

- Get familiar with the AJAX and JSON technologies.
- Use a combination of HTML5, Bootstrap, and Angular on client side.
- Learn to use JavaScript / Node.js on server side.
- Get familiar with Bootstrap to enhance the user experience using responsive design.
- Get hands-on experience of Amazon Web Services/Google Cloud App Engine/Microsoft Azure.
- Learn to use APIs such as Forecast.io API, Google Customized Search API and Twitter API.

2. Background

2.1 AJAX and JSON

AJAX (Asynchronous JavaScript + XML) incorporates several technologies:

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

See the class slides at https://csci571.com/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:

https://csci571.com/slides/JSON1.pdf

2.2 Bootstrap

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). Please use **Bootstrap 4** in this homework. See the class slides at:

https://csci571.com/slides/Responsive.pdf

2.3 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 for .NET, and Apache Tomcat for Java.

To learn more about AWS support for Node.js visit this page:

https://aws.amazon.com/getting-started/projects/deploy-nodejs-web-app/

2.4 Google App Engine (GAE)

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go. App Engine applications automatically scale based on incoming traffic. Load balancing, micro services, authorization, SQL and noSQL databases, memcache, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

To learn more about GAE support for Node.js visit this page:

https://cloud.google.com/appengine/docs/flexible/Node.js/

2.5 Microsoft Azure

Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems.

To learn more about Azure support for Node.js visit this page:

https://docs.microsoft.com/en-us/javascript/azure/?view=azure-node-latest

2.6 Angular

Angular is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit your unique development workflow and feature needs. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

For this homework, Angular 2+ (Angular 2,4,5,6 or 7) can be used, but **Angular 7 is recommended**. However, please note Angular 2+ will be a little more difficult to learn if the developer is not familiar with Typescript and component-based programming.

To learn more about Angular 2+, visit this page:

https://angular.io/

Note: AngularJS cannot be used in this project.

2.7 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.

To learn more about Node.js, visit:

https://Node.js.org/en/

Also, **Express.js** is strongly recommended. Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is in fact the standard server framework for Node.js.

To learn more about Express.js, visit:

http://expressjs.com/

Note: In this document when you see GAE/AWS/Azure it means that you can either use Google App Engine, Amazon Web Services or Microsoft Azure Services.

The only way to implement the client side: Use entirely Angular2+

All APIs calls should be done through your Node.JS server, except call to the ip-api.

3. High Level Description

In this exercise you will create a webpage that allows users to search for weather information using the Forecast.io API and display the results on the same page below the form.

The user provides the location information such as Street address, City and State for which they would want to find the detailed weather information or provide their current location.

Once the user has provided the data and clicks on the Search button, validation must be done to check that the entered data is valid. Once the validation is successful, 3 tabs should be displayed. The 3 tabs correspond to Current tab, Hourly tab and Weekly tab.

Your webpage should also support adding cities to and removing cities from the Favorites tab and sharing the weather info with Twitter. All the implementation details and requirements will be explained in the following sections.

When a user initially opens your webpage, your page should look like Figure 1.

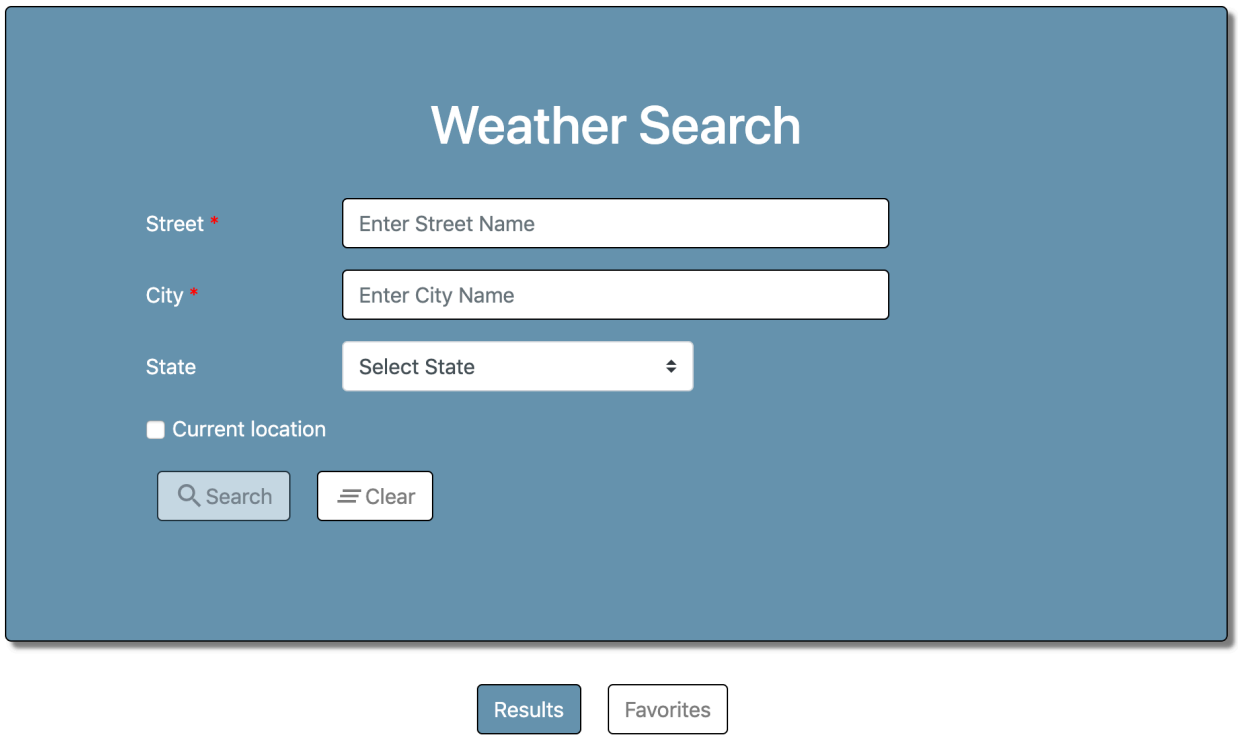


Figure .1 Initial Search Form

3.1 Search Form

3.1.1 Design

You must replicate the search form displayed in Figure 1 using a **Bootstrap form**. The form fields are similar to Homework #6.

There are 3 fields in the search form which are **required** if the **Current Location** is not checked:

1. **Street**: Initially, please show the placeholder shown in the picture.
2. **City**: This input field should support autocomplete which is explained in section 3.1.2. Please note that the user does not necessarily chooses what suggested by the autocomplete. Initially, please show the placeholder shown in the picture.
3. **State**: There are multiple options for the user to select from which contain all the States of US.

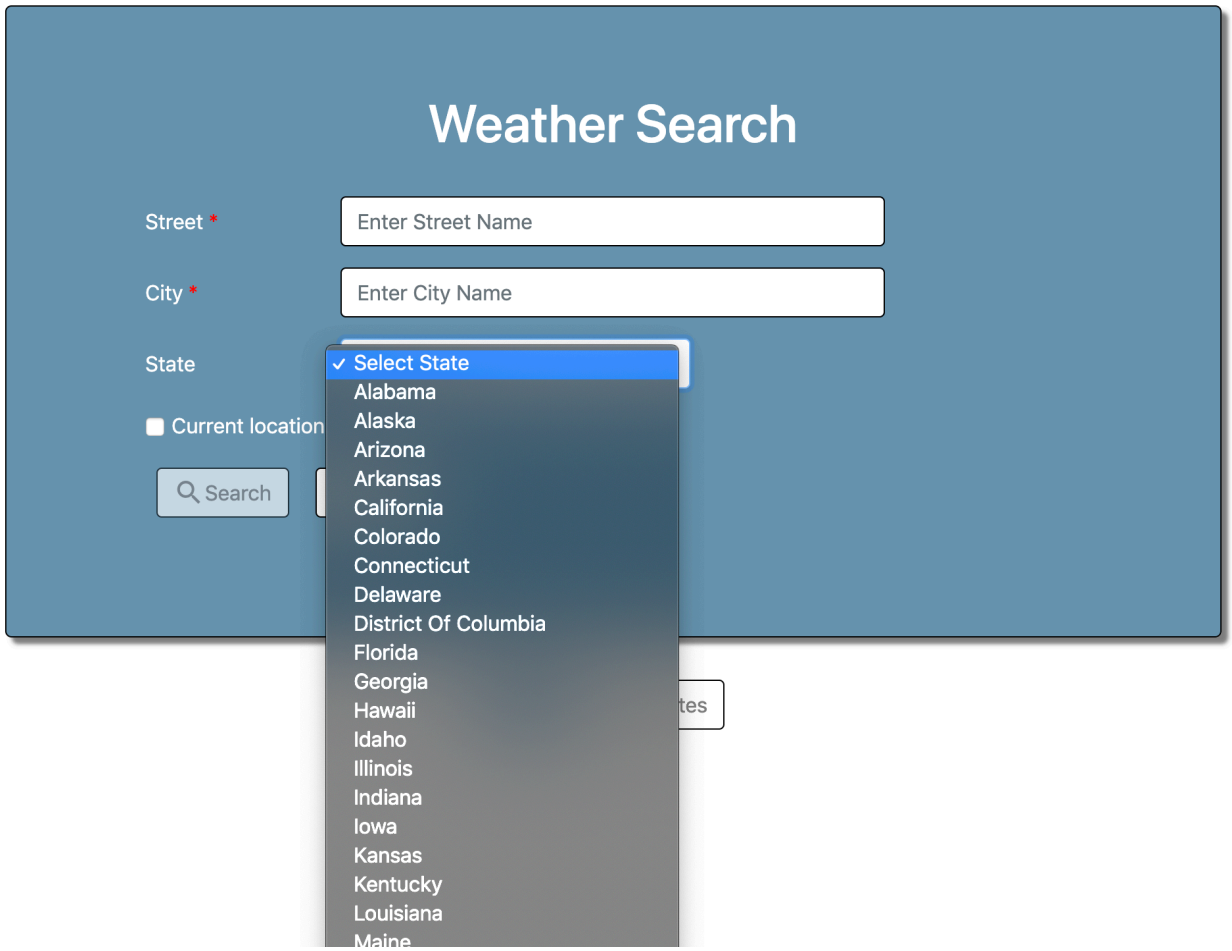


Figure 2. State Options

The search form has two buttons:

1. **Search**: The "Search" button should be disabled whenever either of the required fields is empty or validation fails, or the user location is not obtained yet. Please refer to section 3.1.3 for details of validation. Please refer to section 3.1.4 for details of obtaining user location.
2. **Clear**: This button must reset the form fields, clear all validation errors if present, switch the view to the results tab and clear the results area.

3.1.2 AutoComplete

Autocomplete for Cites is implemented by using the Place Autocomplete service provided by Google. Please go to this page to learn more about this service:

https://developers.google.com/places/web-service/autocomplete

An example of an HTTP request to the *Places Autocomplete API* Get Suggestion that searches for the city textfield "Los" is shown below:

https://maps.googleapis.com/maps/api/place/autocomplete/json?input=LOS&types=(cities)&language=en&key=[Your_API_Key]

To get the API key in the above URL:

1. Please follow the steps provided in the following link: https://developers.google.com/places/web-service/get-api-key

Note:

1. Set the **type** to cities to provide suggestions for cities.
2. Set the **language** to en for specifying that the language is English.
3. **If the API doesn't work, hide the autocomplete feature. This situation should be handled and not throw any error.**

The response is a JSON object similar to the one shown in Figure 3.

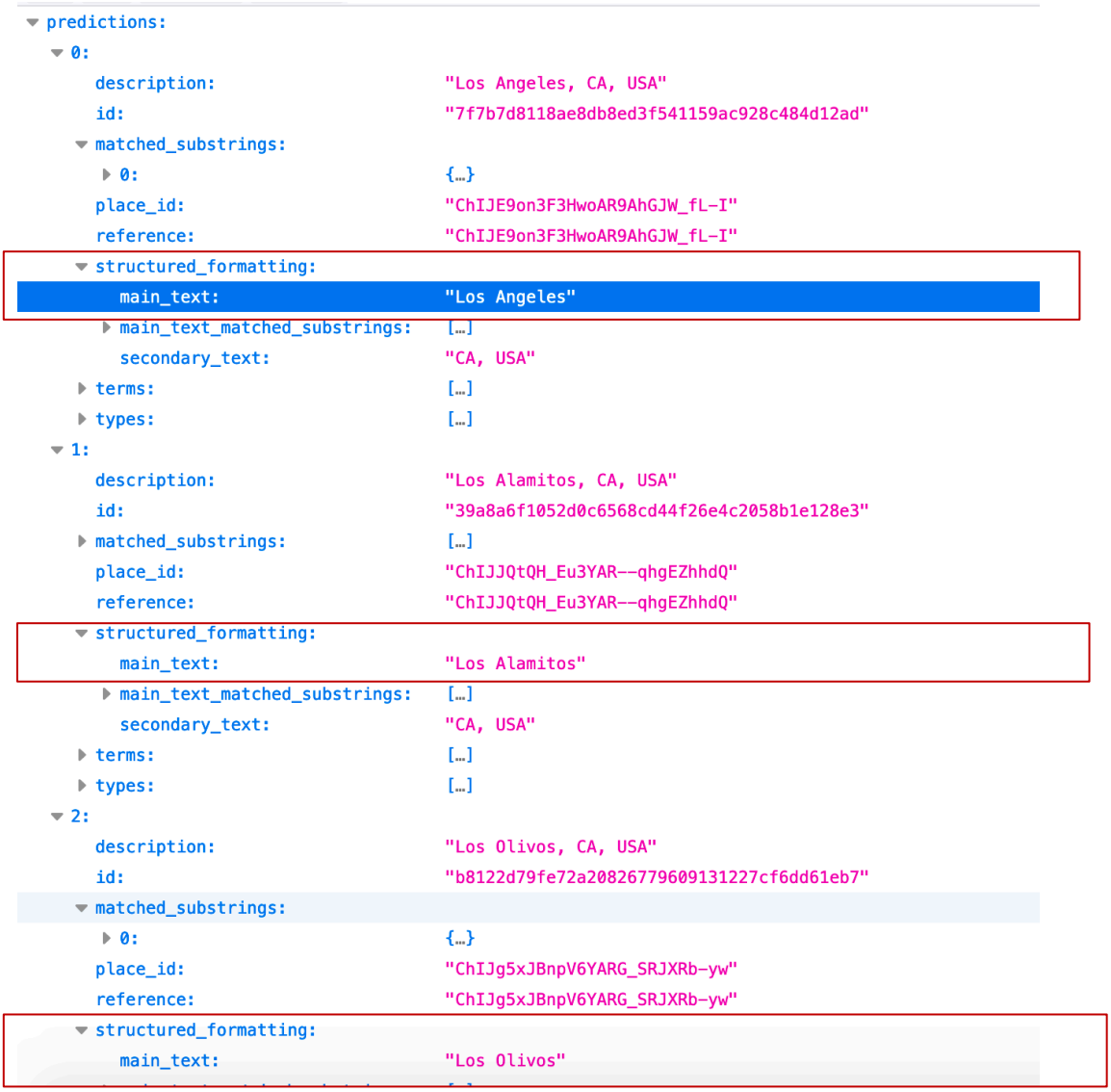


Figure 3. Autocomplete JSON Response

The fields to get the suggested options are highlighted in red in the JSON response above.

For reference: The value is provided by the key **main_text** in the **structured_formatting** object in the **predictions** array.

You must use **Angular Material** to implement the Autocomplete. (See section 6.4).

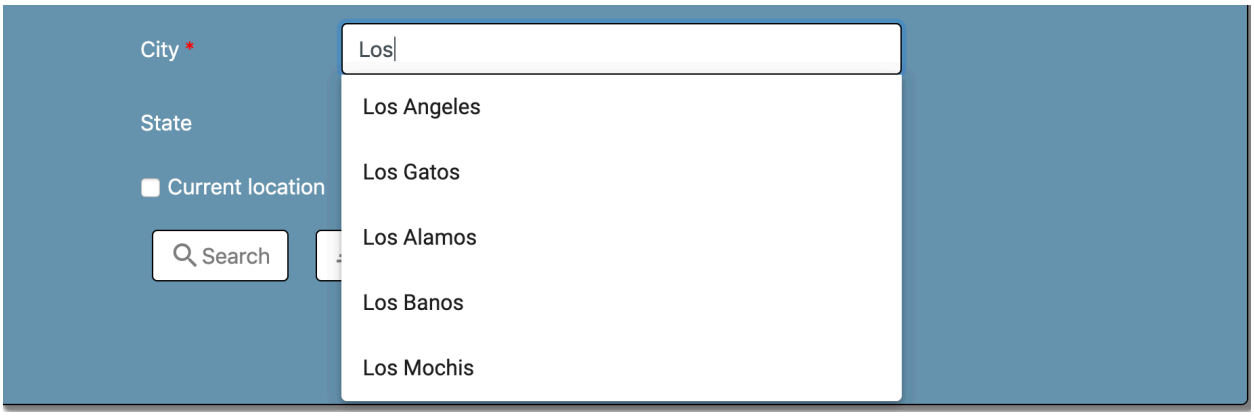


Figure 4. Autocomplete example

Note: Restrict the number of options in the "autocomplete" suggestion to 5.

3.1.3 Validation

Your application should check if the "Street" and "City" edit boxes contain something other than spaces or blank. If not, then it's invalid and an error message should be shown as in Figure 5.

If the Current Location checkbox is checked, then the Street, City and the State controls should be disabled with their values retained.

Please watch the reference video carefully to understand the validation.

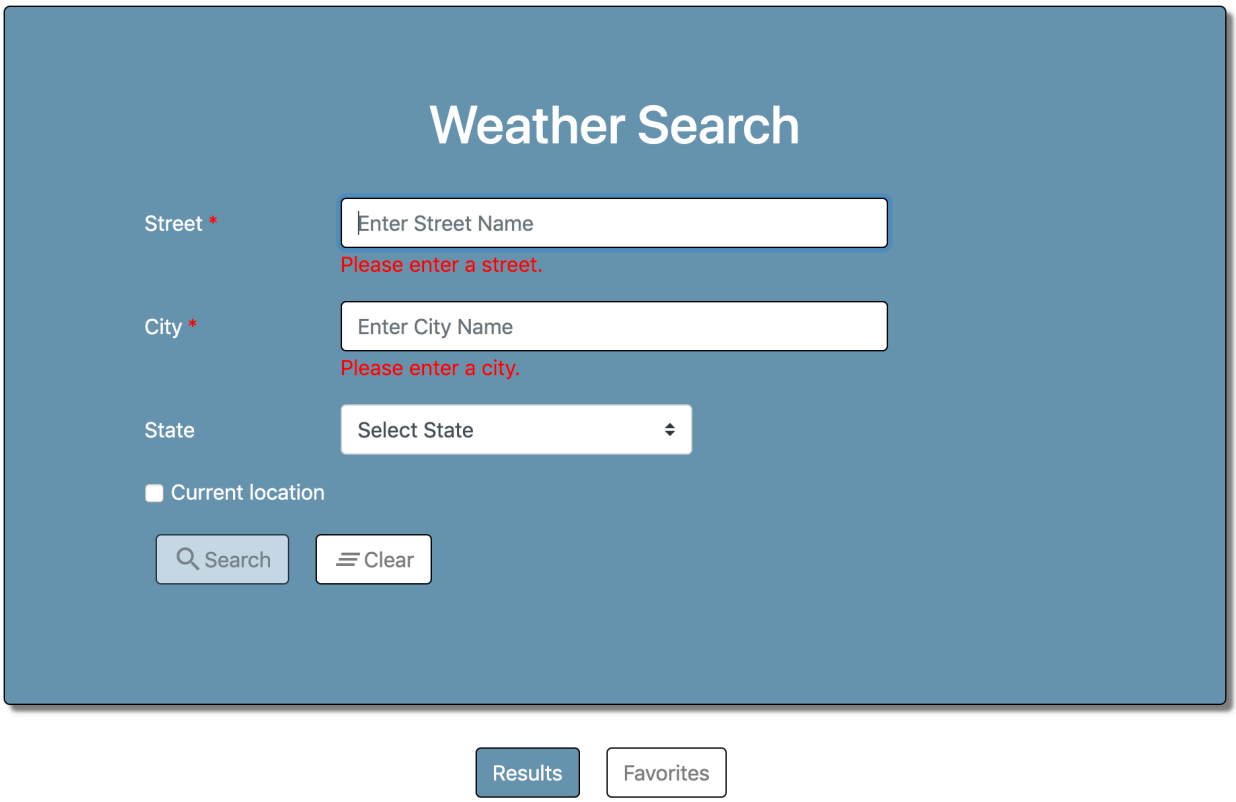


Figure 5. An Invalid Form

3.1.4 Obtaining User Location

As in Homework #6, you should obtain the current user location using one of the geolocation APIs. The usage of this API has been explained in the Homework #6 documents.

http://ip-api.com/json

3.1.5 Search Execution

Once the validation is successful and the user clicks on "Search" button, your application should make an AJAX call to the Node.js script hosted on GAE/AWS/Azure. The Node.js script on GAE/AWS/Azure will then make a request to forecast.io API to get the weather information. This will be explained in the next section.

3.2 Results Tab

The results should be displayed below the form as shown in Figure 6. You are supposed to display the results in a responsive mode compatible with mobile devices. If the page is loading on a smart phone or a tablet, the display should be modified according to the width and height of the devices.

In this section, we outline how to use the form inputs to construct HTTP requests to the Google geocode API and DarkSky API services and display the result in the webpage.

The *Google geocode API* call looks like this:

https://maps.googleapis.com/maps/api/geocode/json?address=[STREET, CITY, STATE]&key=[YOUR_API_KEY]

Note:

1. We use JSON response to get the latitude and longitude from the Google geocode API.
2. Please refer to the Homework 6 documentation to get the Google geocode API key.

Once the latitude and longitude are fetched from the Current Location or the Google geocode API, it has to be passed to the DarkSky API to fetch the weather details.

The *DarkSky API* is documented here:

https://darksky.net/dev/docs

The usage of these two APIs has been explained in the Homework #6 documents.

An example of an HTTP request to the DarkSky API that searches for the weather information from the user's current location or the location provided by the user is shown below:

https://api.darksky.net/forecast/[key]/[latitude],[longitude]

Note: Important changes in the URL from Homework 6

1. **No need to exclude any parameters. Call the API as given above.**

The Node.js script should pass the JSON object returned by the *API* to the client side or parse the returned JSON and extract useful fields and pass these fields to the client side in **JSON format**. You should use TypeScript to parse the JSON object and display the results using 3 tabs. A sample output is shown in Figure 6. See the tabs on upper left.

The display of the results is divided into 3 tabs, i.e. **Current** tab**, Hourly** tab **and Weekly** tab. The detailed description of all the tabs is given in the following sections.

3.2.1 Current Tab

This tab is in a card layout which provides the details of the Current weather for that day at the corresponding location provided by the user.

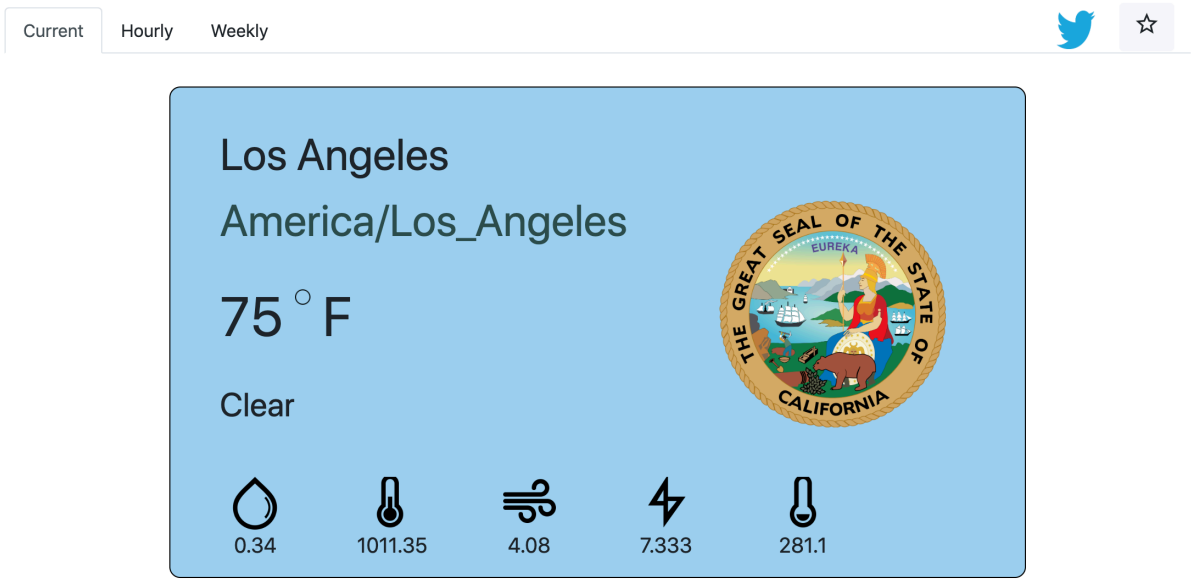


Figure 6. An Example of a Valid Search result

When the search result contains at least one record, you need to map the data extracted from the API results to the card layout to render the HTML result as shown in Figure 6.

Figure 7. Card view of Current Weather with Tooltip

On hovering the icons, a tooltip should be displayed for that corresponding icon, please use Angular Material or bootstrap tooltip to implement this. The style between bootstrap tooltip and Angular Material tooltip will be a slightly different, but both are fine. (See section 6.3 and 6.4)

The details for the Card are from the corresponding JSON structure:


```
{
    "latitude": 33.9147445,
    "longitude": -118.2827372,
    "timezone": "America/Los_Angeles",
    "currently": {
        "time": 1569263535,
        "summary": "Mostly Cloudy",
        "icon": "partly-cloudy-day",
        "nearestStormDistance": 0,
        "precipIntensity": 0,
        "precipProbability": 0,
        "temperature": 74.21,
        "apparentTemperature": 74.62,
        "dewPoint": 63.8,
        "humidity": 0.7,
        "pressure": 1013.83,
        "windSpeed": 5.64,
        "windGust": 8.41,
        "windBearing": 195,
        "cloudCover": 0.74,
        "uvIndex": 5,
        "visibility": 10,
        "ozone": 293.3
    },
```


Figure 8: Sample JSON-formatted object returned from Forecast.io API call for the card layout

Table Column	**Data from result of Forecast.io API call**
City	The value of the city can be taken from the input form if the user provides the address or from ip-api to get current location.
Timezone	The value of the key "*timezone*".
Temperature	The value of the key *"temperature"* in the *currently* object. You should display the value with a degree sign and F to indicate temperature in Fahrenheit. For the degree sign use this image - https://cdn3.iconfinder.com/data/icons/virtual-notebook/16/button_shape_oval-512.png
Summary	The value of *"summary"* in the *currently* object.
Humidity	The value of "*humidity"* in the *currently* object. It should also be provided with the corresponding icon - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-16-512.png
Pressure	The value of "*pressure"* in the *currently* object. It should also be provided with the corresponding icon - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-25-512.png
Wind Speed	The value of *"windSpeed"* in the *currently* object. It should also be provided with the corresponding icon for it - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-27-512.png
Visibility	The value of "*visibility"* in the *currently* object. It should also be provided with the corresponding icon for it - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-30-512.png
CloudCover	The value of "*cloudCover"* in the *currently* object. It should also be provided with the corresponding icon - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-28-512.png

Ozone	The value of "*ozone"* in the *currently* object. It should also be provided with the corresponding icon for it - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-24-512.png

Table 1: Mapping the result from DarkSky API into card layout

The state seal provided in the card is provided by the Google Custom Search API.

The Google Custom Search Engine is documented at:

https://developers.google.com/custom-search/json-api/v1/overview

To retrieve a photo of the State Seal, the request needs 6 parameters (output should be JSON):

- **q**: The search expression
- **cx**: The custom search engine ID to use for this request.
- **imgSize**: Returns images of a specified size.
- **num**: Number of search results to return. (Valid values are integers between 1 and 10, inclusive.). **Please specify 1**.
- **searchType**: Specifies the search type: **image**. If unspecified, results are limited to webpages.
- **key**: Your application's API key. This key identifies your application for purposes of quota management.

An example of an HTTP request to the Google custom search API is shown below:

https://www.googleapis.com/customsearch/v1?q=[STATE]%20State%20Seal&cx=[YOUR_SEARCH_ENGINE_ID]&imgSize=huge&imgType=news&num=1&searchType=image&key=[YOUR_API_KEY]

The JSON response from the Google Custom Search:

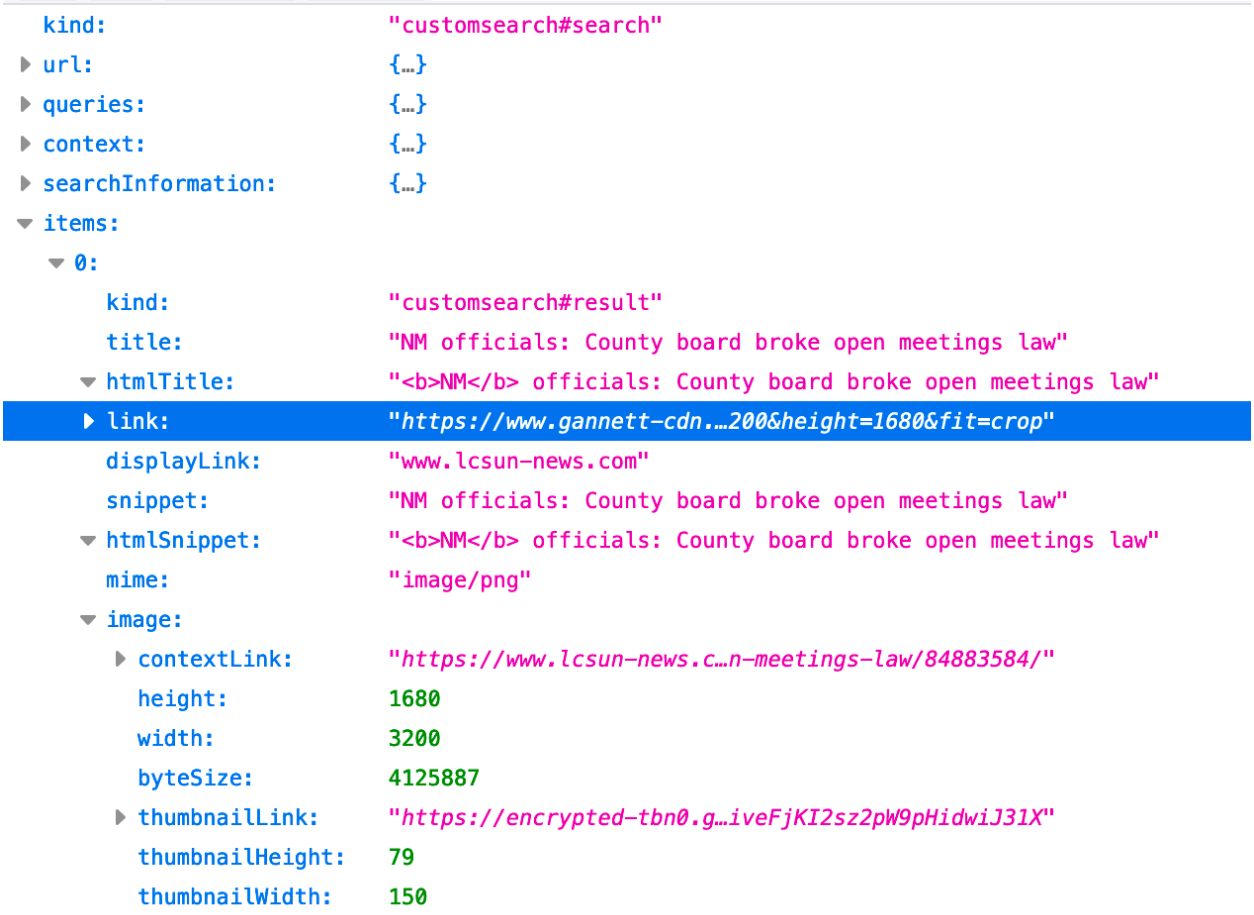


Figure 8. Google Custom Search API response

When the search result contains at least one record, you need to map the data extracted from the API results to the seal symbol.

State Seal in Card	API service response
State Seal	You should display 1 photo, which is present in "*link*" attribute inside the '*items*' array.

Table 2: Mapping the result from Google Custom Search API

Note: If any value is 0 or missing in the first card please do not display them.

3.2.2 Hourly Tab

This tab provides a bar chart for all the parameters of the weather. There are 6 parameters:

1. Temperature (Fahrenheit) vs Time (Hourly).
2. Pressure (Millibars) vs Time (Hourly).

3. Humidity (%) vs Time (Hourly).
4. Ozone (Dobson Units) vs Time (Hourly).
5. Visibility (Miles) vs Time (Hourly).
6. Wind Speed (Miles per hour) vs Time (Hourly).

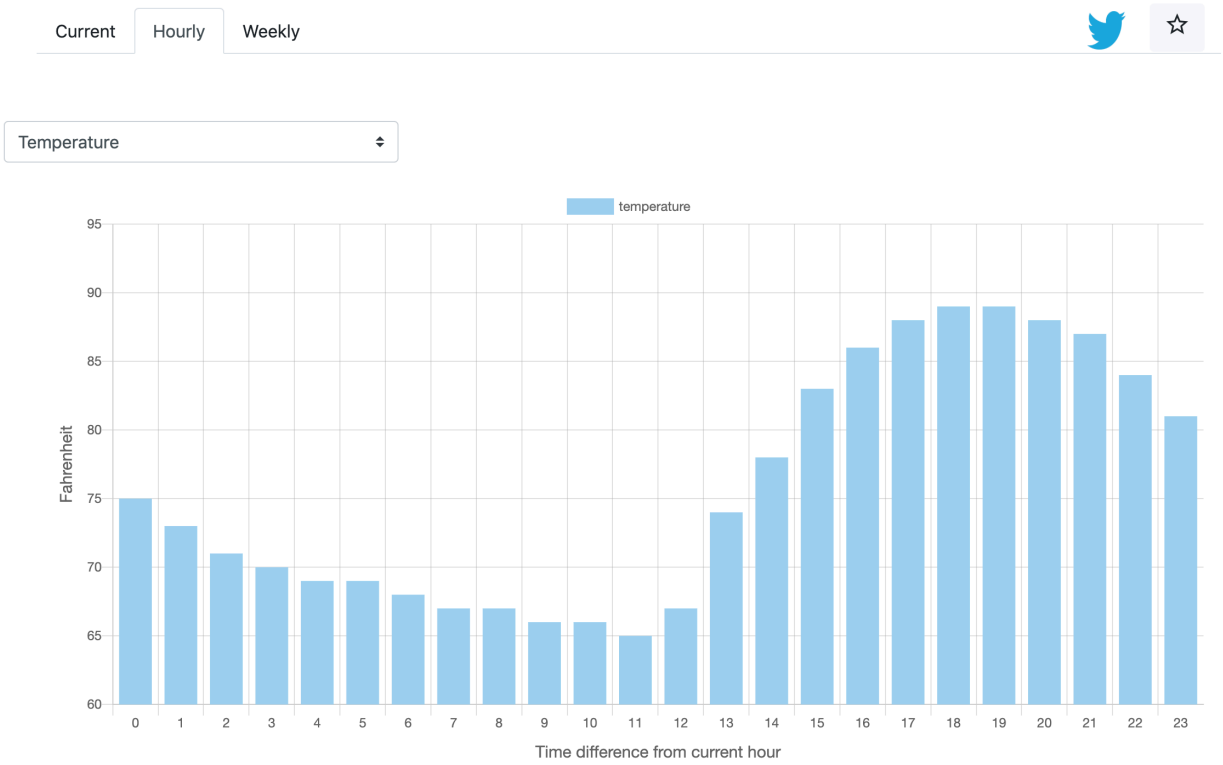


Figure 9 Hourly tab with Temperature vs Time bar chart

There are 6 charts corresponding to the parameter, which is provided by the dropdown menu.

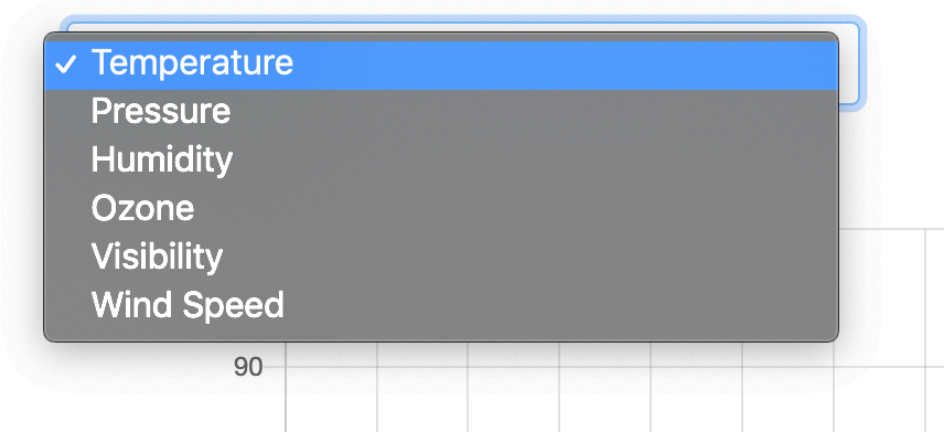


Figure 10. Dropdown for the type of Parameters

The hourly details for the chart are provided by the *DarkSky* API. The following JSON structure provides the data:

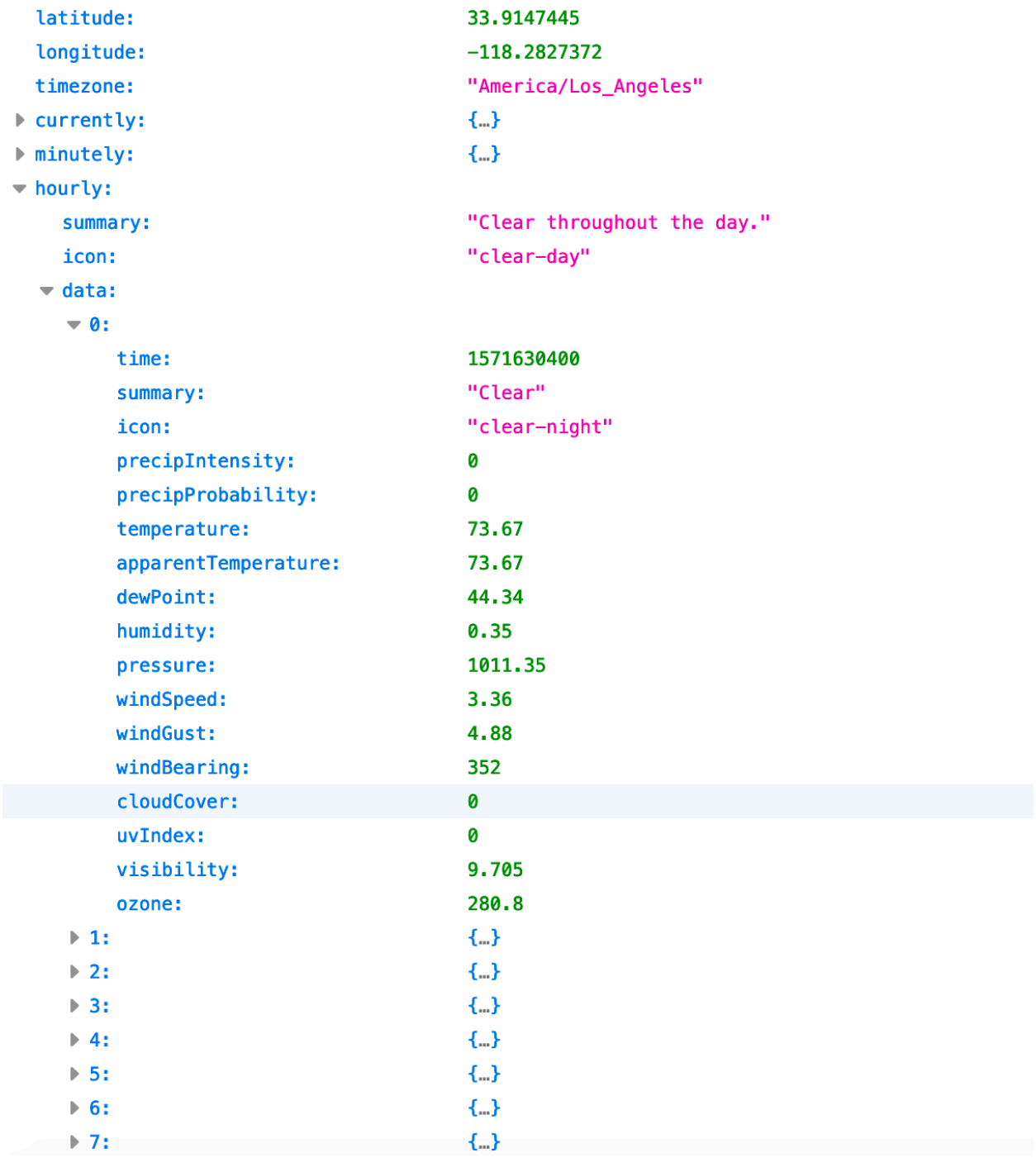


Figure 11: A sample JSON response from DarkSky API to get the hourly data.

Bar Chart	**Data from result of Forecast.io API call**
Temperature vs Time	The value of the key "*temperature*" in the data array present in the hourly object.
Pressure vs Time	The value of the key "*pressure*" in the data array present in the hourly object.
Humidity vs Time	The value of the key "*humidity*" in the data array present in the hourly object.
Ozone vs Time	The value of the key "*ozone*" in the data array present in the hourly object.
Visibility vs Time	The value of the key "*visibility*" in the data array present in the hourly object.
Wind Speed vs Time	The value of the key "*windSpeed*" in the data array present in the hourly object.

Table 3: Mapping the result from DarkSky API to get hourly data for charts

Note: Please take the first 24 values from the data array since it provides hourly data for the next 48 hrs.

The bar chart should be implemented with the help of Chart.js:

1. https://www.chartjs.org/docs/latest/charts/bar.html
2. https://medium.com/codingthesmartway-com-blog/angular-chart-js-with-ng2-charts-e21c8262777f

3.2.3 Weekly Tab

This tab provides a range bar chart of the minimum and maximum temperature for the next 7 days in the week.

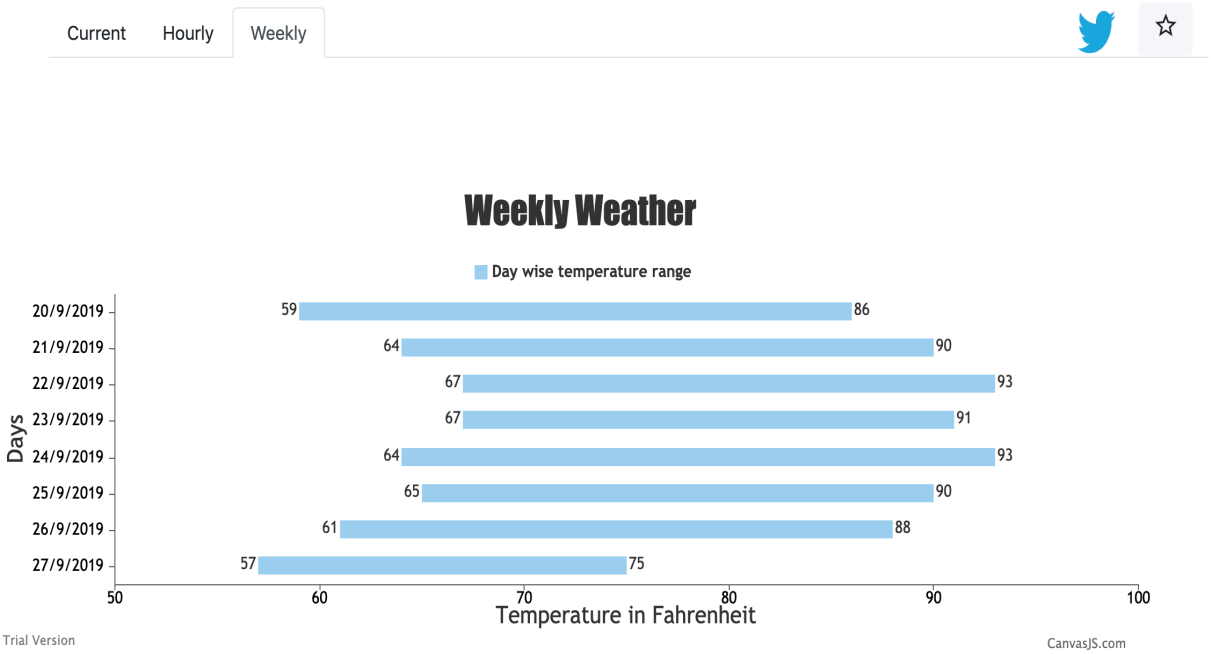


Figure 12: The Weekly tab which provides a chart for temperature range for next 7 days in the week.

This JSON response provides the necessary weekly data:

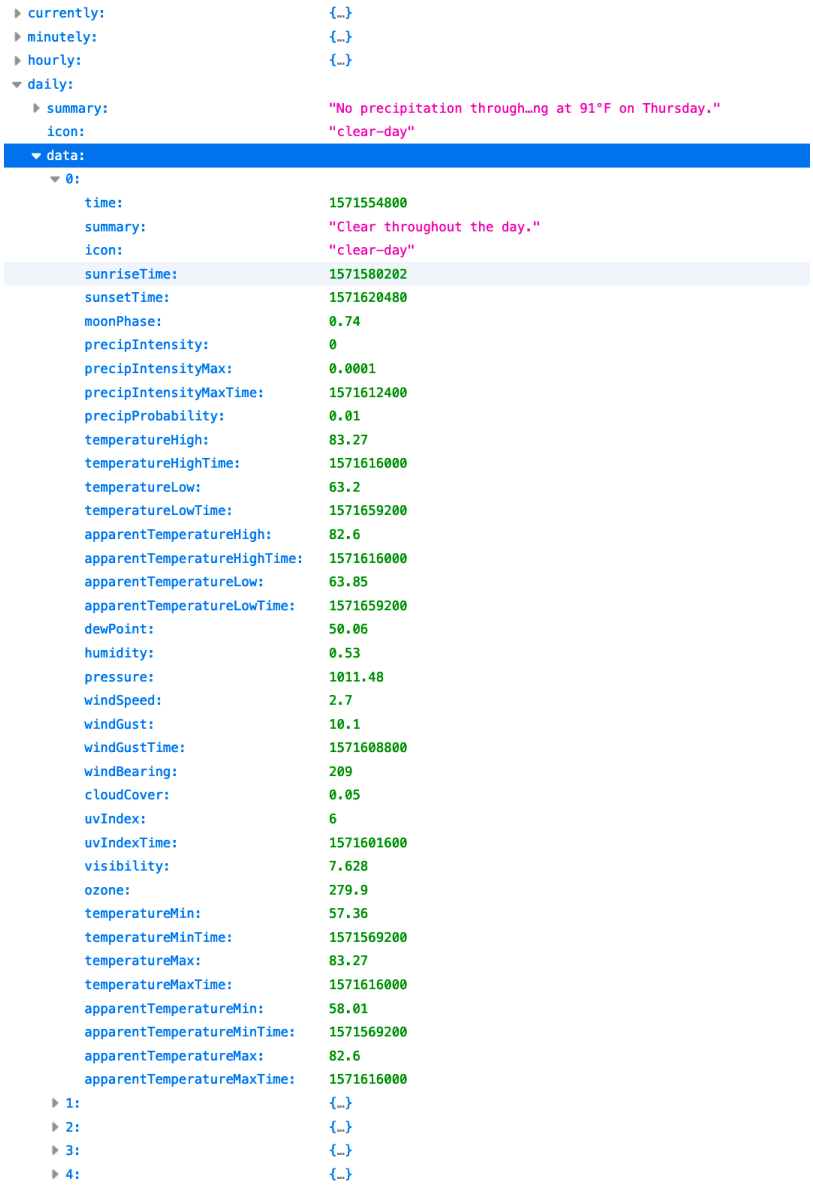


Figure 13: A sample JSON response from DarkSky API to get the weekly data.

Fields in the Chart	**Corresponding response object fields**
time	The value of the "*time*" attribute that is part of the "data" array in the daily object.
temperatureLow	The value of the "*temperatureLow*" attribute that is part of the "data" array in the daily object.
temperatureHigh	The value of the "*temperatureHigh*" attribute that is part of the "data" array in the daily object.

Table 4: Mapping the result from DarkSky API for the range bar chart.

On clicking on any one of the range bar chart rows, a corresponding modal window is displayed for that date. The modal window provides the detailed weather information for that date.

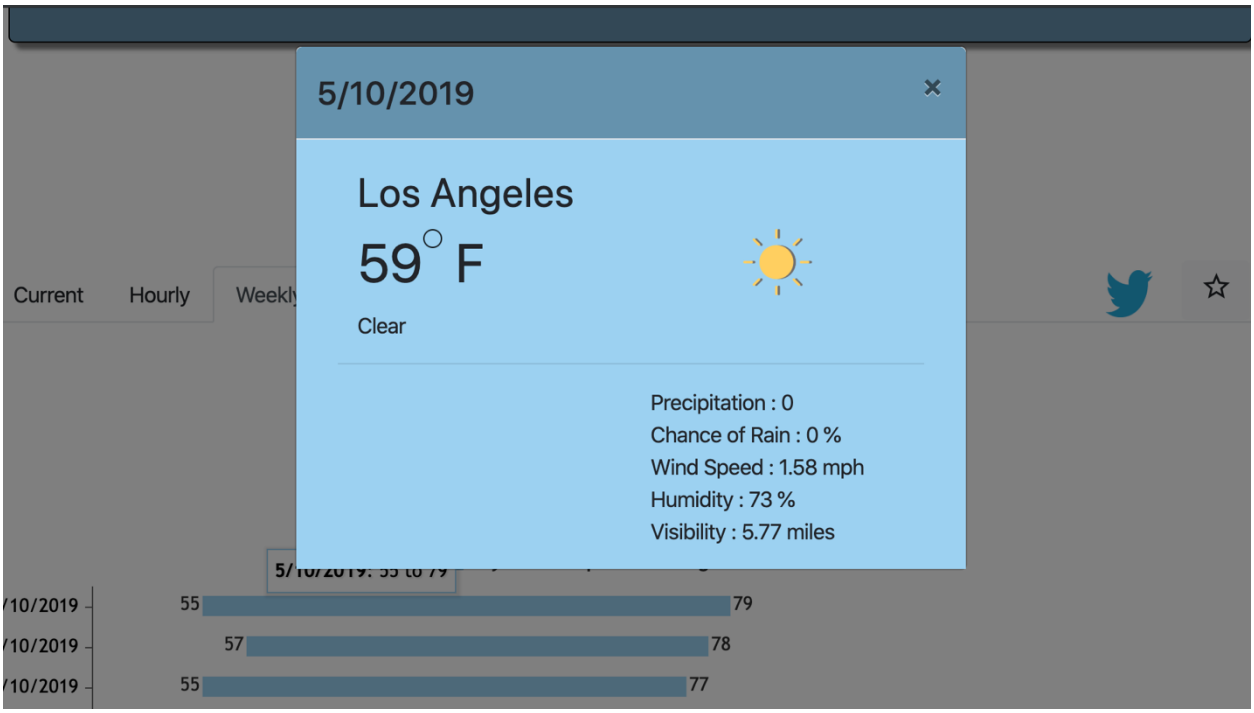


Figure 14: Modal providing weather information for that date.

The data for the modal is provided by another DarkSky API call:

https://api.darksky.net/forecast/[key]/[latitude],[longitude],[time]

Note: The time required here is in Unix format.

The JSON response for the modal data is shown below:

Figure 15: JSON response from the DarkSky API for a particular date.

Fields in the Modal	**Corresponding response object fields**
Date	The value of the "*time*" attribute that is part of the "data" array in the daily object. The time has to be converted into the form dd/mm/yyyy.
City	The value of the city can be taken from the input form if the user provides the address or from ip-api to get current location.
Temperature	The value of the key *"temperature"* in the *currently* object. You should display the value with a degree sign and F to indicate temperature in Fahrenheit. For the degree sign use this image - https://cdn3.iconfinder.com/data/icons/virtual-notebook/16/button_shape_oval-512.png
Summary	The value of *"summary"* in the *currently* object.

Icon	The value of *"icon"* in the *currently* object. It is possibly one of the following values: • clear-day | clear-night: The icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/sun-512.png • rain: The icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/rain-512.png • snow: The icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/snow-512.png • sleet: The icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/lightning-512.png • wind: The icon for this is https://cdn4.iconfinder.com/data/icons/the-weather-is-nice-today/64/weather_10-512.png • fog: The icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/cloudy-512.png • cloudy: This icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/cloud-512.png • partly-cloudy-day | partly-cloudy-night: The icon for this is https://cdn3.iconfinder.com/data/icons/weather-344/142/sunny-512.png
Precipitation	The value of "*precipIntensity"* in the *currently* object. The value is to be rounded to 2 decimal places.
Chance of Rain	The value of "*precipProbability"* in the *currently* object. The value is to be multiplied by 100 to give the percentage.
Wind Speed	The value of *"windSpeed"* in the *currently* object. The value should be provided with the unit *"mph"*. The value should be rounded to 2 decimal places
Humidity	The value of "*humidity"* in the *currently* object. The value is to be multiplied by 100 to give the percentage.
Visibility	The value of "*visibility*" in the currently object. The value should be provided with the unit miles.

Table 5: Mapping the result from DarkSky API for the modal.

Note: If any value is 0, display the 0 value. If the key is null/ not present, display N/A.

3.2.4 Favorite Button and Twitter Button

The **Favorite** button (star) provides the user the ability to add or remove the city to their favorites tab.

The **Twitter** button allows the user to compose a Tweet and post it to Twitter. Once the button is clicked, a new dialog should be opened and display the default Tweet content in this format:

"*The current temperature at (City) is (Temperature). The weather conditions are (Summary) #CSCI571WeatherSearch".*

Replace City, Temperature and Summary with the actual city searched with the temperature and summary or that day. For example, see Figure 16.

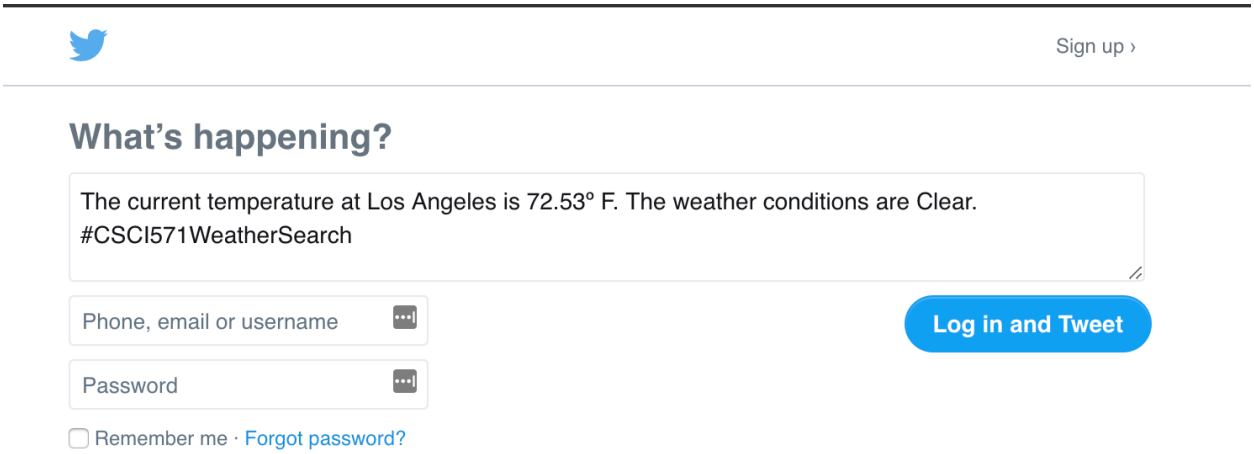


Figure 16 Tweet

Adding the Twitter button to your webpage is very easy. You can visit these two pages to learn how to use Twitter Web Intents:

https://dev.twitter.com/web/intents
https://dev.twitter.com/web/tweet-button/web-intent

The favorites button should be disabled until the content of the event tab and venue tab are available.

Figure 17 Favorite and Twitter Buttons

3.4 Favorite Tab

In the Favorites tab, the favorite cities are listed in a table format. The user can search for weather information for that city by clicking on the City name in the "City" column.

The information displayed in the Favorites tab is saved in and loaded from the **local storage** of the browser; the buttons in the "Favorites" column of the Favorites tab is only used to remove a city from the list and has a "trash" icon for it to be removed from the Favorite. (Refer to 5.5 for the icons).

The columns in this tab are:

1. Counter: Just a counter value indicating the number of cities in the favorites list.
2. Image: The state seal provided by the Google Custom search API.
3. City: The favorite city the user put in their favorites list. On clicking it should provide the weather details for that city.
4. State: The State abbreviations, which are present in the section 3.8 below and in the states.txt file. Same as given in HW 6 documentation.

Please note if a user closes and re-opens the browser, its Favorites will still be there.
If there are no cities in Favorites, please show 'No Records'. (see Figure 20)

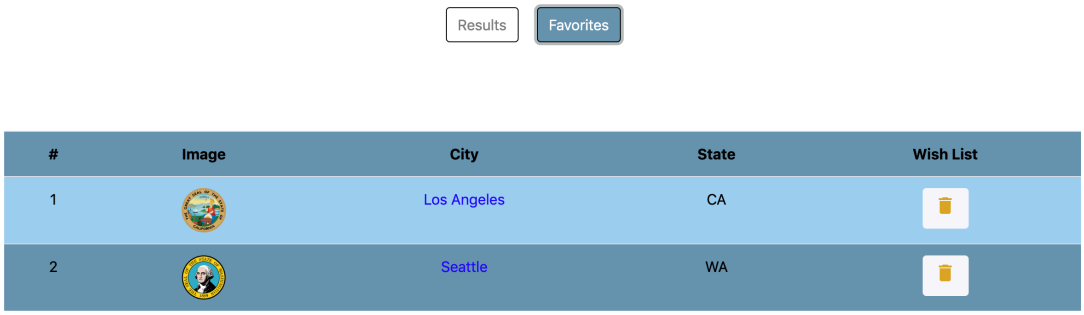
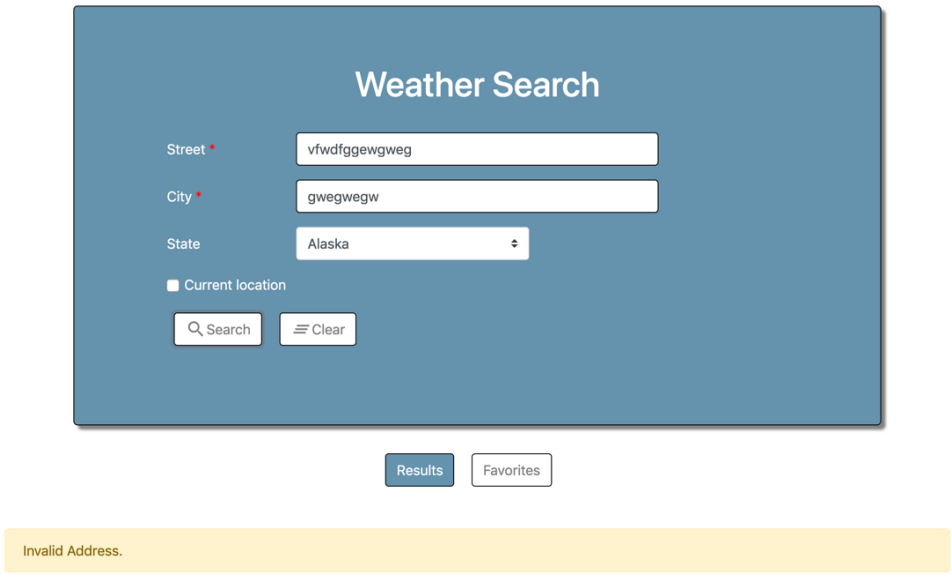


Figure 18 Favorites

3.5 Error Messages

If for any reason an error occurs due to location error, an appropriate error message should be displayed.

Figure 19 Error Message



3.6 No Records

Whenever the search receives no records, an appropriate message should be displayed.

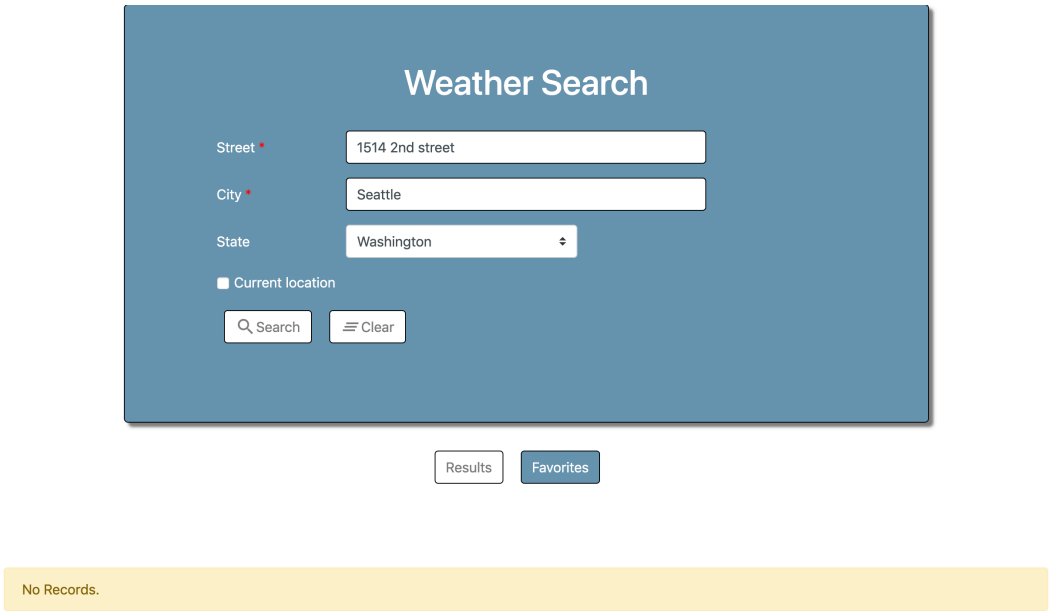


Figure 20 No Records on Favorites

3.7 Progress Bars

Whenever data is being fetched, a dynamic progress bar must be displayed as shown in Figure 23.

You can use the progress bar component of **Bootstrap** to implement this feature. You can find hints about the bootstrap components in the Hints section.

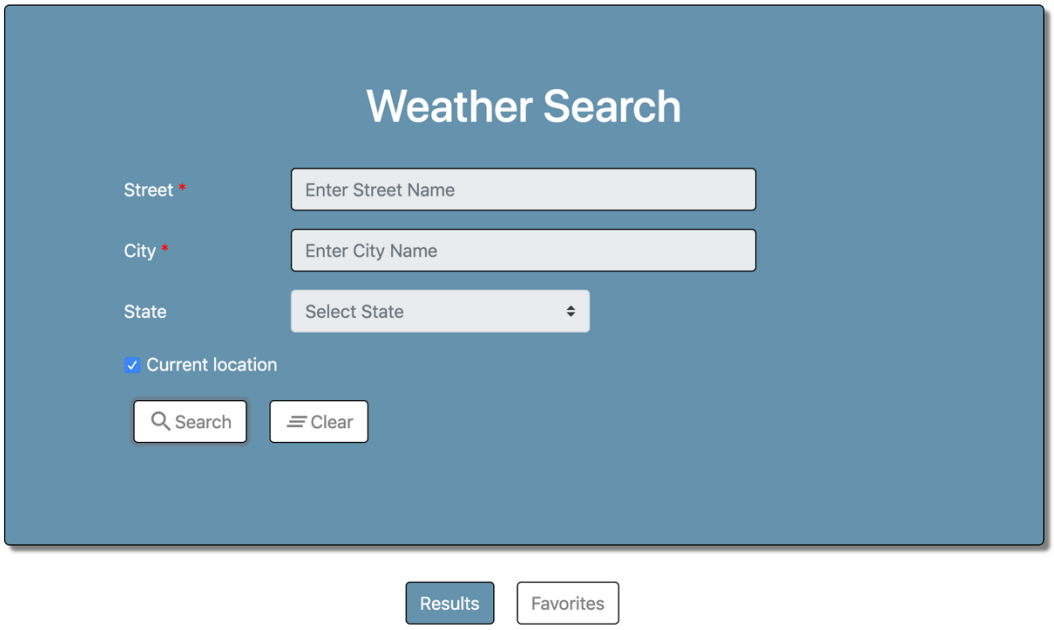


Figure 21 Progress Bar

3.8 List of US States and Their Two-Letter Abbreviations

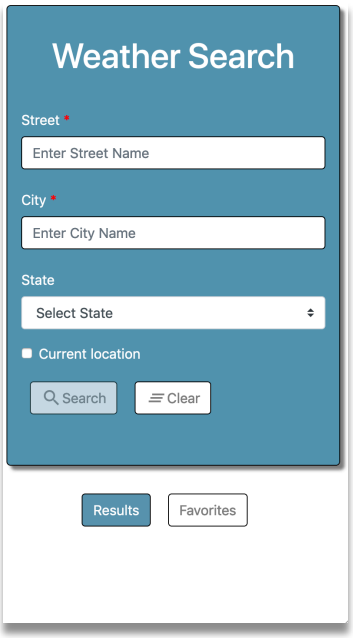
Two-Letter Abbreviation	State
AL	Alabama
AK	Alaska
AZ	Arizona
AR	Arkansas
CA	California
CO	Colorado

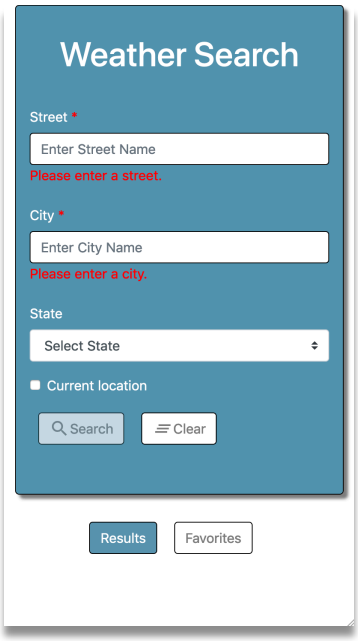
CT	Connecticut
DE	Delaware
DC	District of Columbia
FL	Florida
GA	Georgia
HI	Hawaii
ID	Idaho
IL	Illinois
IN	Indiana
IA	Iowa
KS	Kansas
KY	Kentucky
LA	Louisiana
ME	Maine
MD	Maryland
MA	Massachusetts
MI	Michigan
MN	Minnesota
MS	Mississippi
MO	Missouri
MT	Montana
NE	Nebraska
NV	Nevada
NH	New Hampshire

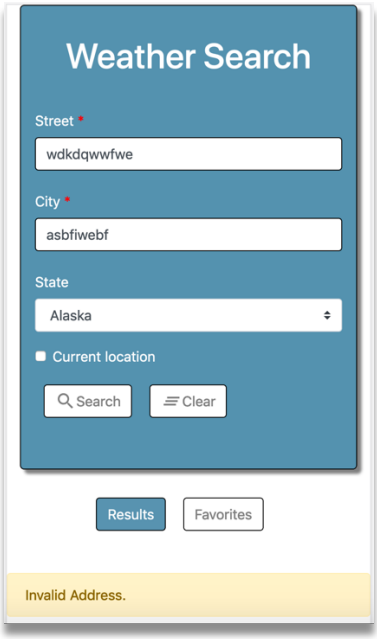
NJ	New Jersey
NM	New Mexico
NY	New York
NC	North Carolina
ND	North Dakota
OH	Ohio
OK	Oklahoma
OR	Oregon
PA	Pennsylvania
RI	Rhode Island
SC	South Carolina
SD	South Dakota
TN	Tennessee
TX	Texas
UT	Utah
VT	Vermont
VA	Virginia
WA	Washington
WV	West Virginia
WI	Wisconsin
WY	Wyoming

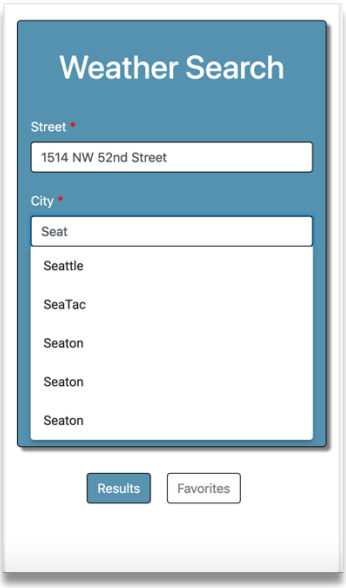
3.9 Responsive Design

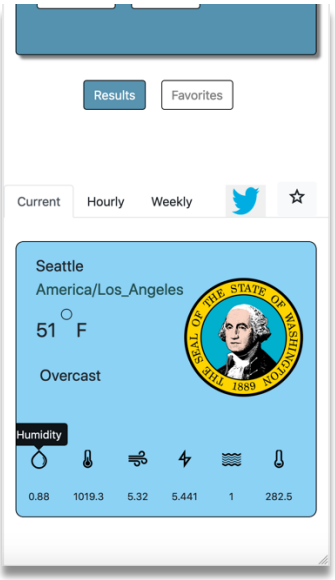
The following are snapshots of the webpage opened with Safari on iPhone 7 Plus. See the video for more details.

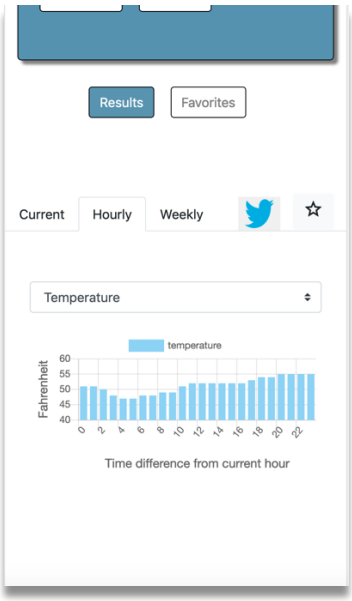

Weather Search
Street *
Enter Street Name
City *
Enter City Name
State
Select State
Current location
Search
Clear
Results
Favorites


Weather Search
Street *
Enter Street Name
Please enter a street.
City *
Enter City Name
Please enter a city.
State
Select State
Current location
Search
Clear
Results
Favorites


Weather Search
Street *
wdkdqwwfwe
City *
asbfiwebf
State
Alaska
Current location
Search
Clear
Results
Favorites
Invalid Address.


Weather Search
Street *
1514 NW 52nd Street
City *
Seat
Seattle
SeaTac
Seaton
Seaton
Seaton
Results
Favorites


Results
Favorites
Current
Hourly
Weekly
Seattle
America/Los_Angeles
51 F
Overcast
Humidity
0.88
1019.3
5.32
5.441
1
282.5


Results
Favorites
Current
Hourly
Weekly
Temperature
temperature
Fahrenheit
Time difference from current hour

You must watch the video carefully to see how the page looks like on mobile devices. All functions must work on mobile devices.

Mobile browsers are different from desktop browsers. Even if your webpage works perfectly on a desktop, it may not work as perfectly as you think on mobile devices. It's important that you also test your webpage on a real mobile device. Testing it in the mobile view of a desktop browser will not guarantee that it works on mobile devices. You can use Firefox "Responsive Design Mode" Tool for initial testing.

4. API Documentation

4.1 DarkSky API
To use the DarkSky API, you need first to register for DarkSky Account. This is the same App id used for the HW6.

4.2 Google Customized Search API
This link will provide the details to get the API key:

https://developers.google.com/custom-search/json-api/v1/overview

Note: You can use any additional Angular libraries and Node.js modules you like.

5. Implementation Hints

5.1 Images
The images needed for this homework are available here:

http://csci571.com/hw/hw8/Images/Twitter.png

5.2 Get started with the Bootstrap Library
To get started with the Bootstrap toolkit, please refer to the link:

https://getbootstrap.com/docs/4.0/getting-started/introduction/.

You need to import the necessary CSS file and JS file provided by Bootstrap.

5.3 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.

At a minimum, you will need to use Bootstrap Forms, Tabs, Progress Bars and Alerts to implement the required functionality.

Bootstrap Forms https://getbootstrap.com/docs/4.0/components/forms/
Bootstrap Tabs https://getbootstrap.com/docs/4.0/components/navs/#tabs
Bootstrap Progress Bars https://getbootstrap.com/docs/4.0/components/progress/
Bootstrap Alerts https://getbootstrap.com/docs/4.0/components/alerts/
Bootstrap Tooltip https://getbootstrap.com/docs/4.1/components/tooltips/
Bootstrap Cards https://getbootstrap.com/docs/4.0/components/card/

5.4 Angular Material
Angular Material (Angular 2+) :https://material.angular.io/
Autocomplete: https://material.angular.io/components/autocomplete/overview

Tooltip: https://material.angular.io/components/tooltip/overview

5.5 Material Icon

Icons for the search, clear_all, star, star_border and delete can be viewed here:
https://google.github.io/material-design-icons/
https://material.io/tools/icons/

5.6 Google App Engine/Amazon Web Services/ Microsoft Azure

You should use the domain name of the GAE/AWS/Azure service you created in Homework #7 to make the request. For example, if your GAE/AWS/Azure server domain is called example.appspot.com/example.elasticbeanstalk.com/ example.azurewebsites.net, the JavaScript program will perform a GET request with keyword="xxx", and an example query of the following type will be generated:

GAE - http://example.appspot.com/weatherSearch?keyword=xxx
AWS - http://example.elasticbeanstalk.com/weatherSearch?keyword=xxx
Azure – http://example.azurewebsites.net/weatherSearch?keyword=xxx

Your URLs don't need to be the same as the ones above. You can use whatever paths and parameters you want. Please note that in addition to the link to your Homework #8, you should also **provide a link like this URL in the table of your Node.JS backend link**. When your grader clicks on this additional link, a valid link should return a JSON object with appropriate data.

5.7 Deploy Node.js application on GAE/AWS/Azure

Since Homework #8 is implemented with Node.js on AWS/GAE/Azure, you should **select Nginx as your proxy server (if available)**, which should be the default option.

5.8 AJAX call

You should send the request to the Node.js script(s) by calling an Ajax function (Angular). 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 Node.js script code is running in the "cloud" on Google GAE/AWS/Azure (see 6.6 above). Please refer to the grading guidelines for details.

Note: jQuery .ajax cannot be used in this project.

5.9 HTML5 Local Storage

Local storage is more secure, and large amounts of data can be stored locally, without affecting website performance. Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server. There are two methods, getItem() and setItem(), that you can use. The local storage can only store strings. Therefore, you need to convert the data to string format before storing it in the local storage. For more information, see:

https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
http://www.w3schools.com/html/html5_webstorage.asp

6. Files to Submit

In your course homework page, you should update the Homework #8 link to refer to your new initial web page for this exercise. Additionally, you need to provide **an additional link** to the URL of the GAE/AWS/Azure service where the AJAX call is made with sample parameter values (i.e. a valid query, with keyword, location, etc. See 6.6).

Also, submit all files (HTML, JS, CSS, TS) you have written yourself, electronically to the GitHub Classroom repository so that can be compared to all other students' code. **Don't include library, node-js modules, any config files, any angular-cli build files, or any images that we provided or that are included in any library or any code generated by the tools.**

Please note, you need to submit your files (HTML, JS, CSS, TS), both backend and frontend, directly to the root folder of your homework 8 GitHub repository. That is, **do not submit any subfolders on GitHub**. If you submit some subfolders, you will receive a 2-points penalty.

IMPORTANT:

All videos are part of the homework description. All discussions and explanations in Piazza related to this homework are part of the homework description and will be accounted into grading. So please review all Piazza threads before finishing the assignment.