HOMEWORK ASSIGNMENT 3

CSCI 571 – Fall 2024

Abstract

Ajax, JSON, Angular, Bootstrap, MongoDB, RWD, Node.js, and the Tomorrow.io API

Marco Papa

papa@usc.edu

Assignment 3: Ajax, JSON, Responsive Design and Node.js

Weather Search

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

1. Objectives

- Achieve familiarity with the AJAX and JSON technologies.
- Create the front-end by integrating and combining HTML5, CSS, Bootstrap, and Angular**.**
- Create the back end using JavaScript on Node.js.
- Both front-end and back-end must be implemented in one of the approved cloud environments.
- Build responsive web design with Bootstrap to enhance UX across multiple screens.
- Deploy your Web app on Google Cloud Platform/Amazon Web Services/Microsoft Azure.
- Leverage APIs for added functionalities. Key APIs you will work with include tomorrow.io API, Google Maps API, Google Geocoding & Places API, IPinfo API, HighCharts and X (aka Twitter) API.
- Learn how to manage and access a NoSQL DBMS like MongoDB Atlas, in the cloud.

2. Background

2.1 AJAX and JSON

AJAX (Asynchronous JavaScript + XML) incorporates several technologies which include Standards-based presentation using XHTML and CSS, displaying results and interactions using the Document Object Model (DOM), data interchange and manipulation using XML and JSON, Asynchronous data retrieval using XMLHttpRequest. In summary, JavaScript binds everything together. Peruse the class slides covering Ajax on D2L Brightspace for detailed information.

JSON, short for JavaScript Object Notation, is a lightweight data interchange format. Its primary application for this task is in AJAX web application programming, where it serves as an alternative to the XML format for data exchange between client and server. Peruse the class slides covering JSON on D2L Brightspace for detailed information.

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. **We recommend Bootstrap 5.3 and ng-bootstrap (v 17.x.x) in this assignment.** In general, you can use **Bootstrap 5.2.3** through **5.3.2**, **Angular 15** through **17**, **ng-bootstrap 14** through **16,** and **Node.js 18 or 20** in this assignment. To learn more details about Bootstrap please refer to the lecture material on Responsive Web Design (RWD). Peruse the class slides covering Responsive Design on D2L Brightspace.

2.3 Google Cloud Platform (GCP)

Google Cloud Platform (GCP) applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With GCP, 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, Memcached, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable in GCP. Peruse GCP support information for Node.js in the ungraded assignment **Cloud Setup (NodeJS)** in the Assignment folder on D2L Brightspace.

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

https://cloud.google.com/appengine/docs/standard/nodejs/

2.4 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. Peruse AWS support information for Node.js in the ungraded assignment **Cloud Setup (NodeJS)** in the Assignment folder on D2L Brightspace.

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

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

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. Peruse Azure support information for Node.js in the ungraded assignment **Cloud Setup (NodeJS)** in the Assignment folder on D2L Brightspace.

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 integrates well with other libraries. Every feature can be modified to suit your

unique development 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 18 is recommended,** but Angular 12, through 17 can be used. However, please note Angular can be difficult to learn if you are not familiar with Typescript and component-based programming.

To learn more about Angular, visit these pages:

https://angular.io/ (v 17 and earlier)
https://angular.dev/ (v 18)

Note: AngularJS (a.k.a Angular 1.0) cannot be used in this project, and will result in a score of zero (0) in the assignment.

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. Peruse Node.js support information in the class slides covering JavaScript Frameworks on D2L Brightspace.

To learn more about Node.js, visit:

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

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. Peruse Express.js support information the class slides covering JS Frameworks on D2L Brightspace. There are other frameworks like Express that are covered in the lectures. Any of these frameworks can be used in this assignment.

To learn more about Express.js, visit:

http://expressjs.com/

Important Explicit Notes:

1. In this document when you see GCP/AWS/Azure it implies that you can either use Google App Engine, Amazon Web Services or Microsoft Azure Services. However, the CSCI 571 staff will only provide full support for GCP on Piazza and only partial support for AWS and Azure, as none of the TAs use either platform.
2. All APIs calls to tomorrow.io must be done through the Node.js server, functioning as a "proxy." All other API calls should be done through your front-end JavaScript.

3. It is recommended to perform all HTTP calls to your Node backend with either fetch() or the Angular HttpClient (as opposed to using jQuery.ajax() / axios() / XMLHttpRequest()), but any asynchronous XHR functionality will be acceptable.

3. High Level Description

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

There are two ways the user can provide the location information for which they are trying to look up the detailed weather information. The first way is by using the fields "Street address" and "City", and/or "State". The second way is by detecting the user's current location.

Dynamic Validation must be performed on all the form text fields – i.e., you must not wait until the submit button is hit to verify that the text input is invalid. More information on this is provided below.

Upon invoking the search button, and after receiving a successful response, the results pane should be displayed, which contains **three tabs** namely "Day View", "Daily Temp. Chart", and "Meteogram" (see **Figure 6**, later in this document).

You must display appropriate error messages in cases where you do not get a valid response from the API. In the time between hitting the Search button and displaying the results / error messages you should display a "progress bar" to indicate that your application is doing something in the background

The webpage should have a **Favorites tab**, which would support the functionality of adding cities and removing cities from that tab. Additionally, the Day View webpage should have a Tweet button to share the weather details on Twitter. Implementation requirements and details will be explained in the next sections.

When a user initially opens your webpage (landing page), your page should look like **Figure 1**.

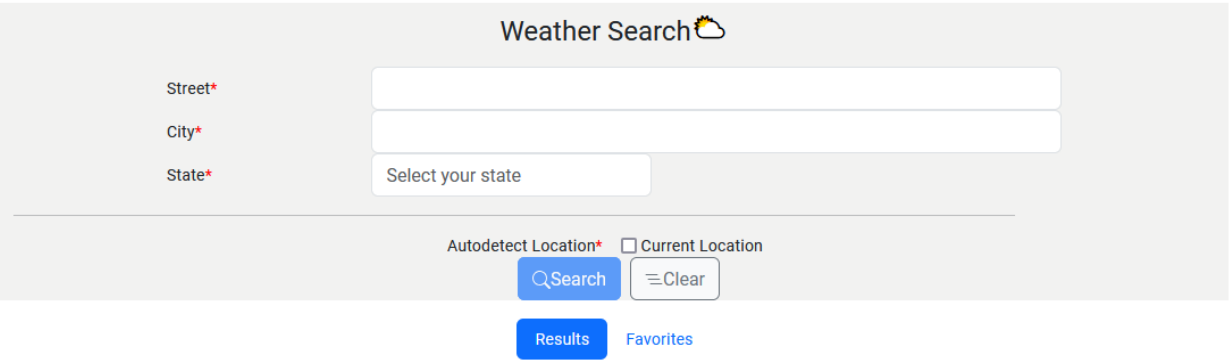


Figure 1. Initial Search Form

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 like the ones used in Assignment #2, but their "look" is different, as they are not implemented with default HTML controls.

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

1. **Street**: Initially, this field is left blank.
2. **City**: This input field should support "autocomplete" which is explained in section 3.1.2. Please note that the user may not necessarily select the autocomplete suggestions. Initially, this field is left blank.
3. **State**: There are multiple options for the user to select from, containing all the names of States of the US as shown in **Figure 2**.

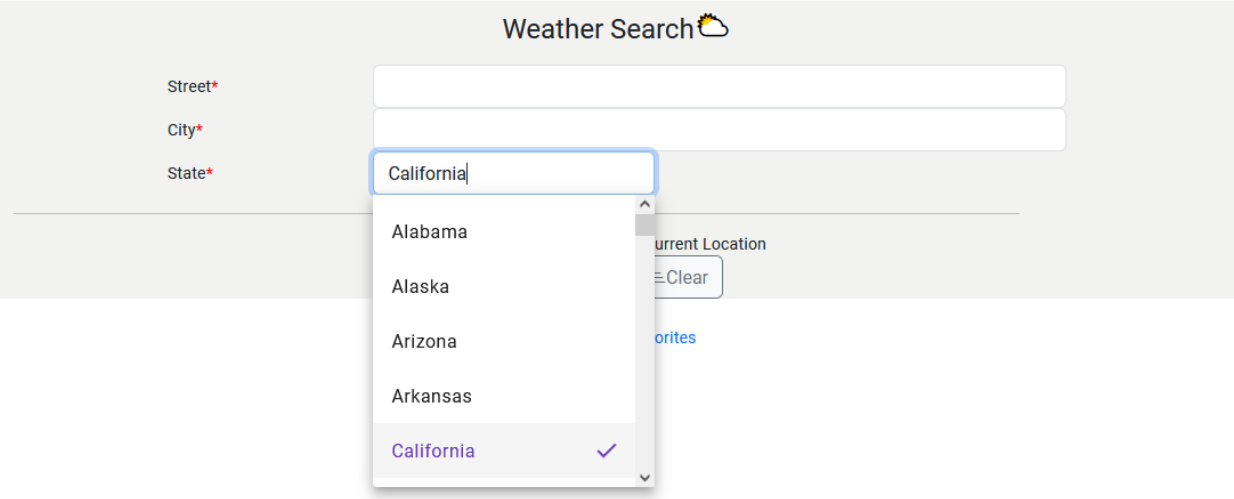


Figure 2. State Options

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 has not been obtained yet. Please refer to section 3.1.3 for validation details. 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 the **City** field is implemented by using the *Google Place Autocomplete API service*. To learn more about this service, you can go to this page:

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

The format of the HTTP request URL, is as follows:

https://maps.googleapis.com/maps/api/place/autocomplete/json?input=[location]&key=[key]

An example of an HTTP request to the *Places Autocomplete API* that searches for matches with the city text field "Los", and returns results in JSON, is shown below:

https://maps.googleapis.com/maps/api/place/autocomplete/json?input=Los&key=[KEY]

To get the **API key** for the Goole Places API service in the above URL. please follow the steps provided in the following link:

https://developers.google.com/maps/documentation/places/web-service/get-api-key

Note:

1. **If the API doesn't work, hide the autocomplete feature. This situation should be handled and not throw any error.**
2. You can try setting the **types** parameter in the request to "cities" to restrict suggestions to only cities.

The HTTP response is a JSON object like the one shown in **Figure 3**.

Figure 3. Autocomplete JSON Response

The fields we suggest extracting from the API response are highlighted above, object "terms", and "value" for each returned match. The idea is to pick up the city and state from the response, and build a list of (city, state) pairs. While only the city value is displayed in the dropdown, when a user selects a city from the autocomplete dropdown, we also populate the state field appropriately.

For reference: The values highlighted are the fields in the terms array for every object in the predictions array.

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

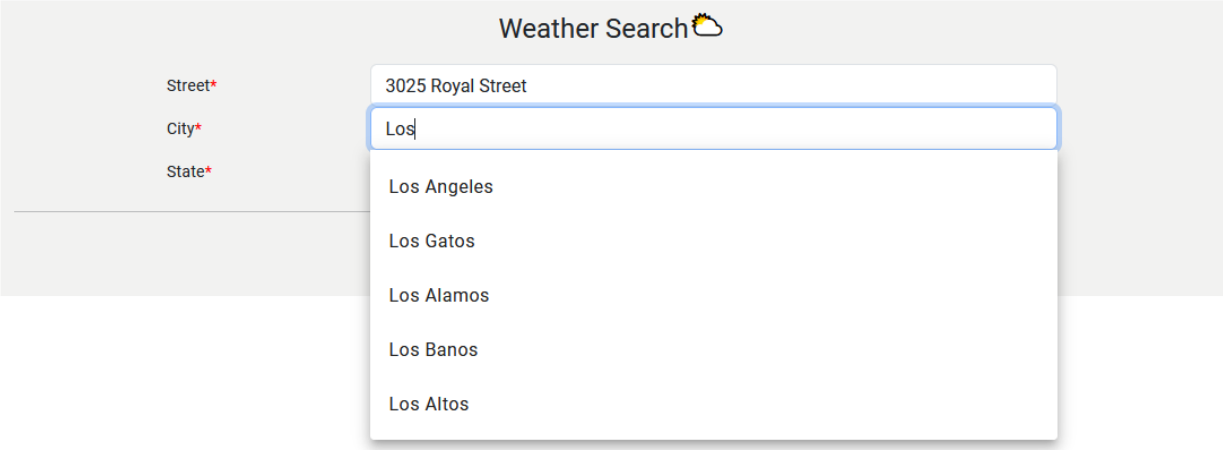


Figure 4. Autocomplete example

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 entry 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 input fields should be disabled, with their values retained.

Please watch the reference video carefully to understand the validation expected behavior.

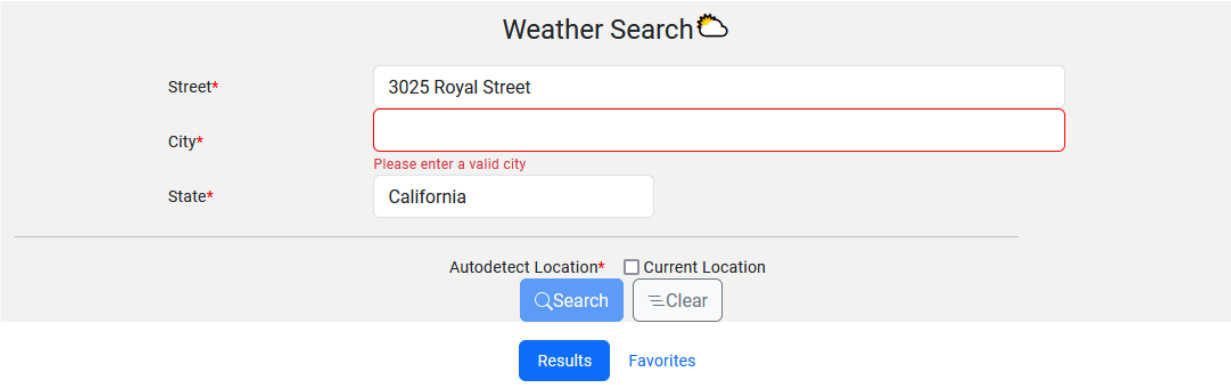


Figure 5. An Invalid Form

3.1.4 Obtaining User Location

As in Homework Assignment #2, you should obtain the current user location using one of the geolocation APIs. The usage of this API has been explained in the Assignment #2 documents. For more information on the IPinfo API, visit: the 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 the tomorrow.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 user's device.

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

A sample *Google geocode API* call looks like this:

https://maps.googleapis.com/maps/api/geocode/json?address=[Street + City + State]&key=[Key]

Note:

1. We use the JSON response to get the latitude and longitude from the *Google geocode API*.
2. Please refer to the Assignment #2 documentation to get the Google geocode API key. You should be able to re-use the key you obtained for Assignment #2.

Once the latitude and longitude are fetched from the Current Location or the *Google geocode API*, they must be passed to the Tomorrow.io API to fetch the weather details.

The *Tomorrow.io API* is documented here: https://docs.tomorrow.io/reference/welcome

The usage of these two APIs has been explained in the Assignment #2 documents. You may refer to that specification for example requests.

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 the upper right.

The display of the results is divided into 3 tabs, i.e., **Day View** tab**, Daily Temp. Chart** tab **and Meteogram** tab. The detailed description of all the tabs is given in the following sections.

3.2.0 Results Title

An appropriate title must be set based on the location for which the user is requesting weather information. For example, if the user queries for the forecast through form text input, you can use the data from the City and State fields to display the title

"Forecast at [City], [State]"

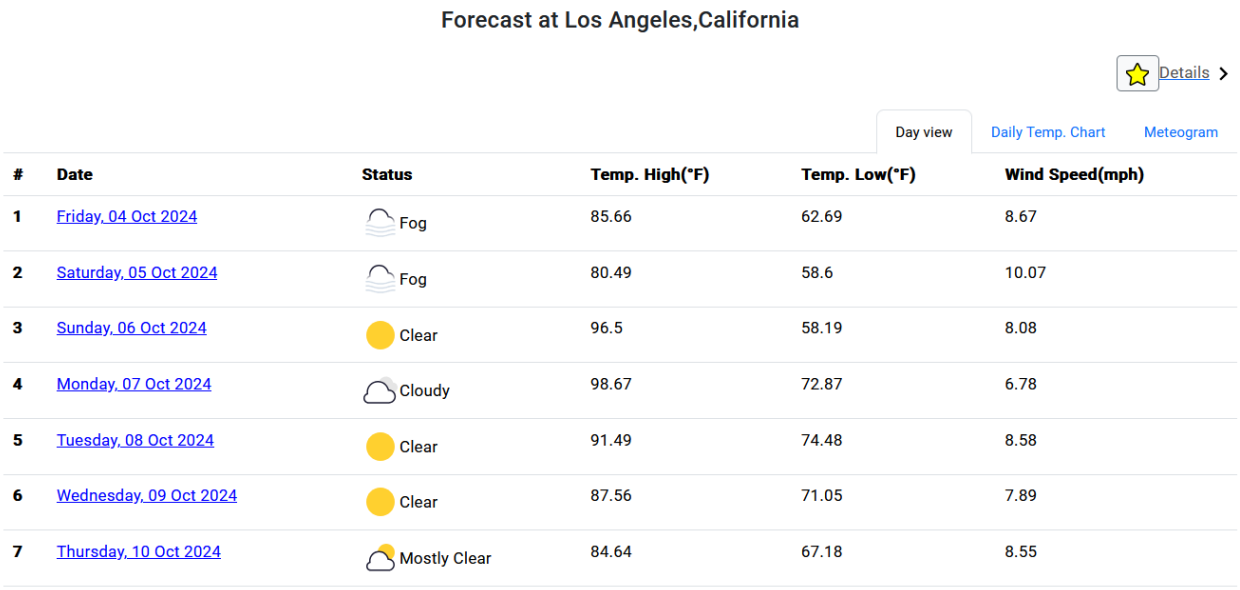
However, when requesting for weather information based on the IP address data through the *IPinfo API*, you may use the "city" and "region" keys from the ip-info response to display the title "Forecast at …".


```
ip: "8.8.4.4"
hostname: "dns.google"
anycast: true
city: "Mountain View"
region: "California"
country: "US"
loc: "38.0088,-122.1175"
org: "AS15169 Google LLC"
postal: "94043"
timezone: "America/Los_Angeles"
```


The **Favorite** button (star) (see **Figure 6**) provides the user the ability to add or remove the city from the favorites tab. The information saved in the Favorites tab must be retained even if we exit the browser and reload the page and visit it back later. You must use **MongoDB Atlas**, in the cloud, to implement this functionality.

3.2.1 Day View Tab

This tab is in a card layout which provides the details of the weather for the next 6 days at the corresponding location provided by the user.



#	Date	Status	Temp. High(°F)	Temp. Low(°F)	Wind Speed(mph)
1	Friday, 04 Oct 2024	Fog	85.66	62.69	8.67
2	Saturday, 05 Oct 2024	Fog	80.49	58.6	10.07
3	Sunday, 06 Oct 2024	Clear	96.5	58.19	8.08
4	Monday, 07 Oct 2024	Cloudy	98.67	72.87	6.78
5	Tuesday, 08 Oct 2024	Clear	91.49	74.48	8.58
6	Wednesday, 09 Oct 2024	Clear	87.56	71.05	7.89
7	Thursday, 10 Oct 2024	Mostly Clear	84.64	67.18	8.55

Figure 6. An Example of a Valid Search result

A sample API call to get the data needed to populate the table and details page, is shown below.

https://api.tomorrow.io/v4/timelines?location=[LAT,LONG]&fields=[FIELD_NAME]&timesteps=1d&units=[UNIT]&timezone=[TIME_ZONE]&apikey=[API_KEY]

This is the same call that you performed for Assignment #2; hence you may refer to that document for details regarding the fields needed.

3.2.2 Daily Temp. Chart Tab

This tab displays a range graph for the min/max daily temperatures for the selected location for the next 15 days. As sample output is shown in **Figure 7**.

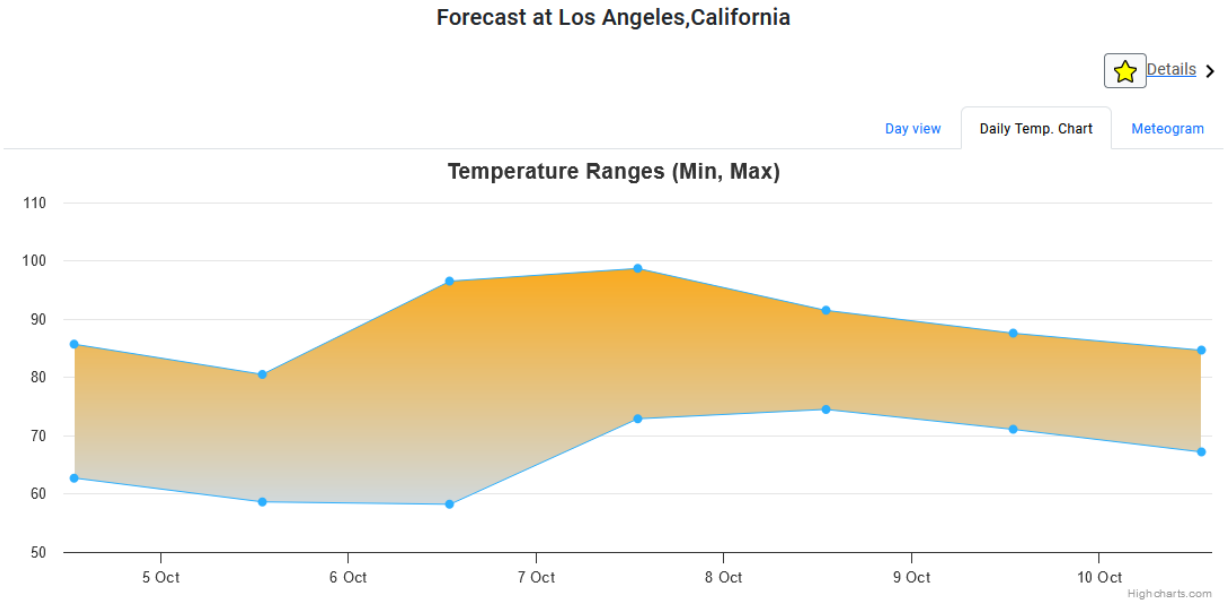


Figure 7. Daily Temperature Range Chart

3.2.3 Meteogram Tab

The meteogram chart should consist of an hourly based plot over a period from current time (not the selected) to next 5 days. An example output is shown in Figure 8. An example of an HTTP request to the Tomorrow.io API that searches for the hourly weather information which is required for this chart, is shown below:

https://api.tomorrow.io/v4/timelines?location=[LAT,LONG]&fields=[FIELD_NAME]&timesteps=1h&units=[UNIT]&timezone=[TIME_ZONE]&apikey=[API_KEY]

Reference for the development of this chart is available at:

https://www.highcharts.com/demo/highcharts/combo-meteogram

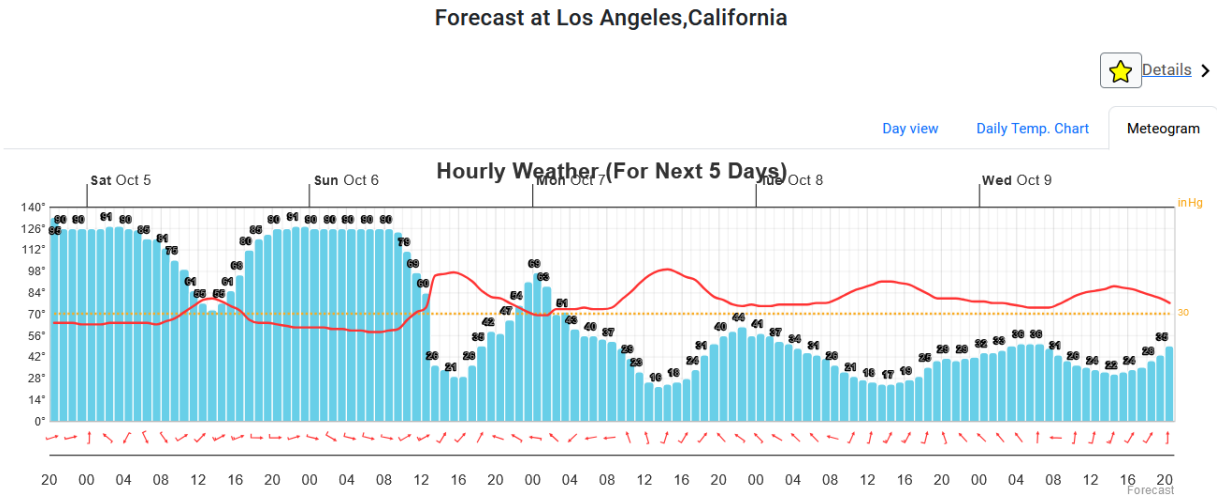


Figure 8: The Meteogram tab which provides a chart for temperature range for next 5 days.

3.3 Details Pane

On clicking on a row from the results table, a new pane with more detailed weather information for the selected day should be displayed. Note: there must be a “slide” transition when moving between the results and details panes.

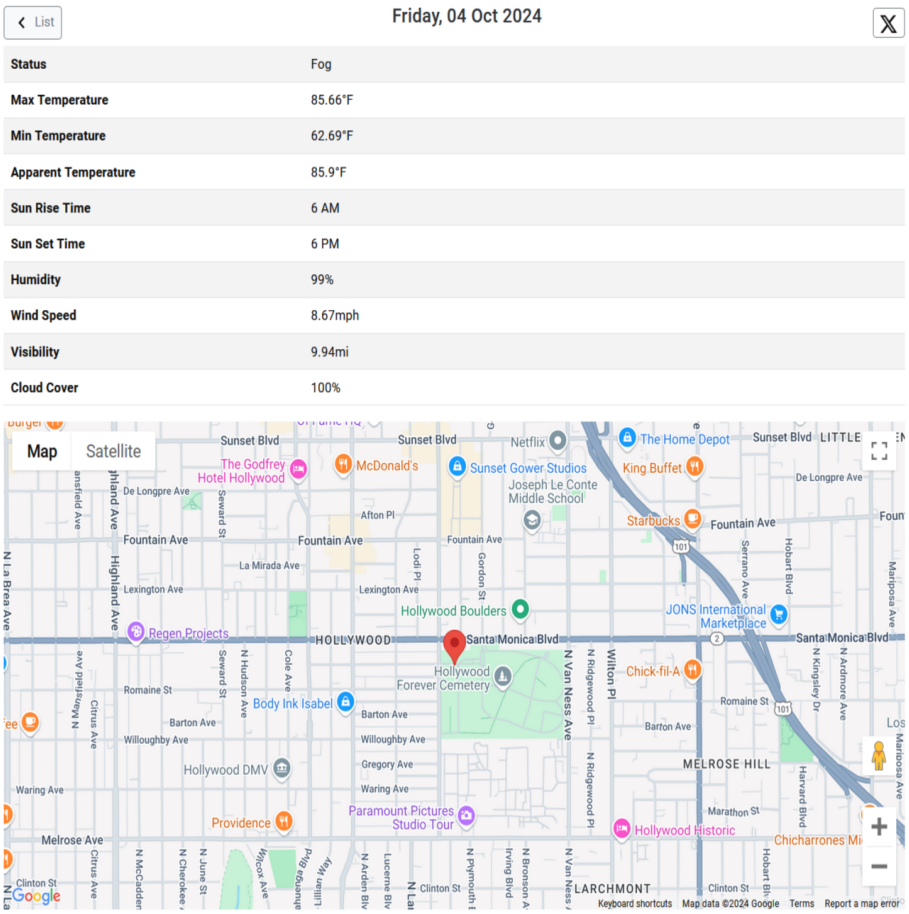


Figure 9. Sample Details Pane

The details pane is a simple table that consists of additional information that is available through the Tomorrow.io API. In addition to the fields from the API, you are also required to display an interactive Google Map section that is centered on the Lat/Long information that you performed in your Tomorrow IO API query on (i.e., lat/long from IPinfo or from the Google Geocode API).

You should use the Google Maps JavaScript Library, documented at:

https://developers.google.com/maps/documentation/javascript/examples

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

"The temperature in (City, State) on (Day of week, Date) is (Temperature). The weather conditions are (Summary) #CSCI571WeatherSearch".

Replace City, State, Date, Temperature and Summary with the actual city searched with the temperature and summary or that day. For example, see Figure 10.

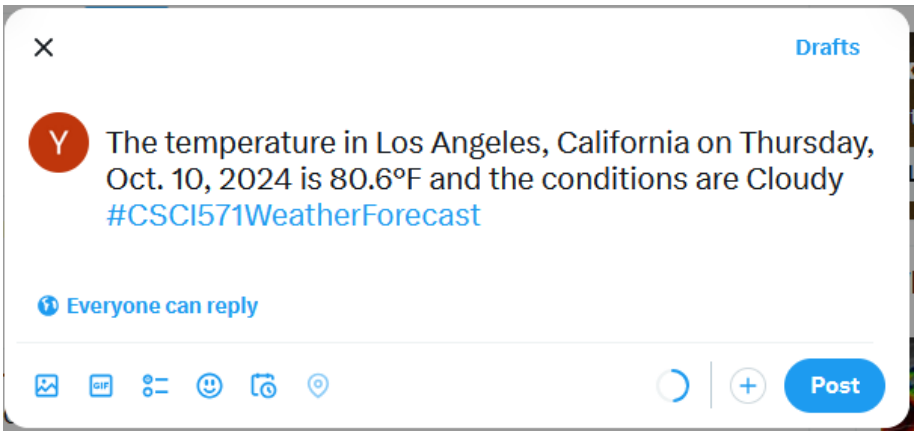


Figure 10. Sample Tweet on X

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

https://developer.x.com/en/docs/x-for-websites/web-intents/overview
https://developer.x.com/en/docs/x-for-websites/tweet-button/overview

Figure 11. Favorites and Twitter Buttons

3.4 Favorites 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 cloud storage provided by MongoDB Atlas. The only information stored in MongoDB is City and State.

The buttons in the "Favorites" column of the Favorites tab are only used to remove a city from the list and have a "trash" icon for it to be removed from the Favorites. (Refer to section 5.5 for the needed icons).

The columns in this tab are:

1. Counter ('#'): Just a counter value indicating the number of cities in the favorites list.
2. City: The favorite city the user put in their favorites list. On clicking it should provide the weather details for that city.
3. State: The State information you get from the form input or ipinfo API.
4. Trashcan Icon: Used to dynamically remove this city from the favorites list.

Figure 12. Favorites

Please note if a user closes and re-opens the browser, its Favorites should still be there.
If there are no cities in the Favorites list, please show 'Sorry! No records found. (see Figure 13).

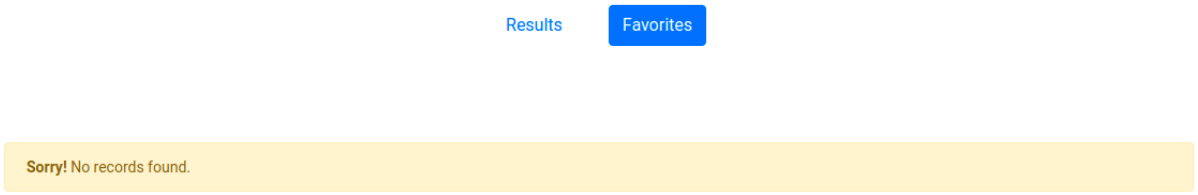


Figure 13. No Favorites

3.5 Error Messages

If for any reason an error occurs due to location error, or reaching the maximum API limit, an appropriate error message should be displayed, as shown in **Figure 14.**

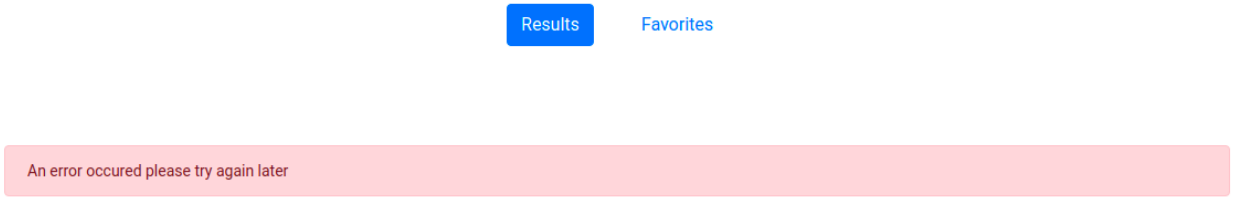


Figure 14. Error Message

3.6 Progress Bars

Whenever data is being fetched, a dynamic progress bar must be displayed as shown in **Figure 15**.

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.

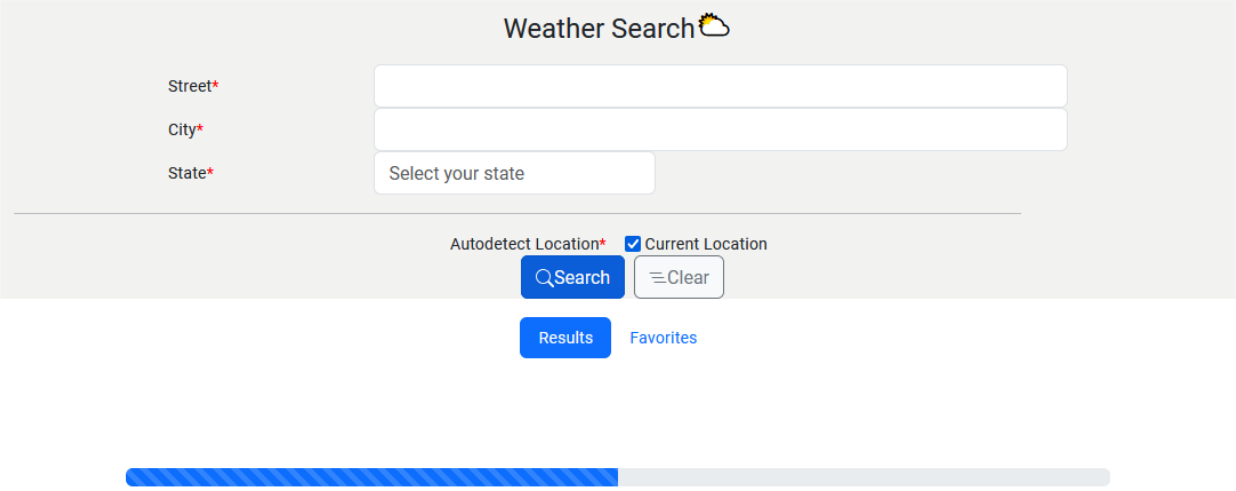


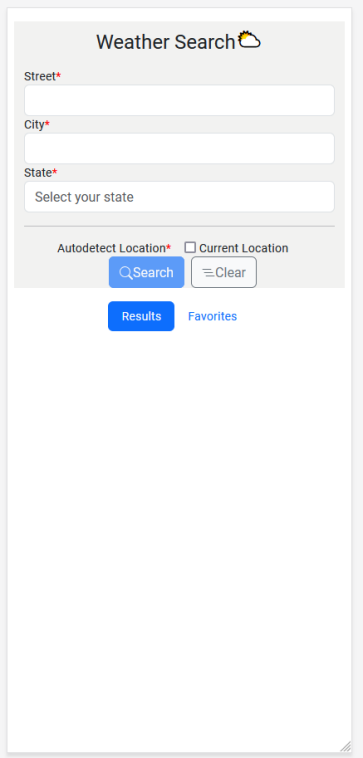
Figure 15. Progress Bar

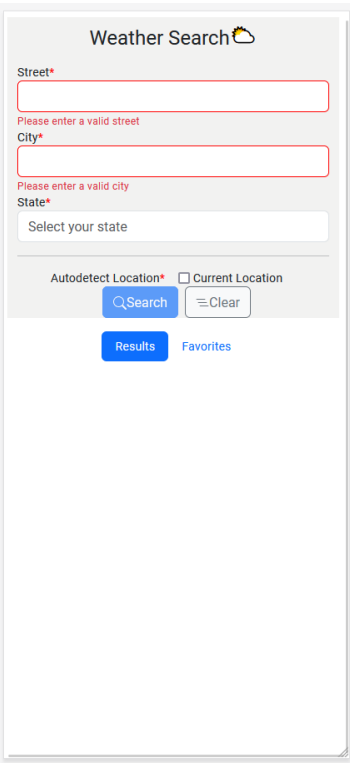
3.7 List of US States

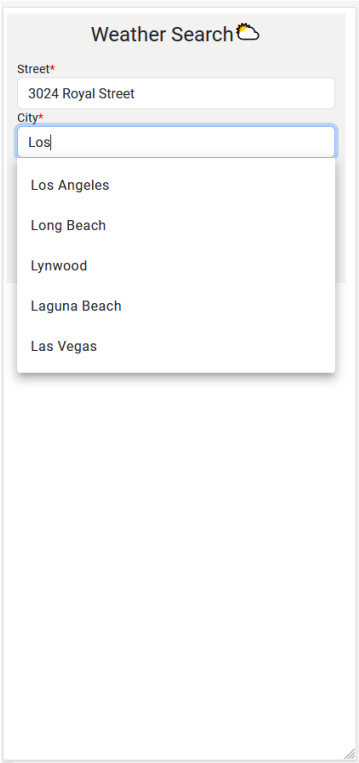
Alabama, Alaska, Arizona, Arkansas, California, Colorado, Connecticut, Delaware, Florida, Georgia, Hawaii, Idaho, Illinois, Indiana, Iowa, Kansas, Kentucky, Louisiana, Maine, Maryland, Massachusetts, Michigan, Minnesota, Mississippi, Missouri, Montana, Nebraska, Nevada, New Hampshire, New Jersey, New Mexico, New York, North Carolina, North Dakota, Ohio, Oklahoma, Oregon, Pennsylvania, Rhode Island, South Carolina, South Dakota, Tennessee, Texas, Utah, Vermont, Virginia, Washington, West Virginia, Wisconsin, Wyoming.

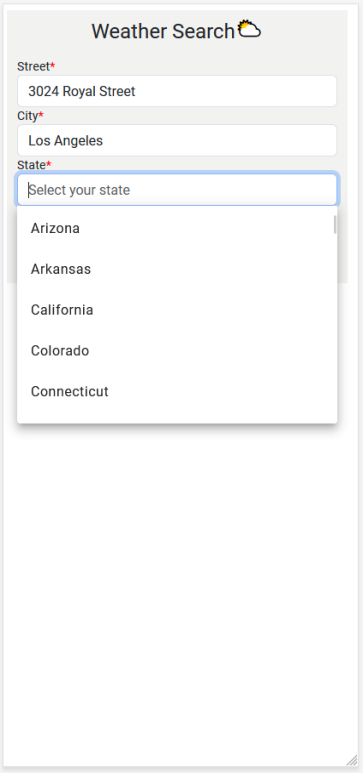
3.8 Responsive Design

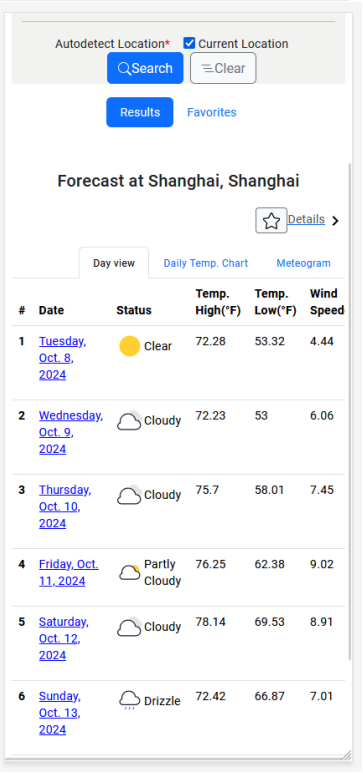
The following are snapshots of the webpage opened with Safari on an iPhone 11 Pro max. See the video for more details.

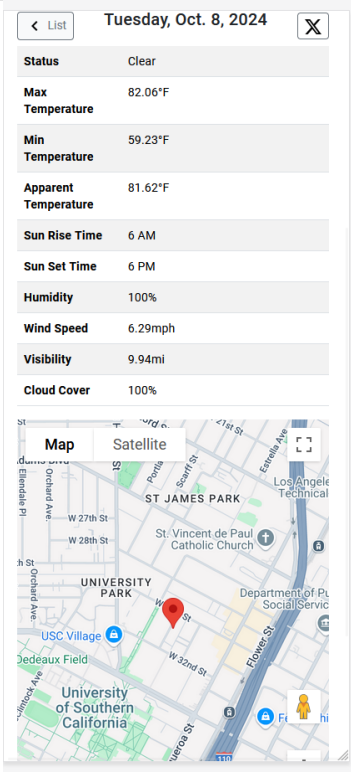

Weather Search
Street*
City*
State*
Select your state
Autodetect Location* Current Location
Search
Clear
Results
Favorites

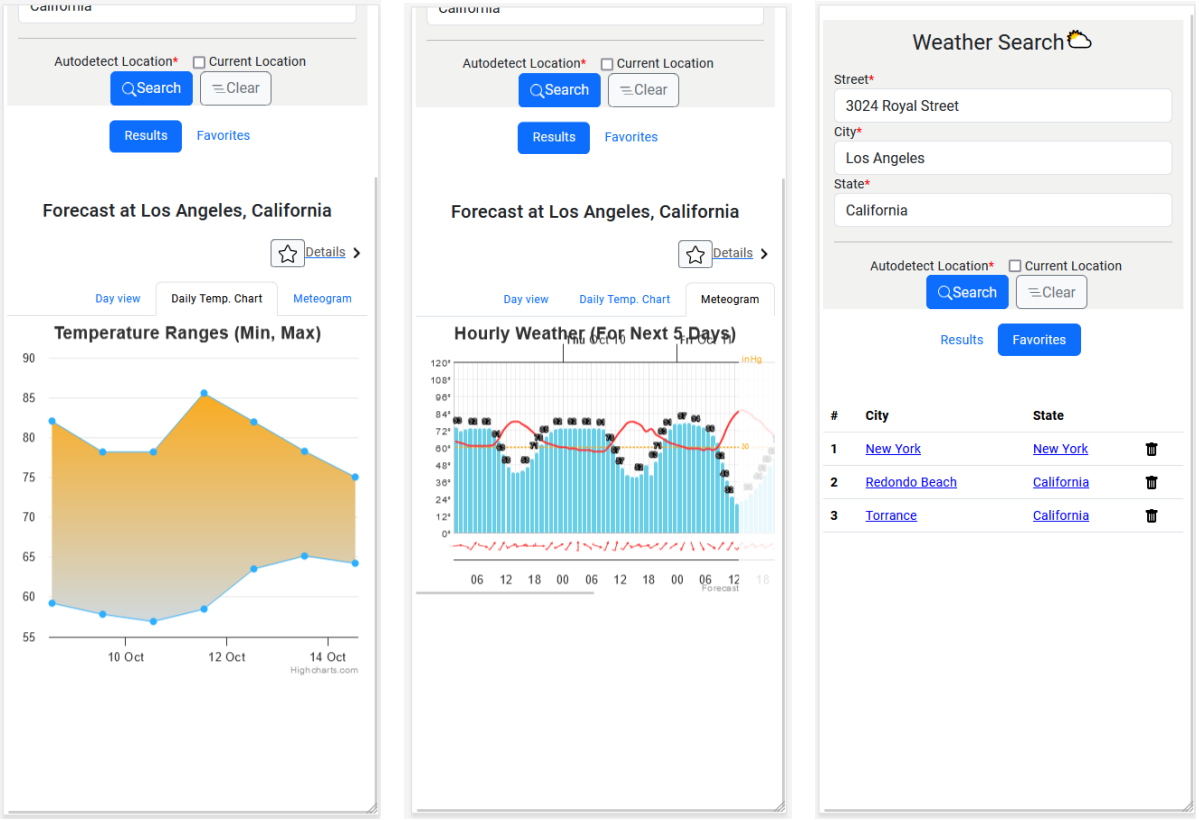

Weather Search
Street*
Please enter a valid street
City*
Please enter a valid city
State*
Select your state
Autodetect Location* Current Location
Search
Clear
Results
Favorites


Weather Search
Street*
3024 Royal Street
City*
Los
Los Angeles
Long Beach
Lynwood
Laguna Beach
Las Vegas


Weather Search
Street*
3024 Royal Street
City*
Los Angeles
State*
Select your state
Arizona
Arkansas
California
Colorado
Connecticut


Autodetect Location* Current Location
Search
Clear
Results
Favorites
Forecast at Shanghai, Shanghai
Details
Day view
Daily Temp. Chart
Meteogram
Date Status Temp. High(°F) Temp. Low(°F) Wind Speed
1 Tuesday, Oct. 8, 2024 Clear 72.28 53.32 4.44
2 Wednesday, Oct. 9, 2024 Cloudy 72.23 53 6.06
3 Thursday, Oct. 10, 2024 Cloudy 75.7 58.01 7.45
4 Friday, Oct. 11, 2024 Partly Cloudy 76.25 62.38 9.02
5 Saturday, Oct. 12, 2024 Cloudy 78.14 69.53 8.91
6 Sunday, Oct. 13, 2024 Drizzle 72.42 66.87 7.01


List
Tuesday, Oct. 8, 2024
Status Clear
Max Temperature 82.06°F
Min Temperature 59.23°F
Apparent Temperature 81.62°F
Sun Rise Time 6 AM
Sun Set Time 6 PM
Humidity 100%
Wind Speed 6.29mph
Visibility 9.94mi
Cloud Cover 100%
Map
Satellite
ST JAMES PARK
UNIVERSITY PARK
USC Village
University of Southern California



You must watch the video carefully to see how the page looks on mobile devices. All functions must work on mobile devices. Graders will test using Firefox Responsive Design Mode, with a smartphone and tablet (like iPhone 14 and iPad) 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. It is recommended that you use Firefox "Responsive Design Mode" Tool for testing.

4. API Documentation

4.1 Tomorrow.io API

To use the Tomorrow.io API, you need first to register for Tomorrow.io Account. This is the same App id used for the Assignment #2.

4.2 Geocoding API

Use Google Geocode API to convert the users form input into location information that can be sent to Tomorrow io. See:
https://developers.google.com/maps/documentation/geocoding/start

4.3 Places API

Use the Google Places Autocomplete API to get the search suggestion for the City text input.
More info: https://developers.google.com/maps/documentation/javascript/places-autocomplete

4.4 IPinfo API
Use IPinfo API to get location information from the users IP. See https://ipinfo.io/

4.5 X (formerly known as Twitter)

Refer the following link for details:
https://developer.twitter.com/en/docs/twitter-for-websites/tweet-button/overview

5. Libraries

Highcharts Angular Tutorial– For each of the charts in the pane/tabs, use HighCharts.

https://www.highcharts.com/blog/tutorials/highcharts-and-angular-7/

Note: You can use any additional Angular libraries and Node.js modules that you like. **You can use React** by you should not expect support on Piazza for it**.**

6. Implementation Hints

6.1 Images
All the icons needed in addition to the images provided for the results table in Assignment #2 can be obtained through bootstrap icons at:
https://icons.getbootstrap.com/

Images for Google Maps and X tweets are also provided.

6.2 ng-bootstrap Library

To get started with the ng-bootstrap toolkit, please see:
https://ng-bootstrap.github.io/#/home

ng-bootstrap will work with both Bootstrap 4 and 5, but with a specific version of Angular. Check compatibility and dependencies at:

https://ng-bootstrap.github.io/#/getting-started

Modules helpful for implementation:

Alerts - https://ng-bootstrap.github.io/#/components/alert/examples
Modal - https://ng-bootstrap.github.io/#/components/modal/examples

6.3 Get started with the Bootstrap Library

To get started with the Bootstrap toolkit, please refer to the links:

https://getbootstrap.com/docs/4.0/getting-started/introduction/
https://getbootstrap.com/docs/5.3/getting-started/introduction/

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

6.4 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 should use **Bootstrap 4 or later**. **Bootstrap 5** is recommended.

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/
https://getbootstrap.com/docs/5.3/forms/overview/

Bootstrap Tabs
https://getbootstrap.com/docs/4.0/components/navs/#tabs
https://getbootstrap.com/docs/5.3/components/navs-tabs/

Bootstrap Progress Bars
https://getbootstrap.com/docs/4.0/components/progress/
https://getbootstrap.com/docs/5.3/components/progress/

Bootstrap Alerts
https://getbootstrap.com/docs/4.0/components/alerts/
https://getbootstrap.com/docs/5.3/components/alerts/

Bootstrap Tooltip
https://getbootstrap.com/docs/4.1/components/tooltips/
https://getbootstrap.com/docs/5.3/components/tooltips/

Bootstrap Cards
https://getbootstrap.com/docs/4.0/components/card/
https://getbootstrap.com/docs/5.3/components/card/

6.5 Angular Material and Routing

- Angular Set up –
  https://angular.io/

- Angular Material Installation - https://material.angular.io/guide/getting-started

- Angular Material Tabs - https://material.angular.io/components/tabs/overview

- Angular Material Spinner – https://material.angular.io/components/progress-spinner/overview

- Angular Material Autocomplete - https://material.angular.io/components/autocomplete/overview

- Angular Routing – https://angular.io/guide/routing-overview

6.6 Material Icons

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/

6.7 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?lat=xxx&long=yyy
AWS - http://example.elasticbeanstalk.com/weatherSearch?lat=xxx&long=yyy
Azure - http://example.azurewebsites.net/weatherSearch?lat=xxx&long=yyy

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 Assignment #3 (this assignment), you should also **provide a sample link like the ones above**. When your grader clicks on this additional link, a JSON object should be returned with appropriate data.

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

Since this assignment 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.

6.9 AJAX call

You should send the request to the Node.js script(s) by calling an Ajax function (Angular HttpClient, fetch(), axios(), or jQuery.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 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. **fetch() or Angular HttpClient are recommended**.

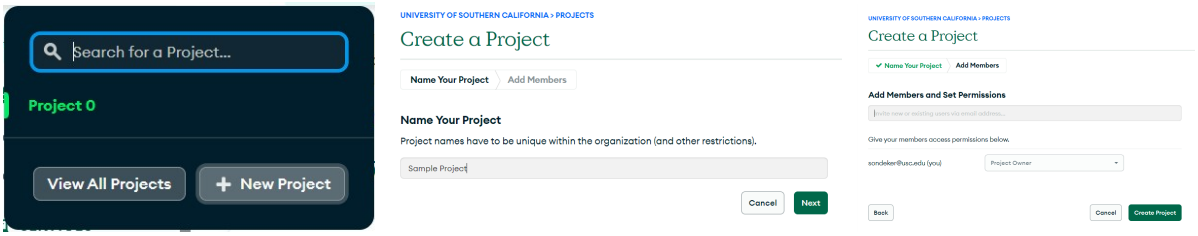
6.10 MongoDB Atlas (NoSQL Cloud Storage)

*MongoDB Atl*as is a source-available cross-platform document-oriented database program. It is classified as a NoSQL database program. MongoDB Atlas uses JSON-like documents with optional schemas. For more information, see: https://www.mongodb.com/docs/

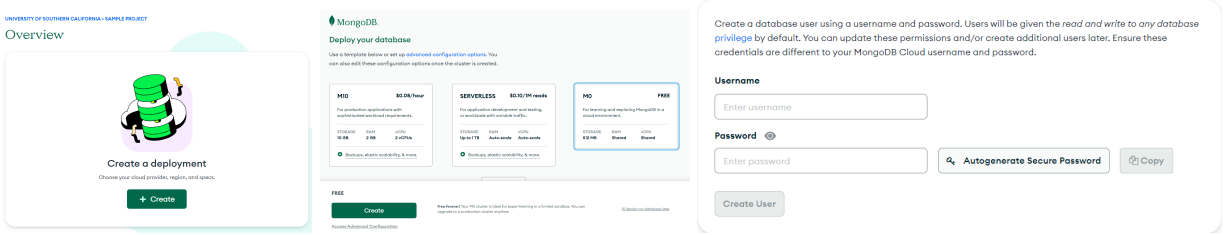
MongoDB on Google Cloud: https://www.mongodb.com/mongodb-on-google-cloud
MongoDB on AWS: https://www.mongodb.com/mongodb-on-aws
MongoDB on Azure: https://www.mongodb.com/mongodb-on-azure

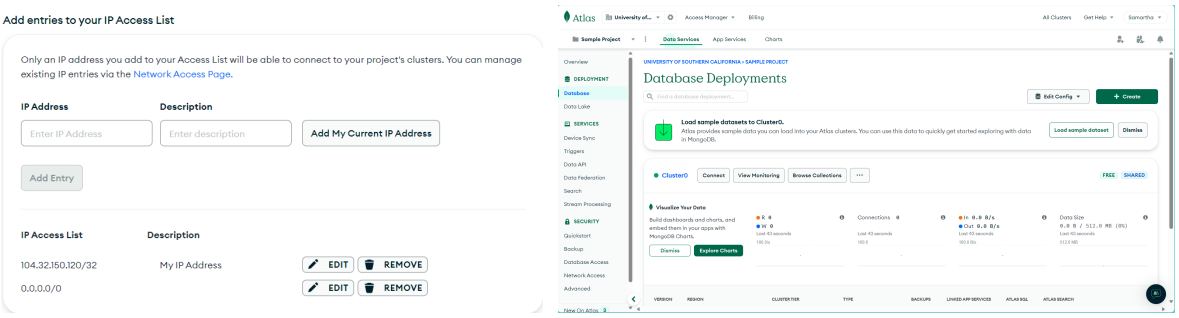
Once you set up an account in MongoDB Atlas, you will have to create a project to store your databases. In a project you will create a database to store collections which hold the favorite list data in NoSQL format. Below are the steps to set up a project and create a database.

Follow these steps create a project in which you can store databases for your application.

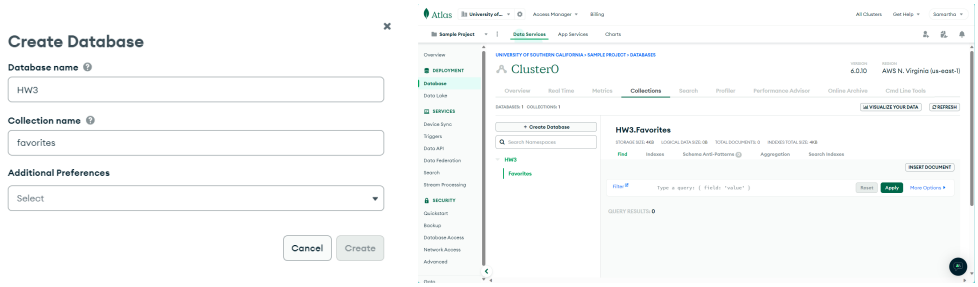


Follow these steps to create a deployment for the project by creating a cluster and providing user access and network access for the same. This would allow your application, running on a different server, to connect to MongoDB Atlas in the cloud.

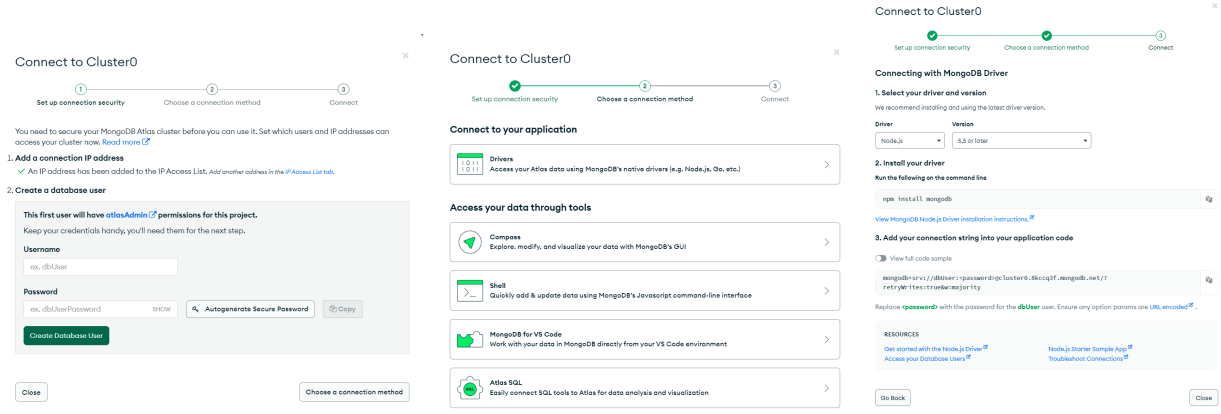




Follow these steps to create a database and a collection in that database. MongoDB stores data records as documents (specifically in BSON format) which are gathered in collections. A database stores one or more collections of documents.



Follow these steps to provide the instructions on how to connect to your MongoDB database from your application.



Once you have set up the database and the collection, you can connect to the database by adding the MongoDB node driver to your application. For more information how to add the driver and run queries on the database, refer to [MongoDB Node Driver — Node.js]

6.11 highcharts-angular

Refer the following documentation on **highcharts-angular** and how to use it.

https://www.npmjs.com/package/highcharts-angular

https://www.tutorialspoint.com/angular_highcharts/angular_highcharts_quick_guide.htm

https://github.com/highcharts/highcharts-angular

7. Files to Submit

In your course Table of Assignment page, on GitHub Pages, you must update the **Assignment #3 link** to refer to your new landing webpage for this exercise. All your files, front end and back-end must be hosted on the same service: Google Cloud, AWS or Azure. Graders will verify that this link is indeed pointing to one of the cloud services. Additionally, you need to provide **an additional link** to the URL of the Google Cloud/AWS/Azure service where the AJAX call is made with sample parameter values (i.e., a valid sample query, with lat/long. See section 6.6). The Graders will expect to see JSON to be returned.

Combine and submit all your front end and back-end files (HTML, JS, CSS, TS) electronically as a **SINGLE ZIP FILE** to the D2L Brightspace folder so that it can be compared to all other students' code.

The ZIP Archive should contain two folders: **source code** for your backend app (should be named backend) and one for your frontend (should be named by the framework you chose - **angular/react**). The folders should preserve the file hierarchy and include all necessary files to run your code on the graders machine, if so needed. Remove all temporary and build folders: node_modules, dist, etc., any images we provided or that are included in any library, or any code generated by the tools. You should submit a single .zip archive containing those two folders only on the top level.

Please do not just ZIP the top FOLDER containing all your files.

IMPORTANT:

- All explanations and clarifications provided in Piazza related to this homework are part of the homework description and grading guidelines. So please review all Piazza threads, before finishing the assignment. If there is a conflict between Piazza and this description and/or the grading guidelines, **Piazza always rules**.
- You **should not call any of the Tomorrow.io APIs directly from JavaScript**, bypassing the NodeJS proxy. Implementing any one of them in "client" JavaScript instead of NodeJS will result in a **4-point penalty.**
- Both front end and backend code must be served by the same cloud service. GitHub Pages should not be used for the front end, resulting in a **3-point penalty** if used.
- **Appearance of all views, tables, and charts should be like the snapshots in this document and the reference videos as much as possible.**