Homework 6: Search Server-side Scripting using Python Flask, JSON, and Tomorrow.io API

1. Objectives

- Get experience with the Python programming language and Flask framework.
- Get experience with the Google API, Tomorrow.io API and HighCharts Service.
- Get experience creating web pages using HTML, CSS, JavaScript, DOM, JSON format and XMLHttpRequest object.
- Get experience using JSON parsers in Python and JavaScript.
- Getting hands-on experience in GCP, AWS or Azure.

1.1. Cloud Exercise

The backend of this homework must be implemented in the cloud on Google Cloud App Engine, AWS or Azure, using Python.

- See assignment #5 for installation of needed components on GCP, AWS or Azure.
- See the hints in section 3; a lot of reference material is provided to you.
- For Python and Flask kick-start, please refer to the Lecture slides on the class website.
- You must refer to the grading guidelines, the video, the specs, and Piazza. Styling will be graded, and the point's breakup is mentioned in the grading guidelines.

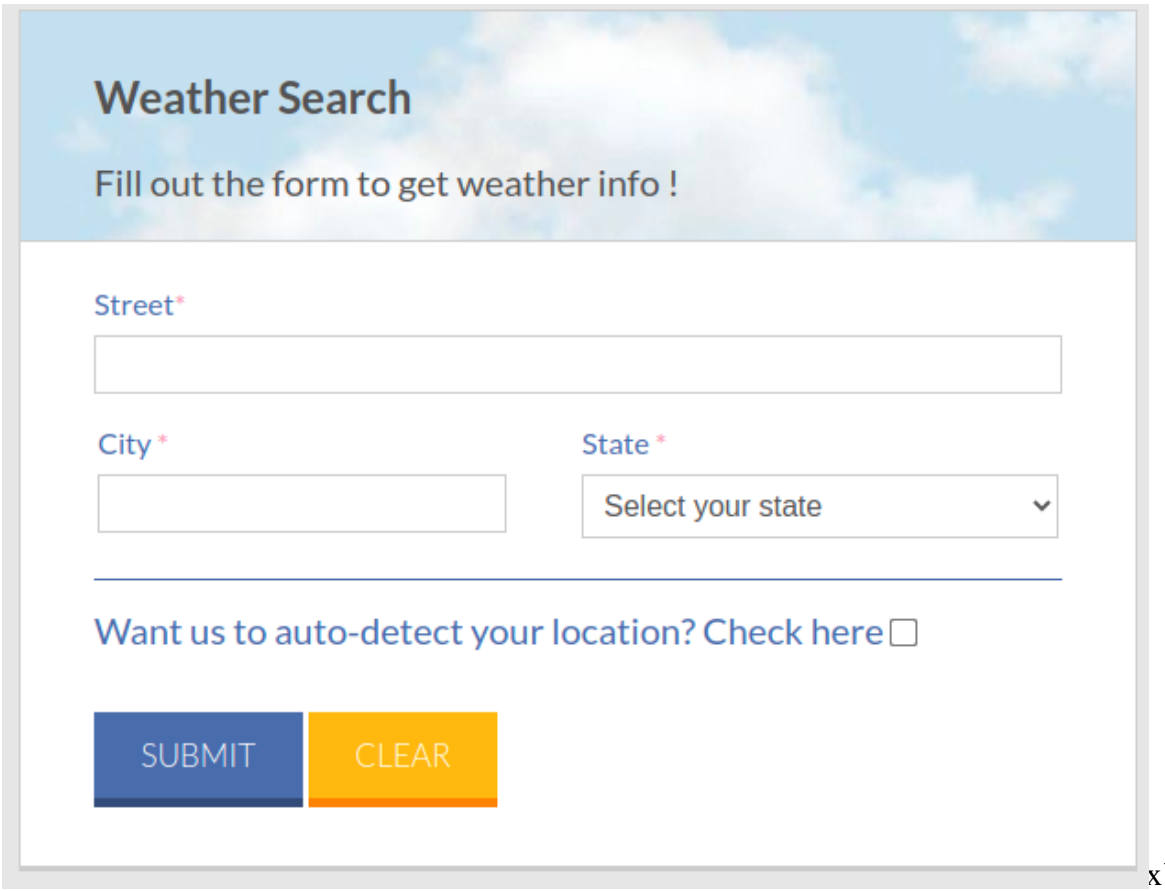
2. Description

In this exercise, you are asked to create a webpage that allows you to search for weather information using the Tomorrow.io API, and the results will be displayed in a card and tabular format. The page will also provide day's weather details.

2.1. Description of the Search Form

The user first opens a web page (for example, **weather.html,** or any valid web page name). You should use the ipinfo.io API (See hint 3.3) to fetch the user's geolocation if the location checkbox is checked else the user must enter a Street, City and State to search.

An example is shown in **Figure 1**.



x`

If the **Check here** checkbox is checked then all the fields i.e., Street, City and State should reset text and disable the fields.

The search form has two buttons:

- **SUBMIT** button: Selecting this button performs a search of the given location, and if location is found it returns weather information. An example of valid input is shown in Figure 2. Once the user has provided valid input, your client JavaScript should send a request to your web server Python script with the form inputs. You must use GET to transfer the form data to your web server (do not use POST, as you would be unable to provide a sample link to your cloud services). A Python script using Flask will retrieve the form inputs and send it to the *Tomorrow.io API* weather information service. You need to use the *Flask* Python framework to make all the API calls.

  If the user clicks on the SUBMIT button without providing a value in the "Street", "City" and "State" field or checking the location checkbox, you should show an error "tooltip" that indicates which field is missing. Examples are shown in Figure 2a, 2b and 2c.

  Using XMLHttpRequest or any other JavaScript calls for anything other than calling your own "cloud" backend will lead to a 4-point penalty. **Do not call the Tomorrow.io API directly from JavaScript.**

Define routing endpoints and make your API call from the Python backend. The recommended tutorial for *Flask* and more importantly, routing, can be found at the following link: https://flask.palletsprojects.com/en/1.1.x/

- **CLEAR** button: This button must clear the result area (below the search area) and set all fields to the default values in the search area. The **CLEAR** operation must be done using a JavaScript function.

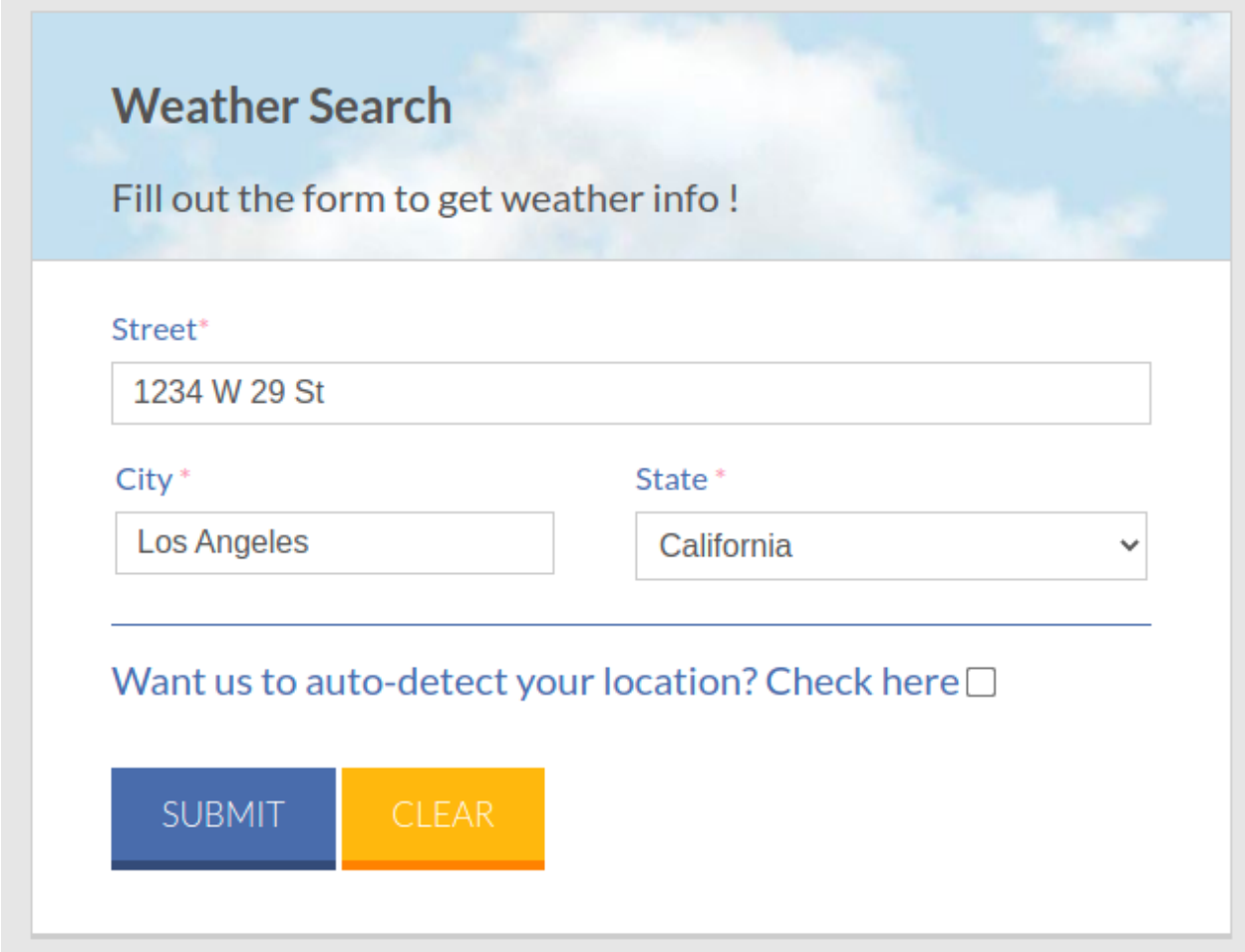


Figure 2: An Example of a Valid Search

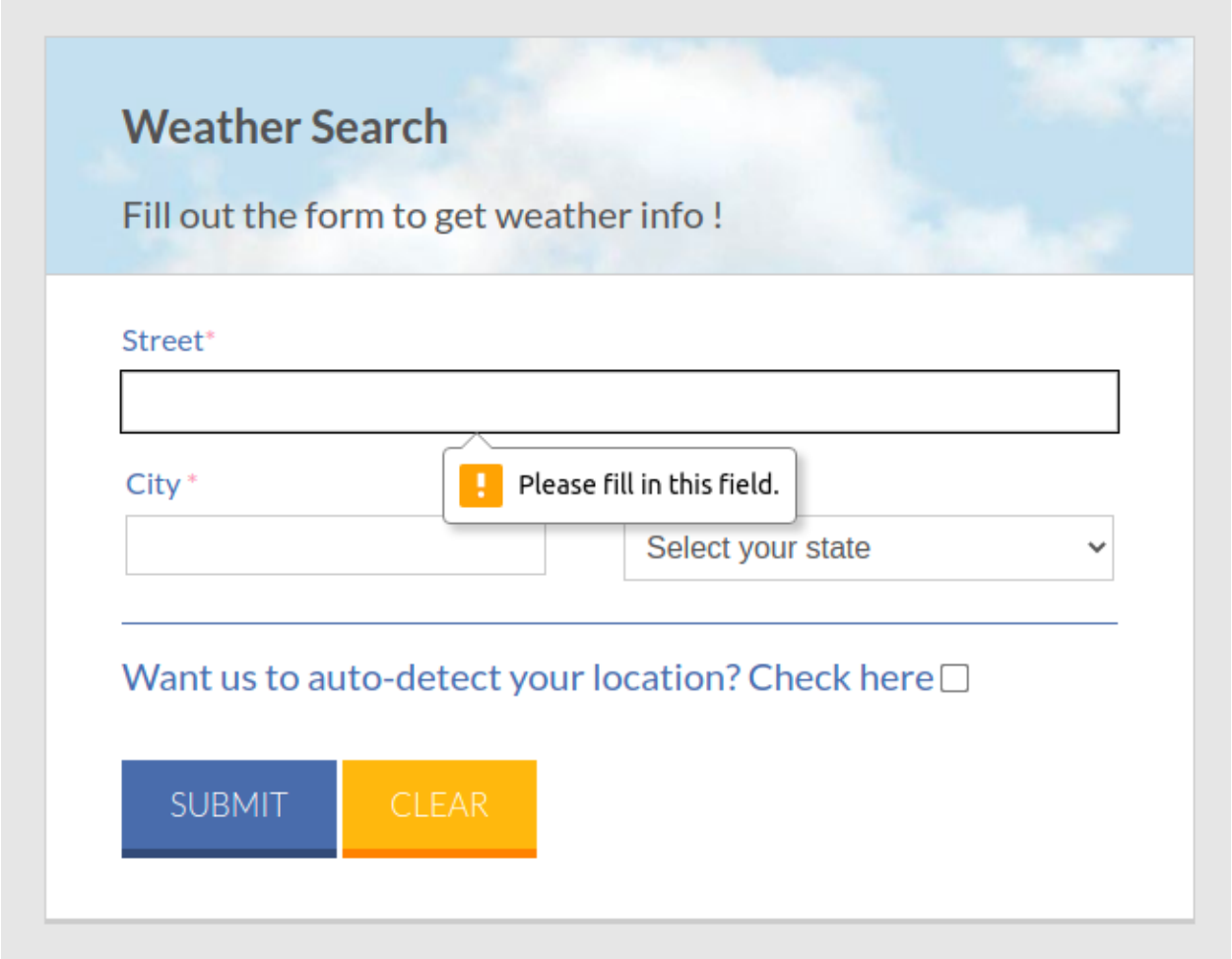


Figure 2: An Example of an Invalid Search

2.2 Displaying Weather Results

In this section, we outline how to use the form inputs to construct the calls to the RESTful web services to the *Tomorrow.io API* service and display the result in the web page.

The *Tomorrow.io API* is documented here:

https://docs.tomorrow.io/reference/welcome

If the Street, City and State information is used to get weather results, your client JavaScript uses the input address to get the geocoding via *Google Maps Geocoding API*. The *Google Maps Geocoding API* is documented here:

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

The Google Maps Geocoding API expects two parameters:

- ***address*:** The street address that you want to geocode, in the format used by the national postal service of the country concerned. Additional address elements such as business names and unit, suite or floor numbers should be avoided.
- ***key*:** Your application's API key. This key identifies your application for purposes of quota management. (Explained in Section 3.2).

2.2.1 Geocoding

An example of an HTTP request to the Google Maps Geocoding API, when the location address is "University of Southern California, CA" is shown below:

https://maps.googleapis.com/maps/api/geocode/json?address=University+of+Southern+California+CA&key=YOUR_API_KEY

The response includes the latitude and longitude of the address.

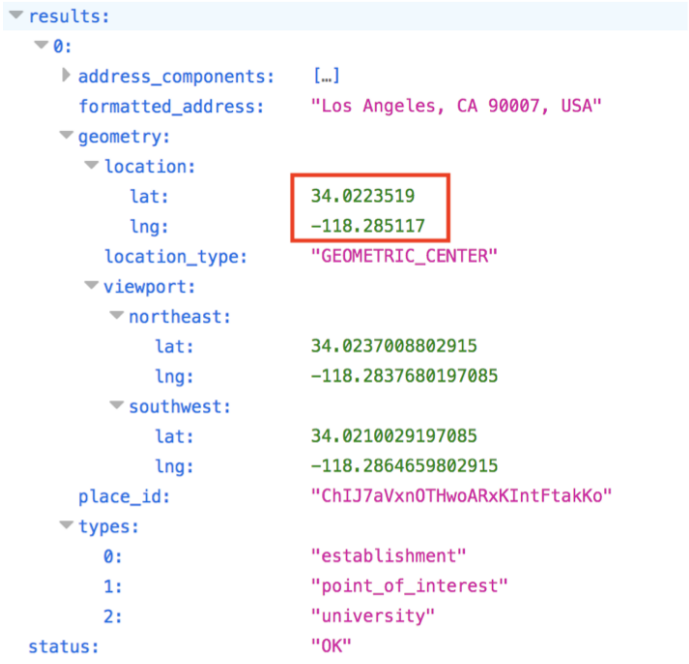


Figure 3 shows an example of the JSON object returned in the Google Maps Geocoding API web service response.

The latitude and longitude of the address are used when constructing a RESTful web service URL to retrieve weather information.

2.2.2. Tomorrow.io API Service

The *Tomorrow.io API Retrieve TimeIines (Basic)* service is documented here:

https://docs.tomorrow.io/reference/get-timelines

Click the **Python** tab in the example, to see the format of the URL, querystring, and headers.

The *Tomorrow.io API Retrieve Timelines* service expects the following parameters:

- ***apikey:*** Your application's API key. This key identifies your application for purposes of quota management.
- ***locations:*** The *location* around which to retrieve weather information. Acceptable formats for location(https://docs.tomorrow.io/reference/api-formats#locations)
- ***fields:*** The fields which you will require are the following:
    - temperature
    - temperatureApparent
    - temperatureMin
    - temperatureMax
    - windSpeed
    - windDirection
    - humidity
    - pressureSeaLevel
    - uvIndex
    - weatherCode
    - precipitationProbability
    - precipitationType
    - sunriseTime
    - sunsetTime
    - visibility
    - moonPhase
    - cloudCover
- ***timesteps:*** The timesteps needed for the assignment are '1h','1d'. Further, details of timesteps are documented here: https://docs.tomorrow.io/reference/data-layers-overview#timestep-availability.
- ***timezone:*** There are various timezones available here https://docs.tomorrow.io/reference/api-formats#timezone. Use "America/Los_Angeles".

- ***units:*** Unit of the fields. There are two options, "imperial" and "metric". Use "imperial".

An example of an HTTP request to the *Tomorrow.io API* that searches for the nearby weather information near the University of Southern California, Los Angeles, CA is shown below:

- Current API call
  usage: Current weather card view

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

- Timestep = 1day API call

usage: Table Details, Detailed Summary of weather card view and Temperature Range(Min, Max) Chart.

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

A sample response is shown in Figure 5.


```
{
    "data": {
        "timelines": [
            {
                "timestep": "1d",
                "startTime": "2021-09-07T06:00:00-07:00",
                "endTime": "2021-09-21T06:00:00-07:00",
                "intervals": [
                    {
                        "startTime": "2021-09-07T06:00:00-07:00",
                        "values": {
                            "temperatureMax": 83.03,
                            "temperatureMin": 63.01,
                            "weatherCode": 2000,
                            "windSpeed": 9.1,
                            "precipitationProbability": 0,
                            "precipitationType": 0,
                            "humidity": 100,
                            "sunriseTime": "2021-09-07T06:30:00-07:00",
                            "sunsetTime": "2021-09-07T19:10:00-07:00",
                            "visibility": 9.94,
                            "moonPhase": 0
                        }
                    },
                    {
                        "startTime": "2021-09-08T06:00:00-07:00",
                        "values": {
                            "temperatureMax": 86.45,
                            "temperatureMin": 63.27,
                            "weatherCode": 2000,
                            "windSpeed": 14.99,
                            "precipitationProbability": 0,
                            "precipitationType": 0,
                            "humidity": 100,
                            "sunriseTime": "2021-09-08T06:33:20-07:00",
                            "sunsetTime": "2021-09-08T19:10:00-07:00",
                            "visibility": 9.94,
                            "moonPhase": 1
                        }
                    },
```


Figure 5: shows an example of the JSON response returned by the *Tomorrow.io API* service response.

The Python script should pass the returned JSON object to the client side or parse the returned JSON and extract useful fields and pass these fields to the client side in a **JSON-formatted object**. You should use JavaScript to parse the JSON object, extract the needed fields, and display the results in a tabular format and a card view containing current day's weather information. A sample output is shown in **Figure 6(a) and 6(b)**. The displayed table includes five columns: **Date**, **Status**, **Temp High**, **Temp Low**, **Wind Speed**.

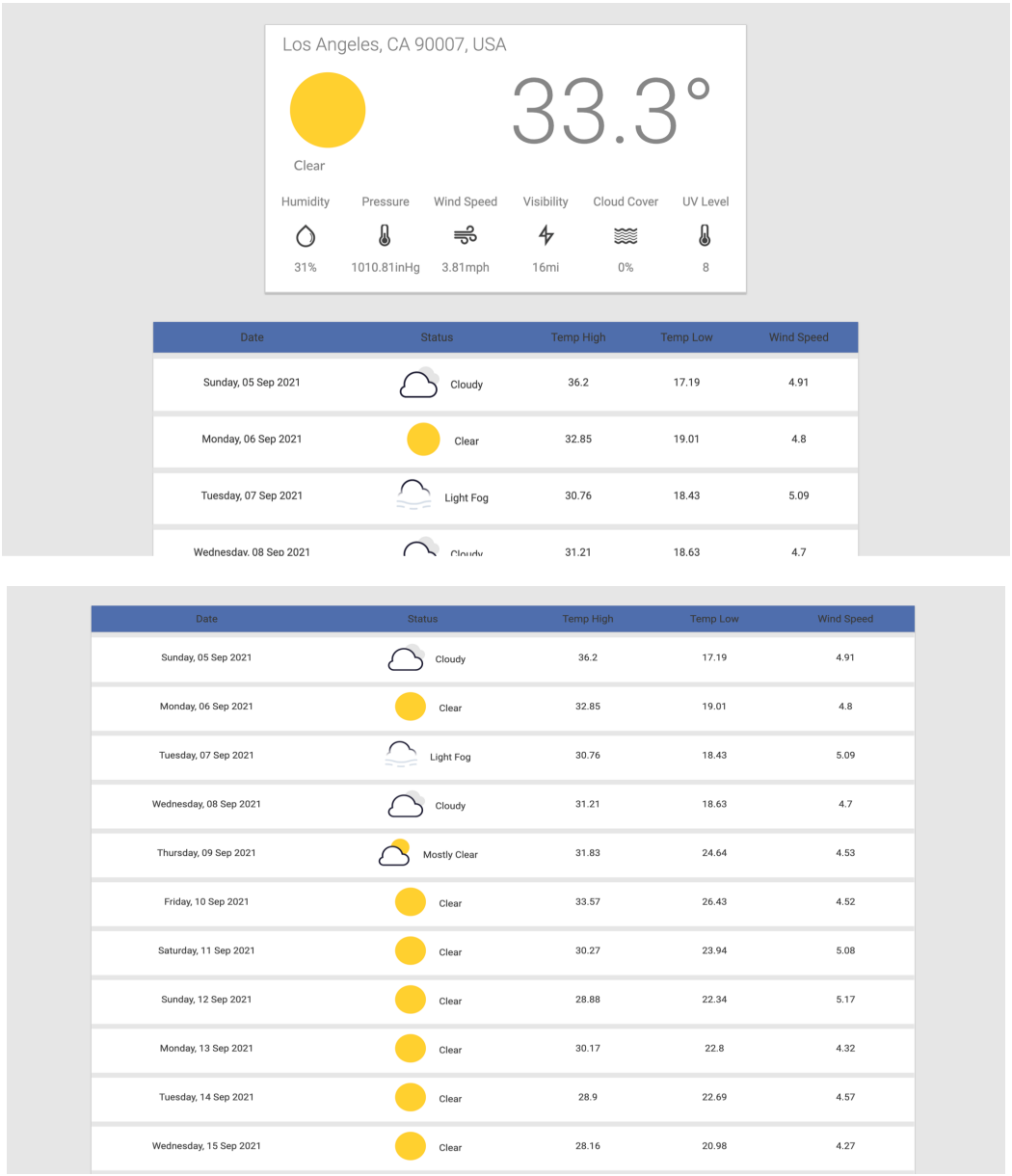


Figure 6(a) and 6(b): Output of Search Results

When the search result contains a record, you need to map the data extracted from the API result to render the HTML result card and table as described in **Table 1** and **Table 2** respectively.

Table 1: Mapping the result from API into HTML card

HTML Card	**API Service Response**
Location	• In case the location is being obtained

	from the google geocode api, the value of the "formatted_address" field. ● In case the location is being detected using ipinfo, you may use the city, region, country field from the response to construct the location string.
Temperature	The value of the *"temperature"* attribute is part of the values object.
Humidity	The value of the "humidity" attribute is part of the values object. It should be shown with the following icon and appropriate units. https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-16-512.png
Pressure	The value of the "pressure" attribute is part of the values object. It should be shown with the following icon and appropriate units. https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-25-512.png
Wind Speed	The value of the "windSpeed" attribute is part of the values object. It should be shown with the following icon and appropriate units. https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-27-512.png
Visibility	The value of the "visibility" attribute is part of the values object. It should be shown with the following icon and appropriate units. https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-30-512.png
Cloud Cover	The value of the "cloudCoveressure" attribute is part of the values object. It should be shown with the following icon and appropriate units. https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-28-512.png
UV Level	The value of the "uvIndex" attribute is part of the values object. It should be shown with the

	following icon and appropriate units. https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-24-512.png

Table 2: Mapping the result from API into HTML table

HTML Table Column	**API Service Response**
Date	The value of "*startTime"* attributes that is part of the intervals objects.
Status	The value of *"weatherCode"* attribute is part of the values object, which is mapped to text description and image.
Temp High	The value of the *"TemperatureMax"* attribute is part of the values object.
Temp Low	The value of the *"TemperatureMin"* attribute is part of the values object.
Wind Speed	The value of the *"windSpeed"* attribute is part of the values object.

Reference for HTML Table Column "Status": Map the weatherCode to text description and image URLs as shown in **Figure 7** and refer to the documentation at:

https://docs.tomorrow.io/reference/data-layers-core :

Weather Code	Description	Icon
4201	Heavy Rain	
4001	Rain	
4200	Light Rain	
6201	Heavy Freezing Rain	
6001	Freezing Rain	
6200	Light Freezing Rain	
6000	Freezing Drizzle	
4000	Drizzle	
7101	Heavy Ice Pellets	
7000	Ice Pellets	
7102	Light Ice Pellets	
5101	Heavy Snow	
5000	Snow	
5100	Light Snow	
5001	Flurries	
8000	Thunderstorm	
2100	Light Fog	
2000	Fog	
1001	Cloudy	
1102	Mostly Cloudy	
1101	Partly Cloudy	
1100	Mostly Clear	
1000	Clear, Sunny	

Figure 7: Map of weatherCode to respective weather description and icon URLs

The raw images for the above table can be found here:

https://github.com/Tomorrow-IO-API/tomorrow-weather-codes/tree/master/color

Light-wind: https://www.clipartmax.com/png/middle/31-318730_cold-wind-blowing-vector-wind-blow-icon.png

Wind: https://www.clipartmax.com/png/middle/31-319198_winds-weather-symbol-vector-weather-symbol-for-wind.png

StrongWind: https://www.clipartmax.com/png/middle/2-27821_wind-clipart-forecast-icon-line-icon-weather-wind-windy-wind-clipart.png

You are required to map the Weather Code you obtain from the api response to the summary text shown above as well as its corresponding icon.

Note: For items with more than one icon (e.g., 1100 - Mostly Clear) you can default to the "morning" icon. On hovering, the Card should have a drop shadow. Please refer to the video for all the features and behavior to be implemented.

2.3 Displaying Detailed Summary of the Weather

In the search result table, if the user clicks on the date of a record, the page should display a detailed description of the daily weather. The request needs same parameters for API call as in 1st (output should be JSON), and the page should request the detailed information using the *Tomorrow.io API* and direct to a section populated with this information, as shown in **Figure 8** and **Figure 9**:

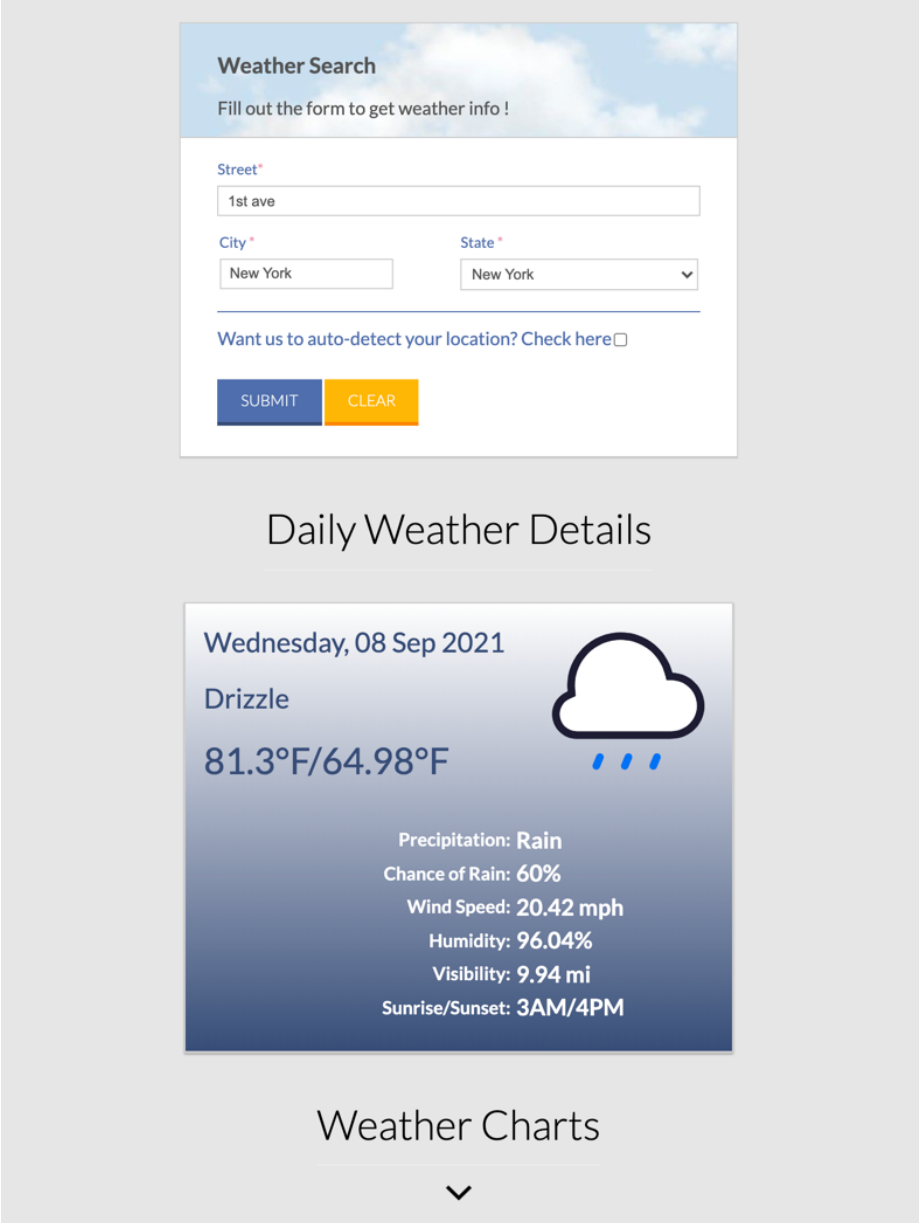


Figure 8: Daily Weather Details

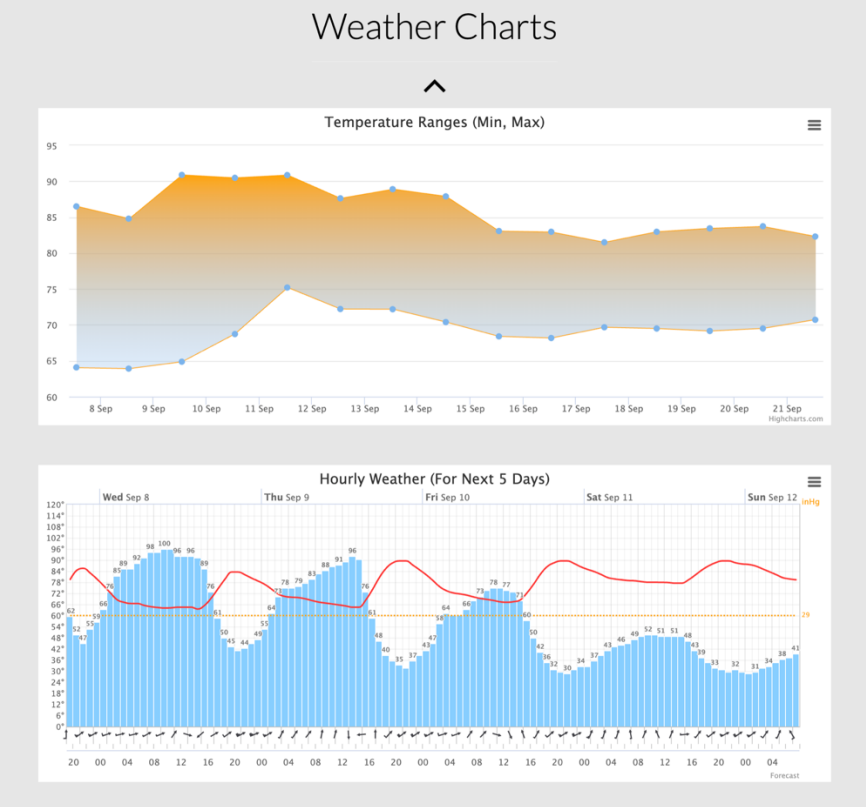


Figure 9: Display on Arrow toggle

Below the Daily Weather Detail, there is a chart for the hourly weather. This is hidden by default. When the user clicks on the button ﹀, the "Weather Charts" sub-section should be expanded, and when the user clicks on the button ^, the "Weather Charts" sub-section should be hidden (if it is open) and vice versa (see the video for the behavior).

2.4 *Highcharts API* Service

The *Highcharts API* service is documented here:

https://www.highcharts.com/docs/index

Temperature Range (Min, Max) weather chart should consist of a daily based plot over a period from current day (not the selected) to next 15 days. For the reference of the development of the chart, see here:

https://www.highcharts.com/demo/arearange

Hourly Weather (For Next 5 days) weather chart should consist of an hourly based plot over a period from current time (not the selected) to next 5 days. 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]

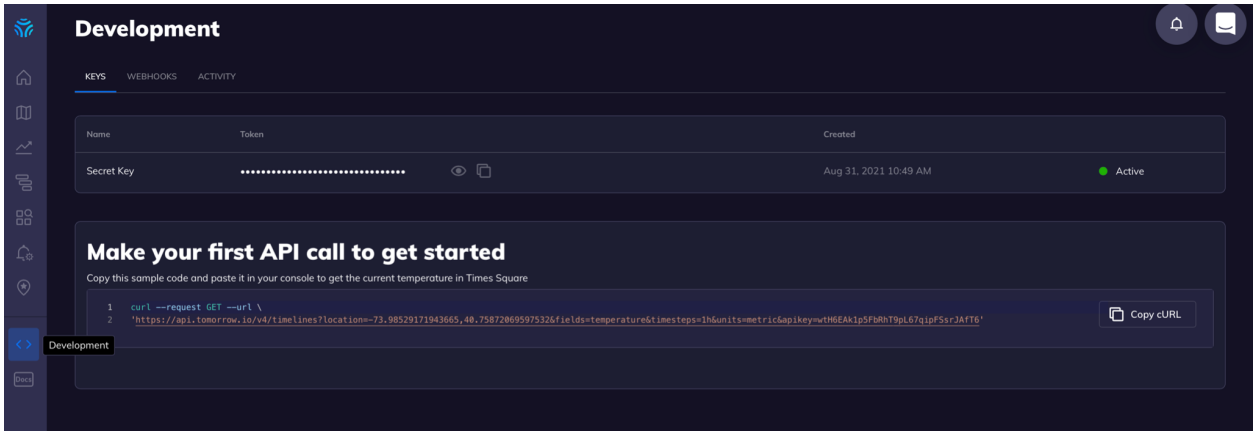
For the reference of development of chart below is the link:

https://www.highcharts.com/demo/combo-meteogram#https://www.yr.no/place/United_Kingdom/England/London/forecast_hour_by_hour.xml

3. Hints

3.1 How to get Tomorrow.io API Key

- To get a Tomorrow.io API key, please follow these steps:
- Create a new account at: https://app.tomorrow.io/signup?planid=60d46beae90c3b3549a59ff3
- Go to the Development Tab on the right panel and Secret Key is your API Key.



3.2 How to get Google API Key

- To get a Google API key, please follow these steps:
- Go to the Google Developers Console: https://console.developers.google.com/flows/enableapi?apiid=geocoding_backend&keyType=SE RVER_SIDE&reusekey=true.
- Create a project.
- At the Google APIs' guide page, click "Get a key" and select a created project.
  Note that you should NOT use a google account associated with a USC email. Preferably use a Gmail account.

3.3 Get IPInfo.io API Key

- Go to https://ipinfo.io/ and sign up for free
- A token would be provided after successful sign up

An example call is as follows: https://ipinfo.io/?token=YOUR_TOKEN_ID

3.4 Deploy Python file to the cloud (GCP/AWS/Azure)

You should use the domain name of the GAE/AWS/Azure service you created in Assignment #5 to make the request. For example, if your GAE/AWS/Azure server domain is called **example.appspot.com** or **example.elasticbeanstalk.com** or **example.azurewebsites.net**, the following links will be generated:

GAE - http://example.appspot.com/index.html
AWS - http://example.elasticbeanstalk.com/index.html
Azure - http://example.azurewebsites.net/index.html

The *example* subdomain in the above URLs will be replaced by your choice of subdomain from the cloud service. You may also use a different page than index.html.

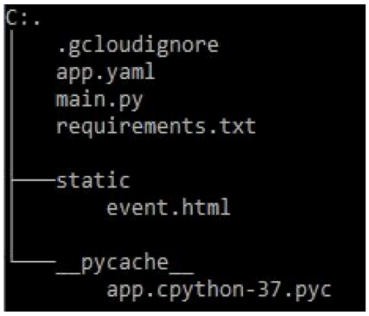
For example, if your GAE server domain is called **example.appspot.com**, the following links will be generated: http://example.appspot.com/index.html

The *example* subdomain in the above URLs will be replaced by your choice of subdomain from the cloud service. You may also use a different page than index.html. Files to deploy:

1. client-side files (HTML+CSS+JS)

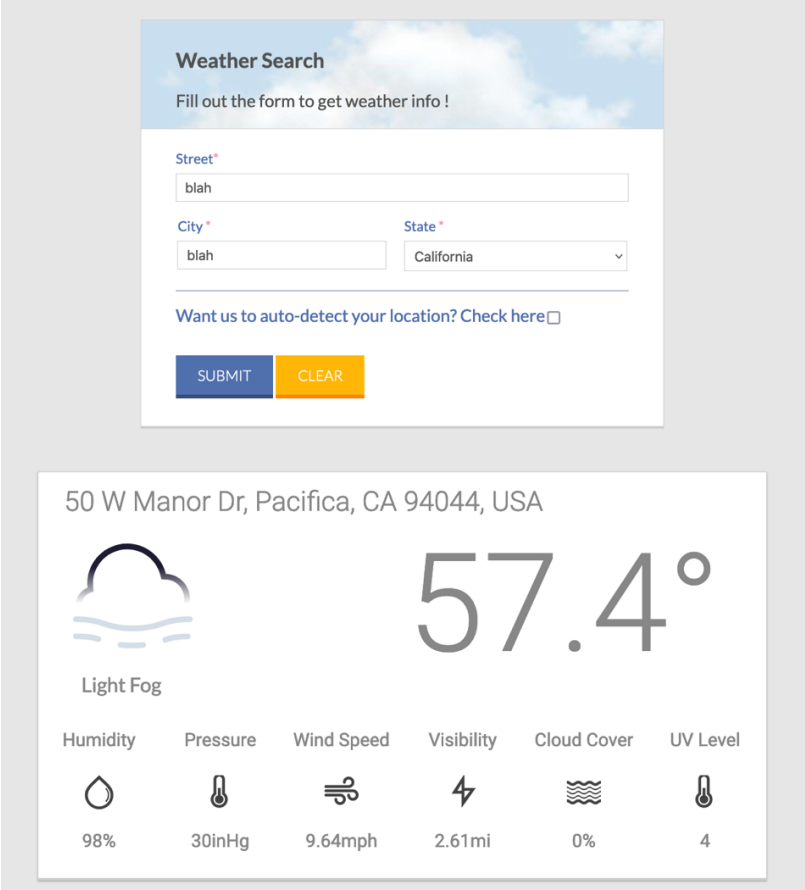
2. server-side file (main.py), .yaml, requirements.txt

The project structure should be like the following one:

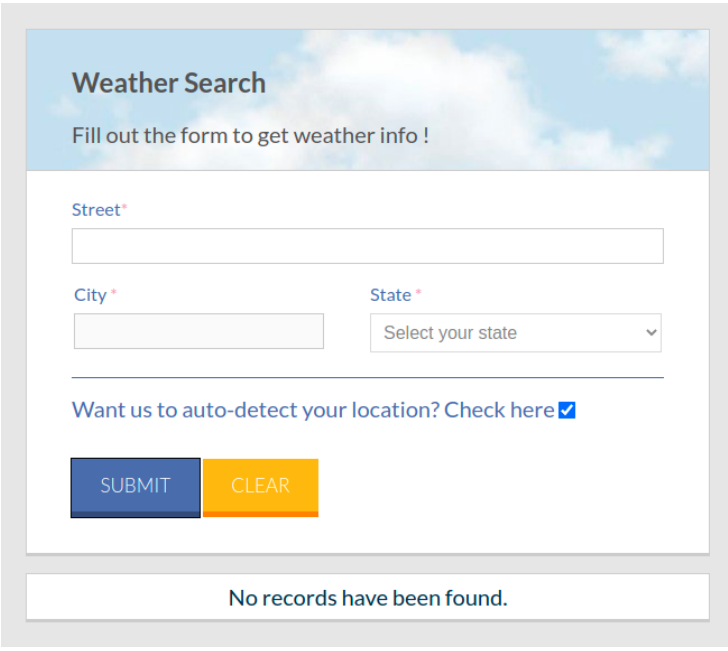


3.5 Behavior for Search Forms

When random/bogus values for the Street and City fields are entered in search form along with any State selected, It should still work and display weather card view and table records with the weather information as shown here:



If the API service returns an empty result set due to limited API calls allowed which would be a rare scenario as Tomorrow.io provide sufficient API calls limit, but still in case it happens then the page should display “No records have been found” as shown here:



3.6 Parsing JSON-formatted data in Python

Information on how to parse JSON-formatted data in Python is available here:

https://docs.python.org/3/library/json.html

If you use your cloud server as a “proxy” pass-through, you do not have to decode and encode the JSON.

3.7 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

4. Files to Submit

In your course homework page, you should update the **Homework 6** link to refer to your new initial web search page for this exercise (for example, **weather.html**). Your files must be hosted on GAE, AWS or Azure cloud service. Graders will verify that this link is indeed pointing to GAE.

Also, submit your source code file to DEN D2L. Submit a ZIP file of both front-end and back-end code, plus any additional files needed to build your app (e.g., yaml file). **The timestamp of the ZIP file will be used to verify if you used any "grace days."**

****IMPORTANT**:**

- All discussions and explanations 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 can use jQuery for Homework 6 but it is not required.
- You **should not call the Tomorrow.io APIs directly from JavaScript**, bypassing the Python proxy. Implementing any one of them in JavaScript instead of Python will result in a **4-point penalty.** Other APIs can be called from JavaScript.
- **APPEARANCE OF CARD VIEW, TABLE AND CHARTS should be similar to the reference video as much as possible.**