

Homework 6: Server-side Scripting

1. Objectives

- Get experience with the PHP programming language
- Get experience with Google Geocode and the forecast.io API.
- Get experience using an XML and a JSON parser in PHP.
- Getting hands-on experience in GCP App Engine, AWS or Azure.

1.1. Cloud exercise

The backend of this homework must be implemented in the cloud on GCP, AWS or Azure using PHP.

See homework 5 for installation of either one of these platforms. You only have to select one platform to implement your backend.

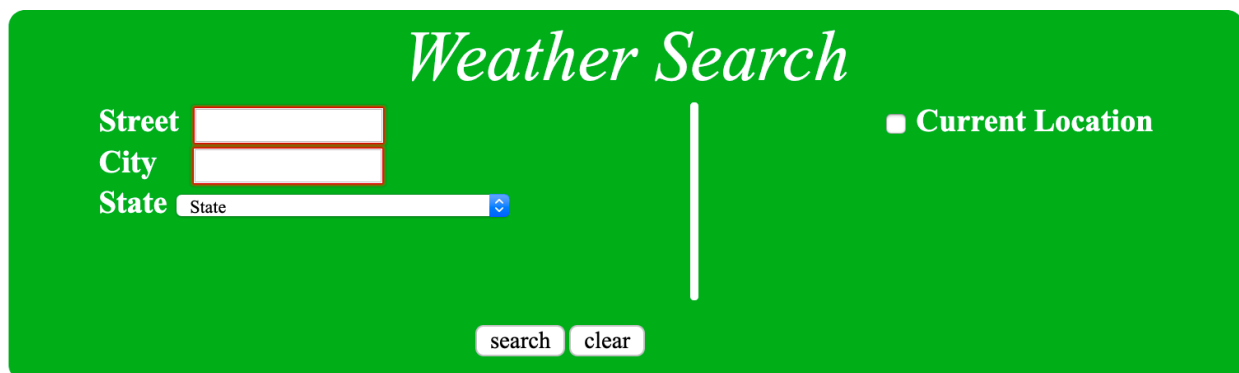
2. Description

In this exercise, you are asked to create a webpage that allows you to search for weather information using the Google Geocode and forecast.io API, and the results will be displayed in tabular format.

2.1. Description of the Search Form

A user first opens a page, called **forecast.php (or any valid web page name)**, where they can enter an address (street, city, and state). Figure 1 shows initial search screen which requests to give the location to provide weather details. In Figure 1, the State field includes a list of all US states, which are provided in section 3.7. In case the user wants to provide their current location, the user can select the Current Location checkbox, which will fetch the location using IP-API as given in section 3.3

If the user selects the current location checkbox, the input fields for Street, City and State should be cleared and disabled. Please refer the video for this behavior.



The image shows a web form titled "Weather Search" in a stylized green font. The form has a light blue background. On the left, there are three input fields labeled "Street", "City", and "State". The "Street" and "City" fields are text inputs, and the "State" field is a dropdown menu showing "State" with a blue arrow. To the right of these fields is a checkbox labeled "Current Location". At the bottom of the form, there are two buttons: "search" and "clear".

Figure 1: Initial Search Screen

The image shows a web form titled "Weather Search" on a green background. The form has three input fields: "Street" with the value "Figueora street", "City" with the value "Los Angeles", and "State" which is currently open as a dropdown menu. The dropdown menu lists US states and the District of Columbia, with "District Of Columbia" highlighted in blue. To the right of the "State" field is a "Current Location" checkbox. Below the input fields are two buttons: "Search" and "clear".

Figure 2: List of States

The search form has two buttons:

- **Search** button: This button validates whether the user provided values for street address, city and state. The validation should be implemented in a JavaScript function. If the user did not enter one of the data items, then an alert should be shown with an appropriate message prompting the user to provide complete information. An example of the alert is shown in Figure 3, and an example of valid input is shown in Figure 3. Once the user has provided valid data, your script should send a request to your web server for **forecast.php (or whatever your valid web page name is)** with the form data. You can use either GET or POST to transfer the form data to the web server. A PHP script will grab the data and send it to the Google GeoCode and forecast.io web services in sequence.
- **Clear** button: This button must clear the result area, all text fields, unselect the State value. The Clear operation is done using a JavaScript function.

The image shows a web form titled "Weather Search" on a green background. On the left, there are three input fields: "Street" with the value "Figueora street", "City" which is empty, and "State" with a dropdown menu showing "State". To the right of these fields is a vertical line and a checkbox labeled "Current Location". At the bottom of the form are two buttons: "search" and "clear". Below the form, a grey rectangular box contains the text "Please check the input address."

Figure 3: An Error Message When Providing Incomplete Address (e.g., the values of city and state fields are missing)

The image shows the same "Weather Search" form as in Figure 3, but with valid input. The "Street" field contains "figueora street", the "City" field contains "Los Angeles", and the "State" dropdown menu shows "California". The "Current Location" checkbox is still present to the right of the vertical line, and the "search" and "clear" buttons are at the bottom.

Figure 4: An Example of Valid Input

2.2. Displaying Results

In this section, we explain how to use the form data to construct web services calls to Google GeoCode and forecast.io API and display the result in your page.

The PHP script **forecast.php** (or whatever your valid page name is) will use the address information (street, city, and state) to construct a web service URL to query the Google Geocode API appropriately:

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

How to create a Google API key (YOUR_API_KEY int the URL above) is explained in section 3.2. The response of this URL is an XML-formatted object. Two key pieces of data returned are the latitude and longitude values for the given address. Figure 5 shows an example of the XML data returned in the Google Geocode web service response. You will see in the next step that you

must use the extracted latitude and longitude from the location tag to look up the weather for a given location.

```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
  <status>OK</status>
  <result>
    <type>route</type>
    <formatted_address>S Figueroa St, Los Angeles, CA, USA</formatted_address>
    <address_component><img alt="redacted" data-bbox="365 238 395 250"/></address_component>
    <address_component><img alt="redacted" data-bbox="365 253 395 265"/></address_component>
    <address_component><img alt="redacted" data-bbox="365 268 395 280"/></address_component>
    <address_component><img alt="redacted" data-bbox="365 283 395 295"/></address_component>
    <address_component>
      <long_name>United States</long_name>
      <short_name>US</short_name>
      <type>country</type>
      <type>political</type>
    </address_component>
    <geometry>
      <location>
        <lat>33.9147445</lat>
        <lng>-118.2827372</lng>
      </location>
      <location_type>GEOMETRIC_CENTER</location_type>
    </geometry>
  </result>
</GeocodeResponse>
```

Figure 5: A Sample Result of Google GeoCode Query

In case the user selects the current location checkbox, the co-ordinates are found using the IP-API as described in section 3.3.

Next, using the latitude and longitude values, another web service URL needs to be constructed to query the forecast.io API to get the weather information for the given location such as:

[https://api.forecast.io/forecast/\[YOUR_API_KEY\]/\[LATITUDE,LONGITUDE\]?exclude=minutely,hourly,alerts,flags](https://api.forecast.io/forecast/[YOUR_API_KEY]/[LATITUDE,LONGITUDE]?exclude=minutely,hourly,alerts,flags)

When constructing the Forecast.io web service API call, you should provide five parameters:

- The first parameter is your Forecast.io API key (a.k.a. the Dark Sky API key). How to create this key is explained in section 3.1.
- The second and the third parameter are the latitude and longitude values which are extracted from the XML-formatted data returned by the Google Geocode API.
- The name of the fourth parameter is *exclude*. The value of this parameter is “*minutely, hourly, alerts, flags*”.

The response from a query of the Forecast.io web service is a JSON-formatted object. An example of a returned JSON-formatted object from the Forecast.io API is available in Figure 6. You need to parse the returned JSON-formatted object and extract some fields. After extracting

the data, the PHP script should display the data in a tabular format below the search form. A sample output is shown in Figure 7.

```
{
  "latitude": 33.9147445,
  "longitude": -118.2827372,
  "timezone": "America/Los_Angeles",
  "currently": {
    "time": 1569114139,
    "summary": "Clear",
    "icon": "clear-day",
    "nearestStormDistance": 126,
    "nearestStormBearing": 122,
    "precipIntensity": 0,
    "precipProbability": 0,
    "temperature": 78.52,
    "apparentTemperature": 78.71,
    "dewPoint": 61.49,
    "humidity": 0.56,
    "pressure": 1009.08,
    "windSpeed": 6.76,
    "windGust": 11.65,
    "windBearing": 213,
    "cloudCover": 0,
    "uvIndex": 0,
  }
}
```

Figure 6: Part of Sample JSON-formatted object returned from Forecast.io API call



| Date | Status | Summary | TemperatureHigh | TemperatureLow | Wind Speed |
|------------|--------|-----------------------------------|-----------------|----------------|------------|
| 2019-09-21 | | Clear throughout the day. | 83.62 | 64.55 | 3.97 |
| 2019-09-22 | | Mostly cloudy throughout the day. | 86.39 | 65.22 | 3.59 |
| 2019-09-23 | | Partly cloudy throughout the day. | 81.21 | 64.4 | 4.72 |
| 2019-09-24 | | Partly cloudy throughout the day. | 87.19 | 65.18 | 5.02 |
| 2019-09-25 | | Clear throughout the day. | 82.9 | 62.94 | 6.55 |
| 2019-09-26 | | Partly cloudy throughout the day. | 77.72 | 64.37 | 7.41 |
| 2019-09-27 | | Overcast throughout the day. | 72.32 | 61.23 | 7.32 |
| 2019-09-28 | | Mostly cloudy throughout the day. | 71.17 | 56.78 | 6.34 |

Figure 7: Search Result

You need to map the data extracted from the result of the Forecast.io API call to your Card view of the current weather at that location and the table providing temperature predictions for the whole week:

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
  },
  "daily": {
    "data": [
      {
        "date": "2019-09-24",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      },
      {
        "date": "2019-09-25",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      },
      {
        "date": "2019-09-26",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      },
      {
        "date": "2019-09-27",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      },
      {
        "date": "2019-09-28",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      },
      {
        "date": "2019-09-29",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      },
      {
        "date": "2019-09-30",
        "open": 72,
        "close": 78,
        "low": 68,
        "high": 82,
        "precip": 0,
        "summary": "Mostly Cloudy",
        "uvIndex": 5,
        "visibility": 10,
        "windSpeed": 5,
        "windGust": 8,
        "windBearing": 195,
        "cloudCover": 0.74,
        "humidity": 0.7,
        "pressure": 1013.83,
        "dewPoint": 63.8,
        "apparentTemperature": 74.62,
        "temperature": 74.21,
        "precipIntensity": 0,
        "precipProbability": 0,
        "nearestStormDistance": 0,
        "icon": "partly-cloudy-day",
        "time": 1569263535
      }
    ]
  }
}
```

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 “ <i>timezone</i> ”. |
| Temperature | The value of the key “ <i>temperature</i> ” in the |

| | |
|------------|---|
| | <p><i>currently</i> 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</p> |
| Summary | The value of “ <i>summary</i> ” in the <i>currently</i> object. |
| Humidity | <p>The value of “<i>humidity</i>” in the <i>currently</i> object. It should also be provided with the corresponding icon - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-16-512.png</p> |
| Pressure | <p>The value of “<i>pressure</i>” in the <i>currently</i> object. It should also be provided with the corresponding icon - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-25-512.png</p> |
| Wind Speed | <p>The value of “<i>windSpeed</i>” in the <i>currently</i> 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</p> |
| Visibility | <p>The value of “<i>visibility</i>” in the <i>currently</i> 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</p> |
| CloudCover | <p>The value of “<i>cloudCover</i>” in the <i>currently</i> object. It should also be provided with the corresponding icon - https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-28-512.png</p> |
| Ozone | <p>The value of “<i>ozone</i>” in the <i>currently</i> 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</p> |

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

```
"daily": {
  "summary": "No precipitation throughout the week, with high temperatures falling to 71°F next Monday.",
  "icon": "clear-day",
  "data": [
    {
      "time": 1569222000,
      "summary": "Partly cloudy throughout the day.",
      "icon": "partly-cloudy-day",
      "sunriseTime": 1569246194,
      "sunsetTime": 1569289817,
      "moonPhase": 0.82,
      "precipIntensity": 0.0006,
      "precipIntensityMax": 0.0025,
      "precipIntensityMaxTime": 1569279600,
      "precipProbability": 0.05,
      "precipType": "rain",
      "temperatureHigh": 78.56,
      "temperatureHighTime": 1569272400,
      "temperatureLow": 63.07,
      "temperatureLowTime": 1569326400,
      "apparentTemperatureHigh": 78.77,
      "apparentTemperatureHighTime": 1569272400,
      "apparentTemperatureLow": 63.37,
      "apparentTemperatureLowTime": 1569326400,
      "dewPoint": 60.9,
      "humidity": 0.73,
      "pressure": 1013.02,
      "windSpeed": 4.35
    }
  ]
}
```

Figure 9: Part of Sample JSON-formatted object returned from Forecast.io API call for the table layout

| Table Column | Data from result of Forecast.io API call |
|--------------|---|
| Date | The value of “ <i>time</i> ” in the <i>data</i> object of the <i>daily</i> object. |
| Status | <p>The value of “<i>icon</i>” in the <i>data</i> object of the <i>daily</i> object. It is possibly one of the following values:</p> <ul style="list-style-type: none">clear-day clear-night: The icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-12-512.pngrain: The icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-04-512.pngsnow: The icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-19-512.pngsleet: The icon for this is |

| | |
|-----------------|--|
| | <p>https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-07-512.png</p> <ul style="list-style-type: none"> • wind: The icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-27-512.png • fog: The icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-28-512.png • cloudy: This icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-01-512.png • partly-cloudy-day partly-cloudy-night: The icon for this is https://cdn2.iconfinder.com/data/icons/weather-74/24/weather-02-512.png |
| Summary | The value of “ <i>summary</i> ” in the <i>data</i> object of the <i>daily</i> object. |
| TemperatureHigh | The value of “ <i>temperatureHigh</i> ” in the <i>data</i> object of the <i>daily</i> object. |
| TemperatureLow | The value of “ <i>temperatureLow</i> ” in the <i>data</i> object of the <i>daily</i> object. |
| Wind Speed | The value of “ <i>windSpeed</i> ” in the <i>data</i> object of the <i>daily</i> object. |



Figure 10: On Hover

On hovering the icons in the card, it should display the type of parameter. Please refer the video for this behavior.

In summary, the search mechanism to be implemented behaves as follows:

- Based on the input data in the search form, construct a web service URL to retrieve the XML-formatted output from the Google GeoCode API.
- Parse the returned XML and extract the latitude and longitude values.
- Call the Forecast.io API (latitude and longitude are parameters in the web service URL) and retrieve the JSON-formatted output.
- Parse the returned JSON-formatted output and extract the weather information.
- Display the weather information in tabular format.

2.3. Displaying Detailed Summary of the Weather

On clicking the summary present in the above table, the page should display a detailed description of the weather. The Daily Weather Detail provides details provided by the forecast.io API. The API call to query the weather information is given as:

[https://api.darksky.net/forecast/\[YOUR_API_KEY\]/\[LATITUDE\],\[LONGITUDE\],\[TIME\]?exclude=minutely](https://api.darksky.net/forecast/[YOUR_API_KEY]/[LATITUDE],[LONGITUDE],[TIME]?exclude=minutely)

Note: The time parameter is from the *data* object selected from the table which is part of the daily object from the 1st API call.

Daily Weather Detail

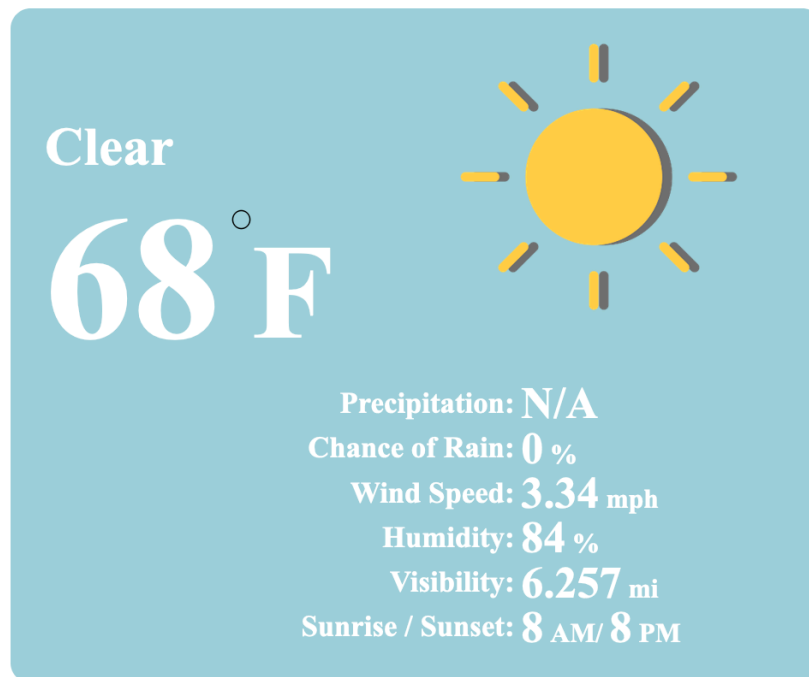


Figure 11: Daily Weather detail

The values displayed in the Daily Weather Detail as shown in the following in table:

```

latitude: 33.9147445
longitude: -118.2827372
timezone: "America/Los_Angeles"
▼ currently:
  time: 1569263951
  summary: "Mostly Cloudy"
  icon: "partly-cloudy-day"
  nearestStormDistance: 0
  precipIntensity: 0
  precipProbability: 0
  temperature: 74.47
  apparentTemperature: 74.87
  dewPoint: 63.77
  humidity: 0.69
  pressure: 1013.76
  windSpeed: 5.82
  windGust: 8.51
  windBearing: 196
  cloudCover: 0.73
  uvIndex: 5
  visibility: 10
  ozone: 293.1
▼ daily:
  ► summary: "No precipitation through...ng to 71°F next Monday."
  icon: "clear-day"
  ▼ data:
    ▼ 0:
      time: 1569222000
      summary: "Partly cloudy throughout the day."
      icon: "partly-cloudy-day"
      sunriseTime: 1569246194
      sunsetTime: 1569289817
      moonPhase: 0.82
      precipIntensity: 0.0006



```

Figure 12: Part of Sample JSON-formatted object returned from Forecast.io API call for the card layout

| Table Column | Data from result of Forecast.io API call |
|--------------|---|
| Summary | The value of “ <i>summary</i> ” in the <i>currently</i> object. |
| Temperature | The value of the key “ <i>temperature</i> ” in the <i>currently</i> 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 |
| Icon | The value of “ <i>icon</i> ” in the <i>currently</i> object. It is |

| | |
|----------------|--|
| | <p>possibly one of the following values:</p> <ul style="list-style-type: none"> • 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 | <p>The value of “<i>precipIntensity</i>” in the <i>currently</i> object. The value is based on the following conditions:</p> <ul style="list-style-type: none"> • value ≤ 0.001, display – “None” • value ≤ 0.015, display – “Very Light” • value ≤ 0.05, display – “Light” • value ≤ 0.1, display – “Moderate” • value > 0.1, display – “heavy” |
| Chance of Rain | <p>The value of “<i>precipProbability</i>” in the <i>currently</i> object. The value is to be multiplied by 100 to give the percentage.</p> |
| Wind Speed | <p>The value of “<i>windSpeed</i>” in the <i>currently</i></p> |

| | |
|------------------|--|
| | object. The value should be provided with the unit “ <i>mph</i> ”. |
| Humidity | The value of “ <i>humidity</i> ” in the <i>currently</i> object. The value is to be multiplied by 100 to give the percentage. |
| Visibility | The value of “ <i>visibility</i> ” in the <i>currently</i> object. The value should be provided with the unit “ <i>mi</i> ”. |
| Sunrise / Sunset | The value of “ <i>sunriseTime</i> ” of the 1st element in the <i>data</i> object present in the <i>daily</i> object. The value of “ <i>sunsetTime</i> ” of the 1st element in the <i>data</i> object present in the <i>daily</i> object. Note: The values should just provide the hour and the period (AM/PM) and in 12 hr format. |

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

Day's Hourly Weather



Day's Hourly Weather

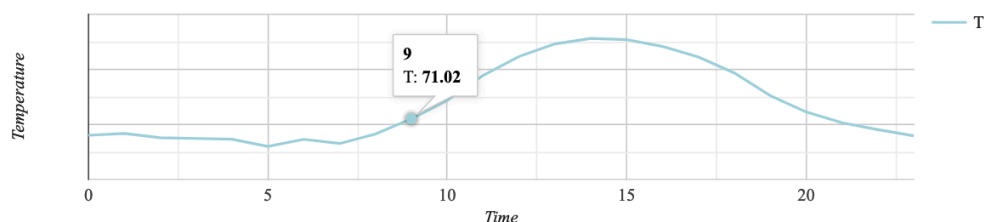


Figure 13: Toggling the buttons

The graph is a chart of the hourly temperature for that day. The hourly temperature can be found from the JSON structure –

Note: The value of the “temperature” is present in the data array of the hourly object.

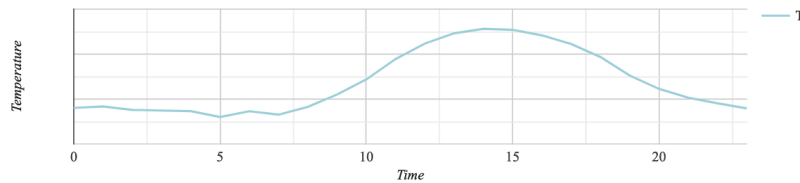


Figure 14: Hourly temperature for that day

The Chart is made with the help of Google Charts. Please find out how to use Google Charts here:

<https://developers.google.com/chart/interactive/docs/gallery/linechart>

This link will provide the necessary details to display a chart of the hourly temperatures for that day.

2.4. Saving Previous Inputs

In addition to displaying the results, your page should maintain the provided values to display the current result. For example, if one searches for “**Street:** *Figueroa St*, **City:** Los Angeles, **State:** California”, one should see what was provided in the search form and the corresponding results. It follows that you need to keep the whole search box/input fields and buttons even while displaying results/errors.

3. Hints

3.1. How to get Forecast.io API Key

Go to <https://darksky.net/dev>. Click on “Try For Free”. You should fill out the form as shown in Figure 15. The API Key (Dark Sky API key) will be displayed. Copy this key as you will use it in the Forecast.io web service API call.

Figure 15: Forecast.io Registration Form

3.2 How to get a 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=SERVER_SIDE&reusekey=true).
- Create or select a project.
- Click Continue to Enable the API.
- Go to Credentials to get a Server key (and set the API Credentials).

3.3. Get Geolocation using IP-API.com

You need to use *ip-api.com* for searching the geolocation based on IP addresses. An example call is as follows:

<http://ip-api.com/json>

The response is a JSON object shown in **Figure 16**.

```
as: "AS20001 Time Warner Cable Internet LLC"
city: "Los Angeles"
country: "United States"
countryCode: "US"
isp: "Time Warner Cable"
lat: 34.0266
lon: -118.2831
org: "Time Warner Cable"
query: "104.32.172.65"
region: "CA"
regionName: "California"
status: "success"
timezone: "America/Los_Angeles"
zip: "90007"
```

Figure 16: Response from *ip-api.com* API

3.4 Parsing JSON-formatted data in PHP

In PHP 5 and 7, you can parse JSON-formatted data using the “*json_decode*” function. For more information, please go to <http://php.net/manual/en/function.json-decode.php>.

You can encode data into JSON-formatted objects using the “*json_encode*” function. For more information, please go to <http://php.net/manual/en/function.json-encode.php>.

3.5 Read and save contents in PHP

To read the contents of a JSON-formatted object, you can use the “*file_get_contents*” function.
To save contents on the server side, you can use “*file_put_contents*” function.

3.6 Deploy PHP file to the cloud (GAE/AWS/Azure)

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

GAE - <http://example.appspot.com/forecast.php>
AWS - <http://example.elasticbeanstalk.com/forecast.php>
Azure - <http://example.azurewebsites.net/forecast.php>

example in the above URLs will be replaced by your choice of subdomain.

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, **forecast.php**). This PHP file must be hosted on GAE, AWS or Azure cloud service. Graders will verify that this link is indeed pointing to one of the cloud services.

Also, submit your source code file (**it must be a single .php file, e.g. forecast.php**) to the GitHub Classroom repository so that it can be graded and compared to all other students' source code via the MOSS code comparison tool.

IMPORTANT: repeating the **REQUIREMENT** that the entire implementation needs to be submitted as a **SINGLE PHP file**, that will include all the required PHP, HTML, CSS and JavaScript. A penalty of 2 points will be assessed if the code is split over more than one PHP file.

****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 should not use JQuery for Homework 6.
- You **should not call the forecast.io APIs directly from JavaScript**, bypassing the Apache/HTTP proxy. Implementing any one of them in JavaScript instead of PHP will result in a **4-point penalty**.

The link to the video is: <https://youtu.be/irN8ZV-GuR8>