Developer’s Guide

Overview

This project allows users to search for a location by city name or zip code. The app suggests city names based on the text that’s been input by using GeoApify’s API. The zip code and city names are interpreted into coordinates through OpenCage Geocode’s API. The search returns 10-day and 24-hour forecasts using OpenMetoeo’s API. The forecasts show the temperature, the likelihood of rain, and an image of the weather type (i.e., sunny).

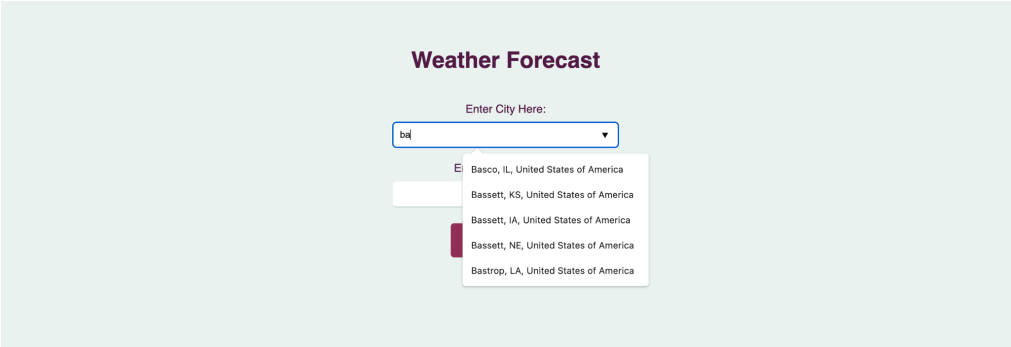
Install/deployment/admin issues

To run this program, the developer should follow these steps:

1. Create an API key for the geocoding by going to OpenCage’s website and signing up.
2. Create an API key for autocomplete by going to Geoapify website and signing up.
3. Once you have both API keys, place the OpenCage key in ***api_geocode.txt,*** then place the Geoapify in ***api_auto_complete.txt.***

(End) User interaction

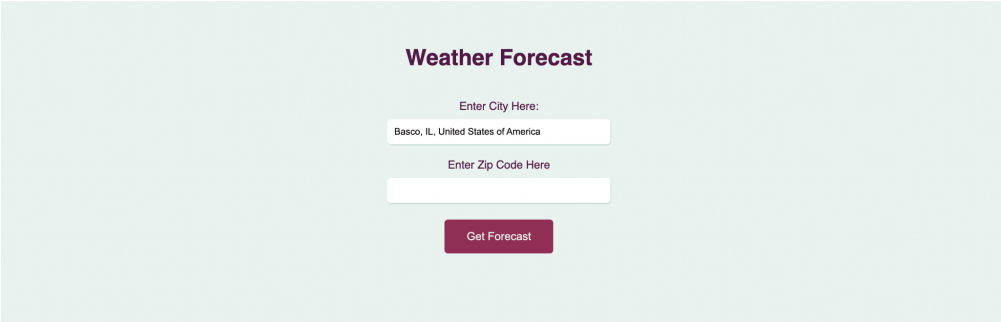
Step 1: User inputs their desired location by city name or zip code



Code

- Python methods used:
    - def home() - This Flask route takes the user to the home page using a html template.
    - def autocomplete() - This flask route handles autocomplete by making API calls to Geoapify using the user's requests and is handled by Flask; this is only for Cities.
- HTML/CSS/JS used:
    - base.html - This creates the user's homepage view by creating the UI elements using HTML and CSS.
    - Javascript - The JS here handles the data given to us by our autocomplete() Python method and serves it to the user.

Step 2: The user clicks the “Get Forecast” button

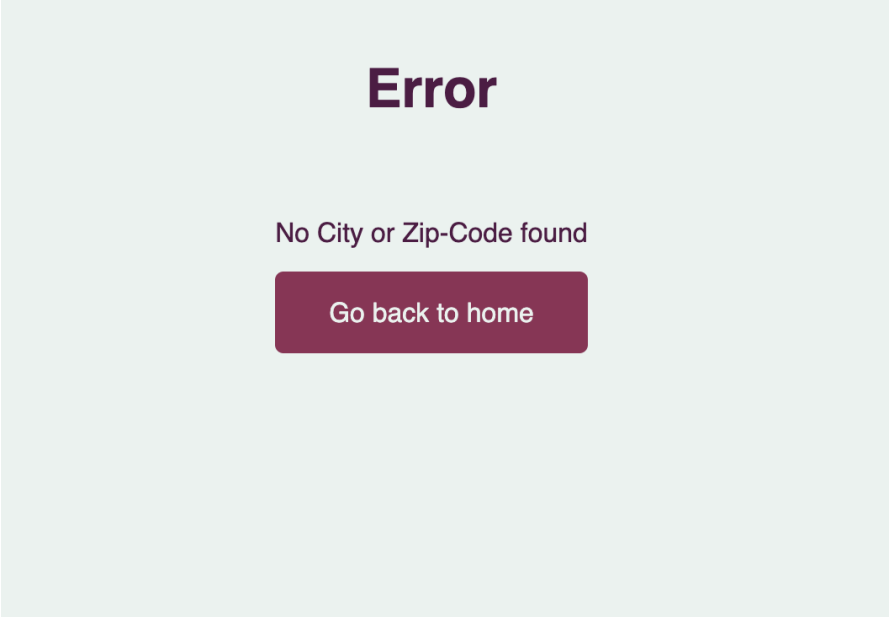


Code:

- Python method used:
    - def forecast() - This method uses the user's request city or zip code and executes the logic to retrieve the weather data. This data is then sent to forecast.html to display.
    - def get_forecast_daily(lat, long, forecast_d) - This creates the API call by adding headers and params needed for the daily forecasts to then send to the open-meteo (Free Keyless API).
    - def get_cord(city) - This uses our geocoder API to get lat and lng data from city names.
    - def error_handle() - This flask route handles requests from users for which no cities or zip code can be found

- HTML/CSS
    - forecast.html - This creates the user's daily forecast view by creating the UI elements using HTML and CSS.
    - error_handle.html - This creates the user's error-handling view by creating the UI elements using HTML and CSS.

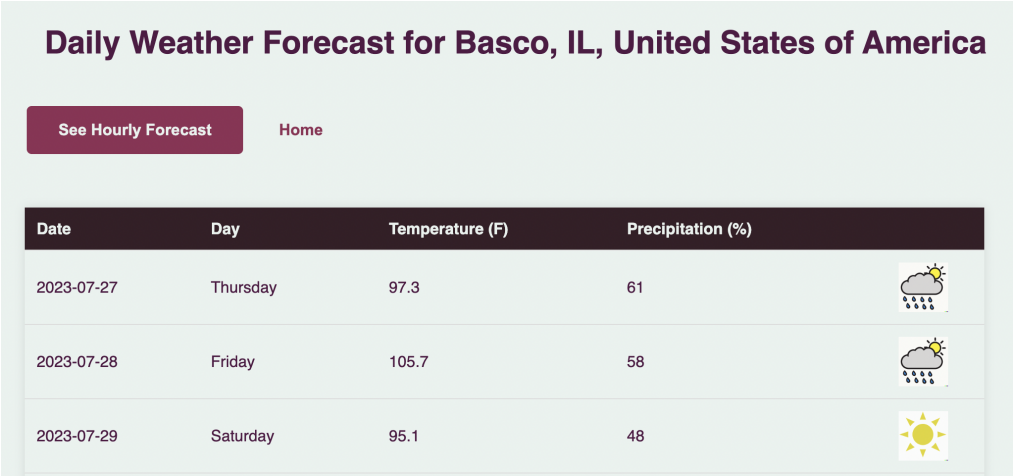
Step 3a: The user types a city or zip code incorrectly, and the error_handle.html UI is shown



Code:

- HTML/CSS
    - error_handle.html - This creates the user's error-handling view by creating the UI elements using HTML and CSS.

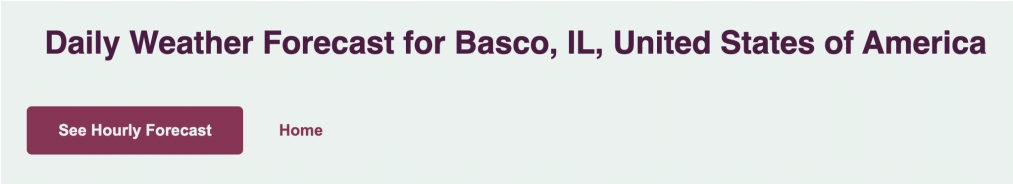
Step 3b: The user views the 10-day forecast to see temperature and precipitation likelihood per day**:**



Code:

- HTML/CSS
    - forecast.html - This loads the UI elements needed for the daily forecast. It loads the image depending on the logic in def forecast().

Step 5: The user clicks the “See Hourly Forecast” button

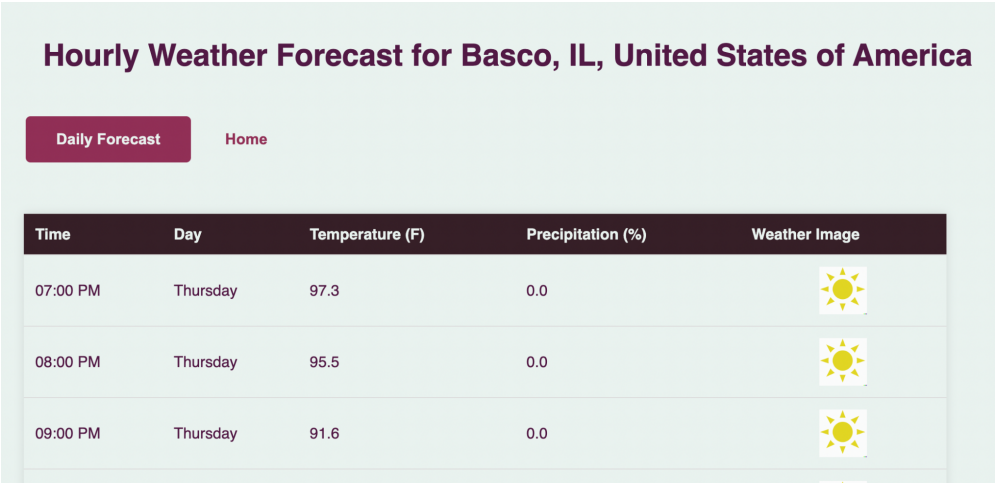


Code:

- Python method used:

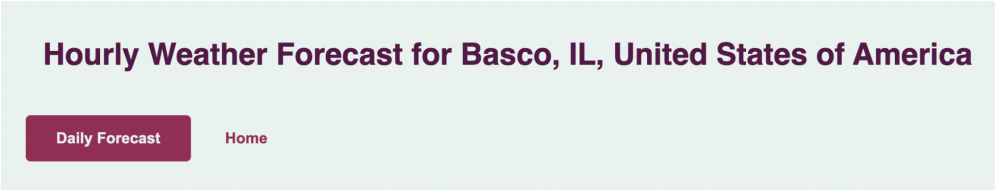
    - def hourly_forecast() - This method is used when the user clicks select hourly forecast. It contains the logic to retrieve the weather data. This data is then sent to hourly_forecast.html to display.
    - get_forecast_hourly(lat, long, forecast_d) - This creates the API call by adding headers and params needed for the hourly forecasts to then send to the open-meteo (Free Keyless API).
    - get_cord(city) - This uses our geocoder API to get lat and lng data from city names.
- HTML/CSS
    - hourly_forecast.html - This creates the user's hourly forecast view by creating the UI elements using HTML and CSS.

Step 6: The user views the 24-hour forecast to see temperature and precipitation likelihood per hour



- HTML/CSS
    - hourly_forecast.html - This loads the UI elements needed for the hourly forecast. It loads the image depending on the logic in def get_forecast_hourly.

Step 7: The user clicks the “Home” button to start a new search

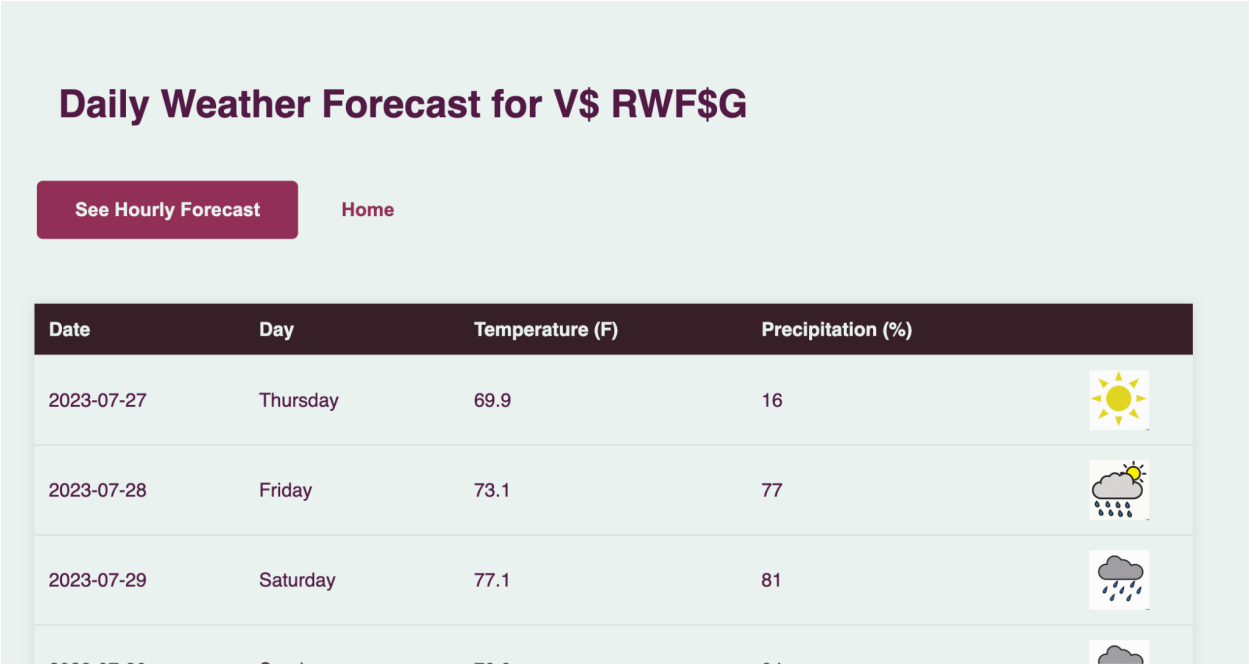


Code:

- Python methods used:
  - def home() - This Flask route takes the user to the home page using a html template.
- HTML/CSS:
  - base.html - This creates the user's homepage view by creating the UI elements using HTML and CSS.

Known Issues

The auto-complete has a slow response that lets users access the daily forecast page with invalid entries; this is due to partial user requests getting picked up early, leading to non-existent city errors like in the screenshot below:



Future work

In the future, there are a few features I would like to add:

- **Additional Weather Types:** Right now, the code only displays images for sunny days, rainy days, and days in between. It should include images for additional weather types such as snow, hail, sun showers, etc.
- **Zip Code Autocomplete:** Have suggestions for zip codes, and they're being typed
- **Daily High and Low:** On the 10-day forecast, including the high and low for each day.
- **Consolidating API:** Currently, the developer has to use two APIs to use the app; we could use Geoapify for geocoding and autocomplete.

TLDR

Please download the API keys and place them in their corresponding folders. For geocoding, use OpenCage, and for autocomplete, use Geoapify. Once this is done, Pip install the required packages and then runs the app.py file. This should give you a link to the site in the console.