README!

Weather Application

Author: Abdul-Awwal Adesalu

Description

The goal of this project is to create a weather app utilizing the `weather.gov` API, this app should take user input for a location and then retrieve the weather information using the API. The application should store the retrieved data in a Redis cache(this can be hosted on `docker`) for faster retrieval should the user request the data again for the same location. Else, the app should make a network call to get the new and updated information and then store the data in the Redis cache.

Functional Requirements:

- The user should be able to input a location.
- The user should be able to see weather info for the location requested by the app sending a request to the `weather.gov` API
- The app should store the retrieved weather data in a redis cache for the specific location
- The app should first check the redis cache for existing data before making a new network call in the case of a new request.
- The retrieved information should be displayed to the user in a readable format.

Non-functional Requirements:

- User input should be sanitized to avoid injection attacks(Use RegEx) and malicious requests.
- User interface should be user friendly and easy to navigate.
- Error handling in the case of network issues, API failures and cache misses

Technology used:

- Redis DB
- React
- JavaScript
- Golang
- Docker
- Gomega
- Postman
- Gingko

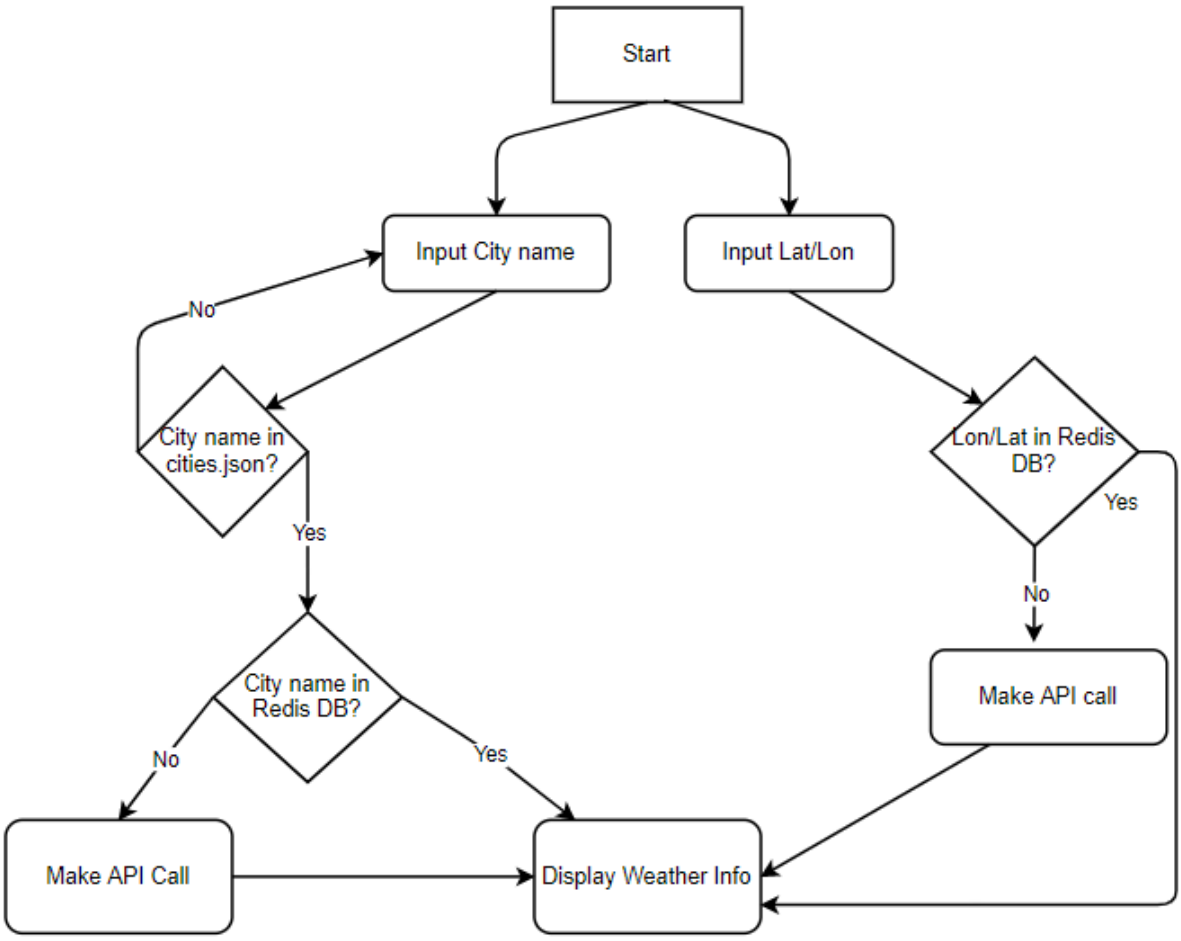
How to run:

- Run `npm init` in the weather-app folder, then `npm start` , if you do not have NPM installed, you can download it from `https://nodejs.org/en/download/package-manager`
- Navigate to the go-backend folder and run `go run main.go` , if you do not have go installed, you can download it from `https://go.dev/doc/install`
- Create a `docker-compose.yml` file for Redis, it should look like this:

```
version: '3'
services:
  redis:
    image: redis:latest
    container_name: redis
    ports:
      - "6379:6379"
```

- Run `docker-compose up -d` to start redis DB.

Sample Application flow:



Code Breakdown:

- `App.js`: Manages the state for weather data, also renders the `WeatherForm.js` component to allow users to input a location and the `WeatherDisplay.js` component to show weather information once it is fetched from the backend.
- `WeatherForm.js`: This component handles user input to get a specific location and fetches the corresponding weather data from the backend, it then updates `App.js` by calling the function `setWeather`.
- `WeatherDisplay.js`: This component presents the final data to the user, it receives the weather data from `App.js` and displays this information to user in a user friendly format.
- `main.go`: HTTP server which receives requests from `WeatherForm.js` and makes an API call to the `weather.gov` API, if the users uses a predefined city name, it parses the longitude and latitude relating to that city name from `cities.json` and uses that as the endpoints in the API call, it then parses the retrieved information into a json format and sends it to the Redis DB to be cached. `main.go` also queries the Redis DB to find matching data to load faster for performance, if it is not found, then it will make an API call.
- `cities.json`: Predefined list of cities for `main.go` to grab longitudes and latitudes from.
- `main_test.go`: Go file which uses Gomega and ginkgo to test units in `main.go`. Specifically tests the `getCoordinates` function to make sure the longitude and latitudes are correctly assigned.

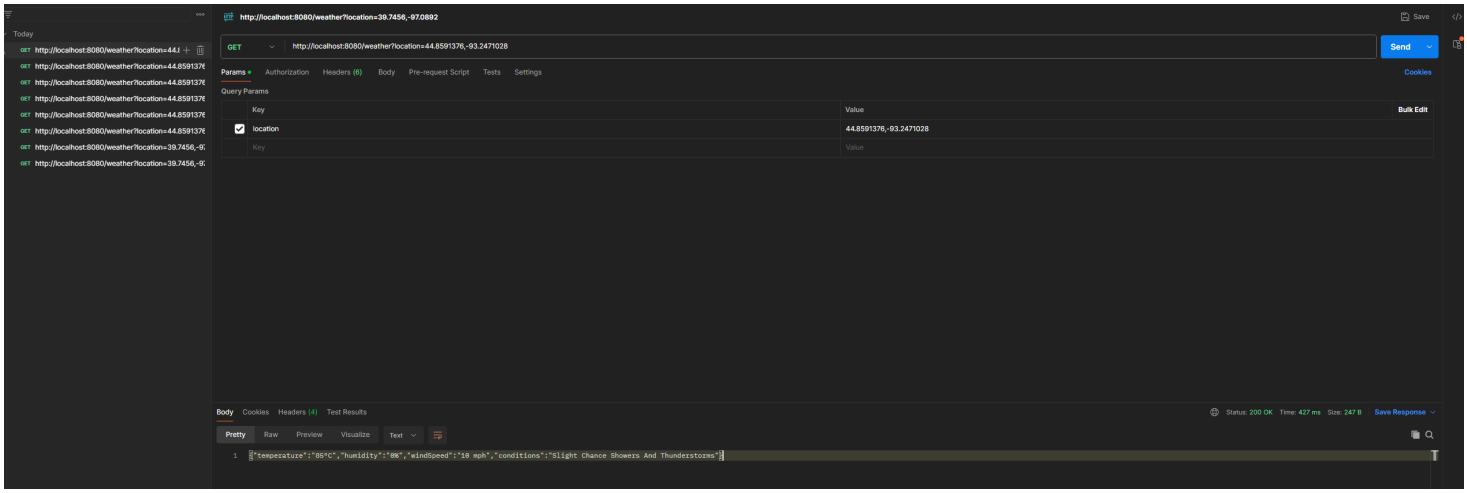
Testing:

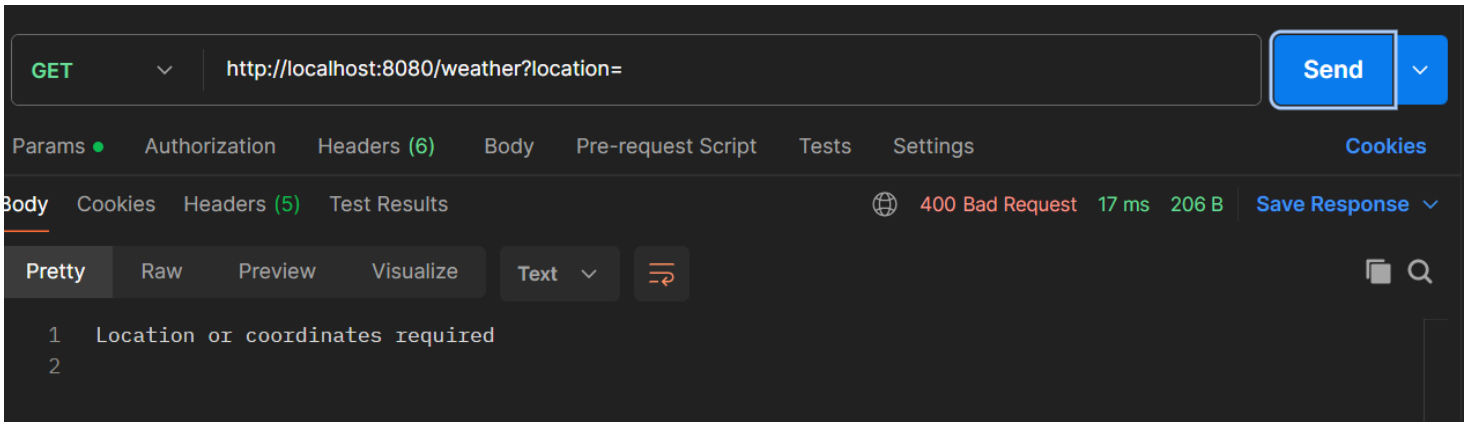
S.NO	Action	Input	Expected Output	Actual Output	Test Result	Test comments
1	Initiate Fetch Request	Click `Get weather` button	Fetch request initated	Fetch request initated	Pass	Testing to see if Front-End could communicate with back-end
2	Fetch request	Random	Dummy data stored on go server	CORS Error (Could not fetch)	Fail	Installed CORS package with go after this
3	Fetch Request	Random	Dummy data stored on go server	Dummy data stored on go server	Pass	Tested with dummy data to see if it could be retrieved and displayed on the front end.
4	Wrong coordinates	City Name, Longitude, Latitude	Provided coordinates of a different city to the `getCoordinate` function, test should executed	Test executed successfully	Pass	Unit testing

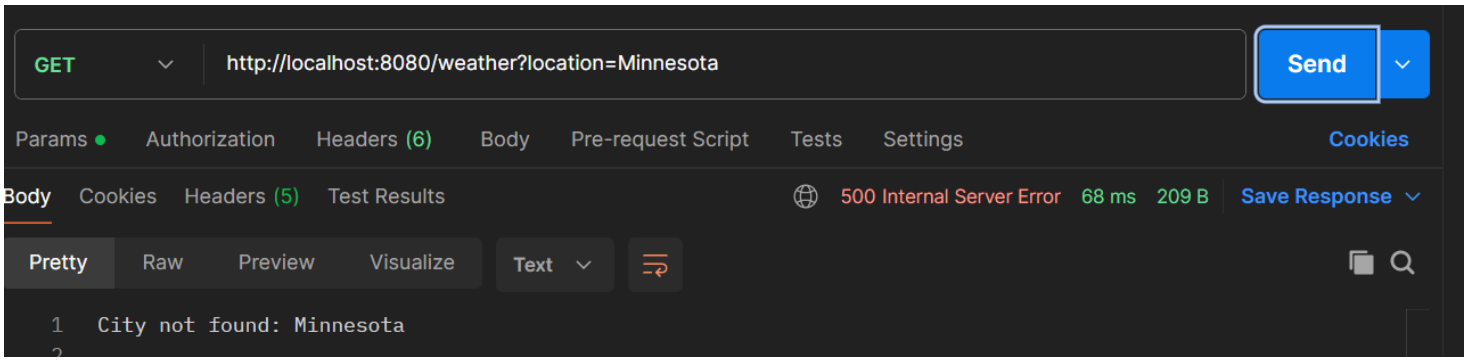
S.NO	Action	Input	Expected Output	Actual Output	Test Result	Test comments
			successfully since coordinates were not equal			
5	Correct coordinates	City Name, Longitude, Latitude	Provided coordinates of a Correct city to the `getCoordinate` function, test should executed successfully since coordinates were equal	Test executed successfully	Pass	Unit testing
6	API Call	Longitude and Latitude	Retrieve and display weather data from `weather.gov` API	No weather could be fetched for this location	Fail	Testing API call, there was a problem with the parameters being sent as location
7	API Call	Longitude and Latitude	Retrieve and display weather data from `weather.gov` API	Retrieved and displayed weather data from `weather.gov` API(See image below)	Pass	Tested API call using postman with direct longitude and latitude, Web App should take user friendly input as longitude and latitudes are hard to remember.
8	Fetch request	City Name	Backend should cross reference city name with a predefined list of cities and use the latitude and longitude of that city to make the API call and return weather information to client	Fetch request successful, weather displayed successfully(See image below)	Pass	Minimum Viable Product achieved. Processing time: 291.7418ms
9	Fetch Request	City Name	Data should be pulled from the	Fetch request successful,	Pass	Processing time:2.8915ms,

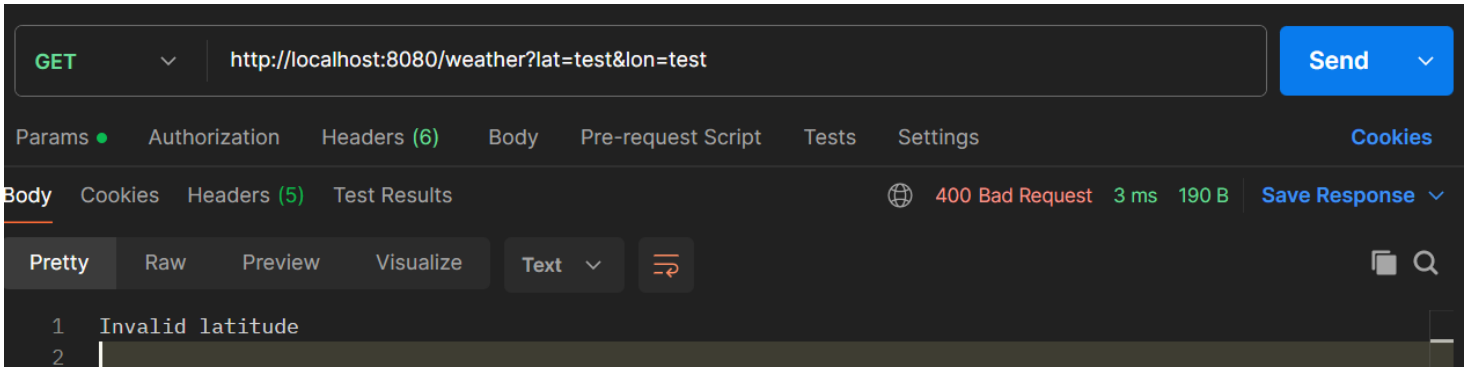
S.NO	Action	Input	Expected Output	Actual Output	Test Result	Test comments
			Redis cache resulting in a faster loading time	weather displayed successfully		a 99.01 percent decrease in processing time. Checked Redis DB to see if data was indeed being stored there using `KEYS *`, I then used the `flushall` command to clear the DB and reran the test to see if fetching will take longer, which it did
10	Fetch request	Latitude and longitude	Backend should use longitude and latitude to make API call and return weather information to client	Fetch request successful, weather displayed successfully(See image below	Pass	Minimum Viable Product achieved
11	Error handling	Invalid city	Backend should log "City not found" in the case of a city that is not in the `cities.json` file and front-end should showcase "Failed to fetch weather data"	"City not found" was logged, "Failed to fetch weather data" was shown.	Pass	Testing error handling
12	Error handling	Random input	Frontend should return "Failed to fetch weather data" in the case that the server did not respond	"Failed to fetch weather data" returned	Pass	Testing error handling

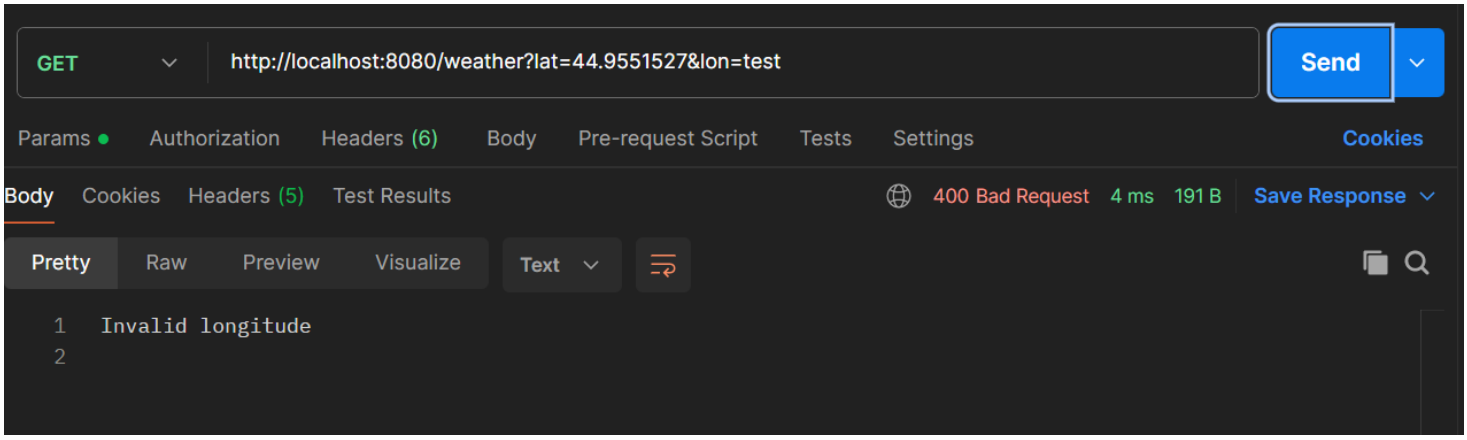
Postman testing API calls from the server:



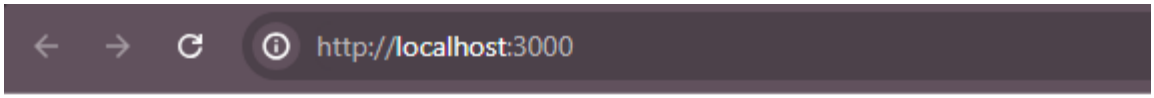








First iteration:



Weather App

City Name Coordinates

Charlotte

Get Weather

Weather Information

Temperature: 98°C

Humidity: 0%

Wind Speed: 7 mph

Conditions: Chance Showers And Thunderstorms

Using Longitude and Latitude

Weather App

City Name Coordinates

37.6739854 -105.2139531

Get Weather

Weather Information

Temperature: 82°C

Humidity: 0%

Wind Speed: 5 to 10 mph

Conditions: Chance Showers And Thunderstorms

Test times:

```
2024/07/16 06:53:14 Total request processing time: 291.7418ms
2024/07/16 06:53:18 Total request processing time: 2.8915ms
2024/07/16 06:58:00 Total request processing time: 100.8225ms
2024/07/16 06:58:04 Total request processing time: 1.4697ms
2024/07/16 06:58:20 Total request processing time: 790µs
2024/07/16 06:58:22 Total request processing time: 2.3267ms
2024/07/16 06:58:23 Total request processing time: 1.1662ms
2024/07/16 06:58:25 Total request processing time: 1.5135ms
2024/07/16 06:58:37 Total request processing time: 48.5704ms
2024/07/16 06:58:38 Total request processing time: 2.2731ms
2024/07/16 06:58:59 Total request processing time: 52.7793ms
2024/07/16 06:59:02 Total request processing time: 1.1305ms
2024/07/16 06:59:24 Total request processing time: 116.8416ms
2024/07/16 06:59:26 Total request processing time: 1.1012ms
```

Notes:

- Encountered a cross origin resource sharing error when testing a fetch request to the go server from the react app, fixed it by installing the CORS package with go.
- Cross origin resource sharing allows a server to indicate origins other than its own from which a browser should permit loading resources.
- When running the Redis DB on docker, access the CLI by using `docker exec -it redis redis-cli`
- Go uses `net/http` package when returning a `http.Response` object, always make sure to close the response before exiting the function.
- In react, the default form submission behavior submits the form to the current URL if no action attribute is specified, in other words, it reloads the page which is less smooth.
- The `weather.gov` API first returns an initial response containing metadata about the location, you can then use a URL contained in the response to make a second API call retrieving detailed weather forecast information.
- The first call to the API takes the most time, every other call greater than 3ms is to the API (a cache miss) whilst less than is from the cache(a cache hit), there is a notable difference between loading from an API call and loading from cache, fastest time being 790 microseconds.
- In the postman test image where it says invalid latitude, the longitude is also invalid, but because server parses the latitude and returns an error if that is not valid, it does not even bother to parse the longitude.
- Latitude and longitude are both parsed as floats
- Initially displayed the error message in the client response, for better security practices, logging is now done on the server side. On the client side, all they will get returned if there is an error is "failed to fetch weather data", except in the case of an invalid city, longitude and latitude.
- `ioutil` in go is deprecated, use `io` instead, does the same thing.
- Needed a way to test unit functions in `main.go`, came across Gomega and gingko.
- Typescript allows you to catch errors during development, reducing bugs and code maintainability
- React allows variables to change values at runtime

- . The server multiplexer is an `http.Handler` that is able to look at a request path and call a given handler function associated with that path(in this case, "/weather", getWeather).
- Slice is a variable-length sequence that stores elements of a similar type

Future works:

- Weather app currently only shows weather for locations within the US, eventually will like to make it show weathers for all locations in the world.
- Implement a geocode API that can translate user input for cities into longitude and latitude, currently reads from a predefined list of 20 cities in `cities.json`. This can be done concurrently by using an async function, the first function which converts the city name to coordinate will return a promise to the `handleSubmit` function, which will then in turn make the function await a value from the converter.
- No current way for the user to know which cities are accepted and which are not, would list current cities in `cities.json` in the UI.
- Implement a loading animation for UI
- Implement React bootstrap into forms for styling
- If geocode cannot be implemented, implement a save location button if the user decides to use longitude and latitude, this way, the list of locations in `cities.json` will be updated to save locations if the user chooses.
- The `cities.json` is acceptable because the file size is small, but expanding `cities.json` can lead to memory problems as it will become memory intensive, use an API or create a database and query as needed
- Implement cache clearing periodically so new data is pulled to reflect the current weather situation (maybe clear cache every hour).
- Sanitize input using regex, here is the code snippet:

```
const sanitizeInput = (input) => { const sanitizedInput = input.replace(/[^a-zA-Z0-9\s]/g, ''); return sanitizedInput; };
```

Security as it relates to OWASP Top 10:

- Input Validation and Sanitization
- Secure Communication
- Error Handling
- Dependency Management
- Authentication and Authorization
- Logging and Monitoring

References:

- `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`

- `https://go.dev/doc/`
- `https://www.weather.gov/documentation/services-web-api
- `https://www.youtube.com/watch?v=oeAocyrzfZY&t=439s`
- `https://onsi.github.io/gomega/`
- `https://onsi.github.io/ginkgo/`