

IN-CLASS ASSIGNMENT 4

**NAMES**

Shashank Bachu -500224639

Avinash Thokala - 500223817

Manikanth Reddy Nallabolu -500224276

Smithan Reddy Tirumalareddy-500223386

Venkata Nishith Kodi- 500223382

Alwin Saji - 500223932

**Course Code: CSDD1005**

**GitHub link:** <https://github.com/AvinashThokala/Assignment4>

**Docker Image:** - manikanthreddy6669/torontoweather:v0.2

**CODE**

main**.go**

package main

import (

    "encoding/json"

    "fmt"

    "io"

    "log"

    "net/http"

)

const weatherAPIURL = "https://api.openweathermap.org/data/2.5/weather?q=Toronto&appid=f59d1d1e04e09cfba52671eb99b77335"

type WeatherInfo struct {

    Weather []struct {

        Description string `json:"description"`

    } `json:"weather"`

    Main struct {

        Temp      float64 `json:"temp"`

        FeelsLike float64 `json:"feels\_like"`

        Humidity  int     `json:"humidity"`

        WindSpeed float64 `json:"wind\_speed"`

    } `json:"main"`

}

func kelvinToCelsius(kelvin float64) float64 {

    return kelvin - 273.15

}

func getTorontoWeather() (\*WeatherInfo, error) {

    resp, err := http.Get(weatherAPIURL)

    if err != nil {

        return nil, err

    }

    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)

    if err != nil {

        return nil, err

    }

    var weatherInfo WeatherInfo

    err = json.Unmarshal(body, &weatherInfo)

    if err != nil {

        return nil, err

    }

    // Convert temperatures to Celsius

    weatherInfo.Main.Temp = kelvinToCelsius(weatherInfo.Main.Temp)

    weatherInfo.Main.FeelsLike = kelvinToCelsius(weatherInfo.Main.FeelsLike)

    return &weatherInfo, nil

}

func torontoWeatherHandler(w http.ResponseWriter, r \*http.Request) {

    weatherInfo, err := getTorontoWeather()

    if err != nil {

        http.Error(w, "Error fetching weather data", http.StatusInternalServerError)

        return

    }

    temperature := weatherInfo.Main.Temp

    feelsLike := weatherInfo.Main.FeelsLike

    humidity := weatherInfo.Main.Humidity

    windSpeed := weatherInfo.Main.WindSpeed

    description := weatherInfo.Weather[0].Description

    fmt.Fprintf(w, "Weather in Toronto:\n")

    fmt.Fprintf(w, "Temperature: %.2f°C\n", temperature)

    fmt.Fprintf(w, "Feels like: %.2f°C\n", feelsLike)

    fmt.Fprintf(w, "Humidity: %d%%\n", humidity)

    fmt.Fprintf(w, "Wind Speed: %.2f m/s\n", windSpeed)

    fmt.Fprintf(w, "Description: %s\n", description)

}

func main() {

    http.HandleFunc("/weather", torontoWeatherHandler)

    fmt.Println("Server is listening on port 8088")

    log.Fatal(http.ListenAndServe(":8088", nil))

}

**Main\_test.go**

package main

import (

    "encoding/json"

    "fmt"

    "net/http"

    "net/http/httptest"

    "testing"

)

func TestGetWeather(w \*testing.T) {

    //Create a req

    req, err := http.NewRequest("GET", "/weather", nil)

    if err != nil {

        w.Fatal(err)

    }

    rr := httptest.NewRecorder()

    handler := http.HandlerFunc(torontoWeatherHandler)

    handler.ServeHTTP(rr, req)

    // Check the status code is what we expect.

    if status := rr.Code; status != http.StatusOK {

        w.Errorf("Handler returned wrong status code: got %v want %v", status, http.StatusOK)

    }

    var response WeatherInfo

    err = json.Unmarshal(rr.Body.Bytes(), &response)

    fmt.Println("Status code is", rr.Code)

    \_ = response.Main.Temp

}

**Output:-**

**A screenshot of a computer

Description automatically generated**

**Testcase Output:-**

A screenshot of a computer

Description automatically generated

**Explanation:-**

* weatherAPIURL constant string represets the OpenWeatherMap API URL for Toronto, including an API key. This URL is used to fetch weather data.
* WeatherInfo is a struct representing the data structure of the JSON response from the OpenWeatherMap API. It contains fields for weather description (Weather) and various weather-related parameters like temperature, feels-like temperature, humidity, and wind speed (Main).
* getTorontoWeather() (\*WeatherInfo, error) Function sends an HTTP GET request to the OpenWeatherMap API using the predefined URL (weatherAPIURL). Reads the response body and unmarshals the JSON data into a WeatherInfo struct.Returns a pointer to the WeatherInfo struct and an error if any.
* torontoWeatherHandler(w http.ResponseWriter, r \*http.Request) Function is an HTTP request handler for the "/weather" endpoint calls getTorontoWeather() to obtain weather information. If there's an error fetching weather data, it returns an HTTP 500 internal server error.If successful, it extracts relevant information from the WeatherInfo struct and formats it.
* main() Function sets up the "/weather" endpoint to be handled by the torontoWeatherHandler function and starts an HTTP server on port 8088.Logs a message indicating that the server is listening on port 8088.If there is an error starting the server, it logs a fatal error.