UTM UNIVERSITI TEKNOLOGI MALAYSIA | Malaysia-Japan International Institute of Technology (MJIIT)

SMJE 4383
ADVANCED PROGRAMMING
ASSIGNMENT 1
PROJECT REPORT

Project Title: Weather Web Application

Group Member(s): 1. Chee Kel Syin (A17MJ0026)

2. Chelsey Joane Lamam (A17MJ0027)

3. Wong Pui Ching (A17MJ0237)

Lecturer: Dr Zool Hilmi Bin Ismail

Date of Submission: 28th November 2020

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Background

Weather forecasting has existed since the ancient times and used as a progressing scientific technology for predicting the atmospheric conditions for specific locations and time. This method consists of quantitative data collection regarding the current atmospheric state of a given place and utilizing meteorology to predict future atmosphere conditions.

Once calculated by hand based mainly upon changes in barometric pressure, current weather conditions, and sky condition or cloud cover, weather forecasting now relies on computer-based models that take many atmospheric factors into account. Regardless, human input is still need to determine the best possible forecast model as a reference for the forecast. This process involves a lot of teleconnections, pattern recognition skills, knowledge of model biases, and knowledge of model performance. Forecasting inaccuracy exists due to the inconsistent nature of the atmosphere, complex equations used to represent the atmosphere state and the tremendous computational power required for solving the equations, initial conditions measurement errors, and a lack of understanding of atmospheric processes. Thus, weather forecast has less accuracy when the difference between current time and the time for which the forecast is being made increases. This is where the ensemble and model consensus help narrow the error and determine the most possible outcome.

Weather forecasting are mainly used to give weather warnings and important forecasts to prevent life loss and property damage. Agriculture depends on the temperature and precipitation forecasts and thus to traders within commodity markets. Weather forecasts are also a huge demand by utility companies to determine how much power to supply to the consumers.

1.2 Project Framework and Interface

Our project implements various software frameworks and interfaces. The main highlights of our implementation to build our weather web application are Django, PyCharm Community, Python Virtual Environment, Ubuntu operating system, and Beautiful Soup package.

Django is an open – source and free web framework that is based on Python programming language. It follows the model – template – views (MTV) architectural pattern. This web framework is maintained by the Django Software Foundation (DSF) and its main goal is to simplify the building process of complicated, database – driven websites. Django puts emphasis on pluggability of components and the reusability as well as minimized coding with low coupling and rapid development. It upholds the principle of no repeating oneself when creating. The Python programming language is widely used for data models, settings and files. Additionally, Django features an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models. Instagram and Mozilla are famous sites that utilizes the Django framework.

This project also implements PyCharm Community and Python Virtual Environment. PyCharm is an IDE or Integrated Development Environment which is popular in computer programming especially for Python programming language. It is a product by JetBrains, a Czech company and provides code analysing, graphical debugging, integrated unit tester, has integration with VCSes or Version Control Systems, and supports data science with Anaconda and web development with Django. PyCharm is compatible with various operating systems such as Windows, macOS and Linux. The most common editions are the Community Edition and the Professional Edition which consist more extra features. On the other hand, Python is a high – level and interpreted programming language for general purposes. This language emphasises the readability of the code and has notable use of significant whitespace. Python is designed for programmers to write understandable and logical coding for projects of any scale and this is done by the specific object – oriented approach and language construct of Python itself. Due to its nature of being garbage – collected and dynamically typed, Python is able to support multiple programming paradigms such as object – oriented, structured and functional programming. It also comes with a comprehensive standard library and is compatible with many operating systems such as Windows, macOS, Linux and so on.

The Ubuntu operating system is an open – source freeware that is distributed by Linux based on Debian. Currently, it is officially released in three editions mainly Desktop, Server and Core for Internet of things devices and robots. These editions can be run using a virtual machine or on the computer itself and is a popular operating system for cloud computing with support for OpenStack. The default desktop is GNOME.

Furthermore, the Beautiful Soup package is a Python package that is designed for parsing HTML and XML documents especially documents that have malformed mark – up such as non – closed tags. It produces a parse tree for the parsed pages that can be use to perform data extraction from HTML. This is widely used in web scraping. Beautiful Soup package is available for Python 2.7 and Python 3.

1.3 Working Principle

The working principle of our project starts by running the weather web application that is built using Python programming Language and Django framework, in the Ubuntu operating server. Simple user inputs of locations by users will lead to web scraping phase where the weather data is obtained from another web source and transfer to be displayed on our weather web application. This process is a looping process whereby the system loop ends when the users decides to exit the weather web application.

1.4 Existing System

Currently, many web applications exist for weather related functions. The most popular weather web applications available are Forecast and OpenWeather. Forecast works by calibrating your location and presenting a simple view of weather data info based on your location. The attractive feature is how the weather data is conveyed casually, like "Partly sunny for the hour" or "Sprinkling off-and-on until next Sunday." When a user interacts with the screen, more animated weather maps appear and shows the past and future weather forecasts of a certain location. Users can select between a local map, a regional map or a global picture. Selecting the “A Day” option gives the hourly temperature predictions along with sky cover. Extra information such as the wind speed, humidity, sunset and sunrise times are also included and presented in an orderly manner. There is also a refresh feature by downward swiping and forward and back motion by left and right swiping. In short, it is a very dedicated web app.

Besides that, OpenWeather Ltd is the owner of the OpenWeatherMap online service. This web app, like Forecast web app, provides worldwide weather data coverage including the current weather data, forecasts, nowcasts and historical data starting from 1979. It utilizes meteorological broadcast services and raw data from airport weather stations, radar stations and other weather stations. OpenWeatherMap provides more than twenty weather APIs, with almost 7000 repositories on GitHub for their 2 million customers. The APIs are able to support different units of measurement and data formats in multiple languages. This web app uses OpenStreetMap to display the weather maps and mostly provides free of charge services however most of the services provided require paid subscriptions and they are not open to user contribution.

1.5 Problem Statement

Python provides a quick and easy way to build applications including web applications. Students are required to study and learn how to use the Python programming language to develop a full feature and working web application that helps to solve any chosen industrial – based problems. The web application must be built using any preferred framework which can be integrated with Python.

For our group, we choose to create a fully functioning weather web application using Python and Django, whereby this web application can help companies and small industries whose business rely on the weather forecast.

1.6 Objectives

This weather web application project consists of a few objectives which are:

a) To solve industrial – based problems

b) To develop a web – based project

c) To successfully build and run a fully functioning weather web application with additional features.

CHAPTER 2

METHODOLOGY

2.1 Software Required

1) Django version 3.1.3
2) Python version 3.8.5
3) PyCharm Community version 2020.2.3
4) Python Virtual Environment
5) Ubuntu operating system version 20.04 LTS
6) Beautiful Soup package version 4.9.2
7) Requests library

2.2 Procedures to Build the Weather Application

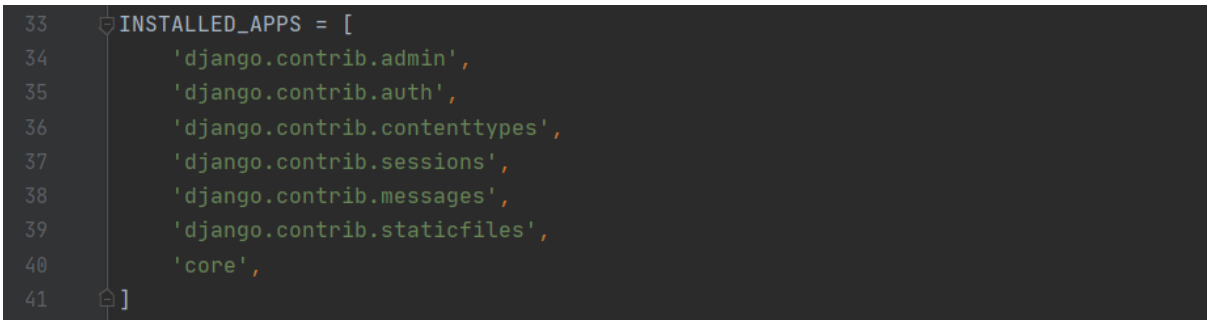
This project aims to build a weather application to solve industrial-based problems where these industries are weather-related businesses. These industries include mining, construction, retail, turf sports and amusement park. This project is built by using PyCharm and Django from scratch. The features of the weather application are displaying region of the city, day and time, status of the weather, temperature, precipitation, humidity and wind.

2.2.1 Installation

First and foremost, a new directory entitled “weather_project” is created in the home of Ubuntu. Next, the terminal is open, and the path is cd to the “weather_project” directory. A virtual environment directory entitled “venv” is created using the command “virtualenv venv”. The purpose of using a virtual environment is to avoid confusion in the Ubuntu operating system when deploying as there are different versions of the software like Django and Python language in the Ubuntu. Then, the virtual environment “venv” is activated by using the command “source venv/bin/activate”. As a result, “(venv)” is seen in front of the folder path. Furthermore, the command “django-admin startproject weatherapp” is used to create a new directory for this project. Lastly, PyCharm is opened. In PyCharm, the project interpreter is changed to the existing environment that has been created in venv which is Python 3.8. Proceed to the local terminal of the PyCharm to install Django by using the command “pip install django”.

2.2.2 Creating the Weather Application

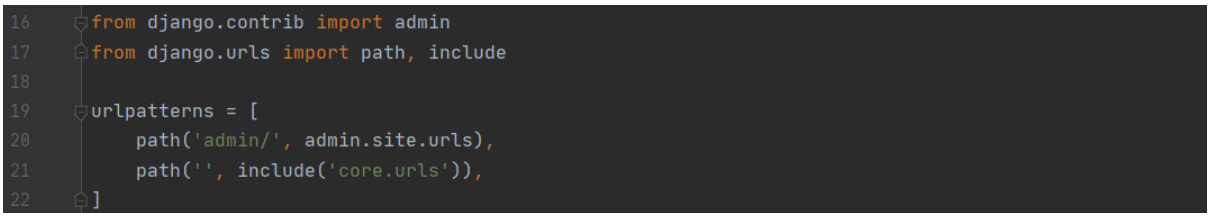
The command "python manage.py startapp core" is used to make a directory entitled "core". Next, go to the "weatherapp" directory to locate the settings.py file and install the core directory in the INSTALLED_APPS by typing "core".



```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'core',
]
```

Figure 2.1: setting.py in "weatherapp" directory

Figure 2.1 shown the code for setting.py that needs to be altered in this project. For this weather application project, only one URL is required, which is the home page. Then, the "urls.py" file in the "weatherapp" directory is located by typing this command "path ('', include('core.urls')),'' in the urlpatterns branch. Besides, the admin, path and include need to be imported as well.

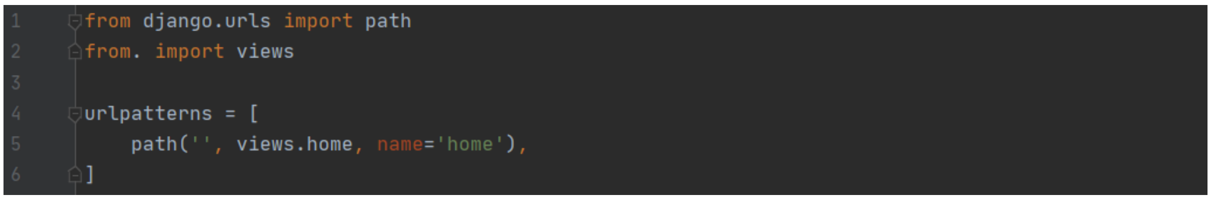


```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('core.urls')),
]
```

Figure 2.2: "urls.py" in weatherapp directory

Figure 2.2 shown the code for urls.py that needs to be altered in this project.

2.2.3 Adding the Template and View



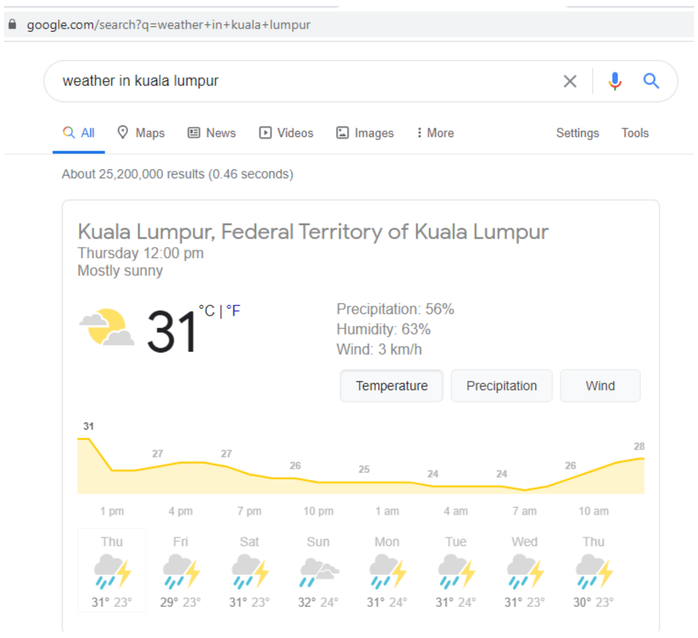
```python
from django.urls import path
from. import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

Figure 2.3: "urls.py" in core directory

A new file entitled “urls.py” is created inside the core directory. For the urlpatterns, since it is in the homepage, a function named “home” and an optional argument “home” are needed. Figure 2.3 shows the code for the “urls.py” file that needs to be altered in this project. Next, go to the “views.py” file in the core directory to define a “home” function with the code “def home(request):” so that it can get a request. The Django server is then run by using the command “python manage.py runserver”. If the server is successfully run, the link for the server http://127.0.0.1:8000/ is given. After that, a new file entitled “templates/core/home.html” is created in the core directory. Basically, the “home.html” is used to design the homepage of the weather application like designing the header, styling, creating a form for the user to insert the name of city, and linking of the submit button to scrape the weather data from Google.

2.2.4 Scraping the weather data from Google

For the “views.py” file in the core directory, the code “if ‘city’ in request.GET:” is added so that when the user enters the name of a city in the “form” section, all the attributes of the city is stored in the “request.GET”. This weather application functions by scraping the weather data from Google instead of fetching weather data from any weather API.


google.com/search?q=weather+in+kuala+lumpur

weather in kuala lumpur

About 25,200,000 results (0.46 seconds)

Kuala Lumpur, Federal Territory of Kuala Lumpur
Thursday 12:00 pm
Mostly sunny

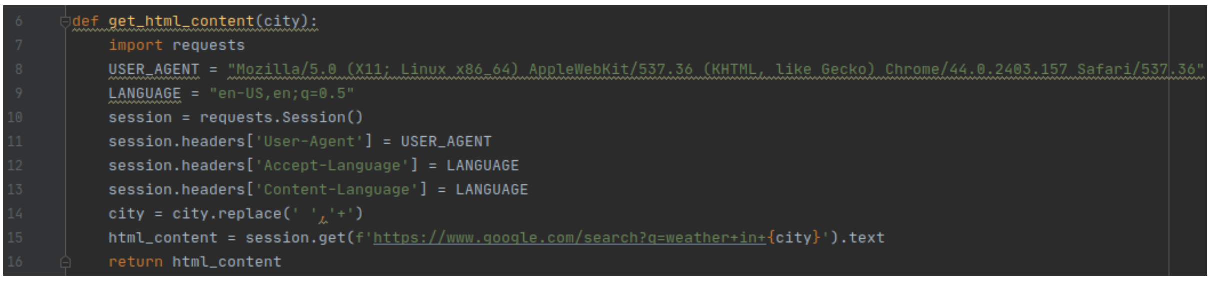
31 °C | °F

Precipitation: 56%
Humidity: 63%
Wind: 3 km/h

Figure 2.4: The weather data from Google website

Figure 2.4 shows the weather data of the city, Kuala Lumpur in Google website. This weather data that consists of the region, day and time, status of weather, temperature, precipitation, humidity and wind will be scraped from Google into our weather application. The URL of this website can be easily obtained at the top of the Google website.

Then, the "requests" library is installed by using the "pip install requests" command and the Beautiful Soup package is installed using "pip install beautifulsoup4" in the local terminal to scrape the weather data from Google. However, Google will not allow users to simply fetch its data and thus we have to add some commands to presume our webpage as a search engine.

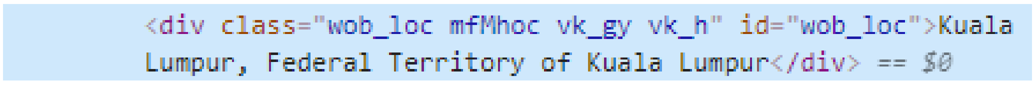


```
6   def get_html_content(city):
7       import requests
8       USER_AGENT = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36"
9       LANGUAGE = "en-US,en;q=0.5"
10      session = requests.Session()
11      session.headers['User-Agent'] = USER_AGENT
12      session.headers['Accept-Language'] = LANGUAGE
13      session.headers['Content-Language'] = LANGUAGE
14      city = city.replace(' ','+')
15      html_content = session.get(f'https://www.google.com/search?q=weather+in+{city}').text
16      return html_content
```

Figure 2.5: views.py in core directory

Figure 2.5 shows the codes in "views.py" file of core directory. By referring to the codes from line 6 to line 16, the "get_html_content" function tends to tell Google that this weather application is only browsing the Google website.

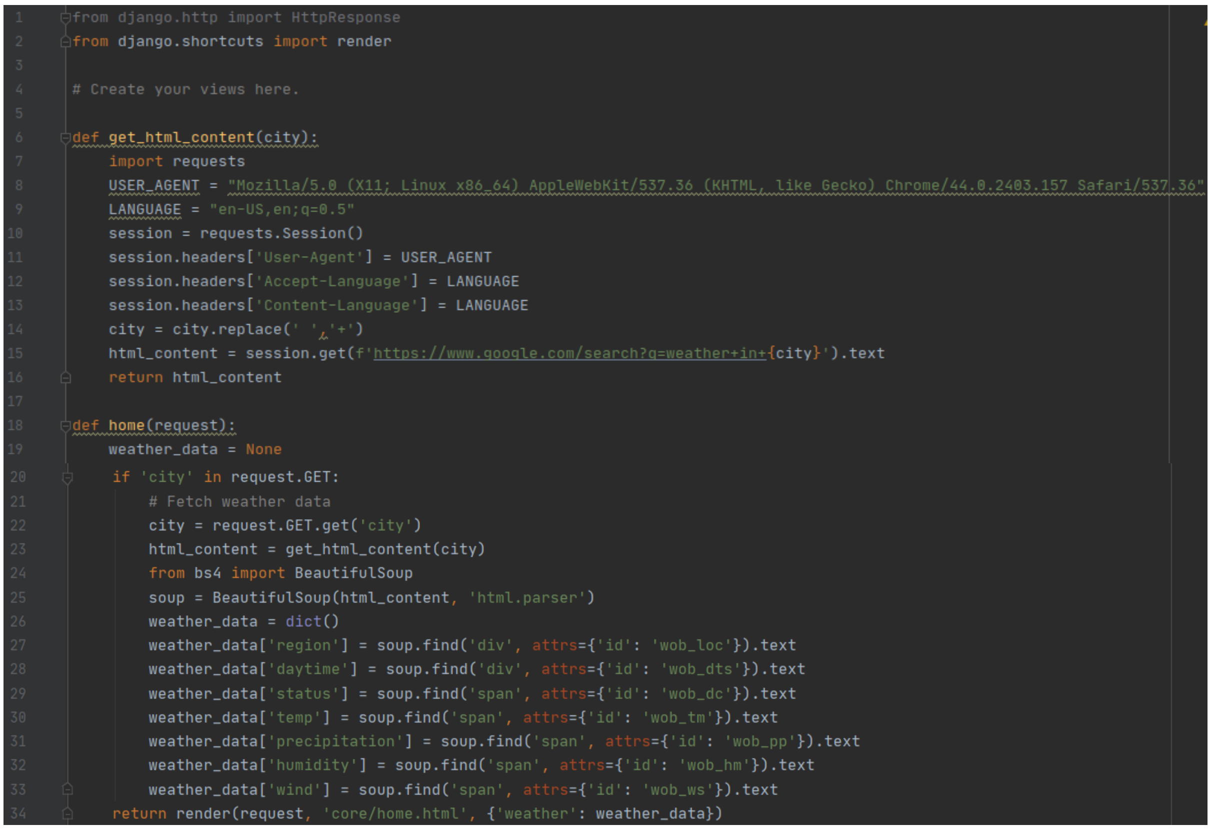
2.2.5 Displaying the Weather Data in the Template



```
<div class="wob_loc mfMhoc vk_gy vk_h" id="wob_loc">Kuala
Lumpur, Federal Territory of Kuala Lumpur</div> == $0
```

Figure 2.6: Result of inspecting the region Google

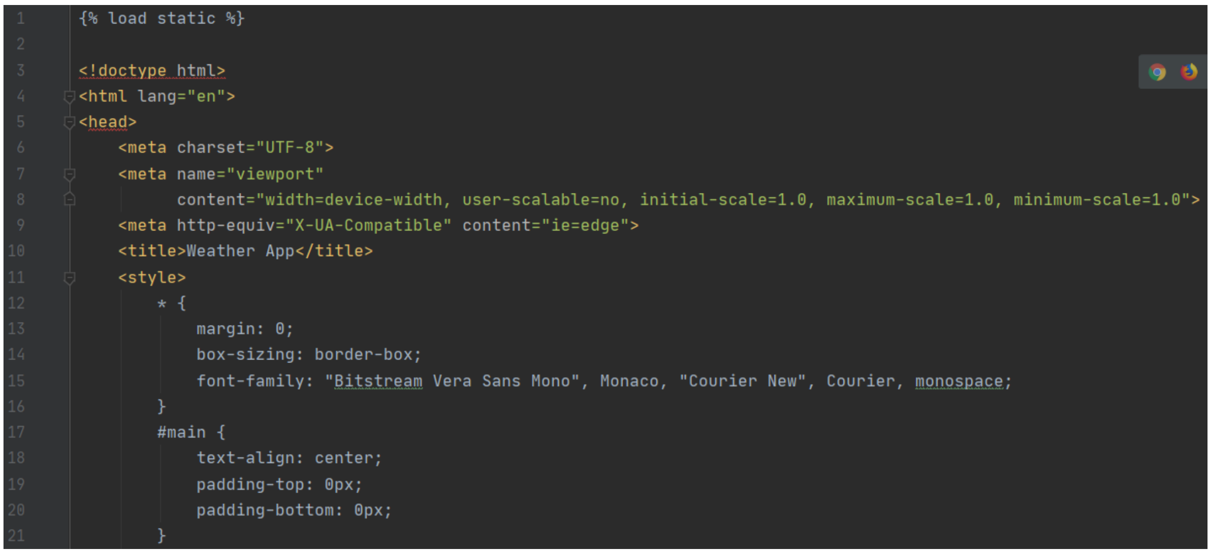
Figure 2.6 shows the result of inspecting the region in Google. The "div" and "attribute id" of the region can be obtained here. The steps are then repeated for day and time, status of the weather, temperature, precipitation, humidity and wind so that we can obtain all this weather information from Google.



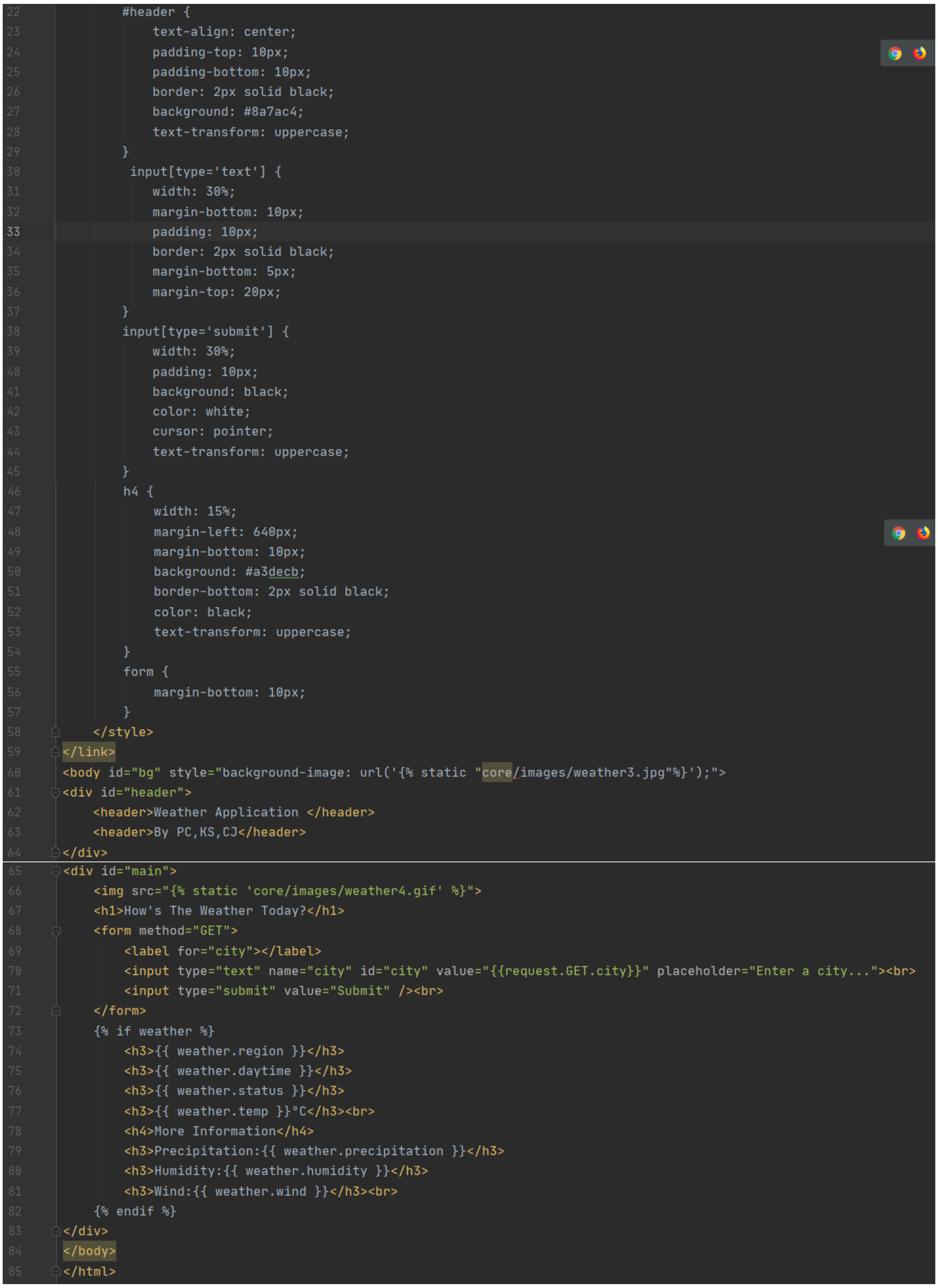
```
1   from django.http import HttpResponse
2   from django.shortcuts import render
3
4   # Create your views here.
5
6   def get_html_content(city):
7       import requests
8       USER_AGENT = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36"
9       LANGUAGE = "en-US,en;q=0.5"
10      session = requests.Session()
11      session.headers['User-Agent'] = USER_AGENT
12      session.headers['Accept-Language'] = LANGUAGE
13      session.headers['Content-Language'] = LANGUAGE
14      city = city.replace(' ','+')
15      html_content = session.get(f'https://www.google.com/search?q=weather+in+{city}').text
16      return html_content
17
18  def home(request):
19      weather_data = None
20      if 'city' in request.GET:
21          # Fetch weather data
22          city = request.GET.get('city')
23          html_content = get_html_content(city)
24          from bs4 import BeautifulSoup
25          soup = BeautifulSoup(html_content, 'html.parser')
26          weather_data = dict()
27          weather_data['region'] = soup.find('div', attrs={'id': 'wob_loc'}).text
28          weather_data['daytime'] = soup.find('div', attrs={'id': 'wob_dts'}).text
29          weather_data['status'] = soup.find('span', attrs={'id': 'wob_dc'}).text
30          weather_data['temp'] = soup.find('span', attrs={'id': 'wob_tm'}).text
31          weather_data['precipitation'] = soup.find('span', attrs={'id': 'wob_pp'}).text
32          weather_data['humidity'] = soup.find('span', attrs={'id': 'wob_hm'}).text
33          weather_data['wind'] = soup.find('span', attrs={'id': 'wob_ws'}).text
34      return render(request, 'core/home.html', {'weather': weather_data})
```

Figure 2.7: “views.py” in core directory

Figure 2.7 shows the “view.py” directory in the core directory where in the home function from line 18 to line 34, BeautifulSoup package that is installed is used to scrape the data from Google.



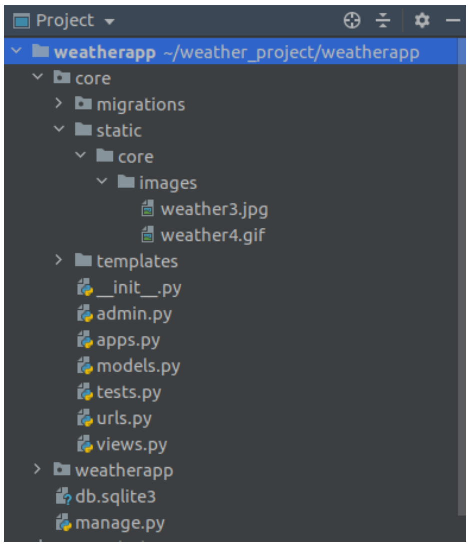
```
{% load static %}

<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Weather App</title>
    <style>
        * {
            margin: 0;
            box-sizing: border-box;
            font-family: "Bitstream Vera Sans Mono", Monaco, "Courier New", Courier, monospace;
        }
        #main {
            text-align: center;
            padding-top: 0px;
            padding-bottom: 0px;
        }
```



```
        #header {
            text-align: center;
            padding-top: 10px;
            padding-bottom: 10px;
            border: 2px solid black;
            background: #8a7ac4;
            text-transform: uppercase;
        }
         input[type='text'] {
            width: 30%;
            margin-bottom: 10px;
            padding: 10px;
            border: 2px solid black;
            margin-bottom: 5px;
            margin-top: 20px;
        }
        input[type='submit'] {
            width: 30%;
            padding: 10px;
            background: black;
            color: white;
            cursor: pointer;
            text-transform: uppercase;
        }
        h4 {
            width: 15%;
            margin-left: 640px;
            margin-bottom: 10px;
            background: #a3decb;
            border-bottom: 2px solid black;
            color: black;
            text-transform: uppercase;
        }
        form {
            margin-bottom: 10px;
        }
    </style>
</link>
<body id="bg" style="background-image: url('{% static "core/images/weather3.jpg"%}');">
<div id="header">
    <header>Weather Application </header>
    <header>By PC,KS,CJ</header>
</div>
<div id="main">
    <img src="{% static 'core/images/weather4.gif' %}">
    <h1>How's The Weather Today?</h1>
    <form method="GET">
        <label for="city"></label>
        <input type="text" name="city" id="city" value="{{request.GET.city}}" placeholder="Enter a city..."><br>
        <input type="submit" value="Submit" /><br>
    </form>
    {% if weather %}
        <h3>{{ weather.region }}</h3>
        <h3>{{ weather.daytime }}</h3>
        <h3>{{ weather.status }}</h3>
        <h3>{{ weather.temp }}°C</h3><br>
        <h4>More Information</h4>
        <h3>Precipitation:{{ weather.precipitation }}</h3>
        <h3>Humidity:{{ weather.humidity }}</h3>
        <h3>Wind:{{ weather.wind }}</h3><br>
    {% endif %}
</div>
</body>
</html>
```

Figure 2.8: home.html in core directory

Figure 2.8 shown the codes in “home.html” file to design the weather application. For line 3 to line 58, the codes are used for styling purposes such as padding, determining the background colour of the text, the front, adjusting the width, adjusting the text-aligning and so forth. For line 27 and line 50, the hex numbers #8a7ac4 and #a3decb represents Django colour codes. From line 61 to line 72, the codes display how the homepage of weather application looks like and a form is created to scrape the weather data from Google. To maintain the name of city that is typed in the form, line 70 is added to access to the request object. Line 73 to 82 are then used to display the scraped weather data from Google.

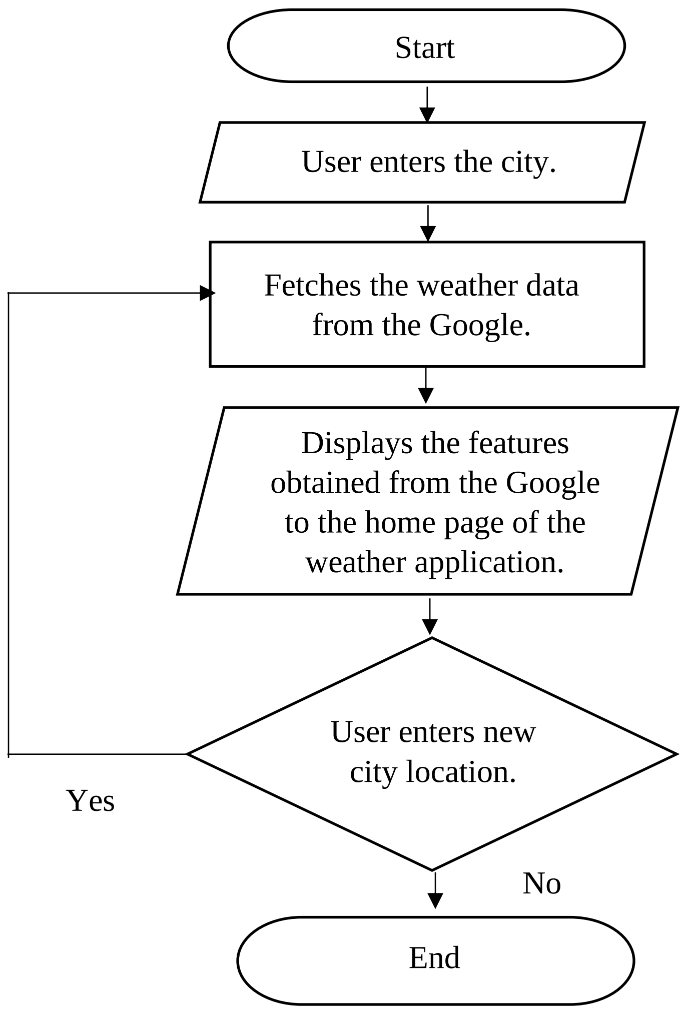


```
Project
weatherapp ~/weather_project/weatherapp
  core
    migrations
    static
      core
        images
          weather3.jpg
          weather4.gif
    templates
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py
  weatherapp
  db.sqlite3
  manage.py
```

Figure 2.9: The “static” directory is created

Figure 2.9 shown the “static” directory created in order to upload images into Django server. The directory path “static/core/images” is created to store the required images. The code “{% load static %}” in line 1 and line 66 are added in “home.html” file to display the uploaded images in Django server whereas line 60 is to display the uploaded image as background of the weather application. The weather application is accomplished.

2.3 Flow Chart of Weather Application



Start
User enters the city.
Fetches the weather data from the Google.
Displays the features obtained from the Google to the home page of the weather application.
User enters new city location.
Yes
No
End

Figure 2.10: Flow chart of weather application

CHAPTER 3

RESULTS AND DISCUSSION

3.1 Web Application Design

Our Weather Application is a feature-complete web-based project developed using Python programming language. A high-level Python Web framework that is known as Django is also employed in this project to develop this web application. Focussing on simplicity, we have developed a user-friendly web application using Python which can be easily reached at the web address of http://127.0.0.1:8000/. Upon entering the website, the webpage of our Weather Application as shown in the figure below will be displayed to users. As observed, our web application is designed emphasizing on simplicity by only displaying useful information for users. The headers show the title of the application and indicates the name of developers. Then, the body shows the main function of our application, which is displaying the weather information of a city based on user's input. By simply entering the name of a city in the input area and submitting it, our Weather Application will then display the current weather information of the specified place neatly on the webpage to be viewed by users.

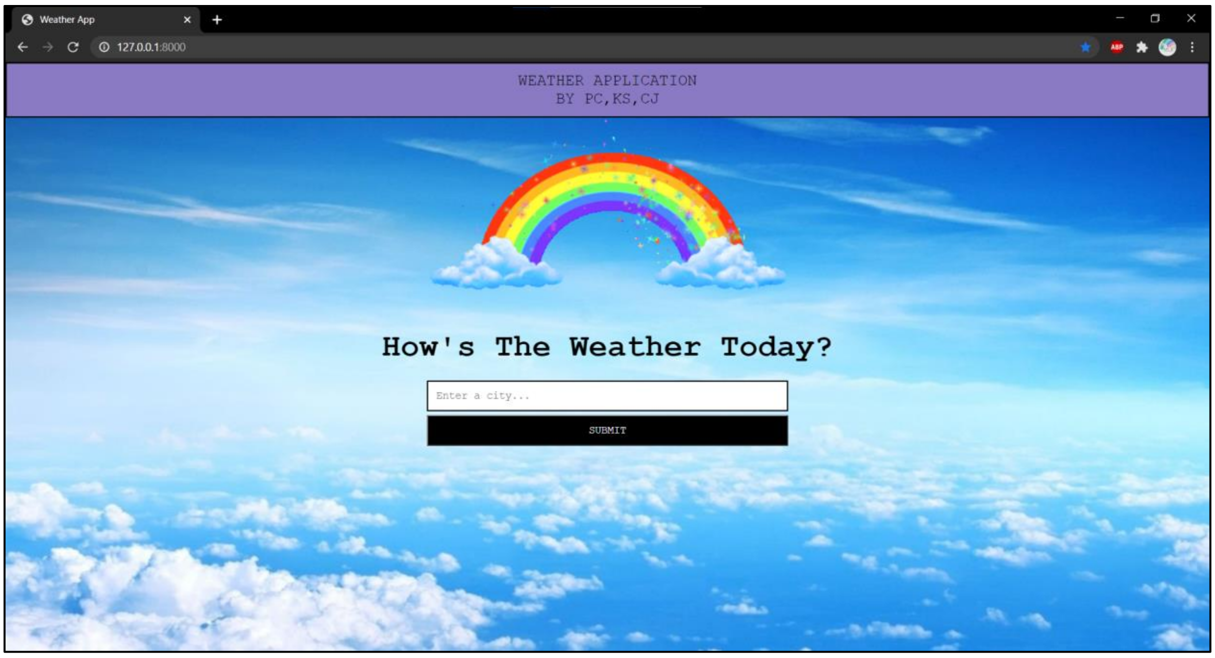
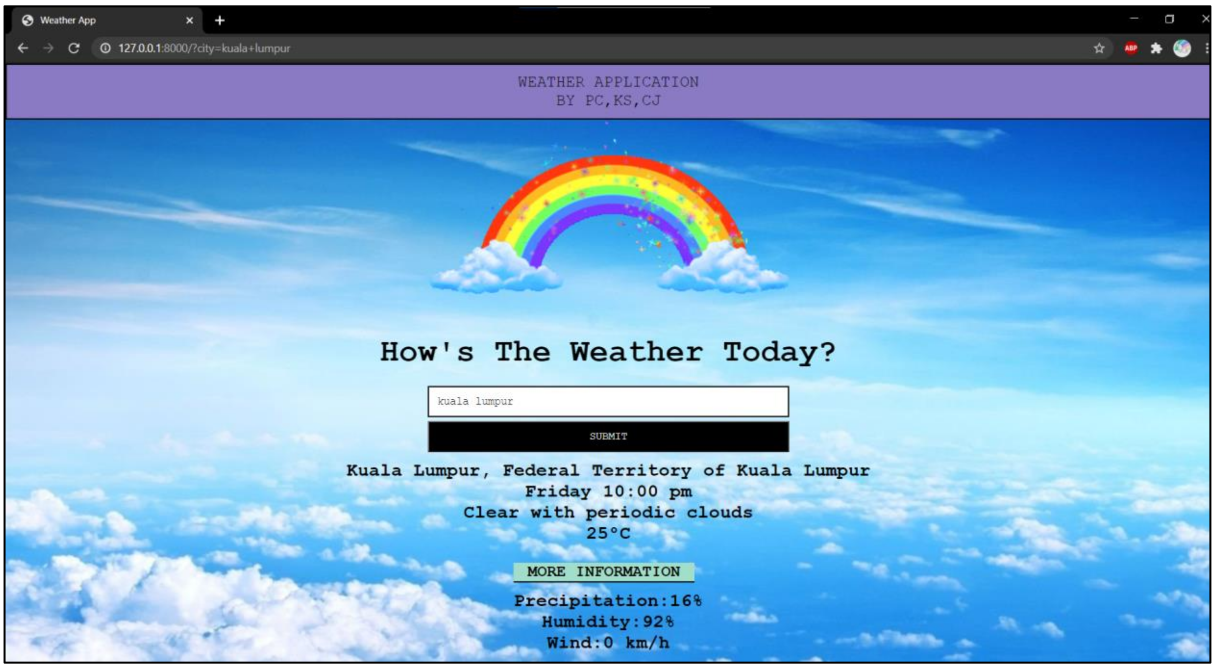


Figure 3.1: Weather Application webpage

3.2 Result Analysis

Upon receiving an input of a city name from the user, our Weather App will scrape the weather information from Google and display them neatly on our webpage as shown in the figure below.



Kuala Lumpur, Federal Territory of Kuala Lumpur
Friday 10:00 pm
Clear with periodic clouds
25°C

Precipitation:16%
Humidity:92%
Wind:0 km/h

Figure 3.2: After receiving input from user

The main features of weather information that are scraped and provided for users include: -

i) Name of city and country
ii) Current day and time of the city
iii) Weather condition
iv) Temperature

Through this information provided for users, they will be able to know the current climate condition of any place around the world easily by using our Weather Application. As our application is developed by scraping data from Google, a wide database is available and thus weather information from any city, be it well-known places or unpopular countries can be obtained. Aside from only the weather condition like sunny or rainy, users can also get to know about the present time of a city that has a different time zone and their current temperature as well. These information are provided to users so that through a single app, they can obtain various kind of useful details on a specified place.

Aside from the four main features displayed for users when they search for weather information of a city using our Weather Application, we also provide further detailed information on the weather intended to solve industrial-based problem. On our webpage, more information on the weather of a certain city is displayed below the main features. These extra features provided to users include: -

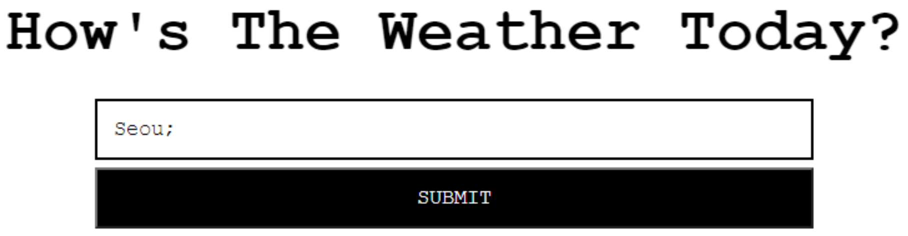
i) Precipitation
ii) Humidity
iii) Wind

With this extra information of weather provided to users, it will be useful for weather-related business or industries like mining, construction, retail, turf sports and amusement park. These industries need detailed weather information of certain cities to carry out their events and activities as planned. For instance, an amusement park needs to obtain a detailed weather information of a city including the weather condition, temperature, precipitation, humidity, and wind so that they can decide on opening their games and activities stations or not. Thus, extra information on the weather of a city are proved to be useful to these business and industries as they act as steppingstones for accurate decision making.

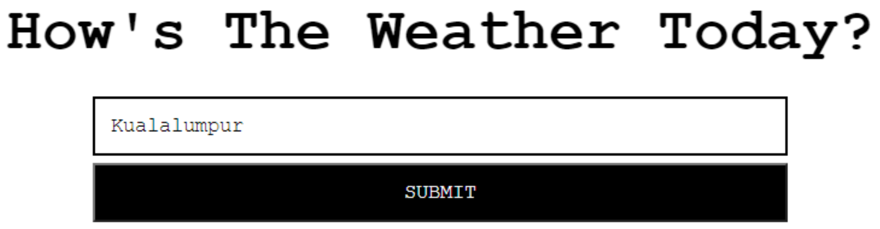
3.3 Handling Input Errors

As our Weather Application is developed by scraping data from Google using the Beautiful Soup package, this serves as an advantage or benefit as a large database is available. Users can input almost every available city from all around the world and our application is able to provide their corresponding weather information as these data are obtained from Google itself. Aside from this, our application can also handle input errors from users when they enter a city name on our webpage. Input errors include: -

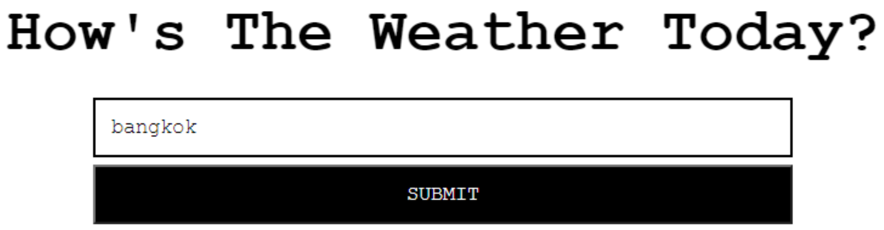
i) Spelling



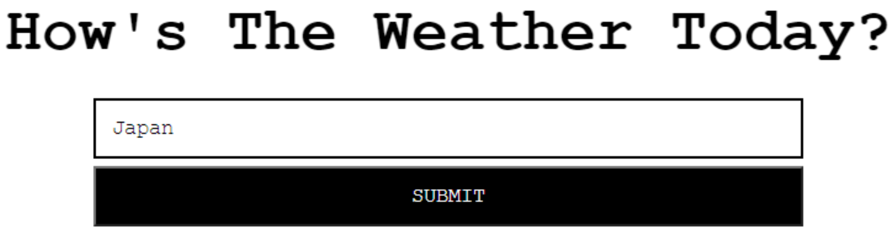
ii) Spacing



iii) Capitalization



iv) Naming



The first figure shows a spelling error where the city named “Seoul” is misspelled as “Seou;” by the user. Next, the second figure shows a spacing error where a user has neglected a spacing in between the city named “Kuala Lumpur”. Followed by this is a capitalization error in which the user neglected the upper-case letters in a city name and lastly, a naming error is shown where the user did not input the correct city name and instead entered a country name in general. All of these are examples of input errors that might happen when users input their desired city name into our Weather Application. By considering these user errors, thus we chose to develop our application by scraping data from Google to have a large database. By doing so, our Weather Application will be able to handle all these minor input errors and recognize the corresponding city names, thus displaying the respective weather information output on our webpage. For the city input with naming error, it will be handled by displaying the weather information in one of the cities in that country name entered. Therefore, as we have considered that there exist some cities that have complicated names and spellings, this user-friendly application which can handle minor input errors from users is developed.

CHAPTER 4

CONCLUSION

Weather forecast systems takes into account the current, past and future data in order to predict future weather conditions. Many systems exist for users to communicate with the local or global weather companies and get almost accurate reading. Some systems require paid subscription for their services. Weather forecast web applications help companies or small industries to grow their business by producing products according to consumer needs based on the weather report.

In this paper, a weather web application is developed successfully. The information to be displayed on the weather web application are fetched from another weather forecast source via web scraping. The user is able to check the current weather condition of any global location that they input.

Furthermore, we have successfully demonstrated and apply the Python programming Language, Django framework and Beautiful Soup package in developing our weather web application. Django framework is very user - friendly and integrating this with the simple and logical programming language of Python and one of its feature packages which is the Beautiful Soup package, we were able to create extra features for our weather web application such as checking additional data values of precipitation, humidity and wind. Overall, our weather web application is very user - friendly and users can easily interact with it by inputting their desired location and the system will output the related data.

REFERENCES

Weather forecasting. (2020, October 20). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/Weather_forecasting

Django (web framework). (2020, November 20). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/Django_(web_framework)

PyCharm. (2020, November 14). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/PyCharm

Python (programming language). (2020, November 22). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/Python_(programming_language)

Ubuntu. (2020, November 21). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/Ubuntu

Beautiful Soup (HTML parser). (2020, November 15). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/Beautiful_Soup_(HTML_parser)

Martin, M. (2020, February 07). Forecast is a great little weather web app. Retrieved November 28, 2020, from https://www.engadget.com/2013-03-28-forecast-is-a-great-little-weather-web-app.html

OpenWeatherMap. (2020, October 26). Retrieved November 28, 2020, from https://en.wikipedia.org/wiki/OpenWeatherMap

Place to store your color Palettes. (n.d.). Retrieved November 28, 2020, from https://colorswall.com/