ABSTRACT

With the use of an API key, this project offers a sophisticated meteorological and air quality data display system that provides an in-depth understanding of the surrounding environment. The system offers a comprehensive perspective on atmospheric dynamics by utilizing real-time data streams for air quality, temperature, wind speed, atmospheric pressure, humidity, and rainfall.

Users can easily access and visualize a wide range of environmental indicators thanks to the system's user-friendly interface. One of the main features is the dynamic air quality index display, which provides important details on the amounts of pollutants in the air. Users can also measure temperature differences, examine wind patterns, keep an eye on changes in atmospheric pressure, evaluate humidity levels, and predict when rain will fall.

Through the utilization of diverse visualization methodologies, including heat maps, scatter plots, and time series graphs, users can obtain practical insights regarding air quality, weather patterns and more weather data trends. Comprehensive accuracy and coverage empower the users to make announced decisions in various domains.

The system provides users with the knowledge and insights required to efficiently traverse the intricacies of the current world by utilizing real-time data and sophisticated visualization techniques. It monitors the common people's health, plans the urban people, and researches the environment or personal well-being. The system of weather data offers more tools for analysis and making decisions.

CHAPTER 1

INTRODUCTION

Understanding and keeping an eye on environmental circumstances have grown more crucial in today's world of fast change. For a variety of uses, including public health, urban planning, and agriculture, having access to real-time data on air quality, temperature, wind speed, atmospheric pressure, humidity, and rainfall is crucial.

This project provides a comprehensive weather and air quality monitoring system that uses HTML and CSS for frontend development, Python for backend processing, and the power of APIs for data retrieval in order to meet this need. The use of an API key to access and obtain vital environmental data in real-time is the foundation of this system. This information, which includes temperature readings, wind speed measurements, air quality indices, atmospheric pressure, humidity, and rainfall data.

The frontend UI and UX developed by HTML and CSS which offers users friendly and intuitive experience. When user comes to this system , they can know the weather data and air quality of the air. The user have to enter the location of them where they want to know in the location map, it works by latitude and longitude by the location of the user. Then it displays the current weather data and air quality. More than this the user can know the information like recommendations for how to react at the time of the weather.

The backend processes the request from the Open weather API key for current weather data and air quality using the python. Python handles the backend process and manipulate data. Through the python library and framework Django, the system manages the API requests, processes incoming data streams, and generates dynamic visualizations for frontend display. This backend architecture ensures the efficiency, reliability, and scalability of the system, enabling seamless integration with various data sources and user interfaces.

This project represents a significant advance in environmental monitoring and data visualization. By harnessing the power of APIs, Python, HTML, and CSS, the system offers a comprehensive solution for accessing, analyzing, and visualizing critical environmental data. Whether it's for scientific research, policy-making, or personal awareness, this system provides the tools necessary to navigate and understand the complexities of our ever-changing environment.

CHAPTER 2

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS

- **PROCESSOR:**

  A modern multi-core processor (e.g., Intel Core i3 or equivalent) for handling computational tasks efficiently.

- **MEMORY:**

  At least 4GB of RAM to ensure smooth performance, particularly when handling multiple user requests and data processing tasks.

- **STORAGE:**

  Sufficient storage space for storing project files, dependencies, and database records. An SSD (Solid State Drive) is recommended for faster read/write speeds.

SOFTWARE REQUIREMENTS

- **OPERATING SYSTEM:**

  The project should be compatible with popular operating systems such as Windows, macOS, and Linux distributions like Ubuntu.

- **PYTHON:**

  The Django framework requires Python to be installed. Ensure you have Python installed on your system, preferably the latest stable version.

- **DATABASE MANAGEMENT SYSTEM:**

  SQLite is commonly used for development purposes, but you may consider using other database systems like MySQL for production environments.

- **TEXT EDITOR:**

  Choose a text editor or IDE suitable for Python development, like Visual Studio Code.

- **VERSION CONTROL:**

  Consider using a version control system like Git for managing project files and collaborating with team members.

CHAPTER 3

METHODOLOGY

DEVOLOPMENT ENVIRONMENT SETUP

Download and install Python from the official website. Additionally, text integrated development environment (IDE) for Project development like Visual Studio Code and install the necessary plugins for Python & Django.

PROJECT STRUCTURE

- Project Root Folder – **WeatherDataApplications**
- App Folder – **WeatherApp**
- HTML Folder – **templates**
- CSS&JS Folder **- static**
- Weather API Modules – **weatherData.py | airData.py | uvIndexData.py | solarradiation.py**
- Visualization modules – **weatherplot.py | airplot.py**
- Django Script Manage File – **manage.py**
- SQL Database File – **db.sqlite3**

Open Weather API key:

- Data Coverage: Provides comprehensive weather data globally.
- Endpoints: Offers various endpoints for accessing current weather, forecasts, air pollution data, and more.
- Parameters: Allows customization of data with parameters like location coordinates, units, and language.
- Authentication: Requires an API key for access, ensuring authentication and managing usage.
- Utility: Valuable resource for developers, businesses, and individuals for integrating weather data into applications and services.
- Example: With the Current Weather Data API endpoint, you can retrieve current weather conditions for a specific location by providing latitude and longitude coordinates. For instance, a request to api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={API_KEY} returns JSON data with information such as temperature, humidity, wind speed, and weather description for the specified location.

CHAPTER 4

REQUIRED PACKAGES

4.1 DJANGO

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It's built on the principle of DRY (Don't Repeat Yourself) and emphasizes reusability and pluggability of components.

Command: ***pip install django***

4.2 REQUEST

The requests package is a popular Python library used for making HTTP requests and interacting with web APIs. It provides a simple and elegant API for sending HTTP requests and handling responses, making it a preferred choice for web scraping, API integration, and other web-related tasks in Python.

To install the requests package, you can use pip, Python's package manager. Open a terminal or command prompt and run the following command:

Command: ***pip install requests***

4.3 MATPLOTLIB

Matplotlib is a comprehensive Python library for creating static, animated, and interactive visualizations. It provides a wide range of plotting functionality, allowing users to generate high-quality graphs, charts, histograms, scatter plots, and more.

Command: ***pip install matplotlib***

4.4 JSON

- JSON (JavaScript Object Notation) is a lightweight data-interchange format.
- It uses key-value pairs to represent data, similar to Python dictionaries.
- JSON supports various data types, including strings, numbers, booleans, arrays, and objects.
- It's commonly used in web development for transmitting data between servers and clients, especially in RESTful APIs.
- Python's built-in json module provides functions for encoding Python objects into JSON format (json.dumps()) and decoding JSON data into Python objects (json.loads()).

- **Example:**

```
return_data = {'success': True,
               'response': 'Success',
               'air_quality_category':air_quality_category,
               'pm25':pm25,
               'pm10':pm10,
                 }
```

4.5 DATA POINTS

Weatherdata:

url= f'https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={api_key}'

Solar radiation:

url= f'https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={api_key}'

UVIndexData:

url= f'https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={api_key}'

Airdata:

url= f'https://api.openweathermap.org/data/2.5/air_pollution?lat={latitude}&lon={longitude}&appid={api_key}'

CHAPTER 5

SYSTEM DESIGN

5.1 ARCHITECTURE

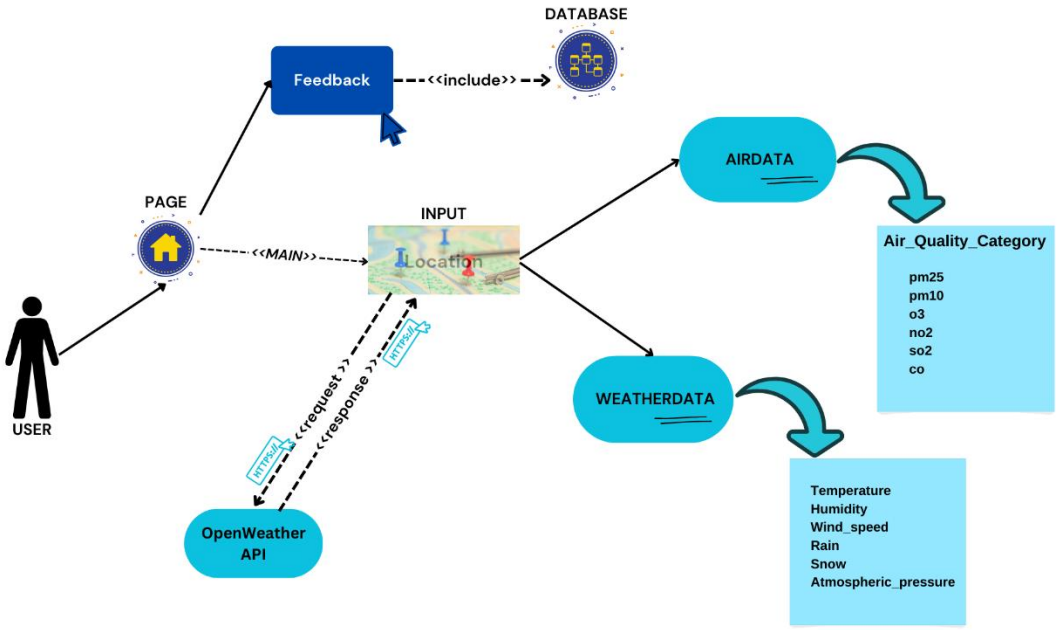
The weather data visualization project follows a client-server architecture, with frontend components implemented using HTML, CSS and backend components developed with Django, a Python-based web framework. The system architecture consists of the following layers:

5.2 DATA FLOW

- **User Interaction:** Users interact with the system by providing location inputs via the frontend interface.
- **Request Processing:** The frontend forwards HTTP requests to the corresponding Django views based on user actions.
- **Business Logic Execution:** Within Django views, the system executes business logic, which involves processing weather data, messages, and interacting with external APIs such as the OpenWeather API.
- **Response Generation:** Modules such as airData, solarradiation, uvIndexData, and weatherData are responsible for generating weather data and visualizations based on user inputs. These modules interact with the OpenWeather API to fetch relevant information.
- **Response Delivery:** Once generated, the response is transmitted back to the frontend. The frontend then presents the data to the user through text and chart format interfaces.

5.3 WORKFLOW:

- **Location Input:** Users can input their latitude and longitude coordinates through a form. This likely happens on the frontend side of your application, where users provide their location information, and then this information is sent to the backend for processing.
- **Weather and Air Quality Data Retrieval:** Once the user submits their location, the backend fetches current weather and air quality data for that location using the provided latitude and longitude coordinates. This data is obtained from the OpenWeatherMap API.
- **Visualization:** The retrieved weather and air quality data is then visualized using matplotlib, and the resulting graphs are saved as images. These images are likely displayed on the frontend to provide a graphical representation of the weather and air quality information.
- **Feedback Submission:** Users can also provide feedback through a form. When a user submits feedback, their name, email, and message are captured and stored in the database.
- **Rendering Data:** The rendered weather and air quality data, along with any feedback submitted by users, are displayed on the frontend. Users can view this information to understand the current weather conditions and air quality at their specified location.



5.4 FUTURE ENHANCEMENT:

History Data Tracking:

- Capture user interactions and preferences to provide personalized recommendations.
- Utilize databases to efficiently store and manage historical data.
- Enable users to review activity history for insights and decision-making.

Predictive Capabilities:

- Develop predictive models based on historical data to anticipate user behavior.
- Offer personalized content recommendations and predictive alerts.
- Continuously refine models for improved accuracy over time.

Health Management Recommendations:

- Integrate health monitoring functionalities for personalized recommendations.
- Collaborate with health professionals to offer evidence-based advice.
- Provide features like meal planning and workout scheduling for well-being.

Notifications:

- Implement a customizable notification system for important updates and reminders.
- Use various channels for timely delivery of notifications.
- Prioritize and schedule notifications intelligently to avoid user overload.

CHAPTER 6

SOURCE CODE

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    {% load leaflet_tags %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title> Welcome! Weather Visualization</title>
    <link rel="icon" type="image/x-icon" href="{% static 'weatherappImage/clouds.png' %}">
    {% leaflet_js %}
    {% leaflet_css %}
    <link rel="stylesheet" href="{% static 'weatherappCss/index.css' %}">
</head>
<body>
    <div class="banner" style="background-image:linear-gradient(rgba(0,0,0,0.60),rgba(0,0,0,0.65)), url('{% static 'weatherappImage/nature.jpg' %}');">
        <div class="navbar">
            <img src="{% static 'weatherappImage/clouds.png' %}" alt="clouds" class="logo">
            <div class="content">
                <h1> Welcome To Weather Visualization</h1>
                <p>“You are the sky. Everything else – it’s just the weather.”</p>
                <form method="post" action="{% url 'home' %}">
```

```
                {% csrf_token %}
                <input type="hidden" id="id_latitude" name="latitude">


                <input type="hidden" id="id_longitude" name="longitude">
                <button type="button" class="button-link" onclick="useCurrentLocation()">Current Location</button>
                <h6>or</h6>
                <h3 style="margin: 20px 10px; width: 150px; margin-left:46%; text-align: center;font-weight: lighter; font-style: italic;">Select Location</h3>
                <div id="map" style="height: 500px; width:350px; margin-left:41%; border-radius: 10%;"></div>
                <button type="submit" class="button-link" >Submit-location</button>
            </form>
        </div>
    </div>
</div>
<script>
    var map = L.map('map').setView([ 11.719238, 78.0800749], 13);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 19
    }).addTo(map);
    var marker;
    map.on('click', function(e) {
        updateLocation(e.latlng.lat, e.latlng.lng);
        if (marker) {
            map.removeLayer(marker);
        }
        marker = L.marker(e.latlng).addTo(map);
    });
```

```
        function useCurrentLocation() {
            if (navigator.geolocation) {

            navigator.geolocation.getCurrentPosition(function(position) {
                updateLocation(position.coords.latitude, position.coords.longitude);
                if (marker) {
                    map.removeLayer(marker);
                }
                marker = L.marker([position.coords.latitude, position.coords.longitude]).addTo(map);
            });
            } else {
                alert("GeoLocation is not supported by this browser.");
            }
        }
        function updateLocation(latitude, longitude) {
            document.getElementById('id_latitude').value = latitude;
            document.getElementById('id_longitude').value = longitude;
        }
    </script>
</body>
</html>
```

main.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Weather-Main Page</title>


<link rel="icon" type="image/x-icon" href="{% static 'weatherappImage/clouds.png' %}">
    <link rel="stylesheet" href="{% static 'weatherappCss/main.css'%}">
</head>
<body>
    <div class="banner" style="background-image:linear-gradient(rgba(0,0,0,0.60),rgba(0,0,0,0.65)), url('{% static 'weatherappImage/nature.jpg'%}');">
        <div class="navbar">
            <img src="{% static 'weatherappImage/clouds.png'%} " alt="clouds" class="logo">
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="{% url 'contact' %}">FEED BACK</a></li>
            </ul>
            <div class="content">
                <h1>Weather Visualization</h1>
                <p>“You are the sky. Everything else – it’s just the weather.”</p>
                <button type="button"><span></span><a href="{% url 'airdata' %}" class="button-link">AirData</a></button>
                <button type="button"><span></span><a href="{% url 'weatherdata' %}" class="button-link" >WeatherData</a></button>
            </div>
        </div>
    </div>
</body>
</html>
```

main.css/style sheet

```
*{
    margin: 0;
    padding: 0;
    font-family: sans-serif;
}

.banner {
    width: 100%;
    height: 100vh;
    /* Double-check the path and ensure it's correct */
    background-image: linear-gradient(rgba(0, 0, 0, 0.75), rgba(0, 0, 0, 0.5));
    background-size: cover;
    background-position: center;
  }

.navbar{
    width: 85;
    margin: auto;
    padding: 10px 0;
    display: flex;
    align-items: center;
    justify-content: space-between;
}
.logo{
```

```
    width: 30px;
    cursor: pointer;


margin-left: 0.3cm;
}
.navbar ul li{
    list-style: none;
    display: inline-block;
    margin: 0 20px;
    position: relative;
}
.navbar ul li a{
    text-decoration: none;
    color:whitesmoke;
    text-transform: uppercase;
}
.navbar ul li::after{
    content: '';
    height: 3px;
    width: 0;
    background: #009699;
    position: absolute;
    left: 0;
    bottom: -8px;
    transition: 0.5s;
}
.navbar ul li:hover::after{
    width:100%;
```

```
}

.content{
    width: 100%;
    position: absolute;
    top: 30%;
    transform: translate(0%);
    text-align: center;
    color:azure;
}
.content h1{
    font-size: 70px;
    margin-top: 80px;
}
.content p{
    margin:10px auto;
    font-weight: 100;
    line-height:25px;
}
button{
    width: 150px;
    padding: 10px 0;
    text-align: center;
    margin: 20px 10px;
    border-radius:25px;
    font-weight: bold;
    border: 2px solid #009688;
```

```
    background: transparent;
    color: #fff;

cursor: pointer;
    position: relative;
    overflow:hidden;

}
span{
    background: #0bc4e0;
    height: 100%;
    width: 0;
    border-radius: 25px;
    position:absolute;
    left: 0;
    bottom: 0;
    z-index: -1;
    transition: 0.5s;
}
button:hover span{
    width: 100%;
}
button:hover {
    border: none;
}
.button-link {
    text-decoration: none;
    color: inherit;
```

```
    display: inline-block;
}

.button-link button {
    width: 150px;
    padding: 10px 0;
    text-align: center;
    margin: 20px 10px;
    border-radius: 25px;
    font-weight: bold;
    border: 2px solid #0bcbed;
    background: transparent;
    color: #fff;
    cursor: pointer;
    position: relative;
    overflow: hidden;
}
.button-link button:hover span {
    width: 100%;
}
.button-link button:hover {
    border: none;
}
```

airdata.py

```
import requests
import os
```

```
def currentAirData(latitude, longitude):
    api_key = '91e35b107a50bb5769307b4aaca1fef4'

    #url Api Call
    url = f'https://api.openweathermap.org/data/2.5/air_pollution?lat={latitude}&lon={longitude}&appid={api_key}'

    #response getter
    response = requests.get(url)

    # Check the HTTP status code
    if response.status_code == 200:
        #getting json data

 air_data = response.json()

        #Accessing the data
        air_quality_index = air_data['list'][0]['main']['aqi']
        air_quality_category = get_air_quality_category(air_quality_index)
        pm25 = air_data['list'][0]['components']['pm2_5']
        pm10 = air_data['list'][0]['components']['pm10']
        o3 = air_data['list'][0]['components']['o3']
        no2 = air_data['list'][0]['components']['no2']
        so2 = air_data['list'][0]['components']['so2']
        co = air_data['list'][0]['components']['co']

        #wrapper data for return
        return_data = {'success': True,
```

```
                'response': 'Success',
                'air_quality_category':air_quality_category,
                'pm25':pm25,
                'pm10':pm10,
                'o3':o3,
                'no2':no2,
                'so2':so2,
                'co':co
        }

        return return_data
    else:
        return {'success': False, 'response': 'Something Went Wrong Please Try again !'}
# air quality category

def get_air_quality_category(aqi):
    if aqi <= 50:
        return 'Good'
    elif aqi <= 100:
        return 'Moderate'
    elif aqi <= 150:
        return 'Unhealthy for Sensitive Groups'
    elif aqi <= 200:
        return 'Unhealthy'
    elif aqi <= 300:
        return 'Very Unhealthy'
    else:
        return 'Hazardous'
```

airplot.py

```
import matplotlib.pyplot as plt
def visualize_air_quality(data):
    labels = ['PM2.5', 'PM10', 'O3', 'NO2', 'SO2', 'CO']
    values = [data['pm25'], data['pm10'], data['o3'], data['no2'], data['so2'], data['co']]
    # Bar chart
    plt.figure(figsize=(10, 6))
    plt.bar(labels, values, color=['blue', 'green', 'orange', 'red', 'purple', 'brown'])
    plt.title('Air Quality Components')
    plt.xlabel('Air Quality Components')
    plt.ylabel('Concentration (µg/m³ or ppm)')
    image_filename                                                                    =
f'/home/mothish/Projects/ArulminiProject/static/weatherappImage/airGraph.png'
    plt.savefig(image_filename)
    plt.close()
```

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from django.urls import reverse
from .forms import LocationForm
import json
from weatherData import currentWeatherData
from airData import currentAirData
from weatherplot import display_weather
```

```
from airplot import visualize_air_quality
from .models import FeedBack


def home(request):
    if request.method == 'POST':
        form = LocationForm(request.POST)
        if form.is_valid():
            latitude = form.cleaned_data['latitude']
            longitude = form.cleaned_data['longitude']
            data = {'latitude': latitude, 'longitude': longitude}
            filename = '/home/mothish/Projects/ArulminiProject/locationInfo.json'
            existing_data = read_json_file(filename)
            existing_data.update(data)

            with open(filename, 'w') as json_file:
                json.dump(existing_data, json_file)

            return render(request,'weatherapp/main.html')
    else:
        form = LocationForm()
    return render(request,'weatherapp/index.html')

def main(request):
    return render(request,'weatherapp/main.html')

def weatherdata(request):
    filename = '/home/mothish/Projects/ArulminiProject/locationInfo.json'
    location = read_json_file(filename)
```

```
    responce = currentWeatherData(location['latitude'], location['longitude'])
    display_weather(responce) # to update the weatherGraph Image

if responce['success']:
        context = {
        'weatherDescription':responce['weather_description'],
        'temperature':responce['temperature'],
        'humidity':responce['humidity'],
        'windSpeed':responce['wind_speed'],
        'rain':responce['rain'],
        'snow':responce['snow'],
        'atmosphericPressure':responce['atmospheric_pressure']
        }
    else:
        context = {
        'weatherDescription':'#',
        'temperature':'#',
        'humidity':'#',
        'windSpeed':'#',
        'rain':'#',
        'snow':'#',
        'atmosphericPressure':'#'
        }

    return render(request,'weatherapp/weatherdata.html',context)

def airdata(request):
    filename = '/home/mothish/Projects/ArulminiProject/locationInfo.json'
```

```
    location = read_json_file(filename)
    responce = currentAirData(location['latitude'], location['longitude'])


visualize_air_quality(responce)
    if responce['success']:
        context = {
            'airQualityCategory':responce['air_quality_category'],
            'pm25':responce['pm25'],
            'pm10':responce['pm10'],
            'o3':responce['o3'],
            'no2':responce['no2'],
            'so2':responce['so2'],
            'co':responce['co']
    }
    else:
        context = {
            'airQualityCategory':'#',
            'pm25':'#',
            'pm10':'#',
            'o3':'#',
            'no2':'#',
            'so2':'#',
            'co':'#'
    }


    return render(request,'weatherapp/airdata.html',context)


def contact(request):
```

```
    if request.method == 'POST':
        name = request.POST.get('name')

 email = request.POST.get('email')
        message = request.POST.get('message')
        FeedBack.objects.create(name=name, email=email, message=message)
        return HttpResponse('Thank you for your feedback! <a href="{}">Return to Home page</a>'.format(reverse('main')))

    return render(request,'weatherapp/contact.html')

def read_json_file(filename):
    try:
        with open(filename, 'r') as json_file:
            data = json.load(json_file)
    except FileNotFoundError:
        data = {}
    return data
```

urls.py

```
from django.urls import path
from . import views
urlpatterns = [

    path('',views.home,name = 'home'),
```

```
    path('main',views.main,name = 'main'),
    path('weatherdata',views.weatherdata,name = 'weatherdata'),

 path('airdata',views.airdata,name = 'airdata'),
    path('contact',views.contact,name = 'contact')
]
```

Models.py

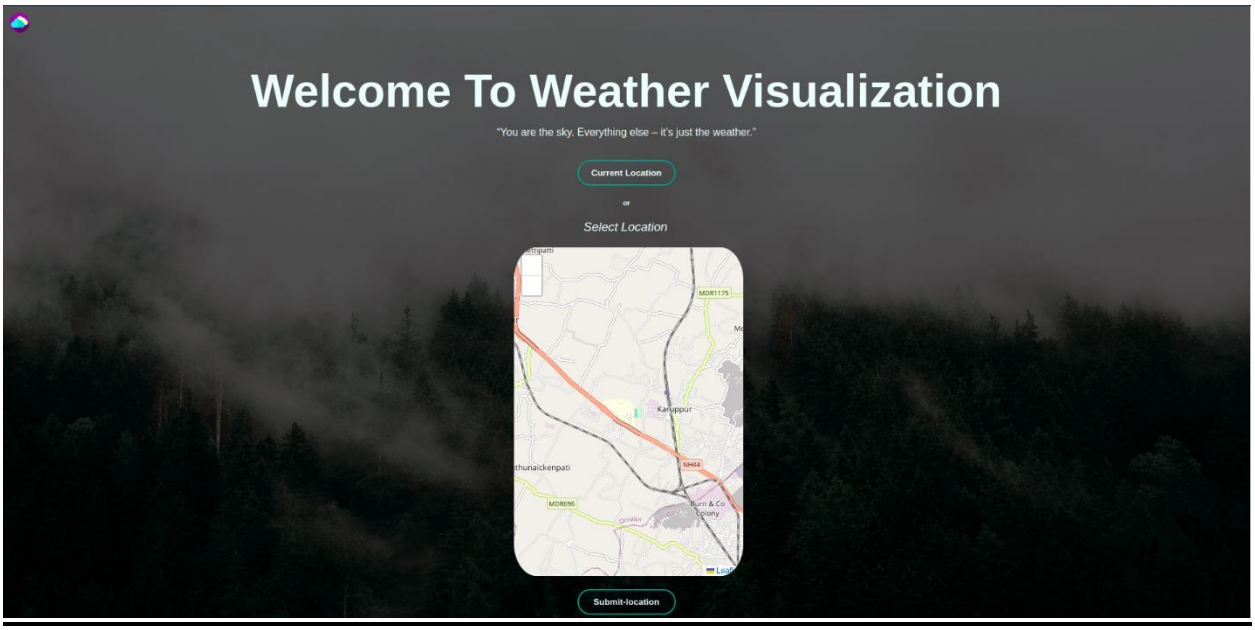
```
from django.db import models
class FeedBack(models.Model):
   name = models.CharField(max_length=100,null=True)
   email = models.EmailField()
   message = models.TextField()
   created_at = models.DateTimeField(auto_now_add=True)

   def __str__(self):
       return self.name + ' - ' + self.email
```

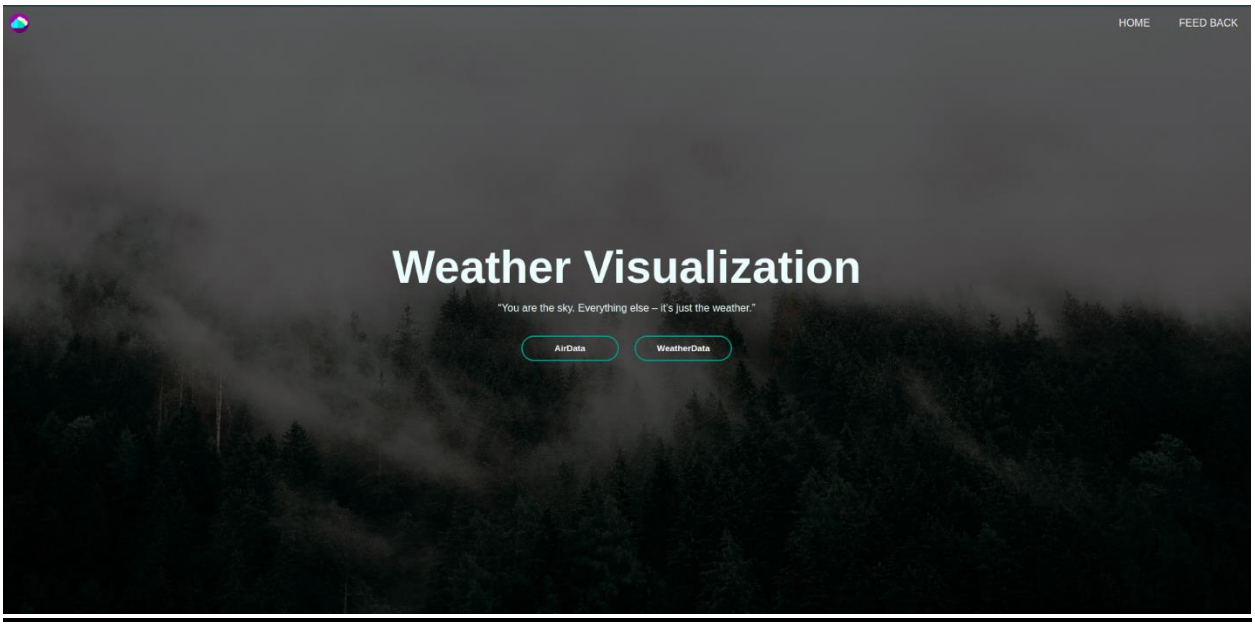
CHAPTER 7

RESULT

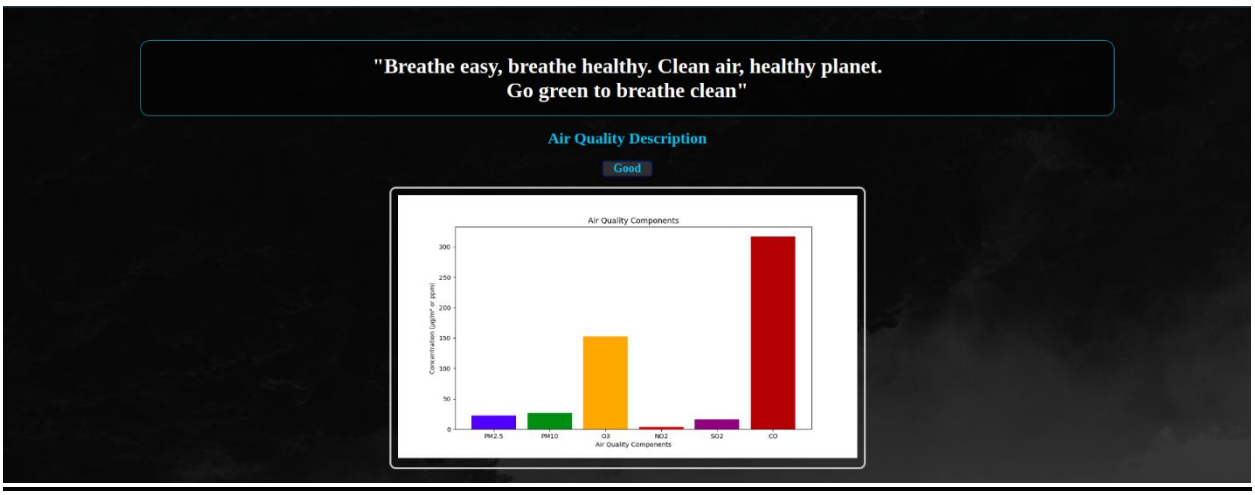
7.1 HOME PAGE:



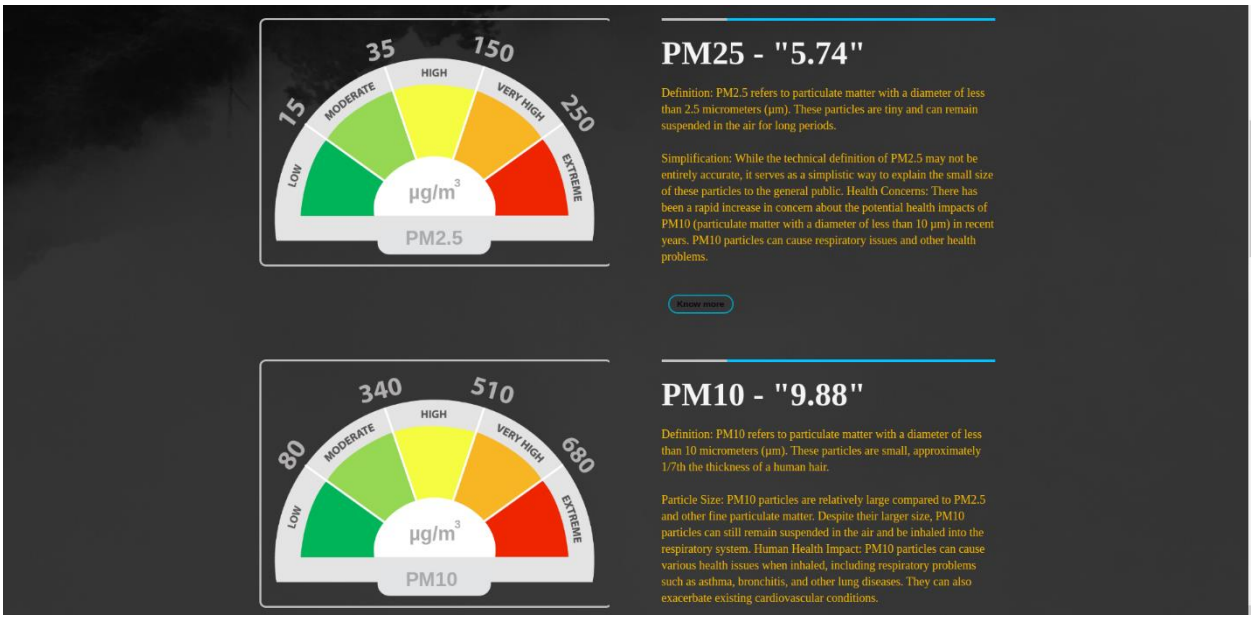
7.2 LANDING PAGE:



7.3 AirData:

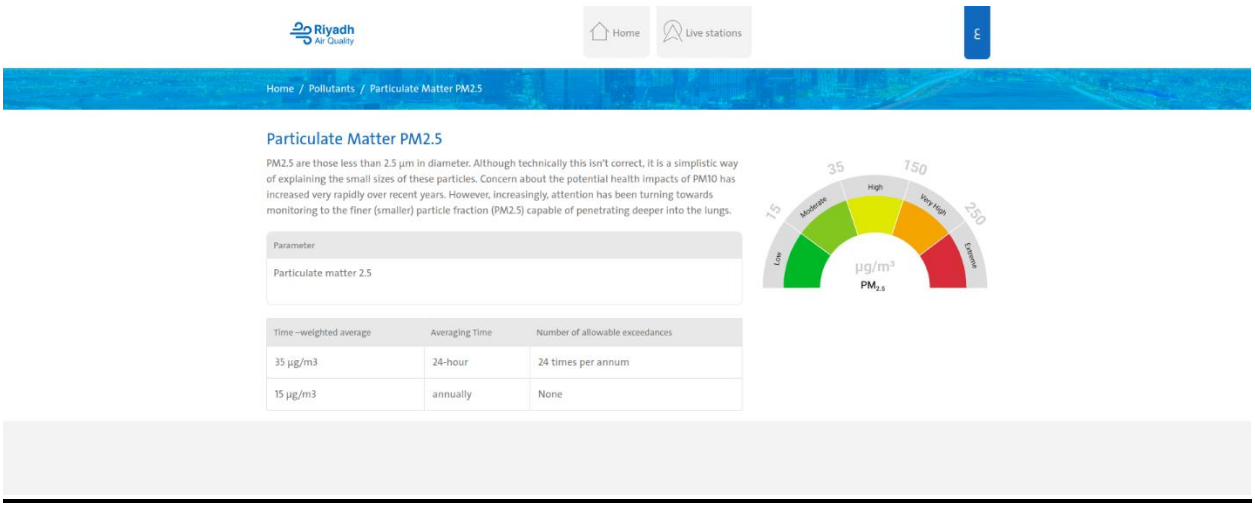


Continue,,,

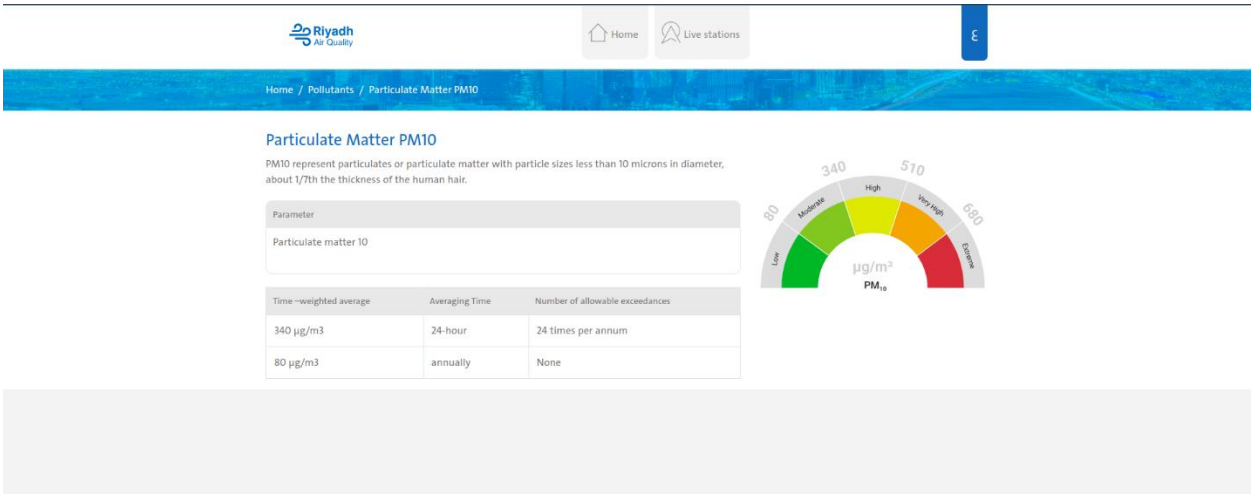


7.3.1 Know More:

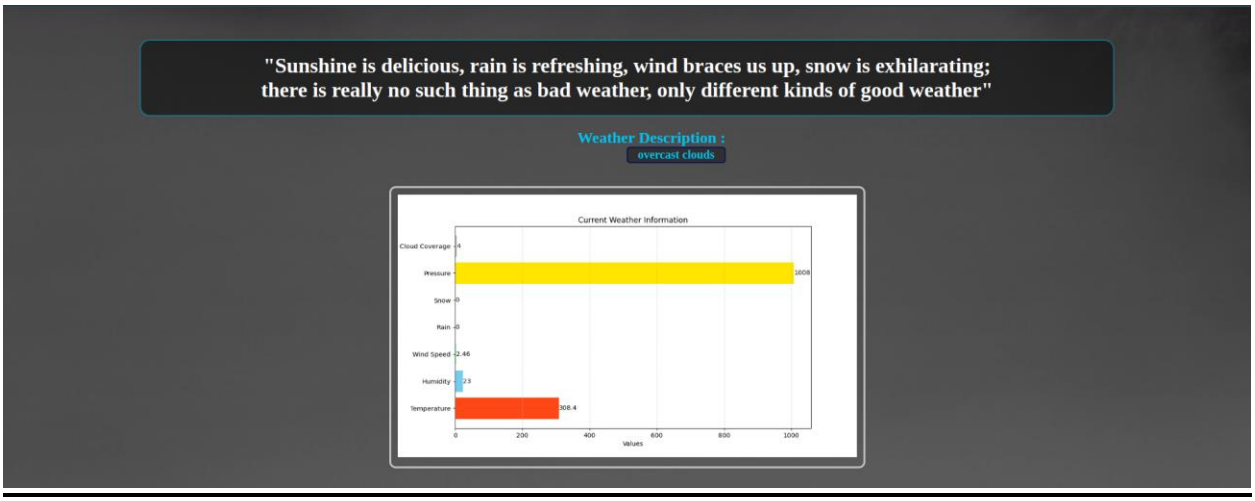
PM2.5



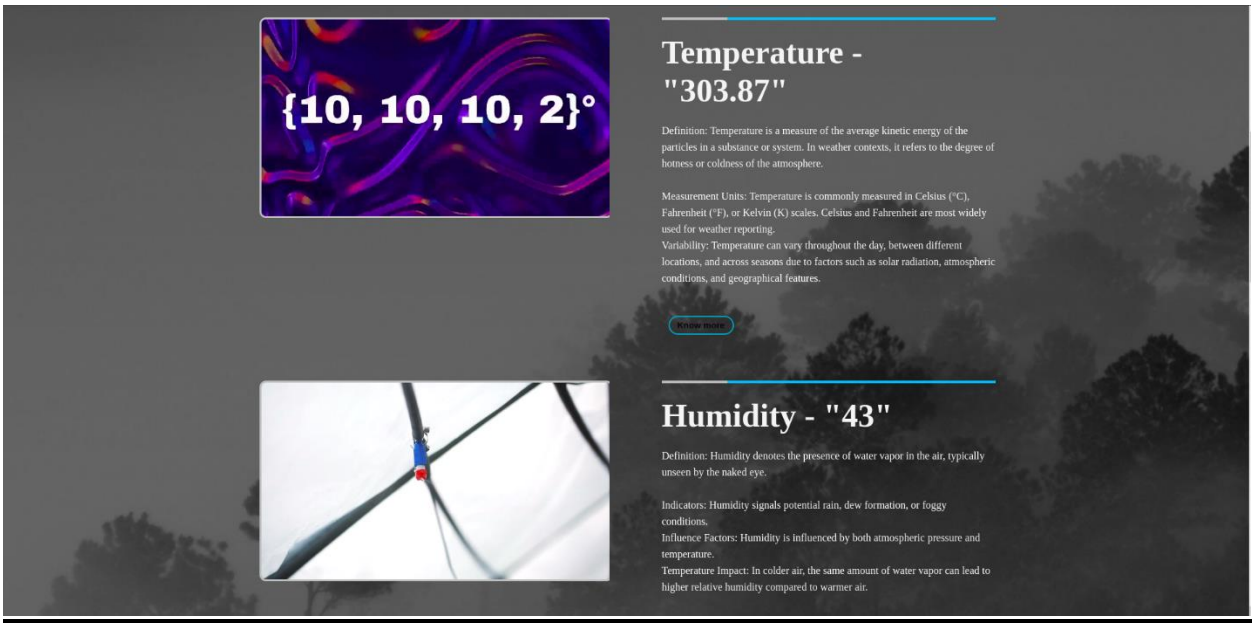
PM10



7.4 WeatherData:

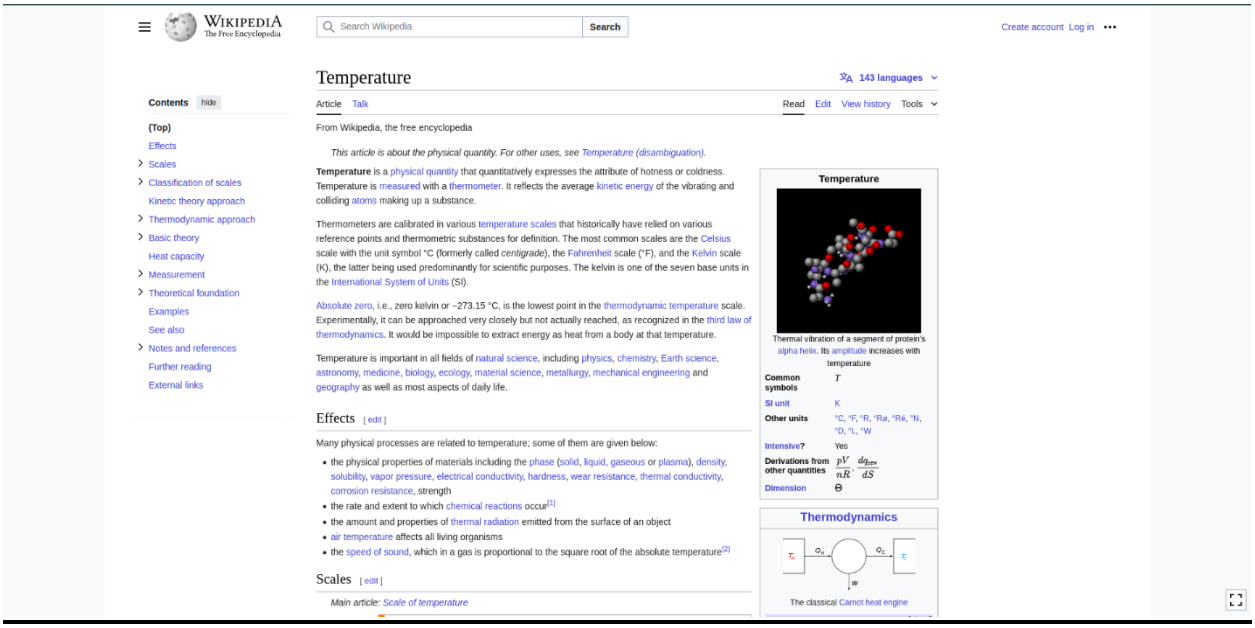


Continue,,,



7.4.1 Know More:

Temperature



Temperature

From Wikipedia, the free encyclopedia

This article is about the physical quantity. For other uses, see Temperature (disambiguation).

Temperature is a physical quantity that quantitatively expresses the attribute of hotness or coldness. Temperature is measured with a thermometer. It reflects the average kinetic energy of the vibrating and colliding atoms making up a substance.

Thermometers are calibrated in various temperature scales that historically have relied on various reference points and thermometric substances for definition. The most common scales are the Celsius scale with the unit symbol °C (formerly called *centigrade*), the Fahrenheit scale (°F), and the Kelvin scale (K), the latter being used predominantly for scientific purposes. The kelvin is one of the seven base units in the International System of Units (SI).

Absolute zero, i.e., zero kelvin or −273.15 °C, is the lowest point in the thermodynamic temperature scale. Experimentally, it can be approached very closely but not actually reached, as recognized in the third law of thermodynamics. It would be impossible to extract energy as heat from a body at that temperature.

Temperature is important in all fields of natural science, including physics, chemistry, Earth science, astronomy, medicine, biology, ecology, material science, metallurgy, mechanical engineering and geography as well as most aspects of daily life.

Temperature

Thermal vibration of a segment of protein's alpha helix. Its amplitude increases with temperature

Common symbols	T
SI unit	K
Other units	°C, °F, °R, °Ré, °N, °D, °L, °W
Intensive?	Yes
Derivations from other quantities	$\frac{pV}{nR}$, $\frac{dE_{kin}}{dS}$
Dimension	Θ

Thermodynamics

The classical Carnot heat engine

Effects

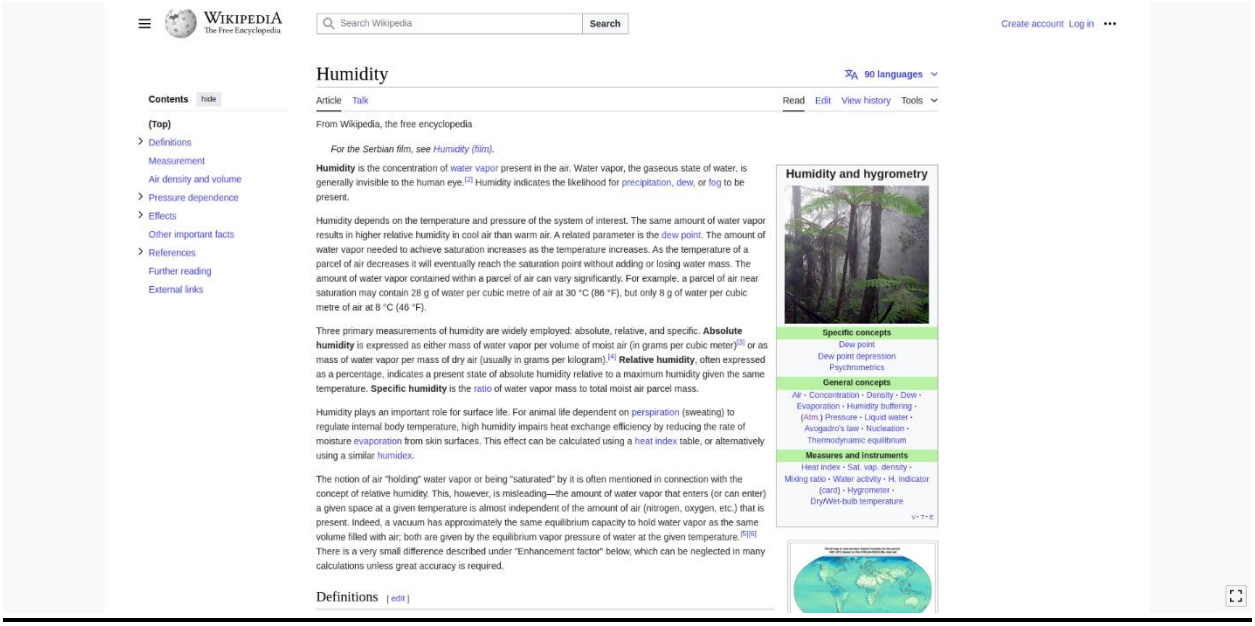
Many physical processes are related to temperature; some of them are given below:

- the physical properties of materials including the phase (solid, liquid, gaseous or plasma), density, solubility, vapor pressure, electrical conductivity, hardness, wear resistance, thermal conductivity, corrosion resistance, strength
- the rate and extent to which chemical reactions occur[1]
- the amount and properties of thermal radiation emitted from the surface of an object
- air temperature affects all living organisms
- the speed of sound, which in a gas is proportional to the square root of the absolute temperature[2]

Scales

Main article: Scale of temperature

Humidity



Humidity

From Wikipedia, the free encyclopedia

For the Serbian film, see Humidity (film).

Humidity is the concentration of water vapor present in the air. Water vapor, the gaseous state of water, is generally invisible to the human eye.[1] Humidity indicates the likelihood for precipitation, dew, or fog to be present.

Humidity depends on the temperature and pressure of the system of interest. The same amount of water vapor results in higher relative humidity in cool air than warm air. A related parameter is the dew point. The amount of water vapor needed to achieve saturation increases as the temperature increases. As the temperature of a parcel of air decreases it will eventually reach the saturation point without adding or losing water mass. The amount of water vapor contained within a parcel of air can vary significantly. For example, a parcel of air near saturation may contain 28 g of water per cubic metre of air at 30 °C (86 °F), but only 8 g of water per cubic metre of air at 8 °C (46 °F).

Three primary measurements of humidity are widely employed: absolute, relative, and specific. **Absolute humidity** is expressed as either mass of water vapor per volume of moist air (in grams per cubic meter)[3] or as mass of water vapor per mass of dry air (usually in grams per kilogram).[4] **Relative humidity**, often expressed as a percentage, indicates a present state of absolute humidity relative to a maximum humidity given the same temperature. **Specific humidity** is the ratio of water vapor mass to total moist air parcel mass.

Humidity plays an important role for surface life. For animal life dependent on perspiration (sweating) to regulate internal body temperature, high humidity impairs heat exchange efficiency by reducing the rate of moisture evaporation from skin surfaces. This effect can be calculated using a heat index table, or alternatively using a similar humidex.

The notion of air "holding" water vapor or being "saturated" by it is often mentioned in connection with the concept of relative humidity. This, however, is misleading—the amount of water vapor that enters (or can enter) a given space at a given temperature is almost independent of the amount of air (nitrogen, oxygen, etc.) that is present. Indeed, a vacuum has approximately the same equilibrium capacity to hold water vapor as the same volume filled with air; both are given by the equilibrium vapor pressure of water at the given temperature.[5][6] There is a very small difference described under "Enhancement factor" below, which can be neglected in many calculations unless great accuracy is required.

Humidity and hygrometry

Specific concepts
Dew point
Dew point depression
Psychrometrics

General concepts
Air · Concentration · Density · Dew · Evaporation · Humidity buffering · (Atm.) Pressure · Liquid water · Avogadro's law · Nucleation · Thermodynamic equilibrium

Measures and instruments
Heat index · Sat. vap. density · Mixing ratio · Water activity · H. Indicator (card) · Hygrometer · Dry/Wet-bulb temperature

Definitions

CHAPTER 8

CONCLUSION

The weather and air quality data visualization system introduced in this project offers a significant advancement in environmental monitoring. Leveraging real-time data and advanced visualization techniques, the system provides users with comprehensive insights into atmospheric conditions.

Through the integration of APIs for data retrieval and Python for backend processing, alongside HTML and CSS for frontend development, the system delivers a user-friendly experience. Its adaptability and scalability make it suitable for various applications, from public health monitoring to urban planning.

By empowering users with actionable insights, this system plays a crucial role in navigating environmental challenges. Its contribution towards fostering sustainability and resilience underscores the importance of technological innovation in addressing complex environmental issues.

REFERENCES

- **"OpenWeatherMap API Documentation."** OpenWeatherMap. https://openweathermap.org/api

- **"Requests: HTTP for Humans."** Requests Documentation. https://docs.python-requests.org/en/latest/

- **"Matplotlib Documentation."** Matplotlib. https://matplotlib.org/stable/contents.html

- **"Django Documentation."** Django. https://docs.djangoproject.com/en/stable/

- **"Python.org."** The Python Standard Library. https://docs.python.org/3/library/index.html