



Project Report
on
WeatherWiseHub

Submitted in partial fulfillment of the requirement for the award of degree of

Master of Computer Application (MCA)

at

University of Mumbai

Submitted by

Pawan Srivastava

Under the guidance of

Prof. Sudeshna Roy



Bharati Vidyapeeth's
Institute of Management and Information Technology
Sector 8, CBD Belapur
Navi Mumbai
2023 - 2025



Bharati Vidyapeeth's
Institute of Management and Information Technology
Navi Mumbai

Certificate of Approval

This is to certify that the Project titled ' **WeatherWiseHub** ' is successfully done by **Pawan Srivastava** during Sem-1 of his course in partial fulfillment of Masters of Computer Application under the University of Mumbai, Mumbai, through the Bharati Vidyapeeths Institute of Management and Information Technology, Navi Mumbai carried out by her under our guidance and supervision.

Sign & Date

Guide

External Examiner

Signature and Date

Principal

Dr.Suhasini Vijaykumar

College Seal

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

(Pawan Srivastava)

Date

Acknowledgement

I have a great pleasure to express my gratitude to all those who have contributed and motivated during my project work. Here you have a liberty to write anything and express your feeling to all those who have helped you.

...

Date:

Pawan Srivastava

Abstract

WeatherWiseHub, an innovative desktop application, leverages the power of Python and Flask to offer users a comprehensive weather and entertainment experience. Focused on delivering real-time weather updates, air quality indices, and curated Spotify playlists, WeatherWiseHub stands as a versatile and user-friendly solution.

Unlike conventional desktop applications, WeatherWiseHub simplifies the process of accessing weather forecasts, air quality data, and entertainment recommendations. Through seamless integration with OpenWeatherMap and Spotify APIs, users can receive up-to-date weather information for their chosen location and discover playlists tailored to current weather conditions.

WeatherWiseHub's design prioritizes user interaction, making it easy for individuals to input their preferred city, receive detailed weather insights, and explore suggested playlists. The application serves as a practical tool for daily weather monitoring and entertainment discovery.

In summary, WeatherWiseHub offers a streamlined and accessible approach to weather information and entertainment recommendations, without the complexities associated with AI-based interactions. This desktop application caters to users seeking a straightforward and functional solution for their weather-related queries and music preferences.

Keywords: *Python, Flask, OpenWeatherMap API, Spotify API, Desktop Application.*

Contents

Abstract	5
List of Figures	8
Nomenclature	9
1 Introduction	1
1.1 Introduction to Project	1
1.2 Problem Description	1
2 System Study	2
2.1 Existing system	2
2.2 Limitations	2
2.3 Proposed system	2
2.4 Objectives	3
2.5 Feasibility Studies	3
2.5.1 Economical Feasibility	3
2.5.2 Technical Feasibility	4
2.5.3 Operational Feasibility	4
3 System Analysis	5
3.1 Gantt chart	5
3.2 DFDs	5
3.3 Operating tools and technology	6
4 System Design	8
4.1 ER Diagram	8
4.2 Activity Diagram	9
4.3 Input	10

5	Code	12
6	System Implementation	15
7	System Testing	20
7.1	Testing Methodologies	20
7.2	Test Cases	20
8	Limitation and Future Enhancement	21
	References	22

List of Figures

3.1	Gantt Chart for the WeatherWiseHub	5
3.2	DFD for the WeatherWiseHub	5
4.1	ER for the WeatherWiseHub	8
4.2	Activity Diagram for the WeatherWiseHub	9
4.3	Input Diagram for the WeatherWiseHub	10
5.1	Code for the WeatherWiseHub	12
5.2	Code for the WeatherWiseHub	13
5.3	Code for the WeatherWiseHub	13
5.4	Code for the WeatherWiseHub	14
5.5	Code for the WeatherWiseHub	14
6.1	Output for the WeatherWiseHub	15
6.2	Output for the WeatherWiseHub	16
6.3	Output for the WeatherWiseHub	16
6.4	Output for the WeatherWiseHub	17
6.5	Output for the WeatherWiseHub	17
6.6	Output for the WeatherWiseHub	18
6.7	Output for the WeatherWiseHub	18
6.8	Output for the WeatherWiseHub	19
6.9	Output for the WeatherWiseHub	19

Nomenclature

AQI	AIR QUALITY INDEX
DEF	DEFINING FUNCTION
API	APPLICATION PROGRAMMING INTERFACE

Chapter 1

Introduction

1.1 Introduction to Project

As an avant-garde weather application, WeatherWiseHub seeks to redefine user expectations by seamlessly integrating weather data, air quality information, and personalized musical experiences. The amalgamation of technical precision, conversational interaction, and personalization underscores its commitment to providing a holistic and delightful user journey. WeatherWiseHub is not merely a weather app; it is a comprehensive weather companion designed to elevate and enrich the daily experience of its users.

1.2 Problem Description

Traditional weather applications often lack the personal touch needed to engage users actively. Users typically seek more than just numerical data; they desire an immersive and customized experience that aligns with their preferences. WeatherWiseHub addresses this gap by harmoniously blending weather data with curated music suggestions, resulting in a unique and compelling user experience. It aims to overcome the limitations of conventional weather apps and enrich the user journey with dynamic content and personalized interactions.

Chapter 2

System Study

This chapter should consist of the information regarding already available solutions/methods of whatever you are proposing.

Another paragraph should explain what are the problems with the solution/methods proposed by others. These methods should be properly supported by proper reference.

2.1 Existing system

The existing landscape of weather applications primarily revolves around delivering factual weather data. Traditional weather apps often lack the sophistication needed to engage users beyond providing numerical information. Users are typically presented with straightforward weather updates, missing out on the potential for a more immersive and personalized experience.

2.2 Limitations

Lack of Personalization: Conventional weather applications often fail to personalize the user experience, providing generic weather updates without considering individual preferences.

Monotonous Interaction: The interaction model of existing weather apps is typically monotonous, offering limited engagement beyond displaying weather statistics.

Isolated Information: Weather and air quality data are usually presented in isolation, without a seamless integration of supplementary features to enhance user engagement.

Limited Forecasting: While many weather apps provide basic forecasting, the depth and accuracy of these predictions are often constrained.

2.3 Proposed system

WeatherWiseHub represents a paradigm shift from conventional weather applications by introducing a dynamic and user-centric approach. The proposed system addresses the limitations of existing weather apps and aims to redefine the user experience by seamlessly integrating weather data, air quality information, and personalized musical recommendations.

2.4 Objectives

1. Weather Information:

Real-time Updates: WeatherWiseHub harnesses the OpenWeatherMap API to provide users with accurate and up-to-the-minute weather information. The integration ensures that users receive relevant and timely updates, enhancing their ability to plan activities based on current weather conditions.

2. Music Recommendations:

Spotify Integration: Going beyond the traditional weather app, WeatherWiseHub integrates with the Spotify API to curate personalized playlists based on the current weather. This innovative feature transforms the app into a multimedia experience, connecting users with music that complements the atmospheric conditions.

3. User Interaction:

Conversational Interface: WeatherWiseHub adopts a conversational approach, allowing users to interact with the app in a natural and intuitive manner. The implementation of natural language processing (NLP) ensures that the app understands user queries and responds in a manner that mimics human conversation.

4. Weather Forecasting:

Forecast Accuracy: WeatherWiseHub enhances user preparedness by providing accurate weather forecasts. Leveraging advanced forecasting techniques and APIs, the app delivers detailed insights into future weather conditions, enabling users to plan ahead effectively.

5. Air Quality Information:

AQI Integration: Recognizing the importance of environmental well-being, WeatherWiseHub incorporates air quality data into its comprehensive weather insights. By integrating Air Quality Index (AQI) information, users gain a holistic view of their surroundings, promoting health-conscious decision-making.

6. Error Handling and Learning:

Robust Error Handling: WeatherWiseHub incorporates robust error-handling mechanisms to interpret ambiguous queries intelligently. By learning from user interactions, the app continuously improves its ability to understand and respond appropriately, enhancing the overall user experience.

2.5 Feasibility Studies

2.5.1 Economical Feasibility

The project is economically feasible, with costs associated with development offset by the benefits of improved learning experiences.

2.5.2 Technical Feasibility

The project is technically feasible, leveraging common web development technologies for effective implementation.

2.5.3 Operational Feasibility

The proposed system is operationally feasible, as it aligns with modern learning practices and can be smoothly integrated into educational workflows.

Chapter 3

System Analysis

3.1 Gantt chart

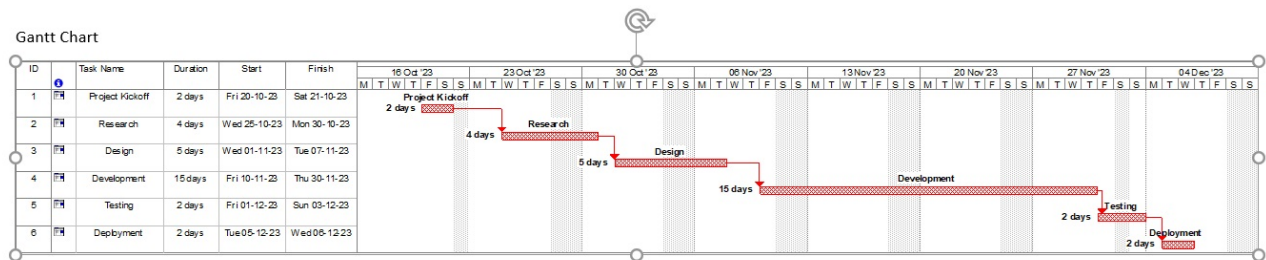


Figure 3.1: Gantt Chart for the WeatherWiseHub

3.2 DFDs

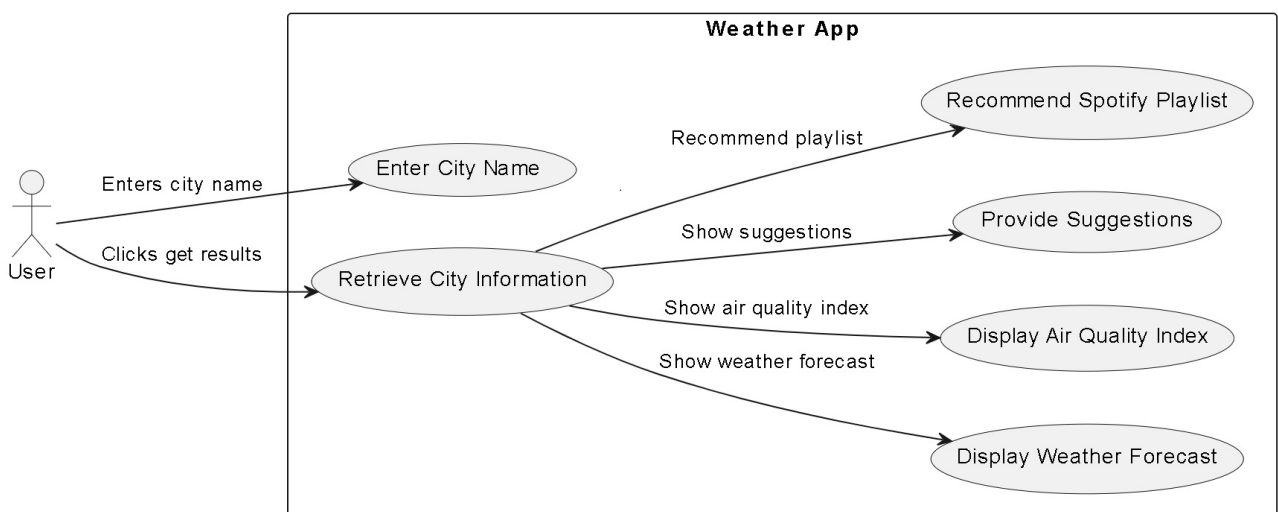


Figure 3.2: DFD for the WeatherWiseHub

A DFD is a graphical representation of the flow of data through an information system, and it illustrates how data is processed by a system in terms of inputs and outputs.

Here's a breakdown of the DFD:

1. User: This is the external entity that interacts with the system. The user performs two actions: - Enters city name: The user inputs the name of a city for which they want to retrieve weather information. - Clicks get results: After entering the city name, the user initiates the process to get the results by clicking a button or similar UI element.

2. Enter City Name: This is the first process within the system. It represents the system's action of accepting the city name entered by the user.

3. Retrieve City Information: This is the second process that takes place after the city name is entered. The system retrieves relevant information about the city, which is likely to include weather data and possibly other related information.

4. Weather App: This is the overall system or application that processes the user's request and provides the output. Within the Weather App, there are three processes: - Show suggestions: This process might involve providing the user with suggestions based on the city name entered, such as nearby locations or commonly searched cities. - Show air quality index: This process involves displaying the air quality index for the specified city, which is a measure of the city's air pollution levels. - Show weather forecast: This process involves displaying the weather forecast for the city, which includes information like temperature, precipitation, wind speed, etc.

5. Outputs: There are three outputs from the Weather App: - Recommend Spotify Playlist: Based on the weather data, the app might recommend a Spotify playlist that fits the current weather conditions. - Provide Suggestions: The app provides suggestions to the user, as mentioned earlier. - Display Air Quality Index: The app displays the air quality index for the city. - Display Weather Forecast: The app displays the weather forecast for the city.

The arrows in the diagram indicate the direction of data flow between the user, processes, and outputs. This DFD provides a high-level overview of the main functions and interactions within the Weather App without going into the details of the internal workings or data storage.

3.3 Operating tools and technology

.Hardware Requirements: WeatherWiseHub demands a robust hardware infrastructure to guarantee optimal performance. This encompasses servers for hosting, high-performance processors, substantial RAM, and ample storage to accommodate weather data, user profiles, and system logs.

.Software Requirements: The software stack encompasses diverse components: - Flask Framework: Integral for constructing the web application and managing backend operations. - Python Programming Language: The principal language for coding the application logic. - OpenWeatherMap API: Integrated to fetch real-time weather data. - Spotify API: Employed to retrieve and recommend personalized playlists.

.Development Tools: - Microsoft Project: Facilitates project management, scheduling, and Gantt chart creation. - Visual Studio Code: A robust source code editor for writing and debugging application code. - Git Version Control: Enables collaborative coding, version tracking, and code management. - Postman: Facilitates API testing and validation during development.

Chapter 4

System Design

4.1 ER Diagram

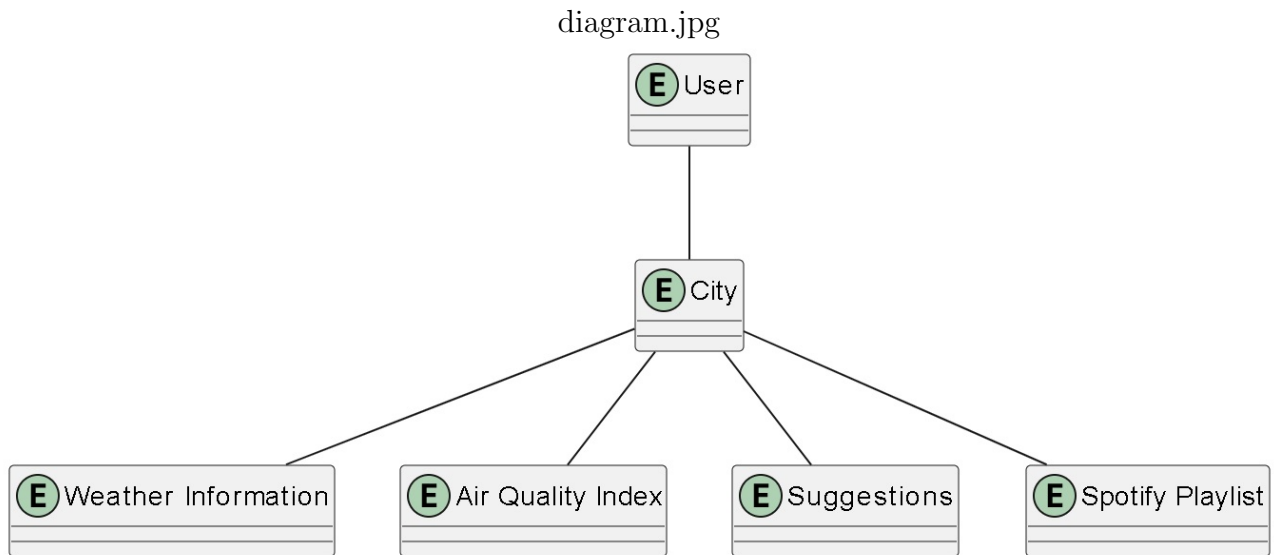


Figure 4.1: ER for the WeatherWiseHub

The ER diagram described represents the structure of a weather forecasting website that uses an API. Here's a breakdown: **User**: This is the starting point of the interaction. The user is likely to input a city name into the website's search function. This **is done** through a search bar **present** on the website's design.

City: This is the entity that the user interacts with. It's not just a simple string of text, but a complex entity that has relationships with other entities. When the user searches for a city, the website fetches data related to that city from its database or an external API.

Weather Info: This is one of the four components linked to the 'City' entity. It likely includes data such as temperature, humidity, wind speed, and forecast for the next few days. This data is probably fetched from a weather API using the city name as a parameter.

AQI (Air Quality Index): This is another component linked to the 'City'. It provides information about the air quality in the city, which can be crucial for users with respiratory

conditions or for general awareness. This data might also be fetched from an environmental API.

AQI-related suggestions: These are recommendations provided to the user based on the AQI. For example, if the AQI is poor, the website might suggest staying indoors or wearing a mask.

Spotify Playlist: This is an interesting feature. The website recommends a Spotify playlist based on the city. The criteria for these recommendations could be based on the city's culture, weather, or popular music trends in that city.

In summary, the ER diagram represents a complex system where a user's input (city name) triggers a series of data fetches and computations, resulting in a rich set of information and recommendations being displayed to the user.

4.2 Activity Diagram

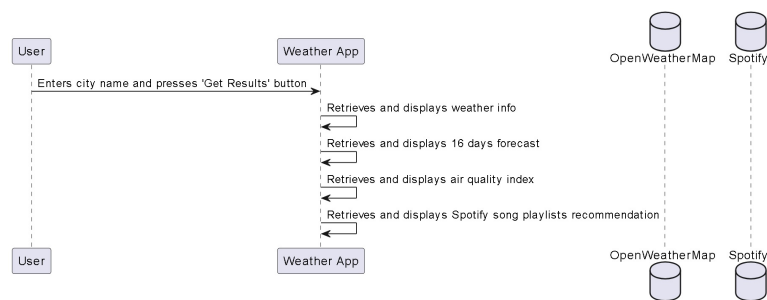


Figure 4.2: Activity Diagram for the WeatherWiseHub

Activity diagram Here's what the diagram is illustrating:

1. A user interacts with a Weather App.
2. The user enters a city name and presses the "Get Results" button.
3. The Weather App then performs a series of actions:
 - It retrieves and displays weather information.
 - It retrieves and displays a 16-day weather forecast.
 - It retrieves and displays the air quality index.
 - It retrieves and displays Spotify song playlist recommendations.
4. To provide this information, the Weather App communicates with two external services:
 - OpenWeatherMap, which is likely the source of the weather information, forecast, and air quality index.
 - Spotify, which is the source of the music playlist recommendations.

The arrows indicate the flow of communication between the user, the Weather App, and the external services. Solid lines with filled arrowheads represent synchronous calls where the Weather App waits for a response. The dashed lines with open arrowheads represent return messages back to the Weather App, indicating that the requested data has been retrieved and is being sent back to the app for display to the user.

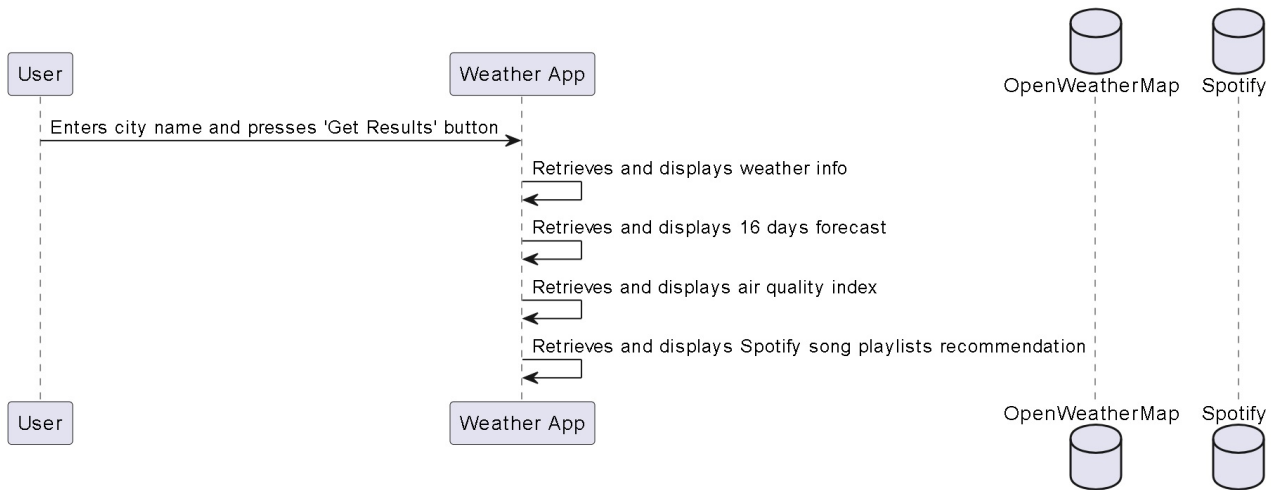


Figure 4.3: Input Diagram for the WeatherWiseHub

4.3 Input

The diagram described as to be a input diagram, which is a type of interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message input chart. This particular input diagram outlines the interactions between a user, a weather application, and external services (OpenWeatherMap and Spotify) to provide various pieces of information based on a city name input by the user.

Here's a step-by-step explanation of the diagram:

1. The user interacts with the Weather App by entering a city name and pressing the 'Get Results' button.
2. The Weather App then performs a series of actions, each involving a request to an external service and then providing the retrieved information back to the user. These actions are as follows:
 - The app retrieves and displays weather information from OpenWeatherMap, which likely includes current weather conditions.
 - It retrieves and displays a 16-day weather forecast, also from OpenWeatherMap, giving the user an extended outlook on the weather.
 - The app retrieves and displays the air quality index, which is an assessment of the air quality based on pollutants and provides this information to the user. This data is also sourced from OpenWeatherMap.
 - Lastly, the app retrieves and displays Spotify song playlists recommendations. This suggests that the app might offer music playlists from Spotify that are suitable for the current weather conditions or the forecasted weather.

Each of these actions involves a back-and-forth communication between the Weather App and the respective external service. The Weather App sends a request, and the service responds with the requested data.

The diagram uses solid lines with filled arrowheads to represent synchronous calls (where the caller waits for the response) and dashed lines with open arrowheads to represent return

messages (the data being sent back to the Weather App).

Overall, the diagram is a high-level representation of the flow of data and control between the user, the Weather App, and the external services it interacts with.

Code

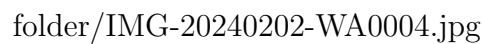
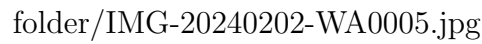
The screenshot shows a VS Code editor window with a Python file named `app.py` open. The Explorer sidebar on the left displays the project structure, including folders like `static`, `templates`, and `env`, and files like `air_quality_data.json`, `app.py`, and `spotify_integration.py`. The `app.py` file is currently selected and open in the editor. The code in `app.py` includes imports for `datetime`, `json`, `requests`, and `os`, and defines a function to fetch weather data from an API and save it to a file. The code is as follows:

```

1 from datetime import datetime
2 from datetime import datetime
3 from datetime import datetime
4 from datetime import datetime
5 from datetime import datetime
6 from datetime import datetime
7 from datetime import datetime
8 from datetime import datetime
9 from datetime import datetime
10 from datetime import datetime
11 from datetime import datetime
12 from datetime import datetime
13 from datetime import datetime
14 from datetime import datetime
15 from datetime import datetime
16 from datetime import datetime
17 from datetime import datetime
18 from datetime import datetime
19 from datetime import datetime
20 from datetime import datetime
21 from datetime import datetime
22 from datetime import datetime
23 from datetime import datetime
24 from datetime import datetime
25 from datetime import datetime
26 from datetime import datetime
27 from datetime import datetime
28 from datetime import datetime
29 from datetime import datetime
30 from datetime import datetime
31 from datetime import datetime
32 from datetime import datetime
33 from datetime import datetime
34 from datetime import datetime
35 from datetime import datetime
36 from datetime import datetime
37 from datetime import datetime
38 from datetime import datetime
39 from datetime import datetime
40 from datetime import datetime
41 from datetime import datetime
42 from datetime import datetime
43 from datetime import datetime
44 from datetime import datetime
45 from datetime import datetime
46 from datetime import datetime
47 from datetime import datetime
48 from datetime import datetime
49 from datetime import datetime
50 from datetime import datetime
51 from datetime import datetime
52 from datetime import datetime
53 from datetime import datetime
54 from datetime import datetime
55 from datetime import datetime
56 from datetime import datetime
57 from datetime import datetime
58 from datetime import datetime
59 from datetime import datetime
60 from datetime import datetime
61 from datetime import datetime
62 from datetime import datetime
63 from datetime import datetime
64 from datetime import datetime
65 from datetime import datetime
66 from datetime import datetime
67 from datetime import datetime
68 from datetime import datetime
69 from datetime import datetime
70 from datetime import datetime
71 from datetime import datetime
72 from datetime import datetime
73 from datetime import datetime
74 from datetime import datetime
75 from datetime import datetime
76 from datetime import datetime
77 from datetime import datetime
78 from datetime import datetime
79 from datetime import datetime
80 from datetime import datetime
81 from datetime import datetime
82 from datetime import datetime
83 from datetime import datetime
84 from datetime import datetime
85 from datetime import datetime
86 from datetime import datetime
87 from datetime import datetime
88 from datetime import datetime
89 from datetime import datetime
90 from datetime import datetime
91 from datetime import datetime
92 from datetime import datetime
93 from datetime import datetime
94 from datetime import datetime
95 from datetime import datetime
96 from datetime import datetime
97 from datetime import datetime
98 from datetime import datetime
99 from datetime import datetime
100 from datetime import datetime

```

Figure 5.1: Code for the WeatherWiseHub

[illegible]

```

171 def get_air_quality(latitude, longitude):
172     base_url = "https://api.openweathermap.org/data/2.5/air_pollution"
173     params = {
174         "lat": latitude,
175         "lon": longitude,
176         "appid": APP_WEATHERMAP_API_KEY
177     }
178     response = requests.get(base_url, params=params)
179     data = response.json()
180
181     if response.status_code == 200:
182         air_data = {}
183         air_data["city"] = "New York"
184         air_data["date"] = datetime.now().strftime("%Y-%m-%d")
185         air_data["components"] = data["components"]
186         return air_data
187     else:
188         return None
189
190 def get_health_recommendations(latitude, longitude):
191     base_url = "https://api.openweathermap.org/data/2.5/health"
192     params = {
193         "lat": latitude,
194         "lon": longitude,
195         "appid": APP_WEATHERMAP_API_KEY
196     }
197     response = requests.get(base_url, params=params)
198     data = response.json()
199
200     if response.status_code == 200:
201         health_data = {}
202         health_data["city"] = "New York"
203         health_data["date"] = datetime.now().strftime("%Y-%m-%d")
204         health_data["recommendations"] = data["recommendations"]
205         return health_data
206     else:
207         return None
208
209 def get_weather(latitude, longitude):
210     base_url = "https://api.openweathermap.org/data/2.5/weather"
211     params = {
212         "lat": latitude,
213         "lon": longitude,
214         "appid": APP_WEATHERMAP_API_KEY,
215         "units": "metric"
216     }
217     response = requests.get(base_url, params=params)
218     data = response.json()
219
220     if response.status_code == 200:
221         weather_data = {}
222         weather_data["city"] = "New York"
223         weather_data["date"] = datetime.now().strftime("%Y-%m-%d")
224         weather_data["weather"] = data["weather"]
225         weather_data["temp"] = data["temp"]
226         weather_data["humidity"] = data["humidity"]
227         weather_data["wind"] = data["wind"]
228         weather_data["visibility"] = data["visibility"]
229         return weather_data
230     else:
231         return None
232
233 def main():
234     city = "New York"
235     coordinates = get_coordinates(city)
236
237     if coordinates:
238         latitude, longitude = coordinates
239         air_data = get_air_quality(latitude, longitude)
240         health_data = get_health_recommendations(latitude, longitude)
241         weather_data = get_weather(latitude, longitude)
242
243         render_template("index.html", city=city, air_data=air_data, health_data=health_data, weather_data=weather_data)
244     else:
245         return None
246
247 if __name__ == "__main__":
248     main()

```

Figure 5.4: Code for the WeatherWiseHub

```

171 def get_coordinates(city):
172     base_url = "https://api.openweathermap.org/data/2.5/geoip3"
173     params = {
174         "q": city,
175         "appid": APP_WEATHERMAP_API_KEY
176     }
177     response = requests.get(base_url, params=params)
178     data = response.json()
179
180     if response.status_code == 200:
181         coordinates = data["coordinates"]
182         return coordinates
183     else:
184         return None
185
186 def get_weather(latitude, longitude):
187     base_url = "https://api.openweathermap.org/data/2.5/weather"
188     params = {
189         "lat": latitude,
190         "lon": longitude,
191         "appid": APP_WEATHERMAP_API_KEY,
192         "units": "metric"
193     }
194     response = requests.get(base_url, params=params)
195     data = response.json()
196
197     if response.status_code == 200:
198         weather_data = {}
199         weather_data["city"] = "New York"
200         weather_data["date"] = datetime.now().strftime("%Y-%m-%d")
201         weather_data["weather"] = data["weather"]
202         weather_data["temp"] = data["temp"]
203         weather_data["humidity"] = data["humidity"]
204         weather_data["wind"] = data["wind"]
205         weather_data["visibility"] = data["visibility"]
206         return weather_data
207     else:
208         return None
209
210 def get_health_recommendations(latitude, longitude):
211     base_url = "https://api.openweathermap.org/data/2.5/health"
212     params = {
213         "lat": latitude,
214         "lon": longitude,
215         "appid": APP_WEATHERMAP_API_KEY
216     }
217     response = requests.get(base_url, params=params)
218     data = response.json()
219
220     if response.status_code == 200:
221         health_data = {}
222         health_data["city"] = "New York"
223         health_data["date"] = datetime.now().strftime("%Y-%m-%d")
224         health_data["recommendations"] = data["recommendations"]
225         return health_data
226     else:
227         return None
228
229 def get_air_quality(latitude, longitude):
230     base_url = "https://api.openweathermap.org/data/2.5/air_pollution"
231     params = {
232         "lat": latitude,
233         "lon": longitude,
234         "appid": APP_WEATHERMAP_API_KEY
235     }
236     response = requests.get(base_url, params=params)
237     data = response.json()
238
239     if response.status_code == 200:
240         air_data = {}
241         air_data["city"] = "New York"
242         air_data["date"] = datetime.now().strftime("%Y-%m-%d")
243         air_data["components"] = data["components"]
244         return air_data
245     else:
246         return None
247
248 def main():
249     city = "New York"
250     coordinates = get_coordinates(city)
251
252     if coordinates:
253         latitude, longitude = coordinates
254         air_data = get_air_quality(latitude, longitude)
255         health_data = get_health_recommendations(latitude, longitude)
256         weather_data = get_weather(latitude, longitude)
257
258         render_template("index.html", city=city, air_data=air_data, health_data=health_data, weather_data=weather_data)
259     else:
260         return None
261
262 if __name__ == "__main__":
263     main()

```

Figure 5.5: Code for the WeatherWiseHub

Chapter 6

System Implementation

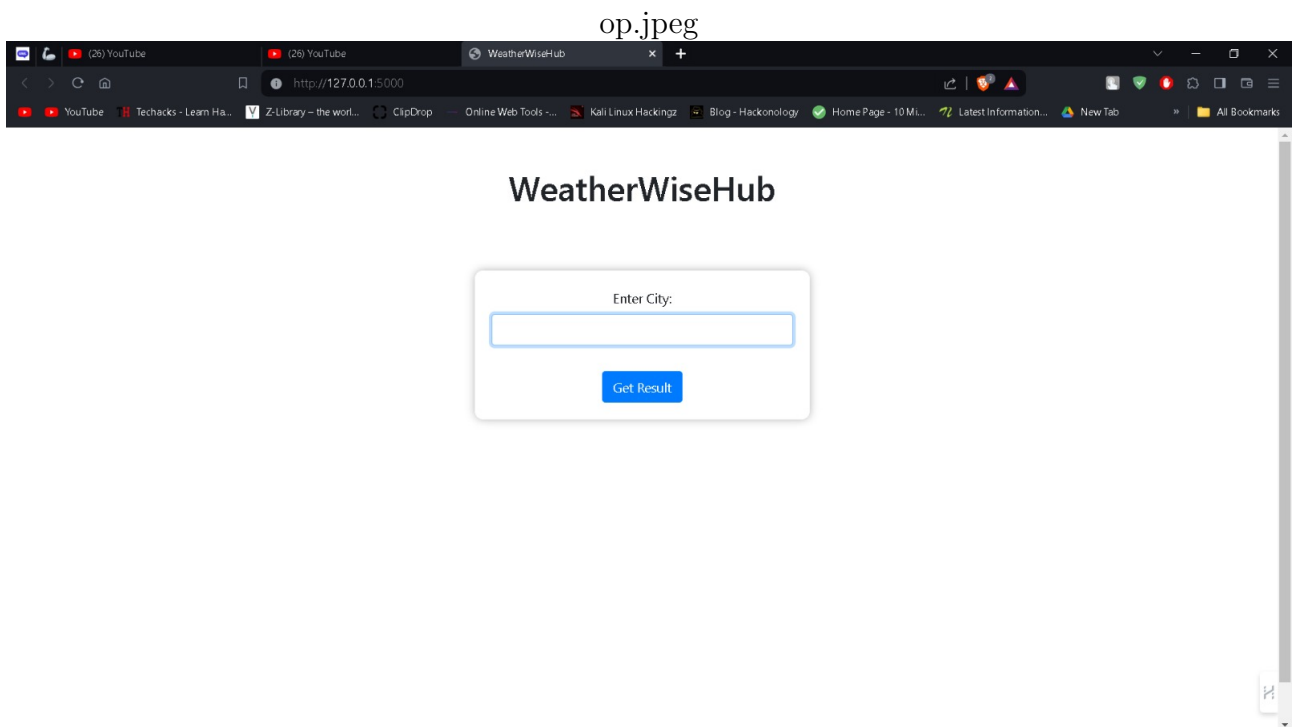


Figure 6.1: Output for the WeatherWiseHub

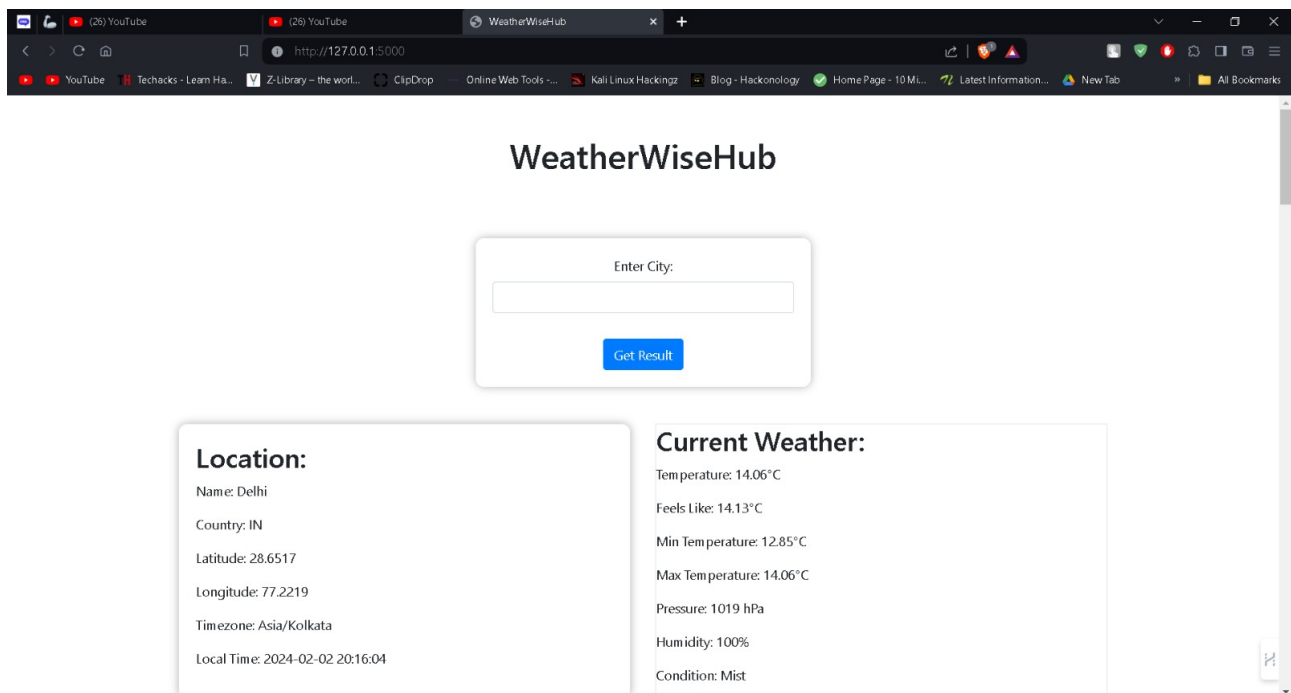


Figure 6.2: Output for the WeatherWiseHub

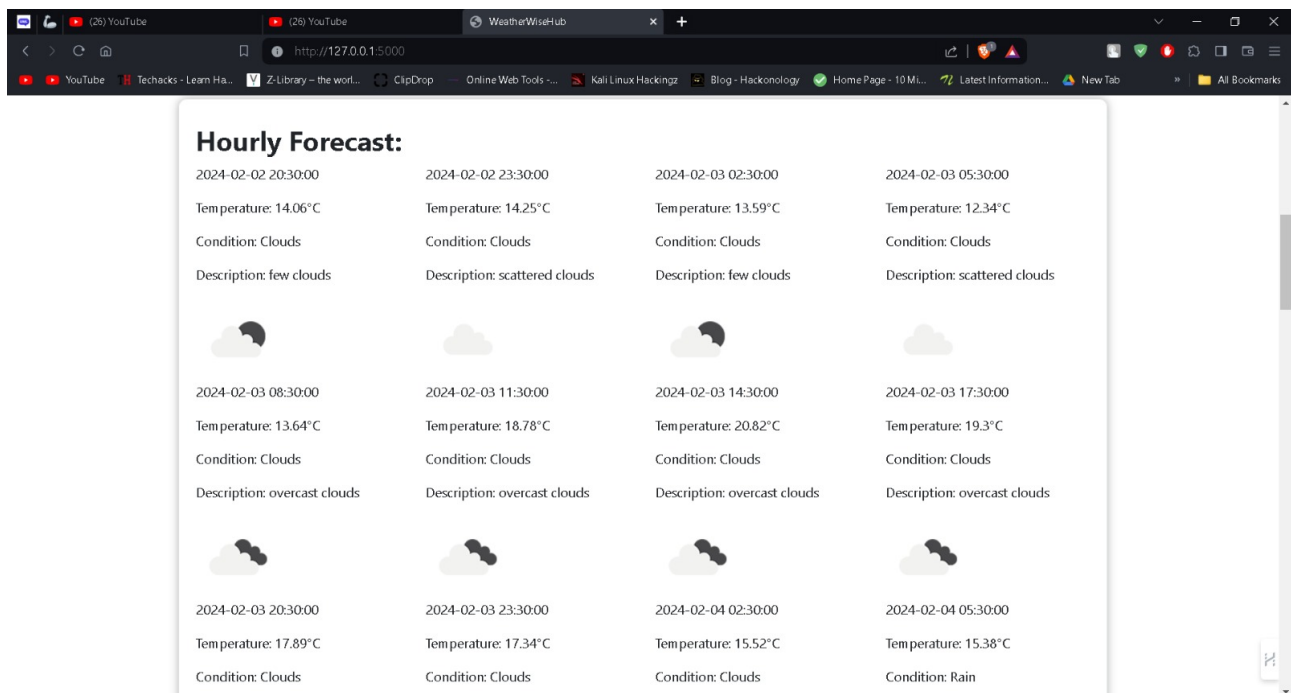


Figure 6.3: Output for the WeatherWiseHub

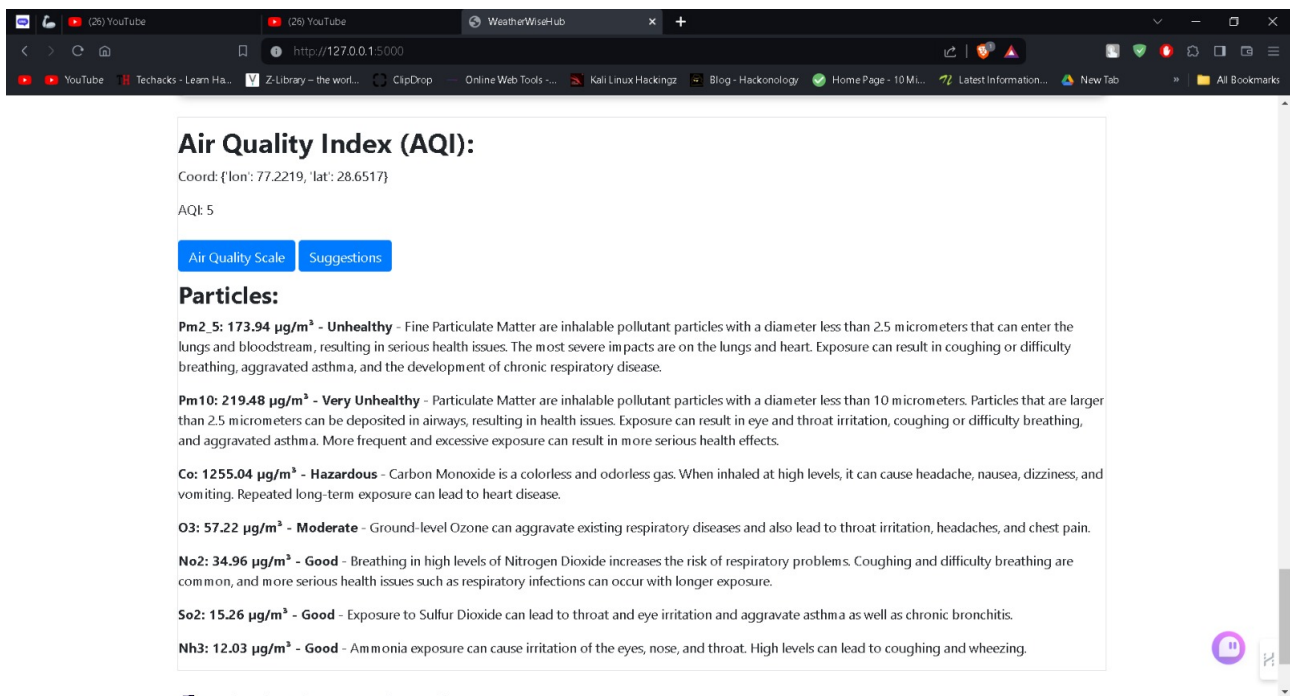


Figure 6.4: Output for the WeatherWiseHub

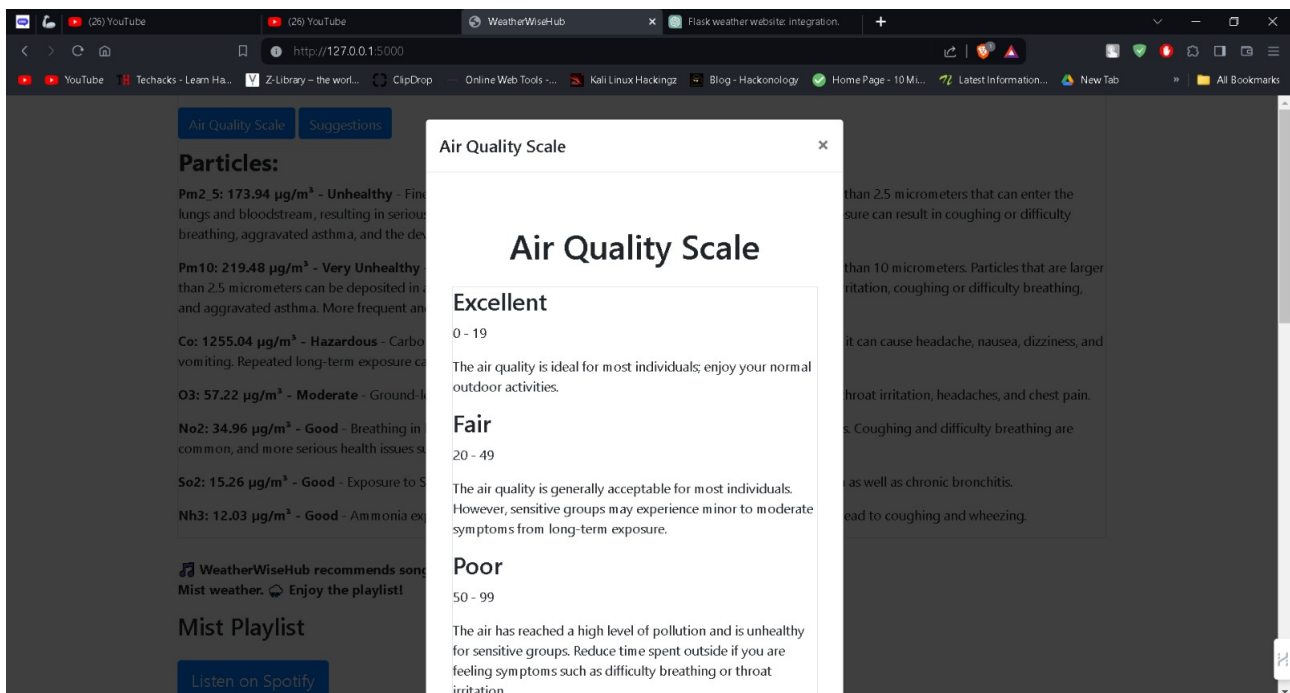


Figure 6.5: Output for the WeatherWiseHub

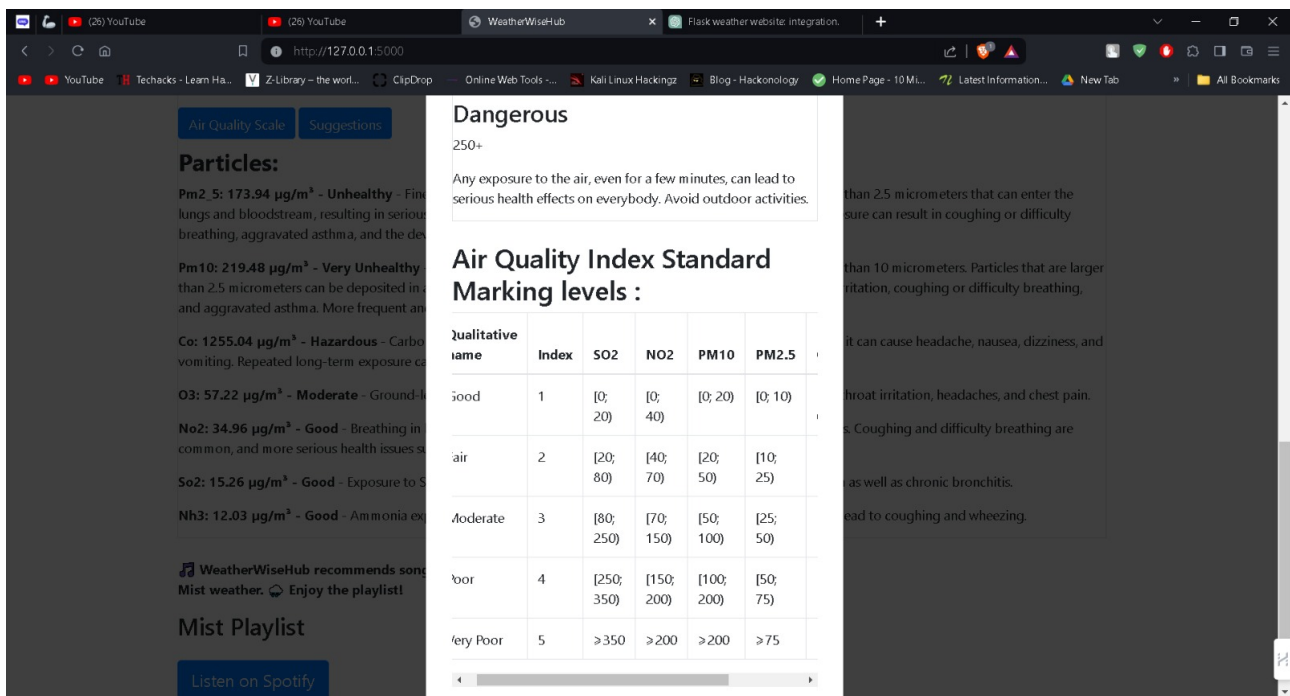


Figure 6.6: Output for the WeatherWiseHub

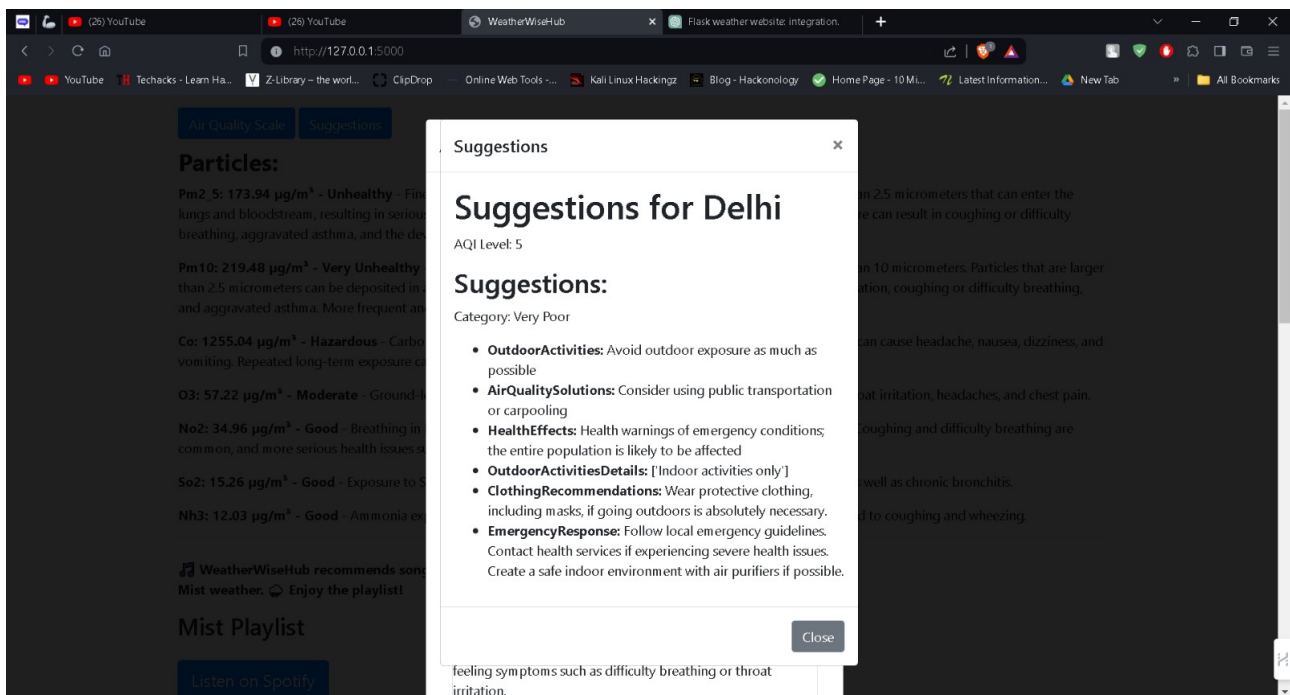
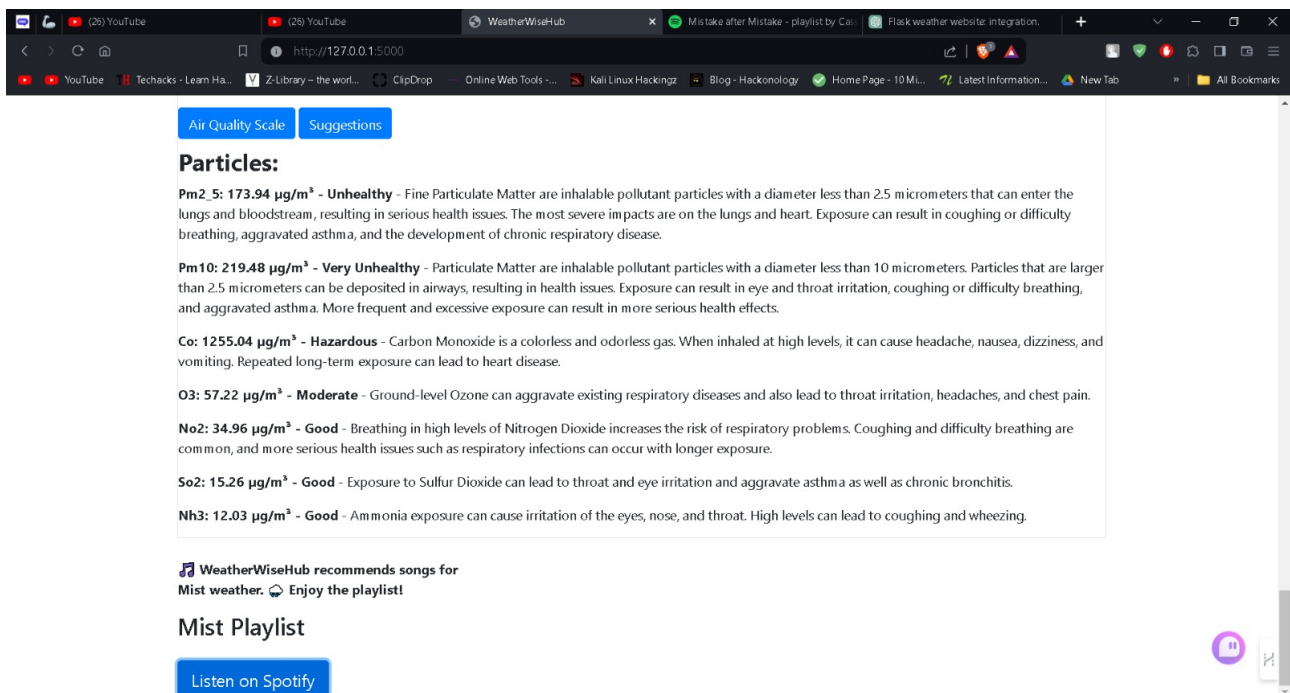


Figure 6.7: Output for the WeatherWiseHub



The screenshot shows a web browser with the WeatherWiseHub website. The page has a dark theme and includes a navigation bar with links like 'Air Quality Scale' and 'Suggestions'. The main content area is titled 'Particles:' and lists various air quality metrics with their corresponding health impacts. Below this, there is a section for 'Mist Playlist' with a 'Listen on Spotify' button. The browser's address bar shows the URL 'http://127.0.0.1:5000'.

Particles:

Pm2.5: 173.94 $\mu\text{g}/\text{m}^3$ - Unhealthy - Fine Particulate Matter are inhalable pollutant particles with a diameter less than 2.5 micrometers that can enter the lungs and bloodstream, resulting in serious health issues. The most severe impacts are on the lungs and heart. Exposure can result in coughing or difficulty breathing, aggravated asthma, and the development of chronic respiratory disease.

Pm10: 219.48 $\mu\text{g}/\text{m}^3$ - Very Unhealthy - Particulate Matter are inhalable pollutant particles with a diameter less than 10 micrometers. Particles that are larger than 2.5 micrometers can be deposited in airways, resulting in health issues. Exposure can result in eye and throat irritation, coughing or difficulty breathing, and aggravated asthma. More frequent and excessive exposure can result in more serious health effects.

Co: 1255.04 $\mu\text{g}/\text{m}^3$ - Hazardous - Carbon Monoxide is a colorless and odorless gas. When inhaled at high levels, it can cause headache, nausea, dizziness, and vomiting. Repeated long-term exposure can lead to heart disease.

O3: 57.22 $\mu\text{g}/\text{m}^3$ - Moderate - Ground-level Ozone can aggravate existing respiratory diseases and also lead to throat irritation, headaches, and chest pain.

No2: 34.96 $\mu\text{g}/\text{m}^3$ - Good - Breathing in high levels of Nitrogen Dioxide increases the risk of respiratory problems. Coughing and difficulty breathing are common, and more serious health issues such as respiratory infections can occur with longer exposure.

So2: 15.26 $\mu\text{g}/\text{m}^3$ - Good - Exposure to Sulfur Dioxide can lead to throat and eye irritation and aggravate asthma as well as chronic bronchitis.

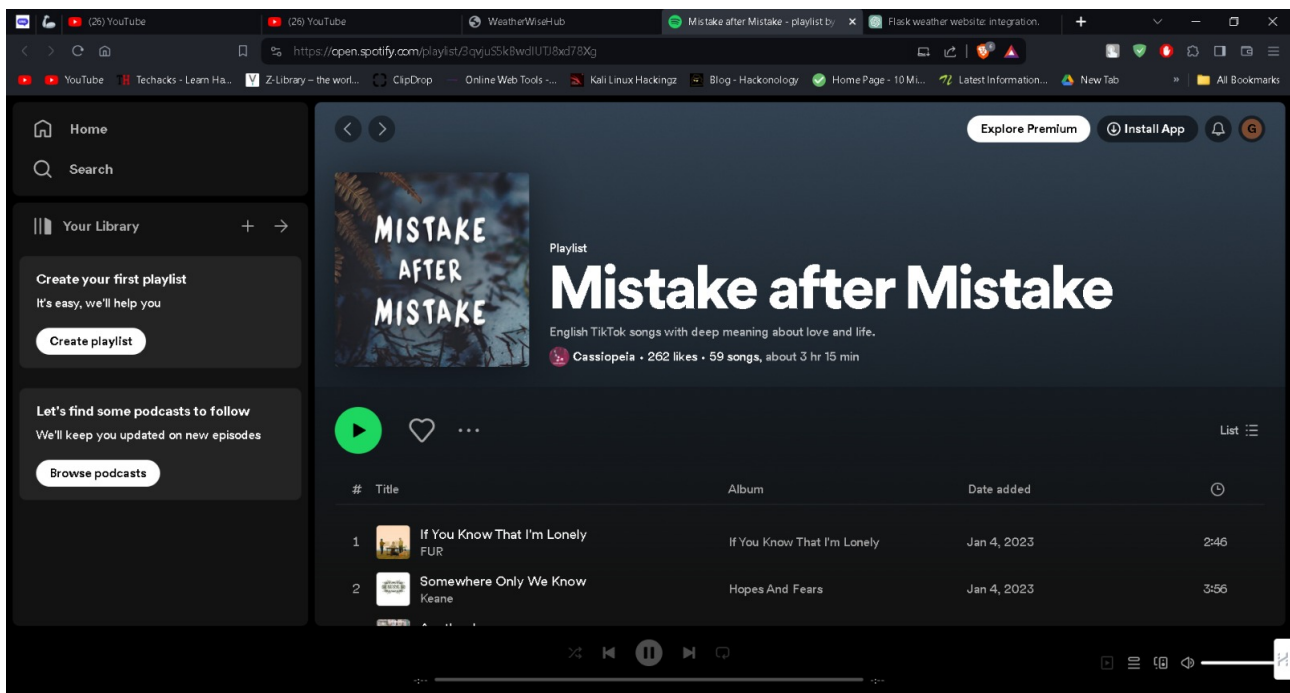
Nh3: 12.03 $\mu\text{g}/\text{m}^3$ - Good - Ammonia exposure can cause irritation of the eyes, nose, and throat. High levels can lead to coughing and wheezing.

WeatherWiseHub recommends songs for Mist weather. Enjoy the playlist!

Mist Playlist

[Listen on Spotify](#)

Figure 6.8: Output for the WeatherWiseHub



The screenshot shows the Spotify web player interface. The main content area displays the 'Mistake after Mistake' playlist, which is described as 'English TikTok songs with deep meaning about love and life.' The playlist has 262 likes and 59 songs, totaling 3 hours and 15 minutes. The track list shows two songs: 'If You Know That I'm Lonely' by FUR and 'Somewhere Only We Know' by Keane. The Spotify interface includes a sidebar with navigation options like 'Home', 'Search', and 'Your Library', and a bottom player bar with playback controls.

Mistake after Mistake

Playlist

English TikTok songs with deep meaning about love and life.

Cassiopeia • 262 likes • 59 songs, about 3 hr 15 min

#	Title	Album	Date added	
1	If You Know That I'm Lonely FUR	If You Know That I'm Lonely	Jan 4, 2023	2:46
2	Somewhere Only We Know Keane	Hopes And Fears	Jan 4, 2023	3:56

Figure 6.9: Output for the WeatherWiseHub

Chapter 7

System Testing

7.1 Testing Methodologies

Approximately 80 percent of testing is planned for WeatherWiseHub, covering functional, integration, usability, performance, and security aspects

Currently, 60 percent of the testing has been successfully completed, with ongoing efforts to address and resolve identified issues.

7.2 Test Cases

Test Case 1: Weather Data Retrieval

1. Input: User requests weather information for a specific location.
2. Expected Output: Accurate and real-time weather data displayed on the user interface.

Test Case 2: Music Recommendation

1. Input: User preferences and location are considered for music recommendations.
2. Expected Output: Tailored music playlist suggestions based on weather conditions. These

test cases represent a subset of the comprehensive testing strategy applied to

WeatherWiseHub, focusing on critical functionalities and their expected outcomes. Each test case is meticulously designed to evaluate specific features, ensuring the overall robustness and reliability of the application.

Chapter 8

Limitation and Future Enhancement

8.1 Limitations

- WeatherWiseHub's precision relies on OpenWeatherMap's coverage, potentially leading to reduced accuracy in regions with sparse monitoring.
- The application's functionality is contingent on external APIs like OpenWeatherMap and Spotify, and disruptions to these APIs may affect performance.
- WeatherWiseHub exclusively integrates with Spotify for music recommendations, limiting options; future versions may explore additional music services.
- Personalization is static, based on user preferences; future iterations could leverage machine learning for dynamic adaptation.
- Current language support is primarily English; expanding to more languages would enhance global accessibility.

8.2 Future Enhancements

- Explore news integration to keep users informed about weather-related events.
- Implement machine learning for music recommendations, adapting to users' evolving tastes.
- Extend personalized recommendations beyond music, including activities, events, and local discounts.
- Expand language support beyond English for a more inclusive user experience.
- Enhance user profiles with additional preferences, fostering a more tailored interaction.
- Improve voice command capabilities for a more intuitive and hands-free experience.
- Introduce limited offline functionality to address connectivity challenges, enhancing reliability.

References

- [1] J. Mantas. Methodologies in pattern recognition and image analysis-a brief survey. *Pattern Recognition*, 1987.
- [2] Samuel Foucher. An evaluation of medical imaging. In *IGARSS*, 2009.
- [3] Sen Lee and Eric Foucher. *Radar Imaging and Applications*. CRC Press, 2009.
- [4] A. Zurada, S. Zage, and E. Nezry. Frequency division FDM. In *Remote Sensing Symposium, 1992. IAPRSS '98.*, pages 80 –82, 1992.
- [5] Shitole Sanjay. Digital image processing. *Recognition*, 1988.
- [6] Y S Rao. Microwave imaging. *Imaging*, 1992.