PREFACE

This project has been submitted in the fulfillment of the requirements for the diploma of engineering. We, the team members of this project, take pleasure in presenting the detailed project report that reflects our efforts in the academic year 2024-25.

Our project involves designing a Grid framework for executing complex applications where the process is divided into threads and accordingly the threads are executed by the executors. The outputs generated by the executors are given back to the manager which in turn gives the results to the owner. This is a site in which the manager can select particular executors to run the application.

Initially the manager is started by connecting it to a storage application. The executors are connected to the manager by providing the required credentials. Once the executors get connected to the manager the execution of the required can be started.

Additionally there is a Grid console which keeps track of the executors connected and the applications running. A record of all the operations performed by either of the loggers is maintained in a log file.

ACKNOWLEDGEMENT

We have immense pleasure in presenting the report for our project entitled "Weather forecasting website".

We would like to take this opportunity to express our gratitude to a number of people who have been sources of help & encouragement during the course of this project.

We are very grateful and indebted to our project guide Subba Rao Dusari

for providing their enduring patience, guidance & invaluable suggestions. They were the one who never let our morale down & always supported us through our thick & thin. They were the constant source of inspiration for us & took utmost interest in our project.

We would also like to thank all the staff members for their invaluable co-operation & permitting us to work in the computer lab.

We are also thankful to all the students for giving us their useful advice & immense cooperation. Their support made the working of this project very pleasant.

Project By:

EDUPULAPATI SAI PRANEETH

INDEX

CHAPTER-1

INTRODUCTION

Introduction

In collaborative software development, multiple developers often work on different modules of a project. To ensure smooth integration, it becomes essential for team members to understand each other's work. This applies to the **Weather Forecasting Web Application**, developed using **Flask (Python)**, where various components like API integration, backend logic, and frontend design work together to deliver a functional system.

The application uses the **OpenWeatherMap API** to fetch real-time weather data for any city and displays it through a **responsive, user-friendly interface**. Rather than following a rigid, algorithmic structure, this project embraces a **modular and object-oriented design**, allowing for easier maintenance, scalability, and better code organization.

By dynamically fetching and displaying live weather data, the project demonstrates practical implementation of API-driven development and reflects core principles of modern web application design.

1.1 OBJECTIVES

To design and develop a **Weather Forecasting Web Application** using **Flask (Python)** that provides real-time weather information for any city

Scope:

This application primarily focuses on the retrieval and presentation of current weather data. It utilizes the **OpenWeatherMap API** to fetch real-time weather parameters such as:

- Temperature
- Humidity
- Wind speed
- Atmospheric pressure
- Weather conditions (e.g., rain, clear sky, clouds)

The scope also includes the development of a responsive user interface to ensure compatibility across devices. The system handles API communication, processes JSON responses, and dynamically updates the UI based on user input. While the project is currently limited to current weather data, it can be extended to include forecast data, map views, or historical trends.

Description of the project:

The Weather Forecasting Application addresses the increasing demand for accessible and accurate weather information. Built using **Flask**, the system adopts a modular and scalable architecture where the front-end and back-end are clearly separated. Upon entering a city name, the application communicates with the **OpenWeatherMap API**, retrieves the corresponding data, and displays it in a clean, user-friendly format.

The application reduces dependency on traditional sources and provides instant access to localized weather data. It serves as a practical demonstration of **API integration**, **real-time data handling**, and **responsive design** principles. In addition, it highlights how object-oriented design and external data services can be used effectively in modern web development.

1.1 INITIAL INVESTIGATION

The first phase in software development is the requirements gathering stage. This phase involves identifying the purpose, scope, and goals of the application. For the Weather Forecasting Application, the idea was developed to address the need for real-time weather information that is accessible through a user-friendly web interface.

The application aims to fetch accurate and up-to-date weather data for any city, using API integration and modern web technologies. During this stage, key modules and functionalities were defined, including city-based weather search, dynamic data display, and responsive design.

Key activities in the requirements gathering phase included:

- Identifying the need for real-time weather updates
- Researching third-party APIs for weather data (e.g., OpenWeatherMap)
- Defining user interface requirements and system functionality
- Establishing project goals, tools, and timelines
- Documenting the technical and functional requirements

1.2 SELECTION OF LANGUAGE

- ✓ PYTHON 3x
- ✓ FLASK
- ✓ OPENWEATHERMAP API
- ✓ HTML/CSS
- ✓ WEB BROWSER

The Weather Forecasting Web Application is developed using **Python** and the **Flask** web framework due to their simplicity and efficiency in building web-based applications.

- **Python** is selected for its readability and strong support for web development and API integration.

- **Flask** is used for handling routing, backend logic, and dynamic content rendering with minimal overhead.

- The application integrates the **OpenWeatherMap API** to fetch real-time weather data including temperature, humidity, wind speed, and weather conditions.

- The frontend is built using **HTML** and **CSS**, ensuring a clean and responsive user interface compatible with modern browsers.

Overview of Flask Features

Flask provides several benefits, including:

- Minimal and flexible architecture
- Built-in development server and debugger
- RESTful request handling
- Support for templating with Jinja2
- Easy integration with APIs and databases
- Modular design for scalability

CHAPTER-2

FEASIBILITY STUDY

2.1 INTRODUCTION

Before developing any system, it is important to study its feasibility to ensure it is practical and beneficial. For the **Weather Forecast Website**, this study helps determine if the project can be developed with available resources, meets user needs, and is worth implementing.

The feasibility study on 3 major questions:

1. Does the candidate system meet the user requirement?

Yes, the system provides accurate, real-time weather updates tailored to the user's location.
It includes features like current temperature, forecasts, and weather icons.
The interface is simple, responsive, and easy to use on all devices.
These features fully meet the basic needs and expectations of users.

2. Is the problem worth solving?

Yes, people depend on weather forecasts for planning daily tasks and travel.
Accurate weather data supports safety, especially during extreme conditions.
Farmers, outdoor workers, and travelers benefit significantly from forecasts.
Hence, offering a reliable weather website solves a real-world problem.

3. What is the impact of the system on the organization?

The website increases user engagement by providing valuable and timely information.
It builds trust and enhances the organization's public image.
It may also create growth opportunities through ad revenue or collaborations.
Overall, it adds long-term value and visibility to the organization.

2.2 TECHNICAL, ECONOMICALLY & OPERATIONAL FEASIBILITY

1. **Financial feasibility:**

   The weather forecast website requires minimal investment, using free or low-cost tools and APIs. It is affordable and financially suitable for small-scale or academic projects.

2. **Technical feasibility:**

   The website can be developed using basic web technologies and free weather APIs, which are easily available. It is technically feasible with the available tools and skills..

3. **Behavioral feasibility:**

   Users are likely to accept and use the website as it provides useful, real-time weather updates. Its simple design and easy access encourage positive user behavior.

4. **Operational feasibility:**

   The website is easy to use and meets the daily needs of users by providing accurate weather updates. It can run smoothly with minimal maintenance.

2.3 FEASIBILITY GAINED BY OUR SYSTEM

The system is feasible in all aspects—simple to build, cost-effective, and user-friendly.

Technical Feasibility

The website uses basic web technologies and free weather APIs. It is simple to develop and technically feasible with available resources

Economical Feasibility

The weather forecast website is affordable to develop and maintain. It uses free or low-cost tools, APIs, and hosting services. This makes it suitable for students, individuals, or small organizations with limited budgets.

Operational Feasibility

The website is user-friendly and delivers accurate weather updates. It can be operated easily with minimal effort and maintenance.

Cost-Benefit Analysis

The development cost is low, using free tools, APIs, and hosting services. In return, the website provides valuable weather information, improves user convenience, and increases accessibility. The benefits outweigh the minimal costs, making the project worthwhile

CHAPTER-3

SYSTEM ANALYSIS

3.1 INTRODUCTION

System analysis helps in understanding the requirements and functionality of the system. It identifies what the system should do and how it should perform efficiently. For our weather forecast website, it involves analyzing user needs and data sources. This ensures the system is reliable, accurate, and user-friendly.

User requirements:

The following requirements were identified during the analysis of user needs for the Weather Forecast Website:

- Users should access weather updates by location or city name.
- The website must display current, hourly, and weekly forecasts.
- Information like temperature, humidity, and wind speed should be shown.
- The interface should be simple, responsive, and mobile-friendly.
- Admin can manage backend settings and API configurations.
- Data should come from a reliable weather API like OpenWeatherMap

After analyzing the user requirements, the next step is to understand the problem and its context. This involves studying existing weather forecast platforms and identifying the needs specific to our project. Understanding both the current systems and the new system's domain is essential for defining accurate functional specifications. A clear understanding ensures proper design, while any gap may lead to an ineffective or misdirected solution

3.2 ROLE OF SYSTEM ANALYSIS

- **Memory constraints:** The system uses minimal memory as it mainly processes weather data from APIs. Optimized code and lightweight design ensure smooth performance with low memory usage.

- **Software interfaces:** The website interacts with external weather APIs (like OpenWeatherMap) to fetch real-time data. It also uses web technologies like HTML, CSS, JavaScript, and may connect to a backend server if needed.

- **System features:** The website provides real-time weather updates, hourly and weekly forecasts, and shows details like temperature, humidity, and wind speed. It has a simple, responsive interface and uses reliable weather APIs for accurate data.

- **Major concepts used:** The project uses concepts like API integration, responsive web design, real-time data fetching, and user interface development using HTML, CSS, and JavaScript..

CHAPTER-4

APPLICATION DESIGN

4.1 FORM DESIGNING

- **Login Form:-**

  The login form allows the admin to securely access backend settings of the weather website. It ensures only authorized users can manage API configurations and system updates.

- **MDI form:**

  This screen acts as the home screen for the weather forecast website. Through this MDI form, the admin can access its child forms. This form provides the user with 5 main options:

  Login Page: It contains options for LOGIN, LOGOUT, and EXIT to manage access to the website.
  Registration: This module allows the admin to register new users or set up access credentials.

- **Form list and details**

1. **Registration Form:**
   Used by the admin to register new users for backend access.
   It ensures only authorized users can manage weather data settings.

2. **Location Search Form:**
   Allows users to search weather data by city or current location.
   Displays real-time results using weather API integration.

3. **Notification Form:**
 Enables admin to post weather alerts and important updates.
 Helps users stay informed about severe or upcoming weather changes.

4. **Fees Form:** This form can be used to generate subscription or service fee reports for users.

4.2 FRONT END - BACK END CONNECTIVITY

Java as Front end

Java offers several benefits for developers creating front-end applications that interact with the backend. Being a platform-independent language, Java can be executed on any architecture that supports the JVM. This flexibility reduces costs related to deployment and hardware/software maintenance. Java-based clients use minimal hardware resources, making them efficient. For businesses, Java-based solutions are advantageous as they can easily be shifted across architectures without additional overhead or costs

MySQL-Back end

MySQL is used as the backend database to store user information, login credentials, and weather search history. It is a reliable, open-source relational database that supports fast data retrieval and secure storage. MySQL easily integrates with Java, making it ideal for dynamic web applications like the weather forecast website.

Setting up MySQL Control:

The first step to use MySQL in the weather forecast project is to add the **MySQLData Control** to the form. The setup includes the following steps:

1. Connecting the front end (Java) to the MySQL database as the data source.

2. Specifying SQL commands to retrieve or store user and weather data.

3. Executing the command to interact with the database (e.g., login, search history).

4. Storing the result in a **RecordSet** for further processing or display.

CHAPTER-5

SYSTEM DESIGN

5.1 INTRODUCTION

The purpose of the Weather Forecasting Web Application Design Document is to outline the design and architecture of the system. This document provides a logical view of the application to help developers understand its structure, data flow, and key components. The system integrates external APIs, server-side scripts, and a responsive frontend to deliver real-time weather updates for cities across the globe.

This application is developed using Python with the Flask framework and utilizes the OpenWeatherMap API to fetch weather data. It can be deployed on any platform that supports Python 3.x. The application files can be run locally or hosted on cloud platforms.

Logical View

The logical view provides an abstract representation of the system functionality, including:

- User interface for city-based weather search
- Backend services for processing API requests
- Data retrieval and parsing from OpenWeatherMap
- Dynamic display of weather parameters

5.2 EVALUATION OF GOOD DESIGN SOFTWARE (LIFE CYCLE MODEL)

System Definition

The Weather Forecasting Web Application is a lightweight, web-based system designed to display current weather information. It adheres to principles of modularity, reusability, and separation of concerns.

Scope and Boundaries

The scope of the system is described as follows:

- It is a web-based application accessible through modern web browsers
- Allows users to enter any city name to retrieve live weather updates
- Integrates a third-party API (OpenWeatherMap) for data accuracy
- Error handling ensures reliability during network or input failures
- No login is required, ensuring ease of access

User View

- Admin (Developer): Has full access to modify backend code, API keys, and deployment configurations. Responsible for maintaining and upgrading the system.

- User: Any visitor to the application. Users can input a city name to retrieve weather details but cannot make changes to the system or data. The interface is designed to be intuitive and user-friendly.

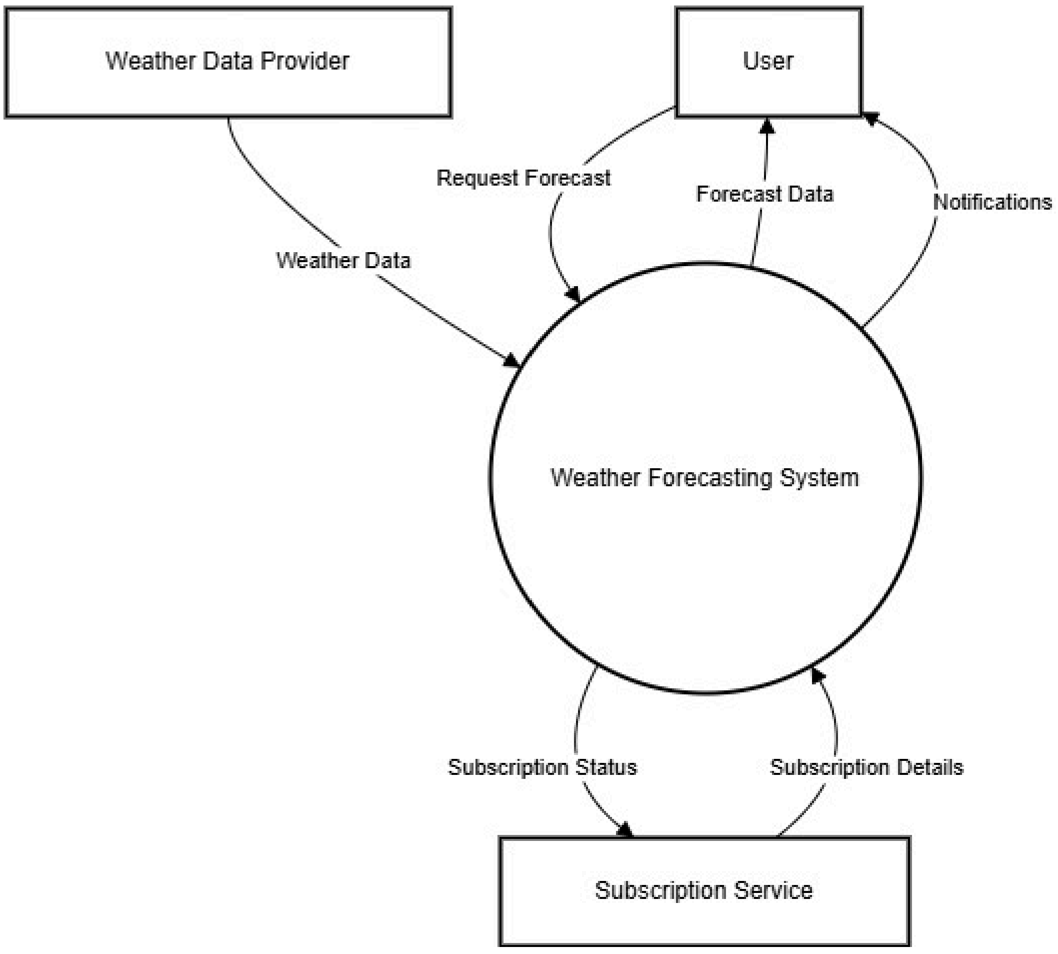
CHAPTER-6

IMPLEMENTATION

DATA FLOW DIAGRAM

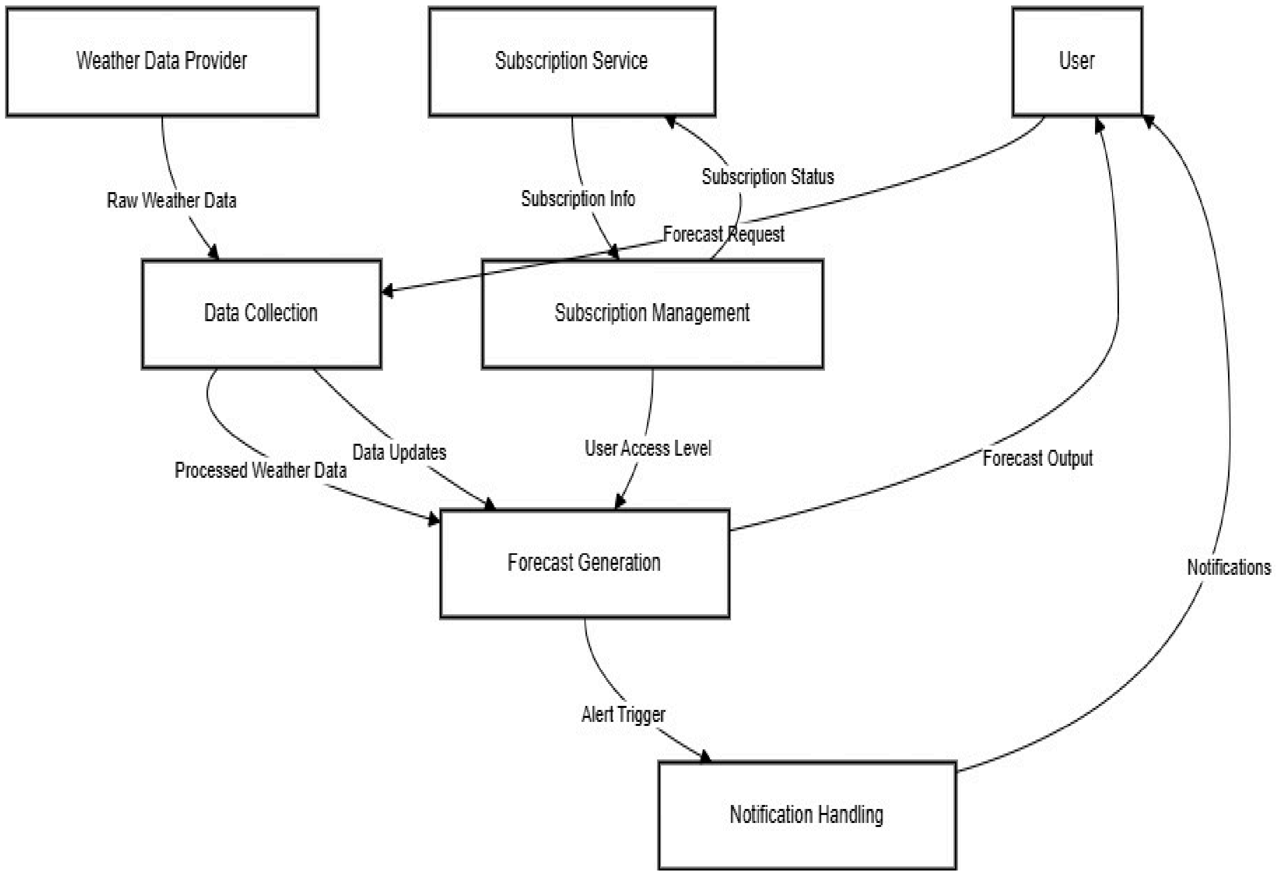
6.1 CONTEXT LEVEL DFD

<u>Level-0</u>



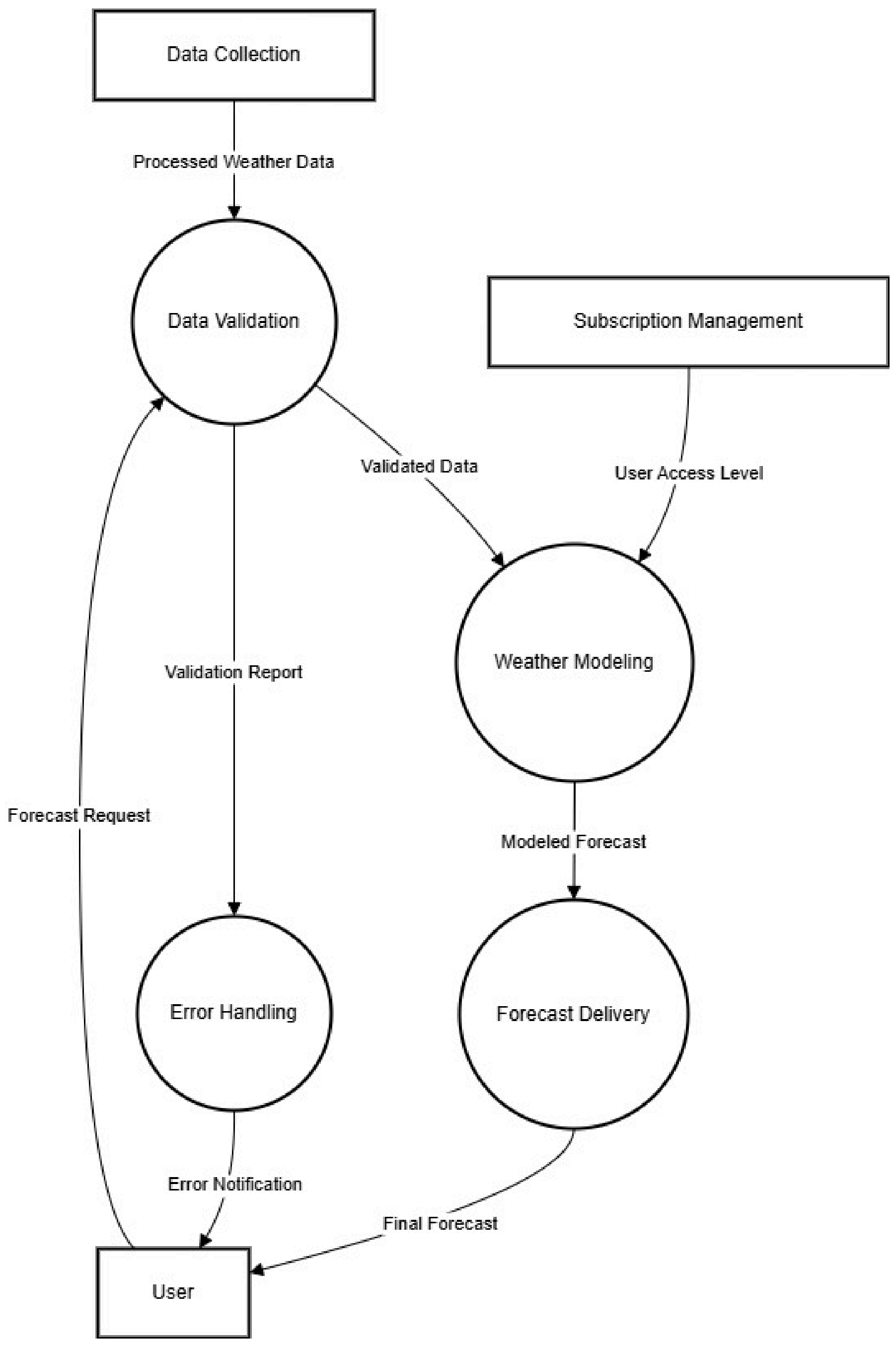
6.2 LEVEL- 1 DFD

<u>Level-1</u>



6.3 LEVEL- 2 DFD

Level -2



CHAPTER-7

TESTING

Testing

The application was tested using various valid and invalid city inputs to ensure reliable performance and accurate data retrieval. Error handling mechanisms were implemented to manage invalid inputs and connectivity issues gracefully.

Both manual testing and functional testing approaches were used to validate the core features, including:

- Fetching weather data for valid cities
- Displaying accurate weather parameters
- Showing appropriate error messages for invalid or empty inputs
- Ensuring smooth user experience across different browsers

The system was found to be stable and responsive under typical usage scenarios.

Types of Testing Performed

1. Unit Testing
    - Individual components, such as API calls, response handling, and data parsing functions, were tested to ensure correct functionality.
2. Integration Testing
    - Interaction between the Flask backend, the OpenWeatherMap API, and the frontend interface was tested to verify seamless data flow and dynamic

updates.

3. Functional Testing

    ○ Key features, such as searching weather by city, displaying correct weather parameters, and handling invalid inputs, were tested to ensure expected outputs.

4. User Interface Testing

    ○ The responsiveness and layout consistency of the application were checked across different browsers and screen sizes.

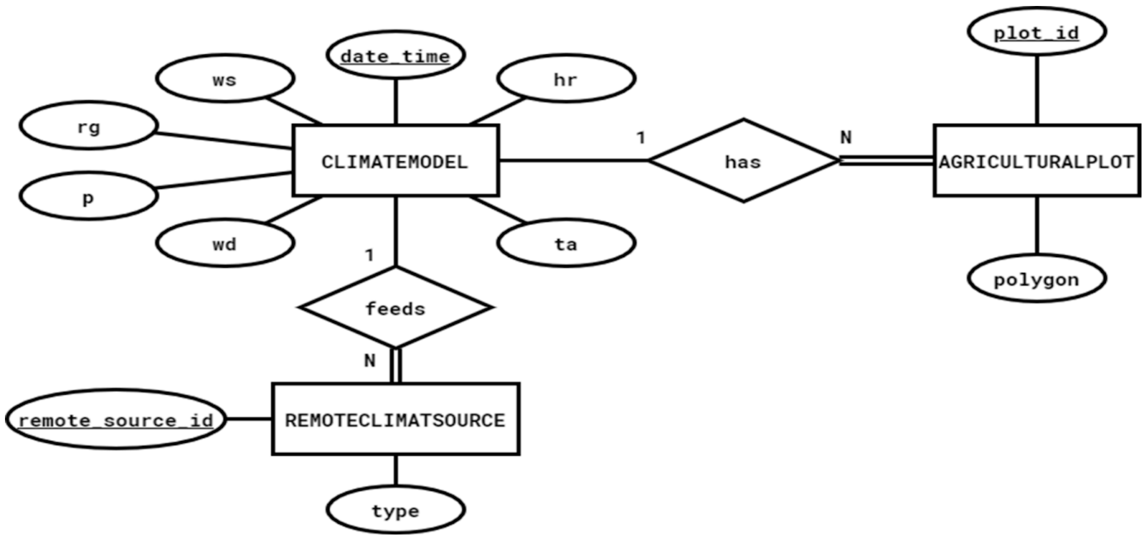
5. Error Handling and Boundary Testing

    ○ Scenarios like empty input, invalid city names, or API errors were tested to ensure proper error messages and system stability.

Testing Tools and Environment

- Postman: For testing API requests and response formats
- Browser Developer Tools: For frontend debugging and responsive testing
- Python's unittest Module: For backend **unit tests**

Entity Relationship Diagram



7.2 DATABASE TABLE

Table name	**Purpose**
1.admin	Admin table contains password and username fields for administrator login and control.
2.location	Location table contains details like city name, state, country, latitude, and longitude.
3.notifications	Notifications table contains weather alerts such as storm warnings, heatwaves, or severe weather updates.
4.weather data	Weather data table contains real-time and historical weather information.
5.userprofile	User profile table holds user details and their preferred locations for weather updates.

6.subscription	Subscription table contains user plans for premium forecasts.

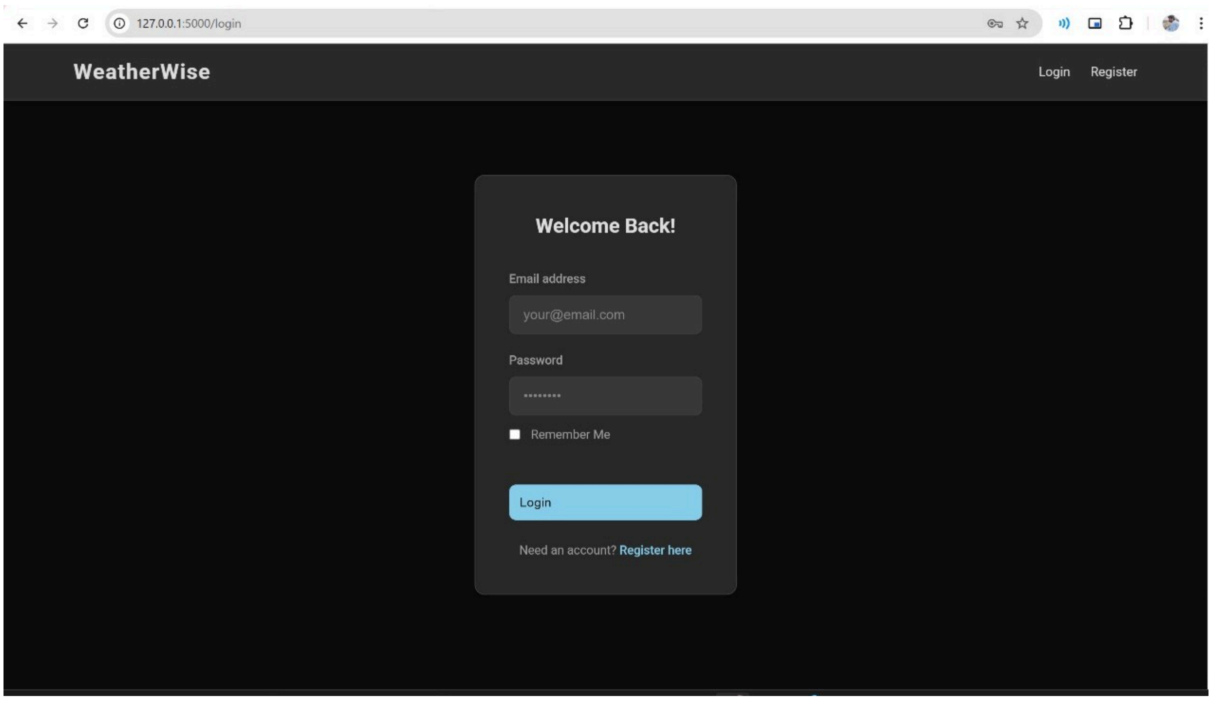
GANTT CHART

	January				February				March				April			
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
1) Problem Definition																
Meet internal guide	🟥	🟦	🟧	🟪												
Identify basic needs		🟪	🟧													
Identify project constraints					🟦	🟧	🟥									
Establish project statement								🟥								
Milestone:-Product statement defined																
2) Feasibility																
Economical feasibility					🟧	🟥	🟦		🟧							
Technical feasibility								🟪	🟪							
Behavioural feasibility										🟦	🟥					
Milestone:- Feasibility study complete																
3) Requirement Determination																
Identify input												🟪	🟧			
Identify output													🟥	🟦		
Process control													🟦	🟥	🟥	🟧
Synopsis																🟥
Milestone:- Synopsis is ready																

G Jaya Prakash Represents:	🟥	E Sai Praneeth Represents:	🟪
	🟦		🟧

D Sriniketh Represents: Devasree Arun Represents:

7.3 SCREENSHOTS



127.0.0.1:5000/login

WeatherWise

Login Register

Welcome Back!

Email address

your@email.com

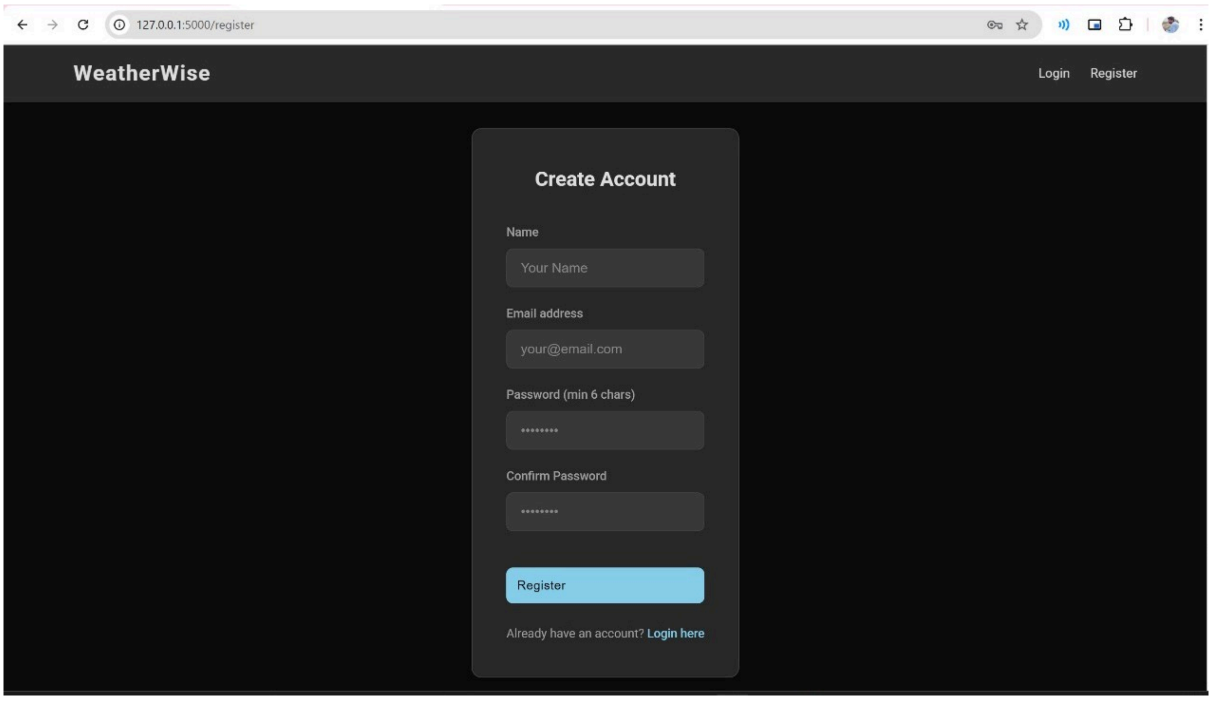
Password

••••••••

Remember Me

Login

Need an account? Register here



127.0.0.1:5000/register

WeatherWise

Login Register

Create Account

Name

Your Name

Email address

your@email.com

Password (min 6 chars)

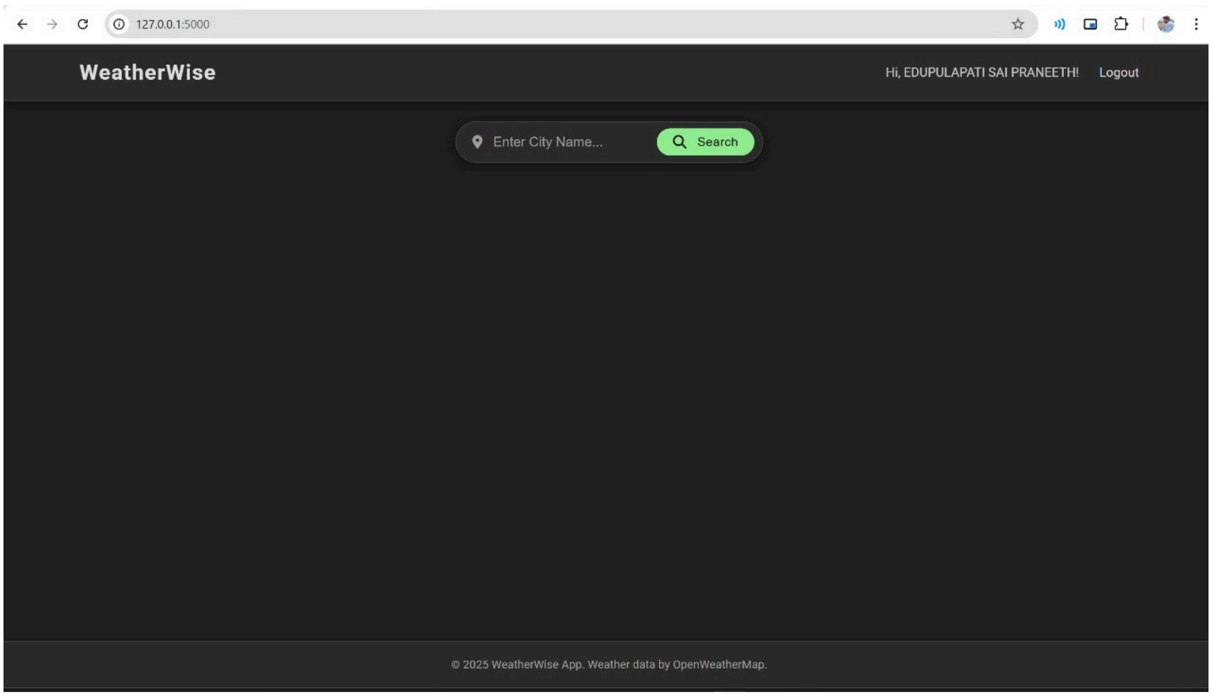
••••••••

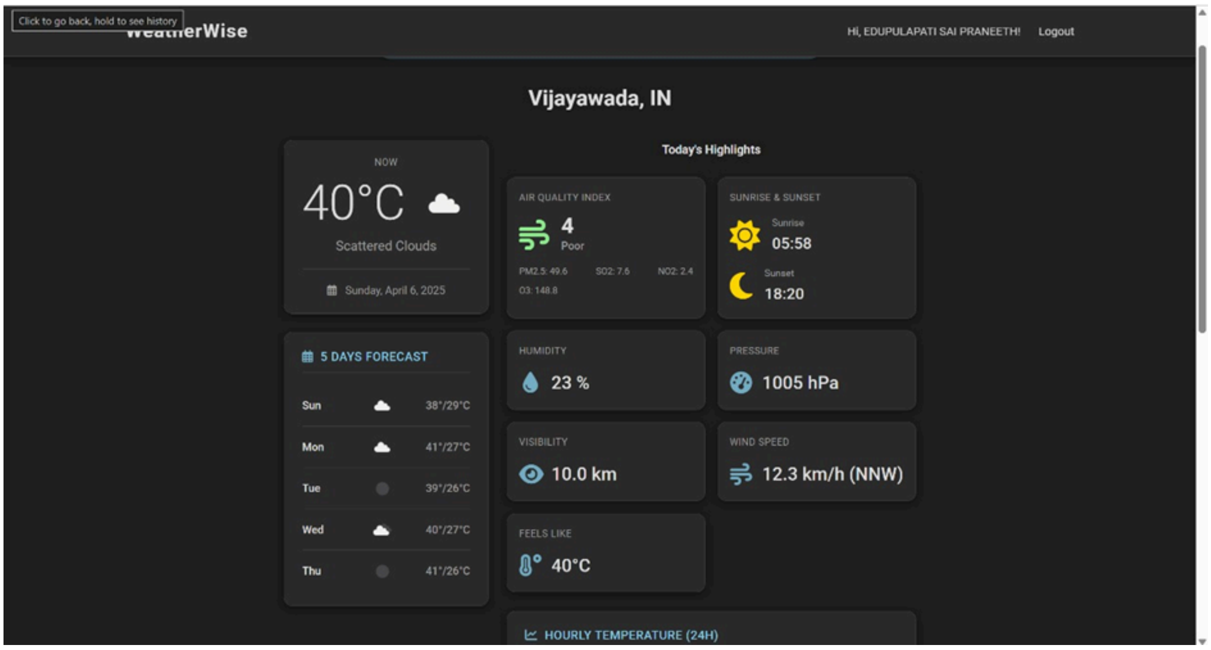
Confirm Password

••••••••

Register

Already have an account? Login here


127.0.0.1:5000
WeatherWise
Hi, EDUPULAPATI SAI PRANEETH!
Logout
Enter City Name...
Search
© 2025 WeatherWise App. Weather data by OpenWeatherMap.


Click to go back, hold to see history
WeatherWise
Hi, EDUPULAPATI SAI PRANEETH!
Logout

Vijayawada, IN

NOW
40°C
Scattered Clouds
Sunday, April 6, 2025

5 DAYS FORECAST

Sun	38°/29°C
Mon	41°/27°C
Tue	39°/26°C
Wed	40°/27°C
Thu	41°/26°C

Today's Highlights

AIR QUALITY INDEX
4
Poor
PM2.5: 49.6 SO2: 7.6 NO2: 2.4
O3: 148.8

SUNRISE & SUNSET
Sunrise 05:58
Sunset 18:20

HUMIDITY 23 %

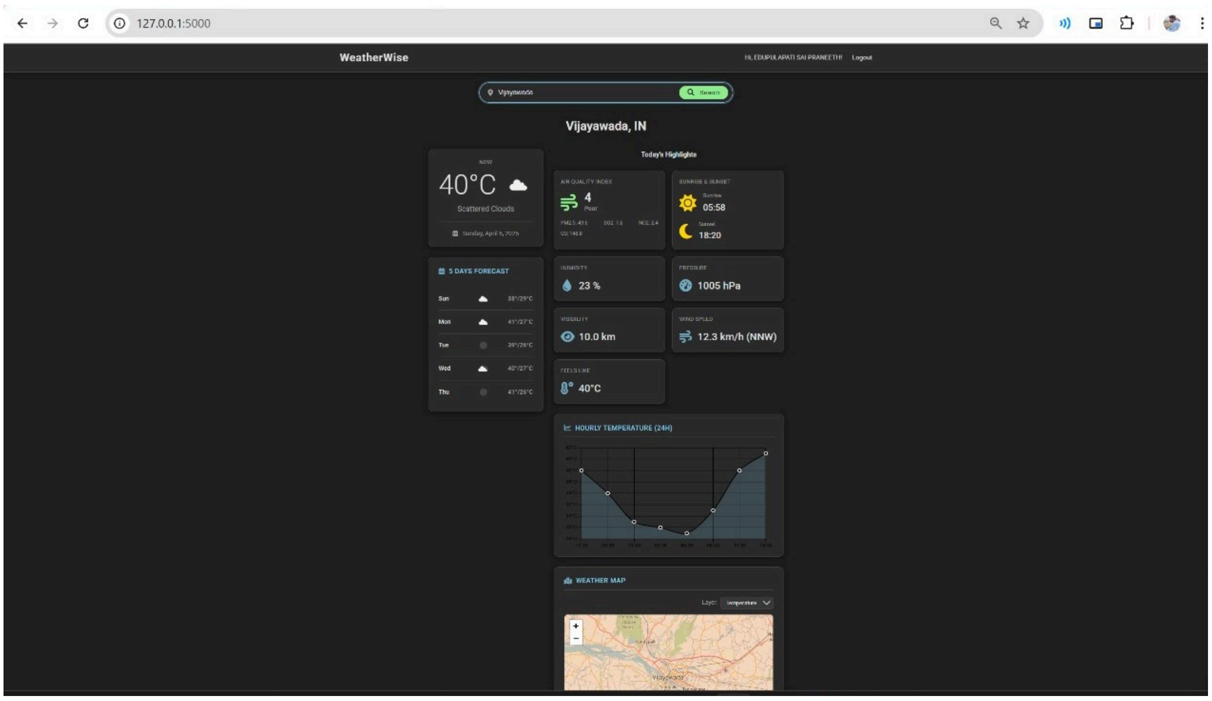
PRESSURE 1005 hPa

VISIBILITY 10.0 km

WIND SPEED 12.3 km/h (NNW)

FEELS LIKE 40°C

HOURLY TEMPERATURE (24H)



Vijayawada, IN

NOW
40°C
Scattered Clouds
Sunday, April 6, 2025

Today's Highlights

AIR QUALITY INDEX
4
Poor
PM2.5: 49.6 SO2: 7.6 NO2: 2.4
O3: 148.8

SUNRISE & SUNSET
Sunrise
05:58
Sunset
18:20

5 DAYS FORECAST

Sun	38°/29°C
Mon	41°/27°C
Tue	39°/26°C
Wed	40°/27°C
Thu	41°/26°C

HUMIDITY
23 %

PRESSURE
1005 hPa

VISIBILITY
10.0 km

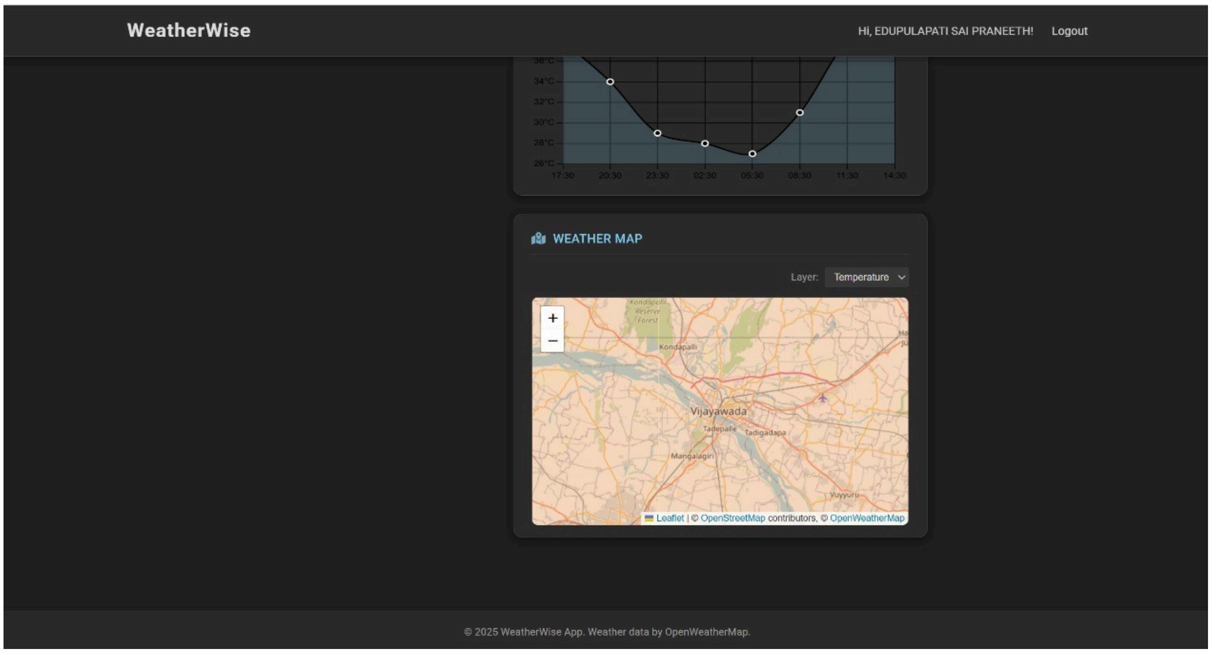
WIND SPEED
12.3 km/h (NNW)

FEELS LIKE
40°C

HOURLY TEMPERATURE (24H)

WEATHER MAP

Layer: Temperature



© 2025 WeatherWise App. Weather data by OpenWeatherMap.

7.4 SYSTEM REQUIREMENTS

Operating system: - Microsoft Windows 10/11.

Front End: - Json, html css javascript.

Back End:- python flask (Database plays an important role in solving the Problem of information management).

SOFTWARE

1. Backend Framework

Flask is a lightweight, flexible microframework for Python. It provides the essential tools for web development, such as routing and templating, but leaves the choice of additional libraries to the developer. In the app, Flask is responsible for handling the HTTP requests from users, routing them to the appropriate view functions, and rendering dynamic HTML templates. It also facilitates user session management, which is essential for user authentication and maintaining states across requests.

2. Database

SQLite:

SQLite is a serverless, self-contained, and lightweight relational database management system (RDBMS). It's an excellent choice for smaller applications or prototypes like my weather app. In this project, SQLite is used to store user login credentials such as names, emails, and password hashes. Since it's embedded directly within the application, it's simple to set up and doesn't require any additional database server.

SQLAlchemy:

SQLAlchemy is a powerful Python ORM (Object Relational Mapper) that provides a way to interact with relational databases using Python objects instead of raw SQL queries. This makes database operations cleaner, more readable, and easier to manage. With SQLAlchemy, I define my data models as Python classes and use its query language to interact with my SQLite database. This abstraction layer makes it easy to switch to other databases in the future if needed, without having to change the core codebase.

3. User Authentication

Flask-Login:

Flask-Login is an extension that handles user session management for Flask applications. It provides functionality for logging in, logging out, and remembering users between sessions. It integrates with Flask's session mechanism to securely store user data such as the current user ID and session timeout. Flask-Login makes it easier to protect routes that require authentication and manage user sessions efficiently.

Flask-Bcrypt:

Flask-Bcrypt is a library used to hash user passwords securely before storing them in the database. It uses the Bcrypt hashing algorithm, which is designed to make it computationally difficult for attackers to recover the original passwords, even if they have access to the hashed versions. When a user logs in, the app hashes the entered password and compares it with the stored hash to authenticate the user.

4. API Integration

OpenWeatherMap API:
OpenWeatherMap provides a comprehensive set of weather data, which my app uses to display current weather and forecasts. By integrating this API, my app can retrieve real-time weather information based on user input (e.g., a city name). The app uses several key endpoints from the OpenWeatherMap API:

- **Geocoding**: Converts city names into geographical coordinates (latitude and longitude).

- **Current Weather**: Fetches the current weather data, including temperature, humidity, and weather conditions.

- **5-day / 3-hour forecast**: Provides weather predictions for the next five days, broken down into 3-hour intervals.

- **Air Pollution Data**: Gathers information on air quality, including pollutants like CO, NO2, and particulate matter, to give users an idea of the environmental conditions.

5. Frontend

Jinja2 Templates (via Flask's `render_template`):

Jinja2 is a templating engine used by Flask to render HTML pages dynamically. It allows me to embed Python-like expressions and control structures within HTML. In my app, Jinja2 is used to inject dynamic content into static HTML templates, such as displaying the current weather for a specific city or user-specific information like login status. The render_template function in Flask is used to pass data from the backend to the template, where it gets rendered into the final HTML page shown to the user.

HTML, CSS, JavaScript (assumed in my templates):

HTML forms the structure of my web pages, providing the skeleton for the content. CSS is used to style and layout the content, ensuring the app is visually appealing and responsive. JavaScript is often used to add interactivity to the frontend, such as handling user inputs, making AJAX calls to fetch weather data without reloading the page, or updating the user interface dynamically based on the data retrieved.

6. Others

Requests:

The Requests library is a simple and easy-to-use HTTP client in Python. In my app, it's used to make requests to the OpenWeatherMap API. I send GET requests to the API endpoints, and in return, I receive JSON-formatted weather data that can be processed and displayed to the user.

Datetime:

The Datetime module is used to manage dates and times within my app. It allows me to display the current date and time dynamically on the frontend or use timestamps for any time-based calculations, such as showing the time of the last update or displaying forecast data for a specific date. The module also ensures that the app's time-related functions are accurate and consistent.

Environment Variables (os.environ):

Environment variables are used to store sensitive or configurable information outside of the codebase, such as the OpenWeatherMap API key, Flask's debug mode setting, and the app's port number. By using os.environ, I can access these variables safely in my code without hardcoding them, which is important for security and portability. It also allows for different configurations in different environments (e.g., development, testing, production).

Future Enhancement:

- Implementing geolocation to automatically detect the user's location and display weather.
- Providing extended 7-day or 10-day forecasts to offer more detailed predictions.
- Adding interactive weather maps with radar and satellite images for real-time visualization.
- Offering personalized recommendations for clothing, activities, or travel based on the weather.

CHAPTER-8

CONCLUSION

CONCLUSION

The Weather Forecasting Web Application is a robust and practical solution designed to provide real-time weather updates for cities around the world. Through the use of the Flask web framework and the OpenWeatherMap API, this project demonstrates how modern web development tools and techniques can be combined to build a responsive and interactive application.

The primary objective of the project was to create an intuitive platform that allows users to access weather information easily and efficiently. This goal has been successfully achieved by ensuring a smooth user experience, fast data retrieval, and clear presentation of weather parameters such as temperature, humidity, wind speed, and weather conditions.

This project also highlights essential development practices, such as:

- Backend and frontend integration
- Effective API usage
- Error handling for invalid inputs and connectivity issues
- Responsive interface design using HTML and CSS

From a learning perspective, the development of this application helped reinforce concepts like RESTful API consumption, web routing in Flask, and dynamic HTML rendering.

Looking forward, the application has a strong potential for expansion. Future enhancements could include displaying multi-day weather forecasts, adding graphical weather representations, enabling location-based automatic weather detection, introducing user accounts for saving search history or preferences.

In conclusion, this project serves as a solid foundation for further development and provides a real-world example of how software can be used to solve everyday problems with technology.

BIBLIOGRAPHY

- https://openweathermap.org/api

- https://flask.palletsprojects.com

- https://developer.mozilla.org