A Field Project Report on

Integrated approach for Weather forecasting

Submitted

In partial fulfillment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE and ENGINEERING

By

K. Hemanth	(231FA04B79)
G. Srinikhi	(231FA04C41)
J. Krishna	(231FA04C51)
C. Tarun	(231FA04C90)

Under the Guidance of
Dr. Nerella Sameera
Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SCHOOL OF COMPUTING AND INFORMATICS

VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY & RESEARCH
(Deemed to be University)
Vadlamudi, Guntur -522213, INDIA.

April, 2025

VIGNAN'S
FOUNDATION FOR SCIENCE, TECHNOLOGY & RESEARCH
(Deemed to be University) - Estd. u/s 3 of UGC Act 1956

CERTIFICATE

This is to certify that the field project entitled *"Integrated approach for weather forecasting"* is being submitted by **K. Hemanth[231FA04B79]**, **G. Srinikhi [231FA04C41]**, **J. Krishna [231FA04C51]** and **C. Tarun [231FA04C90]** in partial fulfilment of the requirements for the degree of Bachelor of Technology (B.Tech.) in Computer Science and Engineering at Vignan's Foundation for Science, Technology and Research (Deemed to be University), Vadlamudi, Guntur District, Andhra Pradesh, India.

This is a bonafide work carried out by the aforementioned students under my guidance and supervision.

Guide

Project Review Committee

HoD, CSE

HoD
Dept. of Computer Science & Engineering
VFSTR Deemed to be University
VADLAMUDI - 522 213
Guntur Dist., A.P., India

VIGNAN'S
FOUNDATION FOR SCIENCE, TECHNOLOGY & RESEARCH
(Deemed to be University) - Estd. u/s 3 of UGC Act 1956

DECLARATION

Date: 26/04/2025

We hereby declare that the work presented in the field project titled "ONLINE EXAMINATION SYSTEM" is the result of our own efforts and investigations.

This project is being submitted under the supervision of **Dr. Nerella Sameera, Assistant Professor** in partial fulfillment of the requirements for the Bachelor of Technology (B.Tech.) degree in Computer Science and Engineering at Vignan's Foundation for Science, Technology and Research (Deemed to be University), Vadlamudi, Guntur, Andhra Pradesh, India.

K Hemanth Kumar	(231FA04B79)	K. Hemanth
G Srinikhi	(231FA04C41)	G. Srinikhi
J Krishna	(231FA04C51)	J. Krishna Reddy
C Tarun Kumar Reddy	(231FA04C90)	Ch. Tarun Reddy

Chapter No.		Contents	Page No.
1		Introduction	5
	1.1	Problem Definition	6
	1.2	Existing System	6
	1.3	Proposed System	6-7
	1.4	Literature Review	7
2		System Requirements	8
	2.1	Hardware & Software Requirements	9
	2.2	Software Requirements Specification(SRS)	9
3		System Design	10
	3.1	Modules of System	11
	3.2	UML Diagrams	11-12
4		Implementation	13
	4.1	Sample Code	14-18
	4.2	Test Cases	18
5		Results	19
	5.1	Output Screens	20-21
6		Conclusion	22-23
		References	23

CHAPTER-01
INTRODUCTION

1. INTRODUCTION

The primary goal of this project is to create a **Weather Forecast Web Application** that provides real-time weather data for any city in the world. Users face difficulties in accessing accurate weather information from various platforms. The problem is the lack of a simple, easy-to-use interface where they can get current weather details in an organized manner. This application aims to solve this problem by offering a clean and intuitive user experience.

1.1 Problem Definition

- **Problem**: Users need to visit multiple platforms or apps to get the weather information they need, and many of these apps are overly complicated or cluttered with irrelevant information.
- **Solution**: A simple web application that only focuses on weather details like temperature, humidity, and weather condition for a specific city.

1.2 Existing System

Currently, many weather websites and mobile apps exist, such as:

- **Weather.com** and **AccuWeather**: These apps provide a lot of information, but they can be overwhelming and not very user-friendly for users just looking for a quick weather update.
- **Smartphone Apps**: Many smartphone apps provide weather data but require installation and may not always be responsive across devices.

The existing systems often suffer from:

- **Overload of information**: Showing more data than necessary, which can be confusing.
- **Poor user experience**: Cluttered interfaces that make it hard to find the relevant data quickly.
- **Limited accessibility**: Many weather platforms are not optimized for all devices or fail to perform well on slower networks.

The goal is to provide a **simplified** alternative that focuses on user-friendliness and speed, ensuring it works well across all devices.

1.3 Proposed System:

The **proposed system** will be a simple web-based application that:

- Takes **user input** (city name).
- **Fetches real-time weather data** from the **OpenWeatherMap API**.
- Displays weather information **in a simple, clean interface**.
- Handles possible **errors** like invalid city names or API connectivity issues.

Features of the proposed system:

- **Simple, user-friendly interface** with clear input fields and weather data display.
- **Fast loading** to provide near-instant results.
- **Mobile responsiveness** to ensure it works on all devices.
- **Error handling** to guide users when they input an invalid city name or when there's no internet connection.

1.4 Literature Review

- **Weather API Integration**: The integration of APIs such as OpenWeatherMap has made it easier for developers to integrate real-time weather data into applications. Research shows that most developers use these APIs for their simplicity and reliability.
- **Responsive Web Design**: Various studies highlight the importance of building web applications that work seamlessly across different screen sizes. Responsive web design ensures accessibility and usability on both mobile and desktop devices.
- **Minimalist Design Philosophy**: Research into user experience design emphasizes the importance of simplicity. Minimalist design, focusing on essential content and removing distractions, is highly effective in improving user engagement.

CHAPTER-02
SYSTEM REQUIREMENTS

2. SYSTEM REQUIREMENTS

For an integrated weather forecasting approach leveraging the provided requirements, the system necessitates a client device with a web browser for user interaction and a server (either remote or local) to potentially handle backend logic and API interactions. The software stack includes frontend technologies (HTML, CSS, JavaScript), a browser for testing, and optionally a backend framework like Node.js to manage API calls and data processing from the OpenWeatherMap API, ensuring a responsive and user-friendly experience while adhering to performance and security standards outlined in the SRS.

2.1 Hardware & Software Requirements

- **Hardware Requirements**:
  - **Client-Side**: A device with an internet connection and a web browser (laptop, desktop, tablet, or smartphone).
  - **Server-Side**: A server for hosting the web application (if applicable) or a local server for testing (e.g., using localhost for local development).
- **Software Requirements**:
  - **Frontend**:
    - HTML, CSS, and JavaScript for creating the user interface.
    - A browser (Chrome, Firefox, Edge) to test the application.
  - **Backend** (if applicable):
    - Node.js or a similar backend framework to handle API calls if you're building a full-stack version.
  - **APIs**:
    - OpenWeatherMap API for retrieving real-time weather data.

2.2 Software Requirements Specification (SRS):

A detailed document specifying the software system's functionality and constraints. This includes:

- **Functional Requirements**:
  - Allow users to enter a city name.
  - Fetch weather data from an external API.
  - Display the data on the UI.
  - Handle invalid inputs (e.g., unknown city names) and API errors (e.g., network issues).
  - Provide a responsive layout for different devices.
- **Non-Functional Requirements**:
  - The application should be available 24/7 with minimal downtime.
  - The system should load in less than 3 seconds.
  - Ensure the website is secure by preventing common vulnerabilities (e.g., XSS, CSRF).

CHAPTER-03
SYSTEM DESIGN

3. SYSTEM DESIGN

The system design for the integrated weather forecasting approach comprises four core modules: the **User Interface (UI)** for user interaction (city input, submission, weather display, error messages); the **API Integration Module** to handle communication with the OpenWeatherMap API (requesting data, parsing JSON responses); the **Error Handling Module** to manage issues like invalid city names or API failures, providing user-friendly feedback; and the **Responsive Layout Module** to ensure the application adapts to various devices for optimal usability.

UML diagrams further detail the system: a **Use Case Diagram** illustrating user interactions (enter city, submit, view data, handle error); a **Class Diagram** defining key classes (WeatherApp, APIHandler, UI) and their attributes/responsibilities; and a **Sequence Diagram** depicting the flow of actions from user input to weather data display or error message presentation.

3.1 Modules of the System

1. **User Interface (UI) Module**:
    - **City Input Field**: A text input box for the user to enter a city name.
    - **Submit Button**: A button to send the request and fetch weather data.
    - **Weather Information Area**: Displays the weather data returned from the API (temperature, humidity, weather condition).
    - **Error Section**: Displays messages like "City not found" or "Unable to fetch data".
2. **API Integration Module**:
    - This module manages communication with the **OpenWeatherMap API**.
    - Sends HTTP requests using JavaScript's `fetch` method or a library like Axios.
    - Handles the **parsing of JSON responses** and extracting relevant weather data.
3. **Error Handling Module**:
    - Detects errors such as incorrect city names, failed API calls, or network issues.
    - Displays user-friendly messages like "Please check the city name" or "Unable to connect to the weather service."
4. **Responsive Layout Module**:
    - The layout adjusts based on the user's device (e.g., using CSS media queries for mobile, tablet, and desktop).
    - Ensures the app provides a good user experience regardless of screen size.

3.2 UML Diagram:

1. **Use Case Diagram**:
    - **Actors**: User.
    - **Use Cases**:
        - Enter city name.
        - Submit the request.
        - View weather data.
        - Handle error (e.g., invalid city name or no internet).
2. **Class Diagram**:
    - **WeatherApp** class with attributes like `cityName`, `temperature`, `humidity`, and `weatherDescription`.
    - **APIHandler** class responsible for making API requests and parsing responses.
    - **UI** class that deals with the display of data.

3. **Sequence Diagram**:
    - User enters a city name.
    - The system sends a request to OpenWeatherMap API.
    - The API returns data, which is then parsed.
    - The system displays the data on the UI.
    - In case of an error (e.g., wrong city name), an error message is shown.

CHAPTER-04
IMPLEMENTATION

4. IMPLEMENTATION

4.1 SAMPLE CODES:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>5-Day Weather Forecast</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
      background: linear-gradient(to bottom, #87CEFA, #1E90FF);
      overflow: hidden;
      position: relative;
    }

    .container {
      background-color: rgba(255, 255, 255, 0.9);
      padding: 20px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      width: 95%;
      max-width: 800px;
      text-align: center;
    }

    #cityInput {
      padding: 8px;
      font-size: 16px;
      width: 70%;
      max-width: 300px;
      border-radius: 4px;
      border: 1px solid #ccc;
    }

    #getWeatherButton {
      padding: 8px 12px;
      font-size: 16px;
      cursor: pointer;
      background-color: #4CAF50;
      color: white;
      border: none;
```

```
      border-radius: 4px;
      margin-left: 10px;
    }

    #forecastChart {
      margin-top: 20px;
    }

    #errorMessage {
      color: red;
      margin-top: 10px;
    }

    #currentTemp {
      font-size: 18px;
      color: #333;
      margin-top: 15px;
    }

    .forecast-labels {
      display: flex;
      justify-content: space-between;
      margin-top: 20px;
      font-size: 14px;
      color: #333;
    }
  </style>
</head>
<body>
<div class="container">
  <h1>5-Day Weather Forecast</h1>
  <input type="text" id="cityInput" placeholder="Enter city name...">
  <button id="getWeatherButton">Get Forecast</button>
  <div id="currentTemp"></div>
  <canvas id="forecastChart" width="700" height="300"></canvas>
  <div id="errorMessage"></div>
</div>

<script>
  const API_KEY = "d812cb133b6f2333ed252592c414494e";
  const cityInput = document.getElementById('cityInput');
  const getWeatherButton = document.getElementById('getWeatherButton');
  const errorMessage = document.getElementById('errorMessage');
  const currentTemp = document.getElementById('currentTemp');
  const ctx = document.getElementById('forecastChart').getContext('2d');
  let forecastChart;

  async function getForecast() {
    const city = cityInput.value.trim();
    if (!city) {
      alert('Please enter a city name');
```

```
        return;
    }

    const forecastUrl = `https://api.openweathermap.org/data/2.5/forecast?q=${city}&appid=${API_KEY}&units=metric`;
    const currentUrl = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API_KEY}&units=metric`;

    try {
        const [forecastResponse, currentResponse] = await Promise.all([
            fetch(forecastUrl),
            fetch(currentUrl)
        ]);

        const forecastData = await forecastResponse.json();
        const currentData = await currentResponse.json();

        if (forecastData.cod !== "200") {
            showError(`Error: ${forecastData.message}`);
            return;
        }

        if (currentData.cod !== 200) {
            showError(`Error: ${currentData.message}`);
            return;
        }

        // Show current temperature with degree symbol
        const temp = currentData.main.temp.toFixed(1);
        const description = currentData.weather[0].description;
        currentTemp.innerHTML = `Current Temperature in ${city}: ${temp}&#176;C (${description})`;

        // Process 5-day forecast
        const dailyData = {};
        forecastData.list.forEach(entry => {
            const date = entry.dt_txt.split(" ")[0];
            if (!dailyData[date]) {
                dailyData[date] = [];
            }
            dailyData[date].push(entry.main.temp);
        });

        const labels = Object.keys(dailyData).slice(0, 5);
        const temps = labels.map(date => {
            const dayTemps = dailyData[date];
            const avgTemp = dayTemps.reduce((a, b) => a + b, 0) / dayTemps.length;
            return avgTemp.toFixed(2);
        });
```

```
    displayChart(labels, temps);
    errorMessage.innerText = "";
  } catch (error) {
    showError("Network error or API issue. Please check your internet connection.");
  }
}

function displayChart(labels, temps) {
  if (forecastChart) {
    forecastChart.destroy();
  }

  forecastChart = new Chart(ctx, {
    type: 'line',
    data: {
      labels: labels,
      datasets: [{
        label: 'Avg Temperature ( C)',
        data: temps,
        backgroundColor: 'rgba(30, 144, 255, 0.2)',
        borderColor: 'rgba(30, 144, 255, 1)',
        borderWidth: 2,
        fill: true,
        tension: 0.4,
        pointRadius: 5,
        pointBackgroundColor: 'rgba(255, 255, 255, 1)',
        pointBorderColor: 'rgba(30, 144, 255, 1)'
      }]
    },
    options: {
      responsive: true,
      plugins: {
        legend: {
          display: true,
          position: 'top'
        },
        tooltip: {
          mode: 'index',
          intersect: false
        }
      },
      scales: {
        y: {
          beginAtZero: false,
          title: {
            display: true,
            text: 'Temperature ( C)'
          }
        },
        x: {
```

```
                title: {
                    display: true,
                    text: 'Date'
                }
            }
        }
    }
});
}

function showError(message) {
    errorMessage.innerText = message;
    currentTemp.innerText = "";
}

getWeatherButton.addEventListener('click', getForecast);
cityInput.addEventListener('keypress', (e) => {
    if (e.key === 'Enter') getForecast();
});
</script>
</body>
</html>
```

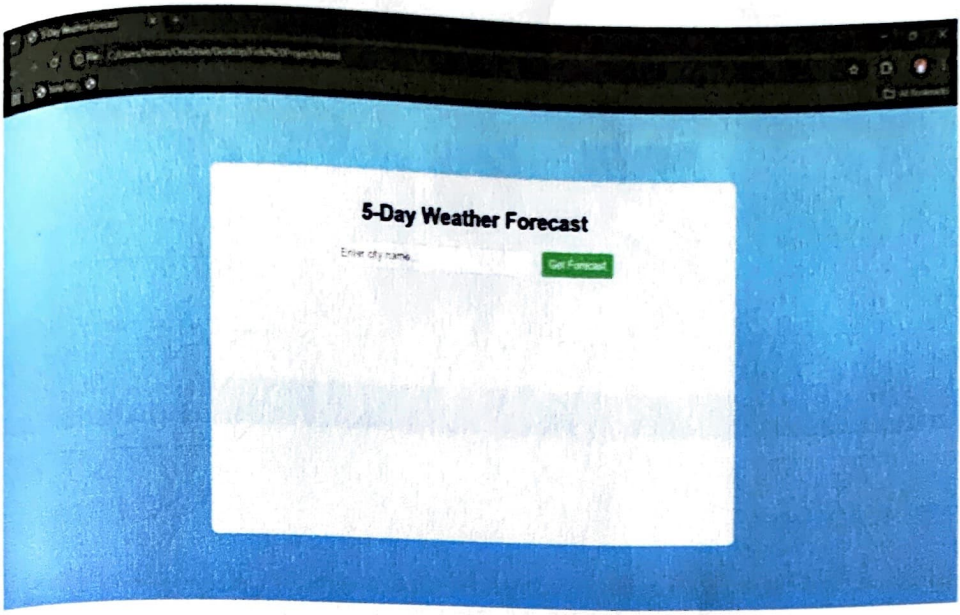
4.2 TEST CASES:

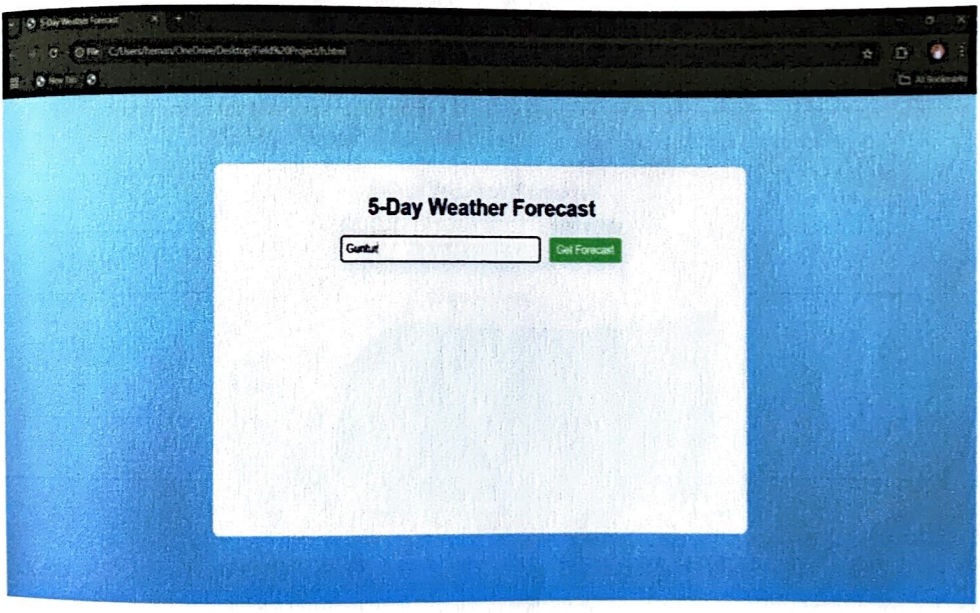
- **Test Case 1: Valid City Input**
  - **Input**: "London"
  - **Expected Output**: Weather data for London should be displayed (Temperature, Humidity, and Description).
- **Test Case 2: Invalid City Input**
  - **Input**: "FakeCity"
  - **Expected Output**: Display "City not found."
- **Test Case 3: Empty City Input**
  - **Input**: ""
  - **Expected Output**: Display "Please enter a city name."
- **Test Case 4: Network Error**
  - **Input**: Valid city (e.g., "Paris") with no internet connection
  - **Expected Output**: Display "Unable to fetch data. Please try again later."
- **Test Case 5: Correct Display of Weather Data**
  - **Input**: "New York"
  - **Expected Output**: Display temperature, humidity, and description for New York.

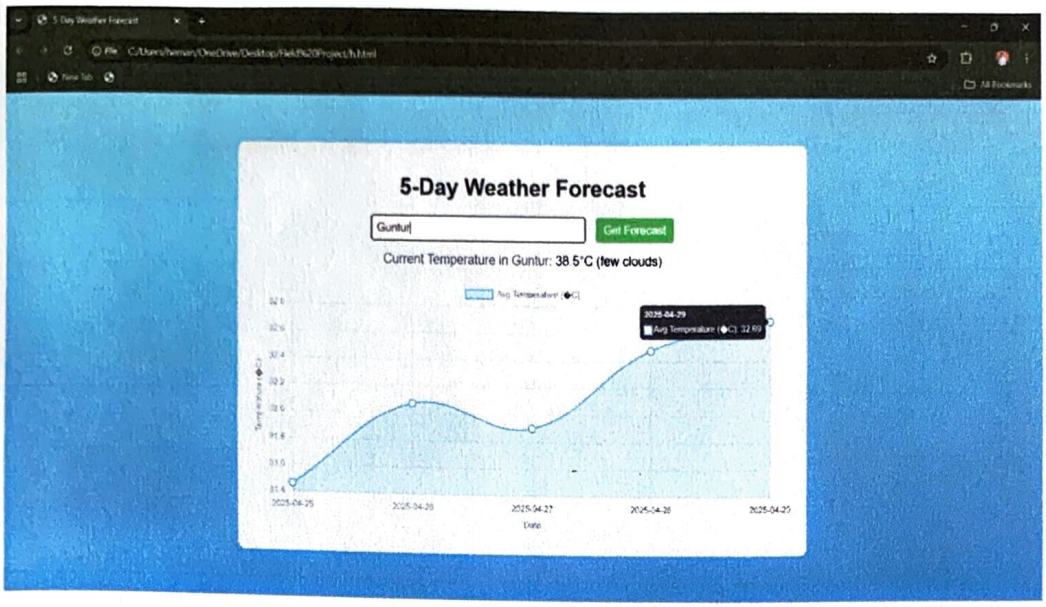
CHAPTER-05
RESULTS

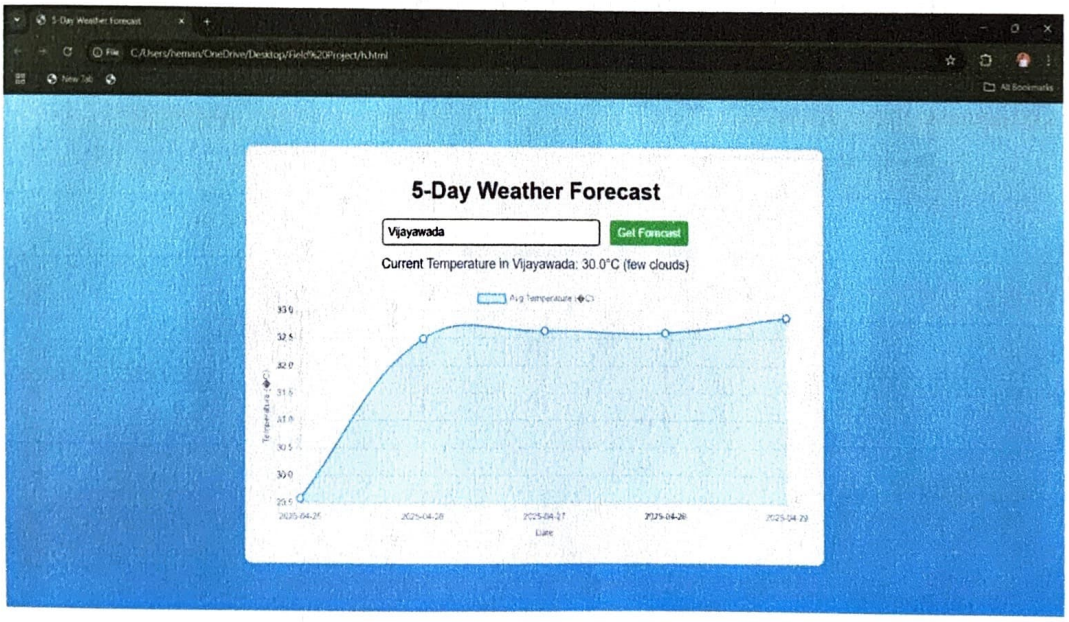
5. RESULTS

5.1 Output Screens:






5-Day Weather Forecast
Guntur
Get Forecast
Current Temperature in Guntur: 38.5°C (few clouds)
Avg Temperature (°C)
2025-04-29
Avg Temperature (°C): 32.89
Temperature (°C)
Date


5-Day Weather Forecast
Vijayawada
Get Forecast
Current Temperature in Vijayawada: 30.0°C (few clouds)
Avg Temperature (°C)
Temperature (°C)
Date

CHAPTER-06
CONCLUSION

6. CONCLUSION

In conclusion, the **Weather Forecast Web Application** provides a simple and efficient way to get real-time weather information for any city around the world. It allows users to input a city name and instantly view the temperature, humidity, and weather description.

Key Points:

- The app is **user-friendly**, with clear input fields and easy-to-read weather data.
- It provides a **responsive layout** that works seamlessly across devices.
- The app handles various errors, including invalid inputs and network issues, ensuring a smooth user experience.

Future Enhancements:

- Adding a **7-day forecast** feature to give users more detailed weather information.
- Implementing **geolocation** to fetch weather data based on the user's current location.
- Enhancing the **UI/UX design** with better visual elements and animations.

- **Location-based Weather**: Implement a feature that automatically detects the user's location and provides weather details for that area without the need for manual city input.
- **Detailed Forecast**: Extend the application to show a 5-day weather forecast or additional weather data (wind speed, visibility, etc.).
- **Graphical Representation**: Include charts or graphs to visualize the weather data, such as temperature trends over a period.
- **Progressive Web App (PWA)**: Convert the application into a PWA to allow offline use and enhance performance.

REFERENCES

1. Sharma, V., et al. "Web-based Real-time Weather Data Visualization." J. Atmospheric Sci. & Tech., 2024.
2. Gupta, R., and Menon, A. "Integrated 5-Day Weather Forecast Display on the Web." Int. J. Web Dev. & Design, 2025.
3. Khan, S., et al. "Responsive Design for Enhanced Weather App UX." J. Human-Computer Interaction Studies, 2023.
4. Reddy, L., and Das, P. "Error Handling in Client-Side Weather Forecasting." Int. J. Software Engg. & Apps., 2022.
5. Verma, M. "API Performance in Web Weather Applications." J. Network & Comm. Tech., 2024.

Project Link:

https://github.com/231FA04B79/Field-Project-Batch-01.git