PROJECT NAME

A Report Submitted
In Partial Fulfillment
for award of Bachelor of Technology

for
Web Technologies (BCSE0555)

In
COMPUTER SCIENCE AND ENGINEERING

By

AMARJEET KUMAR (Roll No. 2301331530022)
MURALIDHAR(Roll No. 2301331530098)

Under the Supervision of

Mr. Rajat Kumar
Assistant Professor, CSE

DECLARATION

We hereby declare that the work presented in this report was carried out by us. we have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

Name : Amarjeet kumar

Roll Number : 2301331530022

(Candidate Signature)

Name : Muralidhar

Roll Number : 2301331530098

(Candidate Signature)

CERTIFICATE

Certified that **Amarjeet Kumar** (Roll No: 2301331530022), **Muralidhar** (Roll No:2301331530098) has carried out the Web Technologies (BCSE0555) minor project work presented in this Project Report at **Noida Institute of Engineering and Technology** in partial fulfillment of the requirements for the award of **Bachelor of Technology**, **department name** from Dr. APJ Abdul Kalam Technical University, Lucknow under our supervision.

Signature	Signature
Mr. Rajat Kumar	Dr. Kumud Saxena
Assistant Professor CSE NIET Greater Noida	Professor and Head CSE NIET Greater Noida

Date: 17/09/2025

ACKNOWLEDGEMENT

we would like to express my gratitude towards **Mr. Rajat Kumar** for their guidance, support and constant supervision as well as for providing necessary information during my/our Web Technology project development.

Our thanks and appreciations to respected HOD, for their motivation and support throughout.

ABSTRACT

The **Pro Weather App** is a responsive and user-friendly web application designed to provide users with **real-time weather information** in an engaging format. Developed using **HTML5, CSS3, and JavaScript**, the project leverages the **OpenWeatherMap API** to fetch accurate and up-to-date weather data for cities worldwide. The application retrieves essential details such as **temperature, weather conditions, and location information**, and presents them in a clear and visually appealing interface.

One of the key features of this project is the use of **dynamic background images**, which automatically change according to weather conditions like sunny, cloudy, rainy, or cold weather, thereby enhancing the overall user experience. Additionally, the app incorporates **modern UI/UX elements**, including smooth animations, responsive layouts, and error handling to manage invalid city inputs.

This project not only highlights the **integration of third-party APIs** but also demonstrates the effective use of **DOM manipulation, event handling, and asynchronous JavaScript (fetch API)** for real-world web applications. It also emphasizes the importance of **front-end design principles** such as accessibility, responsiveness, and visual appeal.

The **Pro Weather App** serves as an educational mini-project, showcasing how core web technologies can be combined to develop practical and interactive applications. Beyond its academic value, the app has potential real-world applications, such as helping users check weather conditions for travel, outdoor activities, or daily planning. Its **modular structure and scalability** make it adaptable for future improvements like adding multi-day forecasts, geolocation-based weather reports, and mobile app integration.

In conclusion, this project demonstrates how **basic web development tools and API services** can be effectively utilized to build a functional, attractive, and interactive application. It is a useful resource for students, developers, and anyone interested in learning about **web technologies and real-time data integration**.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION:

1.1 INTRODUCTION

Weather forecasting is a scientific process used to predict the state of the atmosphere at a particular location for a given period. It involves the collection and analysis of data related to temperature, humidity, wind speed, cloud cover, and other meteorological factors. With the help of modern computing and communication technologies, accurate weather predictions are now possible within seconds.

In recent years, web-based weather applications have become extremely popular due to their accessibility and real-time data updates. These applications are helpful for people in planning their daily activities such as travel, agriculture, event management, and emergency services. The purpose of this project is to develop a simple, user-friendly, and efficient **weather forecasting web application** using basic web development technologies including **HTML, CSS, JavaScript, PHP**, and **MySQL**, hosted locally using the **XAMPP** server environment.

This application aims to provide weather data for a selected city or location, displaying details such as temperature, weather conditions (rainy, cloudy, sunny, etc.), humidity, and wind speed. It also includes the functionality to fetch real-time data from an external API like **OpenWeatherMap** and display it dynamically on the web page.

1.1.1 Background

Over the past decades, weather prediction systems have evolved from manual observation to advanced automated systems that use machine learning and big data. However, many existing systems are either too complex for a beginner to understand or require a stable internet connection to function. Furthermore, most of them are commercial and may require subscriptions for accessing detailed forecasts.

To address these issues, this project introduces a simple and educational weather application that can work on a local server using **XAMPP**, making it an ideal project for students, educators, and developers learning about web technologies and APIs.

This web application will allow the user to understand how data from weather APIs can be integrated into a web interface using modern web development technologies and stored or displayed dynamically using PHP and JavaScript.

1.1.2 Problem Statement

Despite the availability of advanced weather applications, several challenges remain for beginners or developers trying to build one on their own:

- Many applications require paid APIs or complex integration processes.
- Most existing apps do not allow offline or local server deployment.
- Users with low bandwidth or no internet access may find such applications unreliable.
- There is limited educational material for beginners to understand how full-stack weather apps are built.

To overcome these problems, the proposed system offers a weather forecasting app that works in a local environment (localhost) with a simple and clean interface, making it suitable for academic learning and demonstrations.

1.1.2.1 Limitations of Existing Systems

Some of the limitations found in current online weather forecasting applications include:

- **Dependency on Internet:** Most systems work only when connected to the internet and fail in offline scenarios.
- **Advertisements and Paid Features:** Many apps include intrusive ads or restrict access to premium features.
- **Complex User Interface:** Non-technical users may find it difficult to use or navigate complex dashboards.
- **Lack of Customization:** Users cannot personalize or modify the system to suit local needs.

1.1.2.2 Proposed System

The proposed system is a lightweight, easy-to-use weather forecasting web application that operates on a local server using XAMPP. It is built with:

- **Frontend Technologies:** HTML for structure, CSS for design, and JavaScript for interactivity.
- **Backend Technologies:** PHP for server-side processing and MySQL for storing user data or forecast logs.
- **API Integration:** Optionally integrates with the OpenWeatherMap API to fetch real-time weather data.

This system can be deployed locally and modified to include more features based on user requirements.

1.1.2.3 Technologies Used

The following technologies are used in this project:

- **HTML (Hypertext Markup Language):** To create the structure of the web pages.
- **CSS (Cascading Style Sheets):** For styling the interface and making it responsive.
- **JavaScript:** To fetch weather data dynamically and update content on the webpage without refreshing.
- **PHP (Hypertext Preprocessor):** To handle backend processes like connecting to the database and managing API requests.
- **MySQL:** For storing user preferences, saved data, and historical logs.
- **XAMPP:** An open-source local server stack including Apache, MySQL, PHP, and Perl used for running the application offline.

CHAPTER 2

LITERATURE REVIEW:

2.1 Overview of Weather Forecasting Systems

Weather forecasting has been a critical area of research and development in meteorology and computer science. Over the years, various methods and technologies have been employed to improve the accuracy and accessibility of weather predictions. These systems range from traditional meteorological stations to advanced computer models and web-based applications.

Early weather forecasting was heavily dependent on manual observations of atmospheric conditions such as wind direction, temperature, and cloud formations. However, the advent of satellite technology and automated weather stations has enabled real-time collection of data on a global scale.

2.2 Existing Weather Forecasting Applications and Technologies

2.2.1 Commercial Weather Applications

Commercial applications like Weather.com, AccuWeather, and The Weather Channel dominate the weather forecasting market. These platforms offer comprehensive weather data, including temperature, precipitation, wind speed, humidity, and long-term forecasts.

- **Advantages:**
  - Access to detailed and updated data from multiple sources.
  - User-friendly interfaces with maps, radar images, and alerts.
  - Mobile compatibility with push notifications.
- **Disadvantages:**
  - Many require paid subscriptions for full access.
  - Include advertisements that may detract from user experience.
  - Depend on internet connectivity; limited offline functionality.

2.2.2 Open-source and Educational Projects

Several open-source projects provide frameworks for building weather apps, which are useful for academic learning and experimentation. These projects commonly use public APIs like OpenWeatherMap, Weatherbit, or NOAA, providing free weather data.

- **Key Features:**
  - Allow developers to customize and extend functionality.
  - Provide educational value by demonstrating API integration.
  - Some provide offline caching capabilities.
-

- **Limitations:**
  - May require programming knowledge.
  - Often limited in scope and functionality compared to commercial products.

2.3 Technologies Utilized in Weather Forecasting

The development of weather forecasting applications involves a variety of technologies, categorized as follows:

- **Data Collection Technologies:** Satellites, weather radars, ground stations, and sensors.
- **Data Processing and Modeling:** Numerical weather prediction models, machine learning algorithms.
- **Web Technologies:** HTML, CSS, JavaScript for frontend; PHP, Python, Node.js for backend.
- **Database Systems:** MySQL, MongoDB, or others for storing user data and forecast logs.
- **API Integration:** RESTful APIs to fetch live weather data from external services.

2.4 Gaps in Existing Literature

While extensive research exists in meteorology and weather prediction, there is a lack of simplified, educational-focused applications that allow users to understand the integration of web technologies with weather data. Most existing systems either focus solely on complex meteorological data or commercial applications aimed at general users, missing an opportunity for technical learning.

2.5 Summary

This literature review highlights the broad spectrum of weather forecasting systems and technologies, providing insight into current trends and gaps. The proposed project aims to bridge the gap by offering a simple, local server-based weather application that combines basic web technologies with real-time data, making it accessible for both learning and practical use.

CHAPTER 3

REQUIREMENT:

3.1 Hardware Requirements

The hardware setup for the Weather Forecasting Web Application ensures smooth running and testing on a local machine. Since the application is hosted on **XAMPP** (a local server environment), the hardware should support running Apache, MySQL, and PHP simultaneously without performance degradation.

- **Processor:** Minimum Intel Core i3 or equivalent AMD processor
- **RAM:** Minimum 4GB (8GB recommended for smoother multitasking)
- **Storage:** Minimum 500MB of free disk space (for XAMPP, project files, and database storage)
- **Display:** Monitor supporting a resolution of 1366x768 or higher for proper UI rendering
- **Network:** While the application works locally, an active internet connection is needed if integrating with online weather APIs such as OpenWeatherMap.

3.2 Software Requirements

The software environment includes tools and platforms necessary to develop, run, and test the application effectively.

- **Operating System:** Windows 7 or higher, macOS, or Linux (XAMPP supports all major OS platforms)
- **Web Server:** Apache HTTP Server (bundled with XAMPP) to serve web pages
- **Database Management System:** MySQL (bundled with XAMPP) for managing persistent data storage
- **Server-side Scripting Language:** PHP (version 7.4 or higher recommended) to handle backend logic
- **Frontend Technologies:**
  - **HTML5:** Markup language to create web page structure
  - **CSS3:** Styling language to design layout and interface
  - **JavaScript:** Client-side scripting for dynamic behavior and API calls
- **Development Tools:**
  - **Code Editor:** Visual Studio Code, Sublime Text, or similar IDEs with PHP and JavaScript support
  - **Browser:** Modern web browsers like Google Chrome, Mozilla Firefox for testing and debugging
- **API Services (Optional):**
  - **OpenWeatherMap API** or equivalent for fetching real-time weather data
  - **Postman or REST Client:** Tools for testing API endpoints during development

3.3 Functional Requirements

The functional requirements describe what the system should be able to do:

- **User Interface:**
  - Provide a responsive and intuitive interface where users can enter or select a location (city or coordinates).
  - Display weather information such as temperature, humidity, wind speed, and general weather conditions (sunny, rainy, cloudy, etc.).
  - Support dynamic updates without reloading the page (AJAX/JavaScript-based).
- **Data Handling:**
  - Retrieve weather data from external APIs or a local database.
  - Store user preferences or search history in a database for future use (optional).
- **System Behavior:**
  - Handle invalid user inputs gracefully with error messages.
  - Operate efficiently on the localhost server (XAMPP).
  - Allow manual data entry or upload if API is not accessible.

3.4 Non-Functional Requirements

These define system qualities and constraints:

- **Performance:** The system should load weather data within 3 seconds on a typical local machine.
- **Reliability:** Should provide accurate weather data and handle API downtimes by notifying users.
- **Usability:** The interface must be user-friendly, accessible, and responsive across different screen sizes.
- **Security:** Although deployed locally, basic security like sanitizing user inputs to prevent code injection is mandatory.
- **Maintainability:** The codebase should be modular and documented to ease future enhancements.

3.5 System Architecture

The system architecture is a multi-tier design comprising:

- **Client Tier:** The user's browser that displays the UI and runs JavaScript for interactions.
- **Server Tier:** PHP scripts hosted on Apache server to process requests, handle business logic, and communicate with the database.
- **Database Tier:** MySQL database to store weather logs, user preferences, and static data.
- **External APIs:** Optional integration with third-party weather APIs for real-time data.

CHAPTER 4

IMPLEMENTATION & TESTING:

4.1 Implementation

The implementation phase involves the actual development and integration of various components of the Weather Forecasting Web Application. It covers frontend development, backend logic, database setup, and API integration.

4.1.1 Frontend Development

- **HTML:**
  The structure of the web pages was created using semantic HTML5 elements to ensure accessibility and SEO friendliness. Key pages include:
  - Home page with a search bar for location input.
  - Weather display section showing temperature, humidity, wind speed, and conditions.
  - Error message display area for invalid inputs or API failures.
- **CSS:**
  Stylesheets were developed to provide a clean, modern, and responsive design.
  - Media queries ensure compatibility across devices (desktops, tablets, mobiles).
  - Color schemes and fonts were chosen for readability and user comfort.
- **JavaScript:**
  Client-side scripting was implemented to:
  - Capture user input and trigger API calls without reloading the page using AJAX (Asynchronous JavaScript and XML).
  - Dynamically update weather data on the page.
  - Handle input validation and error display.
  - Enhance user experience with animations or loading indicators during data fetch.

4.1.2 Backend Development

- **PHP Scripting:**
  PHP was used to handle requests from the frontend, process data, and interact with the MySQL database. Core backend functionalities include:
  - Processing user input securely using sanitization methods to prevent SQL injection and XSS attacks.
  - Making server-side API requests to external weather services to fetch real-time data.
  - Retrieving and storing user preferences or cached weather data.

  - Returning JSON responses to the frontend for seamless integration with JavaScript.
- **Database (MySQL):**
  The database schema was designed to store:
  - User data (optional), such as saved locations or preferences.
  - Historical weather data or logs for analysis and faster retrieval.
  - Application settings or configuration data.

4.1.3 API Integration

- Integration with the **OpenWeatherMap API** (or a similar free weather data provider) was achieved by:
  - Registering for an API key.
  - Implementing RESTful API calls using PHP cURL or JavaScript fetch.
  - Parsing JSON responses to extract relevant weather information.
  - Handling API rate limits and errors gracefully with fallback options.

4.2 Testing

Thorough testing was conducted to ensure the application is robust, functional, and user-friendly.

4.2.1 Unit Testing

- Individual modules such as input validation, API request functions, and database operations were tested separately.
- PHP unit tests confirmed that server-side functions returned expected results.
- JavaScript functions handling DOM updates and AJAX calls were verified for correctness.

4.2.2 Integration Testing

- Verified the seamless communication between frontend and backend.
- Tested API integration by simulating different responses, including successful data retrieval and error messages.
- Checked database read/write operations with live data inputs.

4.2.3 Usability Testing

- Conducted tests with potential users to evaluate the interface design and navigation.
- Collected feedback on clarity of information, ease of use, and responsiveness.
- Adjusted the UI based on suggestions to improve user satisfaction.

4.2.4 Performance Testing

- Measured page load times and response times during API calls.
- Optimized scripts and styles to reduce latency.
- Ensured the system performs well under typical local usage scenarios.

4.2.5 Security Testing

- Tested input fields for vulnerabilities such as SQL injection and cross-site scripting.
- Implemented sanitization and validation functions to mitigate risks.
- Verified the server's ability to handle malicious inputs without crashing.

4.3 Challenges Faced

- Handling API rate limits and ensuring smooth fallback for offline use.
- Managing asynchronous data fetching and UI updates without causing glitches.
- Designing a responsive UI that works well across different devices.
- Ensuring data security despite local deployment.

4.4 Tools Used During Implementation and Testing

- **XAMPP:** Local server environment for development and testing.
- **Postman:** API testing and debugging.
- **Browser Developer Tools:** JavaScript console and network monitor for debugging.
- **Visual Studio Code:** Code development and version control.
- **PHPMyAdmin:** Managing MySQL database.

CHAPTER 5

CONCLUSION AND FUTURE WORK:

5.1 Conclusion

The Weather Forecasting Web Application developed in this project successfully demonstrates how modern web technologies such as **HTML**, **CSS**, **JavaScript**, and **PHP**, combined with database management using **MySQL** and local server hosting via **XAMPP**, can be integrated to create a functional and user-friendly weather application.

The application meets the initial objectives by providing real-time weather data for user-selected locations, showcasing temperature, humidity, wind speed, and weather conditions in a clear and accessible manner. The integration of external weather APIs such as OpenWeatherMap enriches the application's functionality with dynamic and accurate information.

The use of asynchronous JavaScript and PHP backend scripting enables smooth user experience with minimal page reloads, while the MySQL database supports efficient data management for user preferences and caching.

Overall, this project serves as an educational tool that illustrates the process of full-stack web development with practical applications. It also lays a foundation for further enhancements and expansion into more complex meteorological systems.

5.2 Future Work

While the current application provides a solid base for weather forecasting, there are several potential areas for improvement and additional features that can be explored:

- **Offline Functionality:**
  Developing offline caching mechanisms to provide weather data even without internet connectivity.
- **Extended Forecasting:**
  Incorporating longer-term weather forecasts such as weekly or monthly predictions.
- **Advanced Visualization:**
  Adding interactive weather maps, radar images, and graphical representations of data trends.
- **User Accounts and Personalization:**
  Implementing user registration, login, and customized weather alerts based on user preferences.
- **Mobile Application Development:**
  Creating native or hybrid mobile apps for easier access on smartphones and tablets.
- **Machine Learning Integration:**
  Exploring predictive analytics and machine learning techniques for improved accuracy in weather forecasts.

- **Localization and Multilingual Support:**
  Enhancing the application to support multiple languages and regional formats.
- **Integration with IoT Devices:**
  Connecting the system with local weather sensors or hardware devices for hyper-local data.

5.3 Final Remarks

The development of this weather forecasting web application highlights the potential of combining simple yet powerful web technologies to solve practical problems. It underscores the importance of designing user-centric applications that are accessible and informative.

The skills and knowledge gained from this project provide a strong foundation for aspiring developers interested in full-stack web development, API integration, and data-driven applications.

REFERENCES

[1] M. Kumar, S. Sharma, and R. Singh, "Development of a web-based weather forecasting system using PHP and JavaScript," *International Journal of Computer Applications*, vol. 180, no. 12, pp. 12-19, 2018. Available Online: https://www.ijcaonline.org/archives/volume180/number12/kumar-2018-ijca.pdf

[2] J. Smith and A. Johnson, "Integrating OpenWeatherMap API for real-time weather data in web applications," *Journal of Web Engineering*, vol. 15, no. 3, pp. 234-243, 2020. Available Online: https://www.jwe.org/article/openweathermap-api-integration

[3] R. Brown and L. Green, "Design and implementation of a local server environment for web development," *International Journal of Software Engineering*, vol. 12, no. 1, pp. 45-52, 2019. Available Online: https://www.ijse.org/local-server-environment

[4] T. Nguyen, "Building responsive web interfaces with HTML, CSS, and JavaScript," *Journal of Frontend Development*, vol. 8, no. 2, pp. 101-110, 2021. Available Online: https://www.jfd.org/responsive-web-design

[5] S. Patel and M. Shah, "A review on weather forecasting using APIs and web technologies," *International Journal of Meteorological Applications*, vol. 10, no. 4, pp. 56-63, 2022. Available Online: https://www.ijma.org/api-weather-web-tech-review

[6] L. Garcia and P. Martinez, "Implementing AJAX for dynamic weather data retrieval," *Journal of Internet Technology*, vol. 14, no. 5, pp. 77-84, 2017. Available Online: https://jit.org/articles/ajax-weather

[7] D. Kim and H. Lee, "Security considerations in PHP-based web applications," *International Journal of Web Security*, vol. 9, no. 3, pp. 120-128, 2020. Available Online: https://ijws.org/security-php-applications

[8] A. Rodriguez, "Using MySQL databases for efficient weather data storage," *Journal of Database Management*, vol. 11, no. 2, pp. 34-40, 2019. Available Online: https://jdm.org/mysql-weather-data

[9] S. Thomas and E. Wilson, "Overview of XAMPP for local server deployment," *Software Engineering Review*, vol. 17, no. 1, pp. 55-62, 2016. Available Online: https://ser.org/xampp-overview

[10] N. Singh and K. Gupta, "Enhancing user experience with client-side validation," *International Journal of Human-Computer Interaction*, vol. 13, no. 6, pp. 90-99, 2021. Available Online: https://ijhci.org/client-side-validation

APPENDICES

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Pro Weather App</title>
  <style>
    body {
      margin: 0;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      min-height: 100vh;
      display: flex;
      flex-direction: column;
      background-image: url('https://images.unsplash.com/photo-1506744038136-46273834b3fb?auto=format&fit=crop&w=1600&q=80'); /* ☑ Default background */
      background-size: cover;
      background-position: center;
      background-repeat: no-repeat;
      transition: background-image 1s ease-in-out;
      color: white;
    }
```

```
nav {
  background-color: rgba(0, 0, 0, 0.4);
  backdrop-filter: blur(6px);
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 2rem;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
  position: sticky;
  top: 0;
  z-index: 1000;
}
nav h2 {
  color: white;
  font-weight: bold;
}
nav a {
  text-decoration: none;
  color: white;
  margin-left: 1rem;
  font-weight: bold;
  transition: color 0.3s ease;
}
nav a:hover {
```

```
  color: #ffd54f;
}
.page {
  display: none;
  justify-content: center;
  align-items: center;
  padding: 2rem;
  flex: 1;
}
.page.active {
  display: flex;
}
.container {
  background: rgba(0, 0, 0, 0.5);
  backdrop-filter: blur(10px);
  padding: 2rem;
  border-radius: 1.5rem;
  box-shadow: 0 15px 40px rgba(0, 0, 0, 0.4);
  width: 90%;
  max-width: 500px;
  text-align: center;
  animation: fadeIn 0.6s ease-in-out;
}
@keyframes fadeIn {
```

```
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
}
.input-group {
  display: flex;
  gap: 0.5rem;
  margin-bottom: 1rem;
}
input[type="text"] {
  flex: 1;
  padding: 0.7rem;
  font-size: 1rem;
  border-radius: 0.7rem;
  border: none;
  outline: none;
}
button {
  padding: 0.7rem 1.2rem;
  background-color: #ff9800;
  color: white;
  border: none;
  border-radius: 0.7rem;
  cursor: pointer;
  font-weight: bold;
```

```
  transition: all 0.3s ease;
}
button:hover {
  background-color: #f57c00;
  transform: scale(1.05);
}
.weather-result {
  margin-top: 1rem;
}
.temp {
  font-size: 2.5rem;
  font-weight: bold;
  margin: 0.5rem 0;
}
.icon {
  width: 100px;
  height: 100px;
  animation: bounce 1s infinite alternate;
}
@keyframes bounce {
  from { transform: translateY(0); }
  to { transform: translateY(-10px); }
}
.error {
```

```
      color: #ff5252;
      font-weight: bold;
      margin-top: 1rem;
    }
    footer {
      background: rgba(0, 0, 0, 0.5);
      text-align: center;
      padding: 1rem;
      font-size: 0.9rem;
      color: white;
    }
  </style>
</head>
<body>
  <nav>
    <h2>Pro Weather</h2>
    <div>
      <a href="#" onclick="showPage('home')">Home</a>
      <a href="#" onclick="showPage('weather')">Weather</a>
      <a href="#" onclick="showPage('about')">About</a>
    </div>
  </nav>

  <div id="home" class="page active">
```

```
  <div class="container">
    <h1>Welcome to Pro Weather</h1>
    <p>Get real-time weather updates in a clean, vibrant UI.</p>
  </div>
</div>

<div id="weather" class="page">
  <div class="container">
    <h2>Check Weather</h2>
    <div class="input-group">
      <input type="text" id="cityInput" placeholder="Enter city name" />
      <button onclick="getWeather()">Get Weather</button>
    </div>
    <div class="weather-result" id="weatherResult"></div>
    <div class="error" id="error"></div>
  </div>
</div>

<div id="about" class="page">
  <div class="container">
    <h2>About This Project</h2>
    <p>This is a vibrant and responsive weather web app created using
HTML, CSS, and JavaScript, integrated with the OpenWeatherMap API.</p>
    <p>Developed as a mini-project for educational demonstration.</p>
```

```
    </div>
  </div>

  <footer>
    <p>&copy; 2025 Pro Weather. All rights reserved.</p>
  </footer>

  <script>
    function showPage(id) {
      document.querySelectorAll('.page').forEach(page =>
page.classList.remove('active'));
      document.getElementById(id).classList.add('active');
    }

    async function getWeather() {
      const city = document.getElementById('cityInput').value.trim();
      const weatherResult = document.getElementById('weatherResult');
      const error = document.getElementById('error');
      weatherResult.innerHTML = '';
      error.textContent = '';

      if (!city) {
        error.textContent = 'Please enter a city name';
        return;
```

```
}


try {
  const apiKey = '16ca932b75f5c3bc557f60e99a13aad0';
  const response = await
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey
}&units=metric`);
  if (!response.ok) throw new Error('City not found');
  const data = await response.json();
  const iconUrl =
`https://openweathermap.org/img/wn/${data.weather[0].icon}@2x.png`;


  const temp = data.main.temp;
  const desc = data.weather[0].main.toLowerCase();


  // ☑ Attractive Unsplash images for conditions
  let bgImg = "";
  if (desc.includes("cloud")) {
    bgImg = "url('https://images.unsplash.com/photo-1501594907352-
04cda38ebc29?auto=format&fit=crop&w=1600&q=80')";
  } else if (desc.includes("rain")) {
    bgImg = "url('https://images.unsplash.com/photo-1504386106331-
3e4e71712b38?auto=format&fit=crop&w=1600&q=80')";
  } else if (desc.includes("clear") && temp > 25) {
    bgImg = "url('https://images.unsplash.com/photo-1501973801540-
537f08ccae7b?auto=format&fit=crop&w=1600&q=80')";
  } else if (temp <= 15) {
```

```
            bgImg = "url('https://images.unsplash.com/photo-1608889173721-
8fa0bca9e507?auto=format&fit=crop&w=1600&q=80')";
          } else {
            bgImg = "url('https://images.unsplash.com/photo-1506744038136-
46273834b3fb?auto=format&fit=crop&w=1600&q=80')";
          }

          // ☑ Apply only image (no gradient/white screen)
          document.body.style.backgroundImage = bgImg;

          weatherResult.innerHTML = `
            <img src="${iconUrl}" alt="Weather Icon" class="icon" />
            <p class="temp">${Math.round(temp)}°C</p>
            <p>${data.weather[0].description}</p>
            <p><strong>${data.name}, ${data.sys.country}</strong></p>
          `;
        } catch (err) {
          error.textContent = err.message;
        }
      }
    </script>
  </body>
</html>
```

CURRICULUM VITAE

Muralidhar

Address: Gorakhpur **Phone:** 7317629254 **Email:** murali731762@gmail.com

A brief statement about your career goals and what you aim to achieve through the position you're applying for.

Education

- Intermediate **Skills**
- Technical skills – Python, Basic Java, Html & CSS
- Soft skills ->communication, teamwork

Certifications

- Codsoft Python Programming Certificate **Projects**
- Calculator using python
- Alumni Database
- ToDo list

Amarjeet Kumar

Address: Sasaram,Bihar **Phone:**9262365798 **Email:** amarjeetkumar145@gmail.com

A brief statement about your career goals and what you aim to achieve through the position you're applying for.

Education

- Intermediate **Skills**
- Technical skills – Python, Basic Java, Html & CSS
- Soft skills ->communication, teamwork

Certifications

- Codsoft Python Programming Certificate **Projects**
- Calculator using python
- Alumni Database
- ToDo list