KIET Group Of Institutions, Ghaziabad

Computer Science

Internship Report

On

Weather App

MLSA Internship

Aug-sept

(2024-2025)

Submitted By:

Name: Nandini Sharma

Course: B.tech (CS)

Sem/sec 2nd / 3-B

Roll No.: 2300290120145

ACKNOWLEDGEMENT

I've got this golden opportunity to express my kind gratitude and sincere thanks to my Head of Institution, KIET Group of Institutions of Engineering and Technology, and Head of Department of Computer Science for their kind support and necessary counselling in the preparation of this project report. I'm also indebted to each and every person responsible for the making up of this project directly or indirectly.

I must also acknowledge or deep debt of gratitude each one of my colleague who led this project come out in the way it is. It's my hard work and untiring sincere efforts and mutual cooperation to bring out the project work. Last but not the least, I would like to thank my parents for their sound counselling and cheerful support. They have always inspired us and kept our spirit up.

Name of Student: Nandini Sharma

Course and Branch: B.tech / Computer Science

Semester: 2nd

University Roll No: 2300290120145

CERTIFICATE

This is to certify that me Nandini Sharma of B.tech (CS) 2nd Semester from KIET Group Institution, Ghaziabad has presented this project work entitled “Weather App”, an online auction website in partial fulfilment of the requirements for the award of the degree of bachelor of Computer Applications under our supervision and guidance.

ABSTRACT

Weather prediction is the application of science and technology to predict the state of the atmosphere for a given location. Here this system will predict weather based on parameters such as temperature, humidity and wind. This system is a web application with effective graphical user interface. To predict the future's weather condition, the variation in the conditions in past years must be utilized. The probability that it will match within the span of adjacent fortnight of previous year is very high. We have proposed the use of linear regression for weather prediction system with parameters such as temperature, humidity and wind. It will predict weather based on previous record therefore this prediction will prove reliable. This system can be used in Air Traffic, Marine, Agriculture, Forestry, Military, and Navy etc.

INDEX FOR INTERNSHIP REPORT

Following essentials are to be taken in to consideration during preparation of internship report:

1. Acknowledgement
2. Certificate
3. Abstract
4. Introduction of Project Internship.
5. Details of task
6. Details of Technical learning during delivery of task
7. Conclusion
8. Future scope of work
9. Attach literature review report
10. Attach Daily Log report

INTRODUCTION

In the digital age, access to timely and accurate information is more critical than ever, particularly when it comes to weather data, which can significantly influence daily activities, travel plans, and even business operations. Recognizing this necessity, the Weather Web App project was initiated to create a dynamic and user-friendly application that provides real-time weather updates for users around the globe.

This project is particularly relevant in today's context, where climate patterns and weather conditions can change rapidly, affecting various sectors such as agriculture, transportation, and emergency management. By leveraging modern web technologies, the Weather Web App aims to empower users with instant access to relevant weather information, thereby facilitating informed decision-making.

The application was designed to be intuitive and straightforward, allowing users to simply input a city or location to receive up-to-date weather information. The main features include displaying current temperature, humidity, wind speed, and general weather conditions, enhanced with visually informative weather icons. The use of a responsive design ensures that the app functions seamlessly across various devices, including smartphones, tablets, and desktop computers, catering to the diverse needs of users.

To achieve this, the project employs a technology stack that includes HTML, CSS, and JavaScript, along with the Open Weather API for data retrieval. HTML forms the backbone of the application, structuring the layout and defining the various elements that users interact with. CSS is used to create an appealing user interface, ensuring that the application is not only functional but also visually attractive. JavaScript plays a pivotal role in bringing the app to life by handling user interactions, making API calls, and dynamically updating the UI with real-time data.

The choice of the Open Weather API is significant; it provides reliable and comprehensive weather data, allowing the application to display accurate information to users. By accessing this API, the Weather Web App can deliver updated weather conditions, which are essential for users seeking immediate information.

The development of the Weather Web App was driven by the desire to deepen my understanding of web development technologies, particularly in integrating APIs and managing asynchronous data requests. This project provided an opportunity to apply theoretical knowledge in a practical setting, leading to a better grasp of user interface design and the challenges involved in real-time data handling.

In summary, the Weather Web App project represents a blend of technology, design, and practical application, aiming to enhance the user experience in accessing weather information. As the demand for instant and reliable data continues to grow, the development of such applications is increasingly relevant, offering valuable insights and facilitating informed decision-making for users worldwide. This report details the methodology, technologies utilized, and the overall impact of the project, showcasing the process of bringing the Weather Web App from conception to completion.

METHODOLOGY

The project was developed over seven main phases, each focusing on a core part of the app's functionality and design:

1. **Project Planning and API Research**: Initial research was conducted to understand Open Weather API capabilities, including data endpoints and usage limitations. This phase included obtaining the API key and deciding which weather data (such as temperature, humidity, and wind conditions) would be displayed.
2. **HTML Structure Development**: A simple HTML structure was created, with input fields for user location search and placeholders for displaying weather data. Semantic HTML elements ensured a structured layout, improving readability and accessibility.
3. **CSS Styling and Responsive Design**: The layout was styled using CSS, with a focus on creating a visually appealing and responsive design. Flexbox was used to make the layout adaptable across devices, and additional styling included buttons, input fields, and background colors that adjust based on screen size.
4. **JavaScript Functionality**: JavaScript was used to connect with the Open Weather API and dynamically update the app's content based on user input. JavaScript functions retrieved weather data and updated HTML elements in real time. Error handling was also implemented to manage invalid location inputs and API response issues.
5. **API Integration with Fetch**: The `fetch` function in JavaScript was used to handle asynchronous API calls, retrieving real-time weather data from Open Weather. A sync/await syntax ensured smooth data loading, and JSON parsing was used to interpret the API response data.
6. **Data Presentation and UI Enhancements**: To make the app intuitive, various design enhancements were added, including weather icons and condition descriptions. Dynamic icons representing different weather states (sunny, rainy, cloudy) visually improved the user experience.
7. **Testing and Deployment**: The app was tested across various devices and browsers to ensure compatibility. The final app was then deployed on GitHub Pages, making it publicly accessible.

TECHNIQUES USED

HTML (Hypertext Mark-up Language)

- Used for creating the structure of the app.
- Elements such as , <h1>, <p>, and <input> were used to build the layout.
- Form elements were added to capture user input (e.g., search for a specific city or region).

CSS (Cascading Style Sheets)

- CSS was used to style the HTML structure, making the interface visually appealing and user-friendly.
- **Responsive Design**: Media queries were used to make the app adaptable to different screen sizes, ensuring mobile and desktop compatibility.
- **Animations and Transitions**: CSS transitions were used to add smooth effects, enhancing user experience.

JavaScript

- JavaScript played a central role in making the app interactive and dynamic.
- API Integration: JavaScript was used to make asynchronous HTTP requests to the Open Weather API using fetch() to retrieve live weather data based on user input.
- DOM Manipulation: JavaScript was used to manipulate the Document Object Model (DOM) to display the fetched weather data dynamically without refreshing the page.
- **Error Handling**: JavaScript error handling was implemented to manage cases where the API fails or when a user enters an invalid location.

Open Weather API

- The Open Weather API was used to obtain real-time weather data for a specified location.
- The API provides JSON responses containing weather details such as temperature, humidity, wind speed, and more.
- **API Key**: A unique API key was used to authenticate requests made to the Open Weather API.
- **Query Parameters**: By including query parameters (e.g., location, units), the API was configured to provide specific data in the desired format.

DETAIL OF TASK

1. Project Planning and Initial Setup

- **Define Requirements**: Identify features, including the main functionality of displaying current weather for a given location.
- **Research API**: Explore the Open Weather API documentation to understand available endpoints, required parameters, and data formats.
- **Design Wireframe**: Sketch a layout for the app interface, highlighting sections for location input, weather data display, and error handling.

2. User Interface Development (HTML & CSS)

- **HTML Structure**:
  - Create a basic HTML layout with key sections:
    - Header section with the app title.
    - Input section for users to enter the city name.
    - Display section for current weather data, including temperature, conditions, and an icon.
    - Optional: Footer with developer information or links.
- **CSS Styling**:
  - **Basic Styling**: Add basic styles to structure the layout, such as font styles, colors, and padding.
  - **Responsive Design**: Use media queries to adjust layout for mobile devices.
  - **Animation Effects**: Add CSS animations or transitions to enhance the user experience, such as hover effects on the search button or smooth transitions when displaying weather data.

3. API Integration and Data Fetching (JavaScript)

- **Connect to Open Weather API**:
  - Sign up on Open Weather, get an API key, and explore endpoints for current weather data.
  - Write JavaScript functions to fetch data based on user input.
- **Implement `fetch()` for API Requests**:
  - Create an asynchronous function using `fetch()` to make HTTP requests to the Open Weather API.
  - Use query parameters in the API call to specify location and units (e.g., metric or imperial).
- **Parse and Extract Data**:
  - Process JSON response from the API, extracting key details like temperature, humidity, weather description, and icons.

4. Dynamic Content Display (JavaScript & DOM Manipulation)

- **User Input Handling**:
  - Capture input from the search bar, validate it, and trigger the data fetch function when the user submits.
  - Clear the input field after search submission.
- **Update Weather Information**:
  - Dynamically update HTML elements with the weather data received from the API.
  - Insert weather icons, temperature, humidity, wind speed, and description in appropriate HTML elements.
- **Error Handling**:
  - Display user-friendly messages if the location is invalid, API fails, or no data is available.
  - Implement try-catch blocks and conditional checks to manage API errors.

5. Testing and Debugging

- **Cross-Browser Testing**:
  - Test the application on multiple browsers (e.g., Chrome, Firefox, Edge) to ensure compatibility.
- **Responsive Testing**:
  - Test on various screen sizes (mobile, tablet, desktop) to verify responsive behaviour.
- **Debugging**:
  - Use browser developer tools to debug JavaScript and CSS, checking for issues in data fetching, DOM manipulation, or style inconsistencies.

6. Final Touches and Deployment

- **Enhance Visuals**:
  - Add a suitable background image, such as a scenic or weather-related image, to make the app more visually appealing.
  - Customize UI elements like buttons, icons, and data sections to enhance aesthetics.
- **Optimize for Performance**:
  - Minimize HTTP requests and ensure efficient handling of JavaScript for smooth functionality.

DETAIL OF TECHNICAL LEARNING

1. Understanding API Integration

- **API Structure and Endpoints**: Learned how to navigate API documentation, focusing on endpoints, required parameters, and different response formats.
- **Working with API Keys**: Understood the importance of API keys for accessing third-party services securely and managed API key integration in JavaScript without exposing sensitive information in the frontend.
- **Handling HTTP Requests**: Used the fetch () method to make asynchronous HTTP requests. Learned about HTTP methods, particularly GET for data retrieval, and experimented with a sync/await syntax to manage asynchronous operations cleanly.
- **Parsing JSON Responses**: Became proficient in handling JSON data by parsing responses, extracting specific data points, and understanding how JSON objects and arrays are structured.

2. Frontend Development (HTML & CSS)

- **HTML Semantics**: Gained experience in building a structured HTML layout with semantic elements to improve readability and maintainability.
- **CSS Styling**:
    - **Responsive Design with Media Queries**: Practiced using media queries to create responsive layouts that adapt to different screen sizes, making the app accessible on both desktop and mobile.
    - **CSS Flexbox**: Learned to use Flexbox for aligning items, building grid structures, and managing spacing, which allowed for a more flexible and responsive layout.
    - **Styling for Visual Appeal**: Improved skills in applying colors, gradients, font choices, and icon integration to enhance the app's aesthetic appeal and UX.
- **Background Images**: Experimented with adding background images and handling them with CSS properties like background-size and background-position to ensure they display well across devices.

3. JavaScript Fundamentals and Advanced Concepts

- **Event Handling**: Practiced capturing user inputs and triggering functions on events (e.g., `click` event on a search button or `key down` event for enter key submission).
- **Error Handling in JavaScript**:
    - Applied error handling techniques with `try-catch` blocks to manage API request failures gracefully.
    - Used conditionals to display custom error messages for common issues like network errors or invalid inputs.
- **DOM Manipulation**:
    - Enhanced skills in dynamically updating the HTML content using JavaScript, making the app interactive without requiring page reloads.
    - Managed the visibility and styling of elements based on user input and API responses, giving users instant feedback.

4. Debugging and Testing

- **Console Debugging**: Used browser developer tools to monitor console logs, inspect network requests, and troubleshoot issues with JavaScript, CSS, and API responses.
- **Cross-Browser Testing**: Gained practical experience in testing the app across various browsers and devices to ensure consistent performance.
- **Mobile Responsiveness Testing**: Used emulators and physical devices to verify mobile compatibility, focusing on responsive design and user interactions on small screens.

5. Code Organization and Optimization

- **Modular JavaScript Functions**: Organized code into modular functions for readability and reusability. Divided tasks like data fetching, DOM updates, and error handling into separate functions to keep code clean.
- **Optimizing API Requests**: Limited unnecessary API calls by adding checks before making requests (e.g., ensuring non-empty input).
- **Reducing Reflows and Repaints**: Optimized DOM manipulation techniques to avoid excessive reflows and repaints, improving page performance.

6. Working with Real-Time Data

- **Handling Real-Time Data**: Learned to handle and display real-time data by updating the UI whenever new data is fetched from the API, making the app feel interactive and responsive.
- **Time and Data Formatting**: Encountered data like timestamps and learned to format it appropriately in JavaScript (e.g., converting Unix time to human-readable format).
- **Weather Data Interpretation**: Gained familiarity with weather-related data fields (temperature, humidity, wind speed) and displayed them in a user-friendly way.

CONCLUSION

The Weather Web App project was successful in demonstrating the integration of web development techniques with an external API. By using HTML, CSS, JavaScript, and the Open Weather API, this project showcases a dynamic and interactive user experience. Future enhancements could include adding additional features such as forecasts, weather maps, or temperature comparisons.

FUTURE SCOPE OF WORK

1. Extended Weather Forecasts

- **5- or 7-Day Forecast**: Integrate additional endpoints from the Open Weather API to display an extended forecast, providing users with weather predictions for the coming days.
- **Hourly Forecasts**: Add an hourly forecast option for detailed, short-term predictions, helping users plan their day with more accuracy.

2. Location-Based Features

- **Geolocation Integration**: Use the browser's Geolocation API to detect and display the user's current location's weather automatically, improving user convenience.
- **Nearby Weather Conditions**: Implement functionality to show weather conditions for nearby cities or landmarks, giving users a regional view.

3. Enhanced User Interface and UX

- **Dynamic Backgrounds**: Set up background images or animations that change based on weather conditions (e.g., rainy, sunny, cloudy) to make the app more visually engaging.
- **Dark Mode**: Implement a dark mode option for users to switch between light and dark themes, enhancing readability in various lighting conditions.
- **Accessibility Improvements**: Add features like screen reader support, keyboard navigation, and text-to-speech for visually impaired users.

4. Additional Weather Metrics

- **Air Quality Index (AQI)**: Integrate additional API services to display real-time air quality data, including pollutants and AQI levels.
- **Advanced Weather Details**: Display more detailed weather metrics, such as UV index, visibility, precipitation levels, and sunrise/sunset timings.

5. Data Visualization

- **Graphical Representation of Forecasts**: Use chart libraries like Chart.js or D3.js to create visual representations (e.g., temperature trends, precipitation chances) for a more intuitive data presentation.
- **Weather Map Integration**: Embed interactive maps (e.g., from OpenWeather's Weather Map) to display live weather patterns across different regions.

6. Notifications and Alerts

- **Severe Weather Alerts**: Implement notifications for extreme weather conditions like storms, heat waves, or heavy rainfall using data from the API or third-party sources.
- **Daily Weather Summaries**: Allow users to subscribe to daily or weekly weather summaries, providing a quick overview via notifications or emails.

7. User Accounts and Personalization

- **Saved Locations**: Add user accounts that allow users to save frequently searched locations for easy access.
- **Personalized Settings**: Allow users to set preferences, such as preferred temperature units (°C/°F) and language, and remember these settings for future visits.

8. Performance Optimization and Offline Support

- **Data Caching**: Implement caching strategies to store recent weather data and reduce API calls, improving app load times.
- **Offline Mode**: Use service workers to provide offline support, allowing users to access recently viewed weather information even without an internet connection.

9. AI-Powered Weather Insights

- **Weather Predictions Using AI**: Integrate machine learning models that provide localized weather predictions based on historical data and user patterns.
- **Personalized Insights**: Offer suggestions or insights based on weather trends, such as suggesting umbrellas on rainy days or sunscreen on sunny days.

LITERATURE REVIEW

Web-based weather applications provide users with up-to-date meteorological information through accessible platforms, often built using standard web technologies like HTML, CSS, and JavaScript. By integrating third-party APIs, these applications offer users a seamless experience in retrieving real-time weather data. The following literature review examines current research on the design, development, and user experience of web-based weather applications.

Web Technologies in Weather Applications

HTML, CSS, and JavaScript form the foundational technologies for most web applications, including weather apps, due to their flexibility and cross-platform compatibility. According to Kumar and Singh (2019), HTML provides the structure, CSS ensures visual appeal, and JavaScript handles interactivity and data processing, enabling the creation of dynamic and responsive applications that enhance user engagement. The study also highlights that HTML5 and CSS3 features, such as responsive grids and media queries, facilitate the development of user-friendly interfaces, which are particularly important for real-time applications like weather apps [1].

JavaScript and Asynchronous Data Retrieval

JavaScript, especially with the advent of ES6 and beyond, has become integral to handling asynchronous data retrieval from APIs. Mikkonen and Taivalsaari (2018) discuss the role of JavaScript frameworks and libraries, such as vanilla JavaScript, in simplifying API integration and DOM manipulation. The study emphasizes the importance of asynchronous functions, like `fetch()` with a sync/await, which enable real-time weather data retrieval without disrupting the user experience. For weather applications, this allows for seamless updates in response to user inputs [2].

API Integration for Real-Time Data

The integration of APIs, such as Open Weather API, is crucial for providing real-time weather updates in web applications. In a study on smart cities and web-based weather services, Ali and Imran (2020) discuss the significance of external APIs in retrieving accurate meteorological data and emphasize the API's role in ensuring the application's reliability and accuracy. The study notes that the adoption of weather APIs allows developers to access extensive datasets, enhancing the functionality and user experience of weather applications [3].

User Interface and User Experience Design

A well-designed interface is vital for web-based applications to ensure an intuitive and pleasant user experience. A review by Davis et al. (2021) focused on the role of UI/UX in weather apps, exploring how CSS frameworks and responsive design influence user

satisfaction. Their findings suggest that user-centric design elements, such as visually distinct icons, accessible font sizes, and animations, improve the user's ability to quickly interpret weather data. The study also identifies the importance of responsive design to make these apps functional across different devices and screen sizes [4].

5. Future Trends and Improvements

As web development technologies evolve, there is a growing interest in enhancing weather apps with advanced JavaScript libraries, data visualization techniques, and AI-driven personalization. Bhargava and Dhillon (2022) explore emerging trends in real-time applications, highlighting the use of frameworks such as D3.js and Chart.js for weather data visualization. They note that visual data representations, like charts or weather patterns, offer users a clearer understanding of complex data, and incorporating these techniques into weather apps could provide more value to users [5].

DAILY LOG

Day 1: Project Setup and Planning

- **Tasks Completed**:
  - Defined project goals and main features (e.g., real-time weather display, location-based search).
  - Explored the Open Weather API documentation and obtained an API key.
  - Created a rough sketch and wireframe for the app interface.
- **Challenges**:
  - Initial setup and API key configuration required careful reading of documentation.
- **Outcome**: Clear project plan and API integration requirements.

Day 2: Basic HTML Structure

- **Tasks Completed**:
  - Built the basic HTML structure, including:
    - Search input field for location.
    - Placeholder sections for displaying temperature, humidity, wind speed, and weather icon.
  - Added semantic tags to organize content clearly.
- **Challenges**:
  - None significant; setup of HTML structure was straightforward.
- **Outcome**: Functional HTML skeleton of the app prepared.

Day 3: Styling with CSS

- **Tasks Completed**:
  - Designed the layout using CSS, focusing on readability and responsiveness.
  - Implemented Flexbox for a responsive layout, especially for mobile screens.
  - Experimented with background images and styling elements like buttons and icons.
- **Challenges**:
  - Ensuring responsive design required some trial and error with media queries.
- **Outcome**: Basic styling and responsive layout implemented.

Day 4: JavaScript Setup and Initial API Integration

- **Tasks Completed**:
  - Connected JavaScript to handle the search form submission.
  - Integrated the Open Weather API using `fetch()` to retrieve weather data for user-entered locations.

    - Created a function to log data to the console, testing the API request functionality.
- **Challenges**:
    - Debugged initial API request issues (e.g., handling CORS and errors).
- **Outcome**: Basic API functionality confirmed; data successfully fetched from Open Weather.

Day 5: Displaying Data in the UI

- **Tasks Completed**:
    - Parsed the API response and updated HTML elements dynamically with weather information.
    - Implemented DOM manipulation to display temperature, weather conditions, and other details.
    - Added icons representing weather conditions, retrieved from the API.
- **Challenges**:
    - Mapping API data to HTML elements required careful error handling.
- **Outcome**: Weather data displayed dynamically in the UI upon user search.

Day 6: Error Handling and UI Improvements

- **Tasks Completed**:
    - Added error handling for incorrect city names or failed API calls, displaying friendly error messages.
    - Enhanced the UI with CSS animations (e.g., button hover effects, loading animations).
    - Tested and adjusted layout to improve UX on various devices.
- **Challenges**:
    - Implementing user-friendly error messages required additional JavaScript checks.
- **Outcome**: App more user-friendly with error handling and enhanced UI.

Day 7: Testing, Debugging, and Deployment

- **Tasks Completed**:
    - Conducted cross-browser and mobile testing to ensure compatibility.
    - Fixed minor bugs in responsive design and API data handling.
    - Deployed the app on GitHub Pages, making it publicly accessible.
- **Challenges**:
    - Minor deployment issues with GitHub Pages required troubleshooting.
- **Outcome**: Fully functional, deployed weather app accessible online.

REFERENCES

1. Kumar, A., & Singh, R. (2019). "Developing Responsive Web Applications using HTML5, CSS3, and JavaScript." *International Journal of Web Applications*, 11(2), 34-41.
2. Mikkonen, T., & Taivalsaari, A. (2018). "Web Applications as Universal Applications." *Software: Practice and Experience*, 48(7), 1417-1431.
3. Ali, F., & Imran, M. (2020). "Smart Cities and Web-Based Weather Services: Role of APIs in Real-Time Data Retrieval." *Journal of Smart Technology and Applications*, 4(3), 55-67.
4. Davis, E., Smith, L., & Brown, J. (2021). "UI/UX Best Practices for Weather Applications." *Journal of Interactive Media and Technology*, 9(1), 22-28.
5. Bhargava, P., & Dhillon, K. (2022). "Data Visualization in Real-Time Web Applications: A Case Study on Weather Apps." *Journal of Emerging Web Technologies*, 15(2), 102-113.