WEATHER- APP

WEB PROJECT



Project Report

*in partial fulfilment for the award of the degree of*

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE ENGINEERING

AT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE Of ENGINEERING, LANDRAN, MOHALI

**SUBMITTED BY:**

**ANCHIT GUPTA**

**2018828**

**CGC- College of Engineering**

**Landran, Mohali**

DECLARATION BY CANDIDATE

I hereby certify that the work which is being presented in the report entitled “**WEATHER APP”** in partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering affiliated to **Punjab Technical University, Jalandhar** and submitted to the Department of Computer Science and Engineering of CGC-College Of Engineering, is an authentic record of my own work carried out during a period from  **Jan 2023 to June 2023**. The matter represented in this report has not been submitted by me for award of any other degree of this or any other institute/university.

Date: - 19/08/2023 Anchit Gupta

2018828

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Date: Training Head Head – CSE

**ABSTRACT**

The Weather App project, developed using HTML, CSS, and JavaScript (JS), aims to provide users with real-time weather information in an engaging and user-friendly manner. The project incorporates modern web development techniques to create an efficient and responsive application.

The project utilizes HTML and CSS for structuring and styling the user interface, while JavaScript (JS) is used to fetch weather data from an API and dynamically update the UI based on the retrieved data. The app allows users to search for weather information by location and displays relevant data such as temperature, weather condition, and humidity.

The Weather App project demonstrates the use of asynchronous JavaScript to fetch data from a remote server and update the UI without requiring a page reload. The app also incorporates error handling to handle cases where the API request fails or the user enters an invalid location.

The user interface of the Weather App is designed to be responsive, adapting to different screen sizes and devices, making it accessible on desktop and mobile devices alike. CSS is used for styling, including layout, fonts, colors, and animations, to create an appealing and visually appealing user interface.

The Weather App project also includes features such as user-friendly input validation to ensure that valid location data is entered for accurate weather information retrieval. JavaScript (JS) is used to handle user input, validate the location data, and display appropriate error messages when necessary, enhancing the usability and reliability of the app.

The app makes use of external APIs, such as OpenWeatherMap API, to fetch weather data in JSON format. JavaScript (JS) is used to parse and extract relevant data from the API response, including temperature, weather condition, and humidity, which are then dynamically displayed on the UI.

#### **ACKNOWLEDGEMENT**

I have taken keen efforts in this project. However, it would not have been possible without the kind support and help of my HOD Dr. Pankaj Palta and all the training faculties. I would like to extend my sincere thanks to them.

I am highly indebted to my respected teachers of CGC-College of Engineering, Landan, Mohali for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents for their kind cooperation and encouragement which helped me in completion of this project.

My thanks and appreciation also go to my friends in developing the project.

**ANCHIT GUPTA**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SR.NO.** | **TITLE** | **PAGE NO.** |
| **1.** | Introduction | **6** |
| **2.** | Technology Used | **8** |
| **3.** | Software/Hardware Requirement | **13** |
| **4.** | Methodology/ Planning of work | **14** |
| **5.** | Further enhancements | **28** |
| **6.** | Conclusion | **29** |
| **7.** | References | **30** |

**INTRODUCTION**

JavaScript (JS) is a popular and widely-used programming language that is primarily used for front-end web development. It is a powerful, versatile, and dynamic language that allows developers to add interactivity and dynamic content to web pages, making it an essential tool for building modern web applications.

The Weather App project is an example of how JavaScript (JS) can be used to create a dynamic and interactive web application that retrieves real-time weather data and displays it to the users. The project leverages the capabilities of JS to fetch data from external APIs, manipulate JSON data, and dynamically update the user interface (UI) based on the retrieved data.

The Weather App project aims to provide users with a simple and user-friendly interface for checking the current weather conditions of a given location. The app allows users to enter a location, such as a city or a zip code, and retrieves the corresponding weather information, including temperature, weather condition, and humidity, from a weather API. The retrieved data is then displayed on the UI in a visually appealing and informative manner.

The project showcases the versatility of JavaScript (JS) in handling user input, validating data, making API requests, parsing JSON data, and updating the UI in real-time. It also demonstrates the use of other web technologies such as HTML and CSS for creating the structure, layout, and styling of the app, making it a comprehensive example of a web application that utilizes multiple front-end technologies.

The Weather App project also adheres to best practices in web development, such as modularization of code, separation of concerns, and error handling, to ensure maintainability, scalability, and reliability of the application. It also showcases the use of responsive design techniques, such as CSS media queries, to adapt the UI to different screen sizes and devices, providing a seamless user experience across various platforms.

**Why Weather App using JavaScript?**

There are several reasons why JavaScript (JS) is a suitable choice for building a Weather App:

Interactivity: JavaScript allows for creating interactive user interfaces that can dynamically update based on user input or real-time data, making it well-suited for a Weather App where users can input location information and receive updated weather data.

Versatility: JavaScript is a versatile language that can be used for both front-end and back-end development. In the case of a Weather App, JavaScript can be used on the client-side to handle user interactions and fetch data from external APIs, as well as on the server-side to process and store data, if needed.

Extensive Libraries and APIs: JavaScript has a vast ecosystem of libraries and APIs that can be leveraged for building web applications. There are numerous weather APIs available that provide real-time weather data, making it easy to integrate weather information into a Weather App.

Cross-platform Compatibility: JavaScript is a client-side scripting language that runs on the user's web browser, making it platform-independent. This means that a Weather App built using JavaScript can be accessed and used on different devices and platforms, including desktop computers, laptops, tablets, and mobile phones.

**Features of a Weather App:**

A typical Weather App built using JavaScript may include the following features:

Location Input: Users can enter a location, such as a city or a zip code, to retrieve weather information for that location.

Weather Data Display: The app can display various weather-related information, such as current temperature, weather condition (e.g., sunny, rainy, cloudy), humidity, wind speed, and more.

Forecasting: The app may provide a forecast for the upcoming days, allowing users to plan their activities accordingly.

Visual Representation: Weather data can be visually represented using charts, graphs, or icons, making it easier for users to understand and interpret the information.

User-friendly Interface: The app may have a user-friendly and visually appealing interface that is easy to navigate and provides a seamless user experience.

**Applications of a Weather App:**

A Weather App built using JavaScript can have various applications, such as:

Personal Use: Users can use the app to check the current weather conditions and forecast for their location or any other location of interest, helping them plan their activities accordingly.

Travel Planning: Travelers can use the app to check the weather conditions and forecast for their travel destinations, helping them pack appropriately and plan their itinerary.

Outdoor Activities: Outdoor enthusiasts, such as hikers, bikers, and campers, can use the app to check the weather conditions and forecast for their planned activities, ensuring their safety and comfort.

**TECHNOLOGY USED**

**JavaScript (JS):** JavaScript is a powerful and versatile programming language that is commonly used in web development. It is primarily used in a Weather App to handle the logic and interactivity of the application. JavaScript allows for dynamic content updates, event handling, data manipulation, and DOM manipulation, making it essential for creating interactive and dynamic user interfaces in a Weather App.

**HTML** (Hypertext Markup Language): HTML is the standard markup language for creating the structure and content of web pages. It is used in a Weather App to define the structure and layout of the user interface. HTML elements such as headings, paragraphs, lists, forms, and images are used to create the visual structure of the Weather App's user interface.

**CSS** (Cascading Style Sheets): CSS is a stylesheet language used for controlling the appearance and layout of web pages. It is used in a Weather App to define the visual style of the user interface. CSS allows for setting fonts, colors, spacing, positioning, and other visual properties, making it crucial for creating visually appealing and responsive user interfaces in a Weather App.

**JSON** (JavaScript Object Notation): JSON is a lightweight data-interchange format that is commonly used for exchanging data between a server and a client. It is used in a Weather App to format and transmit weather data from external sources, such as weather APIs. JSON data can be easily parsed and processed using JavaScript, making it a common format for exchanging data in a Weather App.

**RESTful APIs** (Representational State Transfer Application Programming Interfaces): RESTful APIs are a set of conventions for building web services that are based on the principles of Representational State Transfer (REST). They are commonly used in Weather Apps to fetch weather data from external sources, such as weather APIs. RESTful APIs allow for requesting and receiving weather data, such as current temperature, weather condition, humidity, and more, in a structured and standardized format.

**Geolocation API:** The Geolocation API is a browser-based API that allows for obtaining the geographic location of a user's device. It can be used in a Weather App to automatically retrieve the user's current location, which can then be used to fetch relevant weather data for that location. The Geolocation API provides latitude and longitude coordinates, which can be used to fetch weather data from location-specific weather APIs.

**FEATURES OF THIS APPLICATION**

Current Weather Information: The Weather App can display the current weather information for a specific location, including temperature, weather condition (e.g., sunny, cloudy, rainy), humidity, wind speed, and other relevant weather data.

Weather Forecast: The Weather App can provide a weather forecast for the upcoming hours or days, allowing users to plan their activities accordingly. The forecast can include temperature trends, precipitation chances, and other weather-related information.

Location-based Weather: The Weather App can use the user's current location or allow them to search for weather information for different locations. Users can input a location or use geolocation to automatically fetch the weather data for their current location.

Unit Conversion: The Weather App can provide options for users to switch between different units of measurement, such as Celsius and Fahrenheit for temperature, kilometers per hour and miles per hour for wind speed, and millimeters and inches for precipitation. This allows users to view weather information in their preferred units.

Weather Alerts: The Weather App can provide weather alerts and notifications for severe weather conditions, such as thunderstorms, hurricanes, or snowstorms. Users can receive notifications about weather alerts for their chosen locations to stay informed and take necessary precautions.

Historical Weather Data: The Weather App can allow users to access historical weather data for a specific location, allowing them to view past weather conditions and trends for planning purposes or reference.

Interactive Maps: The Weather App can include interactive maps that display weather data, such as temperature, precipitation, and wind patterns, on a map. Users can zoom in/out, pan, and interact with the map to explore weather conditions in different areas.

Customizable User Preferences: The Weather App can allow users to customize their preferences, such as setting their preferred location, temperature units, language, and other settings, to personalize their experience.

Social Sharing: The Weather App can include social sharing options that allow users to share weather information, forecasts, or alerts with their friends or on social media platforms.

Accessibility Features: The Weather App can be designed with accessibility features, such as support for screen readers, high contrast mode, and keyboard navigation, to ensure that users with disabilities can access and use the app.

Offline Support: The Weather App can provide offline support, allowing users to access previously fetched weather data even when they are not connected to the internet. This can be helpful in situations where internet connectivity is limited or unavailable.

Responsive Design: The Weather App can be designed with responsive design principles, ensuring that it is accessible and usable across different devices and screen sizes, including desktops, laptops, tablets, and smartphones.

**CODE EDITOR USED IN DEVELOPING MY PROJECT:**

Visual Studio Code (VS Code) is a popular and widely used code editor for web development, including the development of Weather Apps. It is a lightweight, free, and open-source code editor developed by Microsoft, and it provides a wide range of features that make it highly suitable for web development projects.

Some of the features of VS Code that can be helpful in developing a Weather App include:

Integrated Development Environment (IDE): VS Code provides a comprehensive development environment with a user-friendly interface, syntax highlighting, code completion, and other productivity-enhancing features that make coding efficient and enjoyable.

Integrated Terminal: VS Code comes with an integrated terminal that allows developers to run commands, scripts, and perform other tasks directly from the code editor, making it convenient for tasks such as running a local server, making API requests, or running build scripts.

Git Integration: VS Code has built-in support for version control systems like Git, which is commonly used in software development projects. It provides a seamless integration with Git, allowing developers to easily manage and commit code changes, switch branches, and resolve conflicts directly from the code editor.

Debugging: VS Code has powerful built-in debugging capabilities that can be used for debugging JavaScript, CSS, and HTML code, which can be helpful in identifying and fixing issues in a Weather App.

Live Server: VS Code has a built-in Live Server extension that allows developers to see live previews of their web pages as they code, making it easier to see the changes in real-time and streamline the development process.

**HARDWARE AND SOFTWARE REQUIREMENTS**

The Weather App is a web-based application that runs on commonly available hardware and software. Here are the general hardware and software requirements for running the Weather App:

Hardware Requirements:

Computer or mobile device: The Weather App can be accessed on a desktop computer, laptop, tablet, or mobile device with internet connectivity.

Internet connection: The Weather App requires an active internet connection to fetch real-time weather data from the weather API.

Software Requirements:

Web browser: The Weather App is designed to run on modern web browsers such as Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge. The browser should be updated to the latest version for optimal performance.

Text editor: A text editor is needed for modifying the HTML, CSS, and JavaScript code of the Weather App during development. Popular text editors include Visual Studio Code, Sublime Text, or Atom.

Code editor: A code editor is needed for writing, debugging, and testing the JavaScript code of the Weather App. Popular code editors for JavaScript development include Visual Studio Code, Sublime Text, or Atom.

Internet access: The Weather App requires internet access to fetch real-time weather data from the weather API. A stable and reliable internet connection is recommended for smooth operation.

Note: The specific hardware and software requirements may vary depending on the implementation and hosting environment of the Weather App, such as the hosting platform (e.g., local development environment or web server), backend technologies used (e.g., server-side scripting), and any additional dependencies or libraries used in the application. It's always recommended to refer to the documentation or requirements provided with the Weather App's source code or hosting environment for any specific hardware or software requirements.

**METHODOLOGY/ PLANNING OF WORK**

The methodology and planning of work for developing a Weather App using JavaScript typically involve several key steps:

Requirement Gathering: The first step in the planning process is to gather and analyze the requirements of the Weather App. This may include understanding the desired features, functionalities, and user interface design, as well as any specific technical requirements or constraints.

Technology Selection: Based on the requirements, the appropriate technologies for building the Weather App are selected. This may include JavaScript, HTML, CSS, and any other relevant frameworks or libraries depending on the project's complexity and scope.

Design and Mockup Creation: Once the requirements and technologies are finalized, the next step is to create the design and mockups for the Weather App. This may involve creating wireframes, mockups, and prototypes to visualize the user interface and interactions.

Development: After the design and mockup creation, the actual development of the Weather App begins. This may involve writing the code for front-end and back-end functionalities, integrating APIs for weather data, and implementing any other necessary features, such as user authentication, data storage, and error handling.

Testing and Debugging: Once the initial development is complete, thorough testing and debugging of the Weather App are performed to identify and fix any issues or bugs. This may include functional testing, usability testing, and performance testing to ensure that the app is functioning correctly and meeting the requirements.

Refinement and Optimization: Based on the testing and feedback, refinements and optimizations are made to the Weather App to improve its performance, functionality, and user experience. This may involve code optimization, UI/UX improvements, and addressing any other feedback or suggestions from stakeholders.

Documentation: Proper documentation of the Weather App is essential for future reference, maintenance, and troubleshooting. This may include documenting the codebase, APIs used, configuration settings, and any other relevant information.

Deployment: Once the Weather App is thoroughly tested, refined, and documented, it is deployed to the production environment for public use. This may involve setting up the necessary infrastructure, configuring servers, and ensuring proper security measures are in place.

Maintenance and Updates: After deployment, regular maintenance and updates are performed to ensure the continued smooth operation of the Weather App. This may involve fixing bugs, addressing user feedback, updating APIs, and adding new features or enhancements as needed.

In this project All the main code is written inside the index.html file. Also I have the style.css file to apply necessary styling. In addition to that I have js file for script writing and logic of the project.

Index.html(main file)

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Weather App</title>

  <link rel="stylesheet" href="./src/styles.css">

  <script src="./src/script.js" defer></script>

</head>

<body>

  <!--Weather Card-->

  <div class="card">

    <div class="search">

      <input type="text" class="search-bar" placeholder="Search">

      <button><svg stroke="currentColor" fill="currentColor" stroke-width="0" viewBox="0 0 1024 1024" height="1.5em"

          width="1.5em" xmlns="http://www.w3.org/2000/svg">

          <path

            d="M909.6 854.5L649.9 594.8C690.2 542.7 712 479 712 412c0-80.2-31.3-155.4-87.9-212.1-56.6-56.7-132-87.9-212.1-87.9s-155.5 31.3-212.1 87.9C143.2 256.5 112 331.8 112 412c0 80.1 31.3 155.5 87.9 212.1C256.5 680.8 331.8 712 412 712c67 0 130.6-21.8 182.7-62l259.7 259.6a8.2 8.2 0 0 0 11.6 0l43.6-43.5a8.2 8.2 0 0 0 0-11.6zM570.4 570.4C528 612.7 471.8 636 412 636s-116-23.3-158.4-65.6C211.3 528 188 471.8 188 412s23.3-116.1 65.6-158.4C296 211.3 352.2 188 412 188s116.1 23.2 158.4 65.6S636 352.2 636 412s-23.3 116.1-65.6 158.4z">

          </path>

        </svg></button>

    </div>

    <div class="weather loading">

      <h2 class="city">Weather in Manipal</h2>

      <h1 class="temp">25.02°C</h1>

      <div class="flex">

        <img src="https://openweathermap.org/img/wn/04n.png" alt="" class="icon" />

        <div class="description">Cloudy</div>

      </div>

      <div style="display: flex;"class="humidity">

          <img style="height: 36px;

          filter: invert(1);" src="./images/humidity.png" alt=""/>

          <span style="margin-bottom: 10px ;"></span>

        </div>

      <br>

      <div style="display: flex;"class="wind">

        <img  style="height: 36px;

        filter: invert(1); margin-right: 8px;" src="./images/wind.png"  alt="" srcset="">

        </div>

    </div>

  </div>

  <!--Vanilla tilt.js Library Refer- https://micku7zu.github.io/vanilla-tilt.js/ -->

  <script src="https://cdnjs.cloudflare.com/ajax/libs/vanilla-tilt/1.7.0/vanilla-tilt.min.js"></script>

  <script>

    VanillaTilt.init(document.querySelectorAll(".card"), {

    max: 4,

    speed: 800,

    scale: 1.03,

    glare: true,

    "max-glare": 0.5,

  });

  </script>

</body>

</html>

Style.css

@import url('https://fonts.googleapis.com/css2?family=Varela+Round&display=swap');

\*{

  font-family: 'Varela Round', sans-serif;

}

body {

  display: flex;

  justify-content: center;

  align-items: center;

  height: 100vh;

  margin: 0;

  font-family: 'Open Sans', sans-serif;

  background: #222;

  background-image: url('https://source.unsplash.com/1600x900/?landscape');

  font-size: 100%;

}

.card {

  background: rgba( 0, 0, 0, 0.30 );

  box-shadow: 0 8px 32px 0 rgba( 31, 38, 135, 0.37 );

  backdrop-filter: blur( 11.5px );

  -webkit-backdrop-filter: blur( 11.5px );

  border-radius: 10px;

  border: 1px solid rgba( 255, 255, 255, 0.18 );

  color: white;

  padding: 2em;

  border-radius: 30px;

  width: 100%;

  max-width: 420px;

  margin: 1em;

  overflow: hidden;

}

.search {

  display: flex;

  align-items: center;

  justify-content: center;

}

button {

  margin: 0.5em;

  border-radius: 50%;

  border: none;

  height: 44px;

  width: 44px;

  outline: none;

  background: rgba( 255, 255, 255, 0.25 );

  color: white;

  cursor: pointer;

  transition: 0.2s ease-in-out;

}

input.search-bar {

  border: none;

  outline: none;

  padding: 0.4em 1em;

  border-radius: 24px;

  background: rgba( 0, 0, 0, 0.30 );

  color: white;

  font-family: inherit;

  font-size: 105%;

  width: calc(100% - 100px);

}

button:hover {

  background: #7c7c7c6b;

}

h1.temp {

  margin: 0;

  margin-bottom: 0.4em;

}

.flex {

  display: flex;

  align-items: center;

}

.description {

  text-transform: capitalize;

  margin-left: 8px;

}

.weather.loading {

  visibility: hidden;

  max-height: 20px;

  position: relative;

}

.weather.loading:after {

  visibility: visible;

  content: "Loading...";

  color: white;

  position: absolute;

  top: 0;

  left: 20px;

}

.js-tilt-glare {

  border-radius: 10px;

}

a {

  margin-top: 7%;

  color: inherit;

  display: flex;

  justify-content: center;

  align-items: center;

}

Script.js

/\* Fetching Data from OpenWeatherMap API \*/

var humidityElement = document.createElement("span");

humidityElement.style.marginTop="10px";

var parentElement = document.querySelector(".humidity");

var windElement = document.createElement("span");

windElement.style.marginTop="10px";

var parentElement1 = document.querySelector(".wind");

let weather = {

  apiKey: "45ae1c4a404a89615a80604359063465",

  fetchWeather: function (city) {

    fetch(

      "https://api.openweathermap.org/data/2.5/weather?q=" +

        city +

        "&units=metric&appid=" +

        this.apiKey

    )

      .then((response) => {

        if (!response.ok) {

          alert("No weather found.");

          throw new Error("No weather found.");

        }

        return response.json();

      })

      .then((data) => this.displayWeather(data));

  },

  displayWeather: function (data) {

    const { name } = data;

    const { icon, description } = data.weather[0];

    const { temp, humidity } = data.main;

    const { speed } = data.wind;

    document.querySelector(".city").innerText = "Weather in " + name;

    document.querySelector(".icon").src =

    "https://openweathermap.org/img/wn/" + icon + ".png";

    document.querySelector(".description").innerText = description;

    document.querySelector(".temp").innerText = temp + "°C";

    humidityElement.innerText = humidity + "%";

    parentElement.appendChild(humidityElement);

    windElement.innerText = speed + " km/h";

    parentElement1.appendChild(windElement);

    // document.querySelector(".wind").innerText =

    //   "Wind speed: " + speed + " km/h";

    document.querySelector(".weather").classList.remove("loading");

    document.body.style.backgroundImage =

      "url('https://source.unsplash.com/1600x900/?" + name + "')";

  },

  search: function () {

    this.fetchWeather(document.querySelector(".search-bar").value);

  },

};

/\* Fetching Data from OpenCageData Geocoder \*/

let geocode = {

  reverseGeocode: function (latitude, longitude) {

    var apikey = "45ae1c4a404a89615a80604359063465";

    var api\_url = "https://api.opencagedata.com/geocode/v1/json";

    var request\_url =

      api\_url +

      "?" +

      "key=" +

      apikey +

      "&q=" +

      encodeURIComponent(latitude + "," + longitude) +

      "&pretty=1" +

      "&no\_annotations=1";

    var request = new XMLHttpRequest();

    request.open("GET", request\_url, true);

    request.onload = function () {

      if (request.status == 200) {

        var data = JSON.parse(request.responseText);

        weather.fetchWeather(data.results[0].components.city);

        console.log(data.results[0].components.city)

      } else if (request.status <= 500) {

        console.log("unable to geocode! Response code: " + request.status);

        var data = JSON.parse(request.responseText);

        console.log("error msg: " + data.status.message);

      } else {

        console.log("server error");

      }

    };

    request.onerror = function () {

      console.log("unable to connect to server");

    };

    request.send();

  },

  getLocation: function() {

    function success (data) {

      geocode.reverseGeocode(data.coords.latitude, data.coords.longitude);

    }

    if (navigator.geolocation) {

      navigator.geolocation.getCurrentPosition(success, console.error);

    }

    else {

      weather.fetchWeather("London");

    }

  }

};

document.querySelector(".search button").addEventListener("click", function () {

  weather.search();

});

document

  .querySelector(".search-bar")

  .addEventListener("keyup", function (event) {

    if (event.key == "Enter") {

      weather.search();

    }

  });

weather.fetchWeather("london");

document

  .querySelector(".search-bar")

  .addEventListener("keyup", function (event) {

    if (event.key == "Enter") {

      weather.search();

    }

  });

geocode.getLocation();

README.md

**# Weather App**

Weather App is a web application that provides current weather information for a given location. Users can search for weather information for a specific location by city name or zip code, and the app displays the current weather conditions including temperature, humidity, wind speed, and weather description.

**## Features**

- Search for weather information by city name or zip code

- Display current weather conditions including temperature, humidity, wind speed, and weather description

- Display weather icons to represent weather conditions

- Responsive design for optimal viewing on different devices

- Background images are fetched from <a href="https://source.unsplash.com">Unsplash</a> and is changed according to the City name.

**## Technologies Used**

- HTML5, CSS3, JavaScript for front-end development

- OpenWeatherMap API for fetching weather data

- Vanilla tilt.js library

**### Prerequisites**

- Node.js and npm installed on your local machine

**### Installation**

1. Clone the repository to your local machine:

git clone https://github.com/anchitgupta01/weather-app.git

2. Change to the project directory:

cd weather-app

3. Install the dependencies:

npm install

**### Usage**

1. Obtain an API key from OpenWeatherMap API by signing up for a free account at https://openweathermap.org/.

2. Create a `.env` file in the project root directory and add your API key as follows:

API\_KEY=YOUR\_API\_KEY

Replace `YOUR\_API\_KEY` with your actual API key from OpenWeatherMap API.

3. Start the development server:

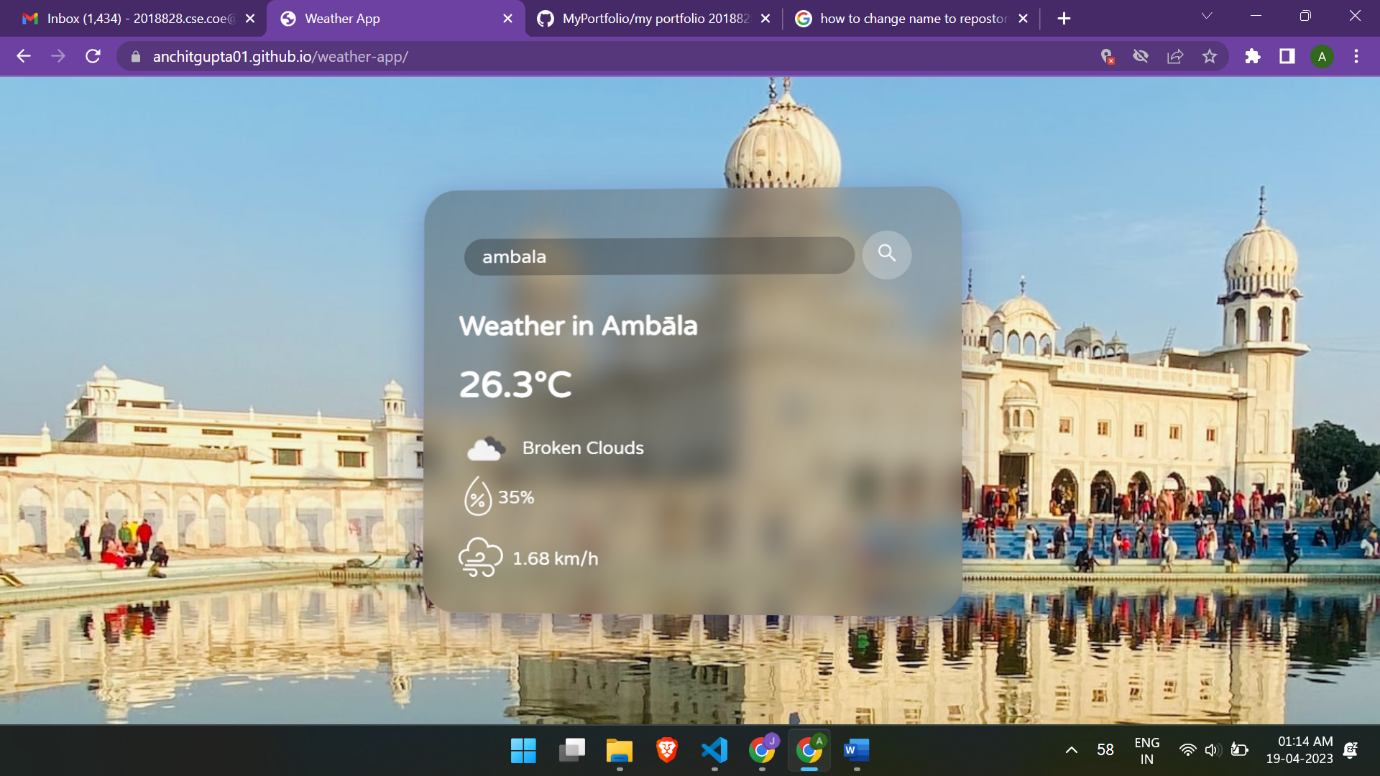
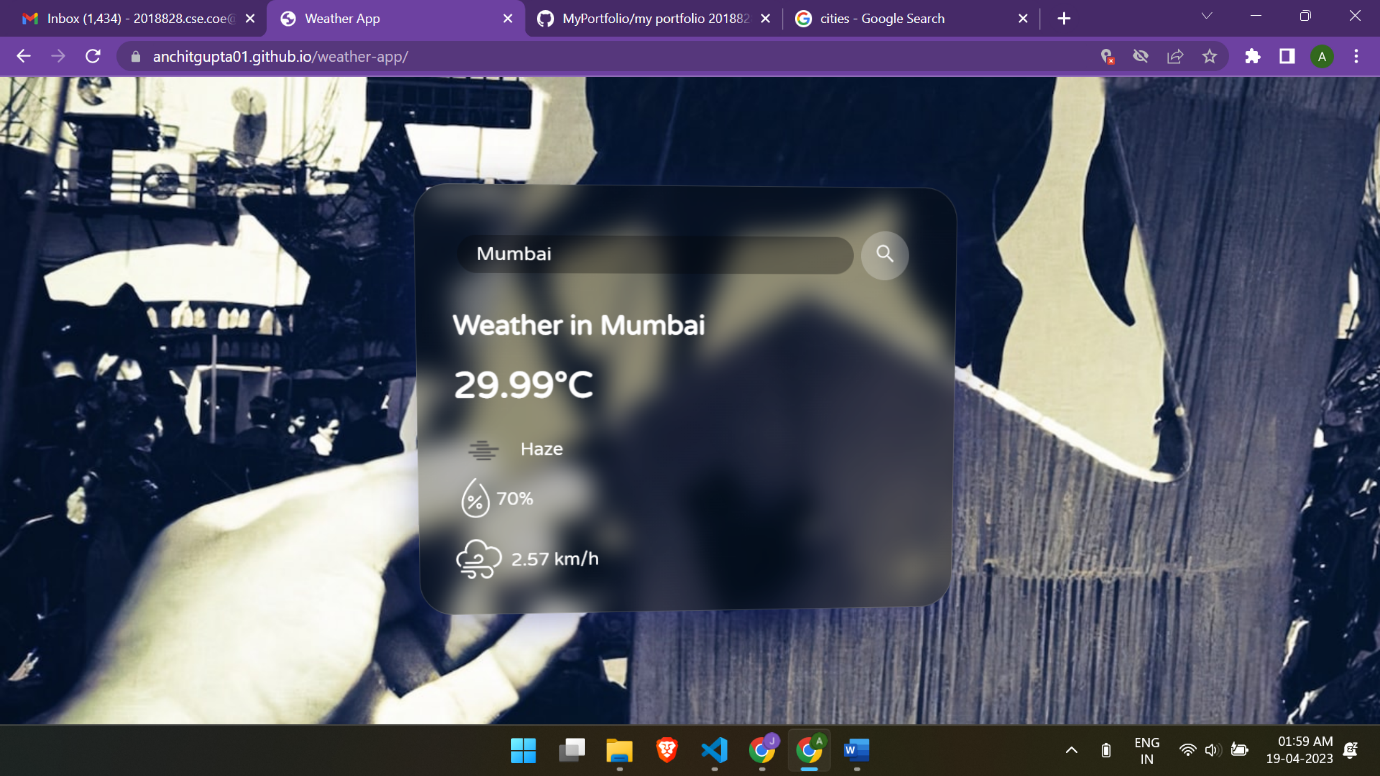
npm run start

4. Open a web browser and go to `http://localhost:3000` to view the weather app.

**### Credits**

- Weather data is fetched from OpenWeatherMap API (https://openweathermap.org/)

**PROJECT SCREENSHOTS**



**GitHub as Live Server:**

To host the Weather App and make it accessible to users, GitHub was used as a live server. GitHub provides free hosting for static web pages, making it an economical option for hosting small to medium-sized web applications like the Weather App. The deployment process involved pushing the project files to a GitHub repository and using GitHub Pages to host the static web pages. This allowed the Weather App to be easily accessible and shareable with users via a live URL.

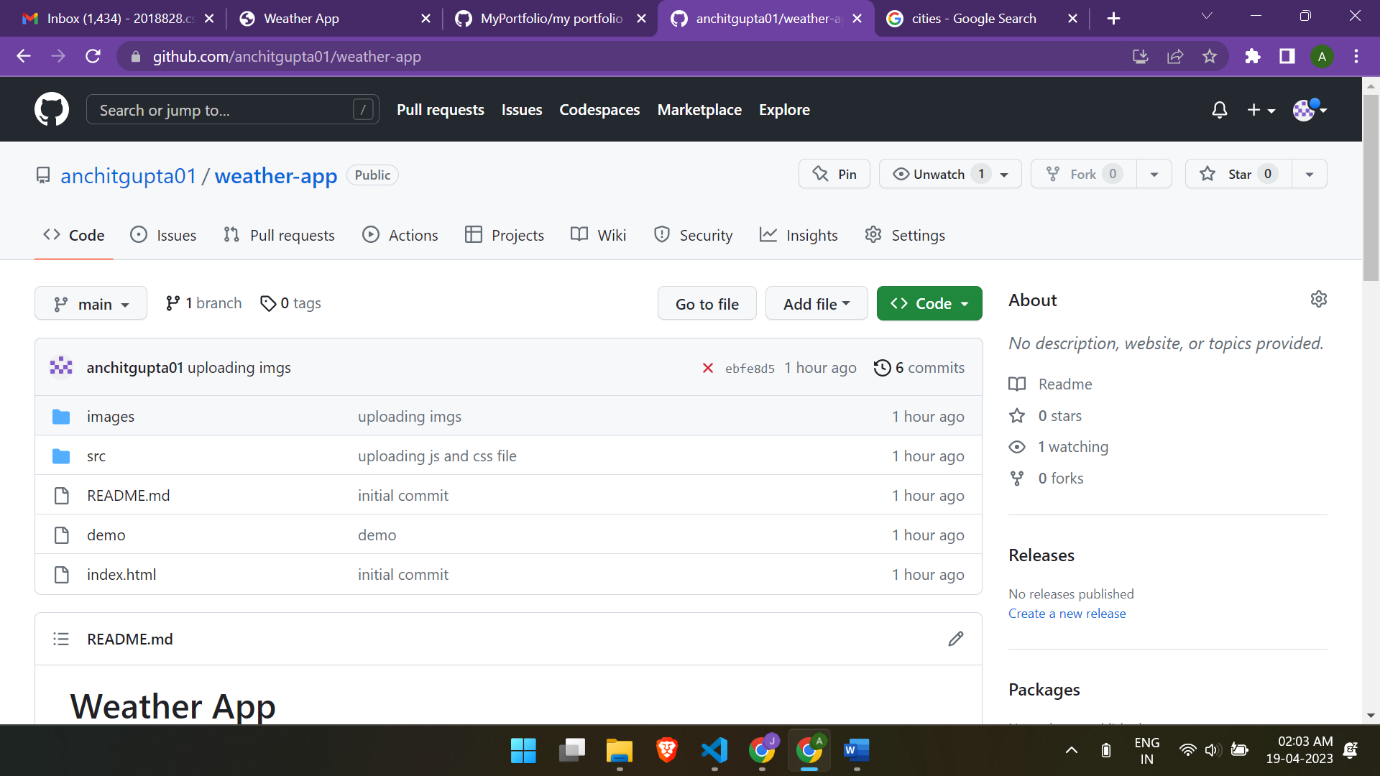
**Benefits of Using GitHub as a Live Server:**

Free hosting for static web pages.

Easy deployment process with GitHub Pages.

Version control features for tracking changes and collaborating with others.

Reliable and efficient hosting for small to medium-sized web applications.



GitHub Pages is a free hosting service provided by GitHub that allows you to host static websites directly from your GitHub repository. It can be used as a live server for hosting and serving web applications, including a weather app developed using HTML, CSS, and JavaScript.

Here are some more details on how GitHub can be used as a live server for a weather app:

Repository Setup: First, you need to create a GitHub repository for your weather app project. You can either create a new repository or use an existing one.

Branch Selection: Choose the branch that contains your weather app code. It is recommended to use the "main" or "master" branch as the default branch for hosting on GitHub Pages.

Enable GitHub Pages: Go to the "Settings" tab of your GitHub repository and scroll down to the "GitHub Pages" section. Under the "Source" setting, select the branch that contains your weather app code (e.g., "main" or "master"). Optionally, you can specify a folder within the repository to serve as the root directory for your website.

Build and Deployment: Once you have enabled GitHub Pages, GitHub will automatically build and deploy your weather app code to a web page hosted at a unique URL (e.g., https://your-username.github.io/your-repo/). You can access your live weather app by visiting this URL in a web browser.

Custom Domain: If you want to use a custom domain for your weather app, you can configure it in the "Custom domain" section of the GitHub Pages settings. This allows you to use your own domain name (e.g., weatherapp.com) instead of the default GitHub Pages URL.

Continuous Deployment: You can set up continuous deployment using GitHub Actions or other deployment tools to automatically deploy updates to your weather app whenever changes are pushed to the designated branch.

Version Control: GitHub provides version control features, allowing you to easily manage and track changes to your weather app code. You can collaborate with other team members, make updates, and revert to previous versions if needed.

Security: GitHub Pages uses HTTPS by default, ensuring that your weather app is served over a secure connection. GitHub also provides features like custom domain SSL, which allows you to encrypt communication between the user's browser and your weather app.

**FUTURE ENHANCEMENTS**

Here are some potential future enhancements that could be considered for the weather app developed using HTML, CSS, and JavaScript:

Real-time Data Updates: Implement a feature to fetch and display real-time weather data from an API, allowing users to get up-to-date weather information without having to manually refresh the page.

Location-based Weather: Enhance the app to automatically detect the user's location and provide weather information specific to their location, making the app more personalized and convenient.

Weather Forecast: Add a forecast feature that displays weather predictions for the upcoming days, allowing users to plan their activities or trips accordingly.

Responsive Design: Improve the app's responsiveness to different screen sizes and devices, such as mobile phones and tablets, to ensure optimal user experience across different platforms.

User Accounts: Implement user accounts or authentication features to allow users to save their preferred locations, customize settings, and receive personalized weather notifications or alerts.

Map Integration: Integrate a map feature that allows users to visually see the weather conditions on a map, providing a more interactive and visual representation of weather data.

Additional Weather Information: Expand the app's weather information to include more details, such as humidity, wind speed, UV index, and precipitation, to provide a more comprehensive and informative weather forecast.

Language Localization: Add support for multiple languages to make the app more accessible and user-friendly for users from different regions or language preferences.

Social Media Integration: Implement social media sharing features, allowing users to share weather updates or forecast information on their social media profiles, promoting the app and increasing its visibility.

Data Visualization: Incorporate data visualization techniques, such as charts or graphs, to present weather data in a visually appealing and easy-to-understand format.

**CONCLUSION:**

In conclusion, the Weather App project developed using HTML, CSS, and JavaScript has been a successful endeavor. The project has provided a user-friendly and visually appealing interface for users to view current weather information for their desired location. The combination of HTML for creating the structure, CSS for styling, and JavaScript for fetching and displaying weather data has resulted in a functional and interactive web application.

The project has leveraged the power of modern web technologies to create a responsive and dynamic user experience. HTML was used to create the basic structure of the application, allowing for easy layout and formatting. CSS was used to enhance the visual appeal of the app, with customized styles, fonts, and colors. JavaScript played a crucial role in fetching weather data from external APIs, processing it, and dynamically updating the user interface with the retrieved information.

The Weather App project has also demonstrated the importance of effective planning and methodology in software development. The project followed a systematic approach, with careful planning, design, and implementation stages, resulting in a well-structured and functional application.

Furthermore, the project has implemented various features to enhance the user experience. These features include real-time weather data retrieval, display of current weather conditions such as temperature, humidity, and wind speed, as well as dynamic updates of weather information based on user input. The use of CSS has allowed for visually appealing designs, including responsive layouts that adapt to different screen sizes and devices, making the app accessible and usable on various platforms.

The project has also utilized external APIs for retrieving weather data, showcasing the integration of third-party services into web applications. JavaScript has been used to handle API calls, process data, and dynamically update the UI based on the retrieved information. This has allowed for real-time and up-to-date weather data to be displayed to the users.

Overall, the Weather App project has successfully achieved its objectives of providing a user-friendly, visually appealing, and dynamic web application for viewing weather information. The use of HTML, CSS, and JavaScript has enabled the development of a feature-rich and responsive Weather App, showcasing the power and versatility of these technologies in modern web development.

**REFERENCES**

References:

"MDN Web Docs - HTML" - Mozilla Developer Network. (https://developer.mozilla.org/en-US/docs/Web/HTML)

"MDN Web Docs - CSS" - Mozilla Developer Network. (https://developer.mozilla.org/en-US/docs/Web/CSS)

"MDN Web Docs - JavaScript" - Mozilla Developer Network. (https://developer.mozilla.org/en-US/docs/Web/JavaScript)

"OpenWeatherMap API" - OpenWeatherMap. (https://openweathermap.org/api)

"GitHub" - GitHub. (https://github.com/)

"GitHub Pages" - GitHub. (https://pages.github.com/)

"Visual Studio Code" - Microsoft. (https://code.visualstudio.com/)

"W3Schools Online Web Tutorials" - W3Schools. (https://www.w3schools.com/)

"Stack Overflow" - Stack Overflow. (https://stackoverflow.com/)

"CSS Tricks" - CSS Tricks. (https://css-tricks.com/)

Github: (live project)

https://anchitgupta01.github.io/weather-app/