December 31, 2024

muhammad saim ayyaz - 221009  
Wasi haider raja - 221039  
abdul babar - 221021  
bilal chattha - 221103

WEB PROJECT

WEB TECHNOLOGY PROJECT REPORT

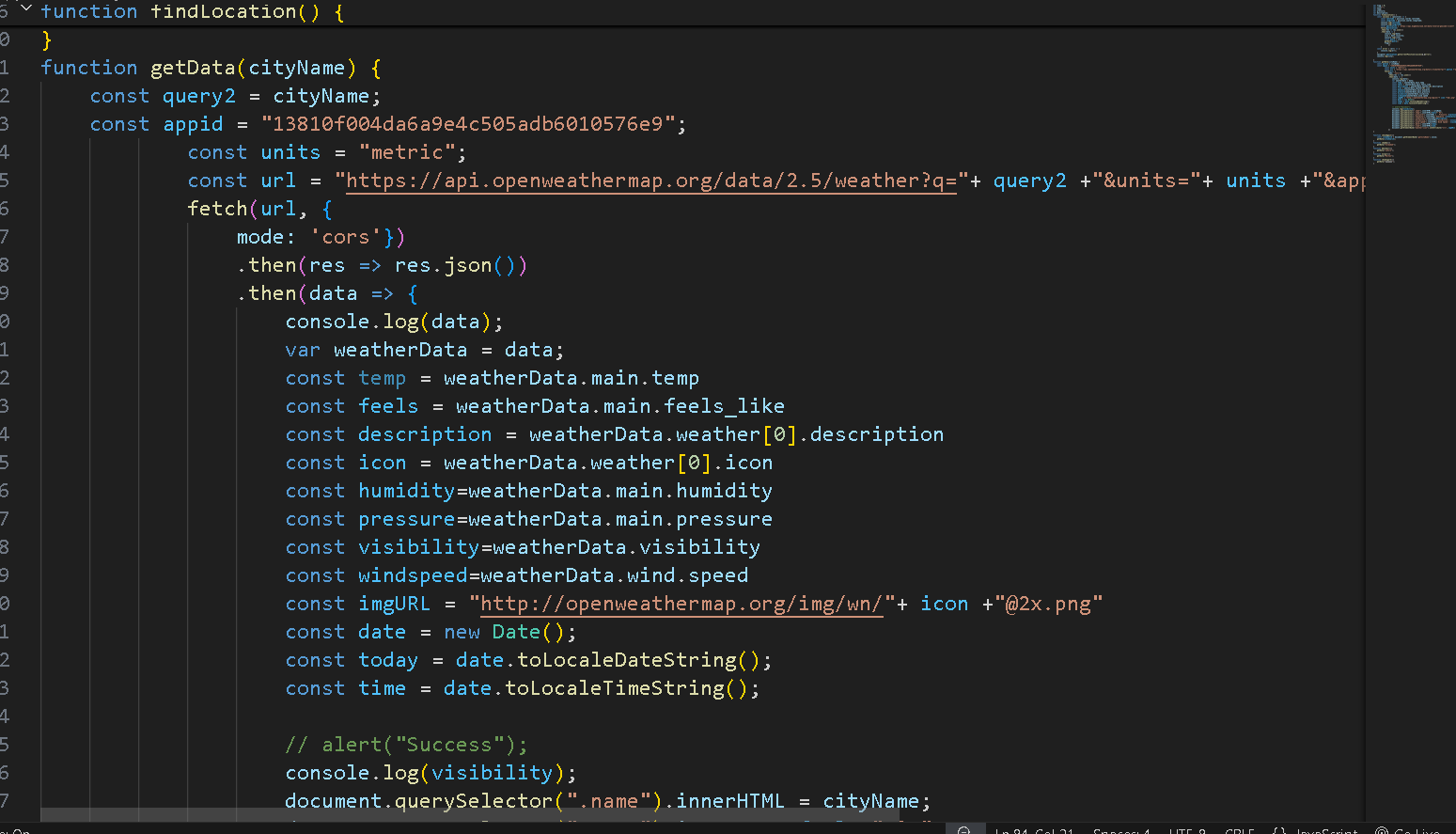
WEATHER FORECAST APP

DESCRIPTION:

In this project we have used html,css,for front end and css for backend.This is a weather forecast system in which we have used time,and weather of cities.

APP.JS:





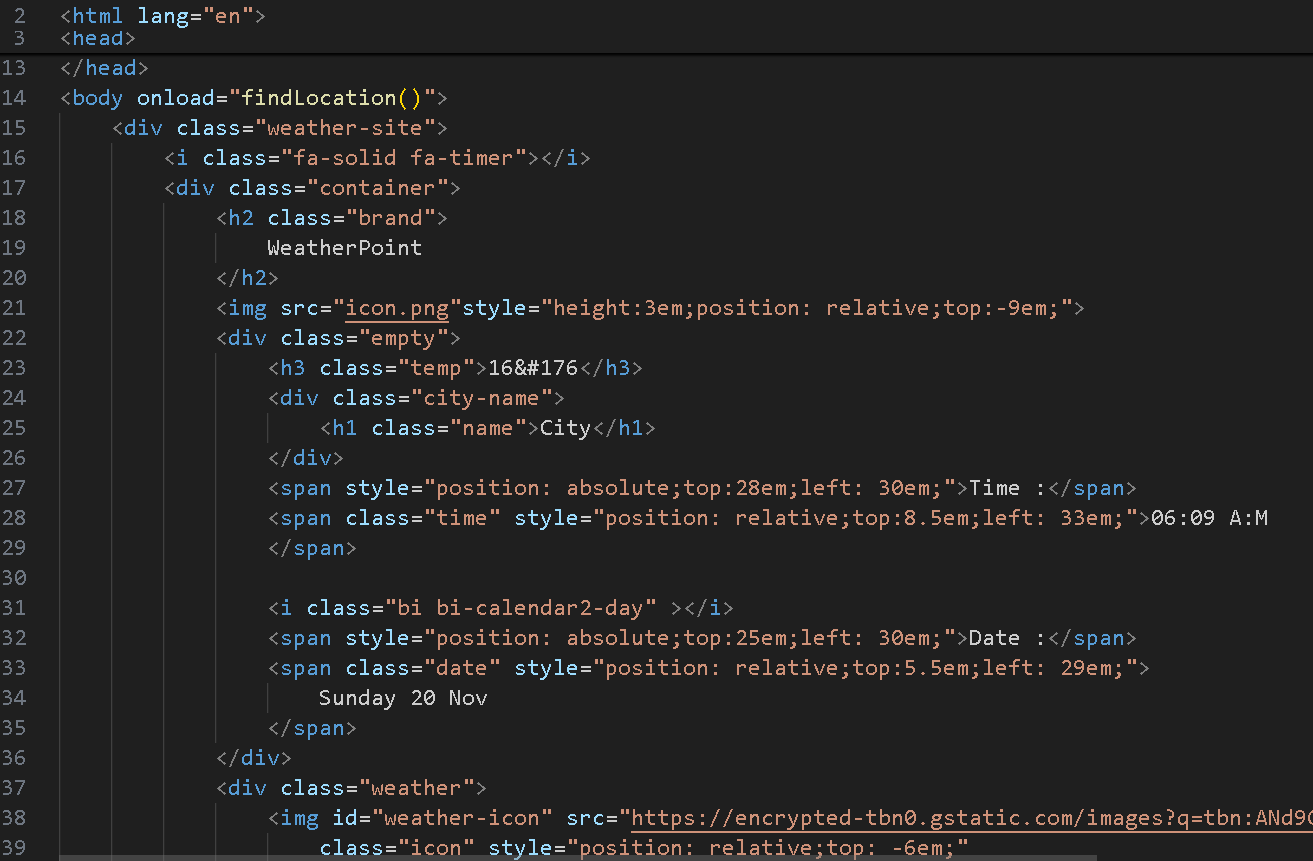


ICON.PNG:



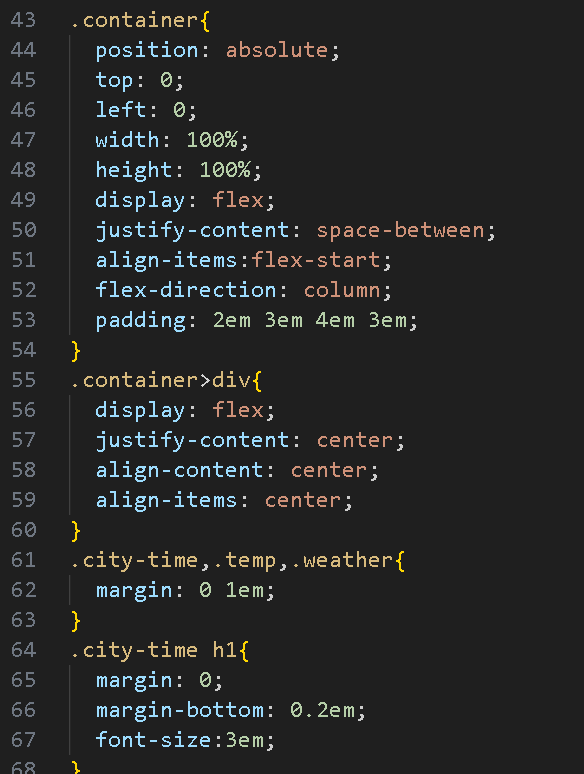
INDEX.HTML:

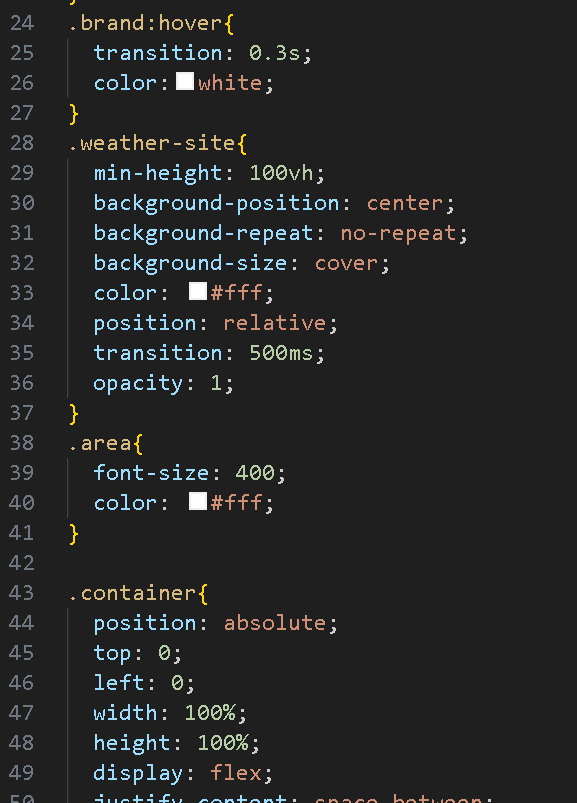




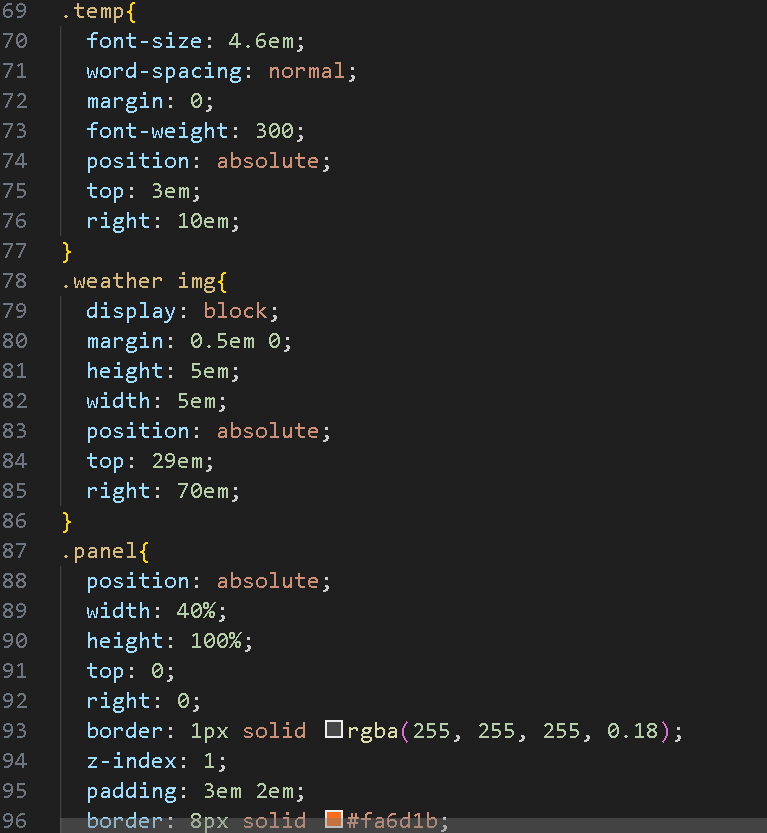








fhf

ddd

**GitHub Project:**

<https://github.com/abdulbabar/Web_tech_project.git>

### 1. **Global Variables:**

* flag: A flag variable used to track whether the location has been found.
* city: A variable to store the name of the city.
* temp, pressure, description: Variables to store temperature, pressure, and weather description data.

### 2. findLocation **Function:**

This function uses the **Geolocation API** to get the user's current latitude and longitude, which is then used to fetch the user's city via reverse geocoding.

javascript

Copy code

function findLocation() {

const success = (position) => {

const latitude = position.coords.latitude;

const longitude = position.coords.longitude;

console.log(latitude)

console.log(longitude)

const geolocation = `https://api.bigdatacloud.net/data/reverse-geocode-client?latitude=${latitude}&longitude=${longitude}&localityLanguage=en`

fetch(geolocation)

.then(res => res.json())

.then(data => {

console.log(data)

city = data.locality;

console.log(city)

const query = city;

getData(query);

flag=1;

})

}

const error = (err) => {

console.log(err);

}

navigator.geolocation.getCurrentPosition(success, error);

console.log(city);

}

* **Geolocation Fetch**: The function retrieves the user's latitude and longitude and then sends a request to the BigDataCloud API to get the user's city using reverse geocoding.
* **API Response**: The city's name is extracted from the response (data.locality), and the getData function is called with the city name to get the weather data.

### 3. getData **Function:**

This function fetches weather data from the **OpenWeatherMap API** using the city name or other identifiers and updates the HTML with the data.

javascript

Copy code

function getData(cityName) {

const query2 = cityName;

const appid = "13810f004da6a9e4c505adb6010576e9";

const units = "metric";

const url = "https://api.openweathermap.org/data/2.5/weather?q="+ query2 +"&units="+ units +"&appid=" + appid;

fetch(url, {

mode: 'cors'

})

.then(res => res.json())

.then(data => {

console.log(data);

var weatherData = data;

const temp = weatherData.main.temp;

const feels = weatherData.main.feels\_like;

const description = weatherData.weather[0].description;

const icon = weatherData.weather[0].icon;

const humidity = weatherData.main.humidity;

const pressure = weatherData.main.pressure;

const visibility = weatherData.visibility;

const windspeed = weatherData.wind.speed;

const imgURL = "http://openweathermap.org/img/wn/"+ icon +"@2x.png";

const date = new Date();

const today = date.toLocaleDateString();

const time = date.toLocaleTimeString();

document.querySelector(".name").innerHTML = cityName;

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

document.querySelector(".pressure").innerHTML = ` Pressure - ${pressure} ATM`;

document.querySelector(".humidity").innerHTML = `Humidity - ${humidity} %`;

document.querySelector(".condition").innerHTML = description;

document.querySelector(".Visibility").innerHTML = `Visibility - ${visibility} meters`;

document.querySelector(".wind-speed").innerHTML = `Wind Speed - ${windspeed} meter/sec`;

document.querySelector(".date").innerHTML = today;

document.querySelector(".time").innerHTML = time;

document.getElementById("weather-icon").setAttribute("src", imgURL);

})

}

* **Weather Data Fetching**: The function builds the request URL using the city name (cityName) and fetches the weather data from the OpenWeatherMap API. The data includes temperature, humidity, weather condition, wind speed, pressure, etc.
* **HTML Update**: The response is used to populate various HTML elements (such as the city name, temperature, pressure, humidity, etc.) with the weather details.
* **Weather Icon**: The weather icon is displayed using an image URL generated from the OpenWeatherMap response.

### 4. checkAgain **Function:**

This function allows the user to input a city name in an input field, and the getData function is called to fetch the weather for the specified city.

javascript

Copy code

function checkAgain() {

const citySearch = document.getElementById('getCityName').value;

getData(citySearch);

}

* **City Search**: The user can type a city name into an input field (with the ID getCityName), and when triggered, it fetches the weather for that city.

### 5. **Predefined City Functions:**

These functions allow the user to quickly fetch the weather for a predefined set of cities.

javascript

Copy code

function chamba() {

getData("Chamba");

}

function bharmour() {

getData("Shimla");

}

function nurpur() {

getData("Nurpur");

}

function chandigarh() {

getData("Chandigarh");

}

* **Predefined City Fetching**: Each function calls getData with a predefined city (e.g., "Islamabad", "Multan", etc.) to display the weather for that location.

### 6. **Improvements / Issues:**

* **Geolocation Timing**: The console.log(city) in the findLocation function may log the city value before it's fetched from the API. It might be better to place this console.log inside the .then() block to log the city only after it's set.
* **Error Handling**: While the geolocation request has an error function, additional error handling for failed API requests could improve the user experience.
* **API Key Security**: The OpenWeatherMap API key is hardcoded. In a production environment, it's a good practice to keep API keys in environment variables or a configuration file, not directly in the code.

#### 1. **Overview**

The provided CSS code is designed to style a weather website that displays current weather information such as temperature, city name, and weather condition. The background of the page features a high-quality image of a landscape, providing a dynamic and visually appealing backdrop. The page layout is responsive, featuring a flexible design that adjusts to various screen sizes.

#### 2. **Key Features of the CSS Design**

* **Background Styling**: The body has a full-screen background image that is centered, not repeated, and covers the entire screen. This sets the tone of the website, making it visually rich.
* **Text Styling**: The body text uses a clean, modern sans-serif font ('Rubik') with a fallback to Arial, Helvetica, or sans-serif. Headings such as h1 and h3 have specific styles to make them stand out. The .brand class uses an italic, bold orange font to highlight branding.
* **Layout Structure**:
  + The .container class uses flexbox to position elements (such as the temperature, weather, and city name) with specific margins and alignments. This ensures the content is centered and spaced correctly.
  + The .panel class is styled to create a side panel with a border and padding for additional information, such as city listings and weather details.
* **Buttons and Interactions**: The .submit button has hover effects with a color change to make it interactive and user-friendly. The .search field has a clean border and placeholder styling for input fields.
* **Weather Information**: The weather information, such as temperature, city name, and weather icons, is styled to be large and easily readable. The temperature uses a large font size (4.6em), while icons (weather-related) are displayed with specific margins and heights.
* **Icons & Images**: Weather icons (#weather-icon) are positioned strategically to complement the weather details, with a fixed size of 10em. Custom styles are applied to icons within the #iconContainer using .icons and .imgClass.

#### 3. **Responsive Design Elements**

* The design uses **absolute positioning** for certain elements (e.g., .panel, .weather img, .time, .date), ensuring that important information like weather and time are positioned relative to the screen.
* The .container is designed with flexbox for a more flexible, responsive layout that adapts well to different screen sizes.

#### 4. **Interactive and Hover Effects**

* Several elements, including .brand, .city, .weather img, and the .pressure, .humidity, and .wind-speed links, have hover effects that change colors, making the website interactive and dynamic.

#### **5. Areas for Improvement**

* **Mobile Responsiveness**: The code uses fixed positioning for some elements, which might not adapt well on very small screens. Using media queries would enhance the layout's responsiveness on mobile devices.
* **Optimization**: Some elements such as .panel and .submit use absolute positioning and may need adjustments on smaller screens. Additionally, the weather icons could be optimized for various screen sizes.

