Document Structure

Head Section

Meta Tags:

<meta charset="UTF-8">: Specifies the character encoding for the document as UTF-8.

<meta http-equiv="X-UA-Compatible" content="IE=edge">: Ensures the document is rendered using the latest Internet Explorer rendering engine.

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Ensures the page is responsive by setting the viewport to match the device's width.

Title:

<title>Weather app</title>: Sets the title of the page, which appears in the browser tab.

Favicon:

<link rel="shortcut icon" href="img/bg.jpg" type="image/x-icon">: Links to a favicon image for the webpage.

Stylesheet:

<link rel="stylesheet" href="style.css">: Links to an external CSS file (style.css) for styling the webpage.

Body Section

Main Application Container:

<div class="app-main" id="parent">: A container for the main application content with a class of app-main and an ID of parent.

Header:

<div class="header"><h4>Get Weather</h4></div>: A header section containing a heading with the text "Get Weather".

Search Input Box:

<div class="searchInputBox"><input type="text" name="" id="input-box" class="input-box" placeholder="enter city name"></div>: A container for the search input box where users can enter a city name. It includes:

An input field of type text with no name attribute, an ID of input-box, a class of input-box, and a placeholder text "enter city name".

Weather Information Container:

<div class="weather-body" id="weather-body"></div>: An empty container with a class of weather-body and an ID of weather-body. This container is intended to be populated with weather information through JavaScript.

JavaScript File:

<script src="script.js"></script>: Links to an external JavaScript file (script.js) that will handle the functionality of the application.

Font Awesome Icon CDN:

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/.../css/all.min.css"...>: Links to a CDN for Font Awesome icons, allowing the use of various icons in the application.

SweetAlert CDN:

<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>: Links to a CDN for SweetAlert, a library for displaying beautiful alerts and dialogs.

Functionality Overview

JavaScript (script.js): This file is expected to handle user interactions, such as fetching the weather data based on the city name entered in the input box and dynamically updating the weather-body div with the retrieved information.

Styling (style.css): This file will contain the CSS rules to style the application, ensuring it looks good and is responsive across different devices.

Purpose

The application allows users to enter a city name and get the current weather information for that city. The weather data will be displayed dynamically within the weather-body container, making use of the structure and styles defined in the HTML and CSS files, and the functionality provided by the JavaScript file

This CSS file styles the weather application defined in the previous HTML file. Here's an explanation of each section:

Font Import

css

```
@import
url('https://fonts.googleapis.com/css2?family=Roboto&family=Ubuntu:wght@300&display=swap');
```

Imports two fonts, 'Roboto' and 'Ubuntu', from Google Fonts.

Global Styles

css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Resets margin and padding for all elements to 0.

Sets box-sizing to border-box for all elements, which includes padding and border in the element's total width and height.

Body Styling

css

```
body {
  font-family: 'Roboto', sans-serif;
  background-image: url("img/bg1.jpg");
  min-height: 92vh;
  overflow: auto;
  background-repeat: no-repeat;
  background-position: top center;
  background-size: cover;
}
```

Applies 'Roboto' font to the entire body.

Sets a background image with specific positioning and sizing.

Ensures the minimum height of the body is 92% of the viewport height.

Allows vertical scrolling if content overflows.

Header Styling

css

```css
.header h4 {
  color: #da1e4e;
  font-weight: 700;
  font-size: 2.4rem;
  font-family: Cambria, Cochin, Georgia, Times, "Times New Roman", serif;
}
```

Styles the header's <h4> element with specific color, weight, size, and font family.

Main Application Container

css

```css
.app-main {
  min-height: 10vh;
  width: 30vw;
  margin: 50px auto;
  padding: 20px;
  text-align: center;
  box-shadow: rgba(60, 64, 67, 0.3) 0px 1px 2px 0px, rgba(60, 64, 67, 0.15) 0px 2px 6px 2px;
  border-radius: 15px;
  background: #c0dfec;
  background: -webkit-linear-gradient(to top, #92fe9d, #c0dfec);
  background: linear-gradient(to top, #92fe9d, #c0dfec);
}
.app-main > * {
  margin-bottom: 20px;
}
```

Styles the main container with a minimum height, width, margin, padding, text alignment, box shadow, and border radius.

Applies a gradient background.

Adds bottom margin to all direct children.

Input Box Styling

css

```
.input-box {
  width: 100%;
  background: azure;
  color: #e4603a;
  font-weight: 500;
  border: none;
  font-size: 1.7rem;
  border-radius: 10px;
  padding: 10px;
  text-align: center;
  outline: none;
  border: none;
}
```

Styles the input box with full width, background color, text color, font weight, font size, border radius, padding, and center-aligned text.

Removes default border and outline.

Weather Information Container

css

```
.weather-body {
  color: #fff;
  padding: 20px;
  line-height: 2rem;
  border-radius: 10px;
  background-color: #eff1f3;
  display: none;

  background: linear-gradient(to top, #da1e4e, #e4603a);

}
```

Styles the weather information container with text color, padding, line height, border radius, background color, and gradient.

Initially hides the container with display: none;.

Weather Details Styling

css

```
.location-deatils {

  font-weight: bold;

}

.weather-status {

  padding: 20px;

}

.temp {

  font-size: 5rem;

  font-weight: 700;

  margin-bottom: 20px 0px;

  text-shadow: 2px 4px rgba(0, 0, 0, 0.1);

}

.weather {

  margin-top: 25px;

  font-size: 2rem;

  margin-bottom: 10px;

}

.min-max {

  font-size: 1.2rem;

  font-weight: 400;

  margin-top: 15px;

}
```

Styles various elements within the weather container to display location details, weather status, temperature, and additional information with specific font sizes, weights, margins, paddings, and text shadow.

Additional Weather Details

css

```
.day_details {
  padding: 20px;
}
.sun-detail,
.basic {
  font-size: 1rem;
}
#weather-icon {
  color: black;
}
```

Provides additional styling for detailed weather information, including padding and font sizes.

Styles the weather icon with a specific color.

Responsive Design

css

```
@media screen and (max-width: 800px) {
  .app-main {
    width: 95%;
    padding: 10px;
  }
  body {
    min-height: 94vh;
  }
}
```

Adjusts the width and padding of the main container for screens narrower than 800px.

Ensures the body maintains a minimum height of 94% of the viewport height on smaller screens

Sure, let's go through the code step-by-step to understand its functionality and flow:

Step 1: Define the Weather API object

javascript

```
const weatherApi = {
  key: '4eb3703790b356562054106543b748b2',
  baseUrl: 'https://api.openweathermap.org/data/2.5/weather'
}
```

weatherApi object contains the API key and base URL for OpenWeatherMap's weather API.

Step 2: Add Event Listener for Enter Key

javascript

```
let searchInputBox = document.getElementById('input-box');
searchInputBox.addEventListener('keypress', (event) => {
  if (event.keyCode == 13) {
    getWeatherReport(searchInputBox.value);
  }
})
```

An event listener is added to the input box. When the Enter key (key code 13) is pressed, the getWeatherReport function is called with the input box value (city name).

Step 3: Fetch Weather Report

javascript

```
function getWeatherReport(city) {
  fetch(`${weatherApi.baseUrl}?q=${city}&appid=${weatherApi.key}&units=metric`)
    .then(weather => {
      return weather.json();
    }).then(showWeaterReport);
}
```

getWeatherReport function constructs the URL using the city name and fetches the weather data from the API.

The fetched data is then parsed to JSON and passed to the showWeaterReport function.

Step 4: Show Weather Report

javascript

```
function showWeaterReport(weather) {
    let city_code = weather.cod;
    if (city_code === '400') {
        swal("Empty Input", "Please enter any city", "error");
        reset();
    } else if (city_code === '404') {
        swal("Bad Input", "entered city didn't matched", "warning");
        reset();
    } else {
        let op = document.getElementById('weather-body');
        op.style.display = 'block';
        let todayDate = new Date();
        let parent = document.getElementById('parent');
        let weather_body = document.getElementById('weather-body');
        weather_body.innerHTML =
            `
        <div class="location-deatils">
            <div class="city" id="city">${weather.name}, ${weather.sys.country}</div>
            <div class="date" id="date"> ${dateManage(todayDate)}</div>
        </div>
        <div class="weather-status">
            <div class="temp" id="temp">${Math.round(weather.main.temp)}&deg;C </div>
            <div class="weather" id="weather"> ${weather.weather[0].main} <i
class="${getIconClass(weather.weather[0].main)}"></i>  </div>
```

```
            <div class="min-max" id="min-max">${Math.floor(weather.main.temp_min)}&deg;C (min) /
${Math.ceil(weather.main.temp_max)}&deg;C (max) </div>

            <div id="updated_on">Updated as of ${getTime(todayDate)}</div>

        </div>

        <hr>

        <div class="day-details">

            <div class="basic">Feels like ${weather.main.feels_like}&deg;C | Humidity
${weather.main.humidity}%  <br> Pressure ${weather.main.pressure} mb | Wind
${weather.wind.speed} KMPH</div>

        </div>

        `;

        parent.append

(weather_body);

changeBg(weather.weather[0].main);

reset();

}

}
```

markdown

- `showWeaterReport` function processes the weather data.

- It checks the response code (`weather.cod`):

  - If the code is `400`, it shows an error message for empty input.

  - If the code is `404`, it shows a warning for an unmatched city name.

  - For valid data, it displays the weather information in the HTML.

- The function dynamically updates the weather details, including city, temperature, weather status, etc.

- It also changes the background image based on the weather status and resets the input box.

### Step 5: Helper Functions

#### Get Current Time

```javascript
function getTime(todayDate) {
    let hour = addZero(todayDate.getHours());
    let minute = addZero(todayDate.getMinutes());
    return `${hour}:${minute}`;
}
```

getTime function returns the current time in HH

format, adding a leading zero if necessary.

Manage Date

javascript

```
function dateManage(dateArg) {
    let days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
    let months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September',
'October', 'November', 'December'];
    let year = dateArg.getFullYear();
    let month = months[dateArg.getMonth()];
    let date = dateArg.getDate();
    let day = days[dateArg.getDay()];
    return `${date} ${month} (${day}) , ${year}`;
}
```

dateManage function formats the current date as "DD Month (Day) , YYYY".

Change Background

javascript

```
function changeBg(status) {
    if (status === 'Clouds') {
        document.body.style.backgroundImage = 'url(img/clouds.jpg)';
    } else if (status === 'Rain') {
        document.body.style.backgroundImage = 'url(img/rainy.jpg)';
    } else if (status === 'Clear') {
        document.body.style.backgroundImage = 'url(img/clear.jpg)';
    } else if (status === 'Snow') {
        document.body.style.backgroundImage = 'url(img/snow.jpg)';
    } else if (status === 'Sunny') {
        document.body.style.backgroundImage = 'url(img/sunny.jpg)';
    } else if (status === 'Thunderstorm') {
        document.body.style.backgroundImage = 'url(img/thunderstrom.jpg)';
    } else if (status === 'Drizzle') {
        document.body.style.backgroundImage = 'url(img/drizzle.jpg)';
    } else if (status === 'Mist' || status === 'Haze' || status === 'Fog') {
        document.body.style.backgroundImage = 'url(img/mist.jpg)';
    } else {
        document.body.style.backgroundImage = 'url(img/bg.jpg)';
    }
}
```

changeBg function changes the background image based on the weather status.

Get Icon Class

javascript

```
function getIconClass(classarg) {
    if (classarg === 'Rain') {
        return 'fas fa-cloud-showers-heavy';
    } else if (classarg === 'Clouds') {
        return 'fas fa-cloud';
    } else if (classarg === 'Clear') {
        return 'fas fa-cloud-sun';
    } else if (classarg === 'Snow') {
        return 'fas fa-snowman';
    } else if (classarg === 'Sunny') {
        return 'fas fa-sun';
    } else if (classarg === 'Mist') {
        return 'fas fa-smog';
    } else if (classarg === 'Thunderstorm' || classarg === 'Drizzle') {
        return 'fas fa-thunderstorm';
    } else {
        return 'fas fa-cloud-sun';
    }
}
```

getIconClass function returns the appropriate icon class for the weather status.

Reset Input Box

javascript

```
function reset() {
    let input = document.getElementById('input-box');
    input.value = "";
}
```

reset function clears the input box.

Add Leading Zero

javascript

```
function addZero(i) {
    if (i < 10) {
        i = "0" + i;
    }
    return i;
}
```

addZero function adds a leading zero to a number if it is less than 10.