JavaScript Weather WebApp Documentation

By: Seif Ehab Tabouzadah

Table of Contents

Index.html

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
   initial-scale=1.0">
    <link

   href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;
   500;600;700;800;900&family=Roboto:wght@300;400;500;700;900&display=s
   wap"
        rel="stylesheet">
    <link rel="stylesheet" href="style.css">
    <title>Day #1 - JavaScript Weather App | Seif Tabouzadah</title>
</head>

<body>

    <div class="container">
        <div class="search-box">
            <i class="fa-solid fa-location-dot"></i>
            <input type="text" placeholder="Enter your location">
            <button class="fa-solid fa-magnifying-glass"></button>
        </div>

        <div class="not-found">
            <img src="images/404.png">
            <p>Oops! Invalid location :/</p>
        </div>

        <div class="weather-box">
            <img src="">
            <p class="temperature"></p>
            <p class="description"></p>
        </div>

        <div class="weather-details">
            <div class="humidity">
                <i class="fa-solid fa-water"></i>
                <div class="text">
                    <span></span>
                    <p>Humidity</p>
                </div>
            </div>
            <div class="wind">
                <i class="fa-solid fa-wind"></i>
                <div class="text">
                    <span></span>
                    <p>Wind Speed</p>
                </div>
            </div>
        </div>

    </div>

    <script src="https://kit.fontawesome.com/7c8801c017.js"
   crossorigin="anonymous"></script>
    <script src="index.js"></script>
</body>

</html>
```

The code above provides the HTML code for the "**index.html**" file. The following is a detailed documentation of the elements and what they contain:

- The top of the page's`<!DOCTYPE html>` element serves as a declaration, informing the web browser what version of HTML the page is written in. It is HTML5 in this scenario.
- The HTML document's base element is the tag `<html>`. It contains all of the additional components. A language is set inside the tag to English.
- The page's information, such as the character set in use, viewport preferences, and connections to external stylesheets and scripts, are all included in the`<head>` element.
    - The character encoding used on the page is specified using the `<meta charset="UTF-8">` element. It is UTF-8 in this instance, a popular character encoding that accommodates a large variety of characters from many writing systems.

    - The tag `<meta http-equiv="X-UA-Compatible" content="IE=edge">` gives the web browser information on how to display the page.
    - The viewport element's `<meta name="viewport" content="width=device-width, initial-scale=1.0">` tag specifies the default width and scale of the page for various devices. The browser is specifically instructed to use the width of the device as the page's width and to set the initial zoom level to 1.0. By doing this, you can make sure that the page will appear properly on screens of various sizes and resolutions.
    - The Poppins and Roboto font definitions from Google Fonts are located in external style sheets that are referred to via the two `<link>` tags. The page has a unified aesthetic thanks to the employment of these typefaces throughout.
    - The title of the page is determined by the `<title>` element and displays in both search engine results and the title bar of the browser. It reads "Day #1 - JavaScript Weather App | Seif Tabouzadah" in this instance.
- The `<body>` tag contains all the visible content of the page.
    - The other page components are contained inside `<div>` tags with the `class="container"` attribute.
    - An input field and a button are included in the `<div>` element with `class="search-box"` so that the user may enter a place and look up the weather. It also has a Font Awesome icon that uses the `fa-solid` class.
        - The `<input>` tag is used to accept user input for the location.
        - The `<button>` tag is used to initiate the weather search based on the user's input.
    - The `<div>` tag with `class="not-found"` is a container for a message and an image that appears when an invalid location is entered. It also includes an icon from Font Awesome.
    - The weather information for the chosen location is shown in the `<div>` element with `class="weather-box"`; this includes the temperature, a picture of the current weather, and a short description of the weather.
    - The `<div>` element with `class="weather-details"` shows more weather statistics like humidity and wind speed. Each sort of information is denoted by a Font Awesome icon.
- The weather app's functionality is provided by external JavaScript scripts, which are loaded via the `<script>` tags at the bottom of the page. The core JavaScript code for the application is being loaded from one file, while the Font Awesome icon library is being loaded from another one.

Style.css

```css
*{
    margin: 0;
    padding: 0;
    border: 0;
    outline: none;
    box-sizing: border-box;
}

body{
    height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    background: #06283D;
}

.container{
    position: relative;
    width: 400px;
    height: 105px;
    background: #fff;
    padding: 28px 32px;
    overflow: hidden;
    border-radius: 18px;
    font-family: 'Roboto', sans-serif;
    transition: 0.6s ease-out;
}

.search-box{
    width: 100%;
    height: min-content;
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.search-box input{
    color: #06283D;
    width: 80%;
    font-size: 24px;
    font-weight: 500;
    text-transform: uppercase;
    padding-left: 32px;
}

.search-box input::placeholder{
    font-size: 20px;
    font-weight: 500;
    color: #06283D;
    text-transform: capitalize;
}

.search-box button{
    cursor: pointer;
    width: 50px;
    height: 50px;
    color: #06283D;
    background: #dff6ff;
    border-radius: 50%;
    font-size: 22px;
    transition: 0.4s ease;
}

.search-box button:hover{
    color: #fff;
    background: #06283D;
}

.search-box i{
    position: absolute;
    color: #06283D;
    font-size: 28px;
}

.weather-box{
    text-align: center;
}
```

```css
.weather-box img{
    width: 60%;
    margin-top: 30px;
}

.weather-box .temperature{
    position: relative;
    color: #06283D;
    font-size: 4rem;
    font-weight: 800;
    margin-top: 30px;
    margin-left: -16px;
}

.weather-box .temperature span{
    position: absolute;
    margin-left: 4px;
    font-size: 1.5rem;
}

.weather-box .description{
    color: #06283D;
    font-size: 22px;
    font-weight: 500;
    text-transform: capitalize;
}

.weather-details{
    width: 100%;
    display: flex;
    justify-content: space-between;
    margin-top: 30px;
}

.weather-details .humidity, .weather-details .wind{
    display: flex;
    align-items: center;
    width: 50%;
    height: 100px;
}
```

```css
.weather-details .humidity{
    padding-left: 20px;
    justify-content: flex-start;
}

.weather-details .wind{
    padding-right: 20px;
    justify-content: flex-end;
}

.weather-details i{
    color: #06283D;
    font-size: 26px;
    margin-right: 10px;
    margin-top: 6px;
}

.weather-details span{
    color: #06283D;
    font-size: 22px;
    font-weight: 500;
}

.weather-details p{
    color: #06283D;
    font-size: 14px;
    font-weight: 500;
}

.not-found{
    width: 100%;
    text-align: center;
    margin-top: 50px;
    scale: 0;
    opacity: 0;
    display: none;
}

.not-found img{
    width: 70%;
}

.not-found p{
    color: #06283D;
    font-size: 22px;
    font-weight: 500;
    margin-top: 12px;
}

.weather-box, .weather-details{
    scale: 0;
    opacity: 0;
}

.fadeIn{
    animation: 0.5s fadeIn forwards;
    animation-delay: 0.5s;
}

@keyframes fadeIn{
    to {
        scale: 1;
        opacity: 1;
    }
}
```

The code above is a CSS stylesheet that defines the styling and layout for the weather search web application. Explanation of the code:

- All components on the page are given default values by the first section of the code, which begins with the symbol `*{`. It specifically utilizes the `border-box` value for the `box-sizing` property and sets the `margin`, `padding`, `border`, and `outline` to `0`. This is a practical method to guarantee uniform size and spacing between items.
- The `body` selector changes the background color of the whole page to the dark blue hue **#06283D**. Also, it sets the `height` of the body element to `100vh`, which indicates that it will have the same height as the viewport. This makes sure the page takes up the whole screen.

- The style for the primary search container is determined by the `container` class. It produces a white box by setting the background color to #fff and the width and height to `400px` and `105px`, respectively. The `padding` property expands the box's inside, while the `border-radius` property softens the box's angles. When the `overflow` attribute is set to hidden, any content that would otherwise overflow the container will be concealed. As the container's size or location changes, the `transition` attribute enhances the transition impact.
- The search input box at the top of the container is defined by the `search-box` class. It sets the height to `min-content` so that the element's height will be decided by its content and the width to `100%` to guarantee that it fits the complete width of the container. Flexibility in layout choices is made possible by the `display` property, which is set to `flex`. The search-content box is centered thanks to the attributes `align-items` and `justify-content`.
- The search box's input field is styled by the `search-box input` class. It changes the color to #06283D, a dark blue hue. To avoid having the input field occupy the whole width of the search box, the width is set to `80%`. The font-weight and font-size are both set to `500`. All characters in the input box are capitalized since the text-transform attribute is set to uppercase. Padding-left provides more room to the left of the input field.
- The `search-box input::placeholder` selector styles the placeholder text in the input field. It sets the font-size to `20px`, font-weight to `500`, and color to #06283D. The text-transform property is set to capitalize, which capitalizes the first letter of each word.
- The `search-box button` selector styles the search button. It sets the `cursor` property to `pointer`, which changes the mouse cursor to a hand when hovering over the button. The width and height properties are set to `50px` to create a circular button. The color property is set to #06283D, and the background to #dff6ff, which creates a light blue color for the button. The `border-radius` property is set to `50%` to create a circular shape. The `font-size` property is set to `22px`, and the `transition` property adds a smooth transition effect when the button is hovered over.
- The `search-box button:hover` selector styles the search button when it is hovered over. It changes the color property to white, and the background property to #06283D.
- The `.weather-box` class defines the weather details section. It contains an image of the weather condition, the temperature, and a description of the weather condition. The temperature has a font size of `4rem` and a font weight of `800`, and the temperature unit is positioned absolutely with a font size of `1.5rem`. The description of the weather condition has a font size of `22px`, a font weight of `500`, and a text-transform of capitalize.

- The `weather-details` class contains additional weather information, such as humidity and wind speed. The humidity and wind sections have a width of `50%` and are displayed flexibly. The icons for the weather details have a font size of `26px` and a color of #06283D, and the text has a font size of `22px` and a font weight of `500`.
- The `not-found` class which is used when the weather data cannot be found for a particular location. It contains an image of a sad face, and a message letting the user know that the weather data could not be found.
- To provide a better user experience, the `fadeIn` animation is used to smoothly transition the weather data and the error message into view. The `animation` property is used to specify the fadeIn animation, which has a duration of `0.5` seconds and an `animation-delay` of `0.5` seconds. The `forwards` value is used to ensure that the element stays in its final state after the animation has completed.
- The `@keyframes` rule is used to define the animation called "`fadeIn`". It specifies how the animation will change over time by defining styles at various stages. In this case, the keyframes are defined from the initial state (`scale: 0`, `opacity: 0`) to the final state (`scale: 1`, `opacity: 1`) over a duration of `0.5` seconds. When the "`fadeIn`" animation is applied to an element using the `animation` property, the element will gradually transition from being hidden to being visible with a smooth animation effect. Using animations like this can enhance the user experience by making the interface more engaging and providing visual cues to indicate changes in the state of the application.

Overall, this code is well-organized, easy to read, and follows best practices for web-development.

Index.js

```
const container = document.querySelector('.container');
const searchBtn = document.querySelector('.search-box button');
const searchInput = document.querySelector('.search-box input');
const weatherBox = document.querySelector('.weather-box');
const temperature = document.querySelector('.weather-box .temperature');
const description = document.querySelector('.weather-box .description');
const weatherDetails = document.querySelector('.weather-details');
const humidity = document.querySelector('.weather-details .humidity
span');
const wind = document.querySelector('.weather-details .wind span');
const notFound = document.querySelector('.not-found');

const apiKey = '[redacted]';
const imageUrl = 'images/';
const weatherImages = {
  Clear: 'clear.png',
  Rain: 'rain.png',
  Snow: 'snow.png',
  Clouds: 'cloud.png',
  Haze: 'mist.png',
};

searchBtn.addEventListener('click', () => {
  const city = searchInput.value.trim();

  if (!city) {
    return;
  }


fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&units=met
ric&appid=${apiKey}`)
    .then(response => {
      if (!response.ok) {
        throw new Error('Invalid location');
      }
      return response.json();
    })
    .then(data => {
      const { main, weather, wind: { speed } } = data;
      const weatherImage = weatherImages[weather[0].main] || '';
      const weatherDescription = weather[0].description || '';

      temperature.innerHTML = `${Math.round(main.temp)}<span>°C</span>`;
      description.innerHTML = weatherDescription;
      humidity.innerHTML = `${main.humidity}%`;
      wind.innerHTML = `${Math.round(speed)}Km/h`;

      if (weatherImage) {
        weatherBox.querySelector('img').src = imageUrl + weatherImage;
      }

      weatherBox.classList.add('fadeIn');
      weatherDetails.classList.add('fadeIn');
      weatherBox.style.display = '';
      weatherDetails.style.display = '';
      container.style.height = '590px';
      notFound.style.display = 'none';
    })
    .catch(error => {
      weatherBox.style.display = 'none';
      weatherDetails.style.display = 'none';
      container.style.height = '400px';
      notFound.style.display = 'block';
      notFound.classList.add('fadeIn');
      console.error(error);
    });
});
```

The above code is the "**index.js**" script that is used to make elements on the HTML page functional. An explanation of the above follows:

- First, I selected some important elements from the HTML document using the `querySelector` method. These elements include:
  - `container`: the container element that holds the entire weather app.
  - `searchBtn`: the button element that triggers the search for a city's weather data.
  - `searchInput`: the input element where the user enters the name of a city.
  - `weatherBox`: the element that displays the weather data (e.g., temperature, weather description, weather image).
  - `temperature`: the element inside '`weatherBox`' that displays the temperature.
  - `description`: the element inside '`weatherBox`' that displays the weather description.
  - `weatherDetails`: the element that displays additional weather details (e.g., humidity, wind speed).
  - `humidity`: the element inside '`weatherDetails`' that displays the humidity percentage.
  - `wind`: the element inside '`weatherDetails`' that displays the wind speed.

- Also, I implemented an element that indicates an error if the user inputs an incorrect location:
    - `notFound`: the element that displays an error message when the search for a city's weather data fails.
- Next, I defined a constant `apiKey` which holds the API key required to make requests to the OpenWeatherMap API.
- I also defined an object `weatherImages` that maps the main weather condition to the appropriate weather icon image filename.
- After selecting the necessary elements and defining the API key and weather icon filenames, I added an event listener to the `searchBtn` button element that triggers the search for a city's weather data. When the button is clicked, the code performs the following actions:
    - It retrieves the value of the `searchInput` element, which contains the name of the city entered by the user.
    - If the `searchInput` element is empty, the function returns early and does nothing.
    - Otherwise, it makes a request to the OpenWeatherMap API using `fetch`. The API endpoint used includes the city name entered by the user, the units used to display the temperature (metric in this case), and the API key required to access the API.
    - If the response from the API is not OK (i.e., if the city name entered by the user is invalid), the code throws an error with the message "*Invalid location*".
    - If the response from the API is OK, the code extracts the necessary data from the JSON response using destructuring.
    - The code updates the `temperature`, `description`, `humidity`, and `wind` elements with the relevant data from the API response.
    - The code checks the `weatherImages` object to determine the appropriate weather icon image filename based on the main weather condition returned by the API. If a matching filename is found, the code updates the `src` attribute of the `img` element inside the `weatherBox` element with the appropriate image URL.
    - The code adds the `fadeIn` class to both the `weatherBox` and `weatherDetails` elements to animate their display using CSS.
    - The code updates the display property of the `weatherBox`, `weatherDetails`, and `notFound` elements to show or hide them based on whether the search was successful.
    - Finally, the code updates the `height` of the `container` element to ensure that the error message is properly displayed if necessary.

Overall, this code provides a simple weather app that uses the OpenWeatherMap API to display the weather data of a city entered by the user. The code is well-organized and follows best practices for web development.