**TITLE: CodTech IT Solutions Internship - Task Documentation: “To-DO LIST” Using CSS, HTML, JAVASCRIPT.**

## INTERN INFORMATION:

**Name:**CHETAN N KALLIMANI

**ID:** COD7068

# INTRODUCTION

In the realm of personal productivity and organizational tools, the weather forecast app occupies a central role, universally recognized for its simplicity and effectiveness in managing tasks and priorities. Transitioning from traditional weather forecasts to a modern digital platform enhances accessibility and functionality, empowering users to stay informed efficiently. This project endeavors to revolutionize weather forecasting by crafting an application leveraging contemporary web technologies: JavaScript, HTML, and CSS.

Embracing digital weather forecasting brings endless customization, real-time updates, and multi-device accessibility. Amidst existing applications, an opportunity exists to craft a more intuitive, user-centric tool. This project aims to harness JavaScript's power and web technologies' versatility for an enhanced forecasting experience.

# Implementation

* JavaScript Framework: Utilize a modern JavaScript framework for building the frontend application.
* HTML/CSS: Use HTML5 and CSS3 for structuring and styling the user interface, ensuring compatibility with various web browsers.
* Responsive Design: Implement responsive design principles to ensure optimal viewing experience across desktop and mobile devices.
* User Interface Components: Utilize UI libraries for designing interactive and visually appealing components.

# CODE: HTML File

<!DOCTYPE html>

<html lang="en">

<head><!--contains the meta information-->

    <meta charset="UTF-8"><!--sets the character encoding for the document-->

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

    <title>Weather Forecast App</title><!--sets the title of the webpage-->

    <link rel="stylesheet" href="styles.css"><!-- links the exyernal css styles sheets-->

</head>

<body>

    <div class="container"><!--this is an container-->

        <h1>Weather Forecast App</h1><!--displays the headings -->

        <input type="text" id="cityInput" placeholder="Enter city name"><!--input fiels to enter the city name-->

        <button onclick="getWeather()">Get Weather</button><!--button that triggers get weather function-->

        <div id="weatherInfo"></div><!--placeholder to display the weather information-->

    </div>

    <script src="script.js"></script><!--links to an external javascript file-->

</body>

</html>

# CSS File

body

{

    font-family: Arial, sans-serif;

    background-color:palevioletred;

} /\*styling for the body\*/

.container

{

    max-width: 600px;

    margin: 200px 400px;

    padding: 20px;

    background-color:rgb(121, 121, 225);

    border-radius: 10px;

}/\*styling for the container, adding the maxwidth, margin, padding, backgroundcolor, border radius\*/

input[type="text"]

{

    width: 90%;

    padding: 10px;

    border-radius: 4px;

    border: 1px solid #ccc;

}/\*input box styling\*/

button

 {

    padding: 10px 20px;

    margin-top: 10px;

    background-color: yellow;

    color:black;

    border-color: black;

    border-radius: 4px;

}/\*styling for the button\*/

button:hover

{

    background-color:  greenyellow;

}/\* color changing when hovered on the button\*/

#weatherInfo

 {

    margin-top: 20px;

# JAVASCRIPT File

async function getWeather() {

    const city = document.getElementById('cityInput').value;

    const apiKey = 'fffc48aca5014d974ca35d7a7d54dea7';

    const apiUrl = `https://api.openweathermap.org/data/2.5/weather?q=${city}&units=metric&appid=${apiKey}`;

    try {

        const response = await fetch(apiUrl);

        const data = await response.json();

        if (data.cod === '404') {

            showError('City not found');

            return;

        }

        const weatherInfo = createWeatherInfo(data);

        displayWeatherInfo(weatherInfo);

    } catch (error) {

        console.error('Error:', error);

        showError('An error occurred');

    }

}

function showError(message) {

    document.getElementById('weatherInfo').innerText = message;

}

function createWeatherInfo(data) {

    return `

        <h2>${data.name}, ${data.sys.country}</h2>

        <p>${data.weather[0].main} - ${data.weather[0].description}</p>

        <p>Temperature: ${data.main.temp}°C</p>

        <p>Feels like: ${data.main.feels\_like}°C</p>

        <p>Humidity: ${data.main.humidity}%</p>

    `;

}

function displayWeatherInfo(weatherInfo) {

    document.getElementById('weatherInfo').innerHTML = weatherInfo;

}

# CODE EXPLAINATION

## HTML Structure: HTML structure organizes webpage content with nested elements, forming a hierarchical layout. Tags like <html>, <head>, <body>, etc., define the backbone of presentation and functionality.

* <div class=”container”>: This <div> element with a class of "container" defines a container for the content, allowing for easier styling and organization.
* <h1>Weather Forecast App</h1>: Displays the title of the weather forecast application.
* <input type="text" id="cityInput" placeholder="Enter city name">: Creates a text input field for users to enter a city name.
* <button onclick="getWeather()">Get Weather</button>: Creates a button to trigger the getWeather() function.
* <div id="weatherInfo"></div>: Creates a placeholder for displaying weather information.
* <script src="script.js"></script>: Links an external JavaScript file for dynamic functionality.

## CSS Styling:

The CSS styles define the look and feel of the to-do list, applying a gradient background, styling the input fields, buttons, and tasks.

Key styling includes:

* body: Sets the font family to Arial or sans-serif and the background color to palevioletred.
* container: Defines a container with a maximum width of 600px, margin of 200px on top and bottom, 400px on left and right, padding of 20px, background color of rgb(121, 121, 225), and border radius of 10px.
* input[type="text"]: Styles text input fields with a width of 90%, padding of 10px, border radius of 4px, and a 1px solid border color of #ccc.
* button: Styles buttons with padding of 10px vertically and 20px horizontally, a margin-top of 10px, background color of yellow, text color of black, border color of black, and a border radius of 4px.
* button: hover: Changes the background color to greenyellow when the button is hovered over.
* #weatherInfo: Adds a margin-top of 20px to the element with the ID "weatherInfo", presumably used to display weather information in the JavaScript functionality.

## JavaScript Functionality:

* async function getWeather () : Defines an asynchronous function to fetch weather data based on user input.
* const city = document. getElementById('cityInput'). value; Retrieves the value entered in the city input field.
* const apiKey = 'fffc48aca5014d974ca35d7a7d54dea7’; Assigns the OpenWeatherMap API key for accessing weather data.
* const apiUrl = Constructs the URL for the OpenWeatherMap API request using the city input and API key.
* try {...} catch (error) {...}: Tries to fetch weather data and handles errors if any occur.
* const response = await fetch (apiUrl): Fetches weather data from the OpenWeatherMap API.
* const data = await response. Json ();: Converts the fetched data into JSON format.
* if (data.cod === '404'): Checks if the response indicates a city not found error.
* showError ('City not found'): Displays an error message if the city is not found.
* const weatherInfo = createWeatherInfo (data); Generates HTML content with weather information.
* displayWeatherInfo (weatherInfo); Displays the weather information on the webpage.
* function showError(message) : Displays an error message on the webpage.
* document. getElementById('weatherInfo'). innerText = message; Sets the text content of the weather info element to the error message.

**USAGE:**  
  
**Adding a Task:** You need to integrate an event listener to capture user input (e.g., pressing "Enter" after typing a task) in the HTML file. In the JavaScript file, create a function to handle adding the task to a list or display area.

**Marking a Task as Completed:** Update the HTML structure to include a way for users to mark tasks as completed, typically by clicking on the task itself. In the JavaScript file, implement a function to toggle the completed status of a task when it's clicked.

# Removing a Task: Add a visual element (e.g., a "remove" button) for each task in the HTML. Implement a function in the JavaScript file to handle removing the task from the list when the corresponding button is clicked. CONCLUSION: The weather forecasting app represents a pivotal tool in personal productivity and organization, known for its simplicity and effectiveness in managing tasks and priorities. Transitioning from traditional forecasts to a modern digital platform enhances accessibility and functionality, enabling users to stay informed efficiently. By leveraging contemporary web technologies such as JavaScript, HTML, and CSS, this project aims to revolutionize weather forecasting. Embracing digital innovation offers endless customization, real-time updates, and multi-device accessibility, paving the way for a more intuitive and user-centric experience. With a focus on harnessing the power and versatility of web technologies, the weather forecasting app endeavors to provide an enhanced and seamless forecasting experience for users.

# OUTPUT: -

# A screenshot of a weather forecast application Description automatically generatedA screenshot of a login form Description automatically generated

# A screenshot of a weather login Description automatically generated