Dynamic Web Application using Unsplash and OpenWeatherMap APIs

Front-End Web Development

Lambton College

By Sushil Thapa
2024/04/10

Catalog

Table of Image:

1. Introduction

Welcome to my dynamic web application project! In this project, I utilized JavaScript and jQuery to create an interactive web application that integrates with the Unsplash and OpenWeatherMap APIs. The primary goal of this project was to enhance my skills in JavaScript and jQuery by working with API data fetching and real-time DOM manipulation. This project involves creating a web application that dynamically displays categorized pictures sourced from the Unsplash API and current weather information obtained from the OpenWeatherMap API. By leveraging these APIs, users can explore a variety of pictures based on categories such as nature, architecture, animals, and more, while also staying informed about the current weather conditions of their location or any other specified location. Below mentioned are the key features of my application:

- Integration with Unsplash API: The application fetches high-quality pictures from Unsplash API based on user-selected categories.
- Integration with OpenWeatherMap API: Real-time weather information is retrieved from the OpenWeatherMap API, allowing users to stay updated about the current weather conditions.
- Dynamic Content Update: The application dynamically updates the displayed pictures and weather information without the need for page refresh, providing a seamless user experience.

To achieve this I have used the mentioned technologies:

- JavaScript: Used to handle API requests, data manipulation, and dynamic content updates.
- jQuery: Utilized for DOM manipulation and simplifying asynchronous operations.
- Unsplash API: Integrated to fetch categorized pictures for display.
- OpenWeatherMap API: Integrated to retrieve real-time weather data.
- HTML & CSS: Used for structuring the web page layout and styling elements.
-

Through this project, I aimed to strengthen my understanding of JavaScript, jQuery, and API integration concepts, as well as improve my ability to develop dynamic web applications that offer valuable real-time information to users.

Homepage

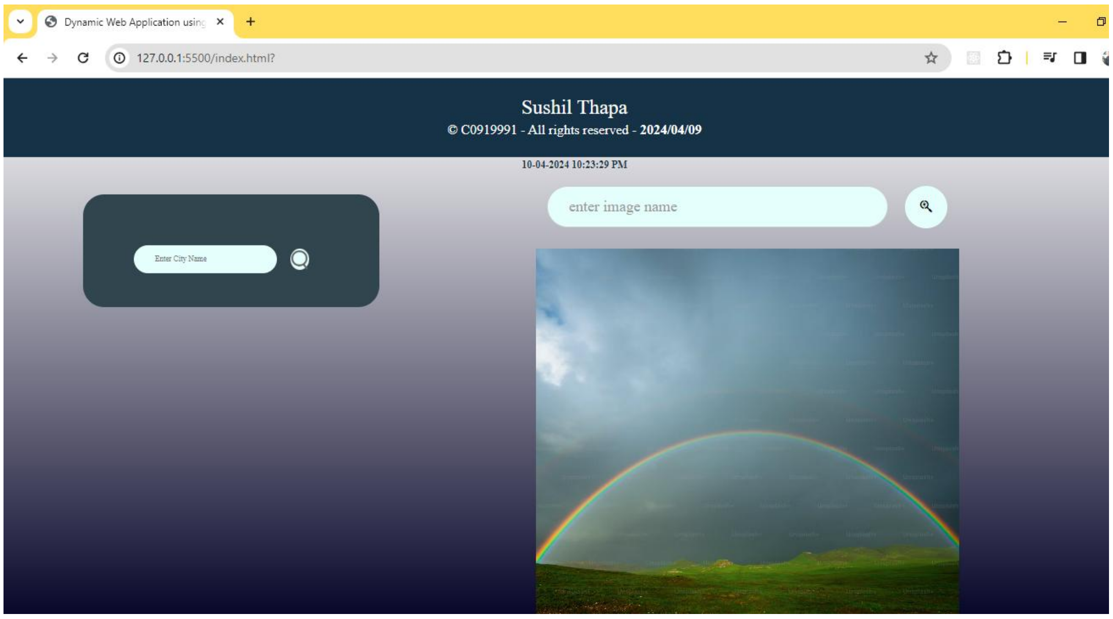
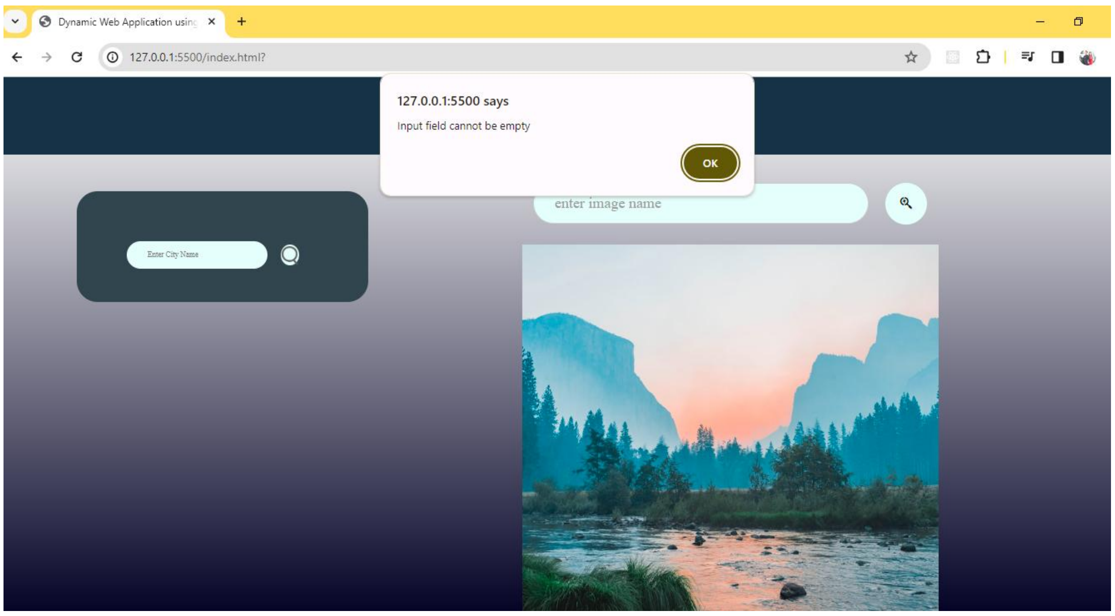

Dynamic Web Application using
127.0.0.1:5500/index.html?
Sushil Thapa
© C0919991 - All rights reserved - 2024/04/09
10-04-2024 10:23:29 PM
enter image name
Enter City Name

Image 1 Screenshot of Homepage Dashboard

Image 1 shows the initial phase of applicati on which display user to enter city name, picture and display local time.

2. Handling of errors

2.1 Empty input field weather city name



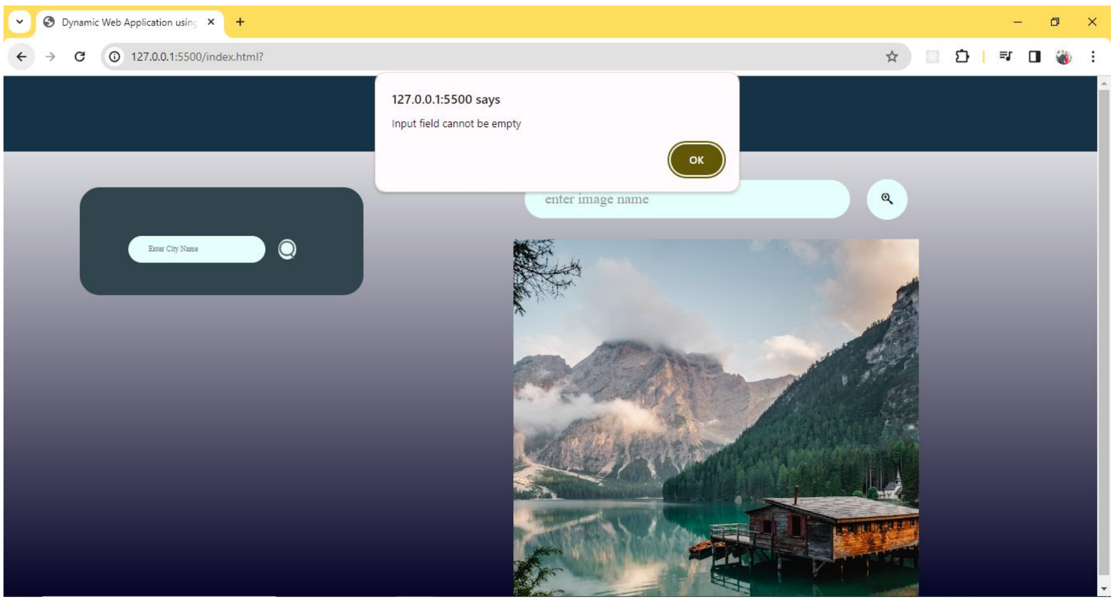
127.0.0.1:5500 says

Input field cannot be empty

Image 2 Screenshot of Empty Input Field Validation

In figure 2, if a empty city name has been parsed then it show an alert to user to enter a city name

2.2 Empty input field unsplash



127.0.0.1:5500 says

Input field cannot be empty

Image 3 Screenshot of Empty Input Field Validation

In figure 3, if a empty city name has been parsed then it show an alert to user to enter a city name

2.3 Invalid City Name

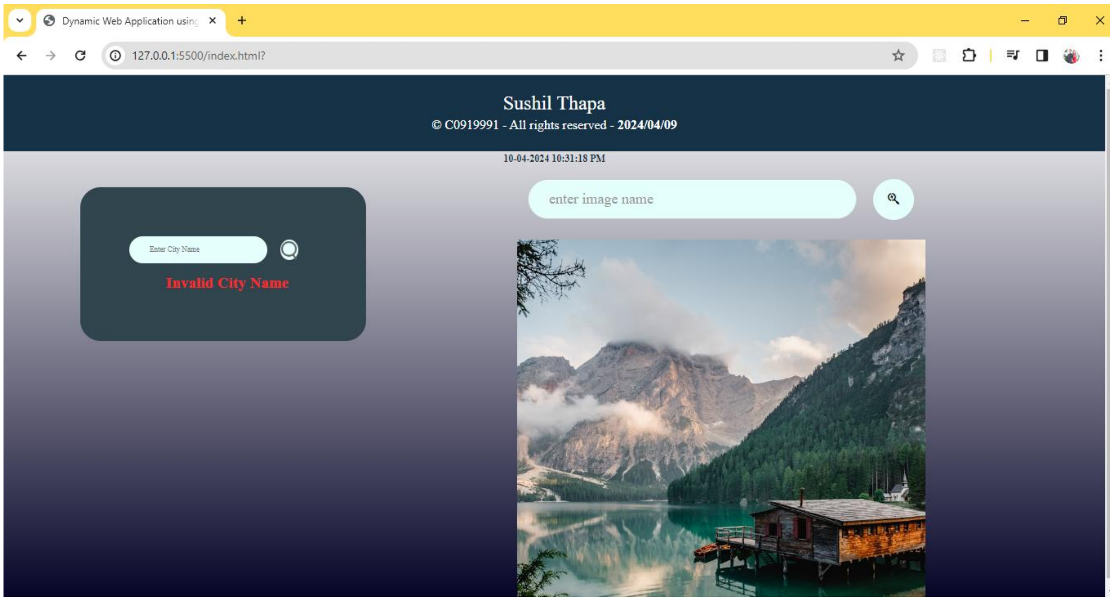

Dynamic Web Application using
127.0.0.1:5500/index.html?
Sushil Thapa
© C0919991 - All rights reserved - 2024/04/09
10-04-2024 10:31:18 PM
Enter City Name
Invalid City Name
enter image name

Image 4 Screenshot of Invalid City Name Validation

In image 4, if a user entered a invalid city name then it shows an error message to user that they have entered invalid city name.

2.4 Loading State

2.4.1 Initial State

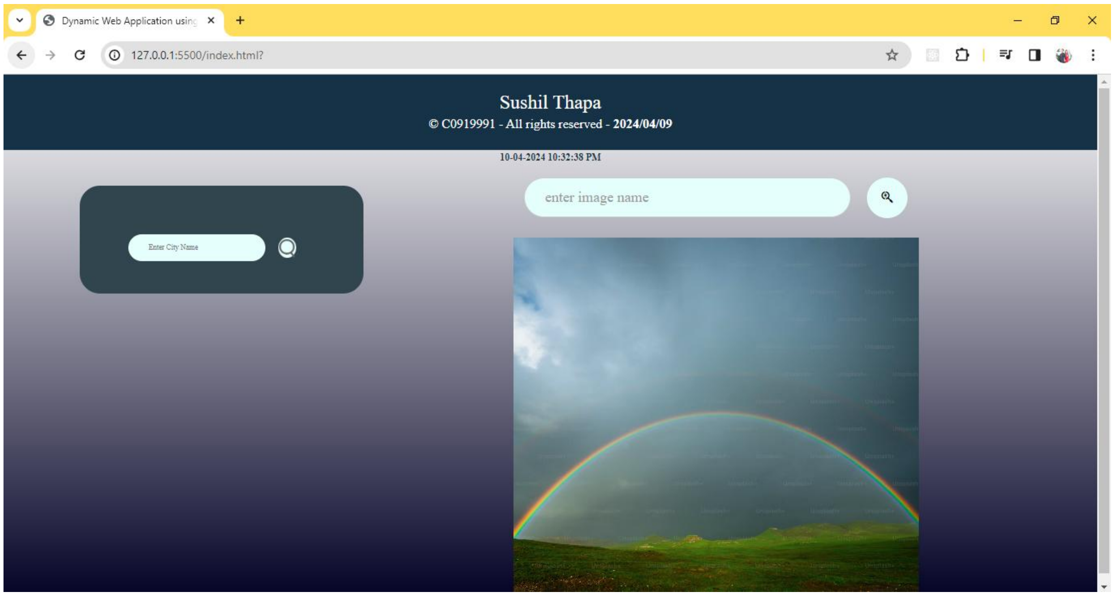

Dynamic Web Application using
127.0.0.1:5500/index.html?
Sushil Thapa
© C0919991 - All rights reserved - 2024/04/09
10-04-2024 10:32:38 PM
Enter City Name
enter image name

Image 5 Initial State of Loading state

2.4.2 In loading state

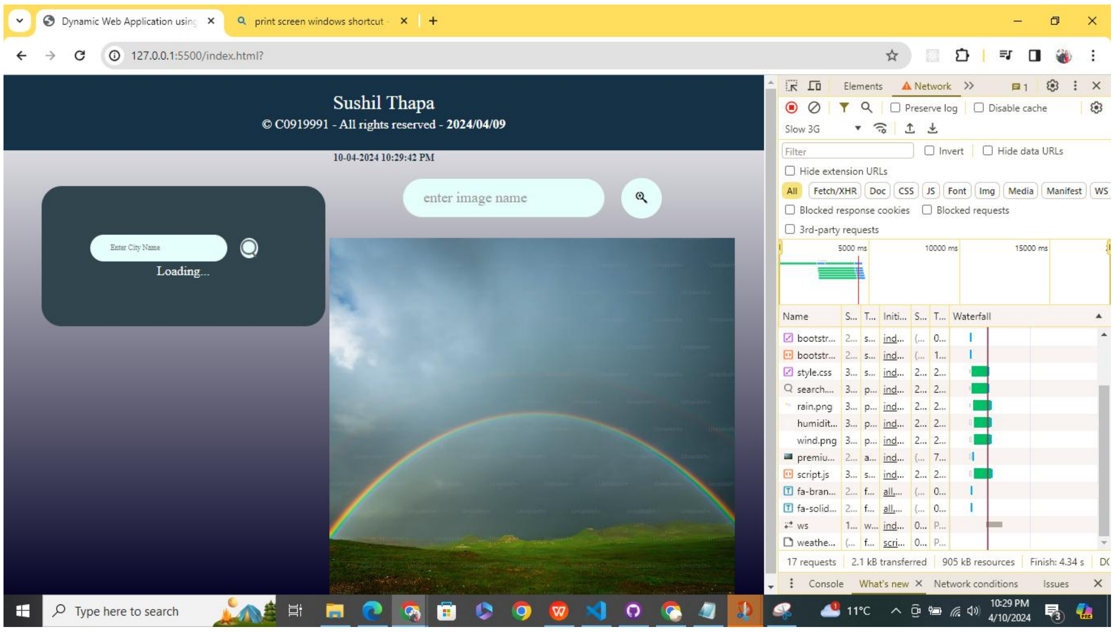


Image 6 While in loading form

2.4.3 After loading state

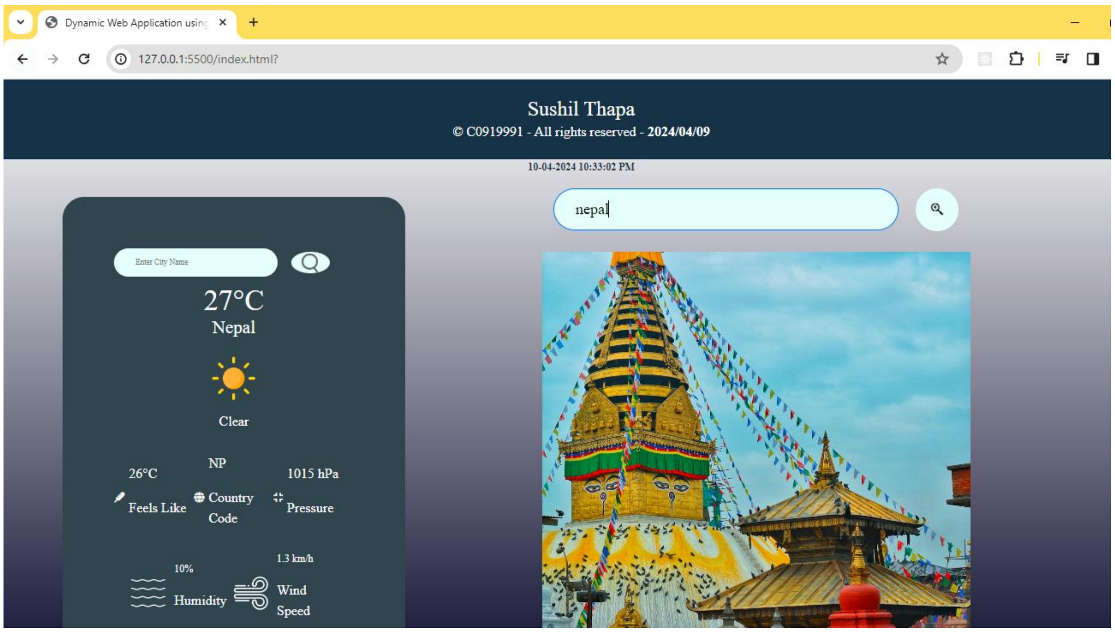


Image 7 After loading state

3. Weather test for different city

3.1.1 Nepal

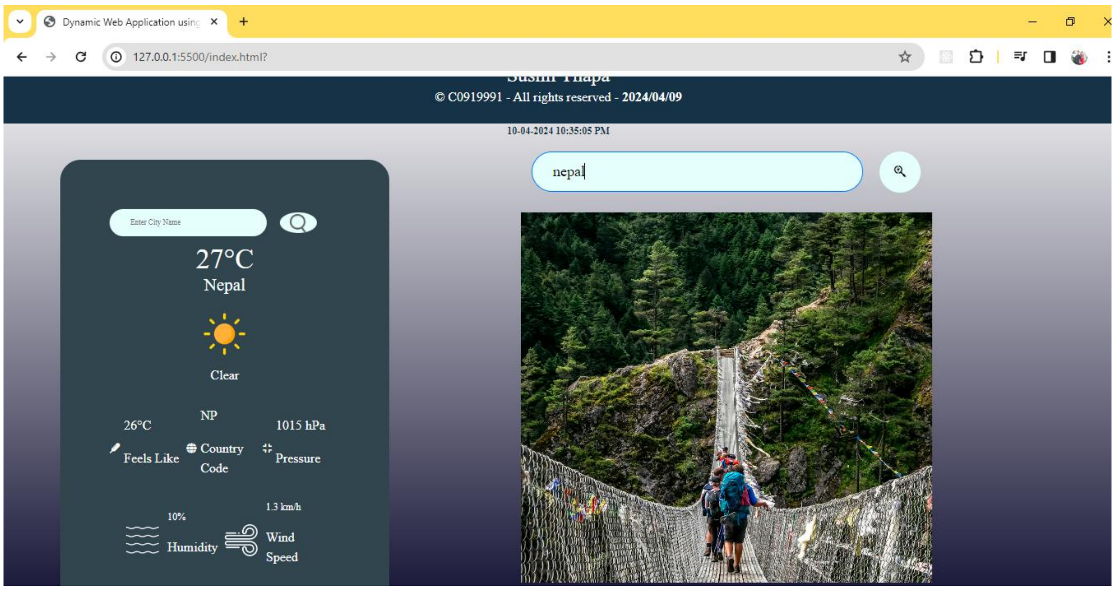


Image 8 Weather test for country Nepal

In image 8, when country name Nepal is entered in search box it displays temperature in degree Celsius and other weather data like Feels like, Humidity, Wind Speed, Country code and Pressure.

3.1.2 Toronto

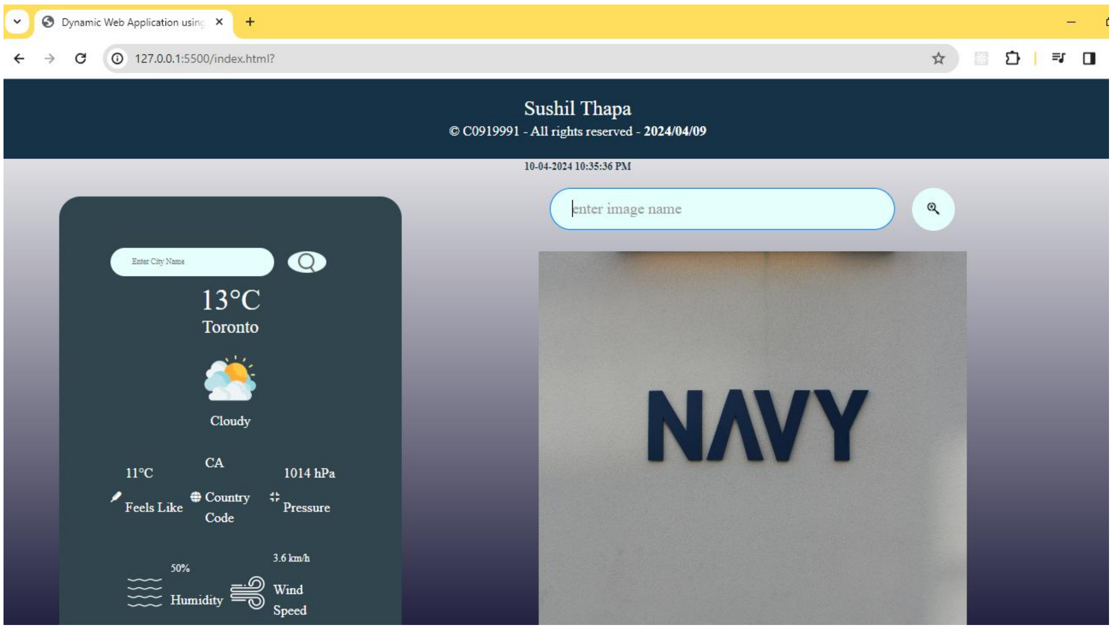


Image 9 Weather test for city Toronto

In image 9, when city name Toronto is entered in search box it displays temperature in degree Celsius and other weather data like Feels like, Humidity, Wind Speed, Country code and Pressure.

3.1.3 Tokyo

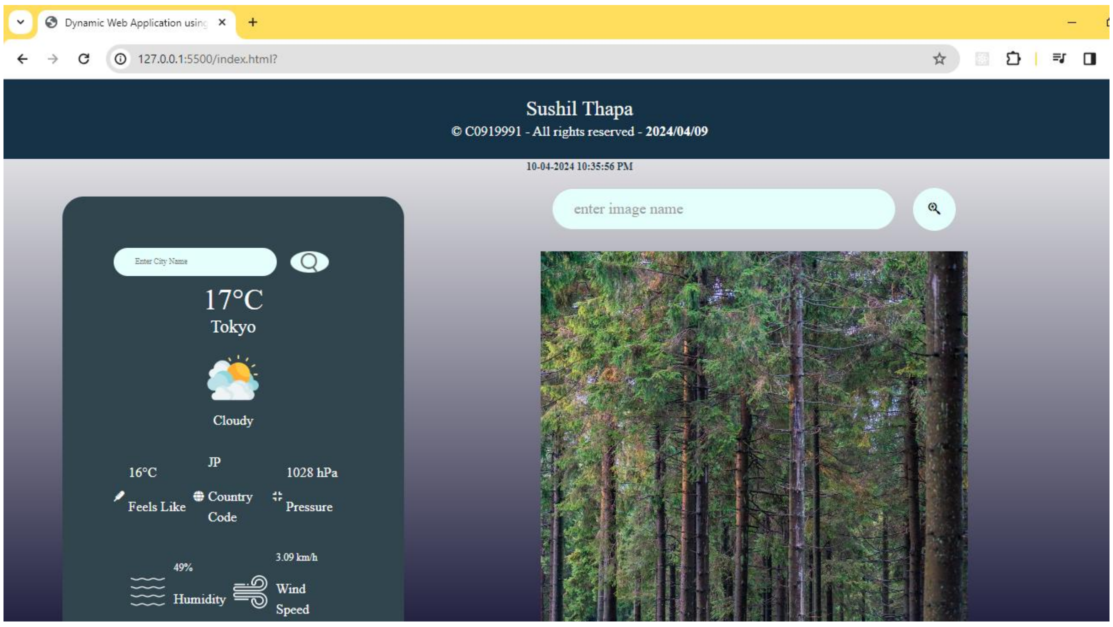


Image 10 Weather test for city Tokyo

In image 10, when country city Tokyo is entered in search box it displays temperature in degree

Celsius and other weather data like Feels like, Humidity, Wind Speed, Country code and Pressure.

3.1.4 Dubai

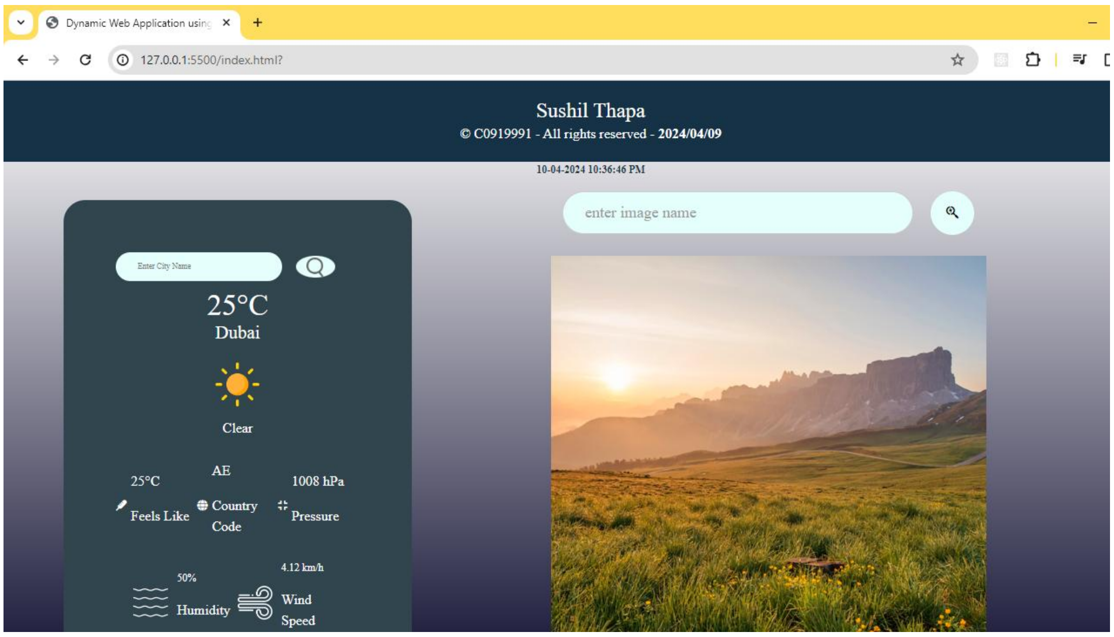


Image 11 Weather test for city Dubai

In image 11, when country name Dubai is entered in search box it displays temperature in degree Celsius and other weather data like Feels like, Humidity, Wind Speed, Country code and Pressure.

3.1.5 Spain

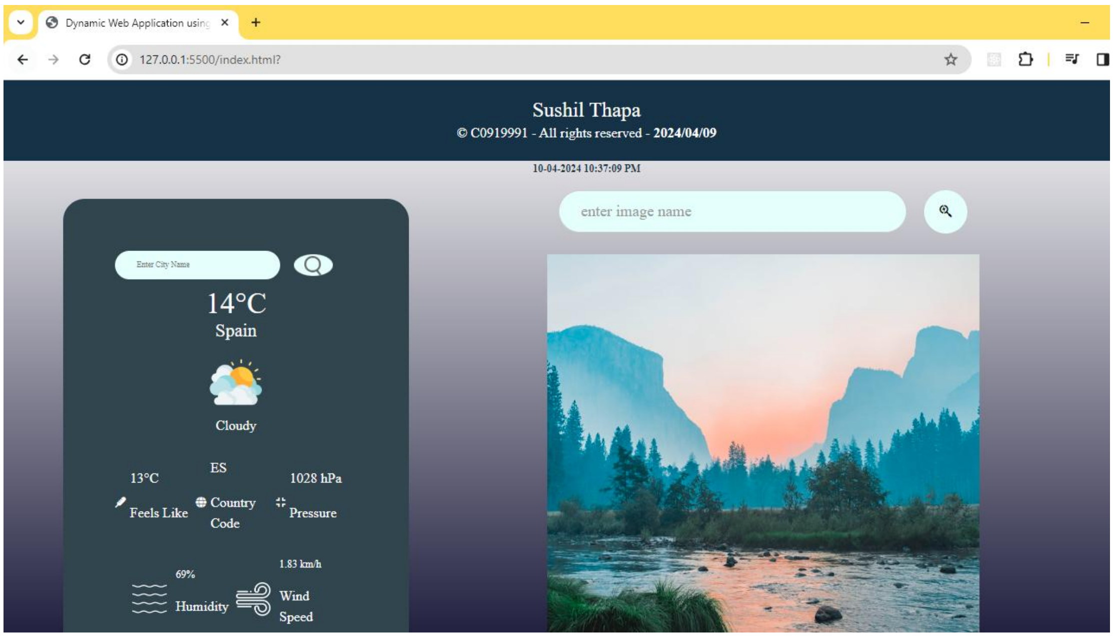


Image 12 Weather test for country Spain

In image 12, when country name Spain is entered in search box it displays temperature in degree Celsius and other weather data like Feels like, Humidity, Wind Speed, Country code and Pressure.

3.2.1 Unsplash test for different categories

3.2.1 Bear

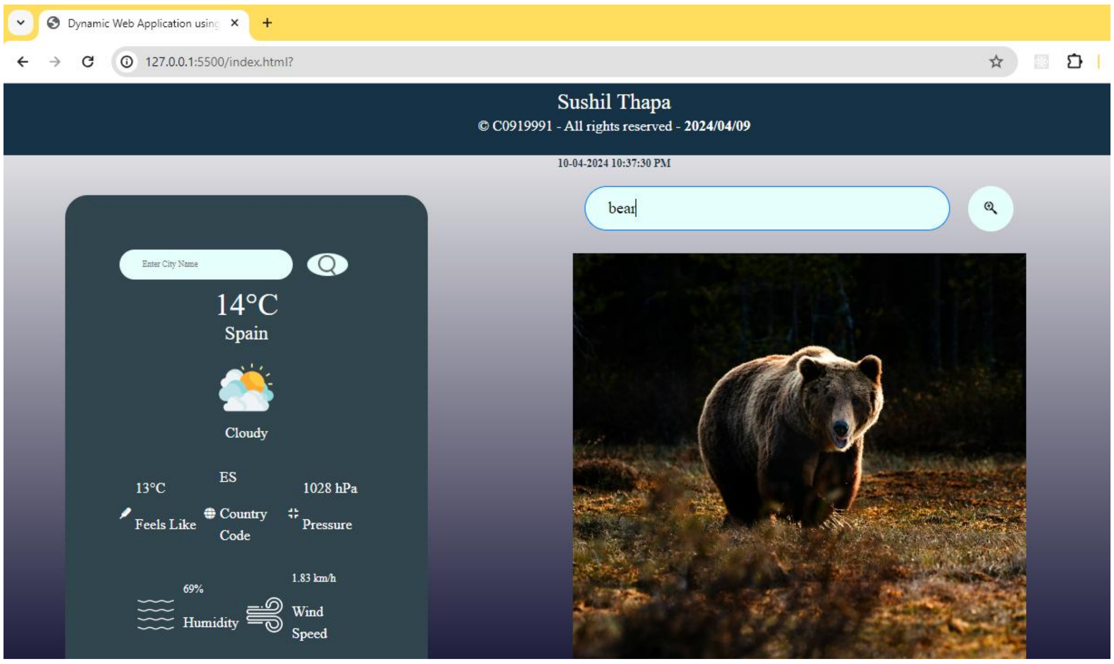


Image 13 Test for category Bear

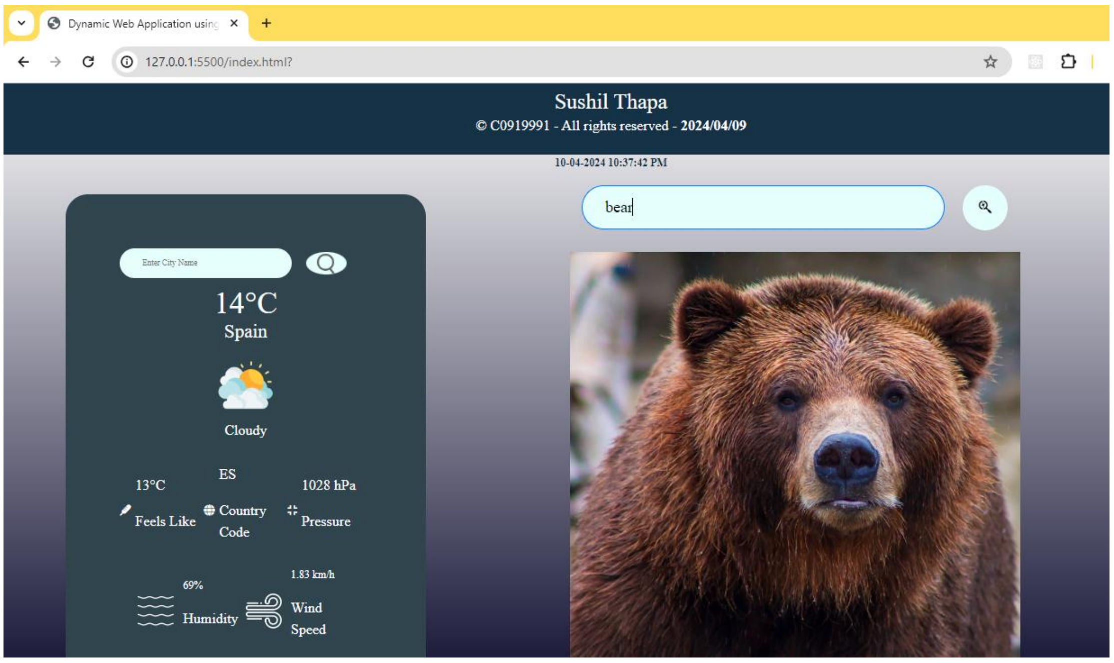


Image 14 Test for category Bear II

3.2.2 Car

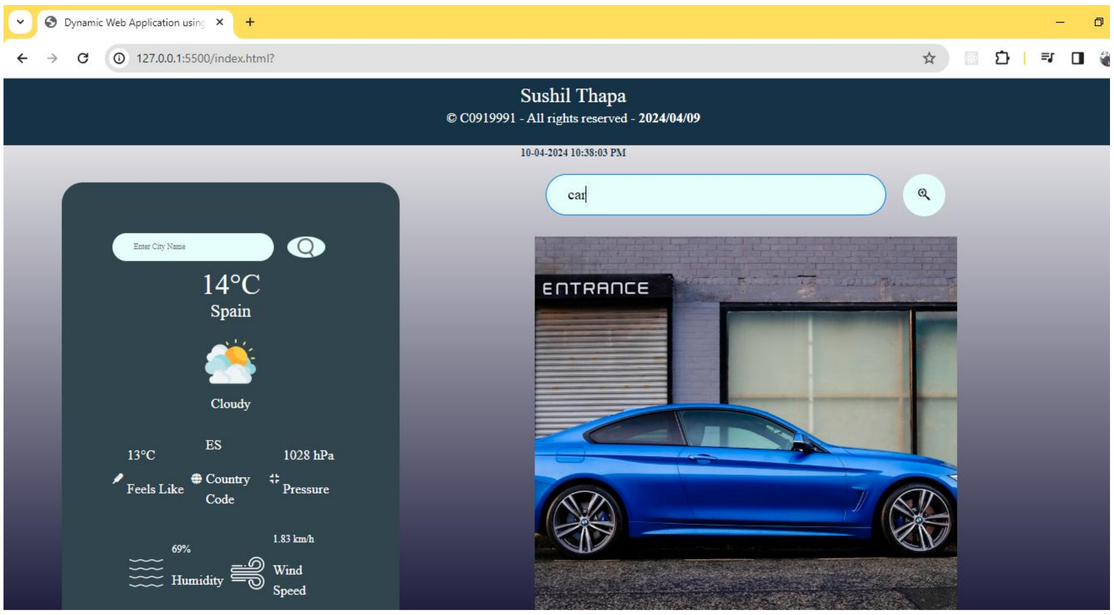


Image 15 Test for category Car

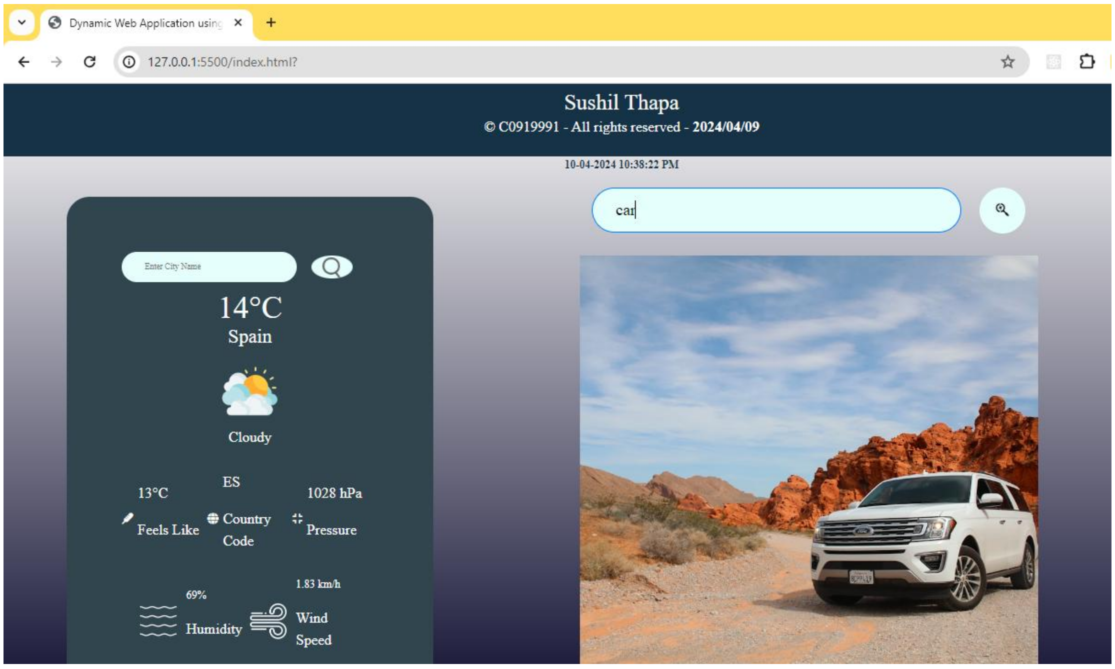


Image 16 Test for category Car II

3.2.3 Space

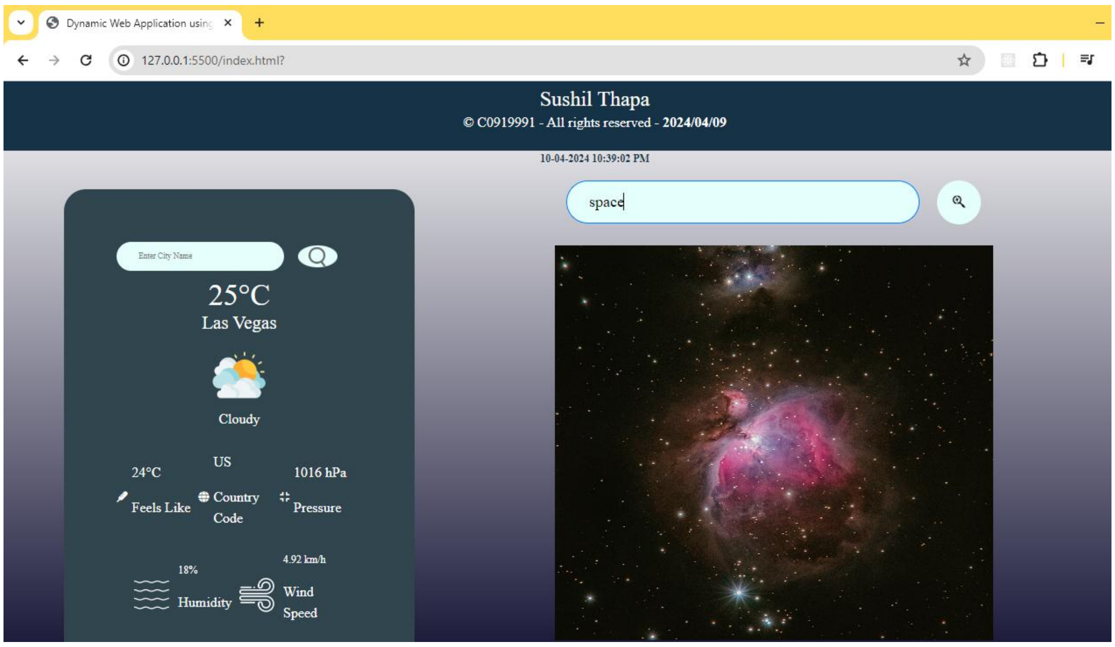


Image 17 Test for category Space

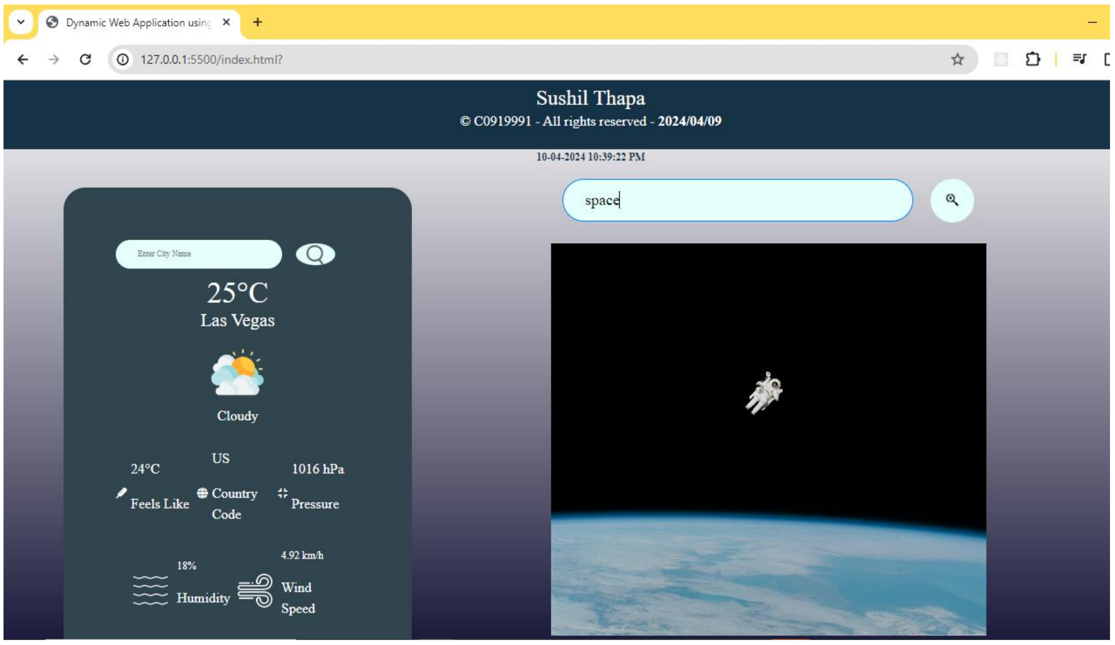


Image 18 Test for category Space II

3.2.4 Adventure

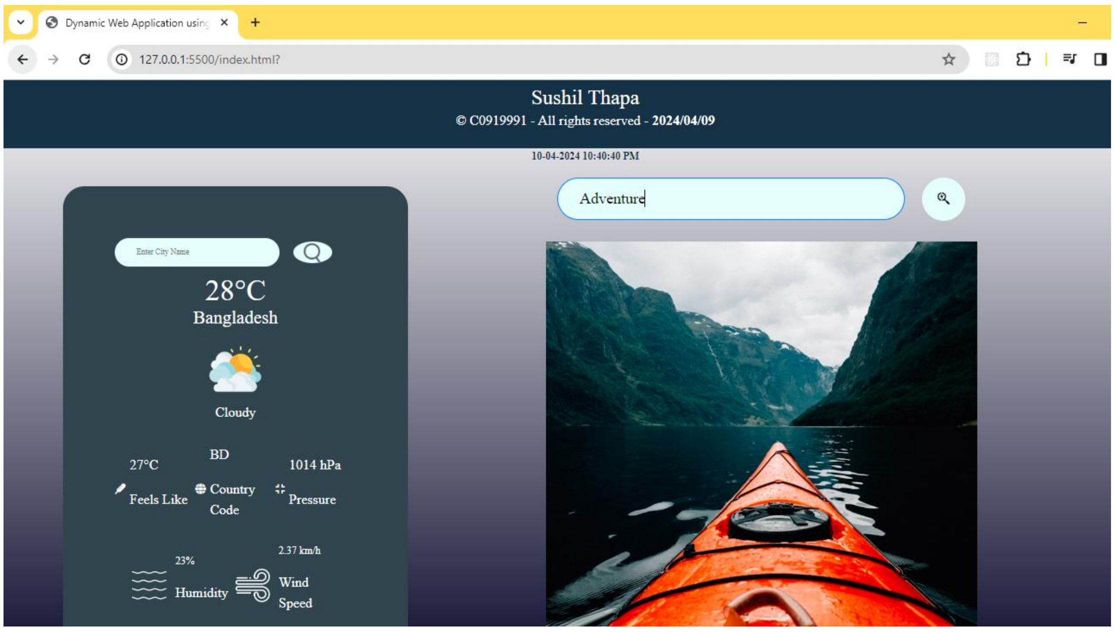


Image 19 Test for category Adventure

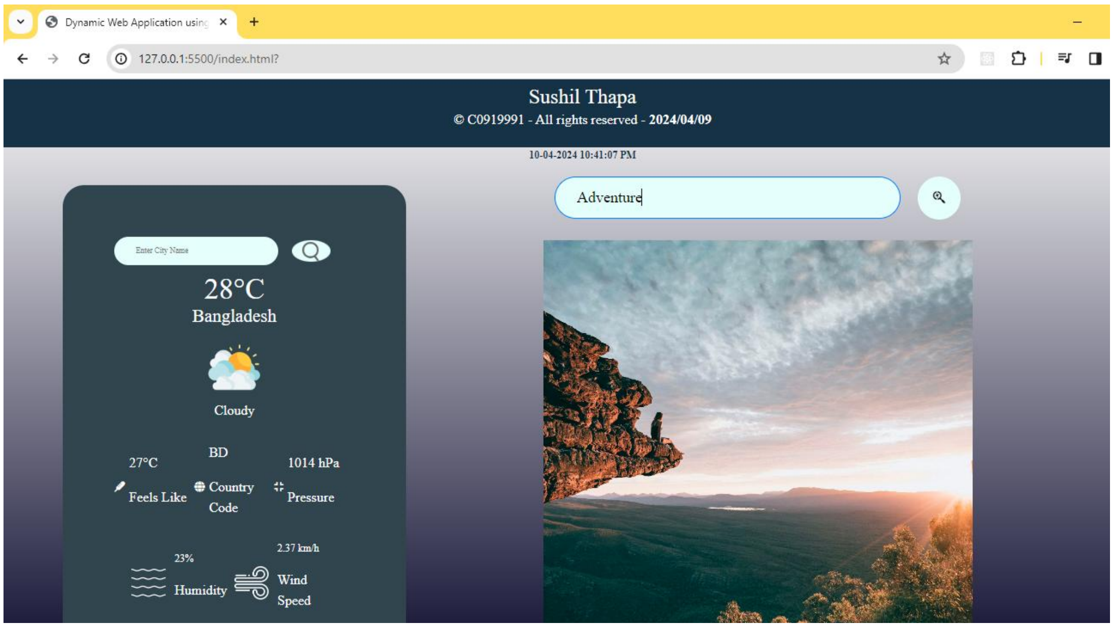
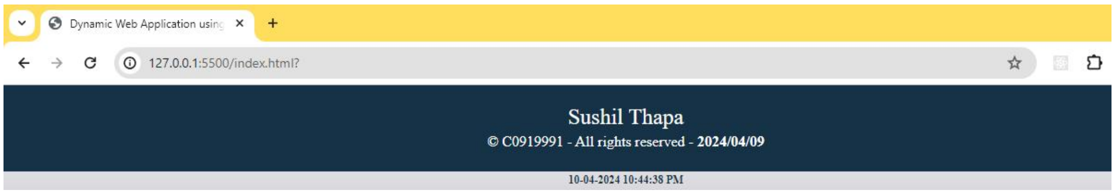


Image 20 Test for category Adventure II

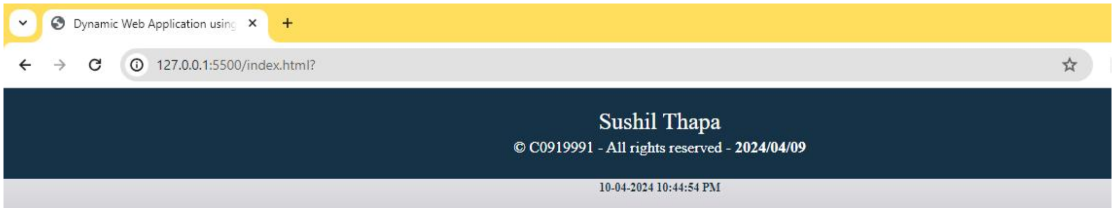
4. Testing on time, weather and picture refresh

4.1 Time



Sushil Thapa
© C0919991 - All rights reserved - **2024/04/09**
10-04-2024 10:44:38 PM

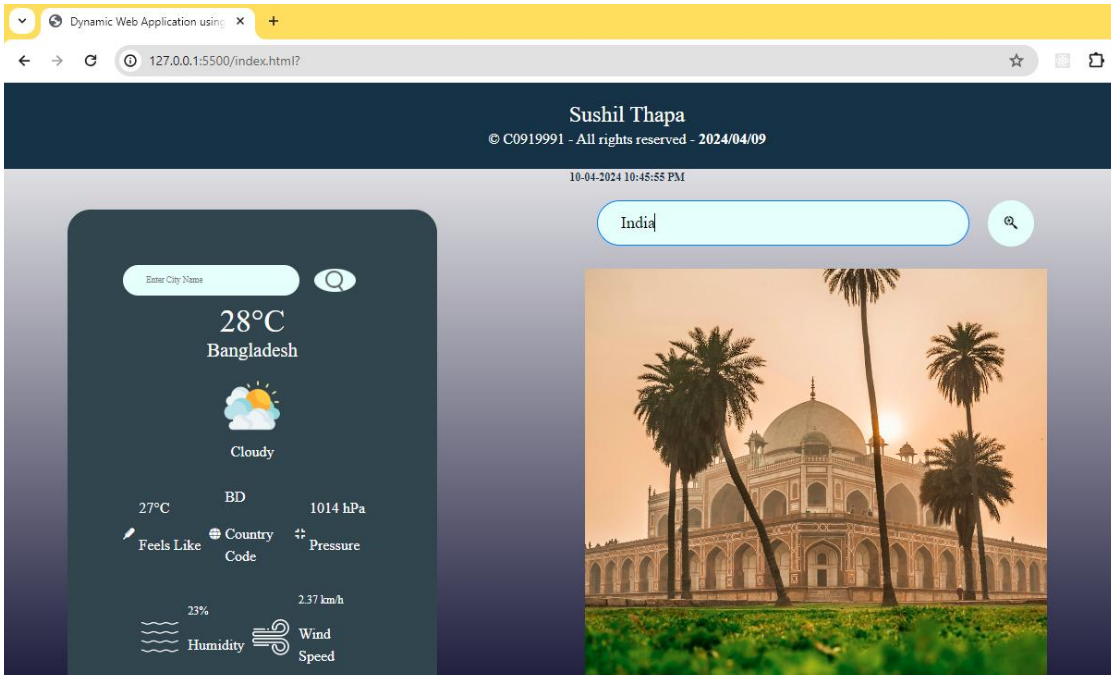
Image 21 Testing on time refresh each second



Sushil Thapa
© C0919991 - All rights reserved - **2024/04/09**
10-04-2024 10:44:54 PM

Image 22 Testing on time refresh each second II

4.2 Picture



Sushil Thapa
© C0919991 - All rights reserved - **2024/04/09**
10-04-2024 10:45:55 PM

Image 23 Testing on picture refresh after 20 second

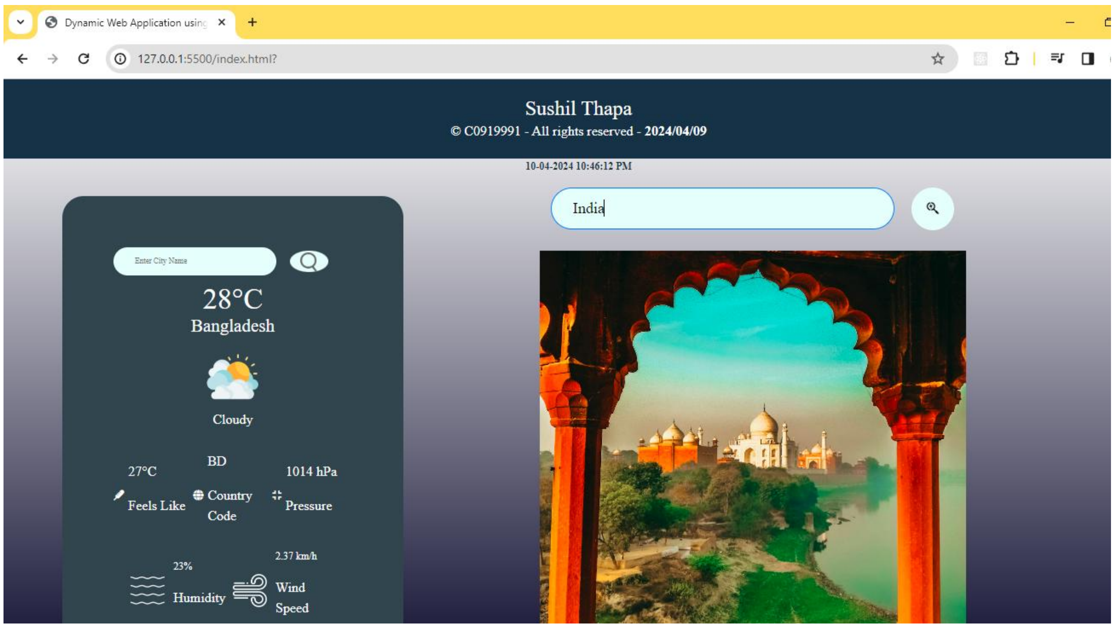


Image 24 Testing on picture refresh after 20 second II

4.3 Weather

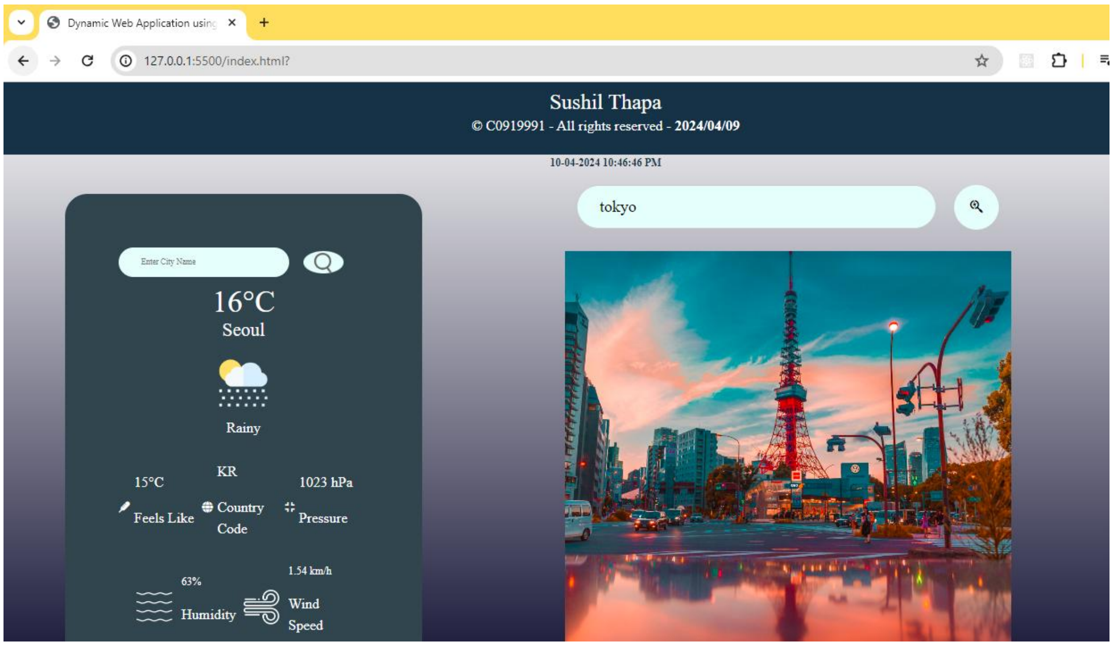
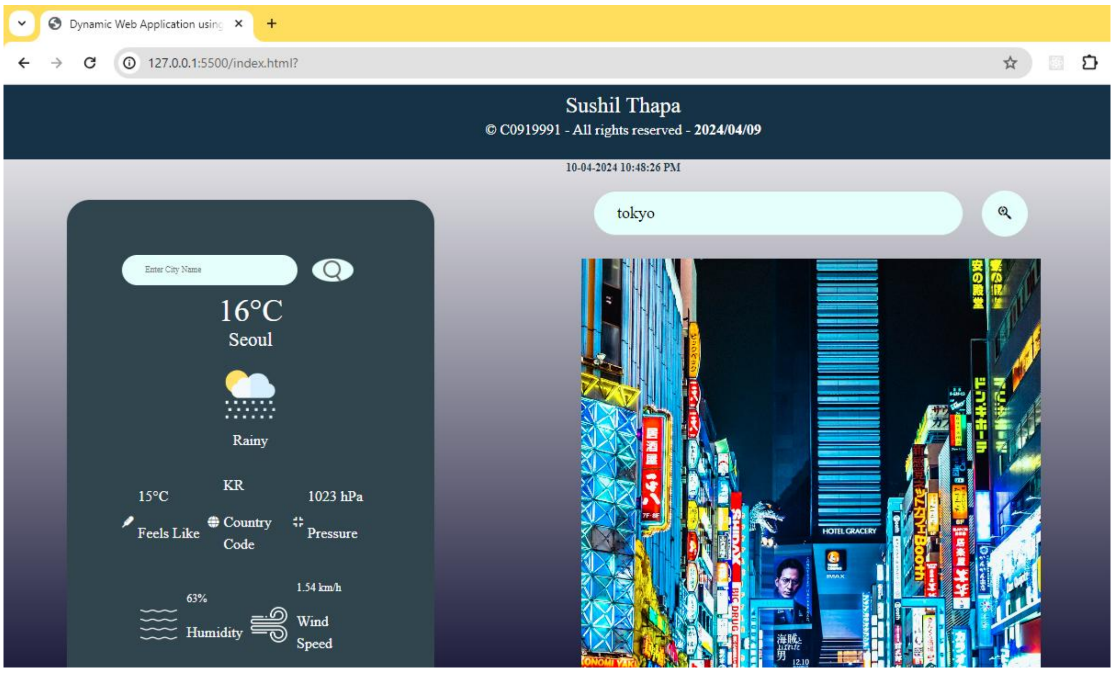


Image 25 Testing on weather refresh after 1 minute



Sushil Thapa
© C0919991 - All rights reserved - **2024/04/09**

10-04-2024 10:48:26 PM

tokyo

Enter City Name

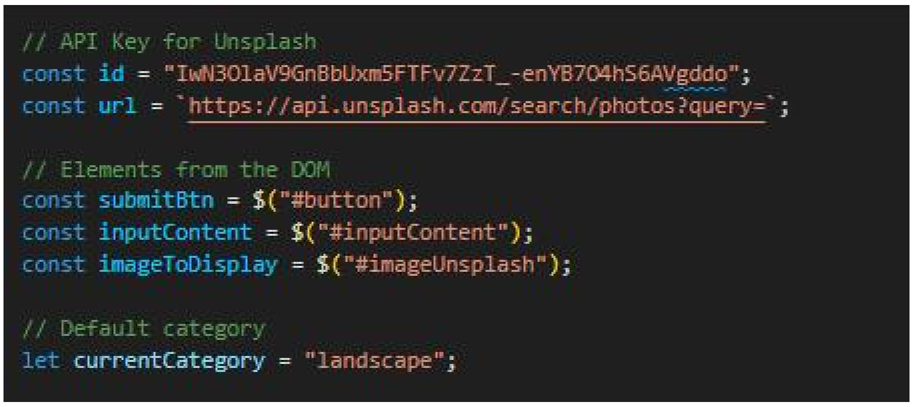
16°C
Seoul

Rainy

15°C Feels Like
KR Country Code
1023 hPa Pressure
63% Humidity
1.54 km/h Wind Speed

Image 26 Testing on weather refresh after 1 minute II

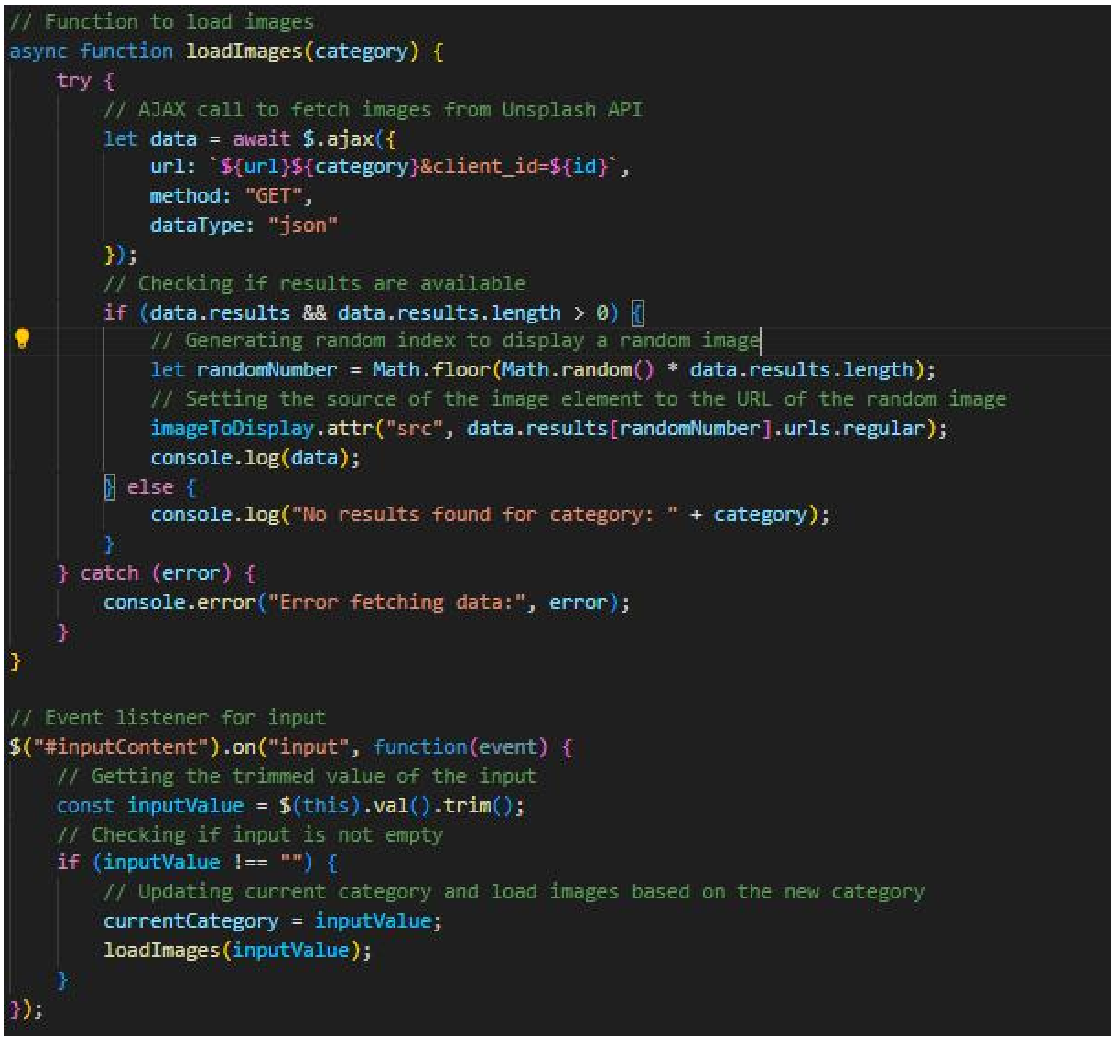
5. Notes on Coding



```
// API Key for Unsplash
const id = "IwN3OlaV9GnBbUxm5FTFv7ZzT_-enYB7O4hS6AVgddo";
const url = `https://api.unsplash.com/search/photos?query=`;

// Elements from the DOM
const submitBtn = $("#button");
const inputContent = $("#inputContent");
const imageToDisplay = $("#imageUnsplash");

// Default category
let currentCategory = "landscape";
```

Image 27 Accessing Selectors for Unsplash

This code from above figure initializes variables for an API key and URL needed to interact with the Unsplash API. It also selects specific elements from the webpage's DOM and sets a default category for photo searches. These preparations enable dynamic interaction with the API and user input handling on the webpage.

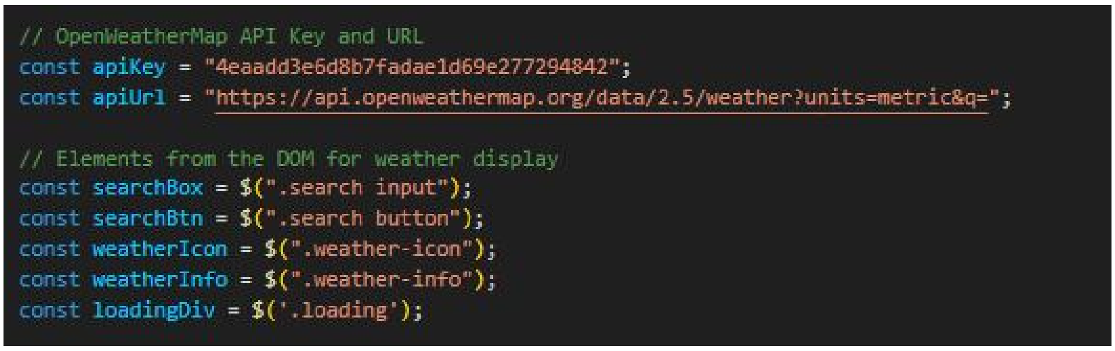


```
// Function to load images
async function loadImages(category) {
    try {
        // AJAX call to fetch images from Unsplash API
        let data = await $.ajax({
            url: `${url}${category}&client_id=${id}`,
            method: "GET",
            dataType: "json"
        });
        // Checking if results are available
        if (data.results && data.results.length > 0) {
            // Generating random index to display a random image
            let randomNumber = Math.floor(Math.random() * data.results.length);
            // Setting the source of the image element to the URL of the random image
            imageToDisplay.attr("src", data.results[randomNumber].urls.regular);
            console.log(data);
        } else {
            console.log("No results found for category: " + category);
        }
    } catch (error) {
        console.error("Error fetching data:", error);
    }
}

// Event listener for input
$("#inputContent").on("input", function(event) {
    // Getting the trimmed value of the input
    const inputValue = $(this).val().trim();
    // Checking if input is not empty
    if (inputValue !== "") {
        // Updating current category and load images based on the new category
        currentCategory = inputValue;
        loadImages(inputValue);
    }
});
```

Image 28 Main Function and Event listener function for Unsplash

The above figure code segment defines a function, loadImages(category), responsible for fetching and displaying images from the Unsplash API based on a specified category. It utilizes jQuery's $.ajax() method to asynchronously retrieve data, constructing the URL dynamically with the provided category and API key (id). Upon processing the fetched data, it checks for the availability of images. If images are found, a random image URL is selected and assigned to an image element (imageToDisplay). In case no images are found for the specified category, a message is logged indicating the absence of images.

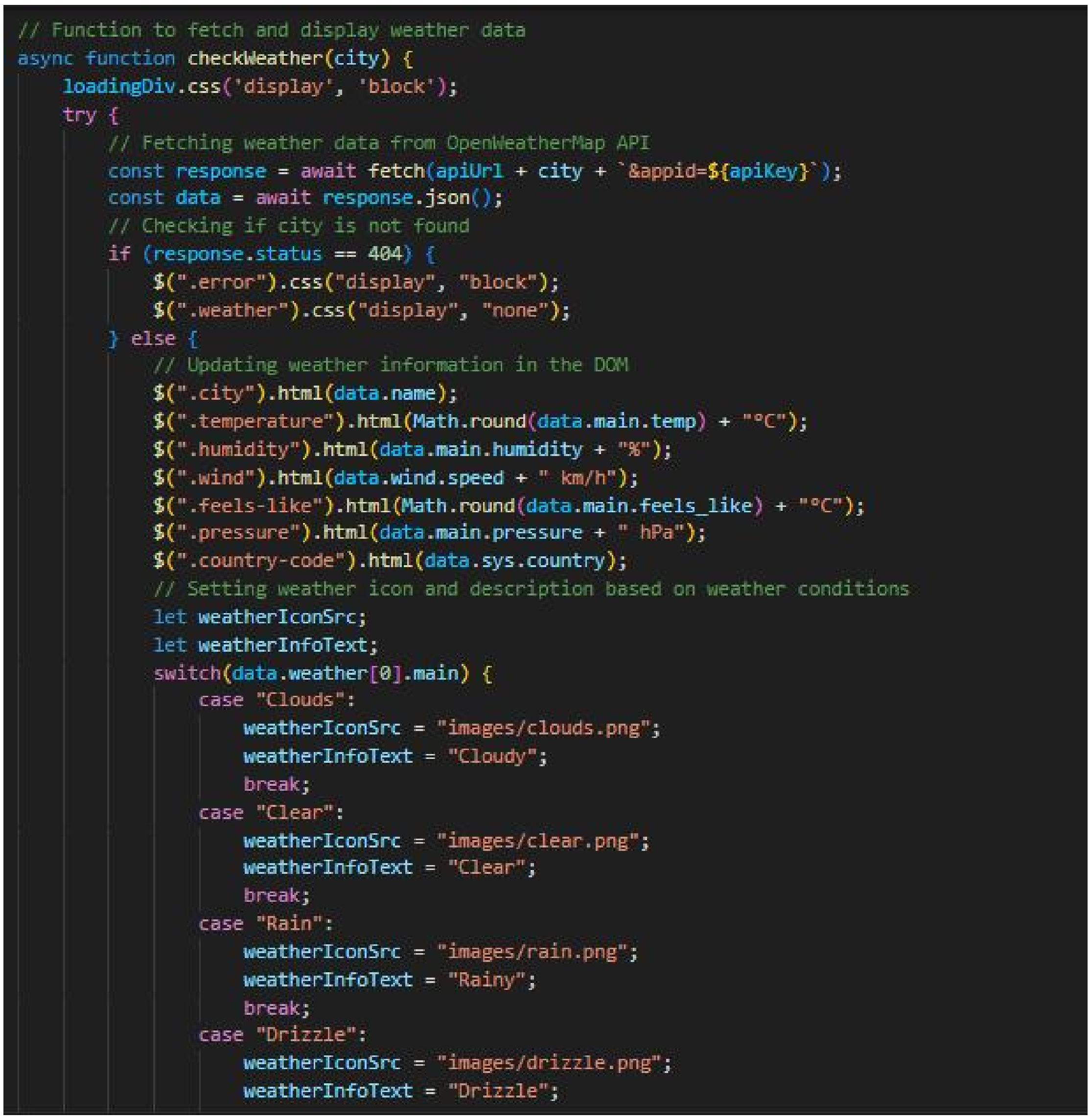
Additionally, an event listener is established for the "input" event on an element identified by the id "inputContent". This listener triggers a function whenever the input content changes. The function retrieves the trimmed value of the input field, updates the currentCategory variable with the new value if it is not empty, and invokes the loadImages() function with the updated category as an argument. This setup facilitates dynamic image loading based on user input, thereby enhancing the interactivity and functionality of the webpage.



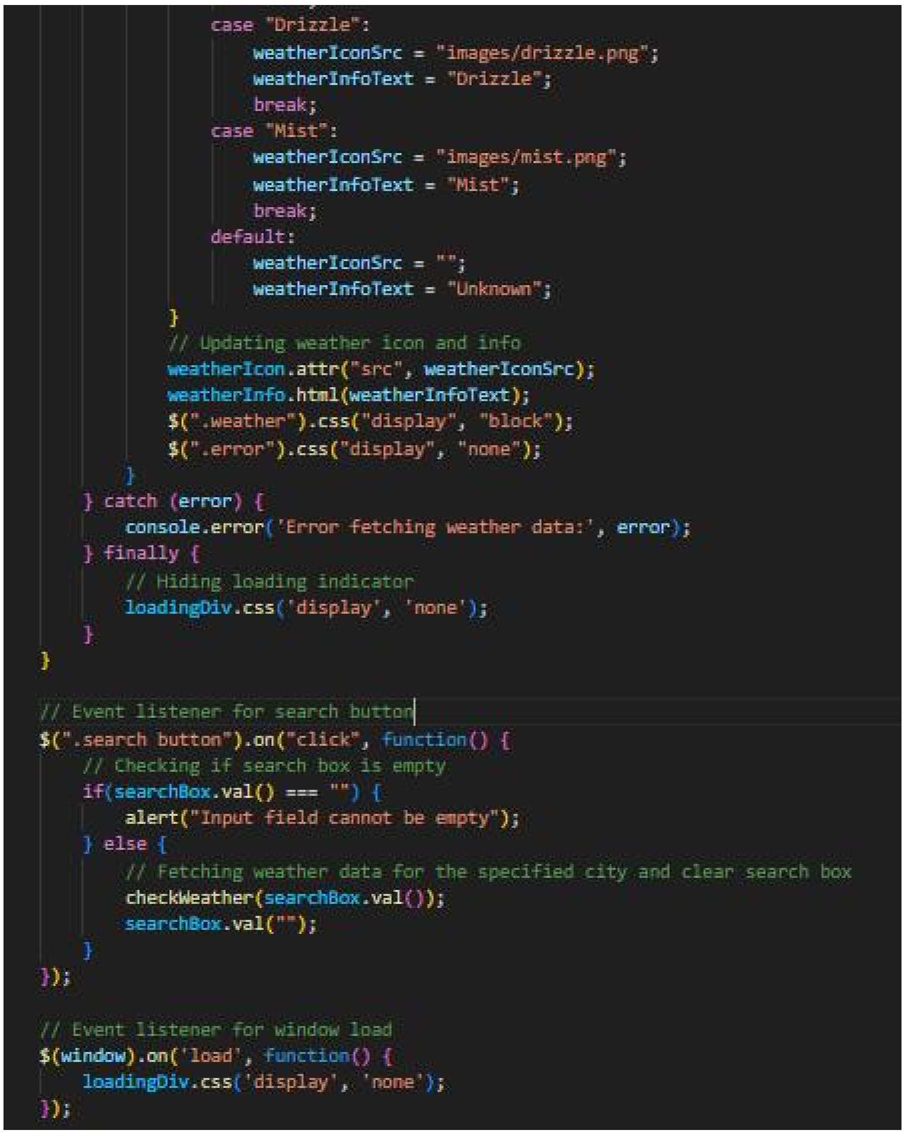
```
// OpenWeatherMap API Key and URL
const apiKey = "4eaadd3e6d8b7fadae1d69e277294842";
const apiUrl = "https://api.openweathermap.org/data/2.5/weather?units=metric&q=";

// Elements from the DOM for weather display
const searchBox = $(".search input");
const searchBtn = $(".search button");
const weatherIcon = $(".weather-icon");
const weatherInfo = $(".weather-info");
const loadingDiv = $('.loading');
```

Image 29 Accessing Selectors for Weather

The code initializes variables containing the API key and URL for the OpenWeatherMap API, crucial for accessing weather data. It also selects specific elements from the webpage's DOM using jQuery, such as search input fields, buttons, weather icons, and loading indicators.



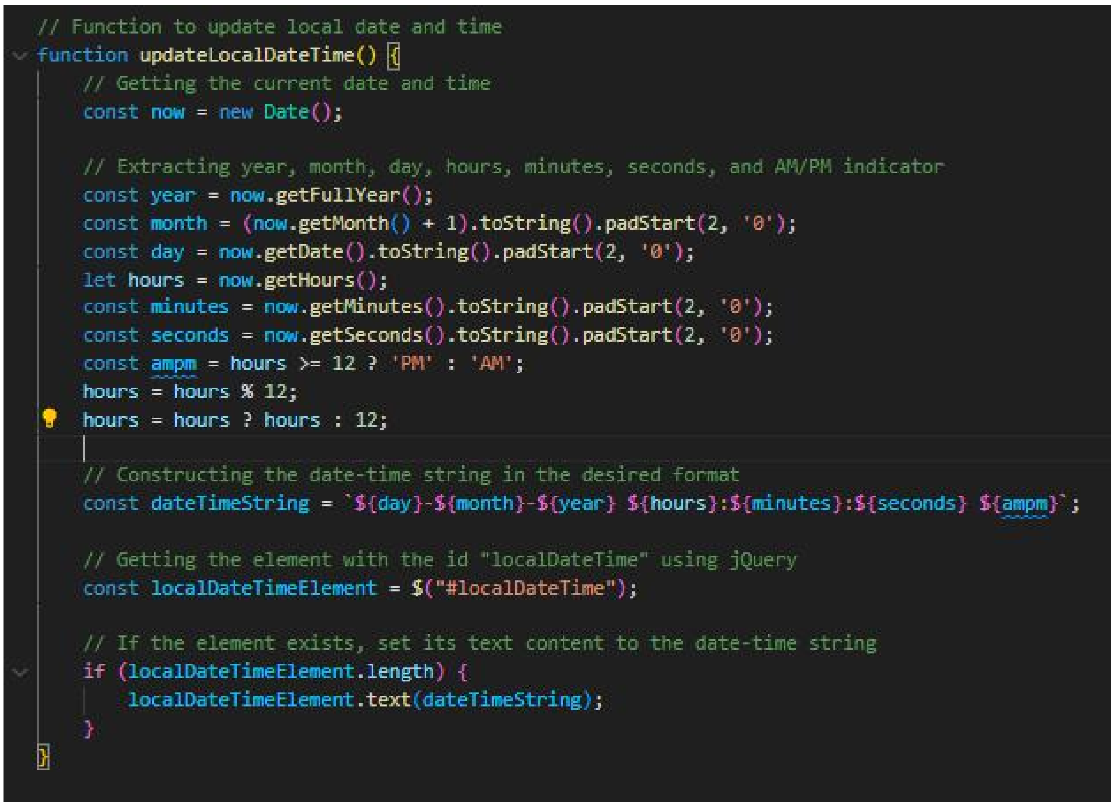
```
// Function to fetch and display weather data
async function checkWeather(city) {
    loadingDiv.css('display', 'block');
    try {
        // Fetching weather data from OpenWeatherMap API
        const response = await fetch(apiUrl + city + `&appid=${apiKey}`);
        const data = await response.json();
        // Checking if city is not found
        if (response.status == 404) {
            $(".error").css("display", "block");
            $(".weather").css("display", "none");
        } else {
            // Updating weather information in the DOM
            $(".city").html(data.name);
            $(".temperature").html(Math.round(data.main.temp) + "°C");
            $(".humidity").html(data.main.humidity + "%");
            $(".wind").html(data.wind.speed + " km/h");
            $(".feels-like").html(Math.round(data.main.feels_like) + "°C");
            $(".pressure").html(data.main.pressure + " hPa");
            $(".country-code").html(data.sys.country);
            // Setting weather icon and description based on weather conditions
            let weatherIconSrc;
            let weatherInfoText;
            switch(data.weather[0].main) {
                case "Clouds":
                    weatherIconSrc = "images/clouds.png";
                    weatherInfoText = "Cloudy";
                    break;
                case "Clear":
                    weatherIconSrc = "images/clear.png";
                    weatherInfoText = "Clear";
                    break;
                case "Rain":
                    weatherIconSrc = "images/rain.png";
                    weatherInfoText = "Rainy";
                    break;
                case "Drizzle":
                    weatherIconSrc = "images/drizzle.png";
                    weatherInfoText = "Drizzle";
```



```
                    case "Drizzle":
                        weatherIconSrc = "images/drizzle.png";
                        weatherInfoText = "Drizzle";
                        break;
                    case "Mist":
                        weatherIconSrc = "images/mist.png";
                        weatherInfoText = "Mist";
                        break;
                    default:
                        weatherIconSrc = "";
                        weatherInfoText = "Unknown";
                }
                // Updating weather icon and info
                weatherIcon.attr("src", weatherIconSrc);
                weatherInfo.html(weatherInfoText);
                $(".weather").css("display", "block");
                $(".error").css("display", "none");
            }
    } catch (error) {
        console.error('Error fetching weather data:', error);
    } finally {
        // Hiding loading indicator
        loadingDiv.css('display', 'none');
    }
}

// Event listener for search button
$(".search button").on("click", function() {
    // Checking if search box is empty
    if(searchBox.val() === "") {
        alert("Input field cannot be empty");
    } else {
        // Fetching weather data for the specified city and clear search box
        checkWeather(searchBox.val());
        searchBox.val("");
    }
});

// Event listener for window load
$(window).on('load', function() {
    loadingDiv.css('display', 'none');
});
```

Image 30 Main function for weather api call

The above figure code segment defines a JavaScript function named checkWeather(city) tasked with fetching and displaying weather data from the OpenWeatherMap API based on a specified city. Initially, it toggles the display of a loading indicator to inform users that data is being fetched. The function then makes an asynchronous request to the OpenWeatherMap API using the fetch() method, constructing the API URL dynamically with the city name and API key. Upon receiving the API response, the function processes the data, checking if the city exists. If the city is not found, it displays an error message and hides any existing weather information. Conversely, if the city is found, the function updates various weather-related details in the webpage's DOM, including city name, temperature, humidity, wind speed, feels-like temperature, pressure, and country code. Additionally, it determines the appropriate weather icon and description based on the retrieved weather conditions. Once the data is updated in the DOM, the loading indicator is hidden, and the weather information is displayed to the user.

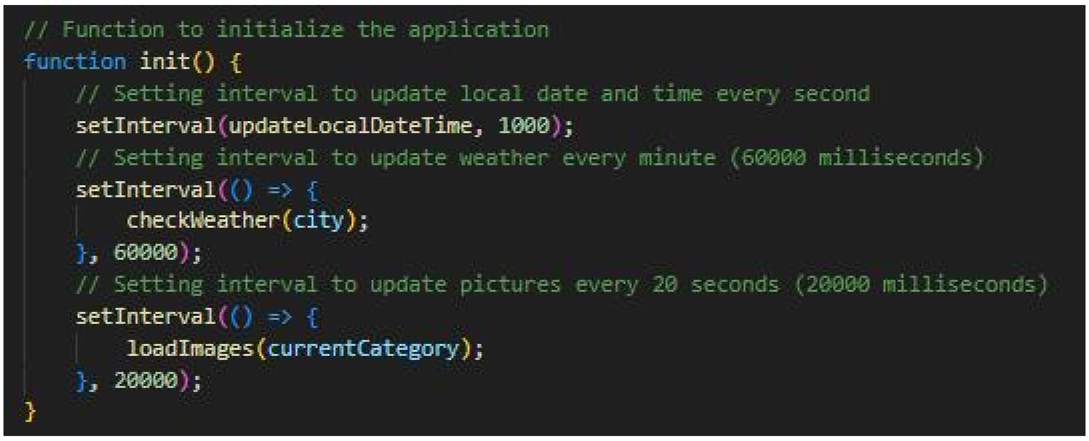


```
// Function to update local date and time
function updateLocalDateTime() {
    // Getting the current date and time
    const now = new Date();

    // Extracting year, month, day, hours, minutes, seconds, and AM/PM indicator
    const year = now.getFullYear();
    const month = (now.getMonth() + 1).toString().padStart(2, '0');
    const day = now.getDate().toString().padStart(2, '0');
    let hours = now.getHours();
    const minutes = now.getMinutes().toString().padStart(2, '0');
    const seconds = now.getSeconds().toString().padStart(2, '0');
    const ampm = hours >= 12 ? 'PM' : 'AM';
    hours = hours % 12;
    hours = hours ? hours : 12;

    // Constructing the date-time string in the desired format
    const dateTimeString = `${day}-${month}-${year} ${hours}:${minutes}:${seconds} ${ampm}`;

    // Getting the element with the id "localDateTime" using jQuery
    const localDateTimeElement = $("#localDateTime");

    // If the element exists, set its text content to the date-time string
    if (localDateTimeElement.length) {
        localDateTimeElement.text(dateTimeString);
    }
}
```

Image 31 Main Function of Update Time and Date

The above figure code defines a JavaScript function, updateLocalDateTime(), tasked with updating the local date and time displayed on a webpage. It first retrieves the current date and time, extracting components such as year, month, day, hours, minutes, and seconds, adjusting the hours to a 12-hour format. It then constructs a date-time string in the desired format.

Using jQuery, the function selects the element with the id "localDateTime" from the DOM. If found, it updates the element's text content with the constructed date-time string, dynamically updating the displayed date and time on the webpage. This functionality ensures real-time updates without requiring a page refresh, enhancing the user experience with accurate and up-to-date information. Overall, it facilitates the dynamic display of local date and time information, improving user interaction and accessibility.

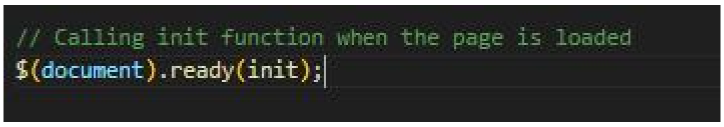


```javascript
// Function to initialize the application
function init() {
    // Setting interval to update local date and time every second
    setInterval(updateLocalDateTime, 1000);
    // Setting interval to update weather every minute (60000 milliseconds)
    setInterval(() => {
        checkWeather(city);
    }, 60000);
    // Setting interval to update pictures every 20 seconds (20000 milliseconds)
    setInterval(() => {
        loadImages(currentCategory);
    }, 20000);
}
```

Image 32 Init Function to set Interval

The above figure code defines a JavaScript function named init() responsible for initializing the application by setting up various intervals for updating different components of the webpage dynamically. Firstly, it establishes an interval that calls the updateLocalDateTime() function every second, ensuring that the local date and time displayed on the webpage are continuously updated to reflect the current time accurately.

Additionally, the init() function orchestrates intervals for updating weather information and displayed images. It schedules the checkWeather(city) function to run every minute (60000 milliseconds) for regular weather updates based on the specified city. Additionally, it schedules the loadImages(currentCategory) function to run every 20 seconds (20000 milliseconds) to refresh displayed images, offering users a dynamic visual experience. By managing these intervals, init() ensures continuous updates of crucial application components like local date and time, weather data, and images, enhancing user interaction by keeping content relevant, accurate, and engaging.



```javascript
// Calling init function when the page is loaded
$(document).ready(init);
```

Image 33 Onload to run init function

This above line of code uses jQuery's $(document).ready() function to execute the init() function when the page has finished loading. It ensures that the initialization tasks, such as setting up intervals for updating various components, occur at the appropriate time, improving the user experience by ensuring functionality is available once the page is fully loaded.