

INFO1111: Computing 1A Professionalism

2023 Semester 1

Self-Learning Report

Submission number: 1

Github link: <https://github.com/Ashley-wmx/Info1111-self-learning>

Student name	Mingxuan Wang
Student ID	520212462
Topic	JavaScript
Levels already achieved	Level A Level B level C
Levels in this report	Level A Level B level C level D

Contents

1.	Level A: Initial Understanding	2
1.1.	Level A Demonstration	2
1.2.	Learning Approach	2
1.3.	Challenges and Difficulties	2
1.4.	Learning Sources	3
1.5.	Application artifacts	3
2.	Level B: Basic Application	9
2.1.	Level B Demonstration	9
2.2.	Application artifacts	9
3.	Level C: Deeper Understanding	11
3.1.	Strengths	11
3.2.	Weaknesses	11
3.3.	Usefulness	11
3.4.	Key Question 1	11
3.5.	Key Question 2	11
4.	Level D: Evolution of skills	12
4.1.	Level D Demonstration	12
4.2.	Application artifacts	12
4.3.	Alternative tools/technologies	18
4.4.	Comparative Analysis	18

1. Level A: Initial Understanding

1.1. Level A Demonstration

Install JavaScript and learn its basic grammar and functions(syntax), then install Vue.js, create a project, run the very first demo, and finally, learn about event-based triggers.

1.2. Learning Approach

To approach self-learning in JavaScript, I focused more on online resources and practicing on the web. First, I search online resources, such as tutorials, youtube videos, and self-learning web page, to help me learn the basics of JavaScript. I made sure to choose resources that were easy to understand and suited to my level of knowledge. Also, I looked for resources that had practical exercises and mini-projects, so that I could apply what I learned and reinforce my understanding of each topic. Then I practiced writing code on online code editors, such as Codefinity and JSFiddle, to gain experience with JavaScript. These practice websites allowed me to experiment with code, test my skills, and get feedback during learning. To ensure that my skills remain current and relevant, I stay informed about the latest developments in the JavaScript ecosystem. I achieve this by regularly reading online blogs, newsletters, and websites that cover updates and advancements in the field. This approach helping me learn the latest techniques in JavaScript programming.

1.3. Challenges and Difficulties

When I first began learning JavaScript, I encountered several challenges that initially appeared difficult. One of the most significant hurdles was understanding the syntax and how it worked. It was an entirely new thought process for me, as I had to grasp concepts like closures, callbacks, and promises, which were quite distinct from those in other programming languages. Another challenge I faced was navigating the vast ecosystem of JavaScript frameworks and libraries, such as React, Angular, Vue, jQuery, Lodash, and Axios. For a beginner, the sheer number of available tools can be overwhelming, and choosing the right ones to learn and use effectively can be a daunting task. Creating a to-do list webpage is full of challenges. I needed to learn HTML and CSS as well to design the page's structure and appearance, and then use JavaScript to add interactivity and functionality. Although HTML and CSS were not as difficult as JavaScript, they still required effort to create a complete webpage.

1.4. Learning Sources

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.).	Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?
https://www.youtube.com/watch?v=W6NZfC05SIk&t=419s	Help me know about JavaScript at beginning
https://https://www.w3schools.com/js/default.asp	I used it to search for functions and syntax
https://www.javascripttutorial.net/	I used it to search for functions and syntax
https://https://vuejs.org/guide/introduction.html	Help me to learn vue.js
https://codefinity.com/profile/my-home	Help me to do some practice

1.5. Application artifacts

1. Basic syntax(grammar and functions)

- Variable

Definition Similar to other programming languages, JavaScript can store data in memory and assign it a variable name. By using this variable name, we can access the data stored at that memory location for later use.

Name of variables When creating variable names, it is essential to make them meaningful and avoid using reserved keywords. Additionally, variable names cannot start with a number and should not include spaces or hyphens. It is also crucial to consider the case sensitivity of the language when naming variables.

Declaration When declaring variables, we can either declare them on a single line separated by commas or on separate lines. However, multiple variable declarations in a single statement cannot be assigned the same value.

- Data Type

Primitive data types and Object data type Primitive data type includes string, number, boolean, null and undefined, object data type includes array, object and function. Primitive data types are simple, immutable values in JavaScript, while object data types are more complex.

String A string can be any text in quotes. we can use single or double quotation marks. We can use the index position to access each character in the string.

Number Numbers in JavaScript can be integers or floating-point values.

Boolean A boolean can only have one of two values: true or false. They're often used to help make choices, manage how the code runs, and show the status of a program or situation.

Null and Undefined In JavaScript, "null" and "undefined" are special values used to indicate the absence of a value or that a variable hasn't been assigned value. "null" is a value that represents the intentional absence of any object or value. "undefined" means that a variable has been declared, but it hasn't been given a value yet. Both "null" and "undefined" are often used to check if something exists or if a value has been assigned before performing an operation on it.

Array An array can store a collection of values. Each value in an array is called an element, and each element has an associated index, which is numeric values starting from 0.

Object An object in JavaScript is like a container that can store a collection of related information in the form of key-value pairs. The keys are usually strings, and the values can be any data type, including other objects. Objects are useful for organizing and representing real-world things or data in a structured and easy-to-use format. They are a fundamental feature of JavaScript and provide the foundation for the language's object-oriented programming abilities.

Function A function is a reusable piece of code that performs a specific computation. Functions allow for modular, well-organized, and maintainable code by enabling user to break down complex tasks into smaller parts. They can take input as arguments and return a value as a result of their execution.

2. Install Vue.js, create a project, run the very first demo, general introduction for vue.js

- About Vue.js

Advantages Vue.js is a JavaScript framework that simplifies web development by using single-file components, combining HTML, CSS, and JavaScript. It allows for independent testing of each component and has easy-to-understand code for developers. Additionally, Vue.js has a small file size, which means that it can load quickly, even with slower internet connections. This can improve the overall performance of the application and reduce the likelihood of users leaving the page due to long loading times.

- Hello Vue

Description This code is a simple HTML file that demonstrates how to use Vue.js, a JavaScript framework that helps developers build interactive user interfaces for their web applications. It includes the Vue.js library using a script tag, which is required for the Vue instance to work. The Vue instance is created by targeting an element with the ID "app". The instance has a single data property called "message", which is set to 'Hello Vue!'. The message property is then displayed within the "app" div element using Vue's built-in syntax (`message`). This allows the text "Hello Vue!" to be displayed on the web page.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title></title>
6 <script src="vue.js" type="text/javascript" charset="utf-8"></script>
7 </head>
8 <body>
9 <div id="app">
10 {{ message }}
11 </div>
12
13 <script type="text/javascript">
14 var app = new Vue({
15   el: '#app',
16   data: {
17     message: 'Hello Vue!'
18   }
19 });
20 </script>
21
22 </body>
23 </html>

```

http://127.0.0.1:8048/vue.js教程/chapter 02 Creating a Vue Instance.html
PC Mode
Hello Vue!

- First Vue instance

Description In this code, we define the container element where Vue will be mounted, and we create an object with a single property called 'a'. This property is set to 1, and it initializes the instance's reactive data system. We use the '\$watch' method to keep track of changes to the 'a' property. Whenever the property changes, a callback function is executed. Finally, we set the value of the 'a' property to "test....". This triggers the '\$watch' callback function, which logs the new and old values of the 'a' property to the console. Overall, this code shows how Vue can be used to monitor and manipulate data in real-time on a webpage.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title></title>
<script src="vue.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
<div id="app">
  {{a}}
</div>

<script type="text/javascript">
var data = { a : 1 };
var vm = new Vue({
  el : "#app",
  data : data
});

vm.$watch('a', function(newVal, oldVal){
  console.log(newVal, oldVal);
})

vm.$data.a = "test...."

</script>

</body>
</html>

```

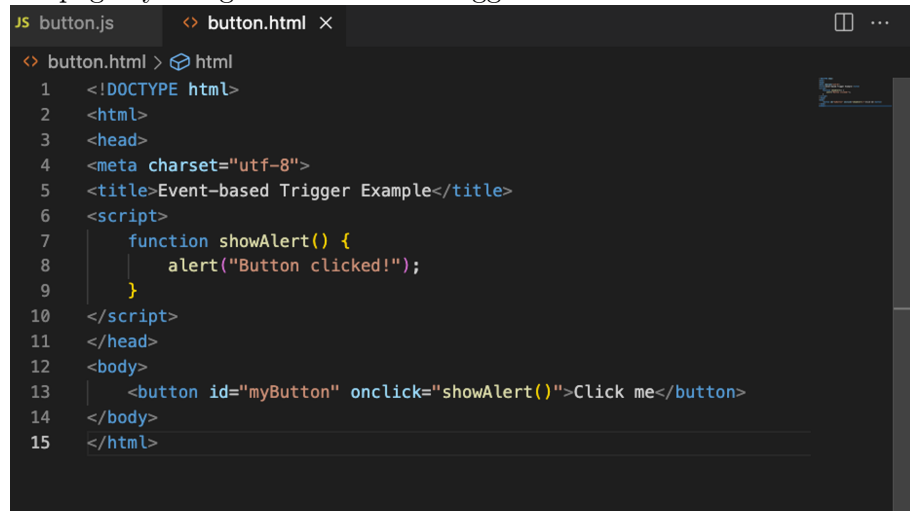
3. About event-based triggers

- Definition

In JavaScript, event-based triggers refer to executing specific functions or pieces of code in response to events, such as user interactions or changes in the application state. Events can include actions like clicking a button, hovering over an element, submitting a form or pressing a key on the keyboard.

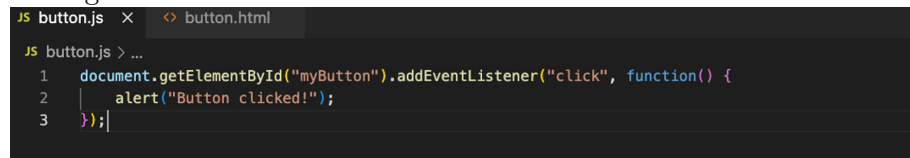
- Example 1

This code shows how to create a button on a webpage that does something when clicked. We create a button with the ID "myButton" in our HTML code. We then use JavaScript to define a function called "showAlert()" that will run when the button is clicked. When the button is clicked, an event is triggered which calls the showAlert() function. This function displays an alert box with the message "Button clicked!". Overall, this code demonstrates how to use JavaScript to add interactive functionality to a webpage by using an event-based trigger.



```
JS button.js  button.html X
<> button.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>Event-based Trigger Example</title>
6  <script>
7      function showAlert() {
8          alert("Button clicked!");
9      }
10 </script>
11 </head>
12 <body>
13     <button id="myButton" onclick="showAlert()">Click me</button>
14 </body>
15 </html>
```

In this example, an event listener is added to a button with the ID "myButton". The listener waits for the "click" event, and when the button is clicked, it triggers the anonymous function which displays an alert with the message "Button clicked!".



```
JS button.js X  button.html
JS button.js > ...
1  document.getElementById("myButton").addEventListener("click", function() {
2      alert("Button clicked!");
3  });
```



- Example

This code demonstrates how to use the 'keypress' event in JavaScript to create interactive functionality on a webpage. An event listener is added to an input field, which listens for the 'keypress' event and triggers a function when a key is pressed on the keyboard. If the key pressed is the 'Enter' key, an alert box is displayed with the message "You pressed the Enter!".

```
<!DOCTYPE html>
<html>
<head>
  <title>Keypress Event Example</title>
</head>
<body>
  <input type="text" id="Input" placeholder="Type something here...">

  <script>
    // Get the input element
    var input = document.getElementById('Input');

    // Add a keypress event listener to the input field
    input.addEventListener('keypress', function(event) {
      if (event.key === 'Enter') {
        alert('You pressed the Enter!');
      }
    });
  </script>
</body>
</html>
```

When a key is pressed on the keyboard while the input field is selected, the event listener is triggered. The listener checks if the key pressed was the 'Enter' key. If it was, it logs a message to the console with the text "You entered:". This code demonstrates how to use the 'keypress' event listener in JavaScript to create interactivity on a webpage by responding to user input in real-time.

```
var inputField = document.getElementById('Input');

inputField.addEventListener('keypress', function(event) {
  if (event.key === 'Enter') {
    console.log('You entered: ' + inputField.value);
  }
});
```


127.0.0.1:5500 says

You pressed the Enter!

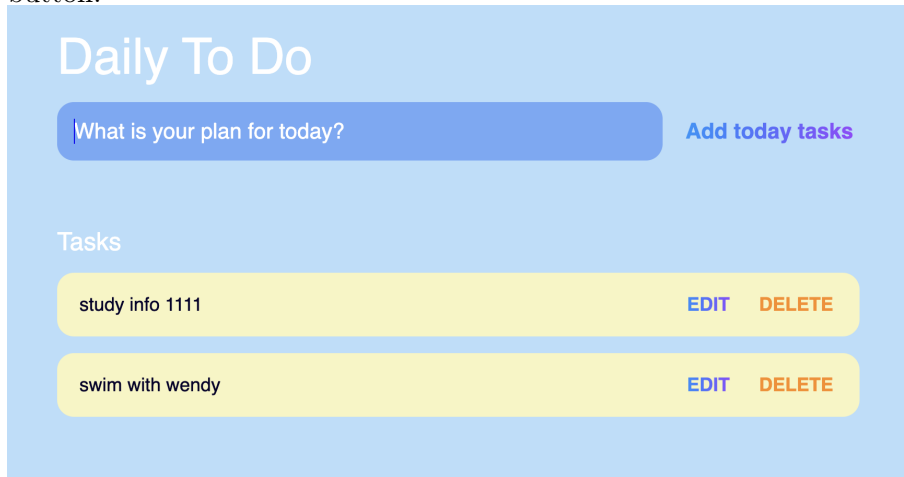
OK

ashley

2. Level B: Basic Application

2.1. Level B Demonstration

The Daily To-Do application is a web page designed to help users manage their daily tasks effectively. The app allows users to add tasks using an input field and displays them in a list format. Each task item comes with two buttons: "Edit" and "Delete." Users can modify tasks by clicking the "Edit" button and remove tasks by clicking the "Delete" button.



2.2. Application artifacts

Event listener for window load: The script starts with an event listener that waits for the window to load before executing the rest of the code. This ensures that all DOM elements are loaded and accessible before any manipulation occurs.

```
window.addEventListener('load', () => {
```

To interact with elements on a webpage, we need to create references to the important DOM elements. In this code, we use JavaScript to create references to the form, input field, and task list elements. We create these references using the `document.querySelector()` method, which allows us to select elements based on their CSS selectors. We assign these references to variables called `taskForm`, `taskInput`, and `taskList`. By creating these references, we can use them throughout our script to interact with the elements. For example, we can use `taskInput` to get the value of the input field or `taskList` to add or remove items from the list.

```
const taskForm = document.querySelector("#add-task-form");
const taskInput = document.querySelector("#task-input");
const taskList = document.querySelector("#tasks");
```

An event listener is attached to the form's 'submit' event. When the form is submitted, the event listener prevents the default form submission behavior and calls a function to create and display a new task element

```
taskForm.addEventListener('submit', (event) => {
  event.preventDefault();
```

When the form is submitted, a new task element is created using various DOM manipulation methods. The task element is a div with a class of "task". Inside this div, there are two child divs, one for the content and one for the actions. The content div has an input field with a class of "text". This input field is set to be read-only, and its initial

value is set to the value of the input field. The actions div has two buttons, "Edit" and "Delete". These buttons are created using the 'createElement()' method, and they are given appropriate classes and text using the 'classList.add()' and 'innerText' methods, respectively. Once the new task element is created, it is appended to the task list using the 'appendChild()' method. Then, the value of the input field in the form is set to an empty string, ready for the user to enter a new task.

```
const newTask = taskInput.value;

const taskElement = document.createElement('div');
//...

taskList.appendChild(taskElement);

taskInput.value = '';
```

An event listener is attached to the "Edit" button's 'click' event. When the "Edit" button is clicked, the event listener toggles the read-only attribute of the input field, allowing the user to edit the task text.

```
editTaskButton.addEventListener('click', () => {
  if (editTaskButton.innerText.toLowerCase() === "edit") {
    editTaskButton.innerText = "Save";
    taskInputElement.removeAttribute("readonly");
    taskInputElement.focus();
  } else {
    editTaskButton.innerText = "Edit";
    taskInputElement.setAttribute("readonly", "readonly");
  }
});
```

The event listener uses the 'removeChild()' method to remove the task element from the list. This means that when the "Delete" button is clicked, the corresponding task element will be removed from the web page.

```
deleteTaskButton.addEventListener('click', () => {
  taskList.removeChild(taskElement);
});
```

(There are comments in the two other files to explain code therefore the detailed explanation will not be shown here.)

3. Level C: Deeper Understanding

3.1. Strengths

JavaScript is a widely-used programming language which has numerous advantages. Its interoperability enables seamless integration with various technologies, encouraging collaboration and simplifying development.[1] Rich interfaces can be crafted, elevating user experiences through interactive, dynamic web elements.[2] With powerful frameworks like React, Angular, and Vue, developers can streamline workflows and create strong applications.[1] JavaScript's cross-browser compatibility ensures a consistent user experience across different browsers, while its simple syntax makes it accessible to beginners.[2] Overall, JavaScript's strengths contribute to the development of efficient, user-friendly, and adaptable web applications.

3.2. Weaknesses

JavaScript, while widely used, has its disadvantages. Client-side security concerns arise due to the potential for malicious users to exploit code for harmful purposes. Insufficient debugging facilities make it challenging to identify and fix issues efficiently.[1] Browser inconsistency can lead to varied user experiences across different browsers, requiring extra effort from developers to ensure compatibility. Additionally, speed issues may arise in JavaScript, particularly in computationally intensive tasks or when managing large data sets, leading to performance limitations. [2]

3.3. Usefulness

JavaScript is useful for creating interactive maps, as it employs various libraries and tools to streamline map creation and customization. Developers choose an appropriate mapping library, like Leaflet or Mapbox, to initialize the map, load map data, and customize map features. JavaScript allows adding custom markers, vector data, and other interactive elements, as well as integrating UI controls for seamless user interaction. Event listeners handle user actions, such as clicks or hovers, to update content or trigger further actions. By using JavaScript's powerful features, developers can build interactive maps that cater to diverse needs in web and mobile applications. [3]

3.4. Key Question 1

When should you use javascript? Not use javascript?

Use JavaScript for web development, desktop & mobile applications, server applications, and embedded systems & IoT when development speed, ease, and cross-platform compatibility are priorities. Avoid using JavaScript for complex web operations (use WASM), performance-sensitive desktop and mobile applications, high-performance server applications, and performance-critical embedded systems & IoT. We should consider alternatives for better performance and specific requirements in these cases. [4]

3.5. Key Question 2

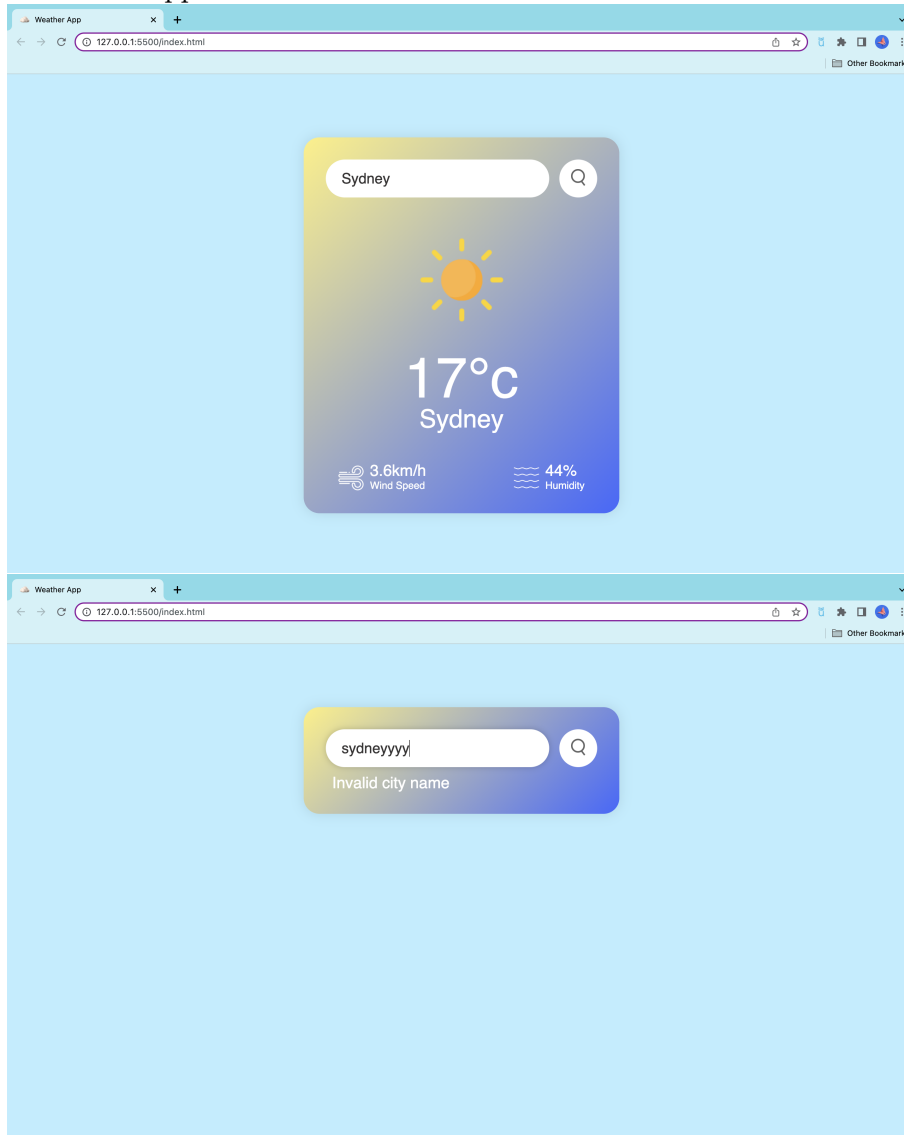
How does javascript make use of frameworks?

JavaScript frameworks offer developers a pre-built foundation for web applications and websites, including pre-written codes, components, and tools that streamline development. Developers select a framework based on project needs, application complexity, and their technical skills. Framework features like built-in functions, reusable components, and data binding enable the creation of interactive, efficient web applications or websites. Utilizing a JavaScript framework allows developers to concentrate on unique features and functionality, leading to efficient development, enhanced code quality, and simpler long-term maintenance.[5]

4. Level D: Evolution of skills

4.1. Level D Demonstration

This application is a web-based weather tool that offers real-time data for any city. Users simply enter a city's name, and the application displays the current temperature, humidity, and wind speed. It also dynamically updates to present relevant weather icons that reflect the current conditions. The application can be used repeatedly, providing consistent and accurate weather updates. Here are screenshots of valid and invalid input of weather app.



4.2. Application artifacts

The core functionality of the application revolves around a user entering the name of a city, which in turn prompts the application to display key weather indicators such as current temperature, humidity, and wind speed. The application enhances user experience by dynamically updating weather icons that reflect current conditions.

This application is built on three fundamental web technologies: HTML, CSS, and JavaScript. HTML structures the content on the webpage, laying out various elements

such as the input field for city names, the display area for weather data, and the weather icons. The styling and aesthetics are managed by CSS, which is used to create a responsive and visually appealing design. This design includes a dark-themed background, rounded input boxes, and a custom-styled display for weather data.

Finally, JavaScript powers the dynamic aspects of the application. It employs event listeners to respond to user inputs and uses asynchronous fetch calls to handle data requests based on the city name input by the user. The data retrieved is then dynamically inserted into the HTML, updating the page with relevant weather information. Furthermore, error handling mechanisms are used, displaying an error message for invalid or unknown city names. Here is the specific explanation for the code:

JavaScript

```
1  const apiKey = "52298ceb7461163029d5dfc03f71c941";
2  const apiUrl = "https://api.openweathermap.org/data/2.5/weather?units=metric&q=";
3
4  const searchInput = document.getElementById('searchInput');
5  const searchBtn = document.querySelector(".search button");
6  const weatherIcon = document.querySelector(".weather-icon");
7
8  // Mapping weather types to icons
9  const weatherIcons = {
10    Clouds: "weather/clouds.png",
11    Clear: "weather/clear.png",
12    Rain: "weather/rain.png",
13    Drizzle: "weather/drizzle.png",
14    Mist: "weather/mist.png",
15    Snow: "weather/snow.png",
16  };
```

In this section, constants are defined for the API key, the API URL, the search input field, the search button, and the weather icon image. The weatherIcons object is a dictionary that maps different weather conditions to their corresponding image URLs.

```
18  // Event listeners
19  searchInput.addEventListener('keydown', function(event) {
20    if (event.key === 'Enter') {
21      search();
22    }
23  });
24
25  searchBtn.addEventListener("click", search);
```

Event listeners are added to the search input and search button. When the user presses 'Enter' in the search input field, or clicks the search button, the search function is called.

```

25  searchBtn.addEventListener("click", search);
26
27  async function search() {
28      const searchTerm = searchInput.value;
29      if (!searchTerm) {
30          alert('Please enter a city name');
31          return;
32      }
33      console.log('Searching for:', searchTerm);
34      try {
35          const data = await getWeather(searchTerm);
36          displayWeather(data);
37      } catch (error) {
38          console.error('Error:', error);
39          displayError();
40      }
41  }

```

The search function fetches the weather data for the city entered by the user. If the input field is empty, an alert message is displayed, and the function returns. If there is an error while fetching the weather data, the error is logged, and an error message is displayed.

```

43  async function getWeather(city) {
44      const response = await fetch(apiUrl + city + `&appid=${apiKey}`);
45      if (!response.ok) {
46          throw new Error(`HTTP error! status: ${response.status}`);
47      }
48      return await response.json();
49  }

```

The getWeather function is an asynchronous function that fetches weather data for a given city from the OpenWeatherMap API. The function accepts one parameter, city, which is the name of the city for which the weather data is to be fetched. Inside the function, the fetch function is used to make a request to the API. The URL for the request is constructed by concatenating the apiUrl, city, and the apiKey. The fetch function returns a promise that resolves to the Response object representing the response to the request. The response.ok property is a boolean indicating whether the response was successful (status in the range 200-299) or not. If response.ok is false, an error is thrown with the status code of the response. Finally, the response is converted to a JSON object using the response.json() method, which also returns a promise. The await keyword is used again to pause the execution of the function until the promise is resolved. The function then returns the resolved value of the promise, which is the weather data in the form of a JSON object.

```

51 function displayWeather(data) {
52     if (!data) {
53         return;
54     }
55
56     // Show weather data
57     document.querySelector(".city").innerHTML = data.name;
58     document.querySelector(".temperature").innerHTML = Math.round(data.main.temp) + "°C";
59     document.querySelector(".humidity").innerHTML = data.main.humidity + "%";
60     document.querySelector(".wind").innerHTML = data.wind.speed + "km/h";
61
62     // Show corresponding weather icon
63     weatherIcon.src = weatherIcons[data.weather[0].main] || "weather/default.png";
64
65     // Show weather info and hide error message
66     document.querySelector(".weather").style.display = "block";
67     document.querySelector(".error").style.display = "none";
68 }

```

The `displayWeather` function is used to display the fetched weather data on the webpage. The function accepts one parameter, `data`, which is the weather data in the form of a JSON object. Inside the function, a series of `document.querySelector` calls are used to select various HTML elements and update their `innerHTML` property with the corresponding values from the `data` object. These HTML elements display the city name, temperature, humidity, and wind speed. The weather icon is updated by setting the `src` property of the `weatherIcon` to the corresponding value from the `weatherIcons` object. The key used to retrieve the value from the `weatherIcons` object is the `main` property of the first object in the `weather` array in the `data` object. If the `main` property doesn't match any of the keys in the `weatherIcons` object, a default image URL is used. Finally, the `display` style property of the `.weather` and `.error` HTML elements are updated to show the weather information and hide any error message.

```

70 function displayError() {
71     // Hide weather info and show error message
72     document.querySelector(".weather").style.display = "none";
73     document.querySelector(".error").style.display = "block";
74 }

```

The `displayError()` function is used to handle and display an error message on the web page when an error occurs, like when the weather data is not successfully retrieved.

HTML

```

<head>
  <meta name = "viewport" content="width=device-width, initial-scale=1.0">
  <title>Weather App </title>
  <link rel="stylesheet" href="style.css">
  <link rel="icon" href="weather/icon.ico" type="image/x-icon">
</head>

```

Inside the `<head>` tag, we have a meta tag specifying the viewport settings to make the page responsive. The title tag specifies the webpage title that is shown in the browser tab. There are two link tags, one linking an external CSS stylesheet (`style.css`), and the other setting a favicon (`icon.ico`).

```

<div class="search">
  <input type="text" id="searchInput" placeholder="Enter city name" spellcheck="false">
  <button onclick="search()"></button>
</div>

<div class = "error">
  <p>Invalid city name</p>
</div>

```

The search div contains an input box, where users can type the name of a city, and a

button with an onclick event listener that calls the search() function when clicked. The button also contains an image which serves as the button's label. The error div serves as a placeholder for showing error messages, such as when the input city name is invalid.

```
<div class="weather">
  
  <h1 class="temperature">23°C</h1>
  <h2 class="city">Sydney</h2>
  <div class="details">
    <div class="cols">
      
      <div>
        <p class="wind">20 km/h</p>
        <p>Wind Speed</p>
      </div>
    </div>
    <div class="cols">
      
      <div>
        <p class="humidity">30%</p>
        <p>Humidity</p>
      </div>
    </div>
  </div>
</div>
```

The weather div contains the actual weather data. This includes an image for the weather icon, as well as paragraphs for temperature, city name, wind speed, and humidity. Each of these paragraphs has a corresponding class for styling and scripting purposes.

```
49 <script src="script.js"></script>
50 </body>
51 </html>
```

Finally, after closing the weathercard div and main tag, we have a script tag linking to an external JavaScript file (script.js). The script is loaded at the end of the body to improve page loading performance.

CSS

```
1  *{
2    margin: 0;
3    padding: 0;
4    box-sizing: border-box;
5    font-family: 'Poppins', sans-serif;
6  }
7  body{
8    background: #baeeff;
9  }
```

This portion of CSS sets the global styling. box-sizing: border-box ensures padding and border are included in an element's total width and height. font-family: 'Poppins', sans-serif; sets a global font.

```

10  .weathercard{
11      width: 90%;
12      max-width: 500px;
13      margin: 100px auto 0;
14      border-radius: 25px;
15      padding: 35px 35px;
16      text-align: center;
17      box-shadow: 0 0 20px rgba(0,0,0,0.1);
18      background: linear-gradient(135deg, #fff07d, #4069ff);
19      color: #fff;
20  }

```

This part styles the primary container of the application. It has a specific width and a max-width limit. It has a central alignment with auto margin. The card has a subtle shadow and a linear gradient background.

```

21  .search{
22      width: 100%;
23      display: flex;
24      align-items: center;
25      justify-content: space-between;
26  }
27  .search input{
28      border: 0;
29      outline: 0;
30      padding: 10px 25px ;
31      height: 60px;
32      border-radius: 30px;
33      flex: 1;
34      margin-right: 16px;
35      font-size: 23px;
36      background: #fff;
37      color: #353333;
38  }
39  .search button{
40      border: 0;
41      outline: 0;
42      border-radius: 50%;
43      width: 60px;
44      height: 60px;
45      cursor: pointer;
46      background: #ffffff;
47  }
48  .search button img{
49      width: 23px;
50  }
51  .search input:focus {
52      transition: box-shadow 0.3s ease-in-out;
53      box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
54  }
55
56  .search button:hover {
57      transform: scale(1.08);
58      transition: transform 0.3s ease-in-out;
59  }

```

The search bar consists of an input field and a button. This section has CSS flex properties to space out the input and button properly. The input field is styled to be round-edged

and without any border or outline. The button is also designed to be circular with a hover effect that slightly scales up the button.

```
94  < .weather{
95      display: none;
96  }
97  < .error{
98      text-align: left;
99      margin-left: 10px;
100     font-size: 25px;
101     font-weight: 500;
102     margin-top: 10px;
103     display: none;
104 }
```

The application has a weather display section and an error message section. Both sections are set to `display: none;` by default and are likely toggled to display based on JavaScript conditions in response to the user's search query.

4.3. Alternative tools/technologies

Django: Django can serve as an alternative to JavaScript (JS) for backend development because it is designed to help developers build complex web applications with ease and efficiency. Django employs the Python language, which is renowned for its simplicity and readability, making it a popular choice for beginners and experts alike. Furthermore, Django offers a comprehensive suite of tools and features out of the box, such as an ORM (Object-Relational Mapper) for database access, user authentication, URL routing, template engine, and more. This means you can build a fully-functional web application with less code compared to using Node.js (the backend version of JavaScript).[6] Here is an example for building a same weather app using Django[7]

PHP: PHP can serve as an alternative to JavaScript for server-side programming. PHP runs on the server and can interact with databases, handle HTTP requests and generate HTML content dynamically. Unlike JavaScript, which runs in the browser and can be viewed or altered by the user, PHP code is hidden from the user, providing an additional layer of security for sensitive operations. PHP's large user community, wealth of available libraries and frameworks, and strong support for various databases make it a versatile tool for creating dynamic web content. Similar to Django, a web application can be built entirely using PHP without relying on JavaScript for server-side logic.[8] Here is an example for building a same weather app using PHP [9]

4.4. Comparative Analysis

JavaScript shines when building highly interactive, dynamic, and responsive client-side applications. It is the preferred choice for manipulating HTML and CSS in real-time, delivering vibrant and user-friendly interfaces. With the advent of Node.js, JavaScript can also handle server-side tasks, making it an ideal choice for full-stack development. When paired with front-end frameworks like Angular or React, or used in server technologies like

MongoDB and Express.js, JavaScript stands out due to its versatility and comprehensive functionality. [6]

PHP:PHP is particularly advantageous when there's a need for complex server-side processing, such as managing user sessions or interacting extensively with databases. Its seamless integration with HTML makes it preferable for creating dynamic web content, particularly when working with solution stacks like LAMP or CMS platforms like WordPress and Joomla. If the project involves extensive back-end tasks and requires the convenient usage of an open-source language, PHP would be preferred. [6]

Django: Django, a Python-based framework, is most beneficial in scenarios where swift application development is crucial, especially when these applications necessitate sophisticated interactions with databases. Due to its readability and inclusive approach, Django is the go-to choice for building scalable and easily maintainable applications. If the project involves dealing with extensive relational databases, relies heavily on external libraries, or prioritizes data security, Django would be the superior choice.[10]

Bibliography

- [1] Daragh Ó Tuama, “Advantages of javascript,” 2023, see <https://codeinstitute.net/global/blog/advantages-of-javascript/>.
- [2] Sasha Bondar, “What are the advantages and disadvantages of using javascript,” 2023, see <https://reintech.io/blog/advantages-and-disadvantages-of-using-javascript>.
- [3] Ayesha Zahra, “Complete guide to javascript interactive map,” 2022, see <https://www.fusioncharts.com/blog/complete-guide-to-javascript-interactive-map/>.
- [4] Diego Salinas Gardón, “Webassembly vs. javascript: How do they compare,” 2022, see <https://snipcart.com/blog/webassembly-vs-javascript>.
- [5] Rory Toal, “What is javascript framework?” 2014, see <https://codeinstitute.net/global/blog/javascript-framework/>.
- [6] , “Django introduction,” 2023, see <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>.
- [7] Rohit Kumar Thakur, “Build a weather app using django and python,” 2021, see <https://python.plainenglish.io/weather-app-using-django-be98cfb33508>.
- [8] Daragh Ó Tuama, “Php vs javascript: When to use,” 2022, see <https://codeinstitute.net/global/blog/php-vs-javascript/>.
- [9] HMA WebDesign, “How to create weather app in php | how to use openweathermap api using php.”
- [10] Jeel Patel, “Django vs node.js: A comparative analysis,” 2021, see <https://www.monocubed.com/blog/django-vs-node-js/>.