Assignment 5B

Aim: Write a JavaScript program to:

a. Implement the concept of Promise(callback)
b. Fetch (Client Server communication)

Theory:

a. Promises-

In JavaScript, promises are a fundamental concept used to manage asynchronous operations. They provide a clean and structured way to handle asynchronous tasks and avoid the callback hell (also known as "pyramid of doom") that can occur when dealing with multiple nested callbacks.

Promises represent a value (or an error) that might be available now, in the future, or never. They have three states:

- Pending: The initial state of a promise. It represents an ongoing or incomplete operation.

- Fulfilled (Resolved): The state of a promise when the asynchronous operation is successfully completed. The promise transitions to this state with a result value.

- Rejected: The state of a promise when the asynchronous operation encounters an error or fails. The promise transitions to this state with a reason (error) value.

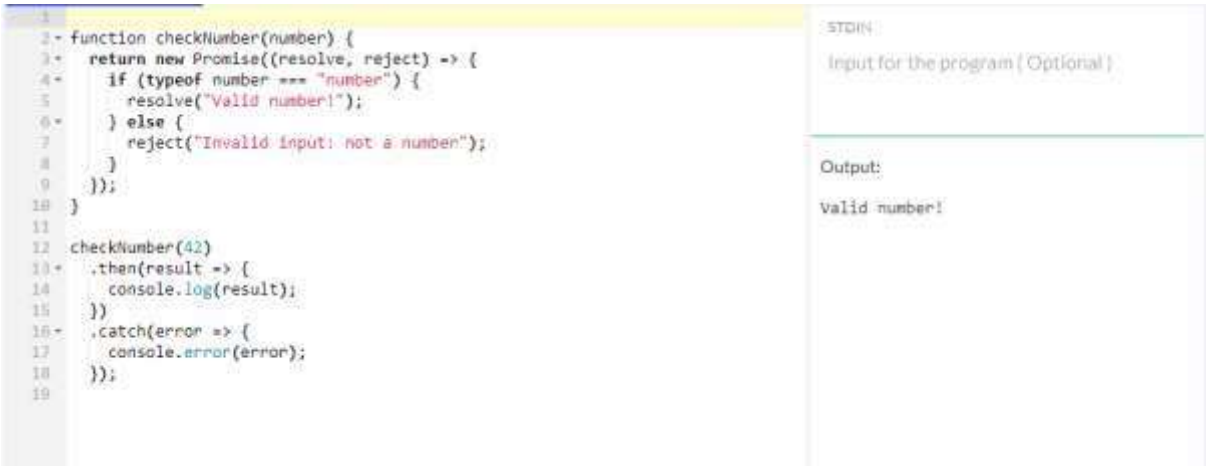
Promises provide a chainable syntax for handling asynchronous operations. The **.then()** method is used to handle fulfilment, and the **.catch()** method is used to handle rejection. Additionally, you can use the **.finally()** method to attach a callback that runs regardless of whether the promise is fulfilled or rejected.

b. Weather-

In JavaScript, the Weather API is a modern and built-in mechanism for making network requests to interact with servers or APIs. It provides a more flexible and powerful alternative to the older **XMLHttpRequest** object for sending and receiving data over HTTP.
The Weather API is based on promises, which allows for a cleaner and more readable syntax when dealing with asynchronous operations.

The **weather()** function is used to make requests. It takes a URL as its argument and returns a promise that resolves to the response of the request. Methods like **response.json()** or **response.text()** to extract the actual data from the response.

Code snapshots:



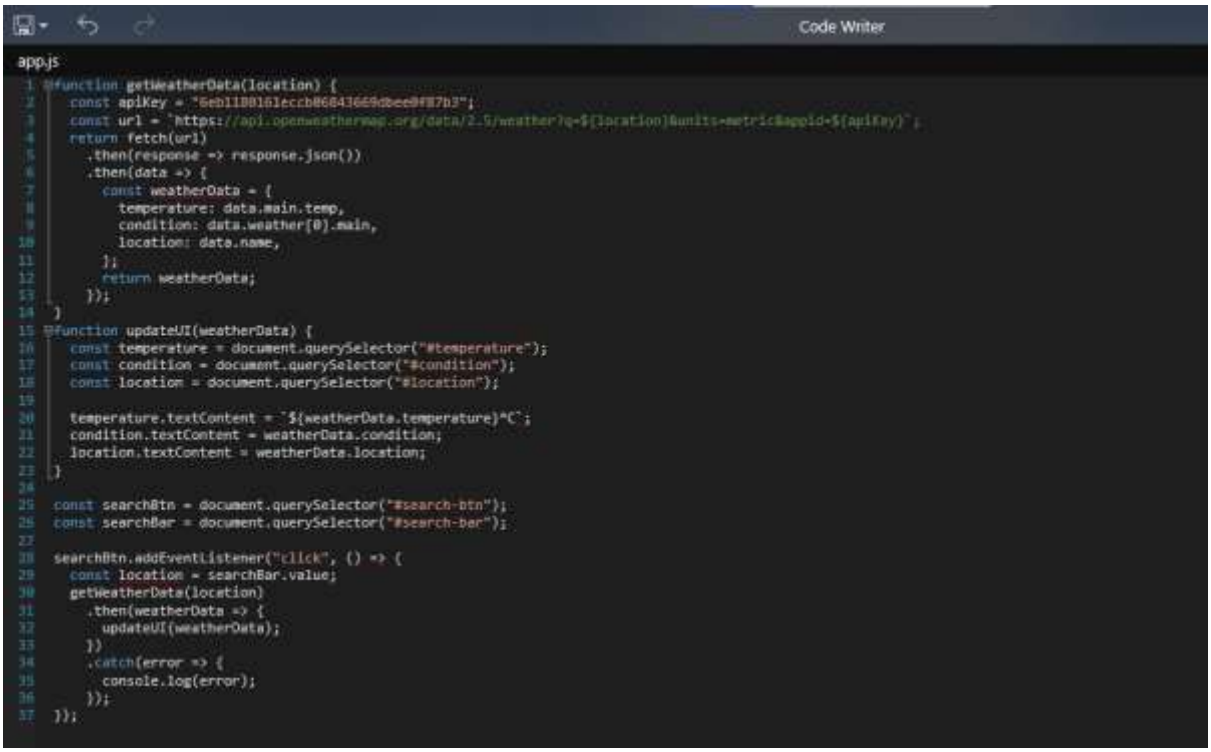
```
function checkNumber(number) {
  return new Promise((resolve, reject) => {
    if (typeof number === "number") {
      resolve("Valid number!");
    } else {
      reject("Invalid input: not a number");
    }
  });
}

checkNumber(42)
  .then(result => {
    console.log(result);
  })
  .catch(error => {
    console.error(error);
  });
```

STDIN

Input for the program (Optional)

Output:

Valid number!

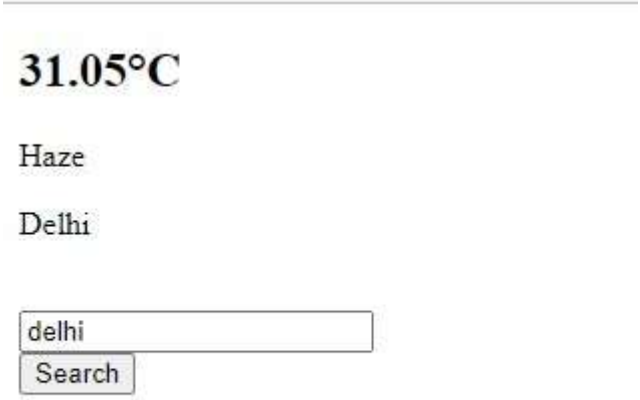


Code Writer

app.js

```
function getWeatherData(location) {
  const apiKey = "6eb11801d1eccb0684366d0bee0f87b3";
  const url = `https://api.openweathermap.org/data/2.5/weather?q=${location}&units=metric&appid=${apiKey}`;
  return fetch(url)
    .then(response => response.json())
    .then(data => {
      const weatherData = {
        temperature: data.main.temp,
        condition: data.weather[0].main,
        location: data.name,
      };
      return weatherData;
    });
}
function updateUI(weatherData) {
  const temperature = document.querySelector("#temperature");
  const condition = document.querySelector("#condition");
  const location = document.querySelector("#location");

  temperature.textContent = `${weatherData.temperature}°C`;
  condition.textContent = weatherData.condition;
  location.textContent = weatherData.location;
}

const searchBtn = document.querySelector("#search-btn");
const searchBar = document.querySelector("#search-bar");

searchBtn.addEventListener("click", () => {
  const location = searchBar.value;
  getWeatherData(location)
    .then(weatherData => {
      updateUI(weatherData);
    })
    .catch(error => {
      console.log(error);
    });
});
```

Out put :



Conclusion:

In this assignment, we've successfully implemented the concept of Promises to handle asynchronous operations. We also observed reject and resolve state of Promise.

We also showcased the Weather API for client-server communication. Using the **weather** function, we made a request to a sample API, retrieved JSON data from the response, and handled it using promises. This demonstrates how modern web applications can interact with servers to retrieve and process data asynchronously.