Roll no :- 48
Name :– Manav Jain

ASSIGNMENT – 5(b)

AIM:- Write a Javascript program.

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

LO MAPPED:- LO4

THEORY:-

Promises are important building blocks for asynchronous operations in JavaScript. A Promise is a special JavaScript object.It produces a value after an asynchronous operation completes successfully, or an error if it does not complete successfully due to time out, network error, and so on.Successful call completions are indicated by the resolve function call, and errors are indicated by the reject function call.They help avoid the so-called "callback hell" and make asynchronous code easier to write and understand.

A Promise represents a value that might not be available yet, but will be resolved in the future. It has three possible states:

- Pending: The initial state of a promise when it's created.
- Fulfilled (Resolved): The state when the promise has successfully completed its operation and resolved to a value.
- Rejected: The state when the promise encountered an error or failed to complete its operation.

create a promise using the promise constructor: The constructor function takes a function as an argument. This function is called the executor function.
let promise = new Promise(function(resolve, reject) { // Make an asynchronous call and either resolve or reject });

The executor function takes two arguments, resolve and reject.These are the callbacks provided by the JavaScript language.For the promise to be effective, the executor function should call either of the callback functions, resolve or reject. The new Promise() constructor returns a promise object.As the executor function needs to handle async operations, the returned promise object should be capable of informing when the execution has been started, completed (resolved) or returned with error (rejected).

result – This property can have the following values:

- undefined: Initially when the state value is pending.
- value: When resolve(value) is called.
- error: When reject(error) is called

These internal properties are code-inaccessible but they are inspectable.This means that we will be able to inspect the state and result property values using the debugger tool, but we will not be able to access them directly using the program

Using .then() and .catch():

Once the promise is created, you can chain .then() and .catch() methods to handle the resolved and rejected outcomes respectively.

```
myPromise
  .then((resolvedValue) => {
    // Handle successful result
  })
  .catch((error) => {
    // Handle error
  });
```

The .then() method takes a callback function that is executed when the promise is resolved. The .catch() method handles errors that might occur during the promise's execution.

The .finally() handler performs cleanups like stopping a loader, closing a live connection, and so on.The finally() method will be called irrespective of whether a promise resolves or rejects. It passes through the result or error to the next handler which can call a .then() or .catch() again

OUTPUT WITH CODE:-

Index.html:

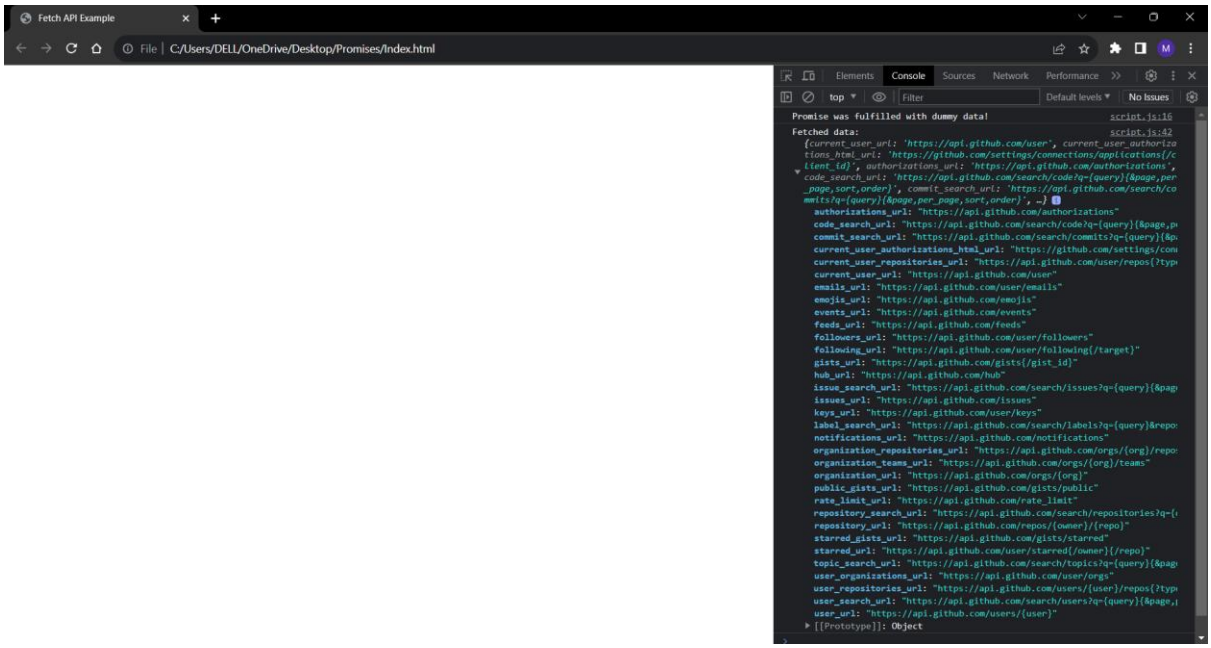
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Fetch API Example</title>
</head>
<body>
   <script src="script.js"></script>
</body>
</html>
```

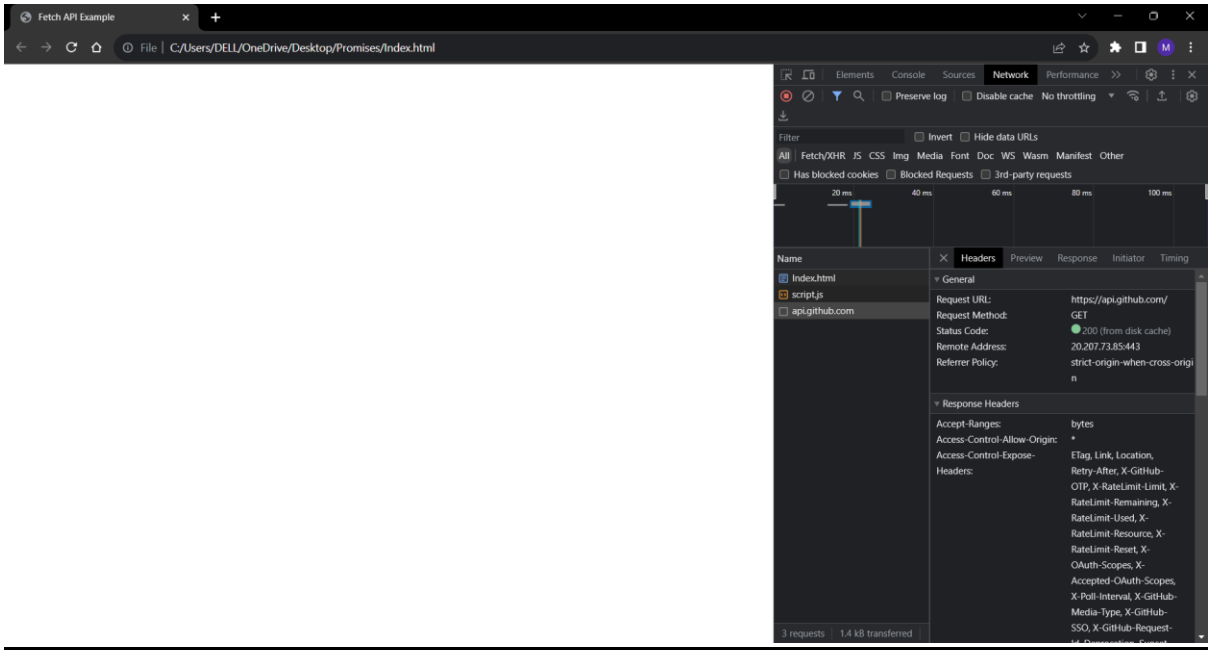
script.js:

```
var dummyPromise='https://some-api.com/posts';
var dummyPromise = new Promise((resolve, reject) => {
   const success = true; // or false
   if (success) {
     resolve("Promise was fulfilled with dummy data!");
   } else {
     reject("Promise was rejected with dummy error!");
   }
 });

 dummyPromise.then(result => {
   console.log(result);
 }).catch(error => {
   console.error(error);
 });

// Using Promises
function fetchData() {
   return new Promise((resolve, reject) => {
     fetch('https://api.github.com/')
       .then(response => {
```

```
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      resolve(data);
    })
    .catch(error => {
      reject(error);
    });
  });
}
// Calling the fetchData function
fetchData()
  .then(data => {
    console.log('Fetched data:', data);
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
```

Output:





CONCULSION: In JavaScript, Promises are essential for managing asynchronous tasks, providing a structured way to handle success and errors. They enhance code readability, support chaining, and integrate well with async/await, making asynchronous programming more organized and dependable.