

## **LAB ASSIGNMENT NO – 5b**

**Aim:** Write a Javascript program to implement the concept of Promise. Access the data from any API using Promise and either resolve or reject the request by taking appropriate action.

### **Theory:**

Promises in JavaScript are a powerful and essential concept for handling asynchronous operations, such as fetching data from a server, reading files, or executing time-consuming tasks without blocking the main thread.

Promises provide a more structured and readable way to manage asynchronous code compared to traditional callback functions.

Here are the key aspects of promises in JavaScript:

#### 1. States:

Promises have three states: `pending`, `fulfilled`, and `rejected`.

A promise starts in the `pending` state, indicating that the asynchronous operation is ongoing.

It transitions to the `fulfilled` state when the operation is successful, returning a result.

If an error occurs during the operation, the promise transitions to the `rejected` state, providing an error reason.

#### 2. Creating Promises:

- You can create a promise using the `Promise` constructor, passing in an executor function with `resolve` and `reject` functions.

```
const myPromise = new Promise((resolve, reject) => {  
  // Asynchronous operation  
  if (operationSuccessful) {  
    resolve(result);  
  } else {  
    reject(error);  
  }  
});
```

### 3. Chaining Promises:

- Promises allow you to chain asynchronous operations using `then` and `catch` methods.

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

### 4. Handling Multiple Promises:

- `Promise.all` waits for all promises to resolve and returns an array of their results.
- `Promise.race` resolves or rejects as soon as one promise in an array resolves or rejects.

### 5. Async/Await:

- The `async` and `await` keywords provide a more synchronous-looking syntax for working with promises.

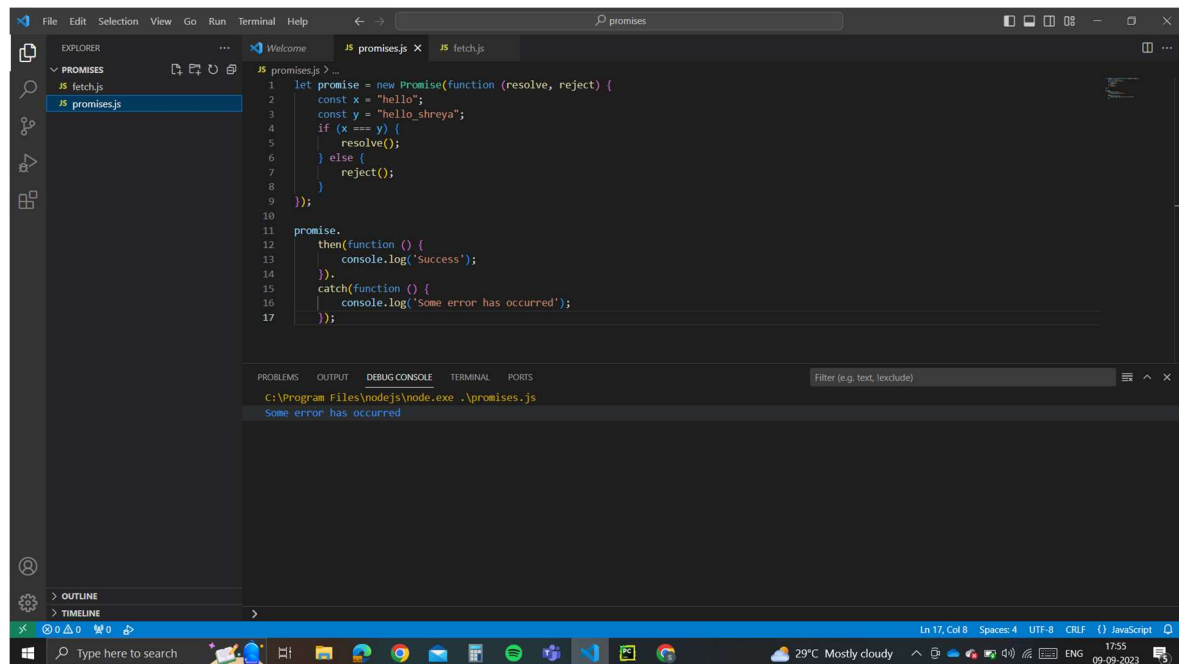
```
async function fetchData() {
  try {
    const result = await fetchSomeData();
    // Handle the result
  } catch (error) {
    // Handle errors
  }
}
```

Promises help in writing clean, maintainable, and error-handling-friendly asynchronous code in JavaScript. They are a fundamental concept for working with modern web APIs, handling network requests, and managing asynchronous tasks in both browser and Node.js environments.

**Conclusion:** This experiment on promises illuminated the power of asynchronous programming in JavaScript. Promises provided a structured and efficient way to manage asynchronous tasks, allowing for clear and

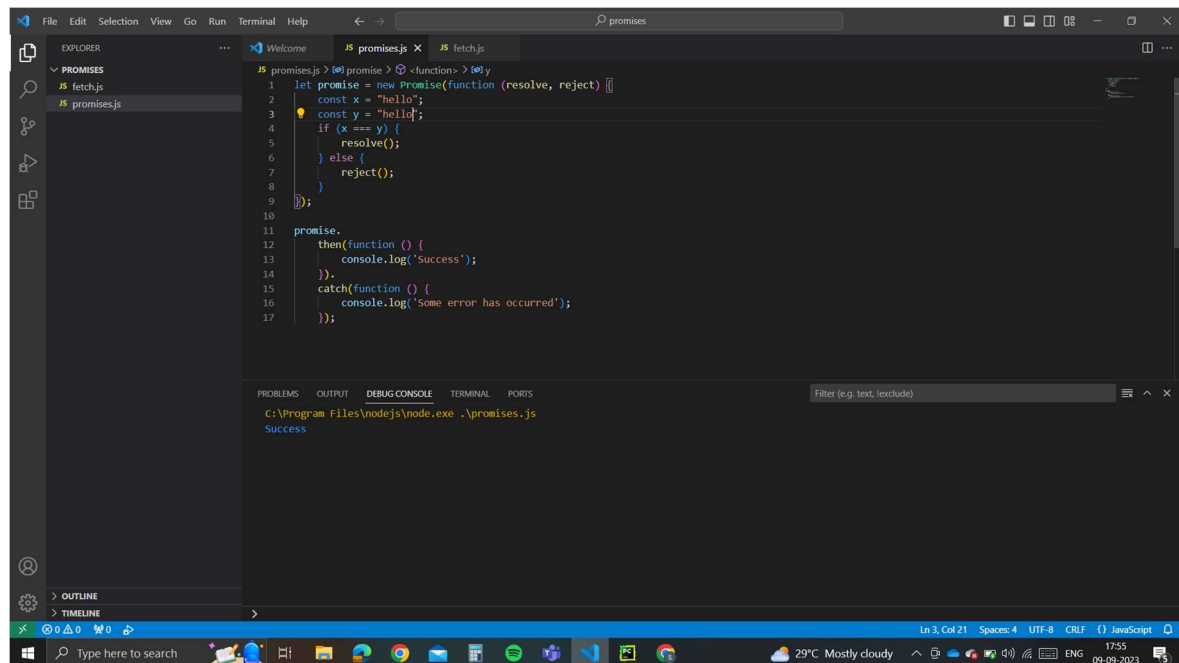
readable code. By understanding promise states, chaining, and error handling, we gained valuable insights into handling asynchronous operations gracefully.

## Code:



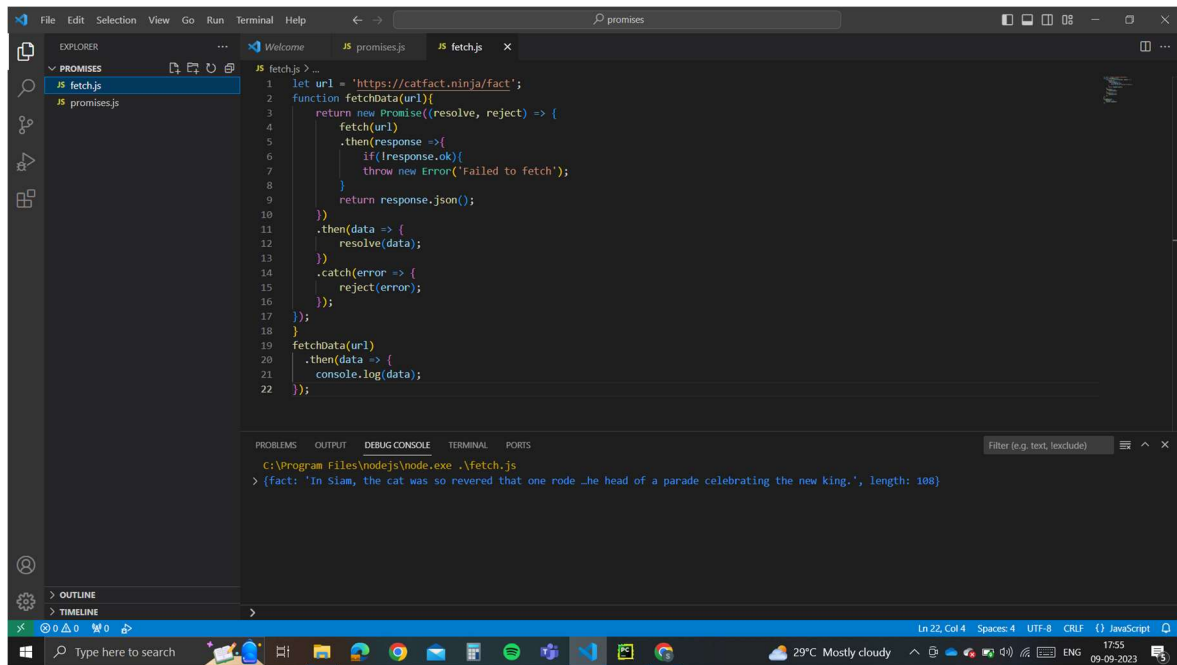
```
1 let promise = new Promise(function (resolve, reject) {
2   const x = "hello";
3   const y = "hello_shreya";
4   if (x === y) {
5     resolve();
6   } else {
7     reject();
8   }
9 }
10);
11
12 promise.
13   then(function () {
14     console.log('Success');
15   });
16   catch(function () {
17     console.log('Some error has occurred');
18   });
```

Terminal Output: C:\Program Files\nodejs\node.exe .\promises.js  
Some error has occurred



```
1 let promise = new Promise(function (resolve, reject) {}
2   const x = "hello";
3   const y = "hello";
4   if (x === y) {
5     resolve();
6   } else {
7     reject();
8   }
9 }
10);
11
12 promise.
13   then(function () {
14     console.log('Success');
15   });
16   catch(function () {
17     console.log('Some error has occurred');
18   });
```

Terminal Output: C:\Program Files\nodejs\node.exe .\promises.js  
Success



```
1 let url = 'https://catfact.ninja/fact';
2 function fetchData(url){
3   return new Promise((resolve, reject) => {
4     fetch(url)
5       .then(response => {
6         if(!response.ok){
7           throw new Error('Failed to fetch');
8         }
9         return response.json();
10      })
11      .then(data => {
12        resolve(data);
13      })
14      .catch(error => {
15        reject(error);
16      });
17  });
18  fetchData(url)
19    .then(data => {
20      console.log(data);
21    });
22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

C:\Program Files\nodejs\node.exe .\fetch.js

> {fact: 'In Siam, the cat was so revered that one rode \_he head of a parade celebrating the new king.', length: 108}

**Lab Outcome :** LO4- Use Javascript to develop interactive web pages.