Alphonz George

T11-15

Assignment 9

**AIM:**

To understand and implement **Asynchronous Programming** concepts in JavaScript using

**Promises**, allowing for efficient handling of operations that may complete at a later time.

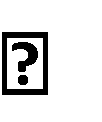
**LABOUTCOME:**

Understand the concept of **asynchronous programming** and how it improves the performance of applications.

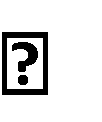
Learn how **Promises** work to manage asynchronous operations in JavaScript.

Write and execute code that uses Promises to handle asynchronous tasks such as API calls or file reading/writing.

**THEORY:**

**Asynchronous Programming:**

* **Asynchronous programming allows code to run without blocking the execution of other operations, enabling tasks like network requests, file handling, or timers to complete in the background.**

**Promises:**

* **A Promise is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value.**
* **A Promise has three states:**
  + **Pending: The initial state; neither fulfilled nor rejected.**
  + **Fulfilled: The operation completed successfully.**
  + **Rejected: The operation failed.**
* **Creating a Promise:**

**let promise = new Promise((resolve, reject) => {**

**// asynchronous task here**

**if (/\* successful \*/) { resolve(result);**

**} else {**

**reject(error);**

**}**

**});**

**PROGRAM:**

The following program simulates a simple asynchronous operation using Promises. It demonstrates a Promise that resolves if a condition is met or rejects if it fails.

// Function to simulate an asynchronous task function checkOrder(orderReady) {

return new Promise((resolve, reject) => { console.log("Processing your order...");

setTimeout(() => { if (orderReady) {

resolve("Your order is ready!");

} else {

reject("There was an issue with your order.");

});

}

}

}, 2000); // simulates a delay of 2 seconds

// Calling the function and handling the promise checkOrder(true) // Change to false to test rejection

.then((message) => {

console.log("Success: " + message);

})

.catch((error) => {

console.log("Error: " + error);

});

**OUTPUT:**

When checkOrder(true) is called, the program waits for 2 seconds and then logs: Processing your order...

Success: Your order is ready!

When checkOrder(false) is called, the program waits for 2 seconds and then logs: Processing your order...

Error: There was an issue with your order.

**CONCLUSION:**

This example demonstrates how **Promises** provide a powerful way to handle asynchronous operations in JavaScript. By leveraging .then() and .catch(), we can define logic to be executed after a Promise is resolved or rejected. Promises help to avoid **callback hell**, making code more readable and easier to maintain for complex asynchronous workflows.