

Lesson 05 Demo 08

Working with Asynchronous JavaScript

Objective: To demonstrate the implementation of asynchronous JavaScript using Promises, `async/await`, for improving responsiveness and reliability in web applications

Tools required: Visual Studio Code and Node.js

Prerequisites: None

Steps to be followed:

1. Create and set up the project
2. Develop the webpage structure
3. Implement JavaScript for asynchronous operations
4. Execute and verify the project

Step 1: Create and set up the project

1.1 Create a new project folder and navigate into it using the following command:

```
mkdir async-demo  
cd async-demo
```

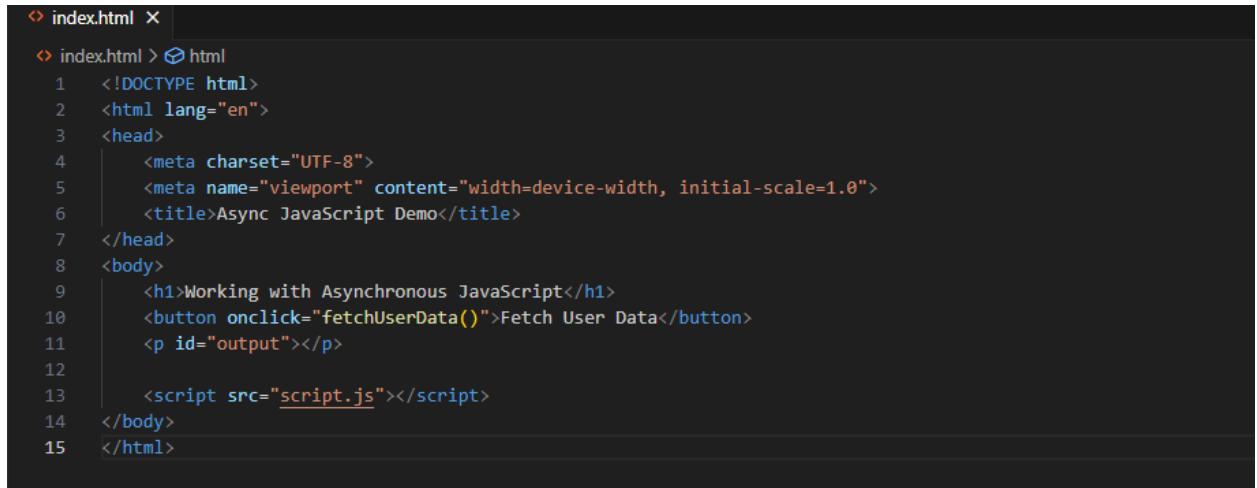
```
● labuser@ip-172-31-32-65:~$ mkdir async-demo  
● labuser@ip-172-31-32-65:~$ cd async-demo  
○ labuser@ip-172-31-32-65:~/async-demo$ █
```

Step 2: Develop the webpage structure

2.1 Create an HTML file named **index.html** and add the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Async JavaScript Demo</title>
</head>
<body>
  <h1>Working with Asynchronous JavaScript</h1>
  <button onclick="fetchUserData()">Fetch User Data</button>
  <p id="output"></p>

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

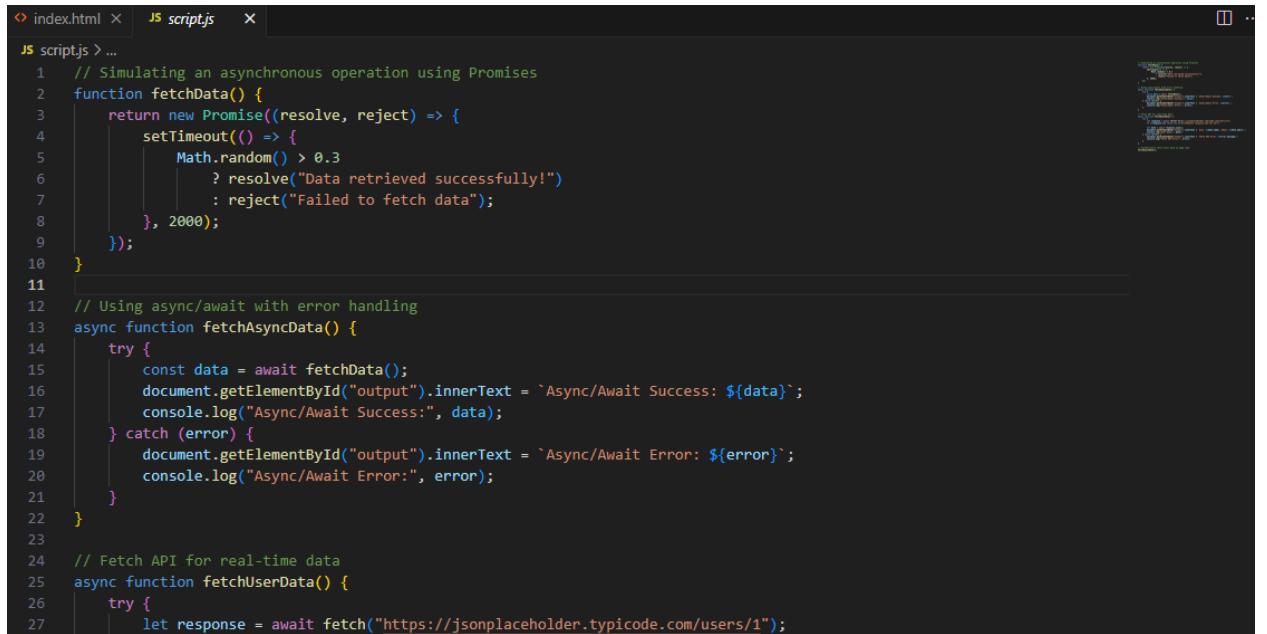


The screenshot shows a code editor window with the file 'index.html' open. The code is displayed with line numbers from 1 to 15 on the left. The code itself is identical to the one provided above, including the DOCTYPE declaration, head and body sections, and the script tag.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Async JavaScript Demo</title>
7  </head>
8  <body>
9    <h1>Working with Asynchronous JavaScript</h1>
10   <button onclick="fetchUserData()">Fetch User Data</button>
11   <p id="output"></p>
12
13   <script src="script.js"></script>
14 </body>
15 </html>
```

Step 3: Implement JavaScript for asynchronous operations

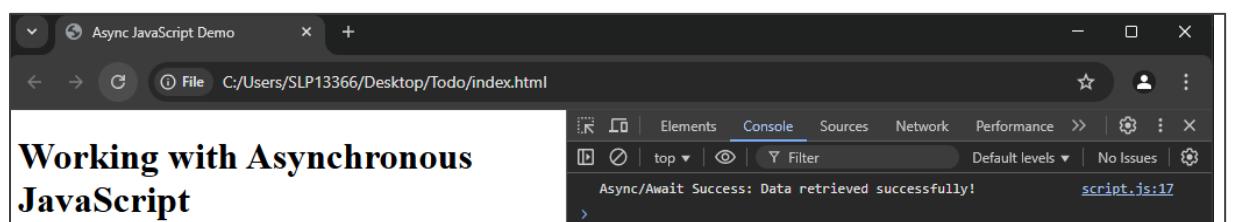
3.1 Create a JavaScript file named **script.js** and add the following code:



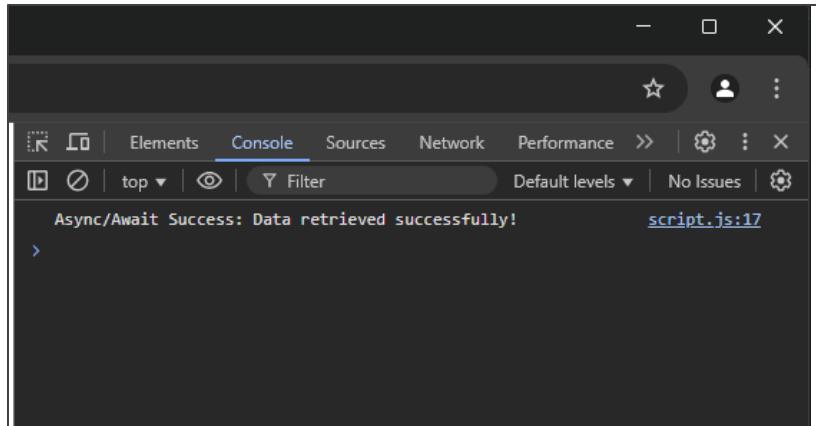
```
// index.html <-- JS script.js <-- ...
JS script.js > ...
1 // Simulating an asynchronous operation using Promises
2 function fetchData() {
3     return new Promise((resolve, reject) => {
4         setTimeout(() => {
5             if (Math.random() > 0.3
6                 ? resolve("Data retrieved successfully!")
7                 : reject("Failed to fetch data");
8         }, 2000);
9     });
10 }
11 // Using async/await with error handling
12 async function fetchAsyncData() {
13     try {
14         const data = await fetchData();
15         document.getElementById("output").innerText = `Async/Await Success: ${data}`;
16         console.log("Async/Await Success:", data);
17     } catch (error) {
18         document.getElementById("output").innerText = `Async/Await Error: ${error}`;
19         console.log("Async/Await Error:", error);
20     }
21 }
22
23 // Fetch API for real-time data
24 async function fetchUserData() {
25     try {
26         let response = await fetch("https://jsonplaceholder.typicode.com/users/1");
27     }
28 }
```

Step 4: Execute and verify the project

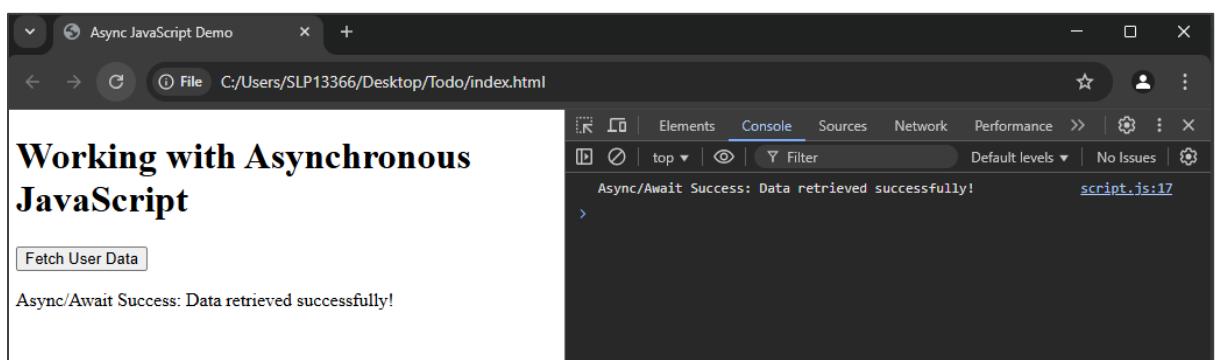
4.1 Open **index.html** in a web browser



4.2 Check the console (F12 > Console tab) for logs



4.3 Click on the **Fetch User Data** button to fetch data from the API



By following the above steps, you have successfully implemented asynchronous JavaScript using Promises, `async/await`, and the Fetch API in separate HTML and JavaScript files, ensuring better code organization and real-time data retrieval with error handling.