

# Hands-on Lab - Async Callback (30 mins)



## Objective for Exercise:

After completing this lab, you will be able to:

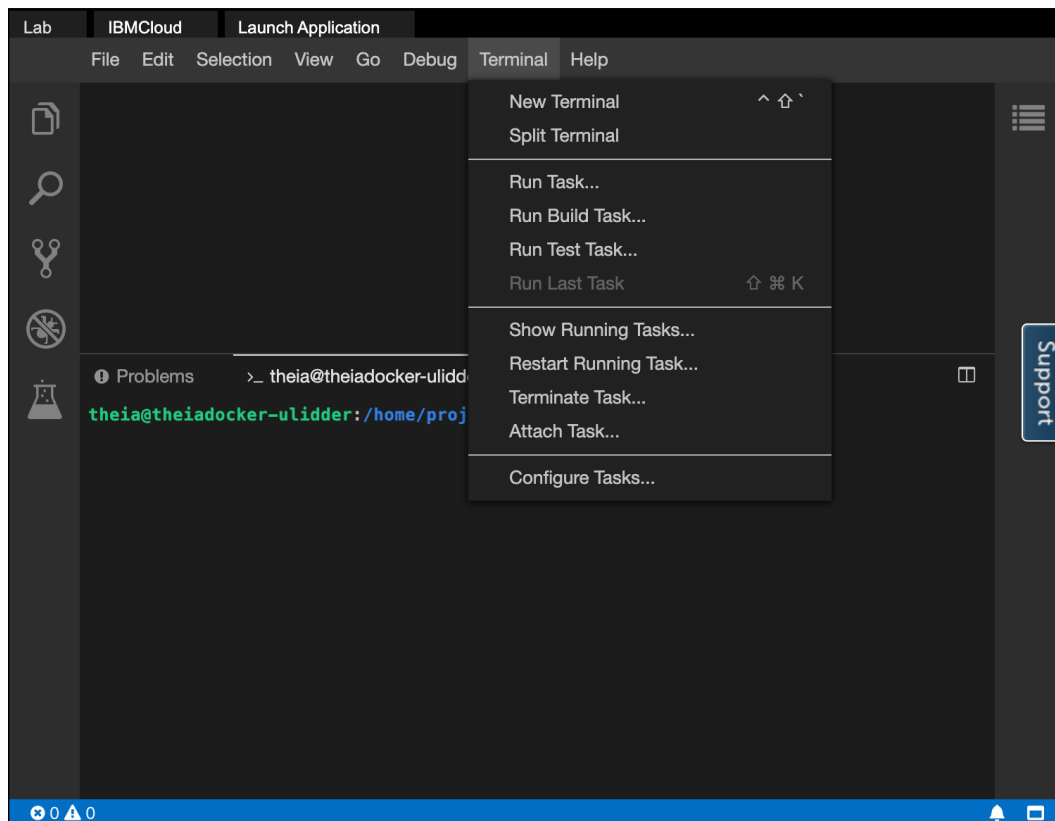
- Describe asynchronous callbacks
- Create a Node.js application

## Prerequisites

- Basic knowledge of JavaScript

## Task 1 : Set-up - Check for the cloned files

1. Open a terminal window by using the menu in the editor: Terminal > New Terminal.



2. Change to your project folder.

1. 1

1. `cd /home/project`

Copied!

3. Check if you have the folder **lkpho-Cloud-applications-with-Node.js-and-React**

1. 1

1. `ls /home/project`

Copied!

If you do, you can skip to step 5.

4. Clone the git repository that contains the artifacts needed for this lab, if it doesn't already exist.

1. 1

```
1. git clone https://github.com/ibm-developer-skills-network/lkpho-Cloud-applications-with-Node.js-and-React.git
```

Copied!

5. Change to the directory for this lab.

```
1. 1
```

```
1. cd lkpho-Cloud-applications-with-Node.js-and-React/CD220Labs/async_callback/
```

Copied!

6. List the contents of this directory to see the artifacts for this lab.

```
1. 1
```

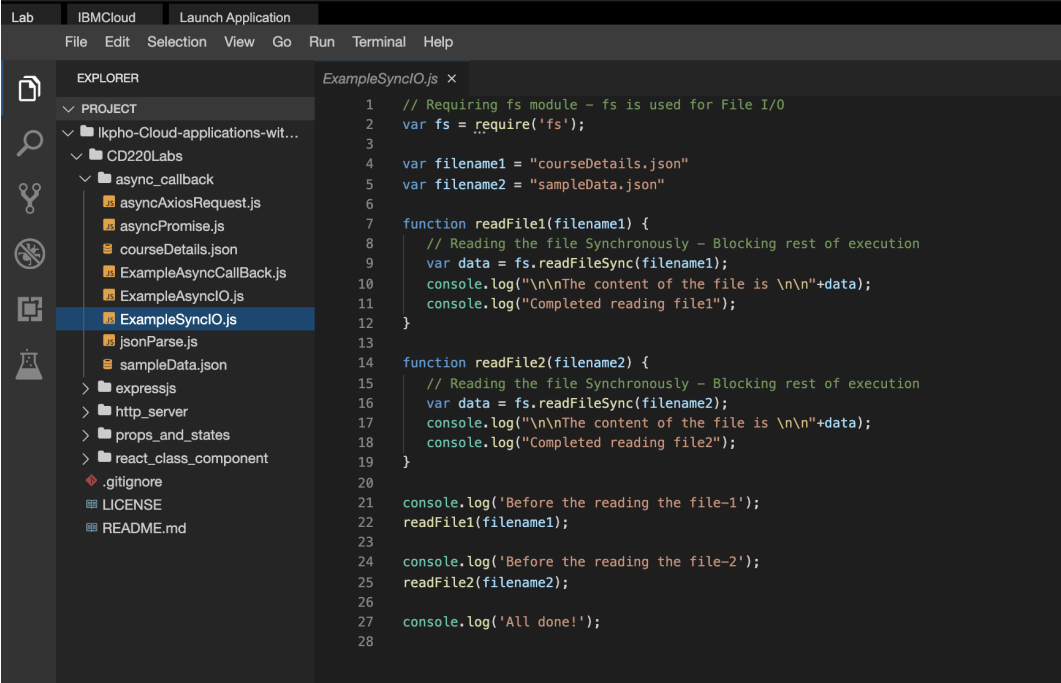
```
1. ls
```

Copied!

You must have a few exercise files that you will be running in the exercises.

## Task 2: Synchronous I/O

In the files explorer view ExampleSyncIO.js. It would appear like this.



The screenshot shows a code editor with a dark theme. On the left is the 'EXPLORER' sidebar showing a file tree. The tree structure is: PROJECT > lkpho-Cloud-applications-wit... > CD220Labs > async\_callback > ExampleSyncIO.js (selected). Other files in the async\_callback folder include asyncAxiosRequest.js, asyncPromise.js, courseDetails.json, ExampleAsyncCallBack.js, ExampleAsyncIO.js, jsonParse.js, and sampleData.json. Below this are folders for expressjs, http\_server, props\_and\_states, and react\_class\_component, along with .gitignore, LICENSE, and README.md. The main editor area shows the code for ExampleSyncIO.js. The code uses the fs module to read two files synchronously: 'courseDetails.json' and 'sampleData.json'. It includes console logs to show the content of each file and a final log saying 'All done!'. The code is as follows:

```
1 // Requiring fs module - fs is used for File I/O
2 var fs = require('fs');
3
4 var filename1 = "courseDetails.json"
5 var filename2 = "sampleData.json"
6
7 function readFile1(filename1) {
8   // Reading the file Synchronously - Blocking rest of execution
9   var data = fs.readFileSync(filename1);
10  console.log("\n\nThe content of the file is \n\n"+data);
11  console.log("Completed reading file1");
12 }
13
14 function readFile2(filename2) {
15   // Reading the file Synchronously - Blocking rest of execution
16   var data = fs.readFileSync(filename2);
17   console.log("\n\nThe content of the file is \n\n"+data);
18   console.log("Completed reading file2");
19 }
20
21 console.log('Before the reading the file-1');
22 readFile1(filename1);
23
24 console.log('Before the reading the file-2');
25 readFile2(filename2);
26
27 console.log('All done!');
28
```

► You can also click here to view the code

2. In the terminal window run the server with the following command.

```
1. 1
```

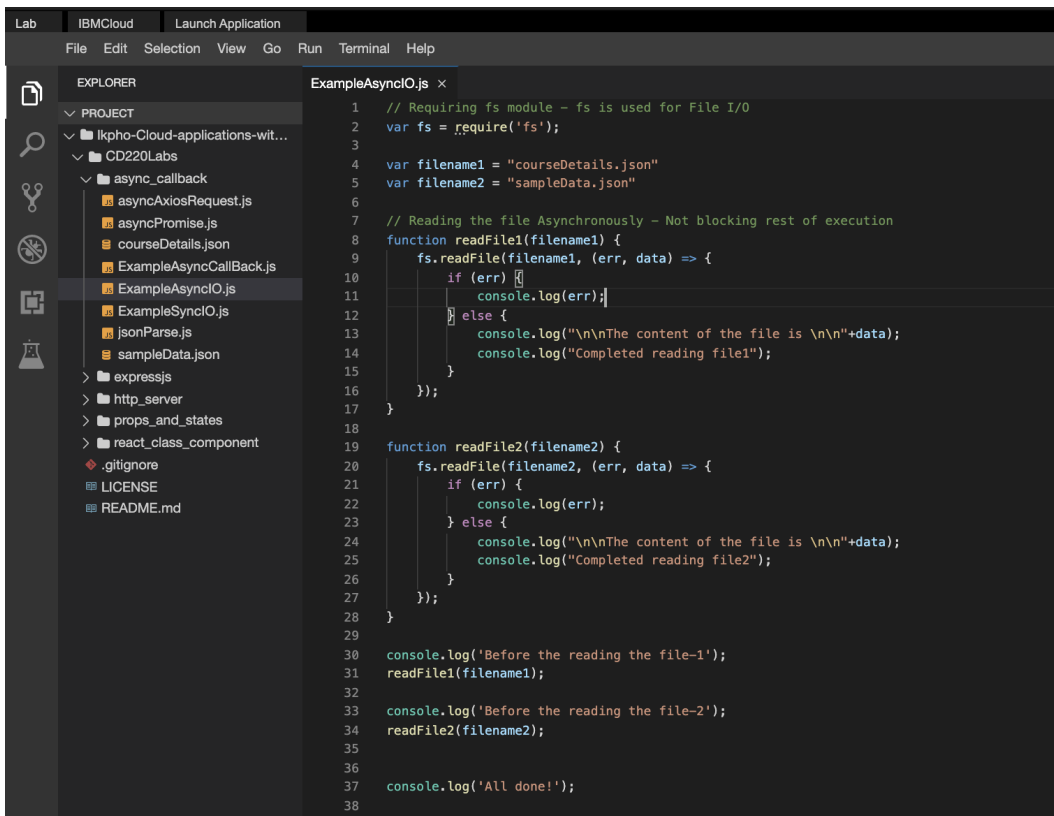
```
1. node ExampleSyncIO.js
```

Copied!

Observe that the two files are being read synchronously, one after the other. This will be evident from the order in which the console log appears.

## Task 3: Asynchronous IO

1. In the files explorer view ExampleAsyncIO.js. It would appear like this.



```
1 // Requiring fs module - fs is used for File I/O
2 var fs = require('fs');
3
4 var filename1 = "courseDetails.json"
5 var filename2 = "sampleData.json"
6
7 // Reading the file Asynchronously - Not blocking rest of execution
8 function readFile1(filename1) {
9     fs.readFile(filename1, (err, data) => {
10         if (err) {
11             console.log(err);
12         } else {
13             console.log("\n\nThe content of the file is \n\n"+data);
14             console.log("Completed reading file1");
15         }
16     });
17 }
18
19 function readFile2(filename2) {
20     fs.readFile(filename2, (err, data) => {
21         if (err) {
22             console.log(err);
23         } else {
24             console.log("\n\nThe content of the file is \n\n"+data);
25             console.log("Completed reading file2");
26         }
27     });
28 }
29
30 console.log('Before the reading the file-1');
31 readFile1(filename1);
32
33 console.log('Before the reading the file-2');
34 readFile2(filename2);
35
36 console.log('All done!');
```

► You can also click here to view the code

2. In the terminal window run the server with the following command.

1. 1

1. node ExampleAsyncIO.js

Copied!

Observe that the two files are being read asynchronously. This will be evident from the order in which the console log appears. The following three console log appears before the file content is printed though the logs are called in the code in different order.

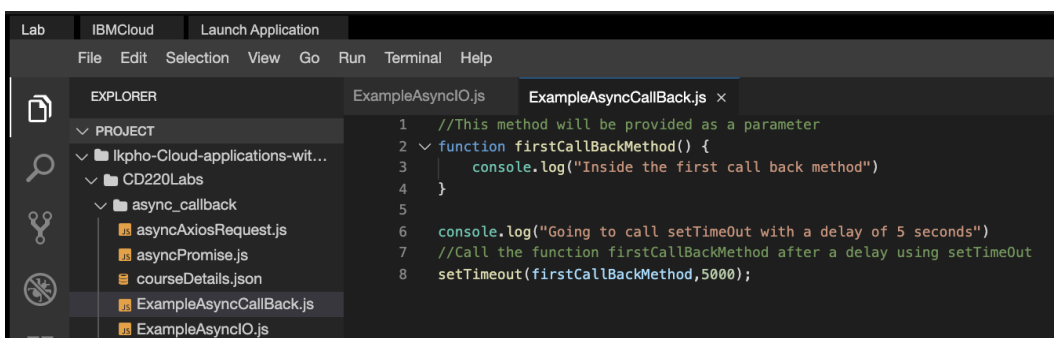
1. 1  
2. 2  
3. 3

1. Before the reading the file-1  
2. Before the reading the file-2  
3. All done!

Copied!

## Task 4: Creating Callback Functions

1. In the files explorer view ExampleAsyncCallBack.js. It would appear like this.



```
1 //This method will be provided as a parameter
2 function firstCallBackMethod() {
3     console.log("Inside the first call back method")
4 }
5
6 console.log("Going to call setTimeout with a delay of 5 seconds")
7 //Call the function firstCallBackMethod after a delay using setTimeout
8 setTimeout(firstCallBackMethod,5000);
```

► You can also click here to view the code

2. In the terminal window run the server with the following command.

1. 1

1. node ExampleAsyncCallBack.js

Copied!

setTimeout is a built-in library method which allows you to pass a method which needs to be called on timeout, as a parameter. Here firstCallbackMethod is defined and then passed as a parameter to setTimeout. As you may have observed, the method will be called after 5 seconds. This is called `callback`

## Task 5: Promises

1. In the files explorer view `asyncPromise.js`. It would appear like this.

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a folder named `async_callback` containing `asyncAxiosRequest.js`, `asyncPromise.js`, `courseDetails.json`, `ExampleAsyncCallback.js`, `ExampleAsyncIO.js`, `ExampleSyncIO.js`, `jsonParse.js`, and `sampleData.json`. The code editor shows the content of `asyncPromise.js`:

```
1 var prompt = require('prompt-sync')();
2 var fs = require('fs');
3
4 const methCall = new Promise((resolve, reject) => {
5   var filename = prompt('What is the name of the file?');
6   try {
7     const data = fs.readFileSync(filename, {encoding: 'utf8', flag: 'r'});
8     resolve(data);
9   } catch (err) {
10    reject(err);
11  }
12 });
13
14 console.log(methCall);
15
16 methCall.then(
17   (data) => console.log(data),
18   (err) => console.log("Error reading file")
19 );
20
```

► You can also click here to view the code

2. In the terminal window run the following command to install `prompt-sync`. Using `--save` ensures that the `package.json` file is updated for dependencies.

1. 1

1. `npm install --save prompt-sync`

Copied!

3. In the terminal window run the server with the following command. It will ask you for a filename. Enter a valid filename from the current directory.

1. 1

1. `node asyncPromise.js`

Copied!

`methCall` here is a promise object. When the promise is full-filled, the console log will be printed. Run the above command again and try to pass an invalid filename. See the console log printed out as the promise is being rejected.

## Task 6: AsyncAxiosRequest

1. In the files explorer view `asyncAxiosRequest.js`. It would appear like this.

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a folder named `async_callback` containing `asyncAxiosRequest.js`, `asyncPromise.js`, `courseDetails.json`, `ExampleAsyncCallback.js`, `ExampleAsyncIO.js`, `ExampleSyncIO.js`, `jsonParse.js`, and `sampleData.json`. The code editor shows the content of `asyncAxiosRequest.js`:

```
1 const axios = require('axios').default;
2
3
4 const connectToURL = (url) => {
5   const req = axios.get(url);
6   console.log(req);
7   req.then(resp => {
8     console.log("Fulfilled")
9     console.log(resp.data);
10  })
11  .catch(err => {
12    console.log("Rejected for url "+url)
13    console.log(err.toString());
14  });
15 }
16
17 //Valid URL
18 connectToURL('https://raw.githubusercontent.com/ibm-developer-skills-network/lkpho-Cloud-app
19 //Invalid URL
20 connectToURL('https://raw.githubusercontent.com/ibm-developer-skills-network/lkpho-Cloud-app
```

► You can also click here to view the code

2. To run this code, we need to install axios package. Run the following command to install axios.

1. 1

1. npm install --save axios

Copied!

3. In the terminal window run the code with the following command.

1. 1

1. node asyncAxiosRequest.js

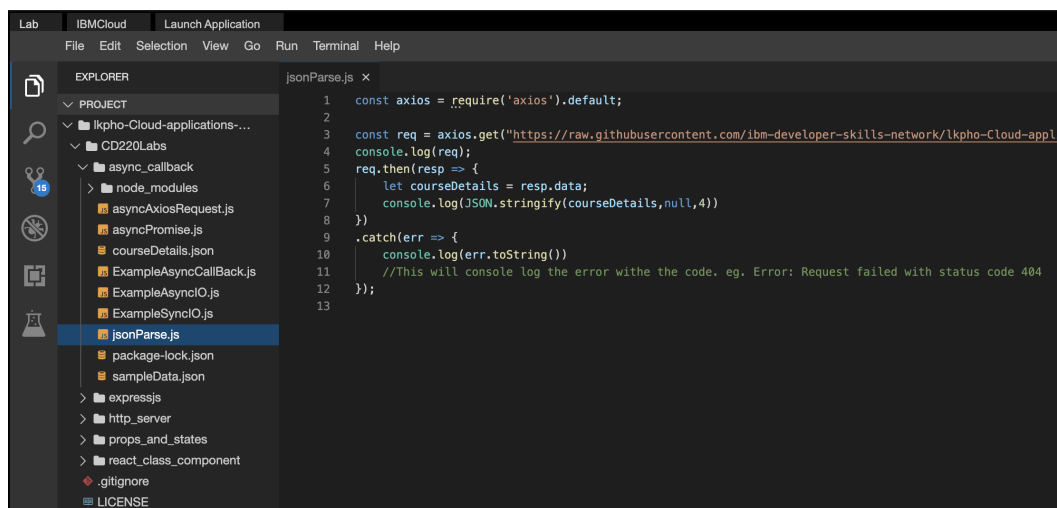
Copied!

When you run the code, the first connectToURL is an axios request to a valid URL which will return JSON object. The second connectToURL is an axiosRequest to an invalid URL. This will return appropriate error message. The output will be as below.

```
Promise { <pending> }
Promise { <pending> }
Rejected for url https://raw.githubusercontent.com/ibm-developer-skills-network/lkpho-Cloud-applications-with-No
de.js-and-React/master/CD220Labs/async_callback/sampleDate.json
Error: Request failed with status code 404
Fulfilled
{ name: 'Jason', age: '25', department: 'IT' }
```

## Task 7: Working with JSON

1. In the files explorer view jsonParse.js. It would appear like this.



► You can also click here to view the code

2. To run this code, we need to install axios package. You will be having the axios module that you installed in the previous exercise. If not, run the following command to install axios.

1. 1

1. npm install --save axios

Copied!

3. In the terminal window run the code with the following command.

1. 1

1. node jsonParse.js

Copied!

When you run the code, an axios request is made to a remote URL which returns a JSON object. This JSON object is stringified(or formatted in a readable form) and logged on the console. The output will be as below.

```
Promise { <pending> }
{
  "course": {
    "name": "Cloud Application Development",
    "modules": {
      "module1": {
        "name": "Introduction to Server-Side JavaScript",
        "topics": [
          "Course Intro",
          "Module Introduction",
          "Introduction to Node.js",
          "Introduction to Server-Side Javascript",
          "Creating a Web Server with Node.js",
          "Working with Node.js Modules",
          "Lab - Introduction to Server-side JavaScript",
          "Module Summary",
          "Practice Quiz",
          "Graded Quiz"
        ]
      },
      "module2": {
        "name": "Asynchronous IO with Callback Programming",
        "topics": [
          "Module Introduction",
          "Asynchronous I/O with Callback Programming",
          "Creating Callback Functions",
          "Promises",
          "Working with JSON",
          "Lab",
          "Module Summary",
          "Practice Quiz",
          "Graded Quiz"
        ]
      },
      "module3": {
        "name": "Express Web Application Framework",
        "topics": [
          "Module Introduction",
          "Extending Node.js",
          "Express Web Application Framework",
          "Your first Express Web Application",
          "Routing, Middleware, and Templating",
          "Lab",
          "Module Summary",
          "Practice Quiz",
          "Graded Quiz"
        ]
      },
      "module4": {
        "name": "Building a Rich Front-End Application using REACT & ES6",
        "topics": [
          "Module Introduction",
          "Introduction to ES6",
          "Introduction to Front End Frameworks and React.JS",
          "Working with React Components",
          "Passing Data and States between Components",
          "Connecting React App to External Services (WIP)",
          "Lab",
          "Module Summary",
          "Practice Quiz",
          "Graded Quiz"
        ]
      }
    }
  }
}
```

**Congratulations! You have completed the lab for the second module of this course.**

## Summary

Now that you have have learned how to use Async Callback programming we will go further and extend the capabilities of our server side.

## Author(s)

[Lavanya](#)

## Changelog

Date	Version	Changed by	Change Description
2020-11-20	1.0	Steve	Initial version created based videos
2022-10-11	1.1	Lavanya Rajalingam	Small UI Change
2022-11-17	1.2	Lavanya	Removed the async exercise as it it isolated in a diff lab now