动手实验 – 异步回调（30分钟）

Skills Network

实验目标：

完成此实验后，您将能够：

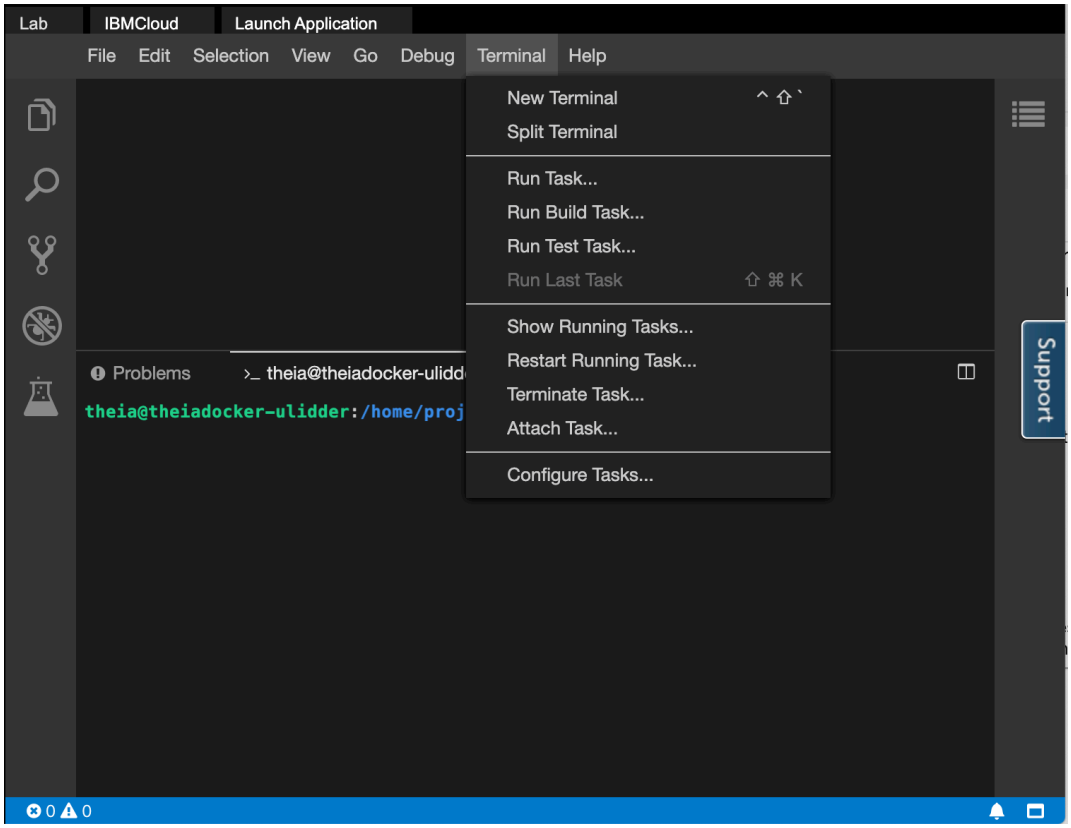
- 描述异步回调
- 创建一个 Node.js 应用程序

先决条件

- 基础的 JavaScript 知识

任务 1：设置 – 检查克隆的文件

1. 通过编辑器菜单打开一个终端窗口：终端 > 新建终端。



2. 切换到你的项目文件夹。

   ```
   cd /home/project
   ```

3. 检查您是否有文件夹 lkpho-Cloud-applications-with-Node.js-and-React

   ```
   ls /home/project
   ```

如果你这样做，可以跳到第5步。

4. 克隆包含本实验所需工件的git仓库，如果它尚不存在的话。

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

5. 切换到此实验的目录。

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

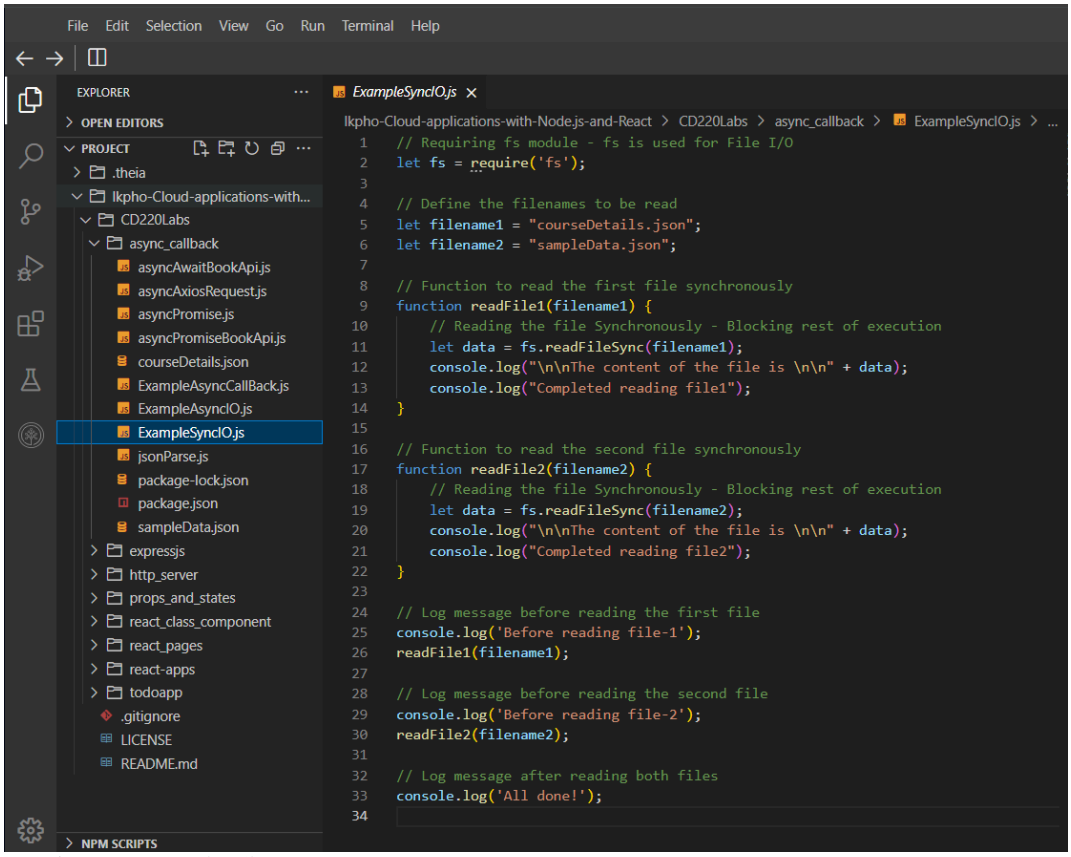
6. 列出该目录的内容以查看本实验的工件。

   ```
   ls
   ```

   你一定有一些练习文件，你将在练习中运行这些文件。

任务 2：同步 I/O

在文件资源管理器视图中，ExampleSyncIO.js 将显示如下。



▶ 您也可以点击这里查看代码

这是代码的解释：

- `require('fs')` 导入了 Node.js 内置的 fs 模块，该模块提供与文件系统相关的功能。
- `readFile1(filename1)` 和 `readFile2(filename2)` 定义了函数，用于分别同步读取 `courseDetails.json` 和 `sampleData.json`，使用 `fs.readFileSync`，这会阻塞后续执行，直到每个文件被读取。
- 在调用 `readFile1(filename1)` 和 `readFile2(filename2)` 函数之前，打印日志消息以指示每个文件读取过程的开始。
- 在启动两个文件的读取后，打印最终的日志消息 "All done!" 以指示代码已执行超出文件读取的启动。

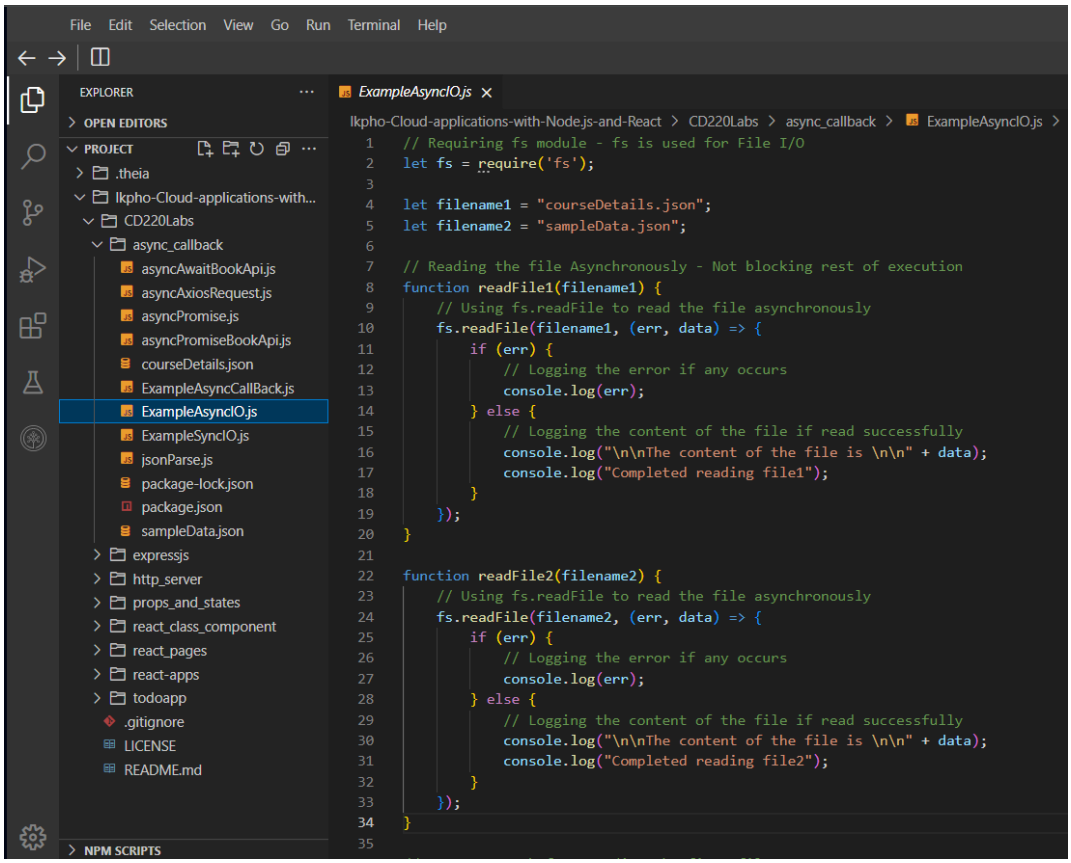
2. 在终端窗口中使用以下命令运行服务器。

   ```
   node ExampleSyncIO.js
   ```

请注意，这两个文件是同步读取的，一个接一个。这将从控制台日志出现的顺序中明显体现出来。

任务 3：异步 IO

1. 在文件资源管理器视图中，ExampleAsyncIO.js 将显示如下。



▶ 您也可以点击这里查看代码

代码说明

- `require('fs')` 导入 Node.js 内置的 fs 模块，该模块提供与文件系统相关的功能。
- `readFile1(filename1)` 和 `readFile2(filename2)` 定义了读取 `courseDetails.json` 和 `sampleData.json` 的函数，分别使用 `fs.readFile`，它异步读取文件，允许其余代码在等待文件读取完成之前执行。
- 在每个函数中，`fs.readFile(filename, callback)` 读取指定的文件。如果在读取过程中发生错误，它会记录错误；否则，它会记录文件的内容和完成消息。
- 在调用 `readFile1(filename1)` 和 `readFile2(filename2)` 函数之前，打印日志消息以指示每个文件读取过程的开始。
- 在启动两个文件的读取后，打印最终日志消息“全部完成！”以指示代码已执行到文件读取启动之后。

2. 在终端窗口中使用以下命令运行服务器。

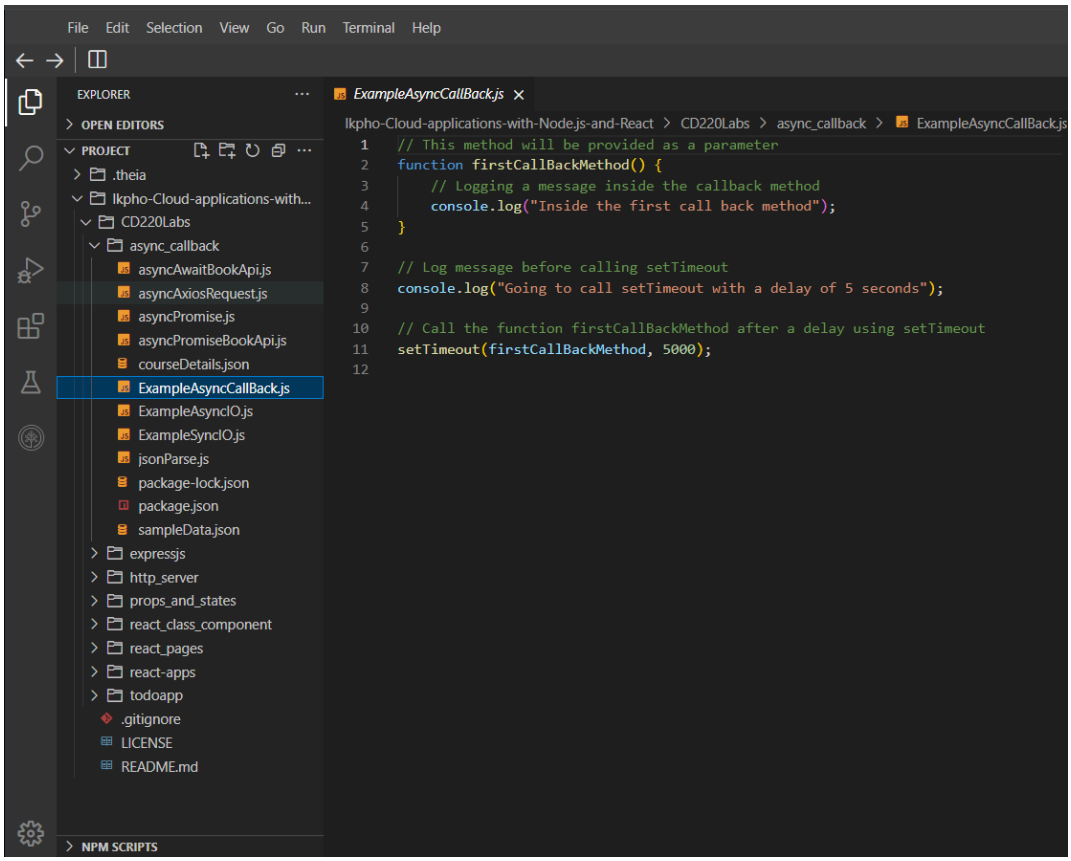
```
node ExampleAsyncIO.js
```

请注意，这两个文件是异步读取的。这将从控制台日志出现的顺序中显而易见。以下三个控制台日志在文件内容打印之前出现，尽管这些日志在代码中的调用顺序不同。

```
Before the reading the file-1
Before the reading the file-2
All done!
```

任务 4：创建回调函数

1. 在文件资源管理器视图中，ExampleAsyncCallBack.js。它会显示如下。



▼ 您也可以点击这里查看代码

```
// This method will be provided as a parameter
function firstCallBackMethod() {
    // Logging a message inside the callback method
    console.log("Inside the first call back method");
}
// Log message before calling setTimeout
console.log("Going to call setTimeout with a delay of 5 seconds");
// Call the function firstCallBackMethod after a delay using setTimeout
setTimeout(firstCallBackMethod, 5000);
```

代码解释

- `firstCallBackMethod()` 是一个记录消息“在第一个回调方法内部”的函数。这个函数将用作回调。
- 在调用 `setTimeout` 之前，打印一条日志消息“将以5秒的延迟调用setTimeout”，以指示即将设置定时器。
- `setTimeout(firstCallBackMethod, 5000)` 在5000毫秒（5秒）后调度执行 `firstCallBackMethod()`，允许其余代码立即继续执行，同时定时器在后台运行。

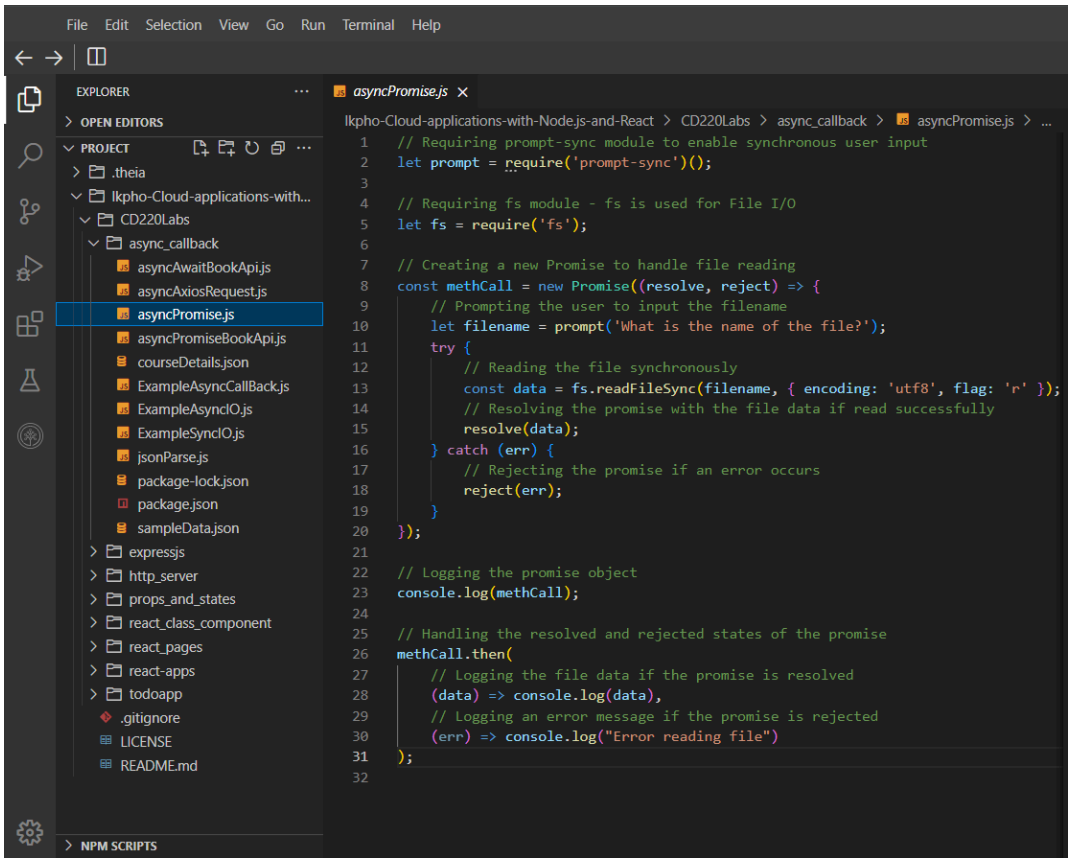
2. 在终端窗口中运行以下命令以启动服务器。

```
node ExampleAsyncCallBack.js
```

setTimeout 是一个内置库方法，允许你传递一个需要在超时后调用的方法作为参数。这里首先定义了 firstCallBackMethod，然后将其作为参数传递给 setTimeout。正如你可能已经观察到的，该方法将在 5 秒后被调用。这被称为 `callback`

任务 5：承诺

1. 在文件资源管理器中查看 asyncPromise.js。它看起来像这样。



▼ 您还可以点击这里查看代码

```
// Requiring prompt-sync module to enable synchronous user input
let prompt = require('prompt-sync')();
// Requiring fs module - fs is used for File I/O
let fs = require('fs');
// Creating a new Promise to handle file reading
const methCall = new Promise((resolve, reject) => {
    // Prompting the user to input the filename
    let filename = prompt('What is the name of the file?');
    try {
        // Reading the file synchronously
        const data = fs.readFileSync(filename, { encoding: 'utf8', flag: 'r' });
        // Resolving the promise with the file data if read successfully
        resolve(data);
    } catch (err) {
        // Rejecting the promise if an error occurs
        reject(err);
    }
});
// Logging the promise object
console.log(methCall);
// Handling the resolved and rejected states of the promise
methCall.then(
    // Logging the file data if the promise is resolved
    (data) => console.log(data),
    // Logging an error message if the promise is rejected
    (err) => console.log("Error reading file")
);
```

代码解释

- `require('prompt-sync')()` 导入 `prompt-sync` 模块以启用同步用户输入。 `require('fs')` 导入 Node.js 内置的 `fs` 模块以实现与文件系统相关的功能。
- `const methCall = new Promise((resolve, reject) => { ... })` 创建一个新的 promise 来处理文件读取过程。该 promise 接受一个带有 `resolve` 和 `reject` 参数的回调函数。
- 在 promise 内部：
  - `let filename = prompt('文件名是什么？')` 提示用户输入文件名。
  - `fs.readFileSync(filename, { encoding: 'utf8', flag: 'r' })` 尝试以 UTF-8 编码同步读取指定文件。
  - 如果文件成功读取，调用 `resolve(data)` 返回文件数据，解决 promise。
  - 如果发生错误，调用 `reject(err)` 返回错误，拒绝 promise。
- `console.log(methCall)` 记录 promise 对象。
- `methCall.then(...).catch(...)` 处理 promise：
  - `then((data) => console.log(data))` 如果 promise 被解决，则记录文件数据。
  - `catch((err) => console.log("读取文件时出错"))` 如果 promise 被拒绝，则记录错误信息。

2. 在终端窗口运行以下命令以安装 prompt-sync。使用 `--save` 确保更新 package.json 文件中的依赖项。

```
npm install --save prompt-sync
```

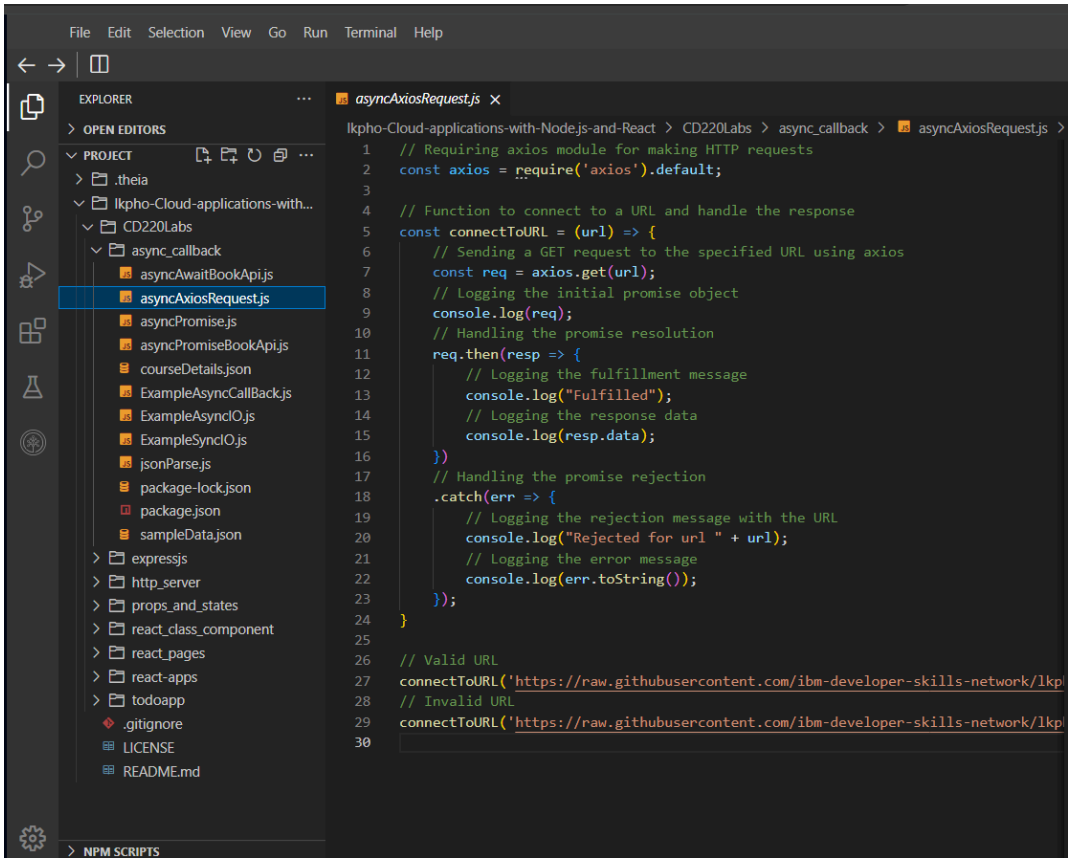
在终端窗口中运行以下命令启动服务器。它会要求您输入一个文件名。请输入当前目录中的有效文件名。

```
node asyncPromise.js
```

methCall 这里是一个 promise 对象。当 promise 被 fulfilled 时，控制台日志将被打印。再次运行上述命令并尝试传递一个无效的文件名。查看控制台日志，当 promise 被拒绝时会打印出来。

任务 6：AsyncAxiosRequest

1. 在文件资源管理器中查看 asyncAxiosRequest.js。它看起来像这样。



▼ 您也可以点击这里查看代码

```
// Requiring axios module for making HTTP requests
const axios = require('axios').default;
// Function to connect to a URL and handle the response
const connectToURL = (url) => {
    // Sending a GET request to the specified URL using axios
    const req = axios.get(url);
    // Logging the initial promise object
    console.log(req);
    // Handling the promise resolution
    req.then(resp => {
        // Logging the fulfillment message
        console.log("Fulfilled");
        // Logging the response data
        console.log(resp.data);
    })
    // Handling the promise rejection
    .catch(err => {
        // Logging the rejection message with the URL
        console.log("Rejected for url " + url);
        // Logging the error message
        console.log(err.toString());
    });
}
// Valid URL
connectToURL('https://raw.githubusercontent.com/ibm-developer-skills-network/lkpho-Cloud-applications-with-Node.js-and-React/master/
// Invalid URL
connectToURL('https://raw.githubusercontent.com/ibm-developer-skills-network/lkpho-Cloud-applications-with-Node.js-and-React/master/
```

代码解释

- `require('axios').default` 导入 `axios` 模块，该模块用于发送 HTTP 请求。
- `connectToURL(url)` 定义了一个函数，该函数以 URL 作为参数，并使用 `axios.get(url)` 向该 URL 发送 GET 请求。它记录了 `axios.get` 返回的初始 promise 对象。
- 在 `connectToURL` 函数内部：
    - `req.then(resp => { ... })` 处理 promise 的解决，通过记录“Fulfilled”和响应数据 (`resp.data`)，如果请求成功。

- `req.catch(err => { ... })` 处理 promise 的拒绝，通过记录“Rejected for url”后跟 URL 和错误信息，如果请求失败。

- `connectToURL()` 函数用于调用一个 URL。提供一个有效的 URL 应该能成功获取数据，而无效的 URL 应该会导致错误并激活拒绝处理程序。

2. 要运行此代码，我们需要安装 axios 包。运行以下命令以安装 axios。

   ```
   npm install --save axios
   ```

3. 在终端窗口中运行以下命令。

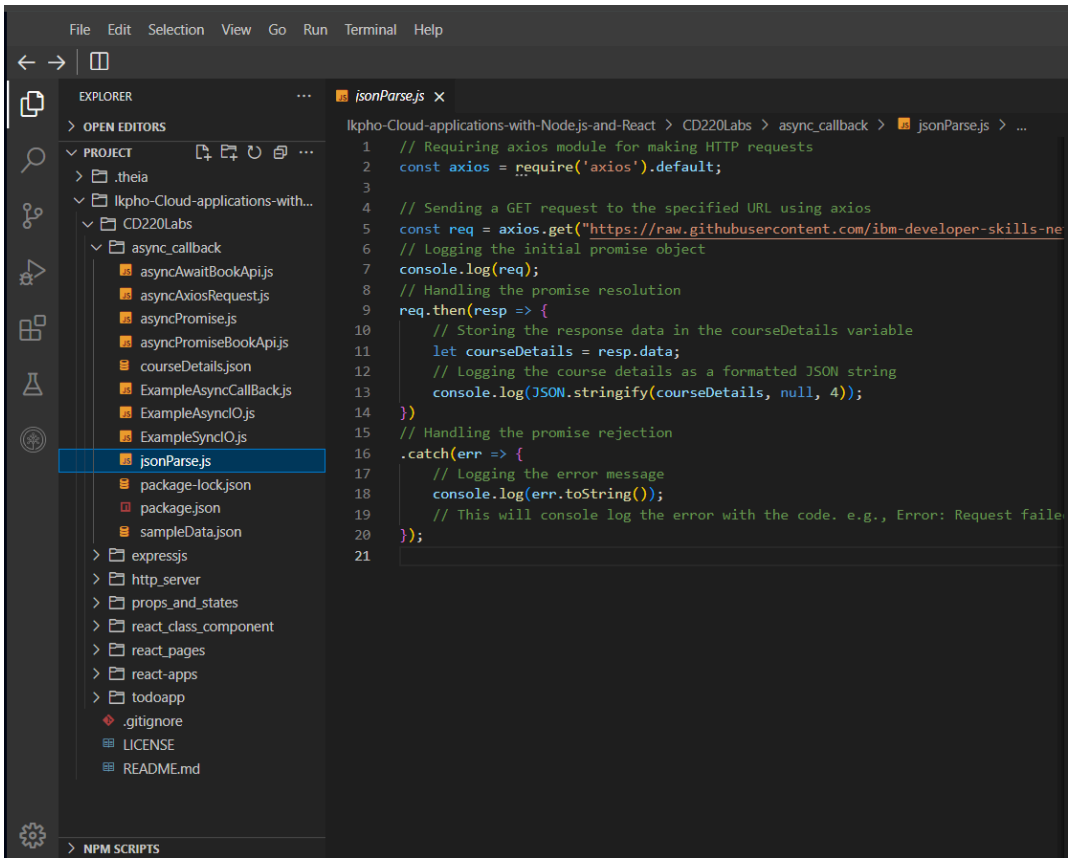
   ```
   node asyncAxiosRequest.js
   ```

当你运行代码时，第一个 connectToURL 是一个 axios 请求，指向一个有效的 URL，将返回 JSON 对象。第二个 connectToURL 是一个 axios 请求，指向一个无效的 URL。这将返回相应的错误消息。输出如下所示。

```
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' }
```

任务 7：处理 JSON

1. 在文件浏览器中查看 jsonParse.js。它会显示如下。



▶ 您也可以点击这里查看代码

代码解释

- `require('axios').default` 导入 `axios` 模块，该模块用于发起 HTTP 请求。

- `axios.get()` 向指定的 URL 发送 GET 请求以获取 `courseDetails.json` 文件。

- `req` 常量存储 `axios.get` 返回的 Promise。

- `console.log(req)` 将初始 Promise 对象记录到控制台。

- `req.then(resp => { ... })` 处理 Promise 解析：

  - `let courseDetails = resp.data` 将响应数据存储在 `courseDetails` 变量中。
  - `console.log(JSON.stringify(courseDetails, null, 4))` 将课程详情以格式化的 JSON 字符串记录，带有缩进。

- `req.catch(err => { ... })` 处理 Promise 拒绝：

  - `console.log(err.toString())` 记录错误信息，如果请求失败（例如，`Error: Request failed with status code 404`），则包括错误代码和描述。

2. 要运行此代码，我们需要安装 axios 包。您将拥有在上一个练习中安装的 axios 模块。如果没有，请运行以下命令以安装 axios。

   ```
   npm install --save axios
   ```

3. 在终端窗口中使用以下命令运行代码。

   ```
   node jsonParse.js
   ```

当您运行代码时，会向远程 URL 发送一个 axios 请求，该请求返回一个 JSON 对象。这个 JSON 对象被字符串化（或格式化为可读的形式）并记录在控制台上。输出将如下所示。



恭喜！您已完成本课程第二模块的实验。

总结

现在您已经学习了如何使用异步回调编程，我们将进一步扩展服务器端的功能。