**CS 5610**

**Lab – Node JS**

**GitHub submission link**

**Activity 1 - Basic File Operations with Node.js (Using the fs module)**

1. Open a terminal or command prompt.
2. Use the cd command to navigate to the folder where you want to create your project.
3. In your terminal, type: *npm init*
4. Follow the prompts to set up your project. A package.json file will be created, but you don’t need to worry about it for now.
5. In your project folder, create a new JavaScript file named fileOperations.js. This file will contain the code for your file operations task.
6. Perform File Operations Using fs:
   1. Import the fs module at the top of the fileOperations.js file. The fs module allows you to interact with the file system. *const fs = require('fs');*
   2. Write to a file -Use fs.writeFile() to write the string "Hello, this is a test message!" into a file called userData.txt. The fs.writeFile() function takes three arguments: the file name, the content to write, and a callback function to handle any errors.

*A computer screen with white text

Description automatically generated*

* 1. Read from the file -After successfully writing to the file, use fs.readFile() to read the contents of userData.txt. This function takes the file name, the encoding (set as 'utf-8'), and a callback to handle the error result.

A computer screen with text on it

Description automatically generated

**Activity 1b – Refactor File Operations Using Promises**

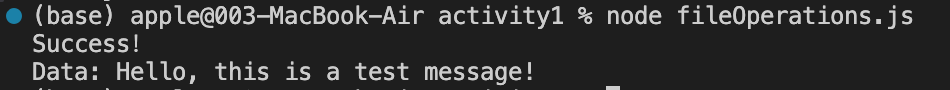
1. Modify the existing fileOperations.js file - You should already have a file named fileOperations.js from Activity 1 that uses callbacks. In this activity, you will modify the same file to refactor it using Promises instead of callbacks.
2. Refactor the writeFilePromise function. Modify the existing function that uses callbacks and refactor it to return a Promise. Use fs.writeFile() inside the Promise. This function returns a Promise and is used to write to a file. It uses fs.writeFile() to write the string "Hello, this is a test message!" to a file named userData.txt.
3. Refactor the readFilePromise function. Similarly, refactor the function to return a Promise and use fs.readFile(). This function returns a Promise and is used to read from a file. It uses fs.readFile() to read the content from the file userData.txt.
4. Use Promises in the Execution Flow.

* Use writeFilePromise() to write content to the file first.
* Once it successfully writes, use readFilePromise() to read the content.
* Ensure that:
  + After successfully writing the file, it outputs "Success!" to the console.
  + After reading the file, it outputs "Data: Hello, this is a test message!".

1. Use .catch() to handle and output any potential errors during file operations.
2. Run the following command:

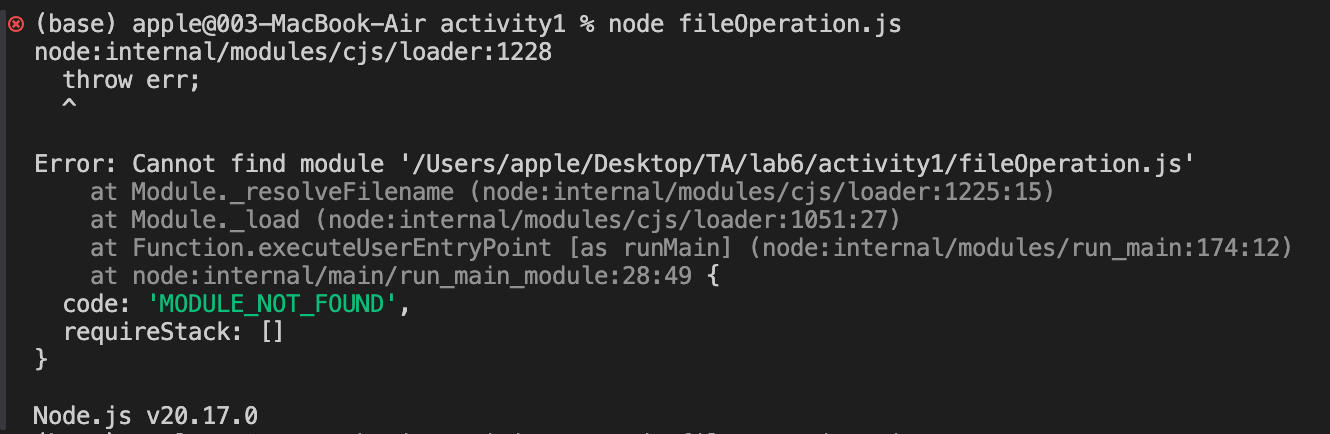
node fileOperations.js

Output:



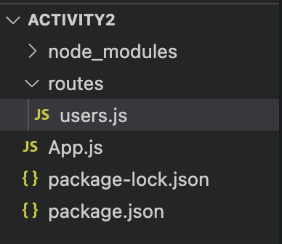
node fileOperation.js

Output:

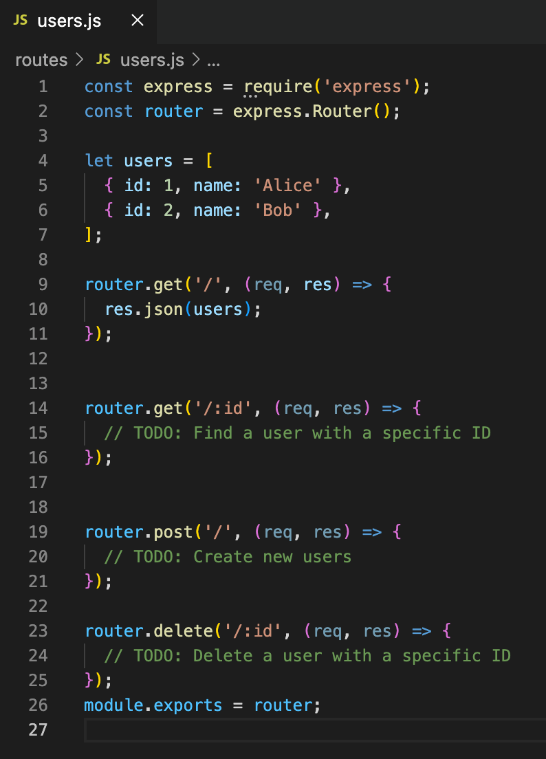
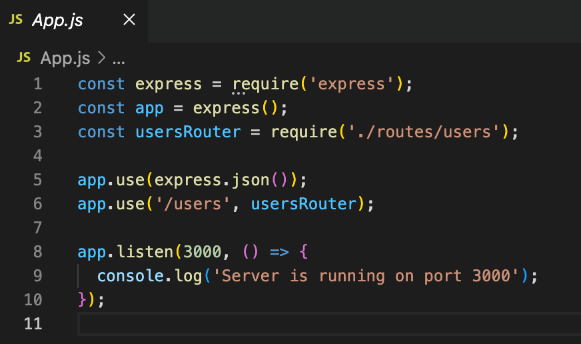


**Activity 2 – Express Routes and Modularization**

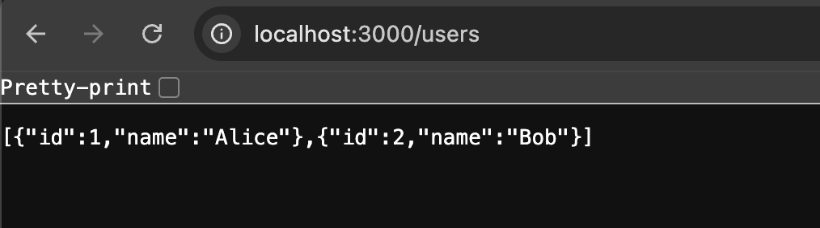
1. Initialize a Node.js project using npm init and follow the prompts to create a package.json file.
2. Install Express using npm install express.
3. Install curl using brew install curl.
4. Create a new file named App.js where you will set up your main Express application.
5. Create a routes folder and inside it, create a users.js file to manage user-related routes.



1. Download and copy the starter code into corresponding file.

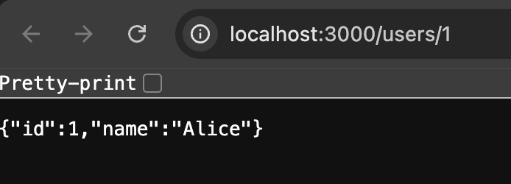
 

1. Run node app.js in your terminal. If the output is Server is running on port 3000, open your browser and navigate to <http://localhost:3000/users> to verify that the server is working correctly.

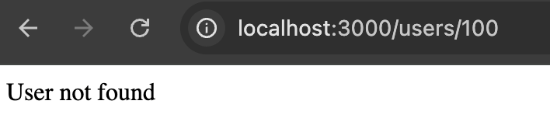


Implementation:

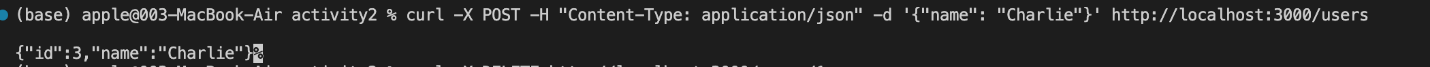
1. You have been provided with some starter code, and your task is to complete it by working on the remaining three TODO sections in users.js:
2. Testing (Keep running node app.js in your terminal):
3. Run http://localhost:3000/users/1 in browser:



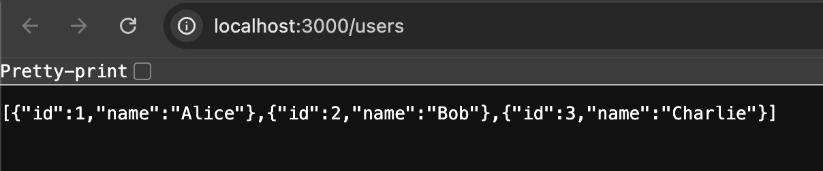
1. Run http://localhost:3000/users/100 in browser:



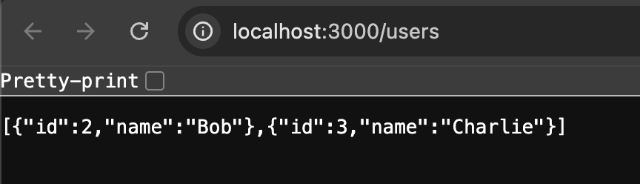
1. Run curl -X POST -H "Content-Type: application/json" -d '{"name": "Charlie"}' in a new terminal:



1. Run http://localhost:3000/users in browser:



1. Run curl -X DELETE http://localhost:3000/users/1 in a new terminal:
2. Run http://localhost:3000/users in browser:



**Activity 4 – Pug**

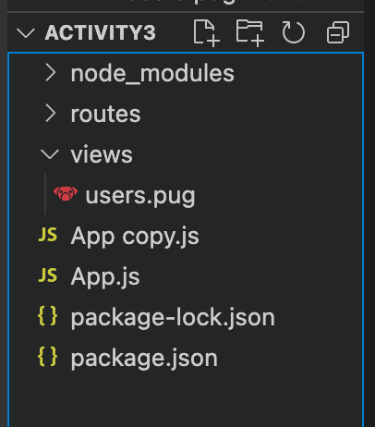
Continue working in the folder from Activity 2.

1. Set Up Pug:

Install Express and Pug: npm install express pug

Create the views folder in your project directory.

Inside the views folder, create users.pug



1. Implementation:

In App.js, copy the starter code and set up Pug as the view engine.

In users.pug, display a list of all users, and include an unordered list (ul) to list users and a link to add a new user.

1. Testing:

Keep running node app.js in your terminal)

Run http://localhost:3000/users in browser:

