HyperionDev

Express – Web Applications

Model Answer Approach

Visit our website

Auto-graded task 1

1. Steps taken to complete the task

The approach to achieving the model answer involves a few key steps:

A. File reading and data handling:

The code reads data from a JSON file to dynamically provide content for the root route. This involves using `fs.readFileSync` to synchronously read the file and `JSON.parse` to convert the JSON string into a JavaScript object.

B. Express setup:

An Express.js server is set up to serve static files from a public directory. The server is configured to respond to HTTP GET requests at the root URL by reading data from the JSON file and sending a personalised welcome message.

C. 404 Handling:

Middleware has been used to catch all unhandled routes and return a 404 status with a user-friendly message.

2. Reasons this approach works well

This approach is fitting for several reasons:

A. Dynamic content:

Reading from a JSON file allows for easy updates to the content without modifying the code. This is useful for content management and maintaining flexibility.

B. Static file serving:

Using Express's static middleware is an efficient way to serve static assets like HTML, CSS, and JavaScript files, which can be directly accessed by the client.

C. Simplicity:

The approach keeps things simple and easy to understand, making it a good choice for a straightforward application or a starting point for more complex projects.

3. Potential challenges and considerations

Some potential pitfalls to be aware of:

A. Synchronous file reading:

Using `fs.readFileSync` is not ideal for performance in production environments because it blocks the event loop. For a more scalable solution, we can consider using `fs.promises.readFile` with asynchronous handling.

B. Error handling limitations:

Basic error handling might not cover all edge cases. For example, if the JSON structure is incorrect, it might still cause issues. More robust validation and error handling should be implemented for production applications. But we can ignore this for now as this is just an introduction to working with Express.js

Auto-graded task 2

To complete this task, the Express server was set up by copying the example folder locally, installing dependencies with `npm install`, and starting the server with the `npm start` command. After confirming that the server was started, Postman was used to test each of the required endpoints, screenshotting each response.

The chosen approach with Postman ensures a thorough testing of each of the endpoints present within the example file, while the screenshots highlight that the server was functioning as expected during the testing.

Some considerations include ensuring that the correct port is being used on each request, as an incorrect port or trailing whitespace could lead to potential errors. Additionally, if the `start` script has not been set up correctly, the server may fail to start with the `npm start` command.

As a whole, the taken approach effectively validates and tests each of the endpoints, and includes the required screenshots.