Backend Track: Week 1 Assessment – Node.js Fundamentals & JSON Data Handling

This assessment is part of a 5-week Backend learning journey. Each assessment builds progressively, guiding learners from fundamentals to advanced concepts in Node.js development.

Objective

This assessment introduces participants to the fundamental concepts of Node.js and its role in backend development. The primary objective is to familiarize participants with working with JSON data, a ubiquitous format in web services, by programmatically creating, reading, and manipulating JSON files. This exercise lays the groundwork for understanding data representation and basic server-side scripting, crucial for building robust backend applications.

Scenario/Problem Statement

In the initial stages of any software project, especially those involving data exchange, understanding how to structure and manage data is paramount. JSON (JavaScript Object Notation) has become the de facto standard for data interchange on the web due to its human-readability and ease of parsing by machines. As a budding backend developer, your first task is to simulate a basic data storage mechanism for user profiles. You need to create a system that can store sample user data in a JSON file, retrieve it, and potentially update it. This simulates a common initial requirement in backend development: managing application data without the immediate complexity of a full-fledged database, focusing instead on the core principles of data serialization and deserialization.

Key Concepts Covered

- **Node.js Basics:** Understanding the Node.js runtime environment, its event-driven architecture, and non-blocking I/O model.
- **File System Module (`fs`):** Using Node.js's built-in `fs` module for reading from and writing to files asynchronously and synchronously.
- **JSON (JavaScript Object Notation):** Understanding JSON syntax, `JSON.parse()` for converting JSON strings to JavaScript objects, and `JSON.stringify()` for converting JavaScript objects to JSON strings.
- **JavaScript Data Structures:** Working with JavaScript objects and arrays to represent structured data.
- **Error Handling:** Implementing basic error handling for file system operations.
- **Asynchronous Programming:** Introduction to callbacks and Promises for handling asynchronous operations in Node.js.

Task Instructions

Participants will develop a Node.js script to manage sample user data stored in a JSON file.

Step 1: Project Setup

1. **Create Project Directory:** Create a new directory for your project, e.g., `user-data-manager`.
2. **Initialize Node.js Project:** Navigate into the directory and initialize a new Node.js project: `bash mkdir user-data-manager cd user-data-manager npm init -y`
3. **Create Data File:** Create an empty JSON file named `users.json` in your project directory. This file will store your sample user data.

Step 2: Generate Sample User Data

1. **Create `generateData.js` :** Create a new JavaScript file named `generateData.js`.

2. **Define User Schema:** In `generateData.js`, define a JavaScript array of objects, where each object represents a user. Each user object should have at least the following properties:
    - `id`: A unique identifier (e.g., a number).
    - `firstName`: User's first name.
    - `lastName`: User's last name.
    - `email`: User's email address.
    - `age`: User's age.
    - Include at least 5 sample user objects in this array.

`javascript // Example user data structure const users = [ { id: 1, firstName: 'Alice', lastName: 'Smith', email: 'alice.smith@example.com', age: 30 }, { id: 2, firstName: 'Bob', lastName: 'Johnson', email: 'bob.j@example.com', age: 24 }, // ... more users ]; 3. **Write to `users.json`:** Use Node.js's `fs.writeFile()` (or `fs.writeFileSync()` for simplicity in this initial task) to write this JavaScript array as a JSON string to the `users.json` file. Remember to use `JSON.stringify()` to convert the JavaScript array to a JSON string.

```javascript const fs = require('fs'); const users = [ / *... your user data ...* / ];

fs.writeFile('users.json', JSON.stringify(users, null, 2), (err) => { if (err) { console.error('Error writing file:', err); return; } console.log('Sample user data written to users.json'); }); `4. **Execute Script:** Run the script from your terminal:` bash node generateData.js `` Verify that `users.json` is created and contains the sample data.

## Step 3: Read and Display User Data

1. **Create `readData.js`:** Create a new JavaScript file named `readData.js`.
2. **Read from `users.json`:** Use `fs.readFile()` (or `fs.readFileSync()`) to read the content of `users.json`.
3. **Parse JSON:** Once the file content is read (which will be a string), use `JSON.parse()` to convert the JSON string back into a JavaScript array of objects.
```

4. **Display Data:** Iterate over the parsed user data and print each user's `firstName` and `email` to the console.

   ```javascript
   const fs = require('fs');

   fs.readFile('users.json', 'utf8', (err, data) => {
     if (err) {
       console.error('Error reading file:', err);
       return;
     }
     try {
       const users = JSON.parse(data);
       console.log('--- All Users ---');
       users.forEach(user => {
         console.log(`Name: ${user.lastName}, Email: ${user.email}`);
       });
     } catch (parseErr) {
       console.error('Error parsing JSON:', parseErr);
     }
   });
   ```

   5. **Execute Script:** Run the script from your terminal:

   ```bash
   node readData.js
   ```

   Verify that the user data is displayed correctly in your console.

Step 4: Add a New User (Optional but Recommended)

1. **Create `addUser.js`:** Create a new JavaScript file named `addUser.js`.
2. **Read Existing Data:** Read the current content of `users.json`.
3. **Parse and Add:** Parse the JSON, add a new sample user object to the array, and then stringify the updated array.
4. **Write Updated Data:** Write the updated JSON string back to `users.json`.
5. **Execute and Verify:** Run `addUser.js` and then `readData.js` to confirm the new user has been added.

Step 5: Documentation and Submission

1. **README.md:** Create a `README.md` file in the root of your project. This file should include:
   - A brief description of the project and its purpose.
   - Instructions on how to run each script (`generateData.js`, `readData.js`, `addUser.js`).
   - A list of the technologies used (Node.js, JSON).
   - Any challenges you faced and how you overcame them.
2. **Code Comments:** Add comments to your JavaScript files to explain the logic, especially for file I/O and JSON parsing.

Deliverables

Participants are required to submit a single ZIP file containing the entire Node.js project directory. The submission should include:

- The complete `user-data-manager` project folder.
- A well-documented `README.md` file.
- Clean, well-structured, and commented JavaScript files (`generateData.js` , `readData.js` , `addUser.js`).
- The `users.json` file with the initial sample data (or updated if `addUser.js` was implemented).

Evaluation Criteria

Assessments will be evaluated based on the following criteria, with a total of 100 points:

- **Project Setup & Initialization (10 points):**
  - Correct Node.js project setup (`package.json`).
  - Proper creation of `users.json` .
- **Data Generation (30 points):**
  - Successful creation of `generateData.js` .
  - Correct definition of sample user data structure.
  - Accurate writing of JSON data to `users.json` using `fs` and `JSON.stringify()` .
- **Data Reading & Display (30 points):**
  - Successful creation of `readData.js` .
  - Correct reading of `users.json` using `fs` .
  - Accurate parsing of JSON data using `JSON.parse()` .
  - Proper display of user data to the console.
- **Adding New User (Optional, up to 15 bonus points):**
  - Successful implementation of `addUser.js` .

    - Correct logic for reading, modifying, and writing back JSON data.
- **Documentation and Professionalism (10 points):**
    - A comprehensive `README.md` file.
    - Well-commented and readable code.
    - Adherence to project structure and submission guidelines.

This assessment provides a solid entry point into backend development with Node.js, focusing on the fundamental skill of handling data in JSON format and interacting with the file system. These skills are essential building blocks for developing more complex server-side applications and APIs in subsequent weeks.

Looking Ahead

In Week 2, you will build upon this foundation by learning how to create a basic web server and define API endpoints using Express.js, moving from file-based data management to serving data over HTTP. Week 3 will introduce database integration, Week 4 will cover authentication and advanced database operations, and Week 5 will focus on deployment and scalability.

Recommended Resources

- **Node.js Official Documentation:** https://nodejs.org/en/docs/
- **MDN Web Docs (JSON):** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
- **Node.js File System Module (fs):** https://nodejs.org/docs/latest/api/fs.html
- **JavaScript Promises - MDN:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- **W3Schools Node.js Tutorial:** https://www.w3schools.com/nodejs/
