Backend Track: Week 2 Assessment – Building RESTful APIs with Express.js

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 challenges participants to build their first RESTful API using Express.js, a popular Node.js web framework. The primary objective is to understand the principles of REST architecture, HTTP methods, and how to create API endpoints that serve JSON data. Participants will develop an API that exposes their skills data (from Week 1) through well-structured endpoints, demonstrating proficiency in server-side development and API design.

Scenario/Problem Statement

Having mastered basic data handling with JSON files in Week 1, the next logical step is to make this data accessible to external applications through a web API. In today's interconnected digital ecosystem, APIs serve as the backbone for communication between different software systems. Your task is to transform your static JSON data into a dynamic, accessible service by creating a RESTful API that can serve your skills data to potential frontend applications or other services. This simulates a common real-world scenario where backend developers must expose data and functionality through well-designed APIs that follow industry standards and best practices.

Key Concepts Covered

- **Express.js Framework:** Understanding the Express.js framework for building web applications and APIs in Node.js.

- **RESTful API Design:** Implementing REST (Representational State Transfer) principles for creating predictable and scalable APIs.
- **HTTP Methods:** Proper use of HTTP methods (GET, POST, PUT, DELETE) for different operations.
- **Routing:** Creating and organizing API routes using Express.js routing capabilities.
- **Middleware:** Understanding and implementing Express.js middleware for request processing, logging, and error handling.
- **JSON Response Handling:** Sending properly formatted JSON responses from API endpoints.
- **CORS (Cross-Origin Resource Sharing):** Enabling cross-origin requests to allow frontend applications to consume the API.
- **Error Handling:** Implementing proper error responses and status codes for different scenarios.

Task Instructions

Participants will create a comprehensive RESTful API for managing skills data using Express.js.

Step 1: Project Setup and Dependencies

1. **Create New Project:** Create a new directory for your API project, e.g., `skills-api`.
2. **Initialize Node.js Project:** `bash mkdir skills-api cd skills-api npm init -y`
3. **Install Dependencies:** Install Express.js and other necessary packages: `bash npm install express cors npm install --save-dev nodemon`
4. **Update package.json:** Add development scripts to your `package.json`: `json { "scripts": { "start": "node server.js", "dev": "nodemon server.js" } }`

Step 2: Create Basic Express Server

1. **Create `server.js`:** Create the main server file with basic Express setup: ```javascript const express = require('express'); const cors = require('cors'); const fs = require('fs').promises; const path = require('path');

   const app = express(); const PORT = process.env.PORT || 3001;

   // Middleware app.use(cors()); app.use(express.json());

   // Basic route app.get('/', (req, res) => { res.json({ message: 'Skills API is running!' }); });

   // Start server app.listen(PORT, () => { console.log(`Server is running on port ${PORT}`); }); ```

Step 3: Create Skills Data File

1. **Create `data/skills.json`:** Create a `data` directory and a `skills.json` file with your skills data: `json [{ "id": 1, "name": "JavaScript", "category": "Programming Language", "proficiency": "Expert", "description": "Modern JavaScript including ES6+ features" }, { "id": 2, "name": "Node.js", "category": "Runtime Environment", "proficiency": "Intermediate", "description": "Server-side JavaScript development" }, { "id": 3, "name": "React", "category": "Frontend Framework", "proficiency": "Advanced", "description": "Building interactive user interfaces" }, { "id": 4, "name": "Express.js", "category": "Backend Framework", "proficiency": "Intermediate", "description": "Web application framework for Node.js" }, { "id": 5, "name": "Problem Solving", "category": "Soft Skill", "proficiency": "Expert", "description": "Analytical thinking and solution development" }]`

Step 4: Implement Skills API Endpoints

1. **GET /api/skills - Retrieve All Skills:** `javascript // Get all skills app.get('/api/skills', async (req, res) => { try { const data = await fs.readFile(path.join(__dirname, 'data', 'skills.json'), 'utf8'); const skills = JSON.parse(data); res.json({ success: true, count:`

`skills.length, data: skills }); } catch (error) { res.status(500).json({ success: false, message: 'Error reading skills data', error: error.message }); } });`

2. **GET /api/skills/:id - Retrieve Single Skill:** ```javascript // Get single skill by ID app.get('/api/skills/:id', async (req, res) => { try { const data = await fs.readFile(path.join(__dirname, 'data', 'skills.json'), 'utf8'); const skills = JSON.parse(data); const skill = skills.find(s => s.id === parseInt(req.params.id));

```
 if (!skill) {
   return res.status(404).json({
     success: false,
     message: 'Skill not found'
   });
 }

res.json({
  success: true,
  data: skill
});
```

   } catch (error) { res.status(500).json({ success: false, message: 'Error reading skills data', error: error.message }); } }); ```

3. **GET /api/skills/category/:category - Filter by Category:** ```javascript // Get skills by category app.get('/api/skills/category/:category', async (req, res) => { try { const data = await fs.readFile(path.join(__dirname, 'data', 'skills.json'), 'utf8'); const skills = JSON.parse(data); const filteredSkills = skills.filter(skill => skill.category.toLowerCase() === req.params.category.toLowerCase());

```
 res.json({
   success: true,
   count: filteredSkills.length,
   data: filteredSkills
});
```

   } catch (error) { res.status(500).json({ success: false, message: 'Error reading skills data', error: error.message }); } }); ```

Step 5: Add Query Parameters Support

1. **Enhance GET /api/skills with Query Parameters:** ```javascript // Enhanced get all skills with query parameters app.get('/api/skills', async (req, res) => { try {

const data = await fs.readFile(path.join(__dirname, 'data', 'skills.json'), 'utf8'); let skills = JSON.parse(data);

```
 // Filter by proficiency if provided
if (req.query.proficiency) {
  skills = skills.filter(skill =>
    skill.proficiency.toLowerCase() ===
req.query.proficiency.toLowerCase()
  );
}

// Filter by category if provided
if (req.query.category) {
  skills = skills.filter(skill =>

skill.category.toLowerCase().includes(req.query.category.toLowerCase())
  );
}

// Sort by name if requested
if (req.query.sort === 'name') {
  skills.sort((a, b) => a.name.localeCompare(b.name));
}

res.json({
  success: true,
  count: skills.length,
  data: skills
});
```

} catch (error) { res.status(500).json({ success: false, message: 'Error reading skills data', error: error.message }); } }); ```

Step 6: Add Middleware and Error Handling

1. **Create Request Logging Middleware:** `` `javascript // Request logging middleware const requestLogger = (req, res, next) => { console.log( ``$`{new Date().toISOString()} `- ${req.method} ${req.path}`); next(); };

   app.use(requestLogger); ```

2. **Add 404 Handler:** `javascript // 404 handler app.use('*', (req, res) => { res.status(404).json({ success: false, message: 'Route not found' }); });`

Step 7: Testing and Documentation

1. **Test API Endpoints:** Use a tool like Postman, curl, or create a simple HTML file to test your API endpoints:
   - `GET http://localhost:3001/api/skills`
   - `GET http://localhost:3001/api/skills/1`
   - `GET http://localhost:3001/api/skills/category/programming%20language`
   - `GET http://localhost:3001/api/skills?proficiency=expert`
2. **Create API Documentation:** Create an `API.md` file documenting all your endpoints: ```markdown # Skills API Documentation

Base URL

`http://localhost:3001`

Endpoints

GET /api/skills

Returns all skills with optional filtering.

Query Parameters: - `proficiency`: Filter by proficiency level - `category`: Filter by category - `sort`: Sort results (currently supports 'name')

Example: `GET /api/skills?proficiency=expert&sort=name` ```

Step 8: Documentation and Submission

1. **Update README.md:** Create a comprehensive `README.md` file including:
   - Project description and purpose
   - Installation and setup instructions
   - API endpoint documentation

  - Example requests and responses
  - Technologies used
  - Challenges faced and solutions

Deliverables

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

- The complete `skills-api` project folder
- A comprehensive `README.md` file
- An `API.md` file with endpoint documentation
- Clean, well-structured, and commented code
- The `skills.json` data file

Evaluation Criteria

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

- **Express.js Setup & Configuration (15 points):**
  - Correct Express.js installation and basic server setup
  - Proper middleware configuration (CORS, JSON parsing)
  - Appropriate project structure
- **API Endpoint Implementation (40 points):**
  - Successful implementation of GET /api/skills (all skills)
  - Successful implementation of GET /api/skills/:id (single skill)
  - Implementation of filtering endpoints or query parameters
  - Proper HTTP status codes and response formatting
- **Error Handling & Middleware (20 points):**

    - Appropriate error handling for file operations
    - Proper 404 handling for non-existent routes
    - Implementation of request logging or other middleware

- **Code Quality & Structure (15 points):**
    - Clean, readable, and well-organized code
    - Proper use of async/await for file operations
    - Consistent coding style and naming conventions
- **Documentation & Testing (10 points):**
    - Comprehensive README.md and API documentation
    - Evidence of testing (screenshots, curl commands, or test files)
    - Clear instructions for running the project

This assessment establishes the foundation for building robust backend APIs, introducing participants to the essential concepts of RESTful design and Express.js development that will be crucial for more advanced backend development in subsequent weeks.

Looking Ahead

In Week 3, you will transition from file-based data storage to integrating a real database with your Express.js API, learning how to perform CRUD operations on persistent data. Week 4 will delve into advanced database operations and crucial security aspects like authentication and authorization. Finally, Week 5 will cover deployment strategies, testing, and scalability considerations for production-ready backend applications.

Recommended Resources

- **Express.js Official Documentation:** https://expressjs.com/
- **MDN Web Docs (HTTP Methods):** https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

- **RESTful API Tutorial:** https://www.restapitutorial.com/
- **CORS Explained:** https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- **Nodemon GitHub:** https://github.com/remy/nodemon
