# Task 3: REST API to Manage Books Using Node.js and Express

**Author:** Soumen Das
**Internship:** Web Development Internship - Elevate Labs & MSME
**Date:** November 17, 2025
**Task:** Create a REST API to manage a list of books using Node.js and Express

## Executive Summary

This report documents the completion of Task 3 from the Web Development Internship program. The task involves building a fully functional REST API using Node.js and Express that manages a list of books with complete CRUD (Create, Read, Update, Delete) operations. The API stores book data in memory and provides endpoints for managing book records through HTTP requests.

## Objective

Build simple REST API endpoints for CRUD operations on books with no database required, storing data in memory.

## Technologies & Tools

| Tool | Purpose | Version |
|---|---|---|
| **Node.js** | JavaScript runtime environment | Latest LTS |
| **Express.js** | Web framework for REST API | 4.18.2+ |
| **npm** | Package manager | Latest |
| **Postman** | API testing tool | Desktop/Web |
| **VS Code** | Code editor | Latest |

## Project Setup Instructions

### Step 1: Initialize the Project

```
mkdir book-api
cd book-api
npm init -y
```

### Step 2: Install Express Framework

```
npm install express
```

### Step 3: Install Development Dependencies (Optional)

For auto-restart during development:

```
npm install --save-dev nodemon
```

### Step 4: Create server.js File

Create a file named `server.js` with the Express server implementation.

### Step 5: Update package.json Scripts

Add the following scripts section:

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

### API Implementation

### Server Configuration

The Express server runs on **port 3000** and uses JSON middleware for parsing request bodies:

```
const express = require('express');
const app = express();
const PORT = 3000;

app.use(express.json());
```

### In-Memory Data Storage

Books are stored in a JavaScript array with the following structure:

```
let books = [
  { id: 1, title: 'The Great Gatsby', author: 'F. Scott Fitzgerald' },
  { id: 2, title: 'To Kill a Mockingbird', author: 'Harper Lee' },
  { id: 3, title: '1984', author: 'George Orwell' }
];
```

```
let nextId = 4;
```

## API Endpoints

### 1. GET /books - Retrieve All Books

**Description:** Fetches all books from the collection

**Request:**

```
GET http://localhost:3000/books
```

**Response (200 OK):**

```
{
  "success": true,
  "message": "Books retrieved successfully",
  "data": [
    { "id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald" },
    { "id": 2, "title": "To Kill a Mockingbird", "author": "Harper Lee" },
    { "id": 3, "title": "1984", "author": "George Orwell" }
  ],
  "count": 3
}
```

### 2. GET /books/:id - Retrieve a Specific Book

**Description:** Fetches a single book by its ID

**Request:**

```
GET http://localhost:3000/books/1
```

**Response (200 OK):**

```
{
  "success": true,
  "message": "Book retrieved successfully",
  "data": { "id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald" }
}
```

**Response (404 Not Found):**

```
{
  "success": false,
```

```
  "message": "Book with ID 999 not found",
  "data": null
}
```

### 3. POST /books - Add a New Book

**Description:** Creates a new book and adds it to the collection

**Request:**

```
POST http://localhost:3000/books
Content-Type: application/json

{
  "title": "The Catcher in the Rye",
  "author": "J.D. Salinger"
}
```

**Response (201 Created):**

```
{
  "success": true,
  "message": "Book added successfully",
  "data": { "id": 4, "title": "The Catcher in the Rye", "author": "J.D. Salinger" }
}
```

**Response (400 Bad Request):**

```
{
  "success": false,
  "message": "Title and author are required",
  "data": null
}
```

### 4. PUT /books/:id - Update a Book

**Description:** Updates an existing book's details by ID

**Request:**

```
PUT http://localhost:3000/books/1
Content-Type: application/json

{
  "title": "The Great Gatsby (Revised Edition)",
  "author": "F. Scott Fitzgerald"
}
```

**Response (200 OK):**

```
{
  "success": true,
  "message": "Book updated successfully",
  "data": { "id": 1, "title": "The Great Gatsby (Revised Edition)", "author": "F. Scott F
}
```

### 5. DELETE /books/:id - Delete a Book

**Description:** Removes a book from the collection by ID

**Request:**

```
DELETE http://localhost:3000/books/1
```

**Response (200 OK):**

```
{
  "success": true,
  "message": "Book deleted successfully",
  "data": { "id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald" }
}
```

### Testing the API with Postman

### Method 1: Testing GET All Books

1. Open Postman
2. Create a new request with method: **GET**
3. URL: `http://localhost:3000/books`
4. Click **Send**

### Method 2: Testing POST (Add Book)

1. Method: **POST**
2. URL: `http://localhost:3000/books`
3. Go to **Body** tab → Select **raw** → Choose **JSON**
4. Enter:

```
{
  "title": "Pride and Prejudice",
```

```
  "author": "Jane Austen"
}
```

5. Click **Send**

### Method 3: Testing PUT (Update Book)

1. Method: **PUT**
2. URL: `http://localhost:3000/books/2`
3. Body (JSON):

```
{
  "title": "To Kill a Mockingbird - Anniversary Edition",
  "author": "Harper Lee"
}
```

4. Click **Send**

### Method 4: Testing DELETE

1. Method: **DELETE**
2. URL: `http://localhost:3000/books/3`
3. Click **Send**

## Running the Application

## Start the Server

**Using npm start:**

```
npm start
```

**Using npm dev (with auto-restart):**

```
npm run dev
```

**Expected Console Output:**

```
 Server is running on http://localhost:3000
 Available endpoints:
   GET    http://localhost:3000/books
   GET    http://localhost:3000/books/:id
   POST   http://localhost:3000/books
   PUT    http://localhost:3000/books/:id
   DELETE http://localhost:3000/books/:id
```

### Key Features Implemented

✓ **GET Endpoint:** Retrieve all books or a specific book by ID
✓ **POST Endpoint:** Add new books with validation
✓ **PUT Endpoint:** Update existing book information
✓ **DELETE Endpoint:** Remove books from the collection
✓ **JSON Responses:** Standardized response format with success flags
✓ **Error Handling:** Proper HTTP status codes (200, 201, 400, 404, 500)
✓ **Input Validation:** Checks for required fields before adding books
✓ **In-Memory Storage:** Fast data operations without database setup

### Learning Outcomes

By completing this task, the following competencies have been achieved:

1. **REST API Fundamentals:** Understanding of REST principles and HTTP methods
2. **Express.js Routing:** Creating and managing multiple route handlers
3. **HTTP Methods:** Proper implementation of GET, POST, PUT, DELETE
4. **JSON Handling:** Request and response parsing with JSON
5. **Error Handling:** Implementing appropriate HTTP status codes
6. **API Testing:** Using Postman to validate endpoints
7. **Middleware:** Understanding and using Express middleware for JSON parsing

### File Structure

```
book-api/
├── server.js           # Main Express server implementation
├── package.json        # Project dependencies and scripts
└── node_modules/       # Installed packages (after npm install)
```

### Troubleshooting

**Issue:** Port 3000 is already in use

- **Solution:** Change PORT constant in server.js to an available port (e.g., 3001, 3002)

**Issue:** "Cannot find module 'express'"

- **Solution:** Run `npm install express`

**Issue:** Postman shows "Could not get any response"

- **Solution:** Ensure the server is running with `npm start`

## Conclusion

This task successfully demonstrates the creation of a complete REST API for managing book records using Node.js and Express. The implementation includes all required CRUD operations, proper error handling, and follows REST API best practices. The API is ready for integration with frontend applications or mobile clients and can be easily extended with database persistence in future iterations.

**Status:** ✓ Complete
**Author:** Soumen Das
**Internship Program:** Web Development - Elevate Labs & MSME
**Date:** November 17, 2025

⁂