Node.js API Server - Beginner's Guide

📋 Table of Contents

🎯 Overview

This is a beginner-friendly RESTful API server built with Node.js and Express.js. It provides basic CRUD (Create, Read, Update, Delete) operations for managing users.

Features:

- RESTful API endpoints
- JSON data format
- Error handling
- CORS support
- In-memory data storage
- Input validation

📚 Prerequisites

Before you begin, make sure you have:

- **Node.js** (version 14.0 or higher) - Download here
- **npm** (comes with Node.js)
- **Postman** - Download here
- Basic understanding of JavaScript

🚀 Installation & Setup

Step 1: Create Project Directory

```bash
```

```bash
mkdir nodejs-api-server
cd nodejs-api-server
```

Step 2: Initialize Node.js Project

```bash
npm init -y
```

Step 3: Install Dependencies

```bash
npm install express cors
npm install --save-dev nodemon
```

Step 4: Create Files

1. Create `server.js` file and copy the server code
2. Update `package.json` with the provided configuration

Step 5: Run the Server

```bash
# For development (auto-restart on changes)
npm run dev

# For production
npm start
```

Step 6: Verify Installation

Open your browser and go to `http://localhost:3000`. You should see a welcome message.

🛠️ API Endpoints

Base URL: `http://localhost:3000`

Method	Endpoint	Description	Body Required
GET	`/`	Welcome message & API info	No
GET	`/users`	Get all users	No
GET	`/users/:id`	Get user by ID	No
POST	`/users`	Create new user	Yes

Method	Endpoint	Description	Body Required
PUT	`/users/:id`	Update user by ID	Yes
DELETE	`/users/:id`	Delete user by ID	No

Request/Response Examples

1. GET All Users

Request: `GET /users` **Response:**

```json
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "John Doe",
      "email": "john@example.com",
      "age": 25
    }
  ],
  "count": 1
}
```

2. Create New User

Request: `POST /users` **Body:**

```json
{
  "name": "Alice Johnson",
  "email": "alice@example.com",
  "age": 28
}
```

Response:

```json
```

```json
{
  "success": true,
  "message": "User created successfully",
  "data": {
    "id": 4,
    "name": "Alice Johnson",
    "email": "alice@example.com",
    "age": 28
  }
}
```

📮 Postman Testing Guide

Setting Up Postman Collection

Step 1: Create New Collection

1. Open Postman
2. Click "New" → "Collection"
3. Name it "Node.js API Testing"
4. Add description: "Testing CRUD operations for Node.js API"

Step 2: Set Base URL Variable

1. In your collection, go to "Variables" tab
2. Add variable: `baseUrl` = `http://localhost:3000`
3. Save the collection

Test Cases for Each Endpoint

Test 1: Welcome Message

- **Method:** GET
- **URL:** `{{baseUrl}}/`
- **Expected:** 200 OK with welcome message

Test 2: Get All Users

- **Method:** GET
- **URL:** `{{baseUrl}}/users`
- **Expected:** 200 OK with array of users

Test 3: Get User by ID

- **Method:** GET
- **URL:** `{{baseUrl}}/users/1`
- **Expected:** 200 OK with single user data

Test 4: Get Non-existent User

- **Method:** GET
- **URL:** `{{baseUrl}}/users/999`
- **Expected:** 404 Not Found

Test 5: Create New User

- **Method:** POST
- **URL:** `{{baseUrl}}/users`
- **Headers:** `Content-Type: application/json`
- **Body (raw JSON):**

```json
{
  "name": "Test User",
  "email": "test@example.com",
  "age": 25
}
```

- **Expected:** 201 Created

Test 6: Create User with Missing Fields

- **Method:** POST
- **URL:** `{{baseUrl}}/users`
- **Headers:** `Content-Type: application/json`
- **Body (raw JSON):**

```json
{
  "name": "Incomplete User"
}
```

- **Expected:** 400 Bad Request

Test 7: Update User

- **Method:** PUT
- **URL:** `{{baseUrl}}/users/1`
- **Headers:** `Content-Type: application/json`
- **Body (raw JSON):**

```json
{
  "name": "Updated Name",
  "age": 30
}
```

- **Expected:** 200 OK

Test 8: Delete User

- **Method:** DELETE
- **URL:** `{{baseUrl}}/users/1`
- **Expected:** 200 OK

Postman Test Scripts

Add these scripts to validate responses automatically:

For GET requests:

```javascript
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response has success field", function () {
  pm.expect(pm.response.json()).to.have.property('success');
});
```

For POST requests:

```javascript
```

```
pm.test("Status code is 201", function () {
  pm.response.to.have.status(201);
});

pm.test("User created successfully", function () {
  const response = pm.response.json();
  pm.expect(response.success).to.be.true;
  pm.expect(response.data).to.have.property('id');
});
```

🧠 Understanding the Code

Key Concepts Explained

1. Express.js Framework

Express is a minimal web framework for Node.js that simplifies:

- Route handling
- Middleware management
- HTTP request/response handling

2. Middleware

Functions that execute during the request-response cycle:

```javascript
app.use(express.json()); // Parses JSON in request body
app.use(cors()); // Enables cross-origin requests
```

3. Routes

Define how the application responds to client requests:

```javascript
app.get('/users', (req, res) => {
  // Handle GET request to /users
});
```

4. HTTP Status Codes

- **200:** OK - Request successful
- **201:** Created - Resource created successfully

- **400:** Bad Request - Invalid request data
- **404:** Not Found - Resource doesn't exist
- **500:** Internal Server Error - Server error

5. RESTful Design

- **GET:** Retrieve data
- **POST:** Create new data
- **PUT:** Update existing data
- **DELETE:** Remove data

Code Structure

```
📁 Project Root
├── 📄 server.js (Main server file)
├── 📄 package.json (Project configuration)
└── 📁 node_modules/ (Dependencies)
```

🔧 Common Issues & Solutions

Issue 1: Port Already in Use

Error: `EADDRINUSE: address already in use :::3000` **Solution:**

```bash
# Kill process using port 3000
npx kill-port 3000
# Or change port in server.js
const PORT = process.env.PORT || 3001;
```

Issue 2: Cannot POST/PUT Data

Problem: Request body is undefined **Solution:** Ensure you're sending JSON with correct Content-Type header:

- Header: `Content-Type: application/json`
- Body format: Raw JSON

Issue 3: CORS Errors

Problem: Browser blocks requests from different origins **Solution:** CORS middleware is already included. For production, configure specific origins:

```javascript
```

```
app.use(cors({
  origin: 'http://localhost:3001' // Your frontend URL
}));
```

Issue 4: Module Not Found

Error: `Cannot find module 'express'` **Solution:**

```bash
npm install express cors
```

🎯 Next Steps

After mastering this basic server, consider learning:

1. **Database Integration** (MongoDB, PostgreSQL)
2. **Authentication & Authorization** (JWT, Passport.js)
3. **Input Validation** (Joi, express-validator)
4. **Testing** (Jest, Mocha)
5. **Deployment** (Heroku, Vercel, DigitalOcean)
6. **Documentation** (Swagger/OpenAPI)

📞 Support

If you encounter issues:

1. Check the console for error messages
2. Verify all dependencies are installed
3. Ensure the server is running on the correct port
4. Check Postman request format and headers

Happy coding! 🚀