

Web Programming 2 – Week 8 – Huynh Khoi Nguyen

In this Lesson we went through how to implement and use RESTful API with node.js, express and MongoDB. We begin by setting up the project using the node.js console, making sure to install all necessary dependencies like Express, mongoose, and nodemon. Next, to ensure the server is operating on the correct port, we will create a folder called todoListApi, initialize it with npm init -y, configure it with a start script in package.json, then launch the server using npm run start.

```
> todolistapi@1.0.0 start
> nodemon server.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Todo List REST API started on port: 3000
Process PID: 30484
```

The next step is to construct the essential parts of the API by making the following folders: todoListModel.js, todoListController.js, and todoListRoutes.js. We use mongoose to establish the schema for our "task" with the necessary fields, including name, date, and status. Using Mongoose methods like find and save, the file todoListController.js manages CRUD operations including create, read, update, and remove tasks. Next, we configure endpoints like /tasks and /tasks/:taskid in the todoListRoutes.js to connect them to the todoListController.js.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "todoListApi".
 - EXPLORER:** package.json, server.js, routes (todolistRoutes.js), Postcode, todolistModel.js, todolistController.js.
 - TODOLISTAPI:** api (controllers (todolistController.js), models (todolistModel.js), routes (todolistRoutes.js)), node_modules, package-lock.json, package.json, server.js, test.postcode.
- Code Editor (Center):** The file "todolistRoutes.js" is open, showing the following code:

```
package.json server.js todolistRoutes.js Postcode todolistModel.js todolistController.js

// routes
// routes/todolistRoutes.js
// app > routes > todolistRoutes.js > ...
1  'use strict';
2
3  module.exports = function(app) {
4    var todolist = require('../controllers/todolistController');
5
6    app.route('/tasks')
7      .get(todolist.list_all_tasks)
8      .post(todolist.create_a_task);
9
10   app.route('/tasks/:taskId')
11     .get(todolist.read_a_task)
12     .put(todolist.update_a_Task)
13     .delete(todolist.delete_a_Task);
14 };
15 }
```
- Bottom Status Bar:** Shows "Ln 15 Col 1 Spaces: 2 UTF-8 CR/LF (1) JavaScript Go Live Go Live"
- Bottom Taskbar:** Shows icons for File, Edit, Selection, View, Go, Run, Terminal, Help, and various system icons.

We test our API using the Postcode plugin in Visual Studio Code after connecting everything in server.js and establishing a connection to MongoDB Compass via the localhost connection string. To ensure that our data is successfully infused and appears in the database, we sent GET and POST. The fundamentals of comprehending API functionality and client-server communication are emphasized in this lab exercise.

The screenshot shows a Microsoft Edge browser window with the following details:

- Address Bar:** todoListApi
- Tab Bar:** package.json, server.js, todolistRoutes.js, test.postcode, Postcode (highlighted in orange), todolistModel.js, todolistController.js
- Left Sidebar (Explorer):**
 - OPEN EDITORS:** package.json, server.js, todolistRoutes.js, apivroutes, Postcode (highlighted in orange), todolistModel.js, apimodels, todolistController.js, apicontrollers
 - TODOLISTAPI:** api, controllers, todolistController.js, models, todolistModel.js, routes, todolistRoutes.js, node_modules, package-lock.json, package.json, server.js, test.postcode
- Request Preview:** POST http://127.0.0.1:3000/tasks
- Request Headers:** Params, Authorization, Headers (4), Body (selected), Options
- Request Body:** none, form-data, x-www-form-urlencoded, raw, binary, GraphQL
- Table:** A table showing the key-value pairs of the JSON object sent in the body.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> name	khoinguyen	2501
Key	Value	Description
- Response Preview:** Status: 201 Created Duration: 11 ms
- Response Body:**

```
1 {  
2   "name": "khoinguyen",  
3   "status": [  
4     "pending"  
5   ],  
6   "_id": "690c890a5eac775171e21192",  
7   "created_date": "2025-11-06T11:22:50.637Z",  
8   "__v": 0  
9 }
```
- Bottom Icons:** OUTLINE, TIMELINE, Go Live, Go Live

MongoDB Compass - 127.0.0.1:27017/Tododb.tasks

Connections Edit View Collection Help

Compass

My Queries Data Modeling

CONNECTIONS (1)

Search connections

127.0.0.1:27017 Tododb tasks admin config local startup_log testdb users

127.0.0.1:27017 > Tododb > tasks

Documents 1 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1-1 of 1

`_id: ObjectId('690c850a5eac775171e21192')
name : 'khoinguyen'
status : Array (1)
Created_date : 2025-11-06T11:22:50.637+00:00
__v : 0`

The screenshot shows the MongoDB Compass interface. On the left, the connection tree displays '127.0.0.1:27017' with its databases: 'Tododb', 'admin', 'config', 'local', 'testdb', and 'users'. Under 'Tododb', the 'tasks' collection is selected. The main panel shows the 'Documents' tab with one document listed. The document details are: '_id: ObjectId('690c850a5eac775171e21192')', 'name : 'khoinguyen'', 'status : Array (1)', 'Created_date : 2025-11-06T11:22:50.637+00:00', and '__v : 0'. Below the document list are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. At the bottom, there are pagination controls showing '25' items, '1-1 of 1', and navigation arrows.