Web Programming

Tutorial 6

To begin this tutorial, please download `tut06-starter.zip` from the course website.

Activity 1 instructs you how to create that project. When you finish, zip your project's contents (except the `node_modules` folder) to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `clc01`, `clc02`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Activity 1 – Hello Node.js

In folder `/nodejs`, create a NodeJS server and run the Hello World program yourself. Follow the instructions below**:**

- `http.createServer`: Creates an HTTP server that listens to requests.
- `req` and `res`: `req` represents the incoming request, and `res` represents the outgoing response.
- `res.statusCode`: Sets the HTTP status code for the response.
- `res.setHeader`: Sets the HTTP headers for the response.
- `res.end`: Ends the response and sends the data to the client.
- `server.listen`: Starts the server and listens for incoming requests on the specified port.

Run the Server

- Open your **terminal** or **command prompt.**
- Navigate to the nodejs folder where `server.js` is located.
- Run the server using Node.js with the following command: `node server.js`

Activity 2 – Simple RESTful API with Node.js

In `nodejs/nodejs-product-api` folder, You will be provided with the following two files:

1. `products.json`**:** this file contains product data that will be used as the data source for the API.

2. `dataProvider.js`: this file contains `getProducts` function to read data from the JSON file.

You need to create an `api.js` file. In the `api.js` file, students will use the provided `getProducts` function to perform the following tasks:

- **Create a Node.js Server:**

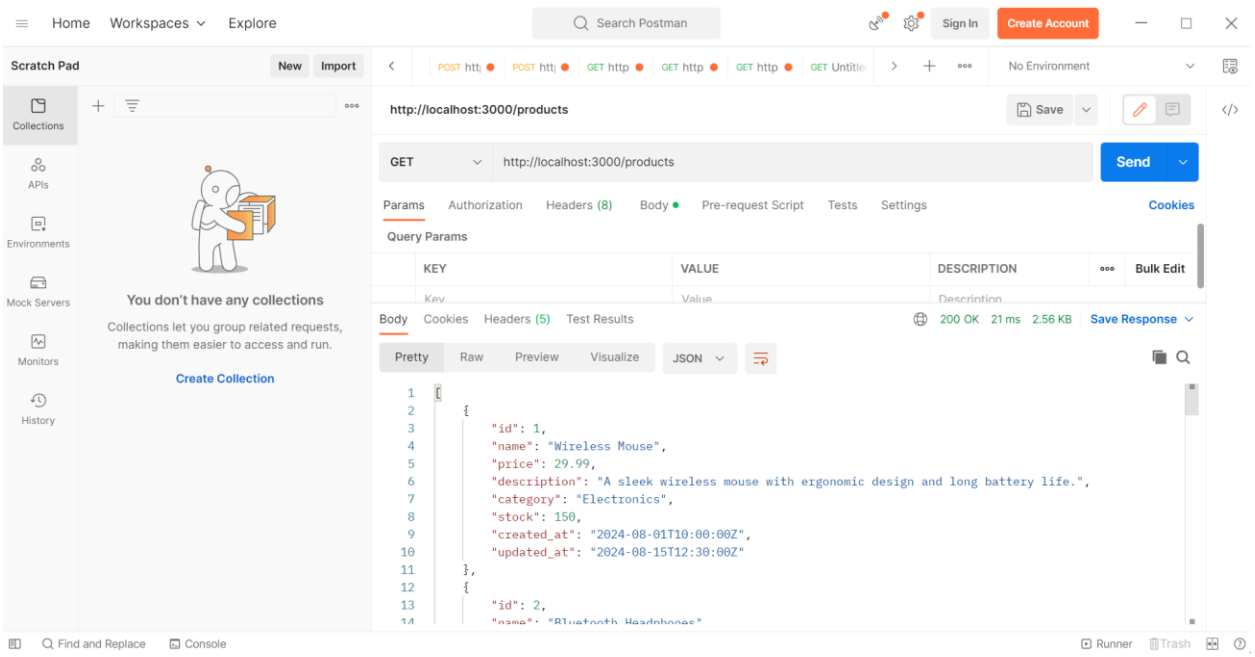
  Use the `http` module to create the server.

- **Write the API endpoints:**
  - Use `HTTP` methods such as `GET`, `POST`, `PUT`, and `DELETE` to handle corresponding requests.
    - `GET /products` : Retrieve a list of all products.
    - `GET /products/:id` : Retrieve information about a specific product by id.
  - Use `JSON.parse()` to handle data from the body of POST and PUT requests.
    - `POST /products` : Add a new product to the list.
    - `PUT /products/:id` : Update information about a product by id.
  - `DELETE /products/:id` : Delete a product by id.
  - Use `res.writeHead` to send appropriate status codes (e.g., **`200 OK`, `404 Not Found`, `201 Created`**).
  - Use `res.end()` to finish the response.

Notes:

You should use a tool like **Postman** or **curl** to send HTTP requests to the server and test the results. Download Postman here.

Example:



Activity 3 – Hello World in Node.js (Express.js)

Inside the `/expressjs` folder, run the command `npm init` to make this folder become a Node.js project. You will need to install any non-core modules using `npm`.

Run the command:

```
npm install <package-name>
```

to do so. At minimum, tutorial will require the `express` package.

Create the file `index.js` in `/expressjs` folder and put the following source code into it:

```
"use strict";
const express = require('express');
const app = express();

app.get('/hello', function (req, res) {
    res.type('text');
    res.send('Hello World!');
});

app.use(express.static('public'));
```

```
app.listen(8000);
```

Also create a folder named `public` inside the project folder and create a static HTML page named `hello.html` inside the `public` folder (put any content in this HTML file). Open a terminal in the directory with the server and enter:

```
node index.js
```

Alternatively, you can run `nodemon` command to start a Node project. `nodemon` is a tool that restarts the server if you make changes to the JS code in order to reflect the changes.

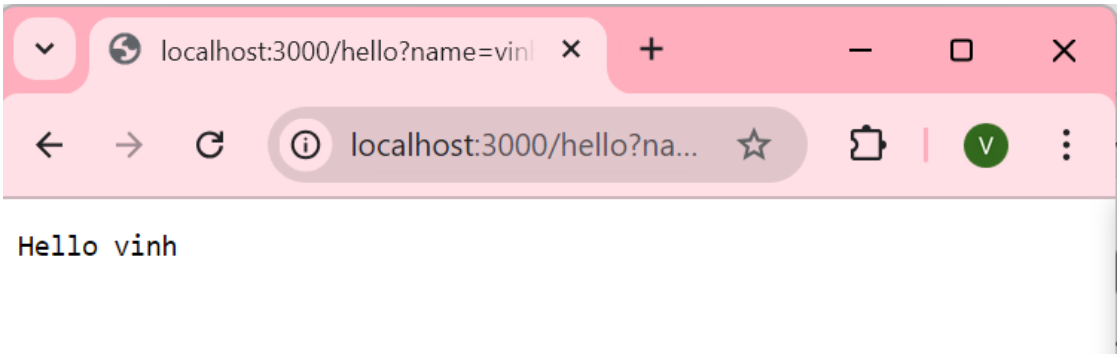
(*) To stop the server, press **Ctrl + C** in the terminal.

Access your page in the browser. Since the server is being hosted locally on your machine, use the URL `http://localhost:8000/hello` (`8000` is the port we specified in `index.js`).

Since we told the server to serve files in the `public` directory, we can access a static file at the url `http://localhost:8000/hello.html`.

Activity 4 – Comparing Node.js and Express.js

Understand the difference between handling requests in plain Node.js and using the Express.js framework by creating a simple route `/hello` that takes a query parameter name and displays `"Hello [name]"` on the result page.



- Using Plain Node.js:
  - Create a simple HTTP server in Node.js (in `/nodejs` folder).
  - Handle requests to the `/hello` path with a query parameter name.

- Extract the name parameter from the URL and display `"Hello [name]"` in the response.
- **No Additional Packages**: For the Node.js part, don't use any external libraries.
  Hint: See URL module https://www.w3schools.com/nodejs/nodejs_url.asp

- Using Express.js:
  - Set up a basic Express.js application (in `/expressjs/act4.js` file).
  - Create a route `/hello` that accepts the query parameter name.
  - Use Express's built-in functionality to easily extract the name parameter and display `"Hello [name]"` in the response.
- After completing both implementations, compare the following in the `comparing_nodejs_expressjs.txt` file:
  - **Code Complexity**: Which method requires more code or is easier to write and maintain?
  - **Readability**: Which code is easier to understand?
  - **Flexibility**: Which approach provides more flexibility or is easier to extend

Activity 5 – Circles

The file `/expressjs/circles.js` has been created for you. Write the code of an Express.js application in this file, then add a new GET endpoint, `/math/circle/:r`, which takes a radius as a URL parameter. It should then respond in JSON with the area and circumference.

```
{"area": 3.14, "circumference": 6.28}
```

The area of a circle is `PI * r * r`, and the circumference is equal to `PI * 2r`. You can access PI with `Math.PI`.

Activity 6 – Rectangles

The file `/expressjs/rectangles.js` has been created for you. In this file, create a GET endpoint, `/math/rectangle/:width/:height`. It should respond in JSON with the *area* and *perimeter* based on the input *width* and *height*.

```
{'area': 25, 'perimeter': 20}
```