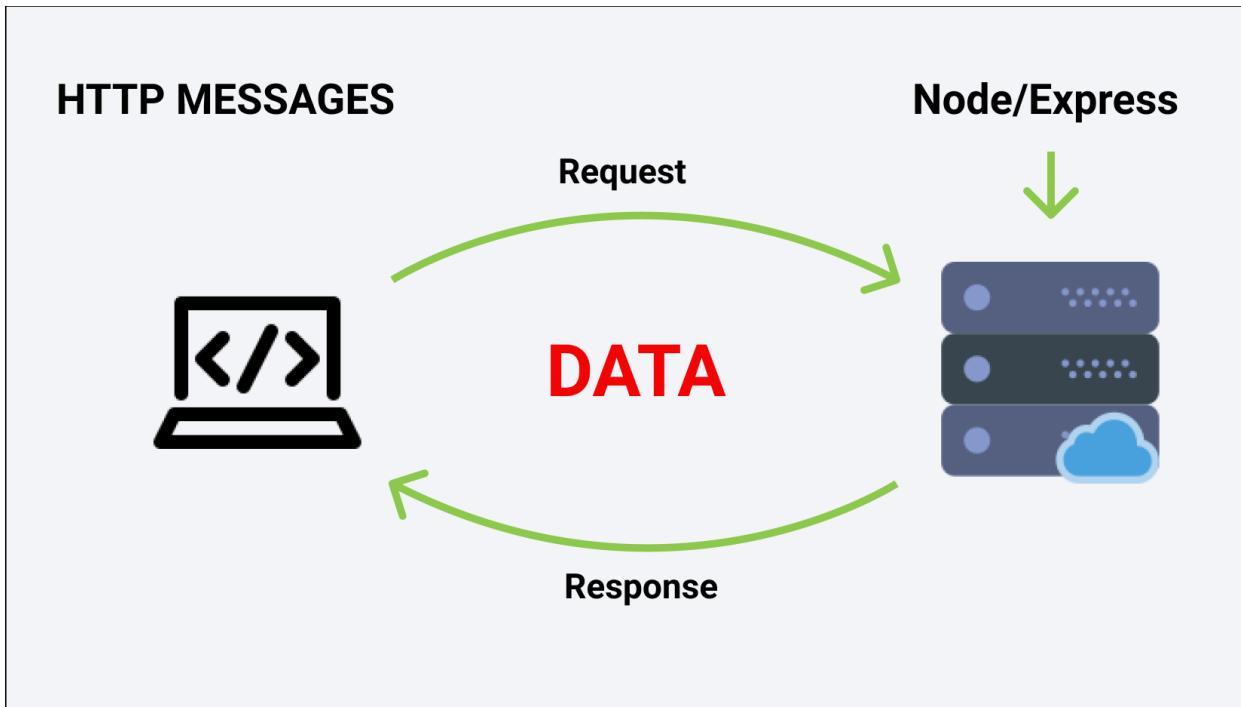


# Express

## HTTP Request/Response Cycle

Every time we open the browser and type the URL, we are performing a request to server that is responsible for serving a response. Now this is done using HTTP protocol and these are called HTTP messages.



User sends a HTTP request message and then server sends an HTTP response message and that's how we exchange data on web.

We mostly use Node but in order make our work easier we use a framework named Express JS. A server job is always to make a resource available. A server doesn't have an GUI (Graphical User Interface). Cloud is nothing but a bunch of servers and computers connected.

## HTTP Messages

Let's see how HTTP messages are structured.



General structure for both messages (request and response) is similar.

They both have a start line, they both have optional headers, a blank line that indicates that all meta info has been sent and effectively headers are that meta info as well as optional body.

Request Messages – messages sent by a user.

Response Messages – messages sent by a server.

In General, when we talk about **request message** in start line there's going to be a method, then URL and then HTTP version as well.

**Methods** is the place where we communicate what we want to do.

Ex: If we want to get the resource then we set it up as GET request.

If we want to add the resource, then we set it up as POST request.

**GET request is the default request that the browser performs (since we open the browser and get some request from web, hence GET is the default request).**

**URL** is just the address. (Ex: freecodecamp.org)

**Headers** is essentially optional; it is meta information about our request. Headers have a key-value pair. We don't need to add headers manually but in few cases we need to add headers. (Basically, it an information about our message).

**Body**, if we just need the data from resource then there is no body but if we want to add a resource to the server then we are expected to provide a body and that is called request payload.

When we talk about response message, the Node JS developers will be creating the response. Start line has the HTTP version, then we have a status code and status text.

**HTTP version** – it is mostly going to be 1.1

**Status Code**, it just signals what is the result of the request.

Ex: Status Code: 200 – Request was successful.

Status Code: 400 – There was an error in the request.

Status Code: 404 – Resource was not found.

**Headers**, we provide info about our message. (It is a setup of key value pairs).

Content-Type: text/html; we are sending back the html.

Content-Type: application/json: we are sending back the data.

When we communicate with API, mostly we are getting back the JSON data because over the web effectively we just send over the string.

In our headers we indicate that we are sending the data in application/json and then that application (web application) which is requesting knows that they are receiving application/json from the server.

## **Starter Project Install**

Clone projects from <https://github.com/john-smilga/node-express-course>

## **Starter Overview**

Express is built on top of Node and specifically built on HTTP module.

Follow the steps below to setup a express-tutorial project.

1. Create a new folder 02-express-tutorial
2. Run command npm init -y // to create a package.json file
3. Run command npm install // to install node modules
4. Run command npm install express // to install express package into our project
5. Run command npm install --save-dev nodemon // to install nodemon as a dev dependency package
6. Create a .gitignore file and add node modules // this will make sure we don't push node modules to github.
7. Create app.js file and write some console.log statement
8. Edit the scripts object in package.json file and write the key start with value of nodemon app.js

## HTTP Basics

In HTTP Protocol, we transfer data over the web using HTTP Request message and HTTP response message. We create a server using HTTP module and start the server. When we start the server, we try to listen to the requests by using a port number.

**Port is a communication endpoint.**

There are lot of port numbers,

Port Number 20: Used for File Transfer Protocol (FTP) Data Transfer.

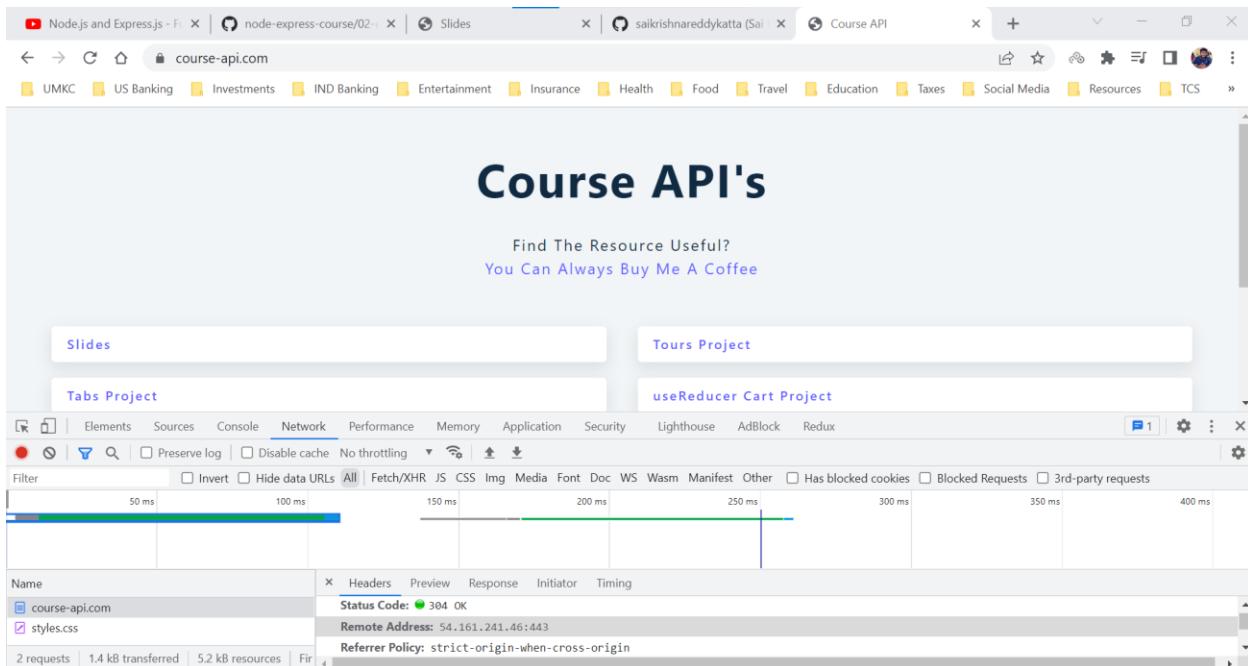
Port Number 21: Used for File Transfer Protocol (FTP) Command Control.

Port Number 80: Used for Hyper Text Transfer Protocol (HTTP) used in the world wide web.

Port Number 443: HTTP Secure (HTTPS) HTTP over TLS/SSL.

As of now in development phase we are using port number 5000 but once in production, we may use 80 or 443.

For course-api, we can see the port number (443) in Remote Address field which also contains the IP address.



While in development we can use any port number, but 0 – 1024 port numbers are already taken.

**Ex: React uses Port Number 3000, Gatsby uses Port Number 8000, Netlify CLI uses Port Number 8080.**

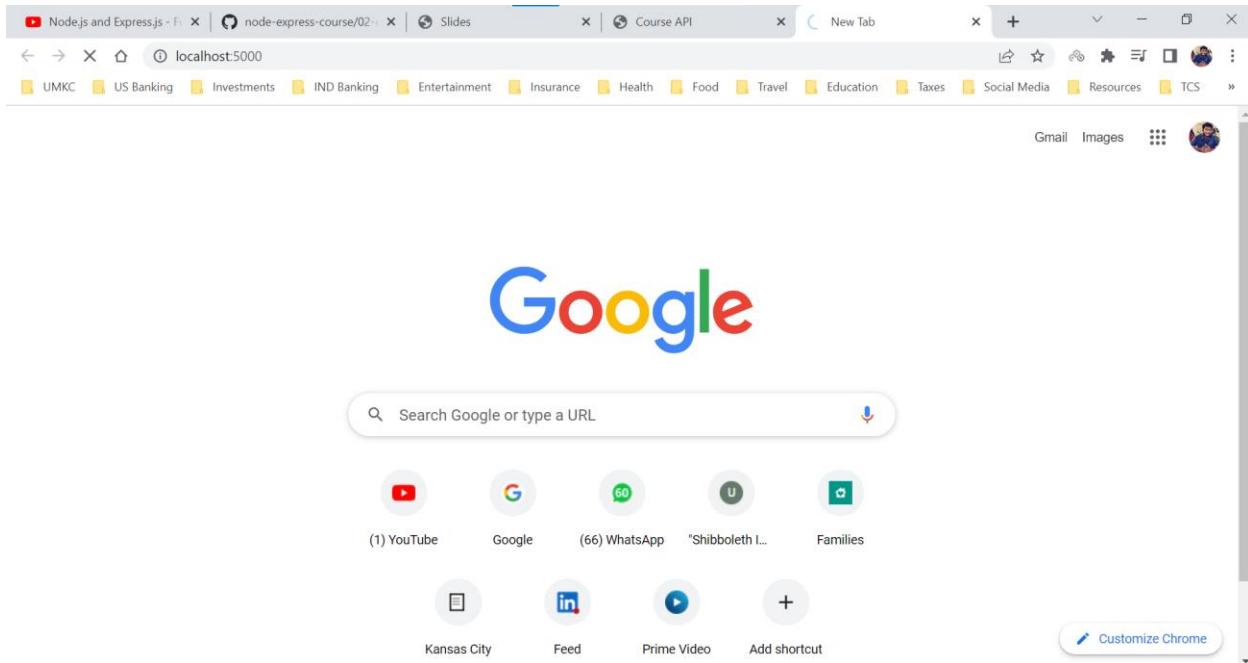
When we don't send any response to the server. We are just console logging the information.

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders, including 'app.js' which is currently selected. The terminal at the bottom shows the following output:

```
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node app.js
User hit the server
User hit the server
User hit the server
```

The status bar at the bottom indicates the code is in JavaScript mode.

Server is waiting for the response, and it is still loading.



**response.end()** – this method signals server that all the response headers and body have been sent, that server should consider this message complete. This method `response.end()` must be called on each response.

`createServer()` method contains a callback which is invoked every time user hits the server and as parameters to the callback function, we have request and response objects.

## app.js

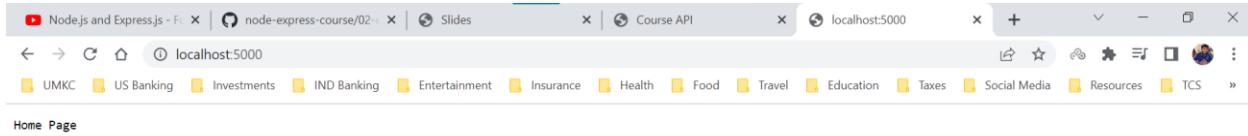
The screenshot shows the Visual Studio Code interface. The left sidebar displays a file tree with files like `app.js`, `package-lock.json`, and `package.json`. The main editor window shows the following code:

```
const http = require("http");
http.createServer((req, res) => {
  console.log("User hit the server");
  res.end("Home Page");
}).listen(5000);
```

The terminal at the bottom shows the output of the application running with Nodemon:

```
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
User hit the server
User hit the server
User hit the server
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
User hit the server
User hit the server
```

and on `localhost:5000`



## HTTP Headers

We have two major issues with our current setup.

1. We don't send any information about the data that we are sending back. (We are sending any metadata about the response body we are sending back). As of now we are just sending back a string.
2. We are not sending data based on the request by user. Whatever may be the request we are sending only one response which is a string.

We use `response.writeHead()` method to write headers/ provide meta data to browser about the response we are sending back to the browser. `response.writeHead()` contains a status code and a headers object (which contains properties like content-type etc.) Browser renders the content of page based on property content-type. We also can add the status text (which is optional).

We use `response.write()` to send the response body to browser.

We use `response.send()` to tell the browser that the message is complete, and we have sent all the required data.

There are many status codes, and it is important to send the correct status code back to the browser.

- 100 – 199 -> Informational Responses
- 200 – 299 -> Successful Responses
- 300 – 399 -> Redirection Messages
- 400 – 499 -> Client Error Responses
- 500 – 599 -> Server Error Responses

**MIME types** – A media type also known as ***Multipurpose Internet Mail Extensions*** indicates the nature and format of a document, file, or assortment of bytes.

A MIME type most-commonly consists of just two parts: a type and a subtype, separated by a slash (/) — with no whitespace between.

Ex: type/subtype.

An optional parameter can be added to provide additional details.

Ex: type/subtype; parameter = value

We use the MIME types to declare the type of data we are sending back to the browser. Types of MIME are

- application/octet-stream – This is default for binary files.
- text/plain - This is the default for textual files. Even if it really means "unknown textual file," browsers assume they can display it.
- text/css - CSS files used to style a Web page must be sent with text/css.
- text/html - All HTML content should be served with this type.
- text/javascript - Per the current relevant standards, JavaScript content should always be served using the MIME type text/javascript.

***Express will take care of Headers, but we are learning here for Node JS.***

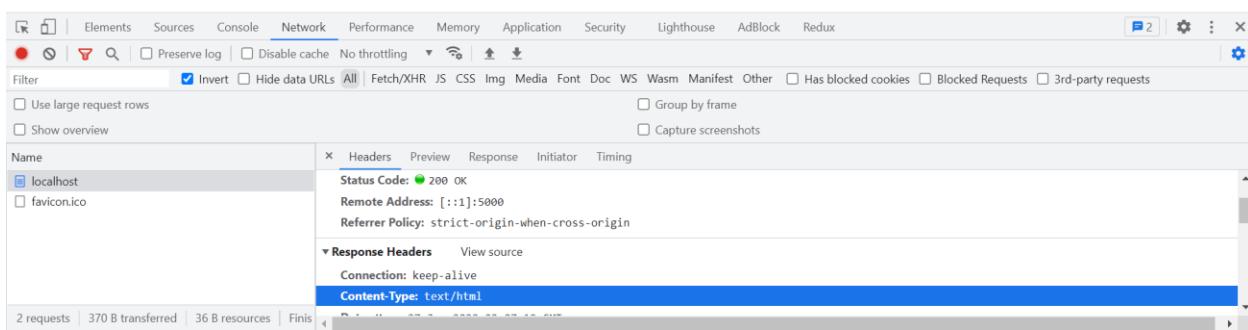
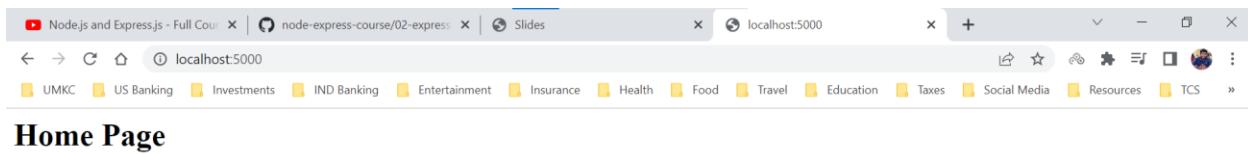
## app.js

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders: 01-node-tutorial, 02-express-tutorial, .gitignore, app.js (which is selected), package-lock.json, package.json, Express-JS-Notes.pdf, Node-JS-Notes.pdf, and README.md. The Terminal tab at the bottom shows the command-line output:

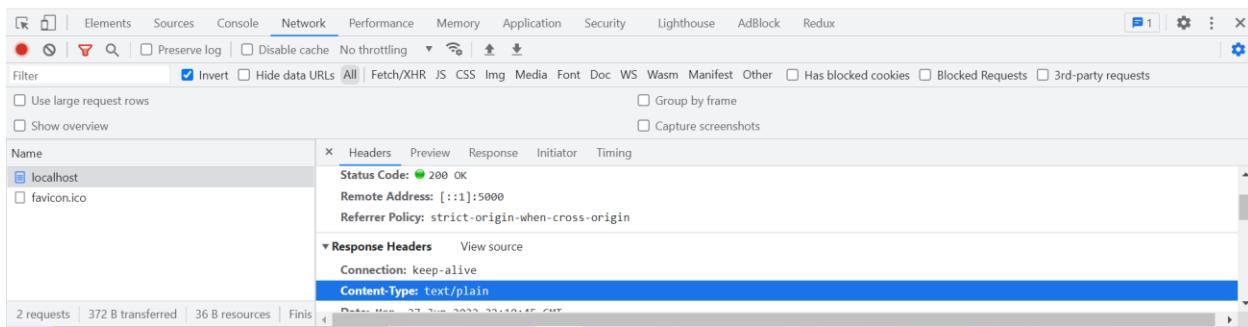
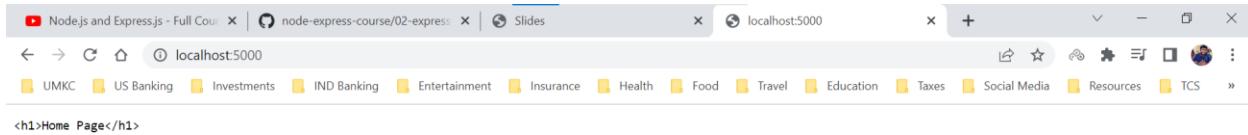
```
User hit the server
User hit the server
User hit the server
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
```

The status bar at the bottom indicates: Line 8, Col 5, Spaces 4, UTF-8, CRLF, JavaScript, Prettier.

When Content-Type is text/html and Status code is 200



When Content-Type is text/plain and Status code is 200



Browser interprets the type of format and then renders the screen.

## HTTP Request Object

Now, let us deal with the request object.

In the request object, we receive HTTP method, URL, HTTP Version, Headers, and Body (which is optional). Since we receive request body from the browser, we need to extract the properties and then send response to browser based on the request by the user.

**request.method** – it is one of the properties which provides information about the HTTP method.

**request.url** – it is one of the properties which provide information about the URL.

Forward Slash would be the Home Page ("/").

If we want to Contact page resource, then URL can be ("/contact").

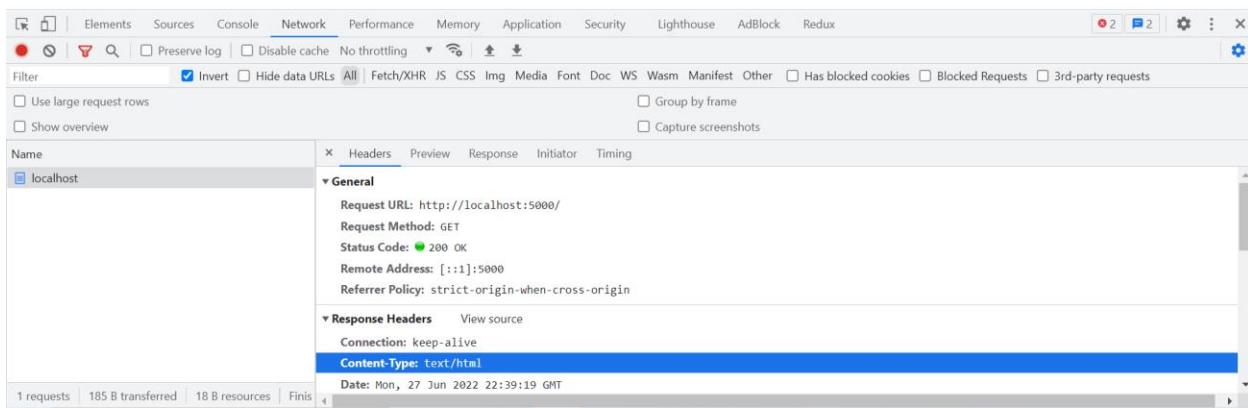
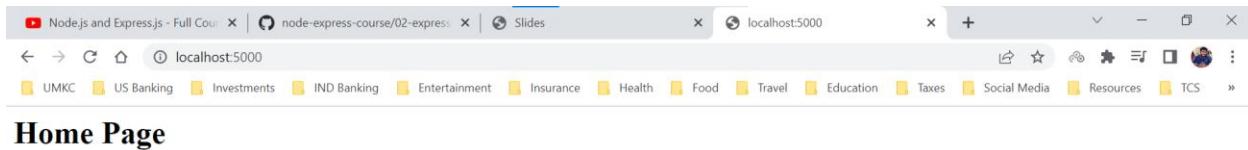
## app.js

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

- File Explorer:** Shows the project structure under "NODE JS TUTORIAL YOUTUBE".
- Code Editor:** The active file is `app.js`, containing Node.js code for a simple Express application. It handles requests for "/" and "/contact".
- Terminal:** Shows the command `[nodemon] starting 'node app.js'`.
- Status Bar:** Displays file statistics: Line 21, Col 44, Spaces 4, UTF-8, CRLF, JavaScript, Prettier.

```
const http = require("http");
http.createServer((req, res) => {
  // console.log(req);
  // console.log(req.method);
  // console.log(req.url);
  const url = req.url;
  if (url === "/") {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write("<h1>Home Page</h1>");
    res.end();
  } else if (url === "/contact") {
    //URL is case-sensitive,
    //if we try with Contact instead of contact, it will redirect to Page Not Found
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write("<h1>Contact Page</h1>");
    res.end();
  } else {
    res.writeHead(404, { "Content-Type": "text/html" });
    res.write("<h1>Page Not Found</h1>");
    res.end();
  }
}).listen(5000);
```

## Home Page



## Contact Page

The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Cou...", "node-express-course/02-express", and "localhost:5000/contact". The main content area displays the text "Contact Page". Below the browser is the developer tools Network tab. The request for "contact" has been selected. The General section shows the request URL is "http://localhost:5000/contact", the method is "GET", the status code is 200 OK, and the remote address is "[::1]:5000". The Response Headers section shows "Content-Type: text/html" highlighted in blue. The Date header is listed as "Mon, 27 Jun 2022 22:41:32 GMT".

Information Page, we are not holding the resource for Information Page.

The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Cou...", "node-express-course/02-express", and "localhost:5000/information". The main content area displays the text "Page Not Found". Below the browser is the developer tools Network tab. The request for "information" has been selected. The General section shows the request URL is "http://localhost:5000/information", the method is "GET", and the status code is 404 Not Found. The remote address is "[::1]:5000". The Response Headers section shows "Content-Type: text/html" and the Date header is listed as "Mon, 27 Jun 2022 22:42:22 GMT".

## HTTP – HTML File

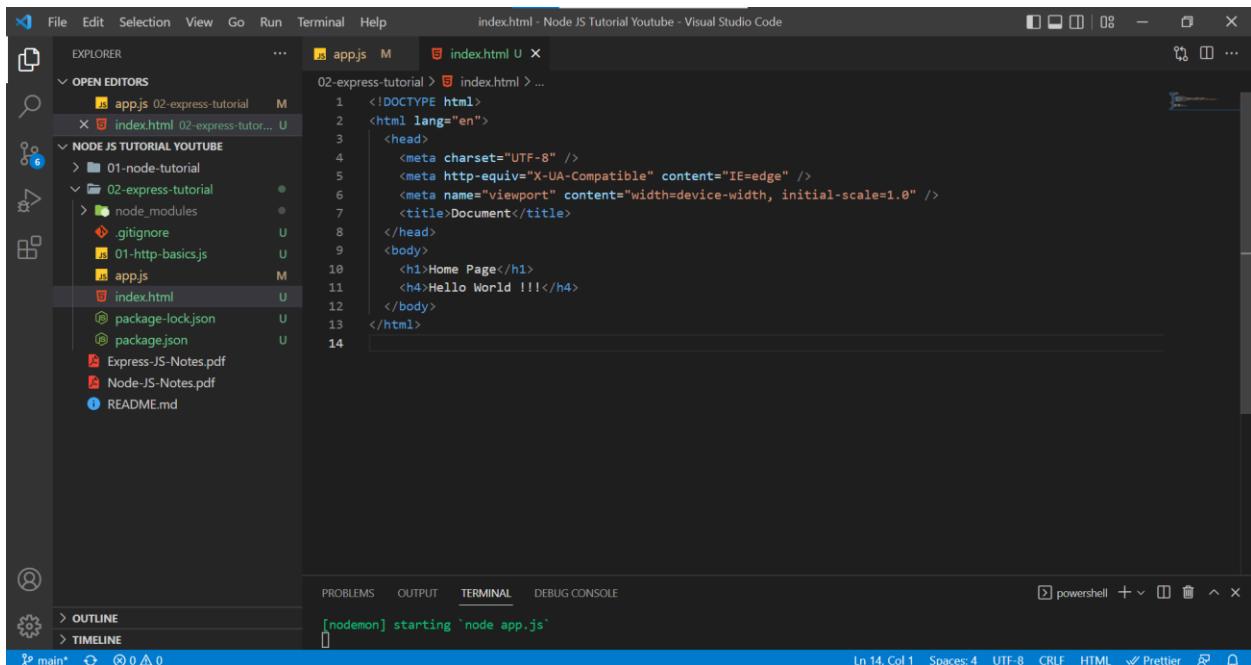
We are not limited to send the HTML directly into `response.write()` method or `response.end()` method. Instead, we can set up a file, request the file using **File System** and just passing it.

Remember that we are passing in the contents of the file not the entire file.

The reason we are using `readFileSync` is

1. We are not invoking the `readFileSync` every time when someone hits the server. We require that file when we instantiate the server (basically the initial time when the server starts running). We are just requesting it only once.
2. It is just an example as of now.

index.html



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
</head>
<body>
    <h1>Home Page</h1>
    <h4>Hello World !!!</h4>
</body>
</html>
```

## app.js

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

- File Explorer:** Shows a project structure under "02-express-tutorial". The "app.js" file is selected.
- Code Editor:** Displays the code for "app.js":

```
1 const http = require("http");
2 const { readFileSync } = require("fs");
3 const homepage = readFileSync("./index.html");
4 http
5 .createServer((req, res) => {
6   const url = req.url;
7   if (url === "/") {
8     res.writeHead(200, { "Content-Type": "text/html" });
9     res.write(homepage);
10    res.end();
11  } else if (url === "/contact") {
12    res.writeHead(200, { "Content-Type": "text/html" });
13    res.write("<h1>Contact Page</h1>");
14    res.end();
15  } else {
16    res.writeHead(404, { "Content-Type": "text/html" });
17    res.write("<h1>Page Not Found</h1>");
18    res.end();
19  }
20 })
21 .listen(5000);
```

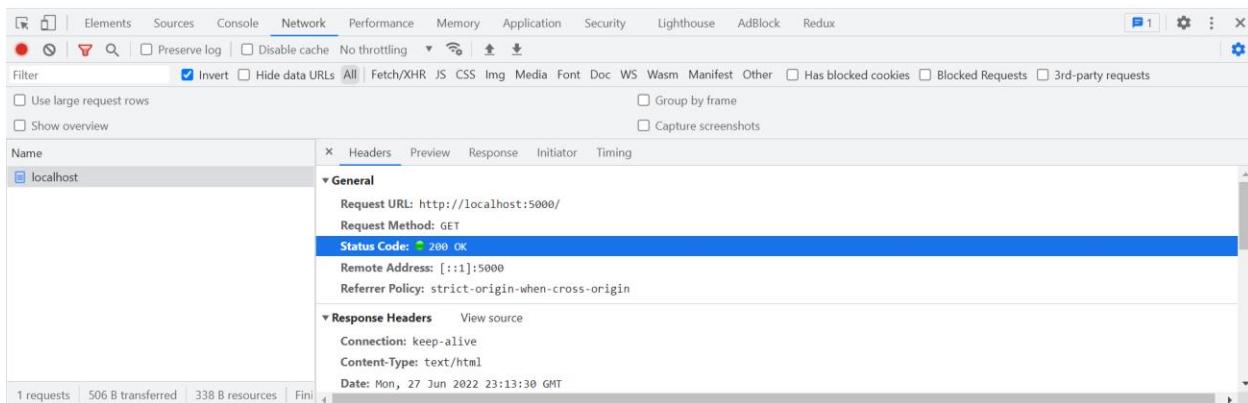
- Terminal:** Shows the command: [nodemon] starting 'node app.js'
- Status Bar:** Shows "Ln 22, Col 1" and other settings like "Spaces 4", "UTF-8", "CRLF", "JavaScript", "Prettier".

Home Page when Content Type is text/html.



## Home Page

Hello World !!!



## Home Page when Content Type is text/plain.

The screenshot shows a browser window with several tabs open. The active tab is 'localhost:5000'. Below the tabs is a navigation bar with various icons. Underneath the navigation bar is a horizontal menu with items like 'UMKC', 'US Banking', 'Investments', etc. The main content area displays the source code of an HTML file:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Document</title>
</head>
<body>
<h1>Home Page</h1>
<h4>Hello World !!!</h4>
</body>
</html>
```

Below the source code is the browser's developer tools Network tab. It shows a single request to 'localhost'. The 'Headers' section of the response details is expanded, showing the following headers:

- Status Code: 200 OK
- Remote Address: [::]:5000
- Referrer Policy: strict-origin-when-cross-origin
- Content-Type: text/plain
- Connection: keep-alive
- Date: Mon, 27 Jun 2022 23:14:26 GMT

At the bottom of the Network tab, there are statistics: 1 requests, 507 B transferred, 338 B resources, and a 'Final' button.

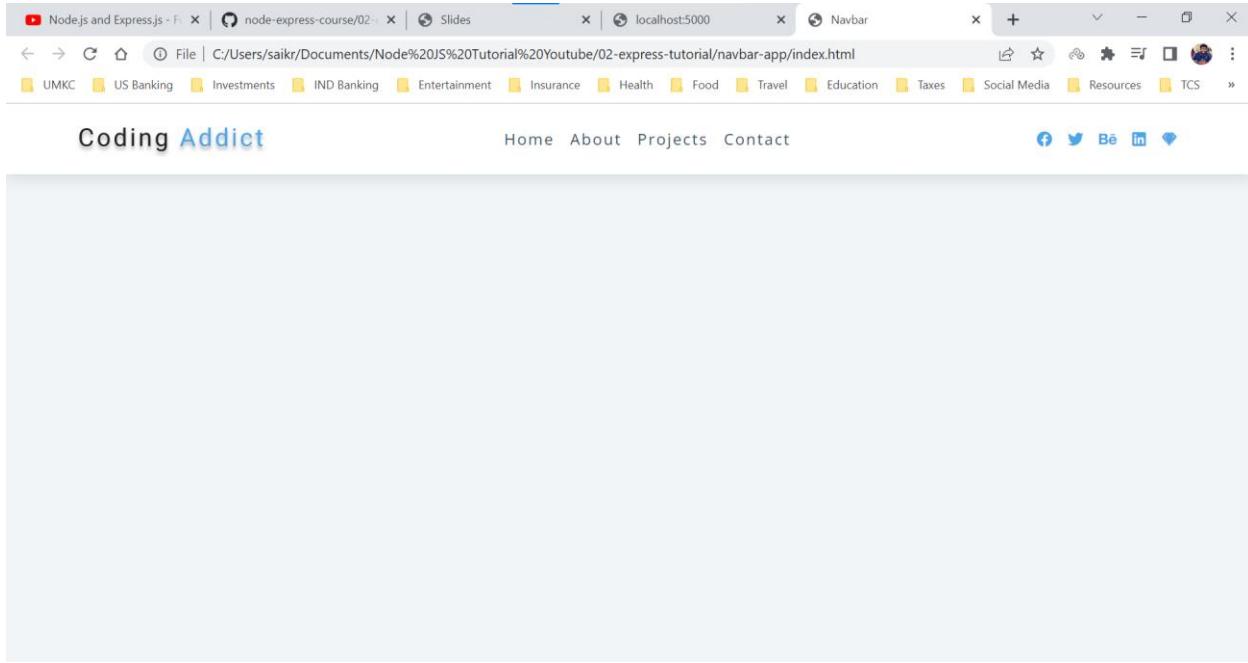
## HTTP – App Example

In this section, we already have a web application named navbar-app which contains a JS file, CSS file, SVG file and an HTML file.

We try to run our server and connect to this application and provide responses based on the request.

We are changing the index.html file path since the web application already has an index.html file.

## Web Application



## app.js

The screenshot shows the Visual Studio Code interface with the file 'app.js' open in the editor. The code implements an Express.js application that handles requests for the root ('/') and contact pages. It uses the 'http' module to create a server and the 'fs' module to read files. The code includes logic to set the content type to 'text/html' and write the respective HTML content for each page. The terminal at the bottom shows the output of the nodemon command, indicating it's watching for changes and restarting the node app.js process.

```
const http = require("http");
const fs = require("fs");
const homePage = fs.readFileSync("./navbar-app/index.html");
const http = require("http");

http.createServer((req, res) => {
  const url = req.url;
  if (url === "/") {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write(homePage);
    res.end();
  } else if (url === "/contact") {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write("<h1>Contact Page</h1>");
    res.end();
  } else {
    res.writeHead(404, { "Content-Type": "text/html" });
    res.write("<h1>Page Not Found</h1>");
    res.end();
  }
}).listen(5000);
```

localhost:5000

The screenshot shows a browser window with a navigation bar containing links for home, about, projects, contact, and social media icons for Facebook, Twitter, Be, LinkedIn, and WhatsApp. Below the browser is a screenshot of the Network tab in the developer tools, showing a list of requests. The requests include:

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	document	Other	2.2 kB	18 ms	
all.min.css	200	stylesheet	(index)	11.1 kB	125 ms	
styles.css	404	stylesheet	(index)	163 B	66 ms	
browser-app.js	404	script	(index)	163 B	24 ms	
logo.svg	404	text/html	(index)	23 B	25 ms	
fa-brands-400.woff2	200	font	all.min.css	78.2 kB	187 ms	
fa-solid-900.woff2	200	font	all.min.css	80.9 kB	203 ms	

7 requests | 173 kB transferred | 219 kB resources | Finish: 402 ms

The reason why the page doesn't have the same outlook after connecting to the server.

Web Application is requesting resources like (index.html, styles.css, logo.svg, browser-app.js) but as of now we are serving the index.html request, hence other requests are receiving 404 error response. We are not handling those requests (styles.css, logo.svg, browser-app.js) in our server.

We can console log the requests by the web application

```
const http = require("http");
const { readFileSync } = require("fs");
const homepage = readFileSync("./navbar-app/index.html");
const url = req.url;
console.log(req.url);
if (url === "/") {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.write(homepage);
  res.end();
} else if (url === "/contact") {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.write("<h1>Contact Page</h1>");
  res.end();
} else {
  res.writeHead(404, { "Content-Type": "text/html" });
  res.write("<h1>Page Not Found</h1>");
  res.end();
}
res.end();
http.createServer((req, res) => [
  .listen(5000);
```

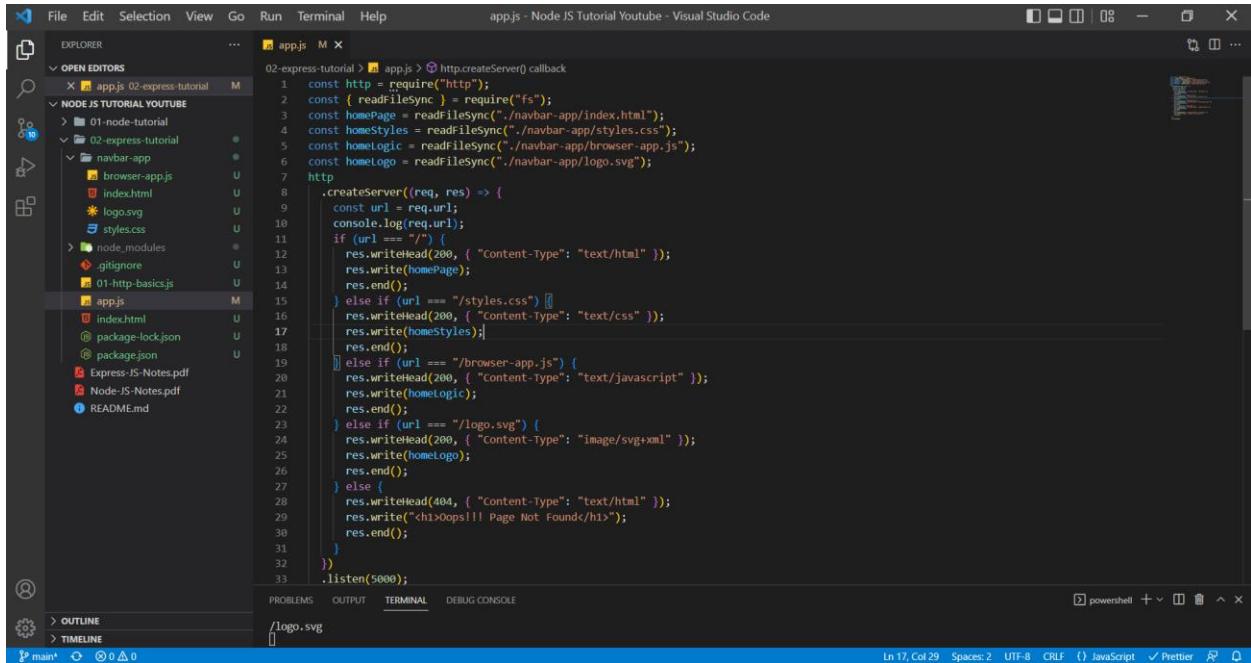
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

/  
/styles.css  
/browser-app.js  
/logo.svg

Ln 7, Col 26 Spaces: 2 UTF-8 CRLF () JavaScript ✓ Prettier

We need to handle those requests and provide responses.

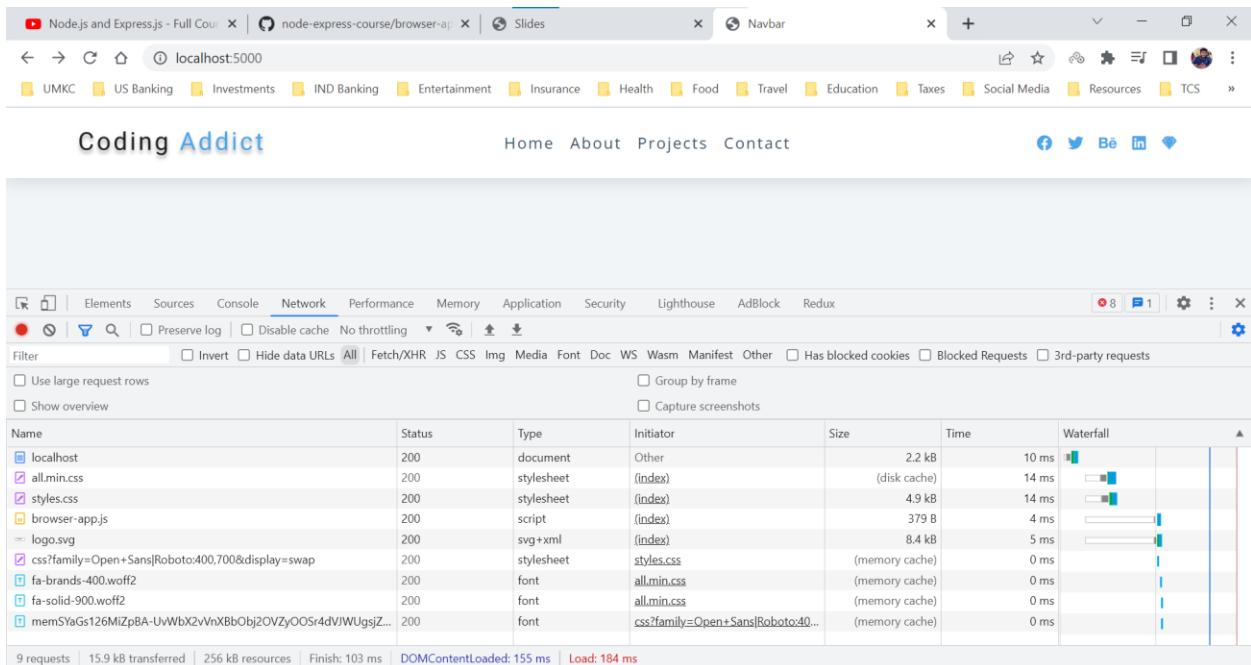
## app.js



```
const http = require("http");
const { readFileSync } = require("fs");
const homepage = readFileSync("./navbar-app/index.html");
const homestyles = readFileSync("./navbar-app/styles.css");
const homelogic = readFileSync("./navbar-app/browser-app.js");
const homologo = readFileSync("./navbar-app/logo.svg");

http.createServer((req, res) => {
  const url = req.url;
  console.log(req.url);
  if (url === "/") {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write(homepage);
    res.end();
  } else if (url === "/styles.css") {
    res.writeHead(200, { "Content-Type": "text/css" });
    res.write(homestyles);
    res.end();
  } else if (url === "/browser-app.js") {
    res.writeHead(200, { "Content-Type": "text/javascript" });
    res.write(homelogic);
    res.end();
  } else if (url === "/logo.svg") {
    res.writeHead(200, { "Content-Type": "image/svg+xml" });
    res.write(homologo);
    res.end();
  } else {
    res.writeHead(404, { "Content-Type": "text/html" });
    res.write("Whoops!! Page Not Found</h1>");
    res.end();
  }
}).listen(5000);
```

We are serving all the requests and let us see the response in browser.



The screenshot shows a browser window with the URL `localhost:5000`. The page content includes a navigation bar with links like Home, About, Projects, and Contact, along with social media sharing icons. Below the browser window is the Network tab of the developer tools. The table lists the following network requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	document	Other	2.2 kB	10 ms	
all.min.css	200	stylesheet	(index)	(disk cache)	14 ms	
styles.css	200	stylesheet	(index)	4.9 kB	14 ms	
browser-app.js	200	script	(index)	379 B	4 ms	
logo.svg	200	svg+xml	(index)	8.4 kB	5 ms	
css?family=Open+Sans Roboto:400,700&display=swap	200	stylesheet	styles.css	(memory cache)	0 ms	
fa-brands-400.woff2	200	font	all.min.css	(memory cache)	0 ms	
fa-solid-900.woff2	200	font	all.min.css	(memory cache)	0 ms	
memSYaGs126MiZpBA-UvWbX2vNxBbObjjOVZyOOSr4dVjWUgsjZ...	200	font	css?family=Open+Sans Roboto:40...	(memory cache)	0 ms	

At the bottom of the developer tools, the stats show: 9 requests | 15.9 kB transferred | 256 kB resources | Finish: 103 ms | DOMContentLoaded: 155 ms | Load: 184 ms

If we try to request a resource which isn't available, then below would be the response.

The screenshot shows a browser window with multiple tabs. The active tab is 'localhost:5000/info', displaying the error message 'Oops!!! Page Not Found'. Below the browser is the Network tab of the developer tools, which lists a single request: 'info' with a status of '404' and type 'document'. The initiator is 'Other'. The size is 205 B and the time is 8 ms. The Waterfall section shows a single vertical bar. At the bottom of the developer tools, it says '1 requests | 205 B transferred | 31 B resources | Finish: 8 ms | DOMContentLoaded: 142 ms | Load: 178 ms'.

## Express Info

We can setup our server just with HTTP module but imagine a scenario where we have a website with tons of resources and then we need to setup for every single resource.

Express JS is a minimal and flexible Node JS Web App Framework designed to develop websites, web apps and API's much faster and easier.

Express is not one of the built-in modules of Node. Express is a standard when creating web applications with Node JS.

Command to install Express JS

***npm install express - -save***

Express JS team suggests using the - -save flag and effectively the reason is because in the earlier Node versions if you didn't add this flag then package wasn't saved to the package.json file meaning whenever we push the code without - -save flag then when another person is using the project, they didn't have reference to the project. Currently that issue is fixed but it still a precaution to use the save flag.

Command to install Express JS with a specific version

***npm install express@4.17.1 - -save***

## Express Basics

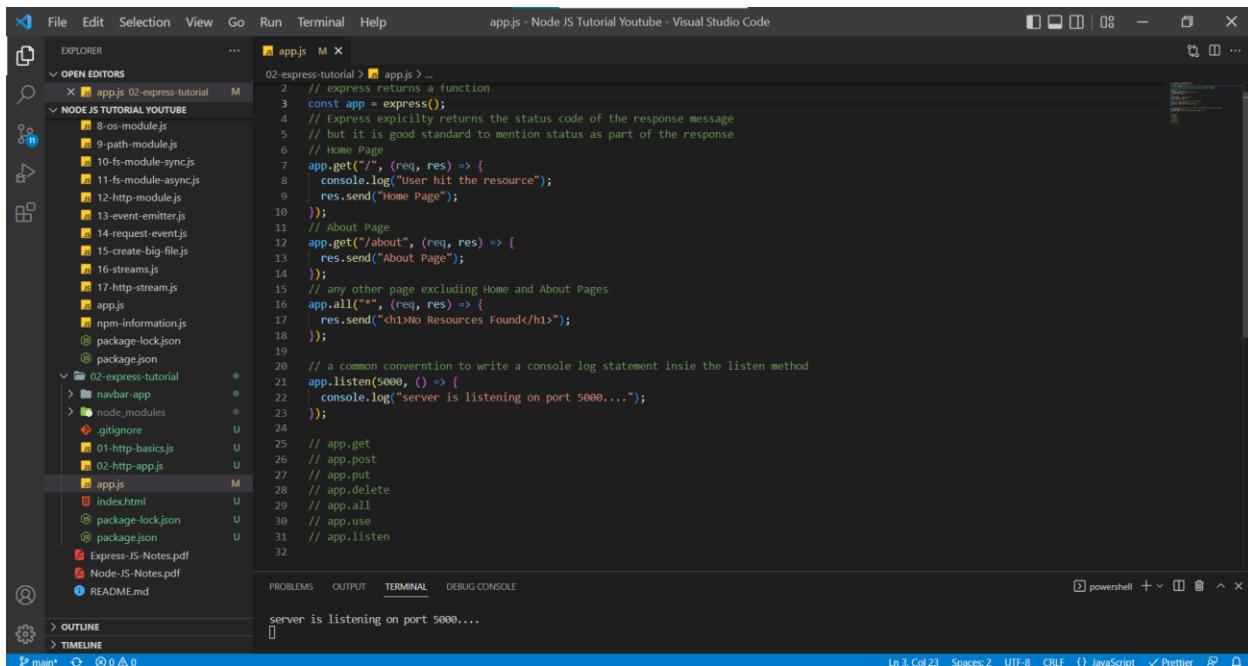
HTTP methods are also known as HTTP verbs, and it is the important part of the HTTP request message that we look for.

HTTP Methods represent what the user is trying to do where to read the data, insert the data, update the data, or delete the data.

By default, all the browsers perform the GET request.

HTTP METHODS		
<b>GET</b>	Read Data	
<b>POST</b>	Insert Data	
<b>PUT</b>	Update Data	
<b>DELETE</b>	Delete Data	
<b>GET</b>	<a href="http://www.store.com/api/orders">www.store.com/api/orders</a>	get all orders
<b>POST</b>	<a href="http://www.store.com/api/orders">www.store.com/api/orders</a>	place an order (send data)
<b>GET</b>	<a href="http://www.store.com/api/orders/:id">www.store.com/api/orders/:id</a>	get single order (path params)
<b>PUT</b>	<a href="http://www.store.com/api/orders/:id">www.store.com/api/orders/:id</a>	update specific order (params + send data)
<b>DELETE</b>	<a href="http://www.store.com/api/orders/:id">www.store.com/api/orders/:id</a>	delete order (path params)

## app.js



```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
02-express-tutorial
  8-os-module.js
  9-path-module.js
  10-fs-module-sync.js
  11-fs-module-async.js
  12-http-module.js
  13-event-emitter.js
  14-request-event.js
  15-create-big-file.js
  16-streams.js
  17-http-stream.js
  app.js
  npm-information.js
  package-lock.json
  package.json
  02-express-tutorial
    navbar-app
    node_modules
      .gitignore
      01-http-basics.js
      02-http-app.js
    app.js
    index.html
    package-lock.json
    package.json
    Express-JS-Notes.pdf
    Node-JS-Notes.pdf
    README.md

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
server is listening on port 5000...
Ln 3, Col 23  Spaces: 2  UTF-8  CRLF  { JavaScript  ✓ Prettier  🔍  ⌂
```

**app.all** – this method works with all of them. It is used handle to requests which doesn't have any resources.

**app.use** – this method is responsible for middleware, and it is a crucial part of Express.

**app.get** – In this method, we need to specially add two things. A path (what resource user is trying to request) and a callback function and this callback function will be invoked every time a user is performing a get request on our route or on our domain.

**app.listen** – In this method our server gets started and server listens to requests sent by the browser.

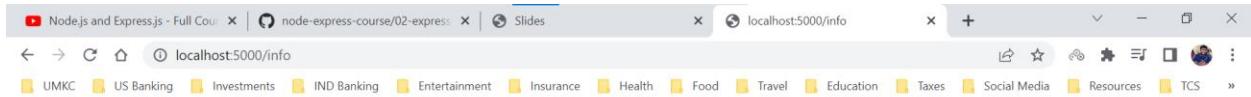
## Home Page

The screenshot shows a browser window with several tabs open. The active tab is 'localhost:5000'. Below the tabs is a navigation bar with links like 'UMKC', 'US Banking', 'Investments', etc. The main content area displays the text 'Home Page'. At the bottom, there is a developer tools Network tab showing a single request to 'localhost' with a status code of 304 OK.

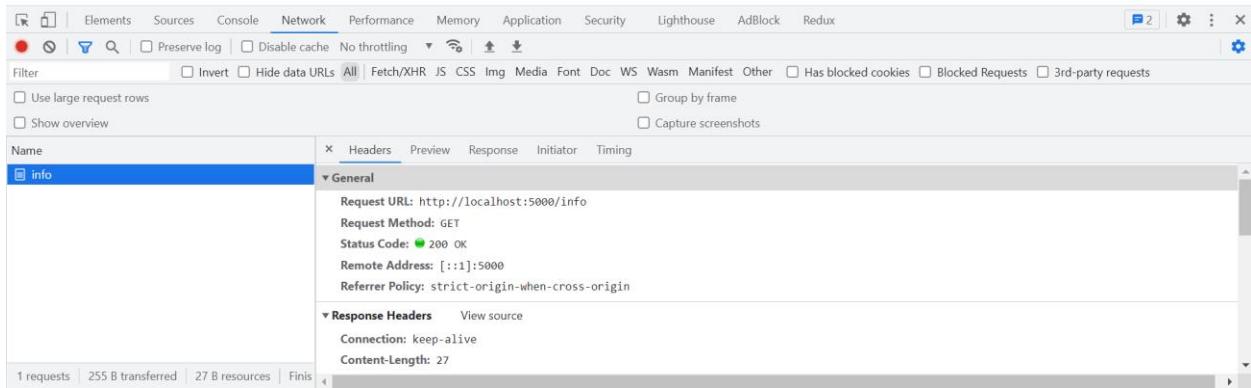
## About Page

The screenshot shows a browser window with several tabs open. The active tab is 'localhost:5000/about'. Below the tabs is a navigation bar with links like 'UMKC', 'US Banking', 'Investments', etc. The main content area displays the text 'About Page'. At the bottom, there is a developer tools Network tab showing a request to 'about' with a status code of 200 OK.

## Resources which are not handled



## No Resources Found



One thing which doesn't add up here is the Status Code. Hence, we need to provide a status code whenever we are sending a response to the browser.

## app.js

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS app.js 02-express-tutorial
NODE JS TUTORIAL YOUTUBE
  8-os-module.js
  9-path-module.js
  10-fs-module-sync.js
  11-fs-module-async.js
  12-http-module.js
  13-event-emitter.js
  14-request-event.js
  15-create-big-file.js
  16-streams.js
  17-http-stream.js
  app.js
  npm-information.js
  package-lock.json
  package.json
  02-express-tutorial
    navbar-app
    node_modules
      .gitignore
      01-http-basics.js
      02-http-app.js
    app.js
    index.html
    package-lock.json
    package.json
    Express-JS-Notes.pdf
    Node-JS-Notes.pdf
    README.md
  OUTLINE
  TIMELINE
  main* powershell + Col 14 Spaces: 2 UTF-8 CR LF () JavaScript ✓ Prettier
  [nodemon] starting 'node app.js'
  server is listening on port 5000...
```

The code in app.js handles two routes: '/' and '/about'. It logs a message for each hit and returns a 200 OK response for the home page. For other pages, it returns a 404 status with the message 'No Resources Found'.

```
const express = require("express");
const app = express();
app.get("/", (req, res) => {
  console.log("User hit the resource");
  res.status(200).send("Home Page");
});
app.get("/about", (req, res) => {
  res.status(200).send("About Page");
});
app.all("*", (req, res) => {
  res.status(404).send("<h1>No Resources Found</h1>");
});
app.listen(5000, () => {
  console.log("server is listening on port 5000...");
});
```

## Resources which are not handled

A screenshot of a browser window. The address bar shows 'localhost:5000/info'. The page content is a single line of text: 'No Resources Found'. Below the browser is a navigation bar with various links like UMKC, US Banking, Investments, IND Banking, Entertainment, Insurance, Health, Food, Travel, Education, Taxes, Social Media, Resources, and TCS.

## No Resources Found

A screenshot of the Network tab in the Chrome DevTools. A request for 'info' has failed with a 404 Not Found status code. The request URL is 'http://localhost:5000/info' and the method is GET. The response headers include 'Connection: keep-alive' and 'Content-Length: 27'. The status message is 'Resource Not Available'.

**As we can it is a way less code when compared to creating a server using built-in HTTP module.**

## Express – App Example

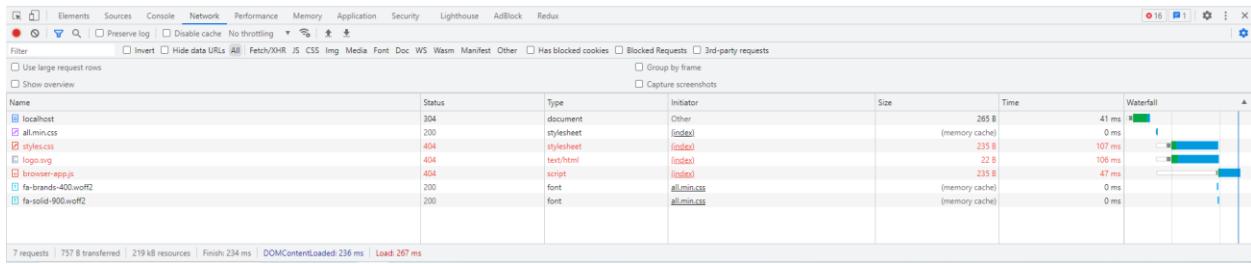
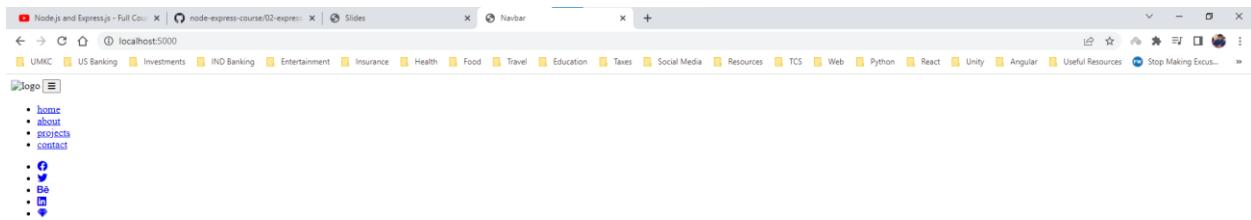
In this section, we are using the same navbar-app example with express framework. Express reduces a lot of code and provides the easier way to develop a server rather than HTTP Module server.

### app.js

A screenshot of Visual Studio Code showing the 'app.js' file for the Express tutorial. The code sets up an Express application, defines a static directory at '/public', and handles requests for '/' and '/index.html'. The terminal shows the server starting on port 5000 and logs for restarting due to changes. The project structure includes '01-express-tutorial' and '02-express-tutorial' folders containing various files like 'index.html', 'index.js', 'logger.js', etc.

```
1 // Importing the express module
2 const express = require("express");
3 // PATH built in module
4 const path = require("path");
5 // express returns a function
6 // It takes a function to configure the express
7 const app = express();
8 // setup static and middleware
9 // storing all the static pages of the project in a folder named public
10 // It will handle the static pages or the status codes, HTML types of the static resources
11 // app.use(express.static("./public"));
12 // Home page
13 app.get("/", (req, res) => {
14   res.sendFile(path.resolve(__dirname, "navbar-app", "index.html"));
15 });
16 // handling the unavailable resources
17 app.all("*", (req, res) => {
18   res.status(404).send("Resource Not Available");
19 });
20 // starting the server
21 app.listen(5000, () => {
22   console.log("Server is listening on port 5000....");
23 });
```

localhost:5000



We are experiencing the same thing, we have sent index.html as part of the response but we haven't sent the styles.css, browser-app.js and logo.svg. These needs to be taken care. In HTTP server we used to write a piece of code for each resource (styles.css, browser-app.js and logo.svg) but in express we don't write 3 different pieces of code instead we only use a method name app.use() which will take care of the middleware and static files of the project.

We place all the static files of the application (browser-app.js, styles.css, logo.svg) in folder named public (common convention name).

With the help of app.use() method we won't need to write paths, status codes, MIME types and content type of the static resources. Express takes care of it all.

Static Asset – it means that it is a file that server doesn't need to change it.

Imagine a scenario where we have 20,000 images and if we must use the HTTP server then we need to have 20,000 pieces of code for each resource. It isn't viable and is not a good approach to have that many lines of code.

## app.js

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders related to a Node.js tutorial project. The main editor area contains the code for `app.js`, which sets up an Express server to handle static files from the `public` directory. The terminal at the bottom shows the server restarting due to changes and listening on port 5000.

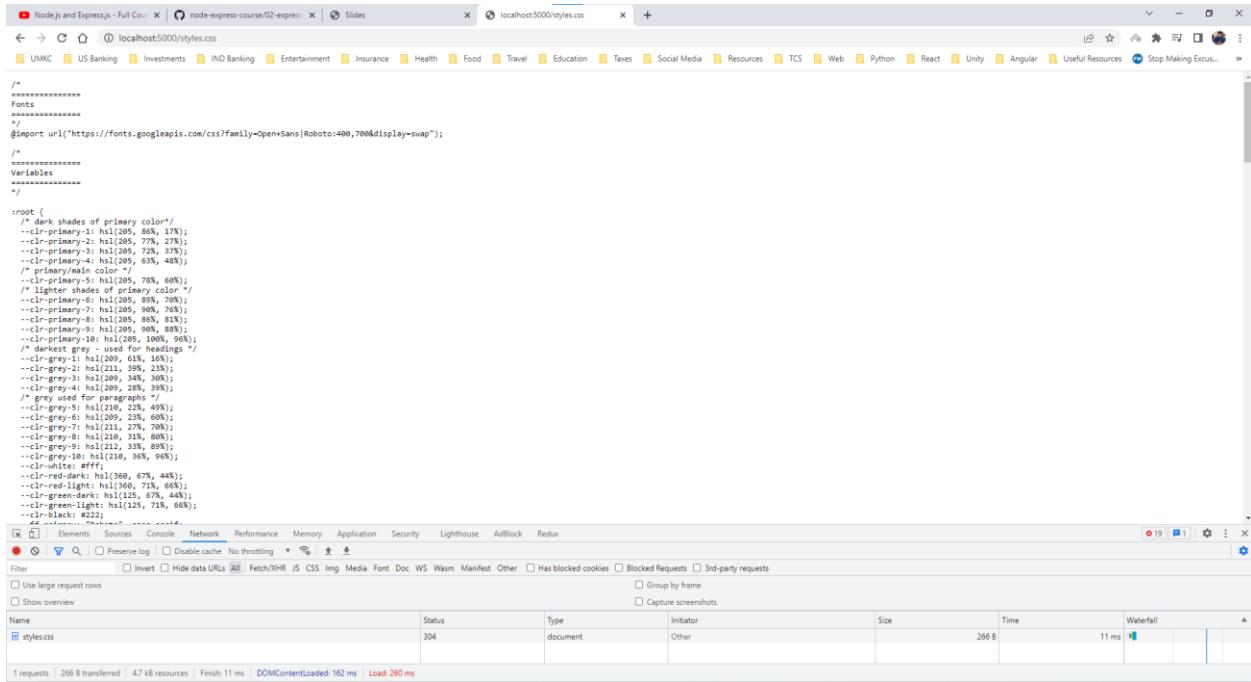
```
02-express-tutorial > node app.js
const express = require('express');
// PATH Built-in module
const path = require('path');
// Importing static files
const fs = require('fs');
// we need to instantiate the express
const app = express();
// static static and middleware
// static will take care of the static pages of the project in a folder named public
// Express will take care of the status codes, MIME types of the static resources
app.use(express.static("./public"));
app.get("/", (req, res) => {
  res.sendFile(path.resolve(__dirname, "navbar-app", "index.html"));
});
// handling the unavailable resources
app.all("*", (req, res) => {
  res.status(404).send("Resource Not Available");
});
// starting the server
app.listen(5000, () => {
  console.log("Server is listening on port 5000....");
});
// restarting due to changes...
[repeated log entries]
[repeated log entries]
```

localhost:5000

The screenshot shows a browser window displaying the `Coding Addict` homepage. The network tab of the developer tools is open, showing a list of requests made by the page. The table includes columns for Name, Status, Type, Initiator, and Size. A detailed timeline for one specific request is shown on the right, indicating it was queued at 58.04 ms and started at 58.04 ms, with a duration of 0.15 ms.

Name	Status	Type	Initiator	Size
localhost	304	document	Other	265
all-min.css	200	stylesheet	(index)	(memory cache)
styles.css	200	stylesheet	(index)	5.1 K
logos.png	200	image	(index)	8.5 K
browser-app.js	200	script	(index)	534
czs7amiba-cOpen+SansjRoboto-400,700&display=swap	200	stylesheet	zhiles.css	(disk cache)
fa-brands-400woff2	200	font	all-min.css	(memory cache)
fa-solid-900woff2	200	font	all-min.css	(memory cache)
mem5t9aG128MZp8A-UwBx2VnX8bOs2Ov2yOC54dWUJgsZ084gaVLuo#2	200	font	czs7amiba-cOpen+SansjRoboto-400,700&display=swap	(disk cache)

## localhost:5000/styles.css



The screenshot shows a browser window with multiple tabs open. The active tab is 'localhost:5000/styles.css'. Below the tabs is a navigation bar with icons for back, forward, search, and refresh. A horizontal menu bar follows, with 'localhost:5000' being the active item. The main content area displays the CSS code for 'styles.css'. At the bottom of the browser window, the developer tools Network tab is visible, showing a single request for 'styles.css' with a status of 200 OK, size of 266 B, and a duration of 11 ms.

```
/*
*****
Fonts
*****
*/
@import url("https://fonts.googleapis.com/css?family=Open+Sans|Roboto:400,700&display=swap");

/*
*****
Variables
*****
*/
:root {
    /* dark shades of primary color */
    --cl-primary-1: hsl(205, 84%, 17%); 
    --cl-primary-2: hsl(205, 77%, 27%); 
    --cl-primary-3: hsl(205, 72%, 37%); 
    --cl-primary-4: hsl(205, 65%, 48%); 
    /* primary/main color */
    --cl-primary-5: hsl(205, 78%, 60%); 
    --cl-primary-6: hsl(205, 80%, 60%); 
    --cl-primary-7: hsl(205, 80%, 70%); 
    --cl-primary-8: hsl(205, 80%, 70%); 
    --cl-primary-9: hsl(205, 80%, 70%); 
    --cl-primary-10: hsl(205, 100%, 96%); 
    /* darkest grey - used for headings */
    --cl-grey-1: hsl(211, 39%, 12%); 
    --cl-grey-2: hsl(211, 39%, 23%); 
    --cl-grey-3: hsl(209, 34%, 30%); 
    --cl-grey-4: hsl(210, 31%, 39%); 
    --cl-grey-5: hsl(210, 22%, 49%); 
    --cl-grey-6: hsl(209, 20%, 60%); 
    --cl-grey-7: hsl(210, 17%, 65%); 
    --cl-grey-8: hsl(210, 31%, 80%); 
    --cl-grey-9: hsl(212, 33%, 89%); 
    --cl-grey-10: hsl(210, 31%, 96%); 
    --cl-white: #fff; 
    --cl-red-dark: hsl(360, 5%, 44%); 
    --cl-red-light: hsl(360, 71%, 66%); 
    --cl-green-dark: hsl(125, 67%, 44%); 
    --cl-green-light: hsl(125, 71%, 66%); 
    --cl-black: #222;
}

/*
*****
Elements
*****
*/

```

## Express – All Static

While working on the above express application, we must get a doubt that even index.html file is static and why are not adding it to the folder.

We can do that in two approaches.

1. Yes, we can add index.html file in the static folder. So, in this section we are going to add the index.html file in static folder.

Index.html file is always a root file, so when the user hits the server by default the server will serve the index.html file present in the static folder. Since our index.html file has all the paths to browser-app.js, logo.svg and styles.css we are going to be in a good shape, and we don't even need to setup the send File option.

2. SSR (Server-Side Rendering)

In this section, let us discuss only about first approach.

## app.js

The screenshot shows the Visual Studio Code interface. The left sidebar displays the project structure for '02-express-tutorial'. The main editor window contains the 'app.js' file:

```
02-express-tutorial M
  app.js
  ...
  02-express-tutorial M
    app.js
    ...
    // Import express - require("express");
    2 // PATH built-in module
    3 const path = require('path');
    4 // express returns a function
    5 // Create a new instance of the express
    6 const app = express();
    7 // setup static and middleware
    8 // storing all the static parts of the project in a folder named public
    9 app.use(express.static("./public"));
   10 // Home Page
   11 app.get('/', (req, res) => {
   12   res.sendFile(path.resolve(__dirname, "navbar-app", "index.html"));
   13 });
   14 // handling the unavailable resources
   15 app.all('*', (req, res) => {
   16   res.status(404).send("Resource Not Available");
   17 });
   18 // starting the server
   19 app.listen(5000, () => {
   20   console.log("Server is listening on port 5000....");
   21 });
   22 });
   23 
```

The terminal window at the bottom shows the command-line output of running the application:

```
PS C:\Users\saikiran\Documents\Node JS Tutorial\Node JS Tutorial\02-express-tutorial> npm start
> 02-express-tutorial@1.0.0 start C:\Users\saikiran\Documents\Node JS Tutorial\Node JS Tutorial\02-express-tutorial
> nodemon app.js

[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,json
[nodemon] starting `node app.js`
Server is listening on port 5000....
```

localhost:5000

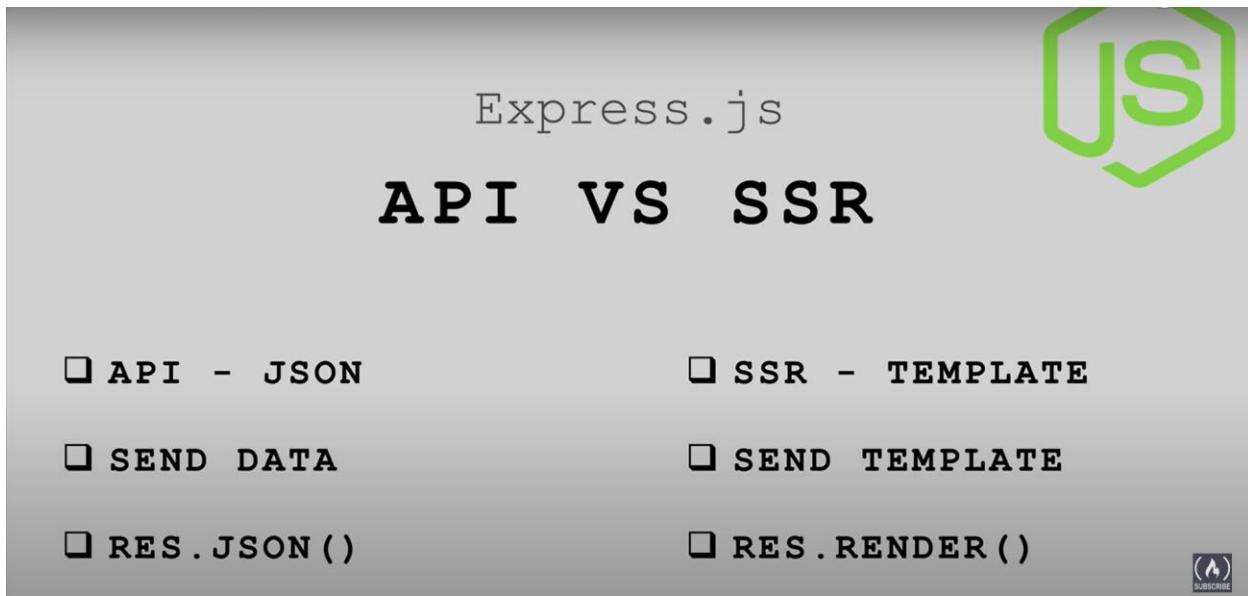
The screenshot shows a browser window displaying the 'Coding Addict' website. The network tab of the developer tools is open, showing the following resource list:

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	304	document	Other	265 B	11 ms	
allmin.css	200	stylesheet	(index)		(disk cache)	
styles.css	304	stylesheet	(index)	266 B	10 ms	
browser-app.js	304	script	(index)	264 B	6 ms	
logos.svg	304	svg+xml	(index)	266 B	9 ms	
css?family=Open+Sans:400,700&display=swap	200	stylesheet	styles.css+Infinity		(memory cache)	0 ms
fe-brands-400.woff2	200	font	allmin.css		(memory cache)	0 ms
fe-solid-900.woff2	200	font	allmin.css		(memory cache)	0 ms
mem:5yaG5t126MjZp8A-UwBx2vNxKBbObj2Ov2jOOS4dVWUjsz2084gaVlwoF2	200	font	css?family=Open+Sans:400,700&display=swap		(memory cache)	0 ms

At the bottom of the network tab, the following statistics are displayed:

9 requests | 1.1 kB transferred | 256 kB resources | Finish: 113 ms | DOMContentLoaded: 196 ms | Load: 255 ms

## API vs SSR (Application Programming Interface vs Server-Side Rendering)



Express can be used in two ways

1. Using Express to create an API
2. Using Express to create templates with Server-Side Rendering

In general, API is used to set up an HTTP interface to interact with data. Data is sent through JSON (JavaScript Object Notation) and to send back our response we use `res.json()` method. This method will do all the heavy lifting like setting up the proper content-type and stringify our data.

In Server-Side Rendering, we will set up templates and send back the entire HTML, CSS and JavaScript to frontend using `res.render()` method.

We will mostly try to learn about API rather than the SSR because API is main part of the Express and once API knowledge is done, we can easily handle SSR part as well.

Main Idea around API is that our server provides the data to any frontend application that's wanting to access and use the data.

## React Tabs Project (API Call)

The screenshot shows a browser window with two tabs: "Node.js and Express.js - Full Course" and "https://course-api.com/react-tabs-project". The second tab is active, displaying a JSON array of objects. Each object represents a job entry with fields like id, order, title, dates, and duties. The duties field contains a long, complex string of words. Below the browser is the Network tab of the developer tools, which shows a request to "https://course-api.com/react-tabs-project". The Headers section shows the URL, method (GET), status code (304), and other metadata. The Response Headers section shows "Content-Type: application/json; charset=UTF-8".

```
[{"id": "recAGJfIU4CeaV0HL", "order": 3, "title": "Full Stack Web Developer", "dates": "December 2015 - Present", "duties": ["Tote bag sartorial mlkshk air plant vinyl banjo lumbersexual poke leggings offal cold-pressed brunch neutra. Hammock photo booth live-edge disrupt.", "Post-ironic selvage chambray sartorial freegan meditation. Chambray chartreuse kombucha meditation, man bun four dollar toast street art cloud bread live-edge heirloom.", "Butcher drinking vinegar franzen authentic messenger bag copper mug food truck taxidermy. Mumblecore lomo echo park readymade iPhone migas single-origin coffee franzen cloud bread tilde vegan flexitarian."], "company": "TOMMY"}, {"id": "recIL6mJNfwObonols", "order": 2, "title": "Front-End Engineer", "dates": "May 2015 - December 2015", "duties": ["Hashtag drinking vinegar scenester mumblecore snackwave four dollar toast, lumbersexual XOXO. Cardigan church-key pabst, biodiesel vexillologist viral squid.", "Franzen af pitchfork, mumblecore try-hard kogi XOXO roof party la croix cardigan neutra retro tattooed copper mug. Meditation lomo biodiesel scenester", "Fam VHS enamel pin try-hard echo park raw denim unicorn fanny pack vape authentic. Helvetica fixie church-key, small batch jianbing messenger bag scenester +1", "Fam VHS enamel pin try-hard echo park raw denim unicorn fanny pack vape authentic. Helvetica fixie church-key, small batch jianbing messenger bag scenester +1"], "company": "BIGDROP"}, {"id": "rec6i1x8G/Y99hQqS5", "order": 1, "title": "Engineering Intern", "dates": "May 2014 - September 2015", "duties": ["I'm baby woke mumblecore stumptown enamel pin. Snackwave prism pork belly, blog vape four loko sriracha messenger bag jean shorts DIY bushwick VHS. Banjo post-ironic hella af, palo santo craft beer gluten-free.", "YOLO drinking vinegar chambray pok pok selfies quinoa kinfolk pitchfork street art la croix unicorn DIY. Woke offal jianbing venmo tote bag, palo santo subway tile slow-carb post-ironic pug ugh taxidermy squid.", "Pour-over glossier chambray umami 3 wolf moon. Iceland kale chips asymmetrical craft beer actually forage, biodiesel tattooed fingerstache. Pork belly lomo man braid, portland pitchfork locavore man bun prism."], "company": "UKER"}]
```

Request URL: https://course-api.com/react-tabs-project  
Request Method: GET  
Status Code: 304 OK  
Remote Address: 52.202.168.65:443  
Referrer Policy: no-referrer

## JSON Basics

JSON stands for JavaScript Object Notation

Whatever may be the frontend, API provides the data to the browser.

`res.json()` sends a JSON response. This method sends a response (with the correct content-type) that is the parameter converted to a JSON string using `JSON.stringify()`

The parameter can be any JSON type, including object, array, string, Boolean, Number, or null and you can also use it to convert other values to JSON.

```
res.json(null)
```

```
res.json({user: "Sai"})
```

```
res.status(500).json({error: "message"})
```

## app.js

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders related to a 'NODE JS TUTORIAL YOUTUBE' project, including '01-node-tutorial' and '02-express-tutorial'. The '02-express-tutorial' folder contains files like 'navbar-app', 'node\_modules', 'public', '.gitignore', '01-http-basics.js', '02-http-app.js', '03-express-basics.js', '04-express-app.js', '05-all-static.js', '06-basic-json.js', 'app.js', 'data.js', 'index.html', 'package-lock.json', 'package.json', 'Express-JS-Notes.pdf', 'Node-JS-Notes.pdf', and 'README.md'. The 'app.js' file is open in the editor, showing the following code:

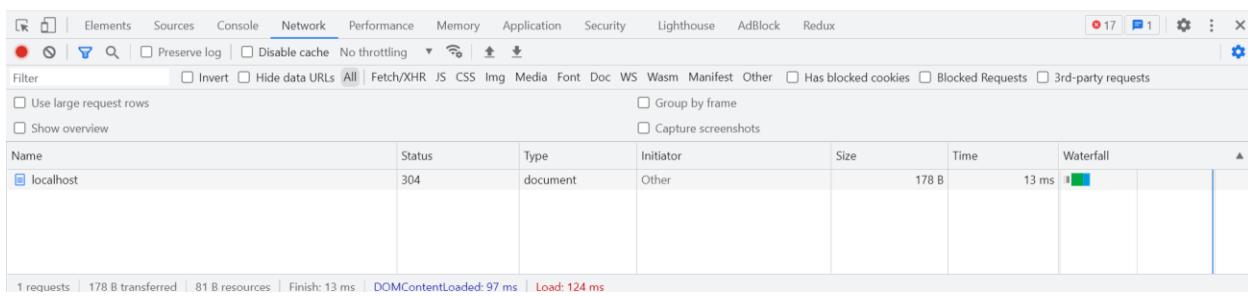
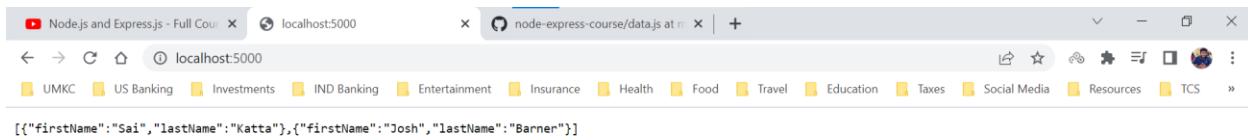
```
02-express-tutorial > app.js > app.get("/") callback
1 const express = require("express");
2 const app = express();
3
4 app.get("/", (req, res) => {
5   res.json([
6     { firstName: "Sai", lastName: "Katta" },
7     { firstName: "Josh", lastName: "Barner" },
8   ]);
9 });
10
11 app.listen(5000, () => {
12   console.log("Server is listening on Port 5000....");
13 });
14
```

The Terminal tab at the bottom shows the command-line output of running the application with nodemon:

```
> 02-express-tutorial@1.0.0 start C:\Users\saikr\Documents\Node JS Tutorial Youtube\02-express-tutorial
> nodemon app.js

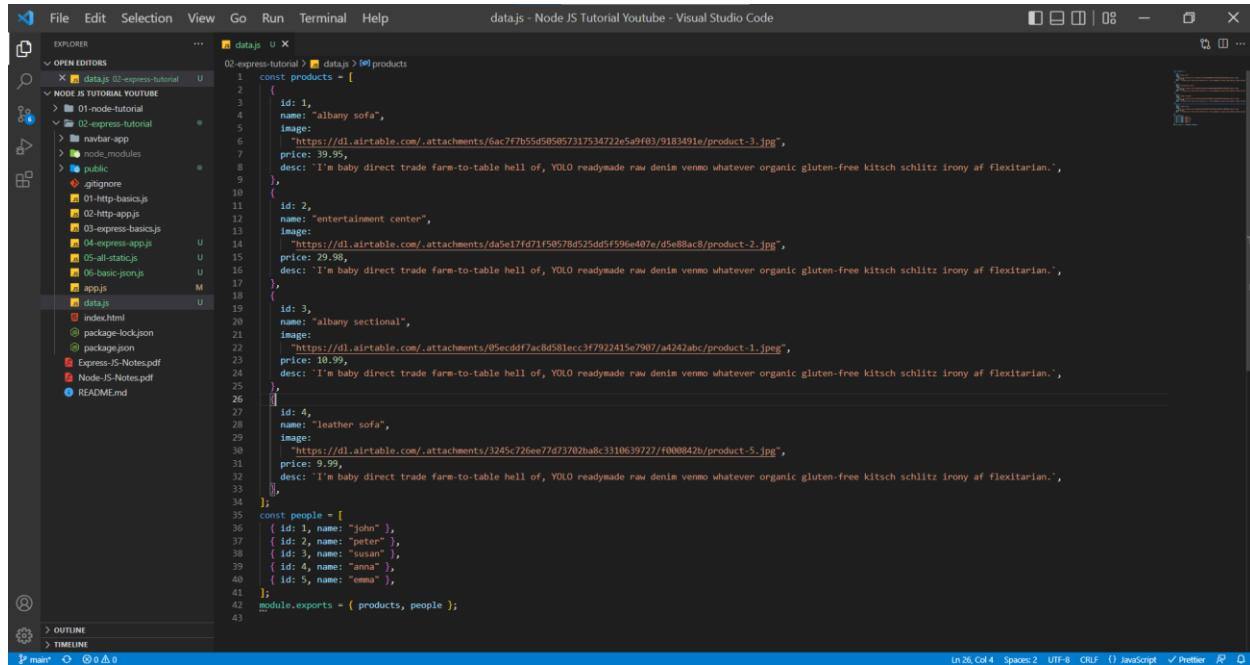
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node app.js
Server is listening on Port 5000....
```

localhost:5000



Now we will a file name data.js which contains data consisting of arrays.

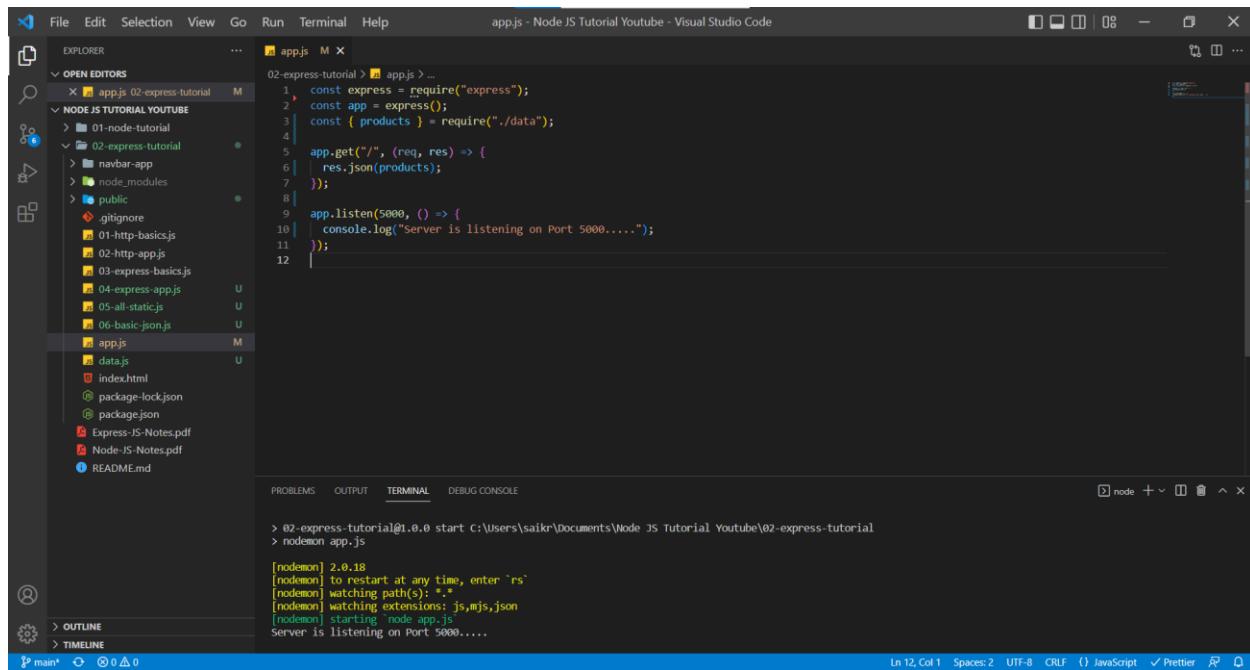
data.js



```
File Edit Selection View Go Run Terminal Help
data.js - Node JS Tutorial Youtube - Visual Studio Code
OPEN EDITORS
02-express-tutorial > data.js > products
1 const products = [
2   {
3     id: 1,
4     name: "albany sofa",
5     image:
6       "https://d1.airtable.com/.attachments/6ac7f755d5057317534722e5e9f03/9183491e/product-3.jpg",
7     price: 39.95,
8     desc: "I'm baby direct trade farm-to-table hell of, YOLO ready-made raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian.",
9   },
10  {
11    id: 2,
12    name: "entertainment center",
13    image:
14      "https://d1.airtable.com/.attachments/d4e17fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg",
15     price: 29.95,
16     desc: "I'm baby direct trade farm-to-table hell of, YOLO ready-made raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian.",
17   },
18  {
19    id: 3,
20    name: "albany sectional",
21    image:
22      "https://d1.airtable.com/.attachments/05ecdd7ac8d581ecc3f7922415e7907/a4242abc/product-1.jpeg",
23     price: 10.99,
24     desc: "I'm baby direct trade farm-to-table hell of, YOLO ready-made raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian.",
25   },
26  {
27    id: 4,
28    name: "leather sofa",
29    image:
30      "https://d1.airtable.com/.attachments/3245c726ee77d73702ba8c3310639727/f000842b/product-5.jpg",
31     price: 9.99,
32     desc: "I'm baby direct trade farm-to-table hell of, YOLO ready-made raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian.",
33   },
34 ],
35 const people = [
36   { id: 1, name: "john" },
37   { id: 2, name: "pete" },
38   { id: 3, name: "susan" },
39   { id: 4, name: "anna" },
40   { id: 5, name: "emma" },
41 ],
42 module.exports = { products, people };
43 
```

Instead of Hardcoding the data in app.js, we will try to import data from data.js file and send it to browser.

app.js

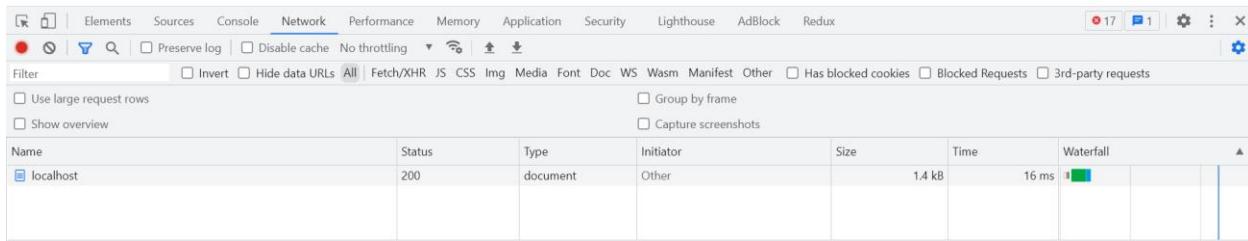
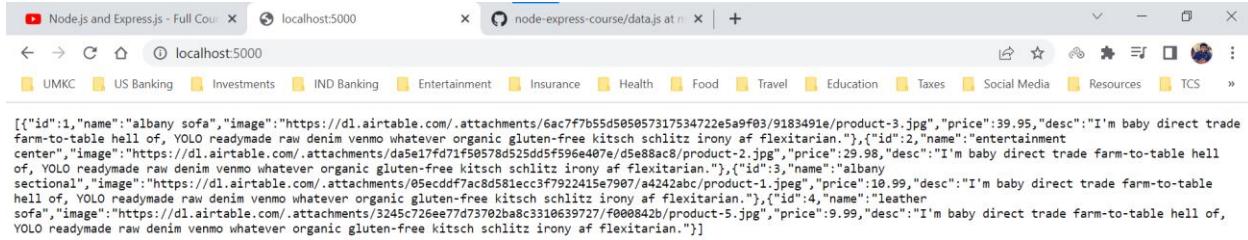


```
File Edit Selection View Go Run Terminal Help
app.js - Node JS Tutorial Youtube - Visual Studio Code
OPEN EDITORS
02-express-tutorial > app.js ...
1 const express = require("express");
2 const app = express();
3 const { products } = require("./data");
4
5 app.get("/", (req, res) => {
6   res.json(products);
7 });
8
9 app.listen(5000, () => {
10   console.log("Server is listening on Port 5000....");
11 });
12 
```

TERMINAL

```
> 02-express-tutorial@1.0.0 start C:\Users\saikr\Documents\Node JS Tutorial Youtube\02-express-tutorial
> node app.js
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server is listening on Port 5000....
```

localhost:5000



## Params, Query String – Setup

### Route Params

In this section, we are going to learn about requesting data based on the parameters. Assume a scenario where we have 2000 resources on the page and user wants to request each resource based upon the click.

We define each resource with an ID which will be unique for a resource. When a user tries to request (either through a click or through entering address on address bar), we need to provide data based on the request.

In general, we need to capture the parameter of the request and provide the data according to that. In a request we have a property name **params** which provides a javascript object containing parameters of that request. We need to extract the parameter and find the respective data.

**Note: We need to remember that values in params object would be of data type String.**

**app.all()** method is not working when we try to request a response from a product which is not available.

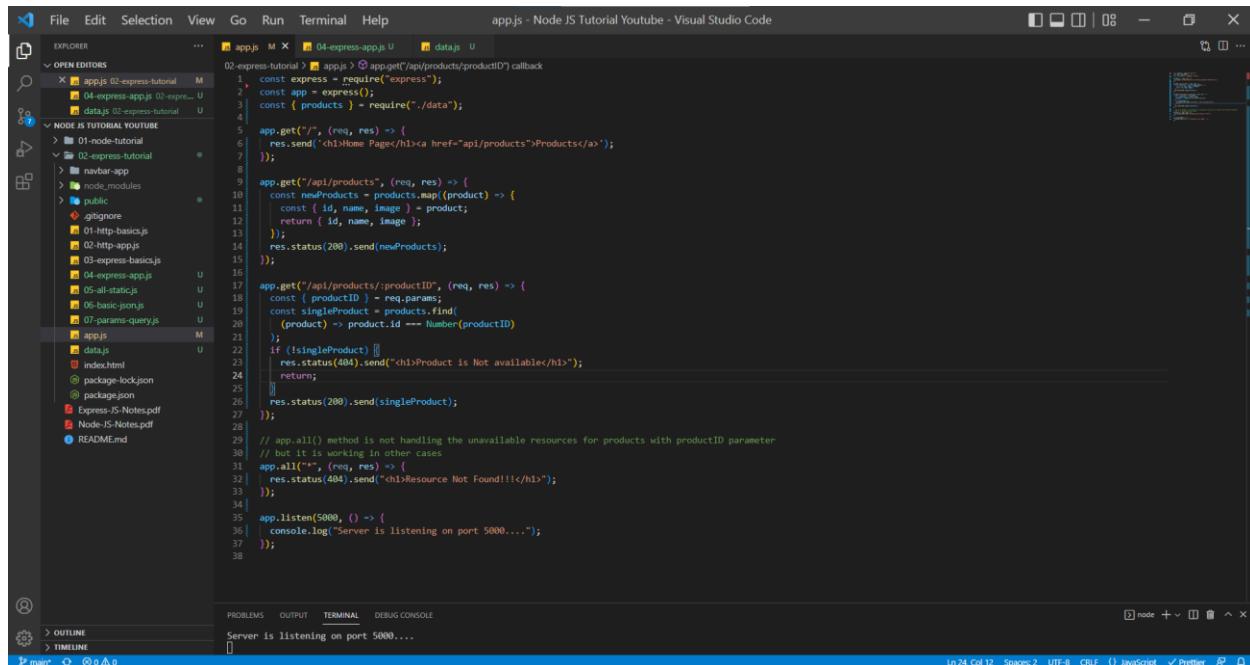
Ex: localhost:5000/api/products/hgjh

But it is working when we are trying to request a below resource

Ex: localhost:5000/api/hgdsjgh – It is providing the Resource Not Available response as expected.

Maye be it is not checking inside the Query Parameters.

## app.js



```
const express = require("express");
const app = express();
const { products } = require("./data");

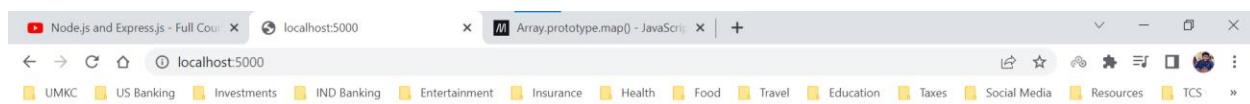
app.get("/api/products/:productId", (req, res) => {
  const product = products.find((product) => product.id === Number(req.params.productId));
  if (!product) {
    res.status(404).send("<h1>Product is Not available</h1>");
    return;
  }
  res.status(200).send(product);
});

app.get("/api/products", (req, res) => {
  const newProducts = products.map((product) => {
    const { id, name, image } = product;
    return { id, name, image };
  });
  res.status(200).send(newProducts);
});

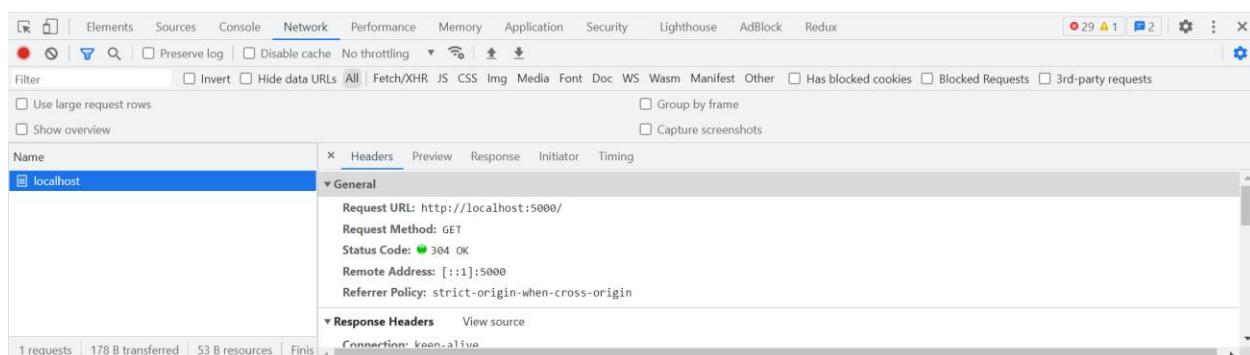
app.get("/", (req, res) => {
  const newProducts = products.map((product) => {
    const { id, name, image } = product;
    return { id, name, image };
  });
  res.send(`<h1>Home Page</h1><a href="/api/products">Products</a>`);
});

app.listen(5000, () => {
  console.log("Server is listening on port 5000....");
});
```

localhost:5000 – Home Page



## Network Tab in DevTools



Name	Headers	Preview	Response	Initiator	Timing
localhost	Request URL: http://localhost:5000/ Request Method: GET Status Code: 200 OK Remote Address: [::1]:5000 Referrer Policy: strict-origin-when-cross-origin				
	Content-Type: application/json Connection: keep-alive				

## localhost:5000/api/products – Products Page

The screenshot shows a browser window with the URL `localhost:5000/api/products`. The page content is a JSON array of product objects:

```
[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg"}, {"id":2,"name":"entertainment center","image":"https://dl.airtable.com/.attachments/dae17fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg"}, {"id":3,"name":"albany sectional","image":"https://dl.airtable.com/.attachments/05ecddf7ac8d581ecc3f7922415e7907/a4242abc/product-1.jpeg"}, {"id":4,"name":"leather sofa","image":"https://dl.airtable.com/.attachments/3245c726ee77d73702ba8c3310639727/f000842b/product-5.jpg"}]
```

The screenshot shows the Network tab in the Chrome DevTools. A request to `localhost:5000/api/products` is listed. The request details show:

- Request URL: `http://localhost:5000/api/products`
- Request Method: GET
- Status Code: 200 OK
- Remote Address: [::1]:5000
- Referrer Policy: strict-origin-when-cross-origin

## localhost:5000/api/products/1 – Product with ID #1

The screenshot shows a browser window with the URL `localhost:5000/api/products/1`. The page content is a JSON object for the product with ID 1:

```
{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}
```

The screenshot shows the Network tab in the Chrome DevTools. A request to `localhost:5000/api/products/1` is listed. The request details show:

- Request URL: `http://localhost:5000/api/products/1`
- Request Method: GET
- Status Code: 200 OK
- Remote Address: [::1]:5000
- Referrer Policy: strict-origin-when-cross-origin

## localhost:5000/api/products/abcd – Unavailable Product

A screenshot of a browser window. The address bar shows 'localhost:5000/api/products/abcd'. The page content says 'Product is Not available'.

### Product is Not available

A screenshot of the Network tab in Chrome DevTools. A request for 'abcd' is selected. The General section shows:

- Request URL: http://localhost:5000/api/products/abcd
- Request Method: GET
- Status Code: 404 Not Found
- Remote Address: [::1]:5000
- Referrer Policy: strict-origin-when-cross-origin

## localhost:5000/qwerty – Unavailable Resource on the Server

A screenshot of a browser window. The address bar shows 'localhost:5000/qwerty'. The page content says 'Resource Not Found!!!'.

### Resource Not Found!!!

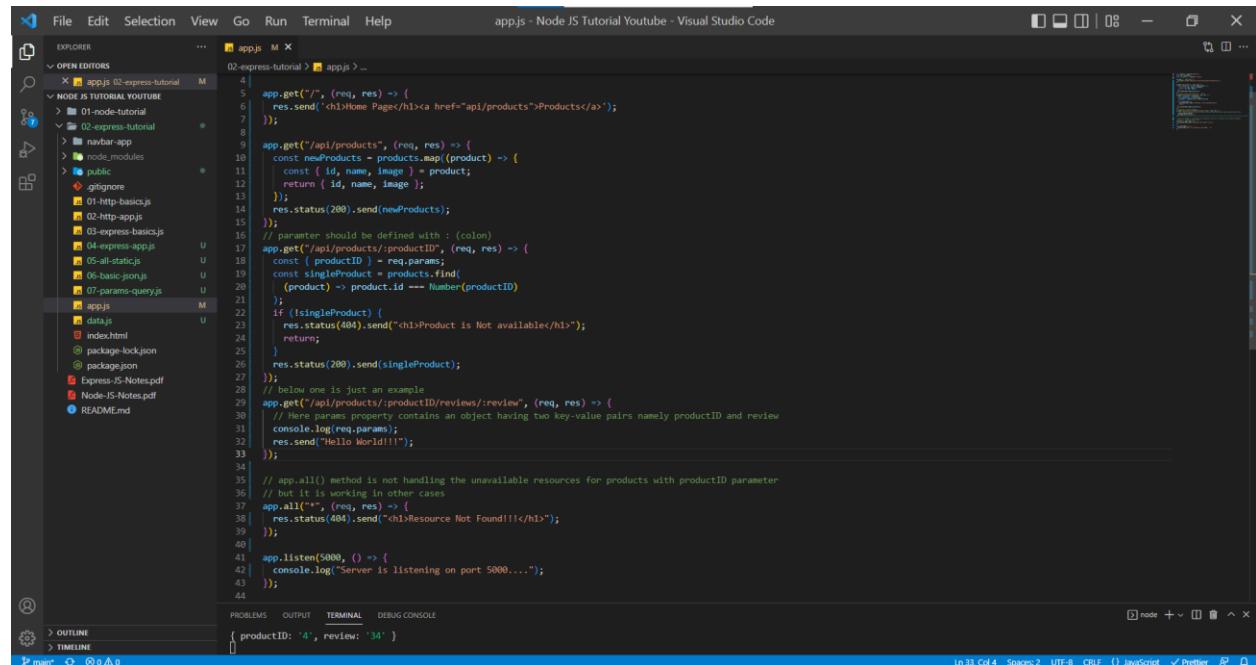
A screenshot of the Network tab in Chrome DevTools. A request for 'qwerty' is selected. The General section shows:

- Request URL: http://localhost:5000/qwerty
- Request Method: GET
- Status Code: 404 Not Found
- Remote Address: [::1]:5000
- Referrer Policy: strict-origin-when-cross-origin

## Params – Extra Info

Route Parameters can get complex way more than the above example.

app.js



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
app.js - Node JS Tutorial Youtube - Visual Studio Code
explorer
node-express-tutorial
01-node-tutorial
02-express-tutorial
  navbar-app
  node_modules
  public
  .gitignore
  01-http-basics.js
  02-http.js
  03-express-basics.js
  04-express-app.js
  05-all-static.js
  06-basic-json.js
  07-params-query.js
  app.js
  datajs
  index.html
  package-lock.json
  package.json
  Express-JS-Notes.pdf
  Node-JS-Notes.pdf
  README.md

app.js
index.html
package-lock.json
package.json
Express-JS-Notes.pdf
Node-JS-Notes.pdf
README.md

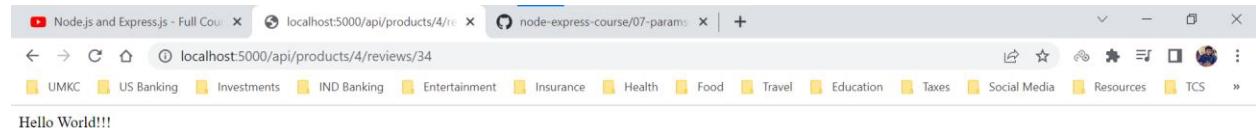
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
productID: '4', review: '34'

In 33 Col 4 Spaces 2 UTF-8 CRLF (1) JavaScript ✓ prettier AP O
```

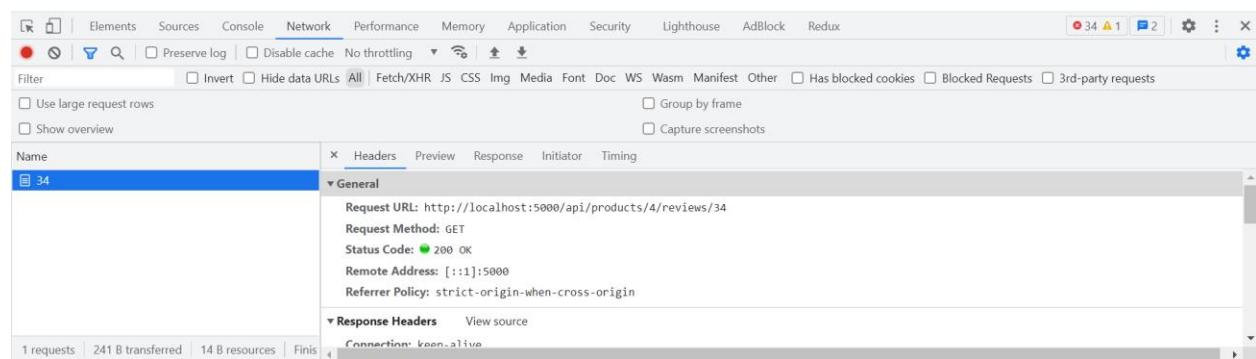
The code in app.js handles route parameters for products and reviews. It includes logic for handling multiple parameters and returning specific products or reviews based on their IDs.

In console statement of app.js, we can find the req.params property data.

localhost:5000/api/products/4/reviews/34



Hello World!!!



When we try to enter localhost:5000/api/products/4/review/34. Here I have changed the name from reviews to review and hence we are getting a message “Response Not Found” by app.all() method.

By this example, we can understand that app.get() provides an error message only when it couldn't find the URL it doesn't take responsibility for the placeholders/ route parameters like (productid and review) in the request URL.

The screenshot shows a browser window with three tabs open: 'Node.js and Express.js - Full Cou...', 'localhost:5000/api/products/4/review/34', and 'node-express-course/07-param...'. The active tab displays a white page with the bold text 'Resource Not Found!!!' centered. Below the browser window is the Network tab of the developer tools. The table shows one request: '34' with a status of '404', type 'document', initiator 'Other', size '265 B', time '9 ms', and a waterfall chart. The bottom of the developer tools shows performance metrics: 1 requests, 265 B transferred, 30 B resources, Finish: 9 ms, DOMContentLoaded: 151 ms, and Load: 257 ms.

Name	Status	Type	Initiator	Size	Time	Waterfall
34	404	document	Other	265 B	9 ms	[Waterfall Chart]

## Query String Parameters

Query String Parameters are also known as URL Parameters. Essentially this is the way for us to send amounts of information to the server using the URL.

This information is usually used as parameters for database query or filtering the results.

Here are few examples to have a look at the Query String Parameters. (HackerNews Story API)

Basically, they won't be the part of the URL and are separated by a question mark (?) in the URL.

### All stories matching foo

<http://hn.algolia.com/api/v1/search?query=foo&tags=story>

### All comments matching bar

<http://hn.algolia.com/api/v1/search?query=bar&tags=comment>

### All URLs matching bar

<http://hn.algolia.com/api/v1/search?query=bar&restrictSearchableAttributes=url>

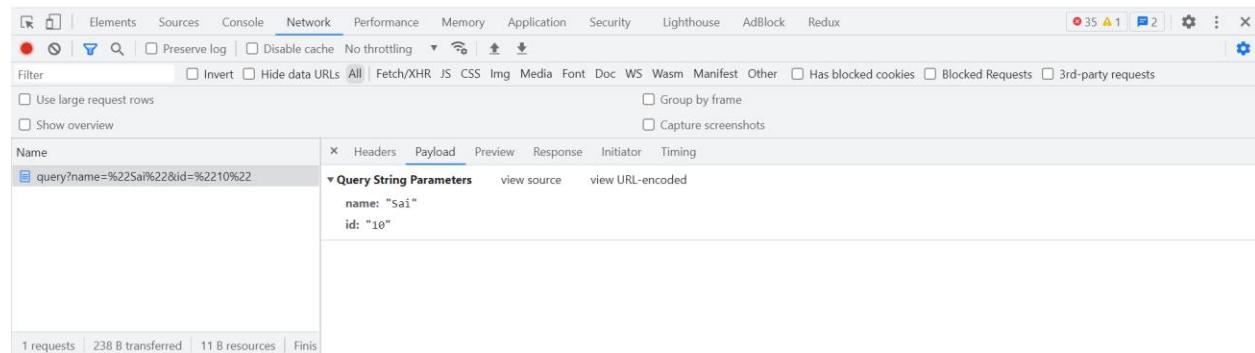
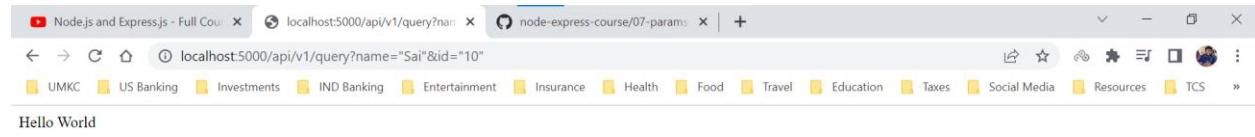
Whatever is after the question mark is not technically part of the URL meaning it is just a way for us to send that data to the server and then server decides what to do with this data.

The phase after question mark is nothing but a specific information about the data a user is requesting to the server.

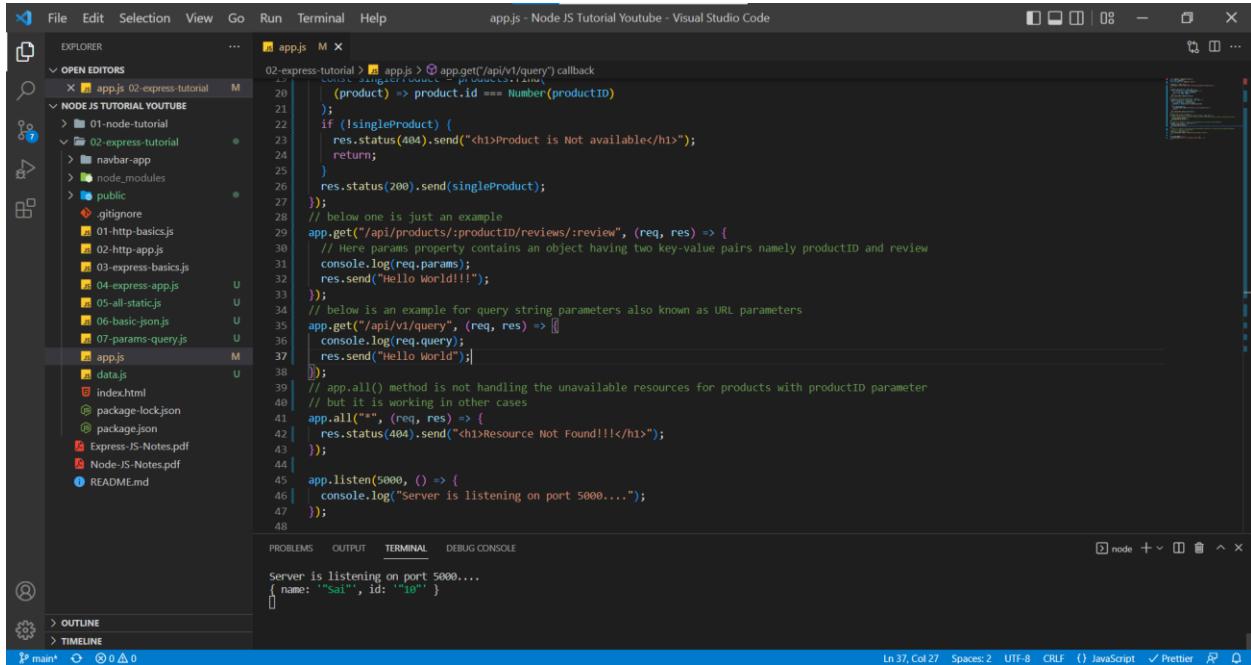
We can add how many query parameters, but they need to start after question mark (?) and all of them need to be combined with ampersand (&), based on the query parameters we can develop the functionality and provide the response.

We can get the parameters as part of the request body, and we can retrieve them using the property named **query**.

`http://localhost:5000/api/v1/query?name="Sai"&id="10"`



We are console logging the query parameters in the terminal and can see the query parameters by `req.query` property.



```

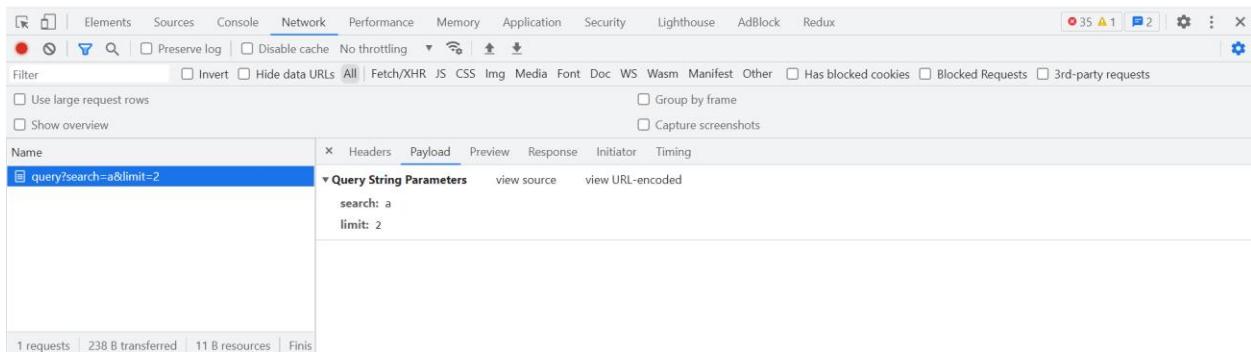
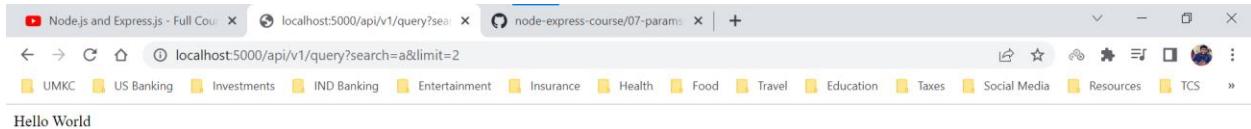
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS NODE JS TUTORIAL YOUTUBE 01-node-tutorial 02-express-tutorial M app.js M
app.js > 02-express-tutorial > app.get('/api/v1/query') callback
  (product) => product.id === Number(productId)
);
if (!singleProduct) {
  res.status(404).send("Product is Not available");
  return;
}
res.status(200).send(singleProduct);
});
// below one is just an example
app.get('/api/products/:productId/reviews/:review', (req, res) => {
  // Here params property contains an object having two key-value pairs namely productId and review
  console.log(req.params);
  res.send("Hello World!!!");
});
// below is an example for query string parameters also known as URL parameters
app.get('/api/v1/query', (req, res) => [
  console.log(req.query);
  res.send("Hello World");
]);
// app.all() method is not handling the unavailable resources for products with productId parameter
// but it is working in other cases
app.all('*', (req, res) => [
  res.status(404).send("Resource Not Found!!!");
]);
app.listen(5000, () => {
  console.log("Server is listening on port 5000....");
});
}
  
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Server is listening on port 5000...  
 { name: "Sai", id: "10" }

Ln 37, Col 27 Spaces: 2 UTF-8 CRLF ( ) JavaScript ✓ Prettier

`localhost:5000/api/v1/query?search=a&limit=2`



We can see those query parameters in the terminal

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

- File Explorer:** Shows the project structure with files like `01-node-tutorial`, `02-express-tutorial`, `navbar-app`, `node_modules`, `public`, `.gitignore`, `01-http-basics.js`, `02-http-app.js`, `03-express-basics.js`, `04-express-app.js`, `05-all-static.js`, `06-basic-json.js`, `07-params-query.js`, `app.js`, `data.js`, `index.html`, `package-lock.json`, `package.json`, `Express-JS-Notes.pdf`, `Node-JS-Notes.pdf`, and `README.md`.
- Terminal:** Displays the output of the Node.js application, showing the server listening on port 5000 and the query parameters passed in the URL.
- Status Bar:** Shows the current file is `app.js`, the line number is 37, column 27, and the file size is 27. It also indicates the file is a JavaScript file and is prettified.

```
02-express-tutorial > node app.js > app.get("/api/v1/query") callback
 20 |   const singleProduct = products.find(
 21 |     (product) => product.id === Number(productId)
 22 |   );
 23 |   if (!singleProduct) {
 24 |     res.status(404).send("<h1>Product is Not available</h1>");
 25 |     return;
 26 |   }
 27 |   res.status(200).send(singleProduct);
 28 | });
 29 // below one is just an example
 30 app.get("/api/products/:productId/reviews/:review", (req, res) => {
 31 // Here params property contains an object having two key-value pairs namely productId and review
 32 console.log(req.params);
 33 res.send("Hello world!!!");
 34 });
 35 // below is an example for query string parameters also known as URL parameters
 36 app.get("/api/v1/query", (req, res) => [
 37   console.log(req.query);
 38   res.send("Hello World!");
 39 ]);
 40 // app.all() method is not handling the unavailable resources for products with productId parameter
 41 // but it is working in other cases
 42 app.all("*", (req, res) => {
 43   res.status(404).send("<h1>Resource Not Found!!!</h1>");
 44 });
 45 app.listen(5000, () => {
 46   console.log("Server is listening on port 5000....");
 47 });
 48
```

```
Server is listening on port 5000...
{
  name: "Sal",
  id: "10"
}
{
  search: 'a',
  limit: '2'
}
```

Note: The data we pass through query parameters are always a string, we don't have to send them as string they will be automatically converted to string and sent to the server.

Here we are creating a scenario where we expect the user to send two query parameters namely search, limit and if we don't get any of those parameters, we will display all products but if we receive those query parameters, we will display the data accordingly.

app.js

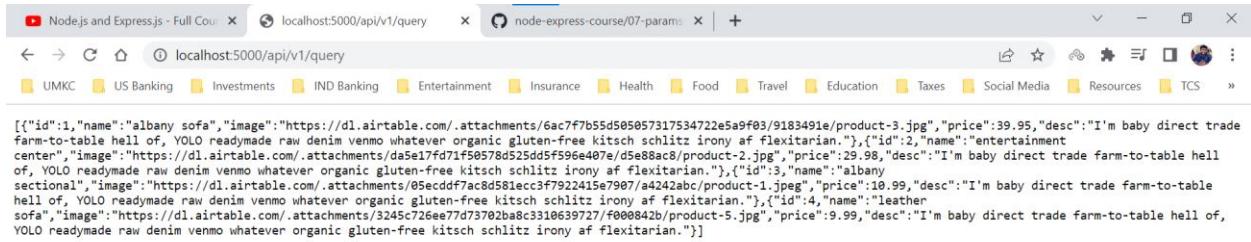
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `01-node-tutorial`, `02-express-tutorial`, `navbar-app`, `node_modules`, `public`, `.gitignore`, `01-http-basics.js`, `02-http-app.js`, `03-express-basics.js`, `04-express-app.js`, `05-all-static.js`, `06-basic-json.js`, `07-params-query.js`, `app.js`, `data.js`, `index.html`, `package-lock.json`, `package.json`, `Express-JS-Notes.pdf`, `Node-JS-Notes.pdf`, and `README.md`.
- Terminal:** Displays the output of the Node.js application, showing the server listening on port 5000 and the query parameters passed in the URL.
- Status Bar:** Shows the current file is `app.js`, the line number is 50, column 40, and the file size is 40. It also indicates the file is a JavaScript file and is prettified.

```
02-express-tutorial > node app.js > app.get("/api/v1/query") callback
 33 });
 34 // below is an example for query string parameters also known as URL parameters
 35 app.get("/api/v1/query", (req, res) => [
 36   const { search, limit } = req.query;
 37   let sortedProducts = [...products];
 38   // if search query parameter is available in the URL
 39   if (search) {
 40     // filtering the array using the first letter of the name
 41     sortedProducts = sortedProducts.filter((product) =>
 42       product.name.startsWith(search)
 43     );
 44   }
 45   // if limit query parameter is available in the URL
 46   if (limit) {
 47     // since the query parameter is a string, we need to convert it into Number data type
 48     sortedProducts = sortedProducts.slice(0, Number(limit));
 49   }
 50   res.status(200).json(sortedProducts);
 51 });
 52 // app.all() method is not handling the unavailable resources for products with productId parameter
 53 // but it is working in other cases
 54 app.all("*", (req, res) => {
 55   res.status(404).send("<h1>Resource Not Found!!!</h1>");
 56 });
 57 app.listen(5000, () => {
 58   console.log("Server is listening on port 5000....");
 59 });
 60
```

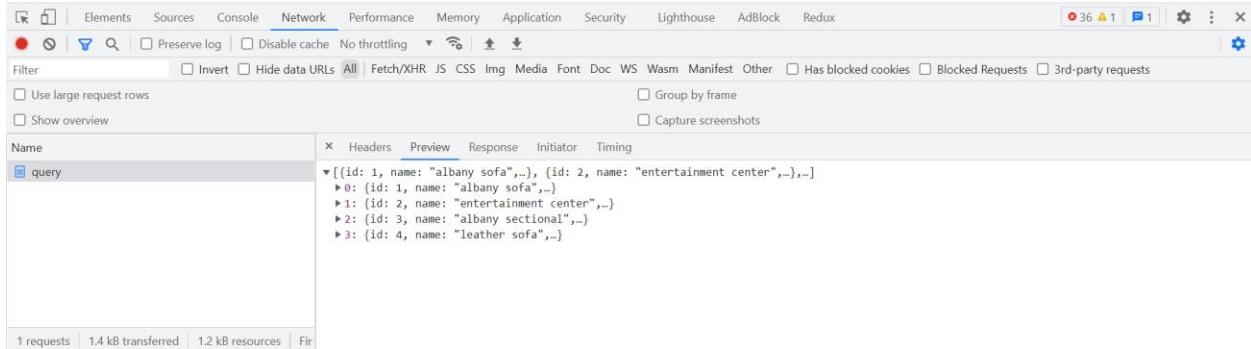
```
[nodemon] starting 'node app.js'
Server is listening on port 5000...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on port 5000...
```

## http://localhost:5000/api/v1/query - With No Query Parameters



The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Cou", "localhost:5000/api/v1/query", and "node-express-course/07-params". The active tab displays a JSON array of four sofa objects. Each object has an id, name, image, price, and desc. The descriptions are identical, referencing YOLO ready-made sofas.

```
[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":2,"name":"entertainment center","image":"https://dl.airtable.com/.attachments/dae517fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg","price":29.98,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":3,"name":"albany sectional","image":"https://dl.airtable.com/.attachments/05ecddf7ac8d81ecc3f7922415e7907/44242abc/product-1.jpeg","price":10.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":4,"name":"leather sofa","image":"https://dl.airtable.com/.attachments/3245c726ee77d73702ba8c3310639727/f000842b/product-5.jpg","price":9.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]
```

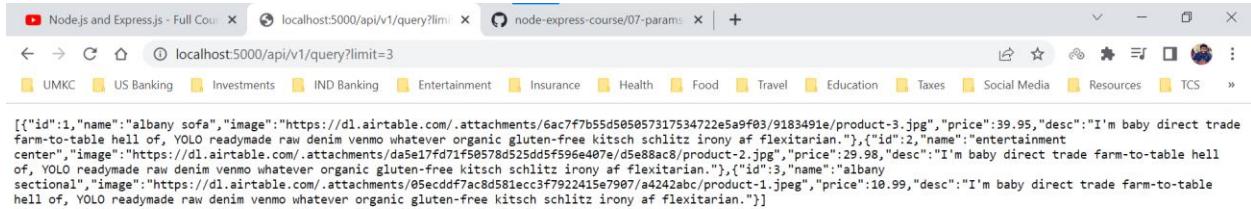


The screenshot shows the Network tab in Chrome DevTools. A single request labeled "query" is listed. The Headers section shows a Content-Type header of "application/json". The Response section shows the same JSON array as the previous screenshot.

Network

Name	Headers	Preview	Response	Initiator	Timing	
query	Content-Type: application/json		[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":2,"name":"entertainment center","image":"https://dl.airtable.com/.attachments/dae517fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg","price":29.98,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":3,"name":"albany sectional","image":"https://dl.airtable.com/.attachments/05ecddf7ac8d81ecc3f7922415e7907/44242abc/product-1.jpeg","price":10.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":4,"name":"leather sofa","image":"https://dl.airtable.com/.attachments/3245c726ee77d73702ba8c3310639727/f000842b/product-5.jpg","price":9.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]			

## http://localhost:5000/api/v1/query?limit=3 - With only limit query parameter



The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Cou", "localhost:5000/api/v1/query?limit=3", and "node-express-course/07-params". The active tab displays a JSON array of three sofa objects, starting from the first one. The descriptions are identical, referencing YOLO ready-made sofas.

```
[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":2,"name":"entertainment center","image":"https://dl.airtable.com/.attachments/dae517fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg","price":29.98,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":3,"name":"albany sectional","image":"https://dl.airtable.com/.attachments/05ecddf7ac8d81ecc3f7922415e7907/44242abc/product-1.jpeg","price":10.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]
```



The screenshot shows the Network tab in Chrome DevTools. A single request labeled "query?limit=3" is listed. The Headers section shows a Content-Type header of "application/json". The Response section shows the same JSON array as the previous screenshots, limited to three items.

Network

Name	Headers	Payload	Preview	Response	Initiator	Timing	
query?limit=3	Content-Type: application/json			[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":2,"name":"entertainment center","image":"https://dl.airtable.com/.attachments/dae517fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg","price":29.98,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":3,"name":"albany sectional","image":"https://dl.airtable.com/.attachments/05ecddf7ac8d81ecc3f7922415e7907/44242abc/product-1.jpeg","price":10.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]			

<http://localhost:5000/api/v1/query?search=a> – With only search query parameter

The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "localhost:5000/api/v1/query?search=a", and "node-express-course/07-params". The active tab displays the JSON response to the search query. The response is an array containing two objects: one for "albany sofa" and one for "albany sectional". Both items have their original descriptions and images removed.

```
[{"id":1,"name":"albany sofa","image":"https://d1.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":3,"name":"albany sectional","image":"https://d1.airtable.com/.attachments/05ecdddf7ac8d581ecc3f7922415e7907/a4242abc/product-1.jpeg","price":10.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]
```

Below the browser window is a screenshot of the Chrome Network tab. It shows a single request labeled "query?search=a". The "Payload" section of the Network tab displays the same JSON response as the browser.

<http://localhost:5000/api/v1/query?search=a&limit=1> – With both search and limit query parameters

The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "localhost:5000/api/v1/query?search=a&limit=1", and "node-express-course/07-params". The active tab displays the JSON response to the search and limit query. The response is an array containing only one object: the "albany sofa" item. The "albany sectional" item is omitted due to the limit of 1.

```
[{"id":1,"name":"albany sofa","image":"https://d1.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]
```

Below the browser window is a screenshot of the Chrome Network tab. It shows a single request labeled "query?search=a&limit=1". The "Payload" section of the Network tab displays the JSON response with only one item.

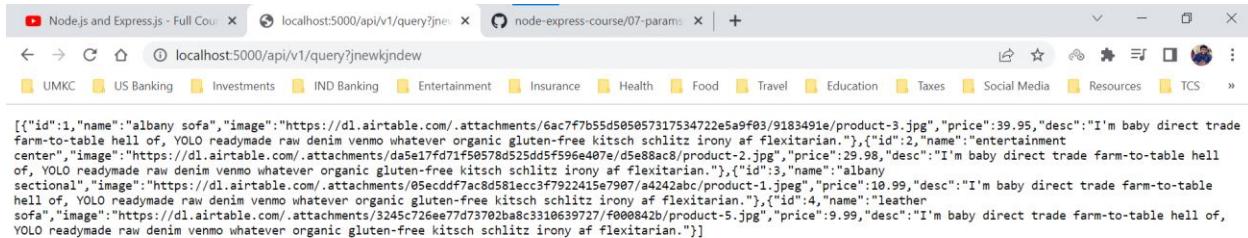
<http://localhost:5000/api/v1/query?search=jhgjhg> – with unavailable search parameter

The screenshot shows the Network tab in the Chrome DevTools developer tools. The URL bar at the top shows the request: `localhost:5000/api/v1/query?search=jhgjhg`. The Network tab is selected, displaying a single request entry. The request name is `query?search=jhgjhg`. The preview pane shows the response body as empty, with the message `No properties`. At the bottom of the Network tab, it indicates `1 requests | 235 B transferred | 2 B resources | Finish`.

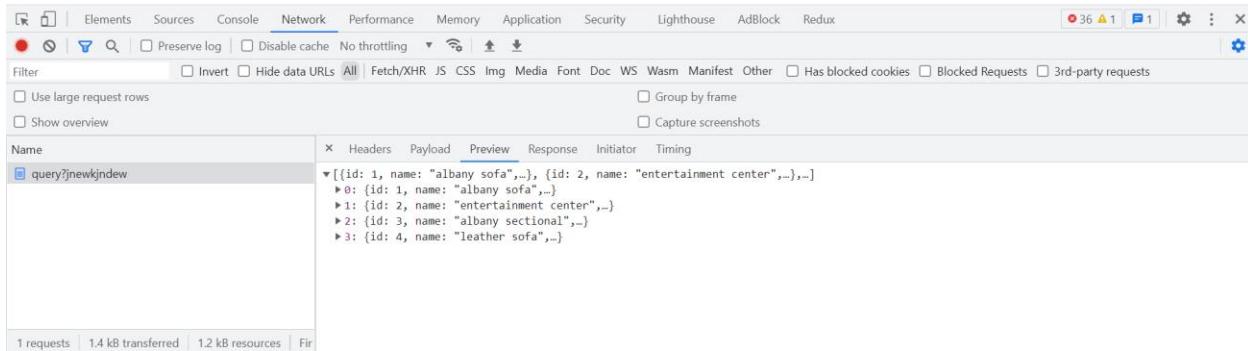
<http://localhost:5000/api/v1/query?limit=abc> – With unavailable/ wrongfule limit parameter

The screenshot shows the Network tab in the Chrome DevTools developer tools. The URL bar at the top shows the request: `localhost:5000/api/v1/query?limit=abc`. The Network tab is selected, displaying a single request entry. The request name is `query?limit=abc`. The preview pane shows the response body as empty, with the message `No properties`. At the bottom of the Network tab, it indicates `1 requests | 235 B transferred | 2 B resources | Finish`.

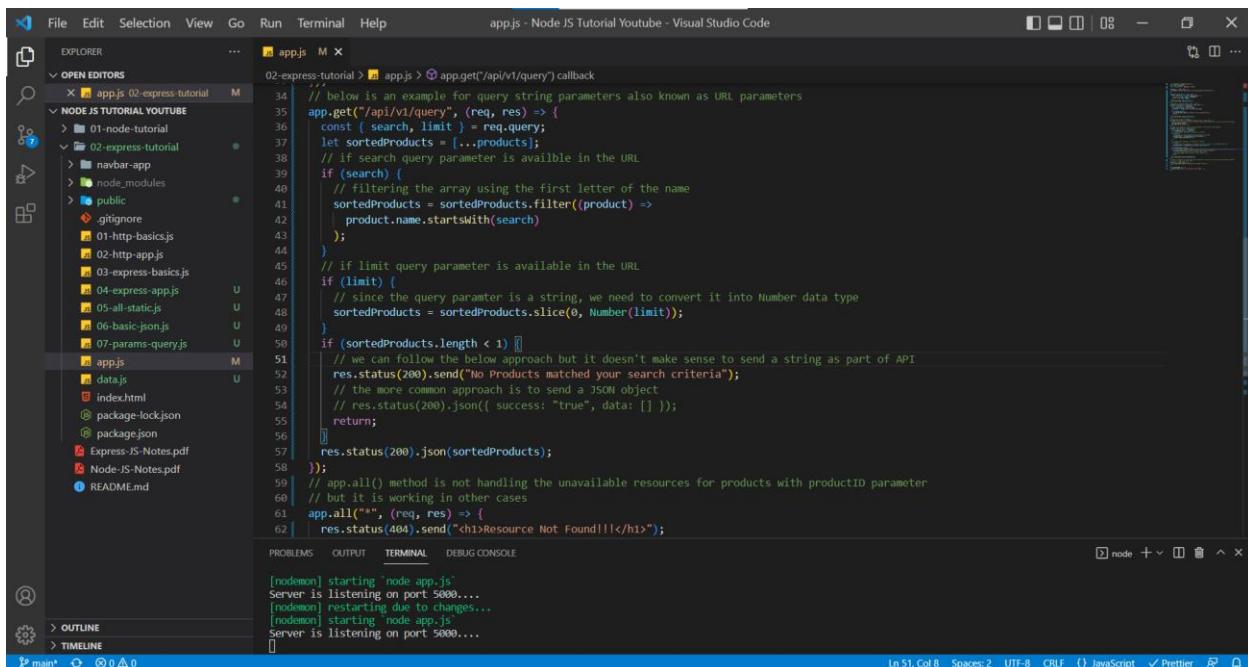
<http://localhost:5000/api/v1/query?jnewkjndew> - with random value after question mark(?)



```
[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b5d50505731753472e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":2,"name":"entertainment center","image":"https://dl.airtable.com/.attachments/dae517fd71f50578d525dd5f596a407e/d5e88ac8/product-2.jpg","price":29.98,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":3,"name":"albany sectional","image":"https://dl.airtable.com/.attachments/05ecddfb7ac8d581ecc3f7922415e7907/44242abc/product-1.jpeg","price":10.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"id":4,"name":"leather sofa","image":"https://dl.airtable.com/.attachments/3245c726ee77d73702ba8c3310639727/f008842b/product-5.jpg","price":9.99,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}]
```



We can handle the display of empty array by adding a small piece of code. We can check the length of the array we are sending as a response.



```
app.get('/api/v1/query', (req, res) => {
  const { search, limit } = req.query;
  let sortedProducts = [...products];
  if (search) {
    sortedProducts = sortedProducts.filter((product) =>
      product.name.startsWith(search)
    );
  }
  if (limit) {
    // since the query parameter is a string, we need to convert it into Number data type
    sortedProducts = sortedProducts.slice(0, Number(limit));
  }
  if (sortedProducts.length < 1) {
    // we can follow the below approach but it doesn't make sense to send a string as part of API
    res.status(200).send("No Products matched your search criteria");
    // the more common approach is to send a JSON object
    // res.status(200).json({ success: "true", data: [] });
    return;
  }
  res.status(200).json(sortedProducts);
});
// app.all() method is not handling the unavailable resources for products with productId parameter
// but it is working in other cases
app.all("*", (req, res) => {
  res.status(404).send("404 Resource Not Found!!</h1>");
});
```

<http://localhost:5000/api/v1/query?search=b> – when no products match the criteria

The screenshot shows a browser window with three tabs open:

- Node.js and Express.js - Full Course
- localhost:5000/api/v1/query?search=b
- node-express-course/07-params

The second tab displays the search results for 'query?search=b'. Below the tabs is a navigation bar with various categories like UMKC, US Banking, Investments, etc. The main content area says "No Product matched your search criteria".

The screenshot shows the Network tab in the Chrome DevTools developer console. It lists one request: "query?search=b". The response body contains the text "No Product matched your search criteria".

The more common approach is to send a JSON object even if we couldn't find the data for the request. We do follow this format because our request was successful but we don't have any data.

The screenshot shows the Visual Studio Code interface with the file "app.js" open. The code handles query parameters for a search function:

```
02-express-tutorial > app.js > ↗ app.get('/api/v1/query') callback
 34 // below is an example for query string parameters also known as URL parameters
 35 app.get('/api/v1/query', (req, res) => {
 36   const { search, limit } = req.query;
 37   let sortedProducts = [...products];
 38   // if search query parameter is available in the URL
 39   if (search) {
 40     // filtering the array using the first letter of the name
 41     sortedProducts = sortedProducts.filter((product) =>
 42       product.name.startsWith(search)
 43     );
 44   }
 45   // if limit query parameter is available in the URL
 46   if (limit) {
 47     // since the query parameter is a string, we need to convert it into Number data type
 48     sortedProducts = sortedProducts.slice(0, Number(limit));
 49   }
 50   if (sortedProducts.length < 1) {
 51     // we can follow the below approach but it doesn't make sense to send a string as part of API
 52     res.status(200).send("No Products matched your search criteria");
 53     // the more common approach is to send a JSON object
 54     res.status(200).json({ success: "true", data: [] });
 55     return;
 56   }
 57   res.status(200).json(sortedProducts);
 58 }
 59 // app.all() method is not handling the unavailable resources for products with productID parameter
 60 // but it is working in other cases
 61 app.all("*", (req, res) => {
 62   res.status(404).send("<h1>Resource Not Found!!!</h1>");
 63 })
```

The terminal at the bottom shows the output of running the application with node app.js, indicating the server is listening on port 5000.

<http://localhost:5000/api/v1/query?search=b> - when no products match the criteria

The screenshot shows a browser window with three tabs: 'Node.js and Express.js - Full Course' (closed), 'localhost:5000/api/v1/query?search=b', and 'node-express-course/07-params' (closed). The active tab displays the URL 'localhost:5000/api/v1/query?search=b'. Below the address bar is a navigation bar with various categories: UMKC, US Banking, Investments, IND Banking, Entertainment, Insurance, Health, Food, Travel, Education, Taxes, Social Media, Resources, and TCS. The main content area shows the JSON response: {"success": "true", "data": []}.

The screenshot shows the Network tab in Chrome DevTools. It lists a single request: 'query?search=b'. The 'Preview' tab is selected, showing the JSON response: {"success": "true", "data": []}. Other tabs include Headers, Payload, Response, Initiator, and Timing. At the bottom of the Network tab, it says '1 requests | 263 B transferred | 28 B resources | Finis'.

## Additional Params and Query Info

We may have a doubt that why are we returning the `res.status().json()` in the if condition where we don't find any items using the search criteria.

In JavaScript if we don't explicitly return then javascript keeps reading the code, so if we emit the return, we get a server error where we send back one response and javascript just keeps reading the code and the express is confused because it already sent a response so why we are sending another response in the same request.

So basically, we cannot send two responses for the same request one after the another. Yes, we can send one response based on the conditions, but we can't send one after another in the same request.

## app.js

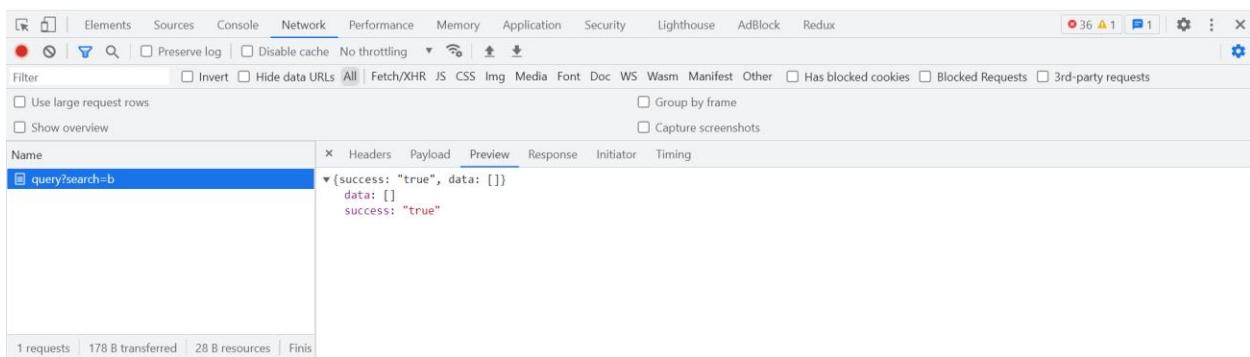
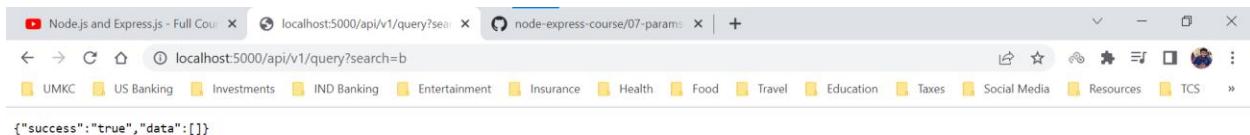
```

File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code
OPEN EDITORS app.js M ×
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
navbar-app
node_modules
public
.gitignore
01-http-basics.js
02-http-app.js
03-express-basics.js
04-express-app.js
05-all-static.js
06-basic-json.js
07-params-query.js
app.js M
data.js U
index.html
package-lock.json
package.json
Express-JS-Notes.pdf
Node-JS-Notes.pdf
README.md
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ln 54, Col 57 Spaces: 2 UTF-8 CHRF ⓘ JavaScript ✓ Prettier ⌂ ⌂
(node) starting 'node app.js'
Server is listening on port 5000...
(node) restarting due to changes...
(node) starting 'node app.js'
Server is listening on port 5000...
[]

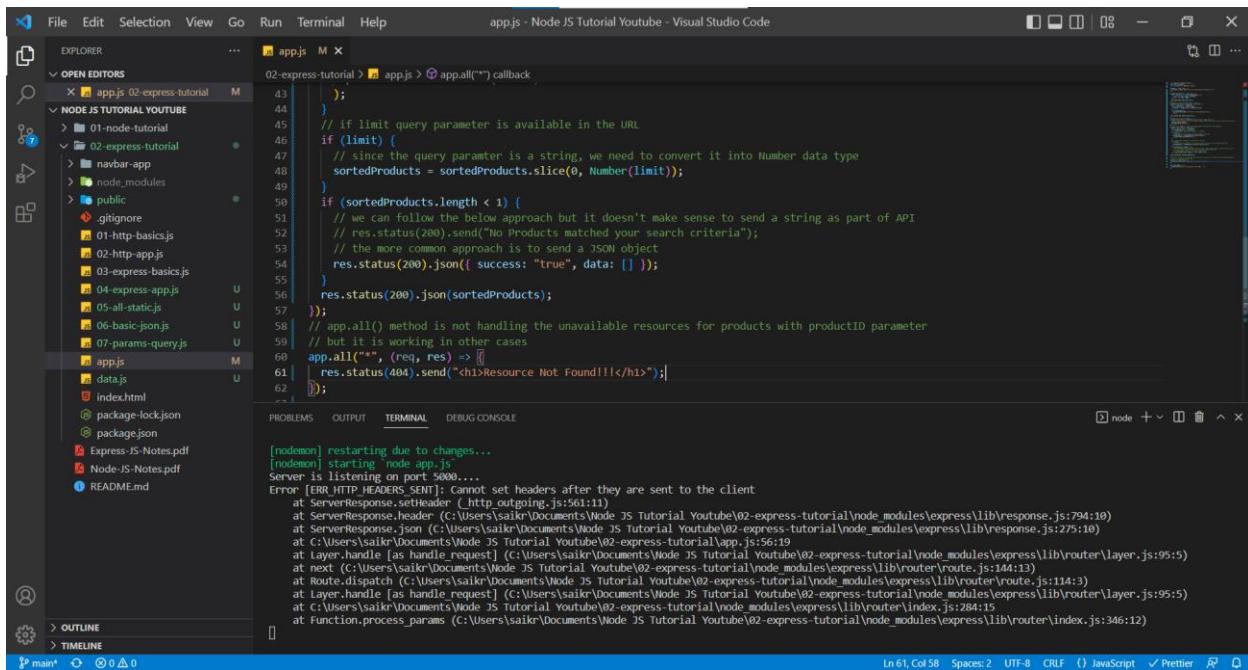
HTTP://localhost:5000/api/v1/query?search=b - when no products match the criteria

```

HTTP://localhost:5000/api/v1/query?search=b - when no products match the criteria



But at the once the request comes in at the same time our server breaks down in the terminal.



The screenshot shows the Visual Studio Code interface with the file `app.js` open in the editor. The code is a Node.js application using Express. A terminal window at the bottom shows an error from nodemon:

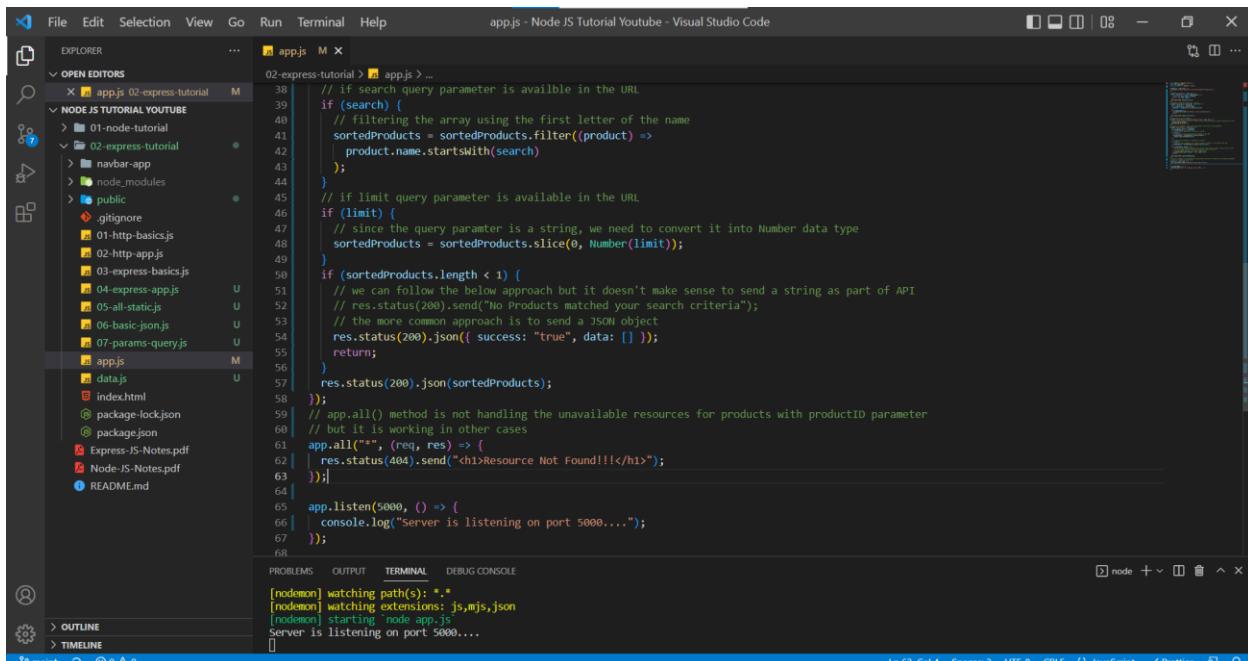
```
[nodemon] restarting due to changes...
[nodemon] starting node app.js
server is listening on port 5000...
Error [ERR_INVALID_HEADERS]: cannot set headers after they are sent to the client
at ServerResponse.setHeader (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\response.js:794:10)
at ServerResponse.header (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\response.js:275:10)
at C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\app.js:56:19
at Layer.handle [as handle_request] (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\router\layer.js:95:5)
at next (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\router\route.js:144:13)
at Route.dispatch (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\router\route.js:114:3)
at Layer.handle [as handle_request] (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\router\layer.js:95:5)
at C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\router\index.js:284:15
at Function.process_params (C:\Users\saikr\Documents\Node JS Tutorial\Youtube\02-express-tutorial\node_modules\express\lib\router\index.js:346:12)
```

The code in `app.js` contains several return statements within the `app.all` block, which is causing the error. The problematic part of the code is:

```
    if (sortedProducts.length < 1) {
      // we can follow the below approach but it doesn't make sense to send a string as part of API
      // res.status(200).send("No Products matched your search criteria");
      // the more common approach is to send a JSON object
      res.status(200).json({ success: "true", data: [] });
    }
    res.status(200).json(sortedProducts);
  });
}
```

So ,we can only one response per request. In order to avoid the above mishap, we always go with the return statement inside the explicit conditions.

### app.js



The screenshot shows the Visual Studio Code interface with the file `app.js` open in the editor. The code has been modified to ensure a single response per request. The terminal window at the bottom shows the server is now listening on port 5000 without errors.

```
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node app.js'
Server is listening on port 5000...
```

The corrected code in `app.js` is:

```
// if search query parameter is available in the URL
if (search) {
  // filtering the array using the first letter of the name
  sortedProducts = sortedProducts.filter((product) =>
    product.name.startsWith(search)
  );
}
// if limit query parameter is available in the URL
if (limit) {
  // since the query paramter is a string, we need to convert it into Number data type
  sortedProducts = sortedProducts.slice(0, Number(limit));
}
if (sortedProducts.length < 1) {
  // we can follow the below approach but it doesn't make sense to send a string as part of API
  // res.status(200).send("No Products matched your search criteria");
  // the more common approach is to send a JSON object
  res.status(200).json({ success: "true", data: [] });
  return;
}
res.status(200).json(sortedProducts);
});
// app.all() method is not handling the unavailable resources for products with productId parameter
// but it's working in other cases
app.all("*", (req, res) => {
  res.status(404).send("Resource Not Found!!!");
});
app.listen(5000, () => {
  console.log("Server is listening on port 5000...");
});
```

## Middleware – Setup

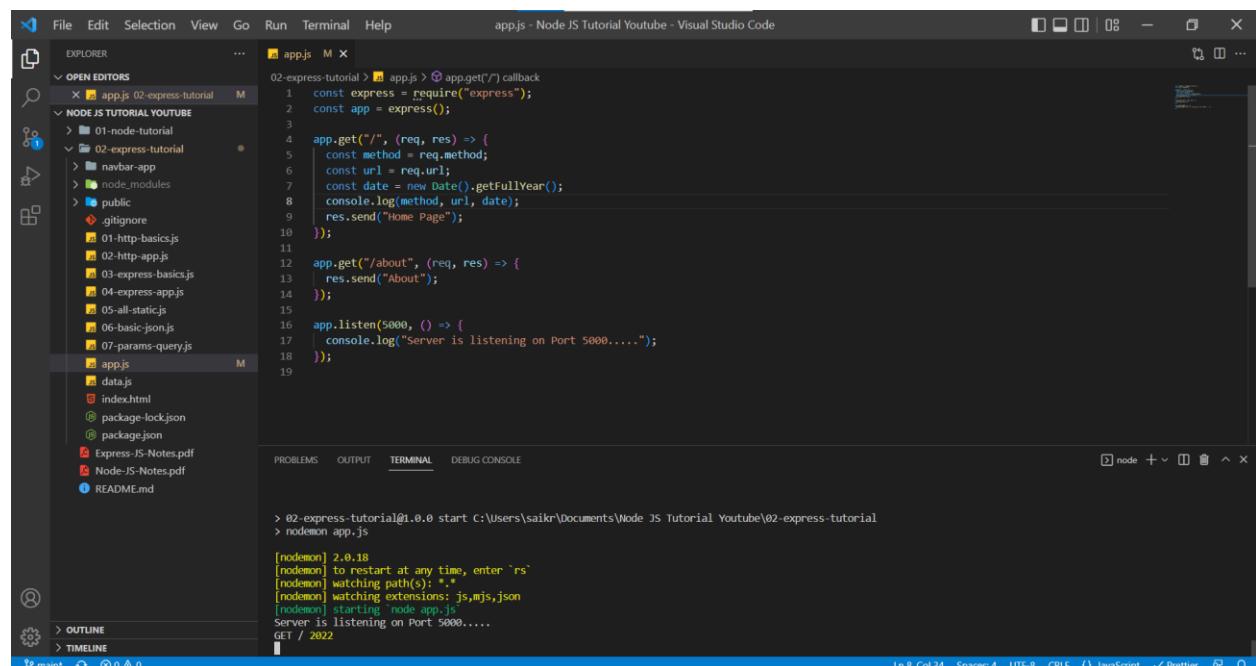
Express Middleware are functions that execute during the request to the server. Each middleware function has access to request and response objects and when it comes to functionality (literally sky is the limit).

Middleware is everywhere in Express. Express is nothing but a bunch of middleware functions stuffed together to make one nice express cake or desert. It is heart and soul of the express.

Middleware sits between request and response. When the request comes in, we do some functionality and then send the response.

In this scenario, we are going to have a two routes namely Home Page and About Page.

### app.js



```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

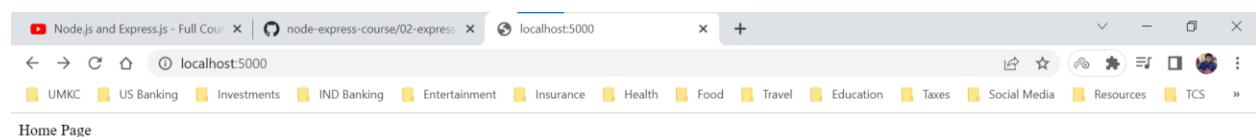
EXPLORER app.js M X
OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
  02-navbar-app
  node_modules
  public
  .gitignore
    01-http-basics.js
    02-http-app.js
    03-express-basics.js
    04-express-app.js
    05-all-static.js
    06-basic-json.js
    07-params-query.js
    app.js M
    data.js
    index.html
    package-lock.json
    package.json
    Express-JS-Notes.pdf
    Node-JS-Notes.pdf
    README.md

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
node + v 🏛 ^ x

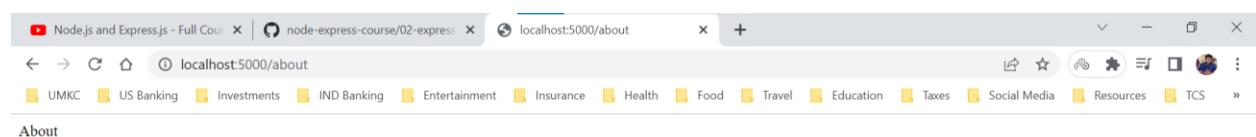
> 02-express-tutorial@1.0.0 start C:\Users\saikr\Documents\Node JS Tutorial Youtube\02-express-tutorial
> nodemon app.js

[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server is listening on Port 5000.....
GET / 2022
Ln 8, Col 34 Spaces: 4 UFT-8 CRLF ( ) JavaScript ✓ Prettier 🏛
```

localhost:5000/

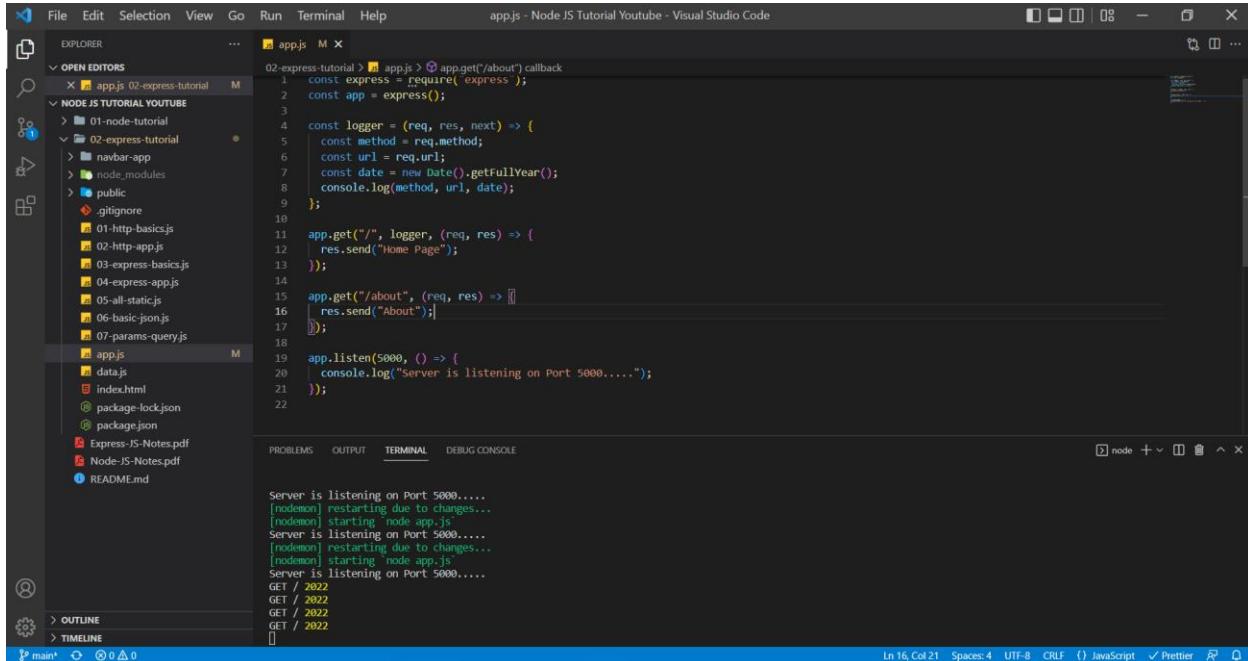


localhost:5000/about



Assume, in this scenario we want to have the same logic of displaying method, URL and full year we can write the same logic for each URL request. To solve this issue, we can create function and write the logic over there and use that function in each URL request.

### app.js



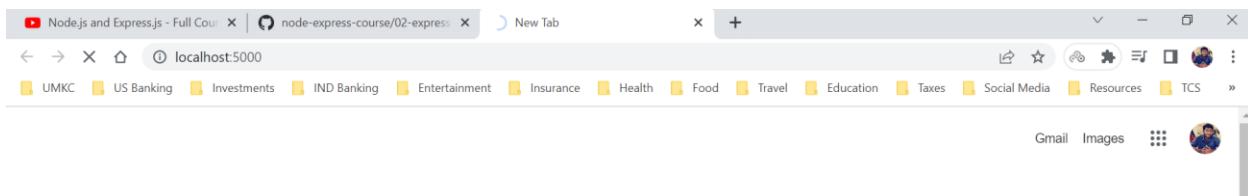
```

File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS 02-express-tutorial > app.js > app.get("/about") callback
NODE JS TUTORIAL YOUTUBE
  01-node-tutorial
  02-express-tutorial
    navbar-app
    node_modules
    public
      .gitignore
      01-http-basics.js
      02-http-app.js
      03-express-basics.js
      04-express-app.js
      05-all-static.js
      06-basic-json.js
      07-params-query.js
    app.js M
    data.js
    index.html
    package-lock.json
    package.json
    Express-JS-Notes.pdf
    Node-JS-Notes.pdf
    README.md
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
node + v ^ ^ x
Ln 16, Col 21 Spaces: 4 UTF-8 CRLF () JavaScript ✓ Prettier R D
@ OUTLINE > TIMELINE
@ main* ○ □ 0 0 0 0
Server is listening on Port 5000.....
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on Port 5000.....
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on Port 5000.....
GET / 2022
GET / 2022
GET / 2022
GET / 2022

```

We created a function named logger and shifted the logic into that function. We are using that function as parameter in the app.get() method.

localhost:5000/

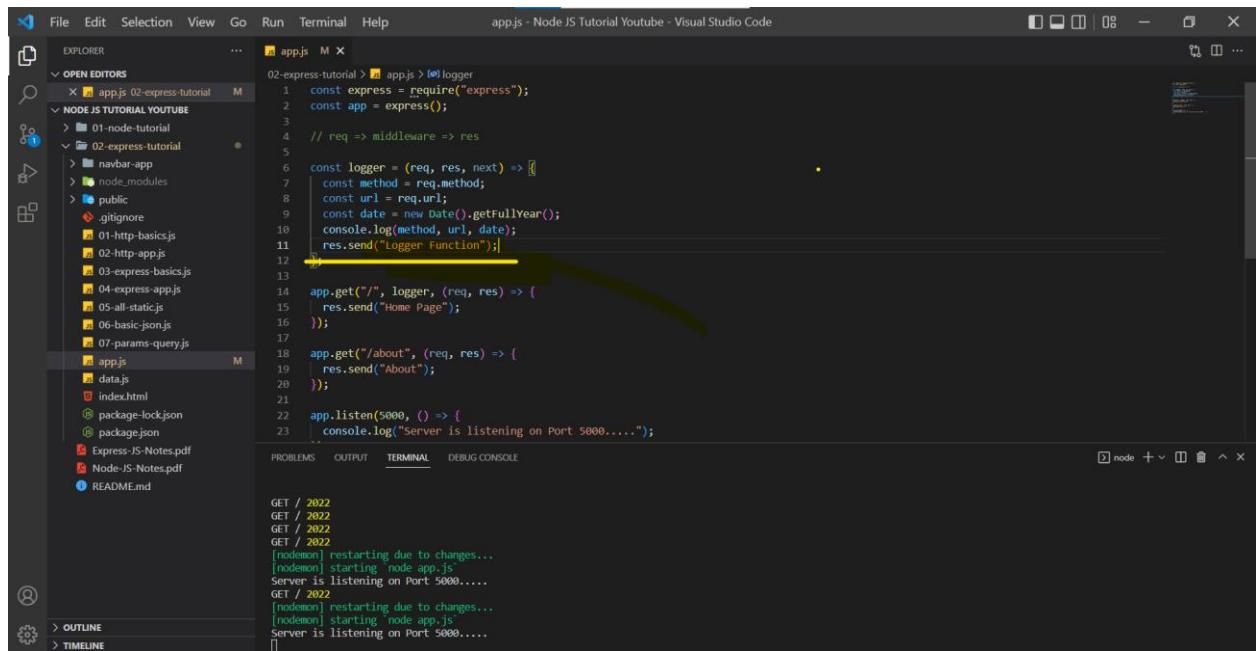


But here we can see the page is still loading and not providing the data. This is because Logger function that we are using is not terminating the response or it is not passing to next middleware in the server.

This issue can be solved in two ways

1. Terminate the response in the function (logger function) itself by sending response object from that function.

## app.js



```
const express = require("express");
const app = express();

// req -> middleware -> res

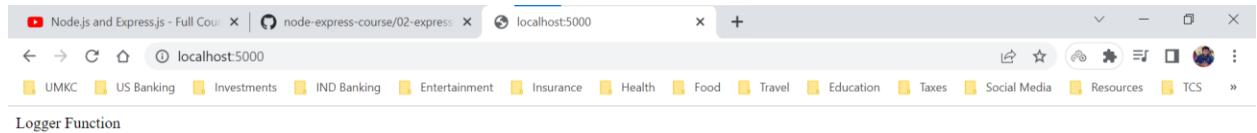
const logger = (req, res, next) => {
  const method = req.method;
  const url = req.url;
  const date = new Date().getFullYear();
  console.log(method, url, date);
  res.send("Logger Function");
}

app.get("/", logger, (req, res) => {
  res.send("Home Page");
});

app.get("/about", (req, res) => {
  res.send("About");
});

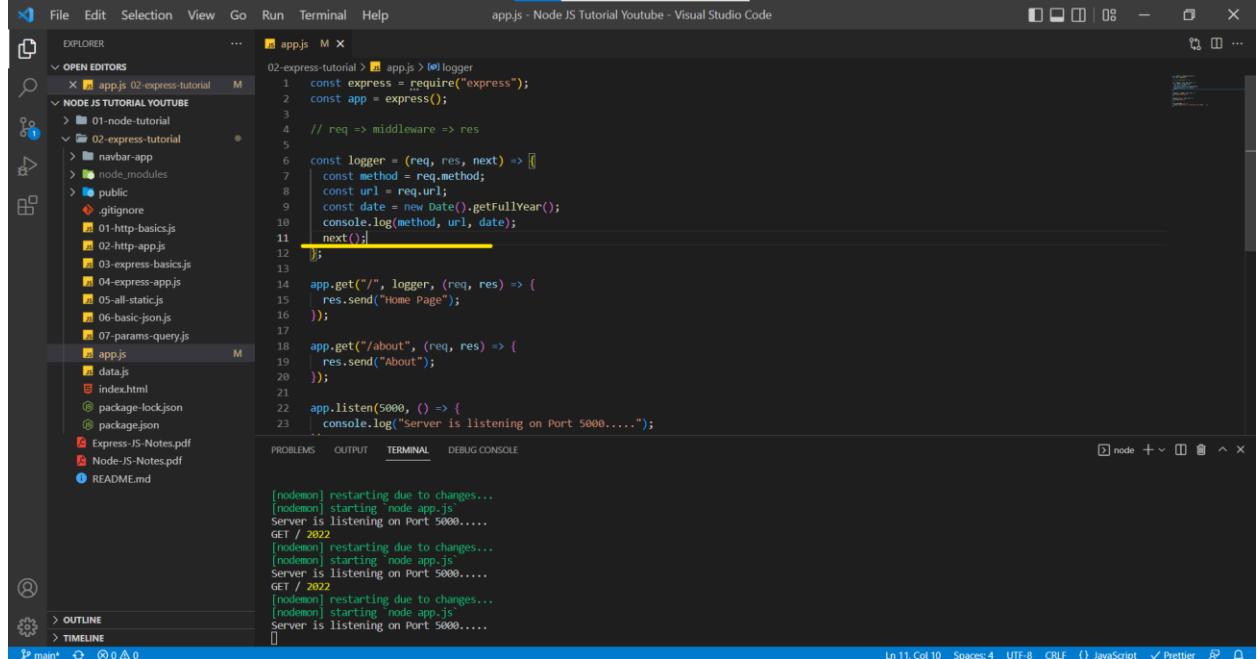
app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
})
```

localhost:5000/



2. We execute the function (logger function) and can pass it over to next middleware (in this case it would be `app.get("/")` method since it is using the logger function as parameter)

app.js



```
const express = require("express");
const app = express();

// req -> middleware -> res

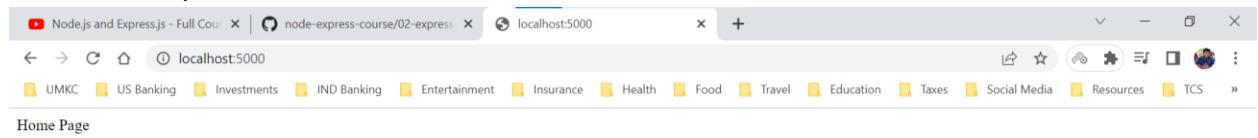
const logger = (req, res, next) => {
  const method = req.method;
  const url = req.url;
  const date = new Date().getFullYear();
  console.log(method, url, date);
  next();
}

app.get("/", logger, (req, res) => {
  res.send("Home Page");
});

app.get("/about", (req, res) => {
  res.send("About");
});

app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
})
```

localhost:5000/



We can use the same logger function for any request from the user.

app.js

The screenshot shows the Visual Studio Code interface with the file 'app.js' open. The code defines a logger middleware function that logs the method, URL, and date for each request. It then uses this logger in the routes for '/' and '/about'. The code also starts the server on port 5000. The terminal below shows the server listening on port 5000 and log messages for requests to '/' and '/about'.

```
const express = require("express");
const app = express();

// req => middleware => res

const logger = (req, res, next) => {
  const method = req.method;
  const url = req.url;
  const date = new Date().getFullYear();
  console.log(method, url, date);
  next();
}

app.get("/", logger, (req, res) => {
  res.send(`Home Page`);
});

app.get("/about", logger, (req, res) => [
  res.send("About")
]);

app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
});
```

TERMINAL

```
Server is listening on Port 5000....  
GET / 2022  
[nodemon] restarting due to changes...  
[nodemon] starting 'node app.js'  
Server is listening on Port 5000....  
[nodemon] restarting due to changes...  
[nodemon] starting 'node app.js'  
Server is listening on Port 5000....
```

**Express provides request and response object to logger function or any other function in the server.**  
**Logger function written above is the first middleware function we have ever written.**

### APP.USE Method

There are two issues with the current setup. The logger function is the first middleware function we have ever written.

1. Our app.js is getting clunky because we have logger function and other requests in one page. (We can have a logger function in a separate file). This approach will keep our app.js file lean.
2. Assume a scenario where we 50 more routes and we don't want to add this logger function to each route manually. (It would be nicer if there would be a method that just adds my middleware function to any route).

To solve the first problem, we create a separate javascript file and post the logger middleware function code over there and export the code through ***module.exports*** and in the main application we import it using the ***require*** global variable.

## app.js

```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS
... app.js M x logger.js U
02-express-tutorial > app.js > ...
1 const express = require('express');
2 const app = express();
3 const logger = require('./logger');
4 // req, res, next
5
6 app.get('/', logger, (req, res) => {
7   res.send("Home Page");
8 });
9
10 app.get('/about', logger, (req, res) => {
11   res.send("About");
12 });
13
14 app.listen(5000, () => {
15   console.log("Server is listening on Port 5000....");
16 });
17
18
|_

```

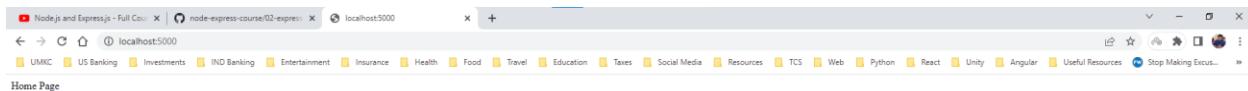
## logger.js

```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS
... app.js M x logger.js U
02-express-tutorial > logger.js > ...
1 const logger = (req, res, next) => {
2   const method = req.method;
3   const url = req.url;
4   const date = new Date().getFullYear();
5   console.log(method, url, date);
6   next();
7 };
8
9 module.exports = logger;
10
11
|_

```

localhost:5000/

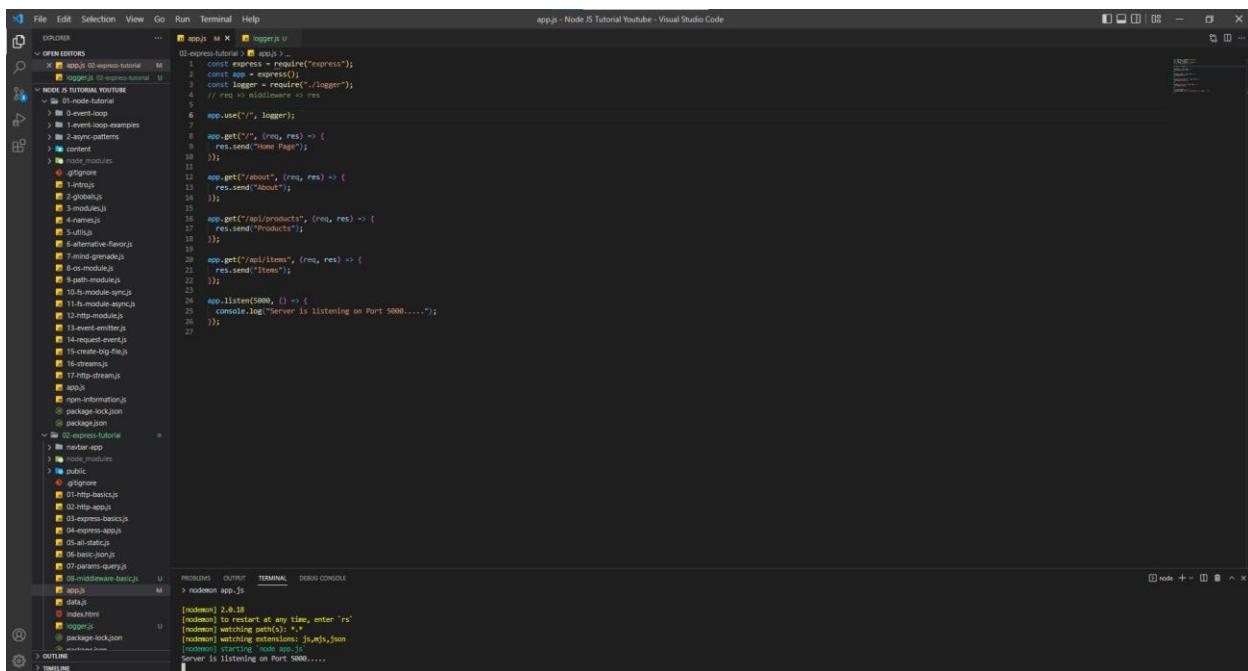


localhost:5000/about

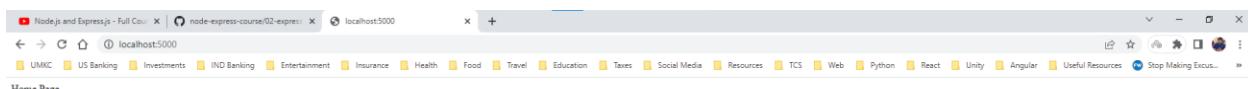


The second problem can be solved by using the `app.use()` method which we applied to all the requests the server receives. If we add `app.use()` method at top of the requests, then it will add to all the request but if it is added after one request then it won't apply for the top request, but it will apply for the bottom requests.

app.js



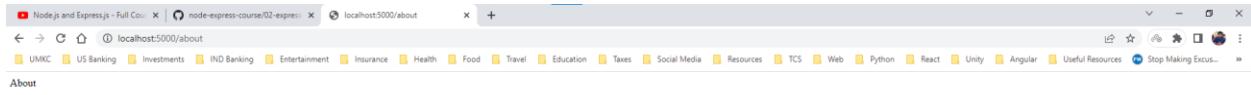
localhost:5000/



Terminal (Here we get `console.log` statement from the logger middleware function)



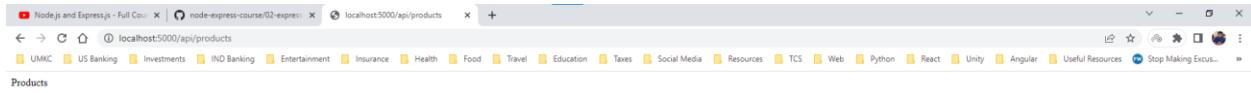
localhost:5000/about



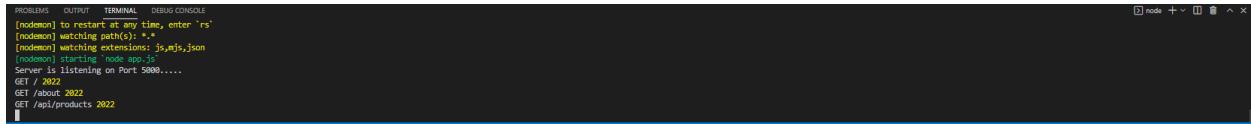
Terminal



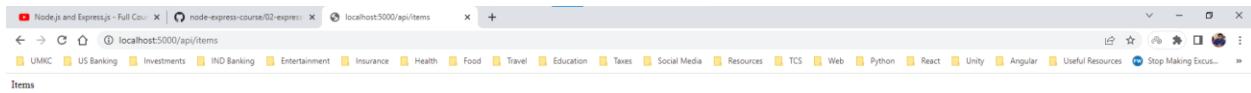
localhost:5000/api/products



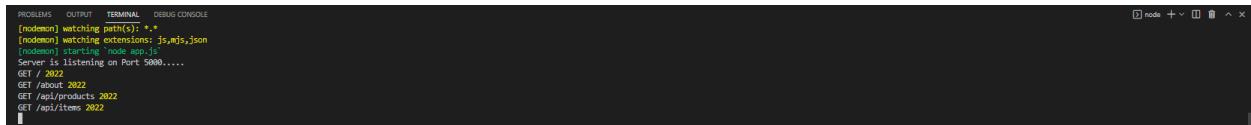
Terminal



localhost:5000/api/items



Terminal



We can also add the parameters of the URL path to which the app.use() method can be applied, and it contains a callback function having request and response object and we need to terminate the request by sending the request by response else browser waits for response and it will be in a loading status.

Now we have mentioned **/api** in the parameters of app/use() method it will provide logger middle function to the request containing the URL **/api**

## app.js

The screenshot shows the Visual Studio Code interface with the file 'app.js' open. The code defines an Express application with various routes. A logger middleware function is used to log each request to the terminal.

```
const express = require('express');
const logger = require('./logger');

const app = express();

app.use('/api', logger);

app.get('/', (req, res) => {
  res.send("Home Page");
});

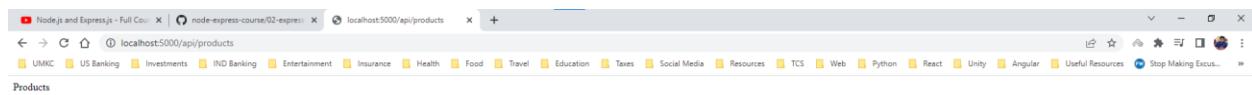
app.get('/about', (req, res) => {
  res.send("About");
});

app.get('/api/products', (req, res) => {
  res.send("Products");
});

app.get('/api/items', (req, res) => {
  res.send("Items");
});

app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
});
```

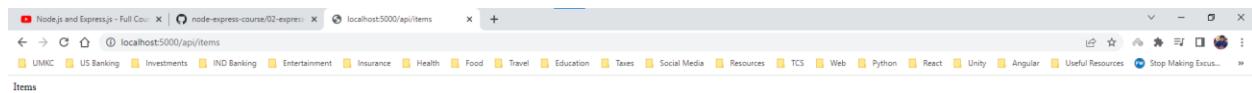
<http://localhost:5000/api/products>



Terminal ( We can see the console log statement of the logger middleware function)

The terminal window shows the server's log output. It includes the header 'Server is listening on Port 5000....' and three log entries from the terminal: 'GET / 2022', 'GET /api/products 2022', and 'GET /api/items 2022'. The last entry also includes the message '[nodemon] restarting due to changes...'.

<http://localhost:5000/api/items>

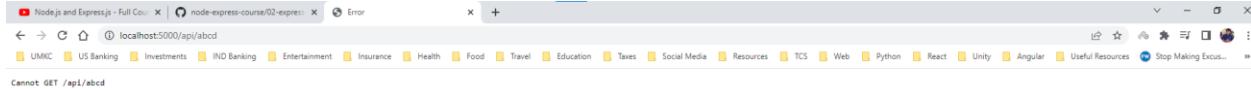


Terminal

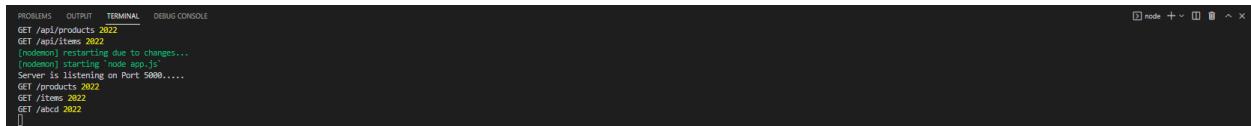
The terminal window shows the server's log output. It includes the header 'Server is listening on Port 5000....' and three log entries from the terminal: 'GET /about 2022', 'GET /api/products 2022', and 'GET /api/items 2022'. The last entry also includes the message '[nodemon] restarting due to changes...'.

Even though `http://localhost:5000/api/abcd` doesn't have a resource in our server but still we console log the statement because we are using `app.use("/api")` hence it will apply for all URL requests starting with /api.

`http://localhost:5000/api/abcd`

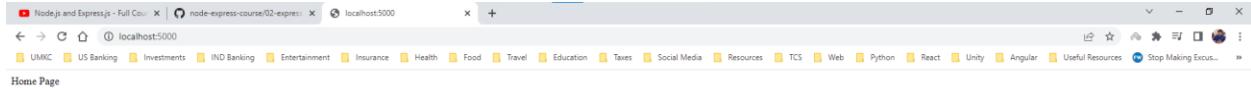


Terminal (We will console log the information)

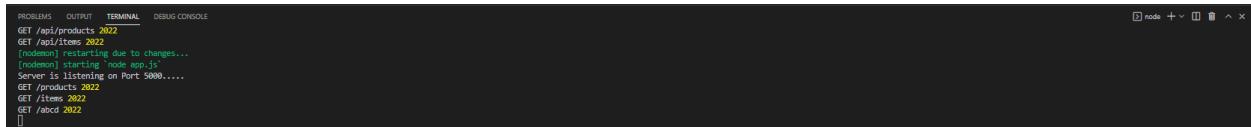


But if we try to use the home Page route, we will send the response to the server, but we won't be console logging the information because the logger middleware function won't be available for requests whose URL doesn't contain /api.

`http://localhost:5000/`



Terminal (We won't see the console log information)



`app.use()` method can also have the request and response object but if we are using the request and response object, we need to handle them as well.

## app.js

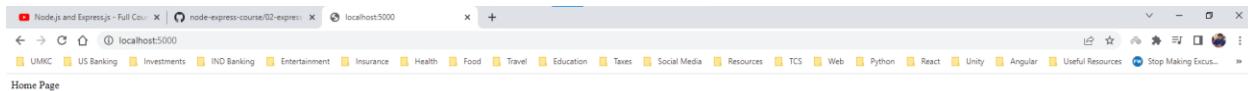
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure for "02-express-tutorial". The "02-express-tutorial" folder contains files like "index.js", "logger.js", "public", "routes", and "utils".
- Code Editor:** The main editor window displays the "app.js" file content.
- Terminal:** The bottom terminal window shows the command "node app.js" being run, followed by the output of the "nodemon" process starting the server on port 5000.

```
const express = require('express');
const app = express();
const logger = require('../logger');

app.use(logger);
app.get('/', (req, res) => {
    console.log(req.url);
    res.send('Home Page');
});
app.get('/about', (req, res) => {
    res.send('About');
});
app.get('/api/products', (req, res) => {
    res.send('Products');
});
app.get('/api/items', (req, res) => {
    res.send('Items');
});
app.listen(5000, () => {
    console.log('Server is listening on Port 5000....');
});
```

<http://localhost:5000/>



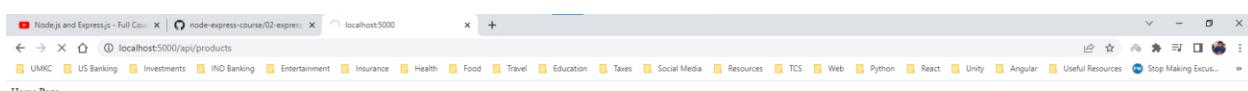
We won't see anything in terminal because logger middleware is not applied to Home and About routes.

The terminal window shows the command "node app.js" being run, followed by the output of the "nodemon" process starting the server on port 5000.

```
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node app.js'
Server is listening on Port 5000....
```

Since we are not sending the response in `app.get()` method, browser will wait response and it will be loading state itself.

<http://localhost:5000/api/products>



## Terminal

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
> nodemon app.js
[nodemon] 2.0.18
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node app.js
Server is listening on Port 5000....
GET /products 2022
/products

```

We can solve this problem sending the response, but it will override the other responses.

## app.js

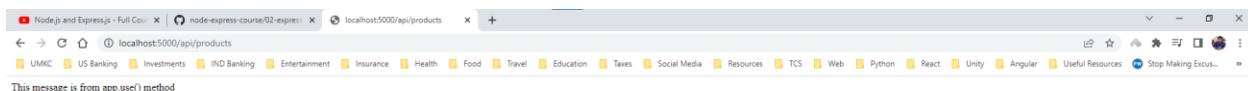
```

File Edit Selection View Go Run Terminal Help
OPEN FOLDERS
... 02-express-tutorial M 02-express-tutorial M
  app.js 02-express-tutorial U
  logger.js 02-express-tutorial U
  ...
  NODE JS TUTORIAL YOUTUBE
    01-node-tutorial
      01-express-tutorial
        app.js
        index.html
        logger.js
        package-lock.json
        package.json
        README.md
      02-express-tutorial
        ...
        logger.js
        package-lock.json
        package.json
        README.md
      03-middleware-basic.js
      04-middleware-advanced.js
      05-apis.js
      06-basic.js
      07-params-query.js
      08-middleware-advanced.js
      app.js
      logger.js
      index.html
      logger.js
      package-lock.json
      package.json
      Express.js-Notes.pdf
      Node.js-Notes.pdf
      README.md

app.js - Node JS Tutorial Youtube - Visual Studio Code
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node app.js
Server is listening on Port 5000...
GET /products 2022
/products
[nodemon] restarting due to changes...
[nodemon] starting node app.js
Server is listening on Port 5000...
GET /products 2022
/products

```

<http://localhost:5000/api/products>



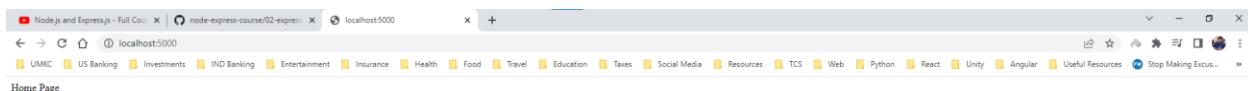
## Terminal

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node app.js
Server is listening on Port 5000...
GET /products 2022
/products
[nodemon] restarting due to changes...
[nodemon] starting node app.js
Server is listening on Port 5000...
GET /products 2022
/products

```

<http://localhost:5000/> (app.use method can't override the home route since it isn't being applied to Home and About route)



## Multiple Middleware Functions

In the earlier scenario, we have seen only 1 middleware function being written but sometimes we need multiple middleware functions which needs to apply to the requests that coming into the server.

Let us create another middleware function named authorize.

authorize.js

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

- File Explorer:** Shows files in the "02-express-tutorial" folder, including "app.js", "logger.js", and "authorize.js".
- Code Editor:** The "authorize.js" file is open, containing the following code:

```
const authorize = (req, res, next) => {
  console.log("authorize");
  next();
};

module.exports = authorize;
```

- Terminal:** The terminal shows output from nodemon:

```
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server is listening on Port 5000....
```

app.js

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

- File Explorer:** Shows files in the "02-express-tutorial" folder, including "app.js", "logger.js", and "authorize.js".
- Code Editor:** The "app.js" file is open, containing the following code:

```
const express = require("express");
const app = express();
const logger = require("./logger");
const authorize = require("./authorize");

app.use(logger);
app.get("/", (req, res) => {
  res.send("Home Page");
});

app.get("/about", (req, res) => {
  res.send("About");
});

app.get("/api/products", (req, res) => {
  res.send("Products");
});

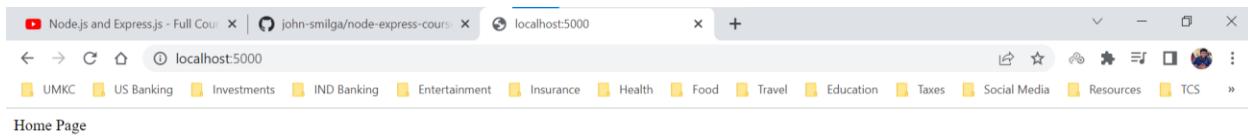
app.get("/api/items", (req, res) => {
  res.send("Items");
});

app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
});
```

- Terminal:** The terminal shows output from nodemon:

```
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server is listening on Port 5000....
```

<http://localhost:5000/>



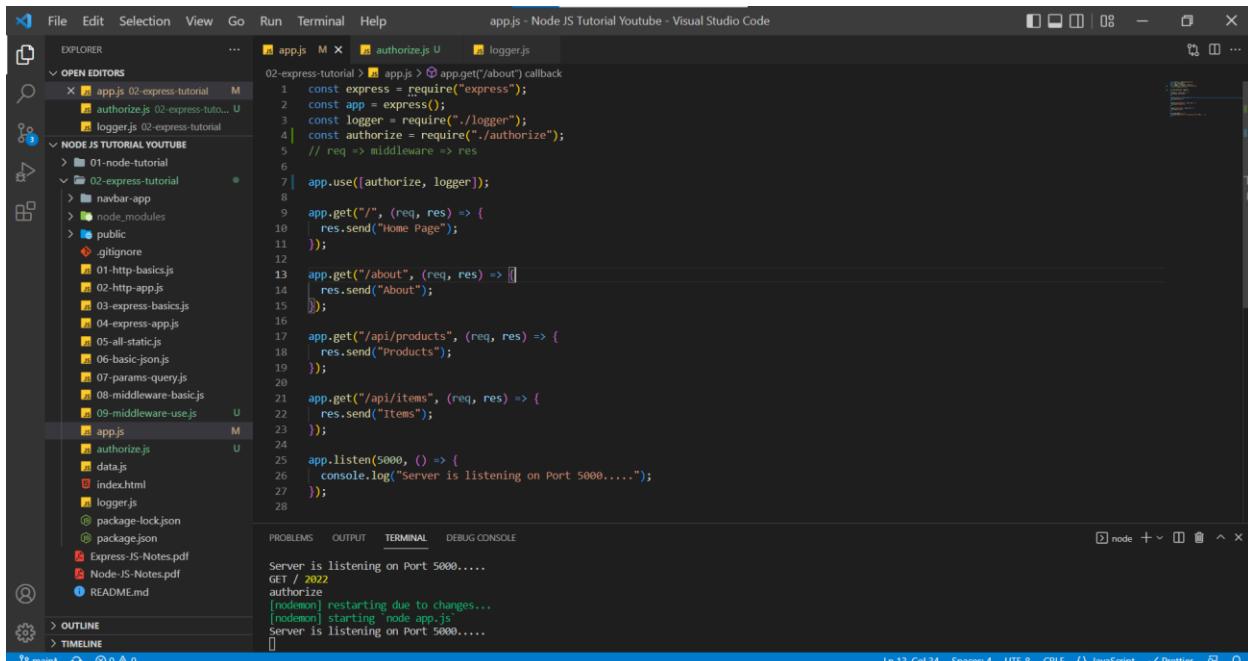
Terminal (Here we can see two console statements, one from logger.js and another from authorize.js)



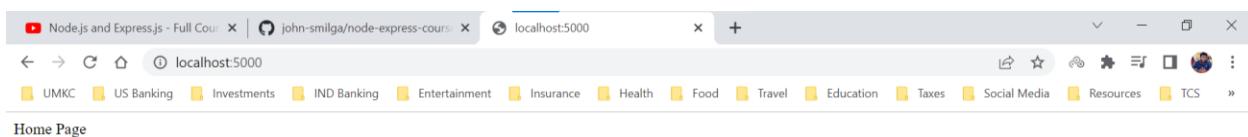
Order is important in middleware functions, the order we enter in the code is the order that Express JS follows. In the above terminal you can see statement of logger.js and then statement of authorize.js.

In the example, we can find the reserve order when we change the order in the app.use() method.

app.js



<http://localhost:5000/>



Terminal (after changing the order and restarting the server, we can see the console log statements in the order we mentioned in `app.use()` method).

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE node + ×

GET / 2022
authorize
[nodemon] restarting due to changes...
[nodemon] starting node app.js
Server is listening on Port 5000.....
authorize
GET / 2022
[]
```

Order of middleware functions is important in `app.use()` method.

In General, that is not how we authorize the webpages, we use the JWT token to authorize. In the below example we will try to authorize using the query string (this is also an example).

## authorize.js

## **Lessons Learned:**

**We cannot use next() method once we sent back the response**

**We must extract the query string from req.query property.**

```
const authorize = (req, res, next) => {
  const { user } = req.query;
  if (user === "john") {
    console.log("Authorized");
    // We can add an parameter to the request object and send it to the next middleware function
    // we can access that parameter in the next middleware function
    req.user = { name: "John", id: "34" };
    next();
  } else {
    console.log("Unauthorized");
    res.status(401).send("Unauthorized");
  }
};

module.exports = authorize;
```

## app.js

The screenshot shows the Visual Studio Code interface with the 'app.js' file open in the editor. The code implements an Express.js application with middleware functions for logging and authorization.

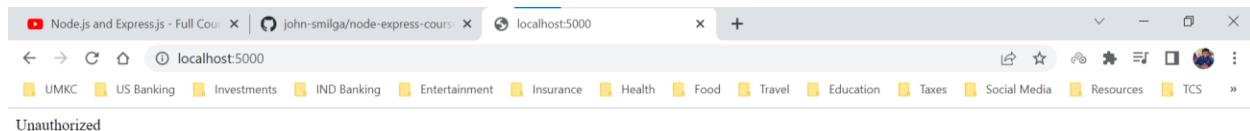
```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

OPEN EDITORS
02-express-tutorial > app.js M x authorize.js U logger.js

2 const app = express();
3 const logger = require("./logger");
4 const authorize = require("./authorize");
5
6 // req => middleware => res
7 app.use(authorize, logger);
8
9 app.get("/", (req, res) => {
10   res.send("Home Page");
11 });
12
13 app.get("/about", (req, res) => {
14   res.send("About");
15 });
16
17 app.get("/api/products", (req, res) => {
18   res.send("Products");
19 });
20
21 app.get("/api/items", (req, res) => {
22   res.send("Items");
23 });
24
25 app.listen(5000, () => {
26   console.log("Server is listening on Port 5000....");
27 });

Ln 19, Col 4 Spaces: 4 UTF-8 CRLF ( JavaScript Prettier 
```

<http://localhost:5000/>

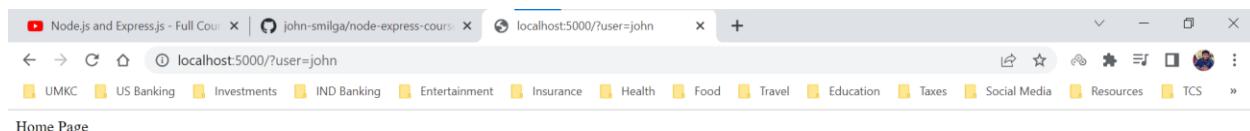


Terminal ( We can see the console log statement from authorize middleware function)

The terminal shows the node.js process starting and listening on port 5000. It also logs the 'Unauthorized' response sent to the client.

```
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node app.js'
Server is listening on Port 5000.....
Unauthorized
```

<http://localhost:5000/?user=john>



Terminal (We can see authorize middleware function and then the logger middleware function)

The terminal shows the node.js process starting and listening on port 5000. It logs the 'Authorize' and 'GET /?user=john 2022' requests, indicating the execution of both middleware functions.

```
Unauthorized
Unauthorized
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on Port 5000.....
Authorize
GET /?user=john 2022
```

We can now access the user parameter that we added to the request object in authorize middleware function.

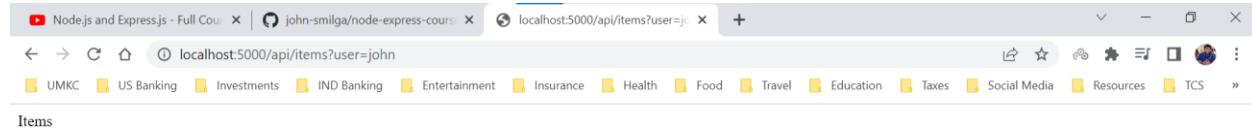
app.js

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

- File Explorer:** Shows the project structure under "OPEN EDITORS". The "02-express-tutorial" folder contains files like app.js, authorize.js, data.js, index.html, logger.js, package-lock.json, package.json, and README.md.
- Code Editor:** The main editor pane displays the content of `app.js`. The code sets up an Express application, defines routes for home, about, products, and API items, and includes middleware for logging and authorization.
- Terminal:** The bottom terminal window shows the output of running the application on port 5000, indicating successful execution and listening on the specified port.

We are retrieving the user information for the route /api/items hence we need to visit the route.

<http://localhost:5000/api/items?user=john>



## Terminal

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

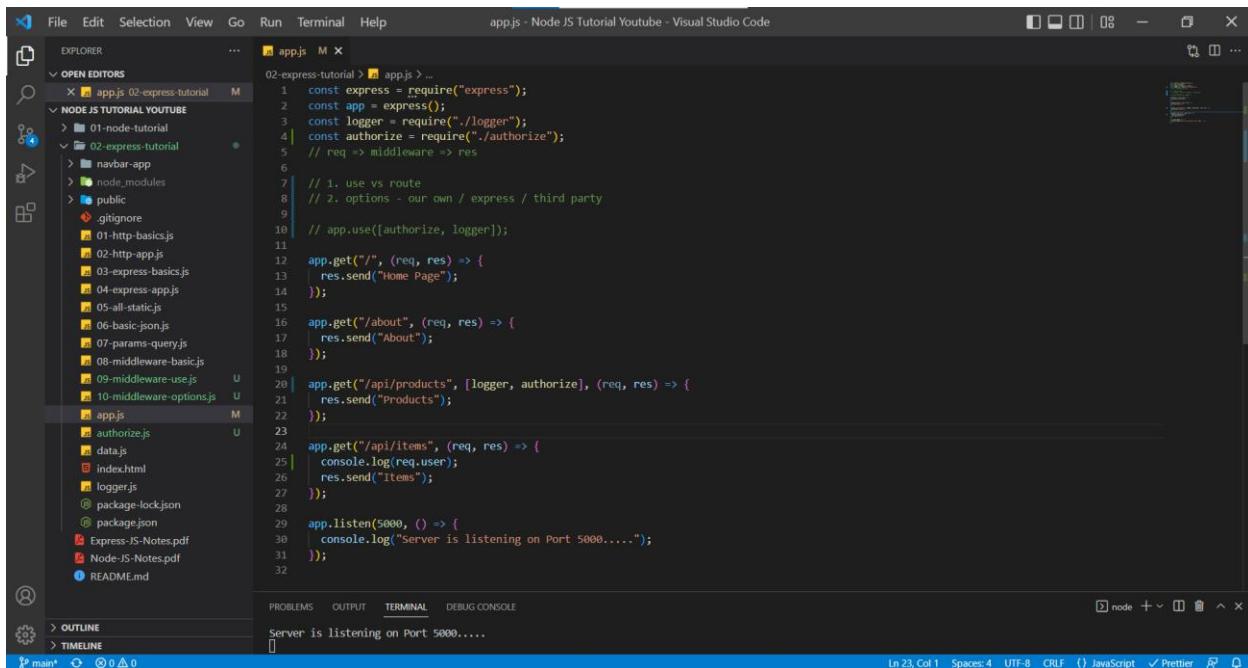
node + ×

In the terminal, we can see 3 console log statements. The first one being the authorize middleware function, second one being the logger middleware function and then third one being the console log statement from /api/items route.

## Additional Middleware Info

Assume a scenario, where we don't want to add `app.use()` method to all the routes. In the above scenario we applied `app.use()` method to all the routes in the `app.js` but we will apply only to `/api/products` route.

## app.js

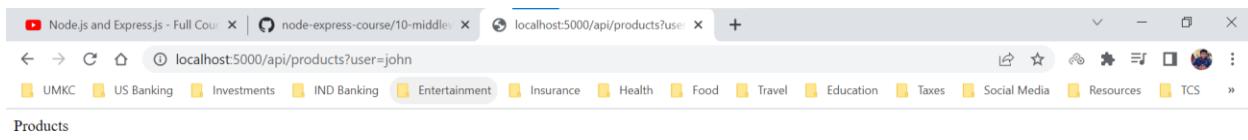


```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
navbar-app
node_modules
public
.gitignore
01-http-basics.js
02-http-app.js
03-express-basics.js
04-express-app.js
05-all-static.js
06-basic.json.js
07-params-query.js
08-middleware-basics.js
09-middleware-use.js
10-middleware-options.js
app.js
authorize.js
data.js
index.html
logger.js
package-lock.json
package.json
Express-JS-Notes.pdf
Node-JS-Notes.pdf
README.md

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ln 23, Col 1 Spaces: 4 UTF-8 CR/LF {} JavaScript ✓ Prettier
Server is listening on Port 5000....
```

<http://localhost:5000/api/products?user=john>

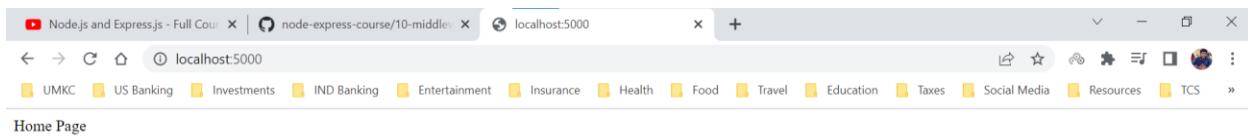


## Terminal

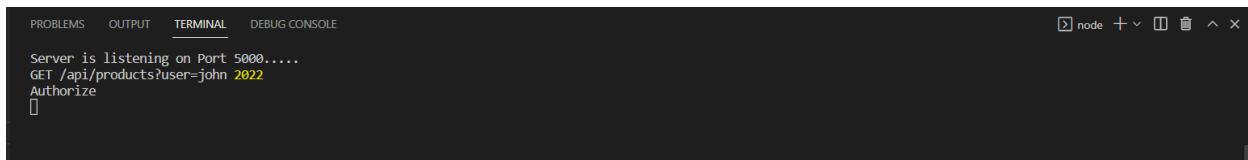


```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ln 23, Col 1 Spaces: 4 UTF-8 CR/LF {} JavaScript ✓ Prettier
Server is listening on Port 5000.....
GET /api/products?user=john 2022
Authorize
```

<http://localhost:5000/>



Terminal (we won't see any console log statements because middleware functions are not applied to home route )



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ln 23, Col 1 Spaces: 4 UTF-8 CR/LF {} JavaScript ✓ Prettier
Server is listening on Port 5000.....
GET /api/products?user=john 2022
Authorize
```

The options when we come to middleware functions

1. Own Middleware Functions (Ex: logger.js, authorize.js)
2. Express Middleware Functions (Ex: static)
3. Third Party Middleware Functions

We have created our own middleware functions like logger.js and authorize.js

There are many complicated Express middleware functions which we can handle but as of now we have an example of static middleware function which requires all the static files of the project that needs to send to the browser.

To use the third-party middleware functions we need to install the package.

In this example we are going to install the package **morgan** and we need to run command **npm i morgan**.

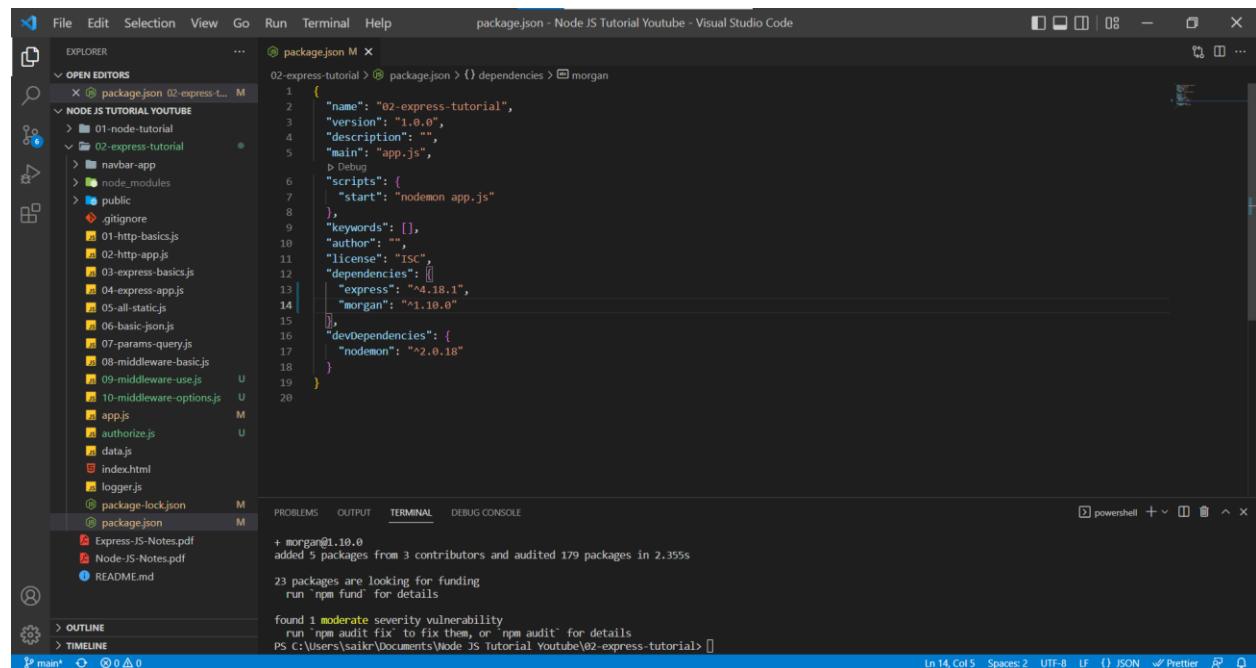
**morgan is a HTTP request logger middleware for node.js**

In a default way, they provide information of request came to the same server with a format, we can edit the format, but the default format is

**'*:method :url :status :res[content-length] - :response-time ms*'**

We installed morgan package.

package.json



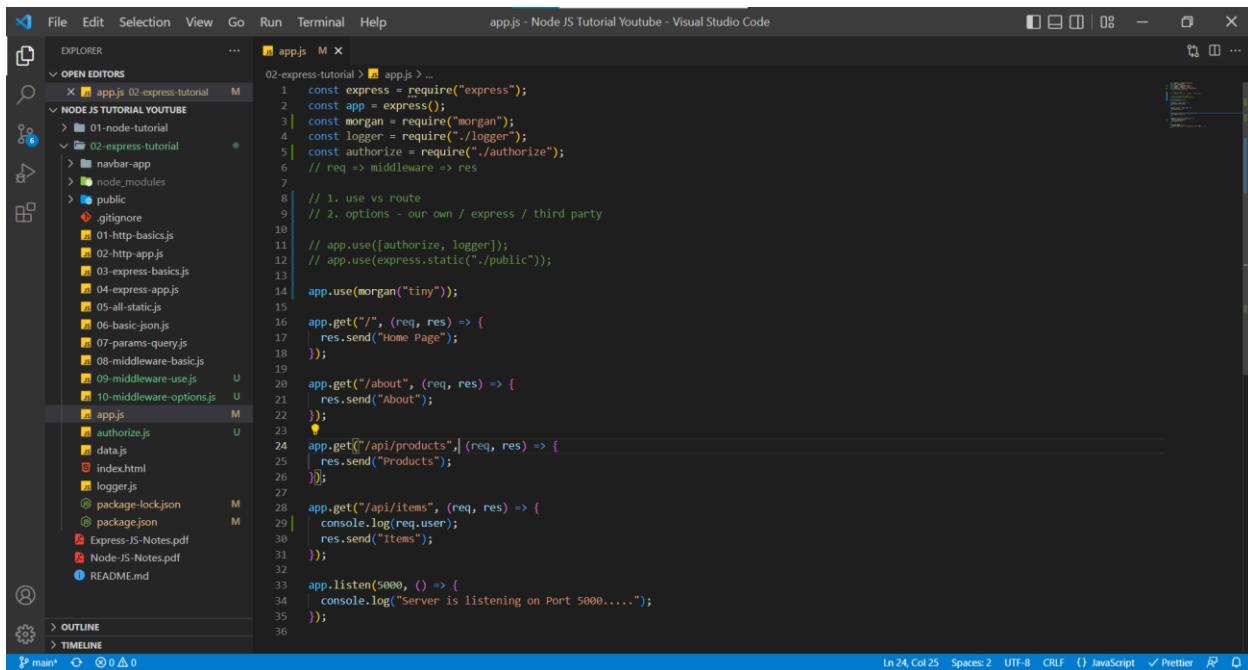
The screenshot shows the Visual Studio Code interface with the package.json file open in the center editor. The file contains the following JSON code:

```
1 {  
2   "name": "02-express-tutorial",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "app.js",  
6   "scripts": {  
7     "start": "nodemon app.js"  
8   },  
9   "keywords": [],  
10  "author": "",  
11  "license": "ISC",  
12  "dependencies": {  
13    "express": "^4.18.1",  
14    "morgan": "1.10.0"  
15  },  
16  "devDependencies": {  
17    "nodemon": "^2.0.18"  
18  }  
19}  
20
```

Below the editor, the terminal tab is active, showing the output of an npm command:

```
+ morgan@1.10.0  
added 5 packages from 3 contributors and audited 179 packages in 2.355s  
23 packages are looking for funding  
  run 'npm fund' for details  
found 1 moderate severity vulnerability  
  run 'npm audit fix' to fix them, or 'npm audit' for details  
PS C:\Users\saker\Documents\Node JS Tutorial Youtube\02-express-tutorial>
```

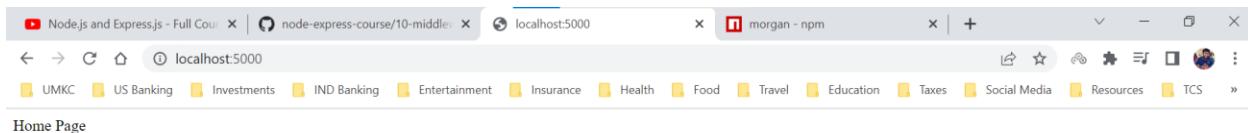
## app.js



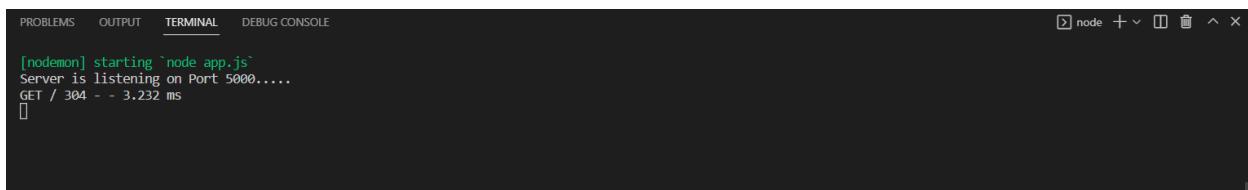
The screenshot shows the Visual Studio Code interface with the file `app.js` open in the editor. The code implements an Express.js application with various routes and middleware configurations.

```
02-express-tutorial > app.js > ...
1  const express = require("express");
2  const app = express();
3  const morgan = require("./morgan");
4  const logger = require("./logger");
5  const authorize = require("./authorize");
6  // req => middleware => res
7
8  // 1. use vs route
9  // 2. options - our own / express / third party
10
11 // app.use([authorize, logger]);
12 // app.use(express.static("./public"));
13
14 app.use(morgan("tiny"));
15
16 app.get("/", (req, res) => {
17   res.send("Home Page");
18 });
19
20 app.get("/about", (req, res) => {
21   res.send("About");
22 });
23
24 app.get("/api/products", (req, res) => {
25   res.send("Products");
26 });
27
28 app.get("/api/items", (req, res) => {
29   console.log(req.user);
30   res.send("Items");
31 });
32
33 app.listen(5000, () => {
34   console.log("Server is listening on Port 5000....");
35 });
36
```

<http://localhost:5000/>



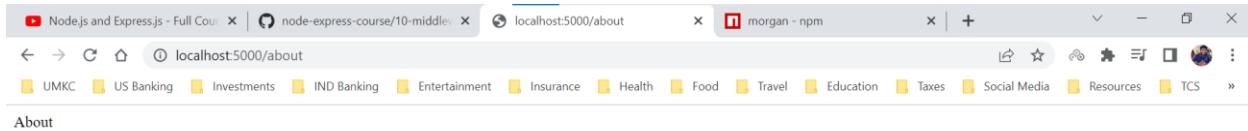
## Terminal



The terminal window shows the output of running the application with nodemon:

```
[nodemon] starting `node app.js`
Server is listening on Port 5000.....
GET / 304 - - 3.232 ms
[]
```

<http://localhost:5000/about>



## Terminal



```
[nodemon] starting `node app.js`
Server is listening on Port 5000.....
GET / 304 - - 3.232 ms
GET /about 304 - - 0.501 ms
GET /about 304 - - 0.548 ms
[]
```

## Methods – GET

**GET** is the default method for a request body to the server.

Below are the HTTP methods.

HTTP METHODS		
<b>GET</b>	Read Data	
<b>POST</b>	Insert Data	
<b>PUT</b>	Update Data	
<b>DELETE</b>	Delete Data	
<b>GET</b>	<a href="http://www.store.com/api/orders">www.store.com/api/orders</a>	get all orders
<b>POST</b>	<a href="http://www.store.com/api/orders">www.store.com/api/orders</a>	place an order (send data)
<b>GET</b>	<a href="http://www.store.com/api/orders/:id">www.store.com/api/orders/:id</a>	get single order (path params)
<b>PUT</b>	<a href="http://www.store.com/api/orders/:id">www.store.com/api/orders/:id</a>	update specific order (params + send data)
<b>DELETE</b>	<a href="http://www.store.com/api/orders/:id">www.store.com/api/orders/:id</a>	delete order (path params)

Generally, we would contact the database to get data and provide the data back to the user but in this case, we are getting the data from a file named data.js and sending the JSON data back to the user.

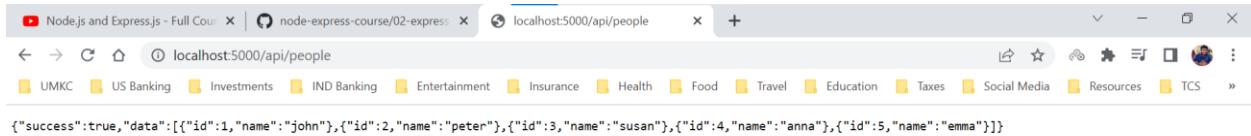
## app.js

The screenshot shows the Visual Studio Code interface. The left sidebar displays a tree view of files and folders, including 'OPEN EDITORS' (app.js, data.js), 'NODE JS TUTORIAL YOUTUBE' (01-node-tutorial, 02-express-tutorial), and various sample files like 01-http-basics.js, 02-http-app.js, etc. The main editor area contains the following code:

```
02-express-tutorial > app.js > ...
1 const express = require("express");
2 const app = express();
3 const { people } = require("./data");
4
5 app.get("/api/people", (req, res) => {
6   res.status(200).json({ success: true, data: people });
7 })
8
9 app.listen(5000, () => {
10   console.log("Server is listening on Port 5000....");
11 });
12
```

The terminal below shows the Node.js process starting and listening on port 5000. The status bar at the bottom indicates the file is a JavaScript file.

<http://localhost:5000/api/people>



We are not logging anything in the terminal in this example.

## Methods – POST

POST method is used to add/insert data on to the server.

We can setup the POST requests using two methods

1. Traditional Form Example
2. JavaScript option

In this case, we will a little bit of static data. In the GitHub (<https://github.com/john-smilga/node-express-course/tree/main/02-express-tutorial>) we have a folder named methods-public.

We are this folder because we cannot simply perform a POST request from the browser.

We cannot simply configure our browser to perform a post request. Either we need to use a tool named **Postman** or another tool named **Insomnia**, or we need simply set up a working application.

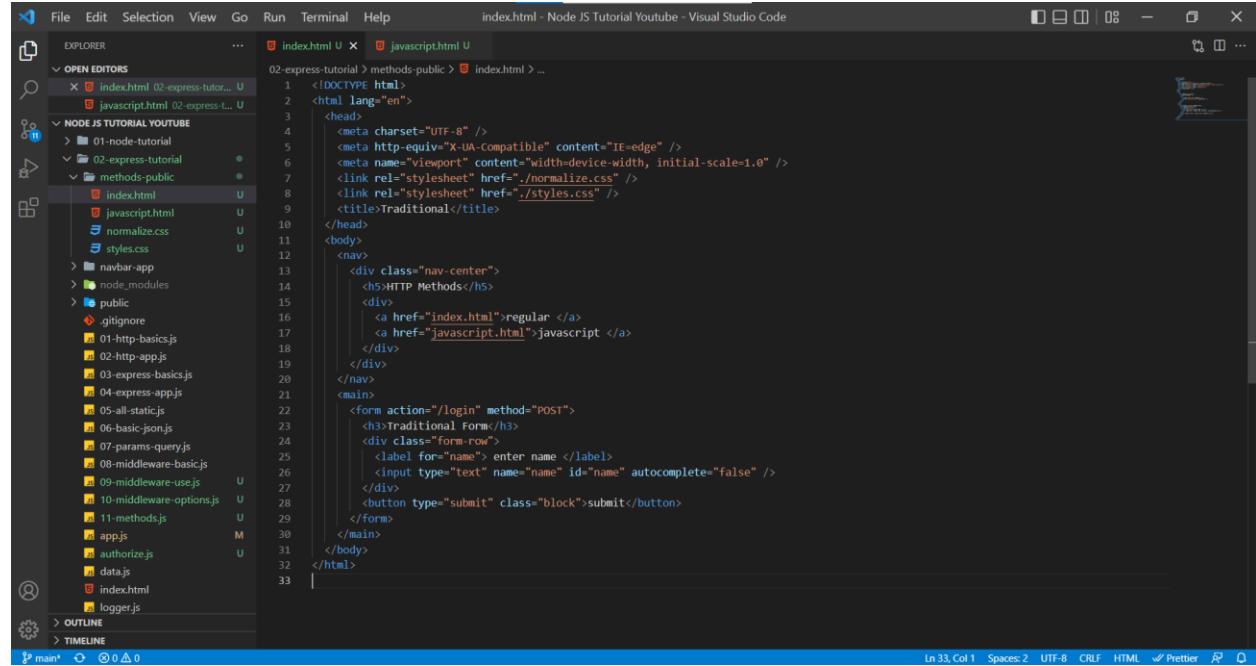
In this case we are going with the Traditional Form Application.

## Methods – POST (Form Example)

In this example we are using the traditional form application. We have route name /login and method is POST.

First, we have imported methods-public folder code from GitHub.

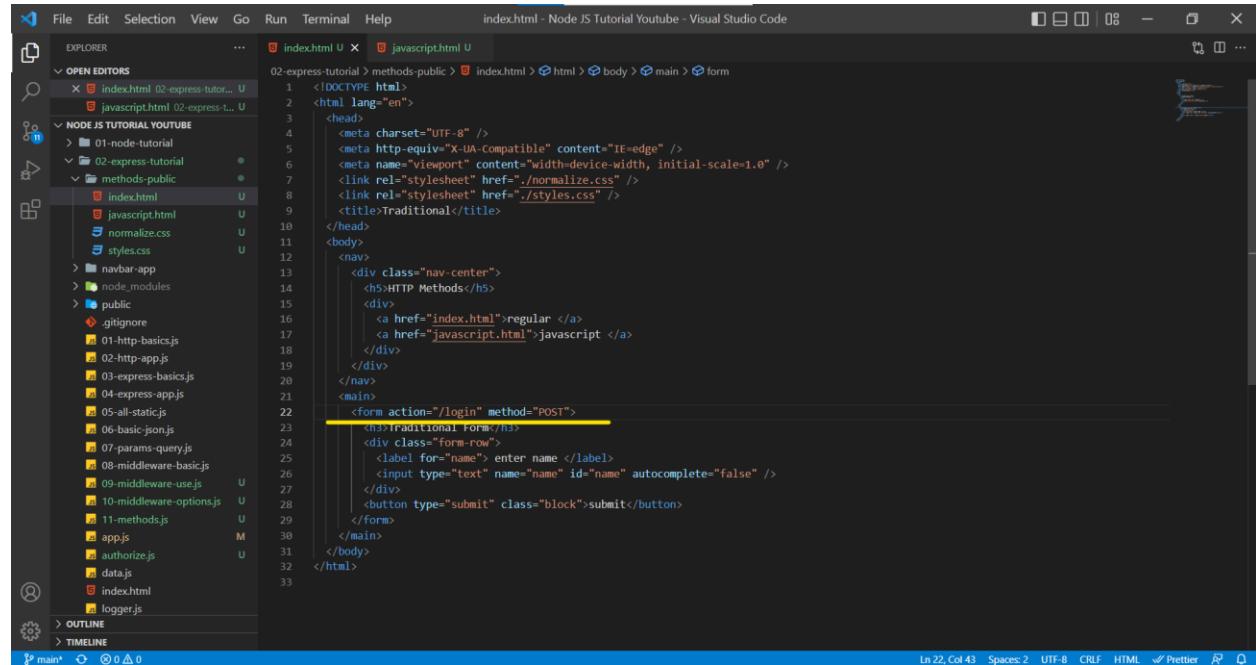
### methods-public folder (it contains two html and two css files)



The screenshot shows the Visual Studio Code interface with the 'index.html' file open in the editor. The file content is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="./normalize.css" />
    <link rel="stylesheet" href="./styles.css" />
    <title>Traditional</title>
  </head>
  <body>
    <nav>
      <div class="nav-center">
        <h5>HTTP Methods</h5>
        <div>
          <a href="index.html">regular </a>
          <a href="javascript.html">javascript </a>
        </div>
      </nav>
    <main>
      <form action="/login" method="POST">
        <h3>Traditional Form</h3>
        <div class="form-row">
          <label for="name"> enter name </label>
          <input type="text" name="name" id="name" autocomplete="false" />
        </div>
        <button type="submit" class="block">submit</button>
      </form>
    </main>
  </body>
</html>
```

### index.html



The screenshot shows the Visual Studio Code interface with the 'index.html' file open in the editor. A yellow selection bar highlights the 'method="POST"' attribute in the first form tag. The file content is identical to the one shown in the previous screenshot:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="./normalize.css" />
    <link rel="stylesheet" href="./styles.css" />
    <title>Traditional</title>
  </head>
  <body>
    <nav>
      <div class="nav-center">
        <h5>HTTP Methods</h5>
        <div>
          <a href="index.html">regular </a>
          <a href="javascript.html">javascript </a>
        </div>
      </nav>
    <main>
      <form action="/login" method="POST">
        <h3>Traditional Form</h3>
        <div class="form-row">
          <label for="name"> enter name </label>
          <input type="text" name="name" id="name" autocomplete="false" />
        </div>
        <button type="submit" class="block">submit</button>
      </form>
    </main>
  </body>
</html>
```

If we see at the line 22 action defines the route path whereas the method defines the type of HTTP method for that request.

As we know the body is optional in HTTP messages but it crucial when it comes to POST because we need to add data on the server hence, we require the body from request object.

## app.js

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists a project structure for 'NODE JS TUTORIAL YOUTUBE' with various files like 'index.html', 'methods-public.js', and 'normalize.css'. The 'app.js' file is open in the editor tab, showing the following code:

```
const express = require("express");
const app = express();
const { people } = require("./data");

// we will 304 status code when there is no change in our response
// it compares two etag values and if there is no change then it wont call the new response
app.disable("etag");

// static files
app.use(express.static("./methods-public"));

app.get("/api/people", (req, res) => {
  res.status(200).json({ success: true, data: people });
});

app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
});
```

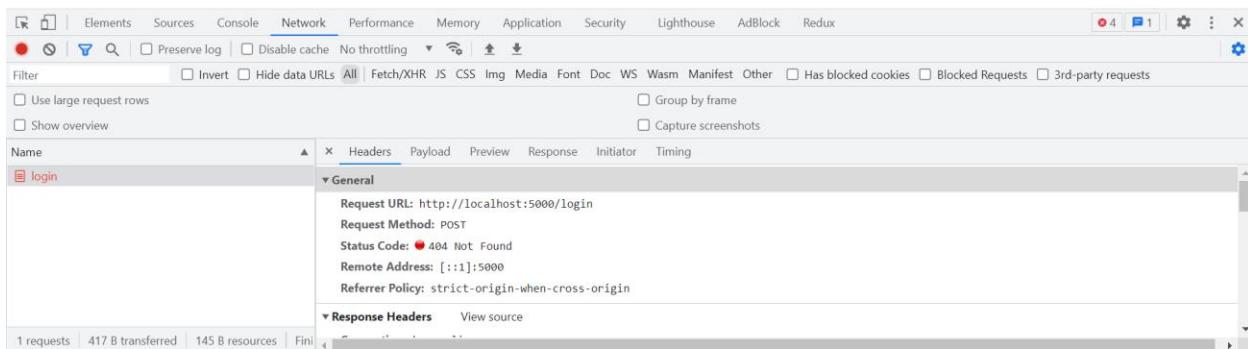
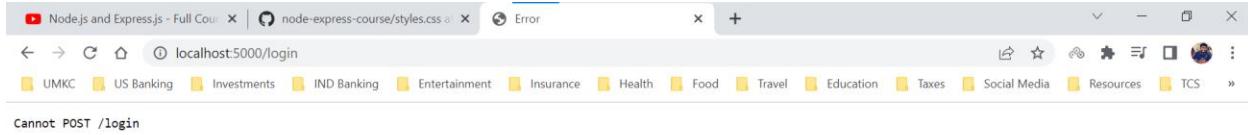
The terminal tab at the bottom shows the command line output of running the application with npm start, indicating it's listening on port 5000.

<http://localhost:5000>

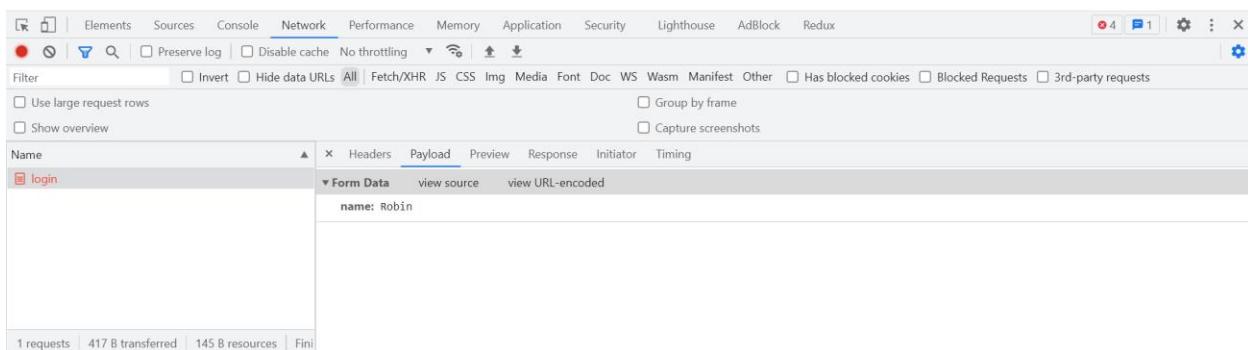
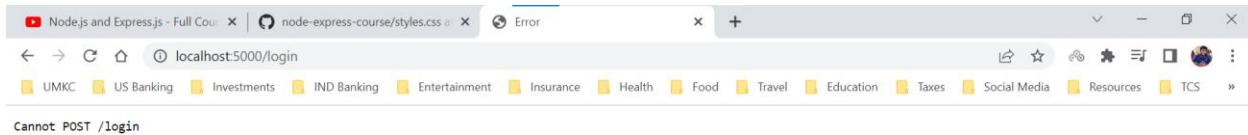
The screenshot shows a browser window with a blue header bar containing the text 'HTTP Methods' and 'Regular Javascript'. Below the header is a 'Traditional Form' section with a text input field labeled 'Enter Name' and a blue 'submit' button. At the bottom of the page is a navigation bar with links to various categories like UMKC, US Banking, Investments, etc.

At the bottom of the browser window, the developer tools Network tab is visible, showing a list of requests and their details. The top of the Network tab shows tabs for Elements, Sources, Console, Network, Performance, Memory, Application, Security, Lighthouse, AdBlock, and Redux. The Network tab itself has sections for Filter, Headers, and Body.

When we enter some value and click on submit, we are redirected to route `http://localhost:5000/login` and since we are not handling that route in app.js, we are getting 404 error. Also we can see the HTTP method being POST.

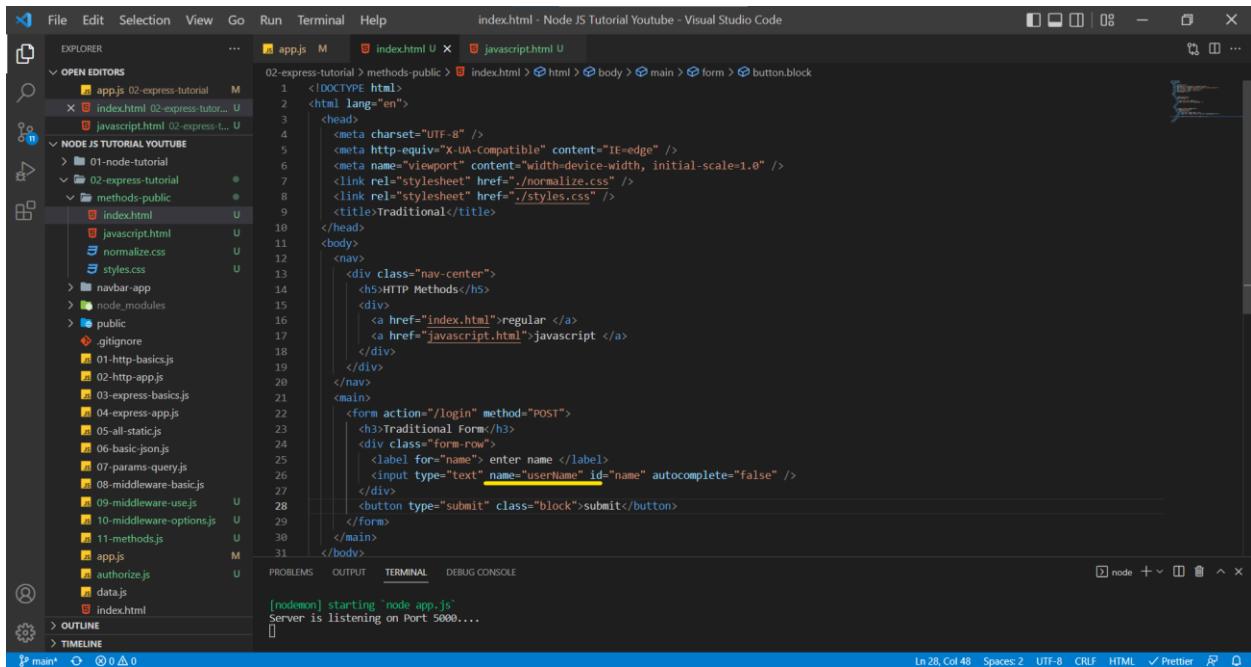


We entered names as Robin and clicked Submit button and hence we see the name Robin in Payload.



We have changed the name attribute in input type from name to userName.

## index.html

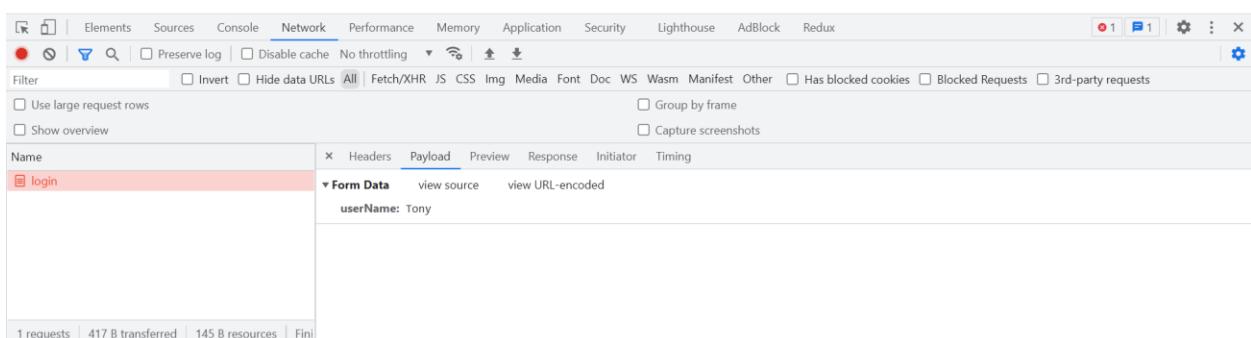
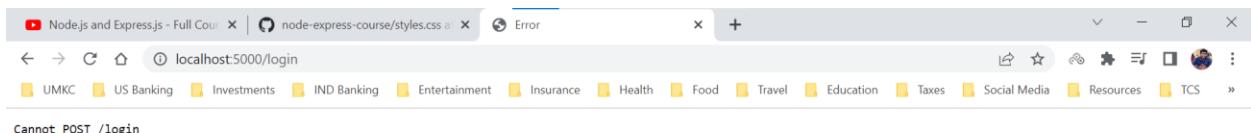


The screenshot shows the Visual Studio Code interface with the 'index.html' file open in the editor. The code is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href=".normalize.css" />
    <link rel="stylesheet" href=".styles.css" />
    <title>Traditional</title>
  </head>
  <body>
    <nav>
      <div class="nav-center">
        <h3>HTTP Methods</h3>
        <div>
          <a href="index.html">regular</a>
          <a href="javascript.html">javascript</a>
        </div>
      </div>
    </nav>
    <main>
      <form action="/login" method="POST">
        <h3>Traditional Form</h3>
        <div class="form-row">
          <label for="name"> enter name </label>
          <input type="text" name="userName" id="name" autocomplete="false" />
        </div>
        <button type="submit" class="block">submit</button>
      </form>
    </main>
  </body>
</html>
```

The bottom status bar shows: Ln 28, Col 48 Spaces: 2 UFT-8 CRLF HTML ✓ Prettier

<http://localhost:5000/login>

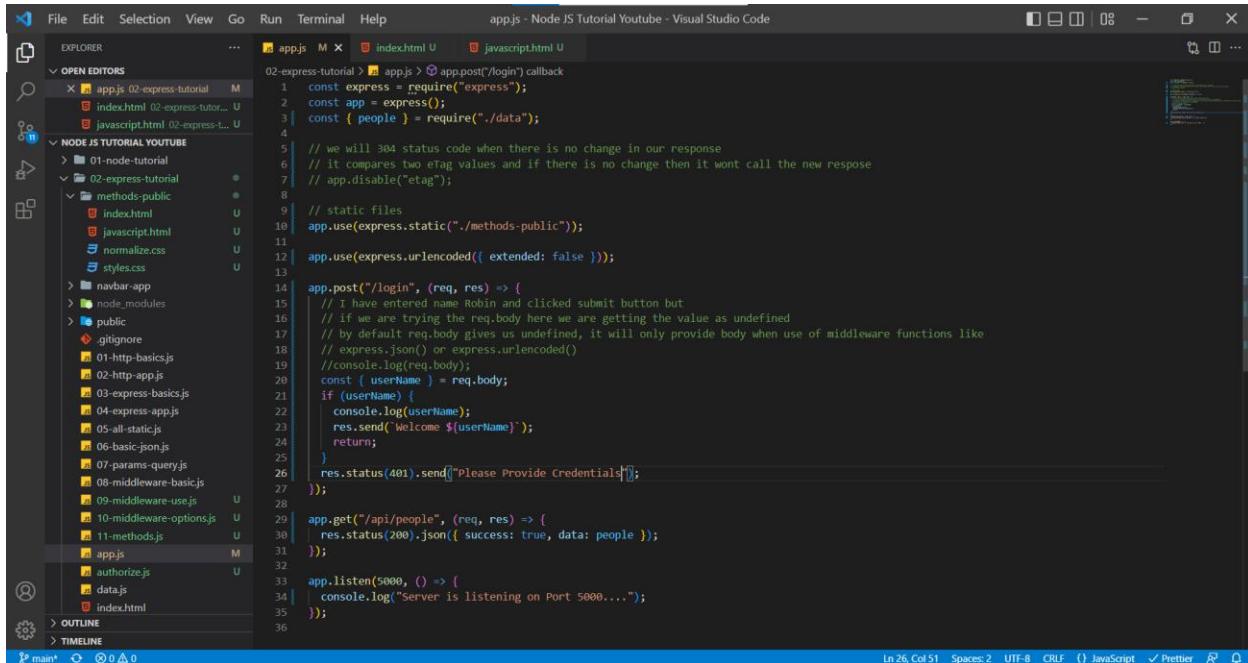


The attribute name in the input will be key and the whatever enter in the form would act like value and that is the reason we see them as a key value pair the request body of /login route.

We now must create a `app.post()` to handle route `/login` and to get access to request body we need to use inbuilt middleware function named `express.urlencoded`.

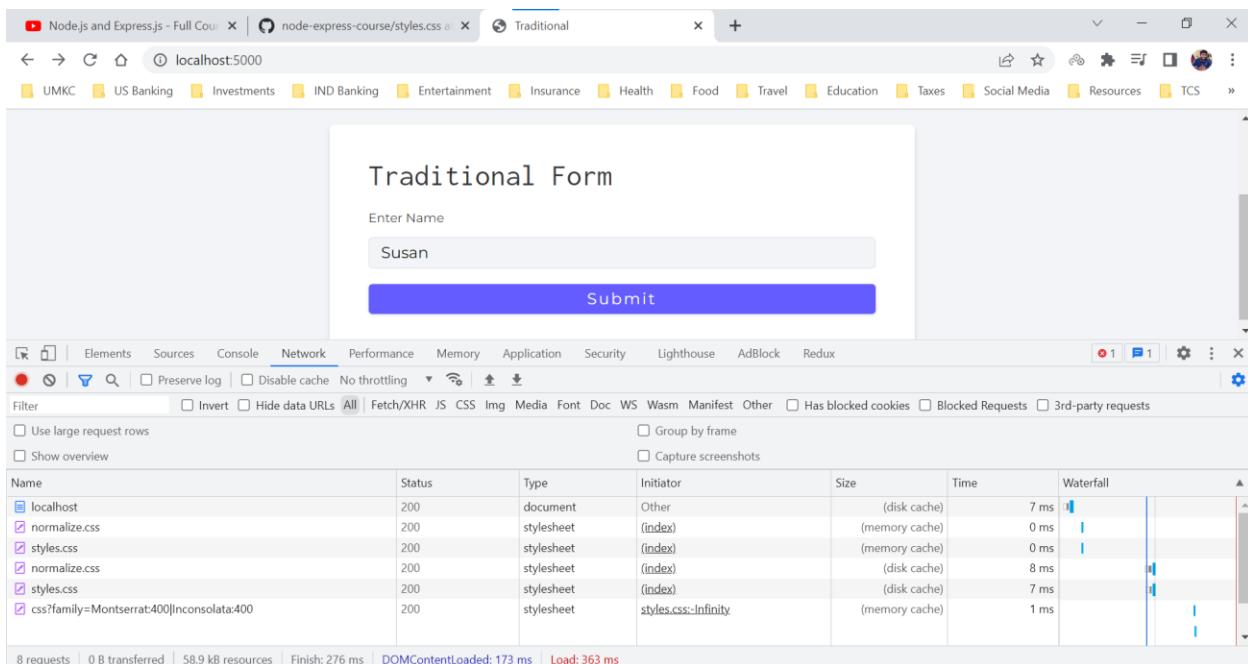
We can have a though using the `req.body` but by default it will provide us `undefined`, it will provide the request body only when we use middleware functions like `express.json()` or `express.urlencoded()`.

## app.js



```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS NODE JS TUTORIAL YOUTUBE 02-express-tutorial > app.js > app.post("/login") callback
1 const express = require("express");
2 const app = express();
3 const { people } = require("./data");
4
5 // we will 304 status code when there is no change in our response
6 // it compares two etag values and if there is no change then it wont call the new response
7 // app.disable("etag");
8
9 // static files
10 app.use(express.static("./methods-public"));
11
12 app.use(express.urlencoded({ extended: false }));
13
14 app.post("/login", (req, res) => {
15   // I have entered name Robin and clicked submit button but
16   // if we are trying the req.body here we are getting the value as undefined
17   // by default req.body gives us undefined, it will only provide body when use of middleware functions like
18   // express.json() or express.urlencoded()
19   // console.log(req.body);
20   const { userName } = req.body;
21   if (userName) {
22     console.log(userName);
23     res.send(`Welcome ${userName}`);
24     return;
25   }
26   res.status(401).send("Please Provide Credentials");
27 });
28
29 app.get("/api/people", (req, res) => {
30   res.status(200).json({ success: true, data: people });
31 });
32
33 app.listen(5000, () => {
34   console.log("Server is listening on Port 5000....");
35 });
36
```

<http://localhost:5000/>



The browser window shows a simple "Traditional Form" with an input field containing "Susan" and a "Submit" button. The Network tab of the developer tools shows the following request details:

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	document	Other	(disk cache)	7 ms	
normalize.css	200	stylesheet	(index)	(memory cache)	0 ms	
styles.css	200	stylesheet	(index)	(memory cache)	0 ms	
normalize.css	200	stylesheet	(index)	(disk cache)	8 ms	
styles.css	200	stylesheet	(index)	(disk cache)	7 ms	
css?family=Montserrat:400 Inconsolata:400	200	stylesheet	styles.css:-Infinity	(memory cache)	1 ms	

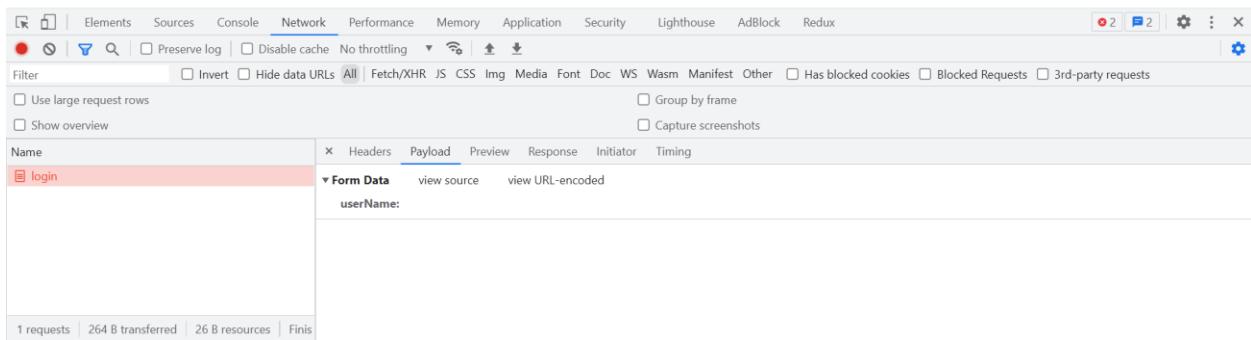
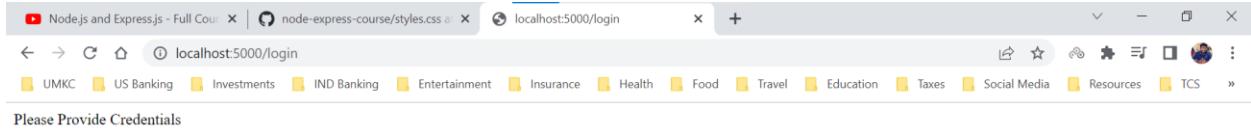
## After click on submit button

The screenshot shows a browser window with two tabs: "Node.js and Express.js - Full Course" and "localhost:5000/login". The main content area displays "Welcome Susan". Below the browser is the Chrome DevTools Network tab. It lists one request: "login" (POST). Under the "Payload" section, it shows the form data: "userName: Susan". At the bottom of the Network tab, it says "1 requests | 240 B transferred | 13 B resources | Fins".

## Terminal

The screenshot shows the VS Code terminal tab. It has tabs for "PROBLEMS", "OUTPUT", "TERMINAL" (which is active), and "DEBUG CONSOLE". The terminal output shows the message "Server is listening on Port 5000...." followed by "Susan".

If we don't enter name in the form, then we display the message "Please Provide Credentials"



extended property inside the express.urlencoded() middleware helps to choose the format of encoded data.

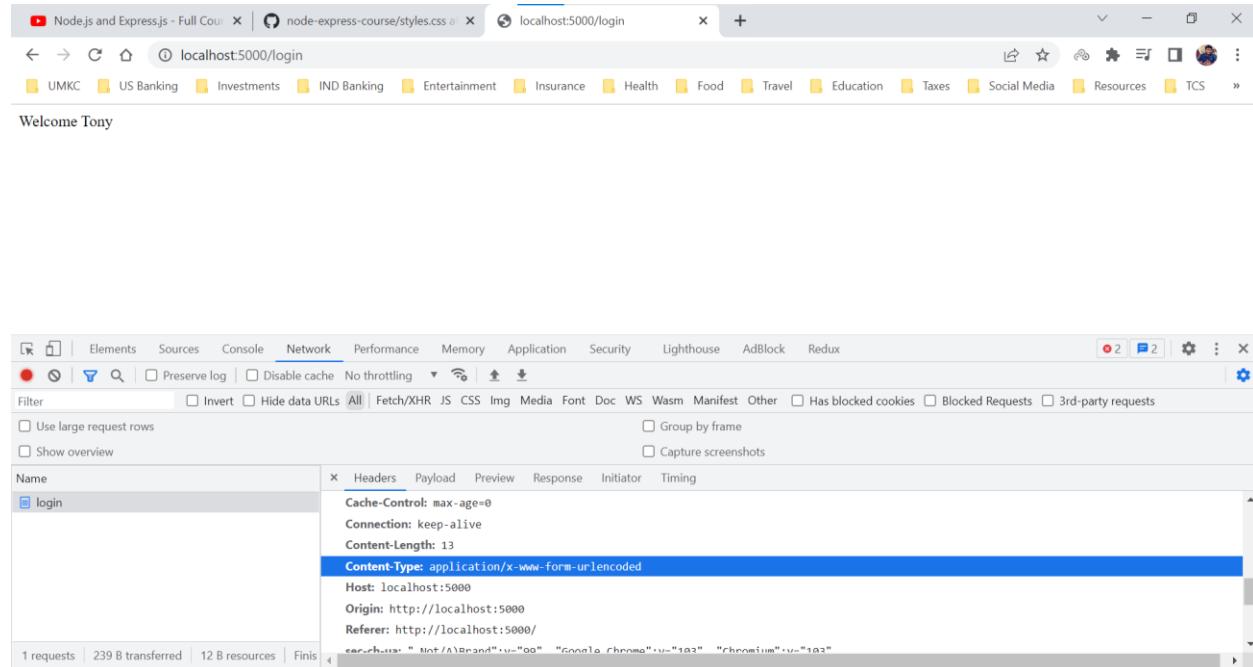
By default, it will be **true**,

**true** - parsing the URL-encoded data with qs library

**false** - parsing the URL-encoded data with querystring library

## Methods – POST (JavaScript Example)

If we look at the Content-Type in request headers, we find something interesting we have **application/x-www-form-urlencoded**.

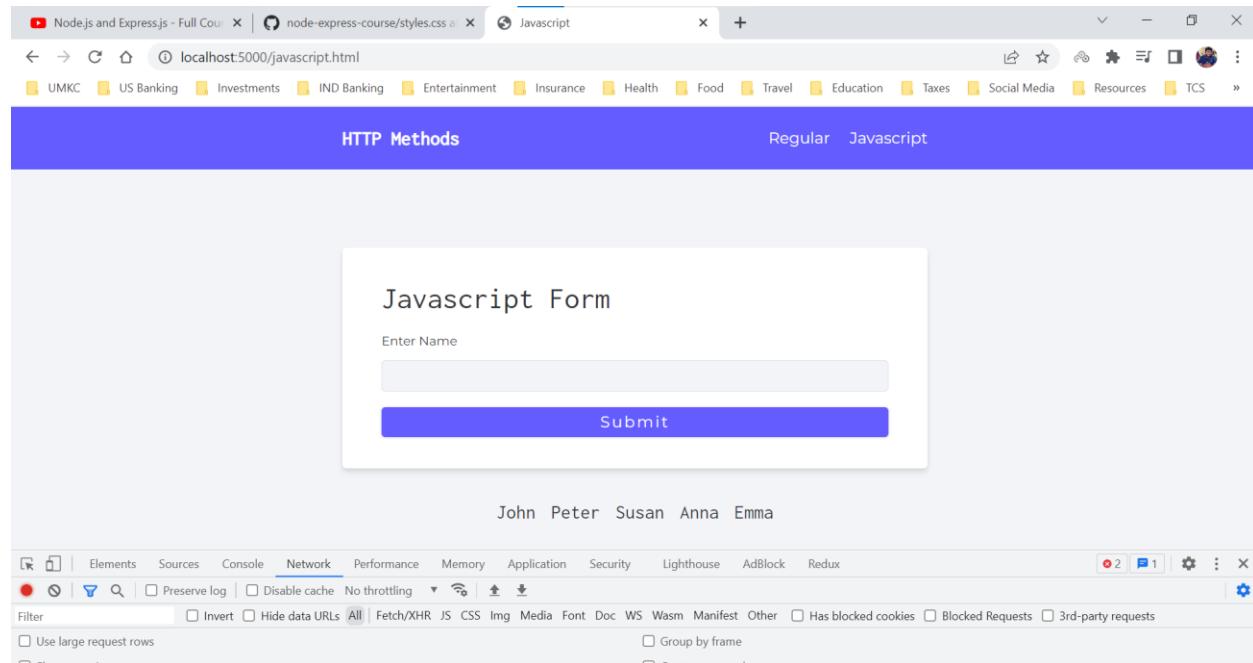


The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "node-express-course/styles.css", and "localhost:5000/login". The main content area displays the message "Welcome Tony". Below the browser is the Chrome DevTools Network tab. A request labeled "login" is selected. The Headers section shows the following:

- Cache-Control: max-age=0
- Connection: keep-alive
- Content-Length: 13
- Content-Type: application/x-www-form-urlencoded**
- Host: localhost:5000
- Origin: http://localhost:5000
- Referer: http://localhost:5000/

The Network tab also shows other details like "1 requests", "239 B transferred", and "12 B resources".

We are showing that now because for javascript that is going to be different.



The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "node-express-course/styles.css", and "localhost:5000/javascript.html". The main content area displays a "Javascript Form" with fields for "Enter Name" and a "submit" button. Below the form, the names "John", "Peter", "Susan", "Anna", and "Emma" are listed. Below the browser is the Chrome DevTools Network tab. A request is selected, and the Headers section shows:

- Content-Type: application/x-www-form-urlencoded

The Network tab also shows other details like "1 requests", "239 B transferred", and "12 B resources".

In this case we are handling the form strictly using the javascript. We strictly use javascript to send requests and of course the Content-Type is going to be different.

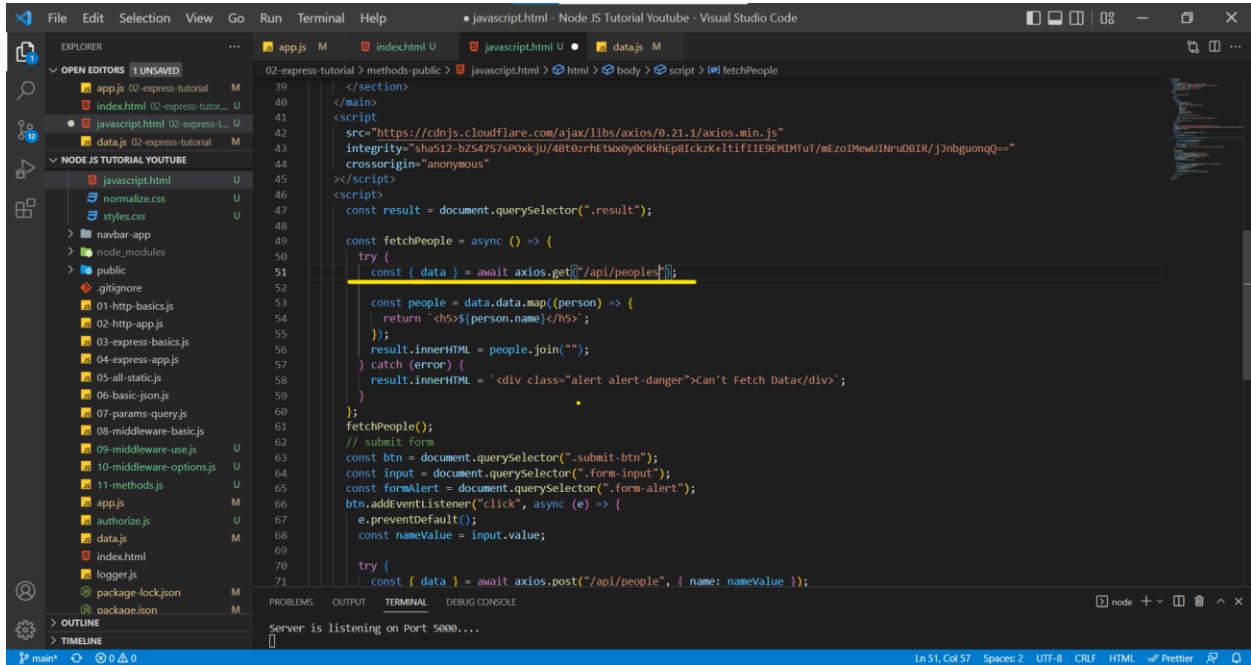
In javascript.html file, we don't have any onClick method for submit button.

Axios is one package which makes HTTP requests easier to setup. Instead of using in-built fetch we will be using axios because it provides clean API and better error messages.

We are using a function `fetchPeople` in frontend which is going to fetch people from the server and the path is `/api/people`.

On load itself we are trying to fetch data from people object in `data.js`

if we don't provide the proper route or api we will get an error.



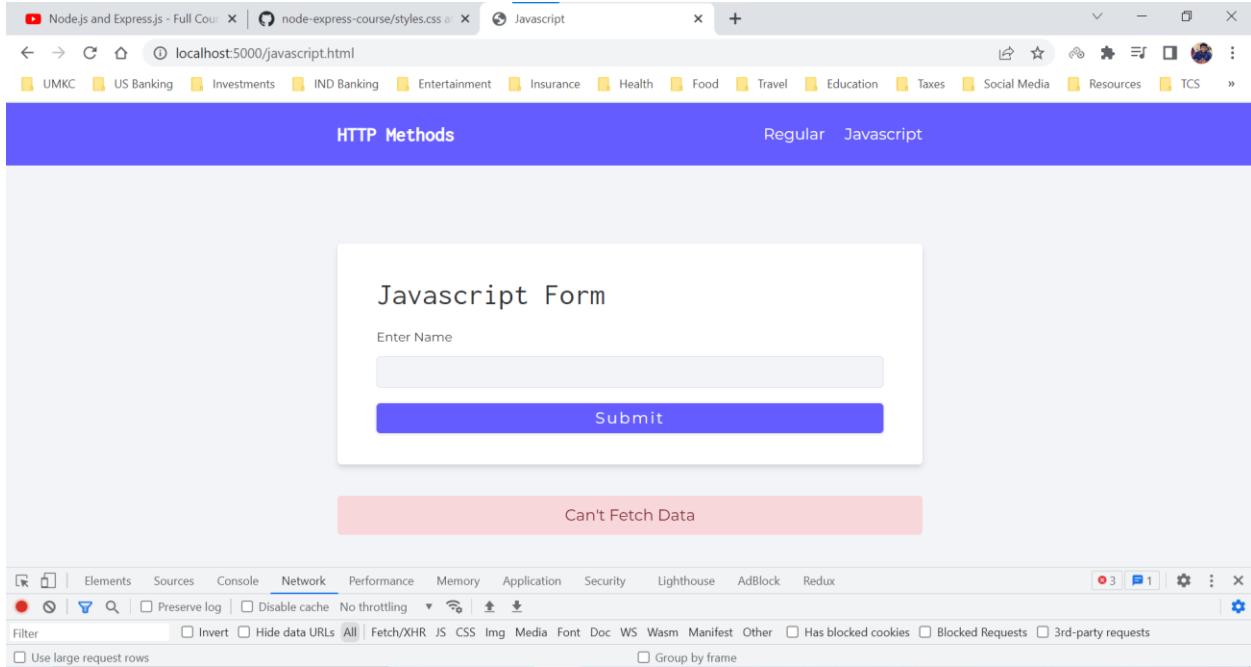
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "NODE JS TUTORIAL YOUTUBE".
- Code Editor:** The file "javascript.html" is open, showing the following code:

```
39     </section>
40   </main>
41   <script>
42     src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.21.1/axios.min.js"
43     integrity="sha512-hz547S7sPOxkjU4Bt0zrhEtWx0y0CRkhEp81ckzK+1tF1IE9EMIMTu/mEzoIMewUINruDBIR/j3nbguonq=="
44   </script>
45   <script>
46     const result = document.querySelector(".result");
47
48     const fetchPeople = async () => {
49       try {
50         const { data } = await axios.get("/api/people");
51
52         const people = data.data.map((person) => {
53           return `<h5>${person.name}</h5>`;
54         });
55         result.innerHTML = people.join("");
56       } catch (error) {
57         result.innerHTML = `<div class="alert alert-danger">Can't Fetch Data</div>`;
58       }
59     };
60
61     fetchPeople();
62
63     // submit form
64     const btn = document.querySelector(".submit-btn");
65     const input = document.querySelector(".form-input");
66     const formAlert = document.querySelector(".form-alert");
67     btn.addEventListener("click", async (e) => {
68       e.preventDefault();
69       const nameValue = input.value;
70
71       try {
72         const { data } = await axios.post("/api/people", { name: nameValue });
73       } catch (error) {
74         formAlert.innerHTML = `<div class="alert alert-danger">${error.message}</div>`;
75       }
76     });
77   
```

- Terminal:** Shows the message "Server is listening on Port 5000..."
- Status Bar:** Shows "Ln 51, Col 57" and other settings like "Spaces: 2", "UTF-8", "CRLF", "HTML", "Prettier".

We aren't handling the route /api/peoples and hence the output can be



**201 is the status code that we send back the user when we successfully executed a POST HTTP method request.**

**400 is the status code that we send we want to send an error to the user.**

Generally, we need to add the Content-Type in request header since we are using axios, it will take care of it.

<http://localhost:5000/javascript.html> (as mentioned earlier the Content-Type will be different, since we are using the javascript it is application/json)

The screenshot shows a browser window with a tab titled "Node.js and Express.js - Full Course". The main content area displays a "Javascript Form" with a single input field labeled "Enter Name" and a blue "submit" button. Below the form, there is a list of names: John, Peter, Susan, Anna, Watson. At the bottom of the browser window, the developer tools' Network tab is open, showing a request to "javascript.html". The Request Headers section is expanded, displaying the following headers:

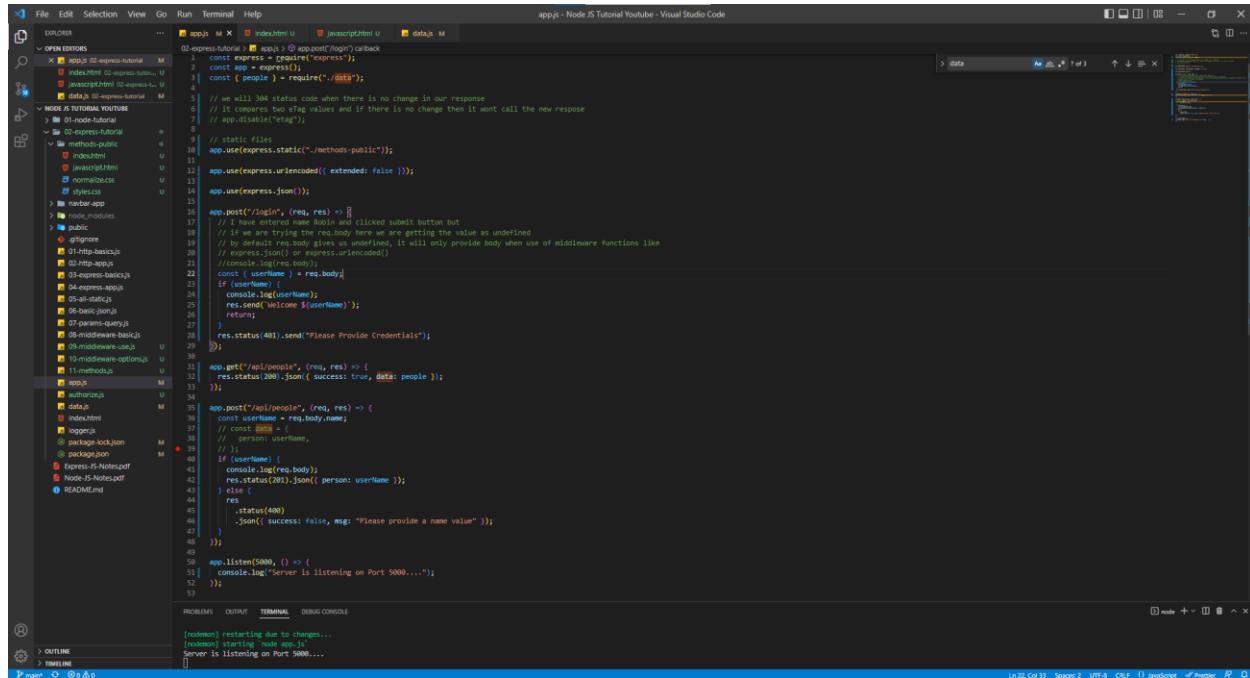
```
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
Host: localhost:5000
If-None-Match: "e99-aef74lpzofjUE69v1200y17y9s"
Referer: http://localhost:5000/javascript.html
sec-ch-ua: ".Not/A)Brand";v="99", "Google Chrome";v="103", "Chromium";v="103"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
```

Since the Content-Type is application/json, to access the request body in the server we need to use a express middleware function named express.json().

This is a built-in middleware function in Express. It parses incoming requests with JSON payloads and is based on body-parser.

Returns middleware that only parses JSON and only looks at requests where the Content-Type header matches the **type** option. This parser accepts any Unicode encoding of the body and supports automatic inflation of gzip and deflate encodings.

## app.js



```
const express = require('express');
const app = express();
const people = require('./data');

// we will 304 status code when there is no change in our response
// it compares two etag values and if there is no change then it wont call the new response
// app.disable('etag');

// static files
app.use(express.static("./methods-public"));

// app.use(express.urlencoded({ extended: false }));

app.use(express.json());

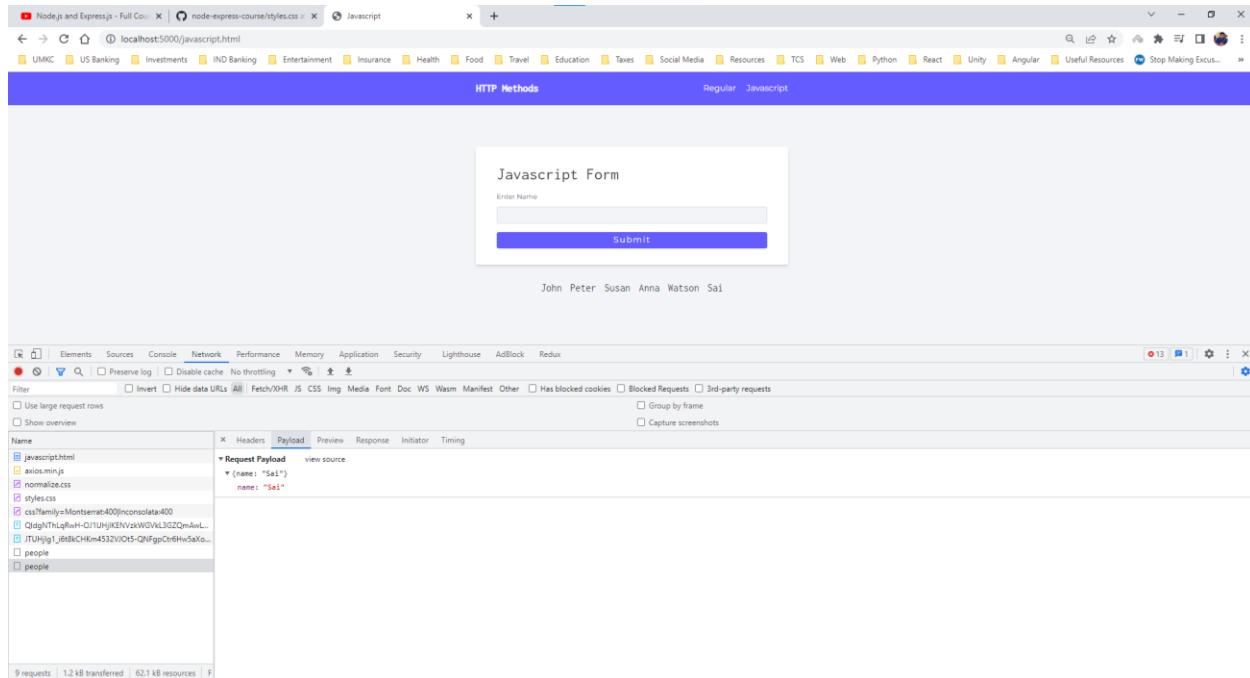
app.post("/login", (req, res) => {
  // I have entered name Robin and clicked submit button but
  // when I am trying the same again we are getting the value as undefined
  // by default req.body gives us undefined. It will only provide body when use of middleware functions like
  // express.json() or express.urlencoded()
  // console.log(req.body);
  const {username} = req.body;
  if (username) {
    console.log(username);
    res.send(`Welcome ${username}`);
  } else {
    res.status(401).send("Please Provide Credentials");
  }
});

app.get("/api/people", (req, res) => {
  res.status(200).json({ success: true, data: people });
});

app.post("/api/people", (req, res) => {
  const username = req.body.name;
  if (!username) {
    console.log(req.body);
    res.status(201).json({ person: userName });
  } else {
    res.status(400)
      .json({ success: false, msg: "Please provide a name value"});
  }
});

app.listen(5000, () => {
  console.log("Server is listening on Port 5000....");
});
```

If username is entered by user, we will add it to the bottom list



HTTP Methods Regular Javascript

Javascript Form

Enter Name

Submit

John Peter Susan Anna Watson Sai

Network

Request Payload

name: "Sai"

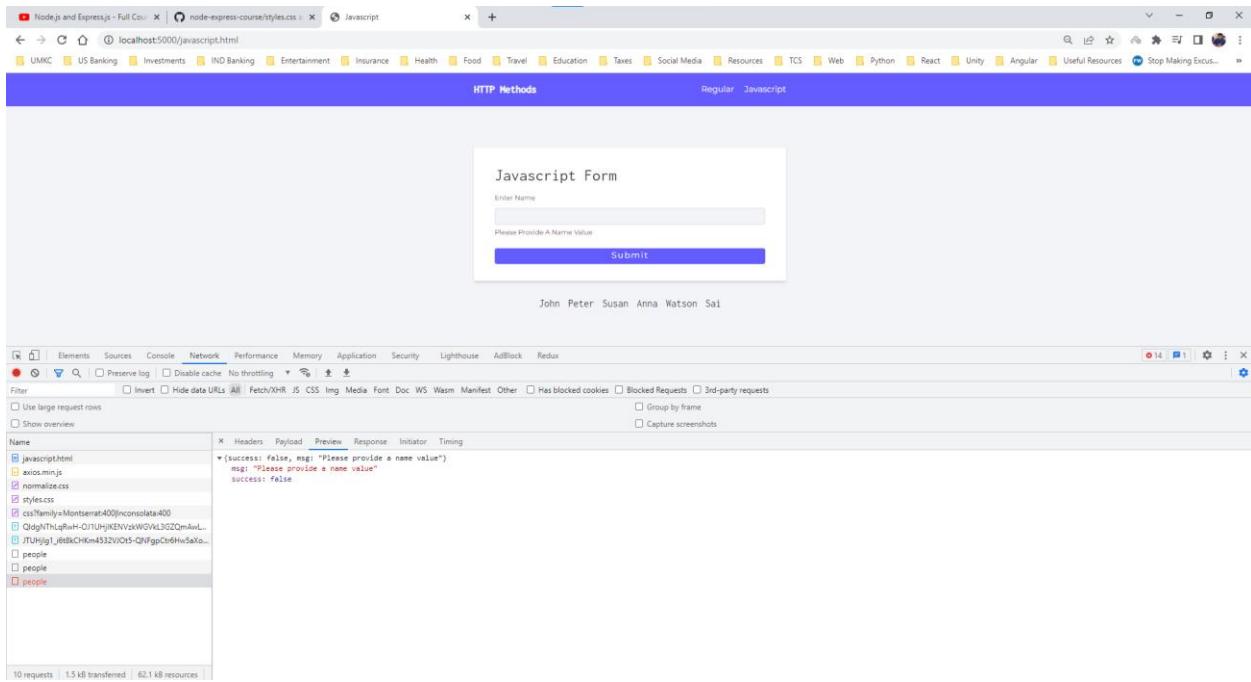
name: "Sai"

## Terminal



```
[nodemon] starting 'node app.js'
Server is listening on Port 5000...
{ name: 'Sai' }
```

If submit is clicked without any username



The screenshot shows a browser window with two tabs: "Node.js and Express.js - Full Course" and "node-express-course/styles.css". The main content area displays a "Javascript Form" with a single input field labeled "Enter Name" and a button labeled "Submit". Below the form, there is a list of names: John, Peter, Susan, Anna, Watson, and Sai. At the bottom of the page, there is a "Network" tab in the developer tools showing network requests. One request is highlighted, showing a response body with the message: "{'success': false, msg: 'Please provide a name value'}".

In axios, whatever we receive from the API's it is of data object and we can add properties to that and then we can retrieve data from that.

## Install Postman

However, there is a major problem in our current setup and the problem is,

If we need frontend to create testing for every route, then our development will be extremely slow because it will take way longer time for us to setup the frontend than just setting up one simple route.

In order reduce the effort on developing frontend we have a tool named Postman which is used to quickly test our API's.

In Postman we can group our requests into Collection. Postman helps us to enter an URL and the type of HTTP method (GET, POST, PUT, DELETE, PATCH) etc. for request body.

## app.js

```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
methods-public
index.html
normalize.css
styles.css
navbar-app
node_modules
public
.gitignore
01-http-basics.js
02-http-app.js
03-express-basics.js
04-express-app.js
05-all-static.js
06-basic.json.js
07-params-query.js
08-middleware-basic.js
09-middleware-use.js
10-middleware-options.js
11-methods.js
app.js
authorize.js
data.js
index.html
logger.js
package-lock.json

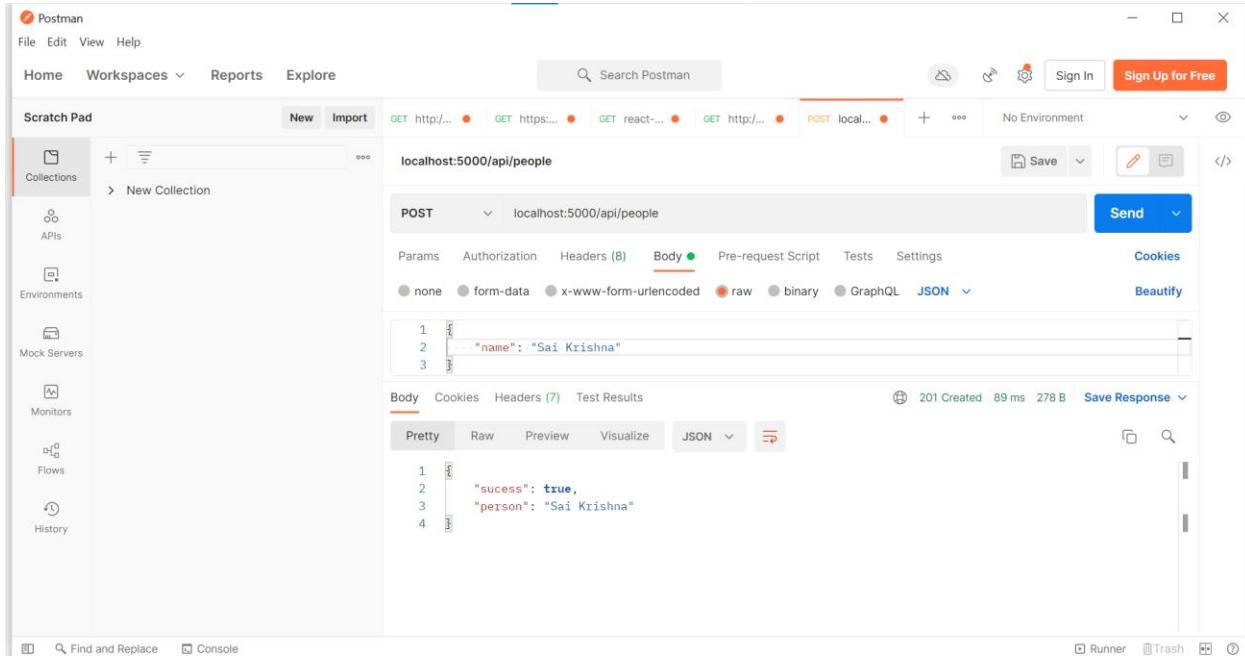
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Server is listening on Port 5000...
In 38, Col 25% Spaces: 2 UTF-8 CR LF {} JavaScript Preferences
```

```
02-express-tutorial > app.js > app.post("/api/people") callback
21 | //console.log(req.body);
22 | const { userName } = req.body;
23 |
24 | if (userName) {
25 |   console.log(userName);
26 |   res.send(`Welcome ${userName}`);
27 |   return;
28 | }
29 | res.status(401).send("Please Provide Credentials");
30 |
31 | app.get("/api/people", (req, res) => {
32 |   res.status(200).json({ success: true, data: people });
33 | });
34 |
35 | app.post("/api/people", (req, res) => [
36 |   const userName = req.body.name;
37 |   // const data = [
38 |   //   person: userName,
39 |   // ];
40 |   if (userName) {
41 |     console.log(req.body);
42 |     res.status(201).json({ sucess: true, person: userName });
43 |   } else {
44 |     res
45 |       .status(400)
46 |       .json({ success: false, msg: "Please provide a name value" });
47 |   }
48 | ]);
49 |
50 | app.listen(5000, () => {
51 |   console.log("Server is listening on Port 5000....");
52 | });
53 |
```

In Postman, localhost:5000/api/people with GET Method

```
Postman
File Edit View Help
Home Workspaces Reports Explore Search Postman Sign In Sign Up for Free
Scratch Pad New Import GET http://... ● GET https://... ● GET react-... ● GET http://... ● GET local... ● + ⚙ No Environment
localhost:5000/api/people
GET localhost:5000/api/people
Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies
Query Params
KEY VALUE DESCRIPTION Bulk Edit
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON
1
2
3
4
5
6
7
8
9
10
11
12
"success": true,
"data": [
{
  "id": 1,
  "name": "john"
},
{
  "id": 2,
  "name": "peter"
}
]
200 OK 42 ms 380 B Save Response
```

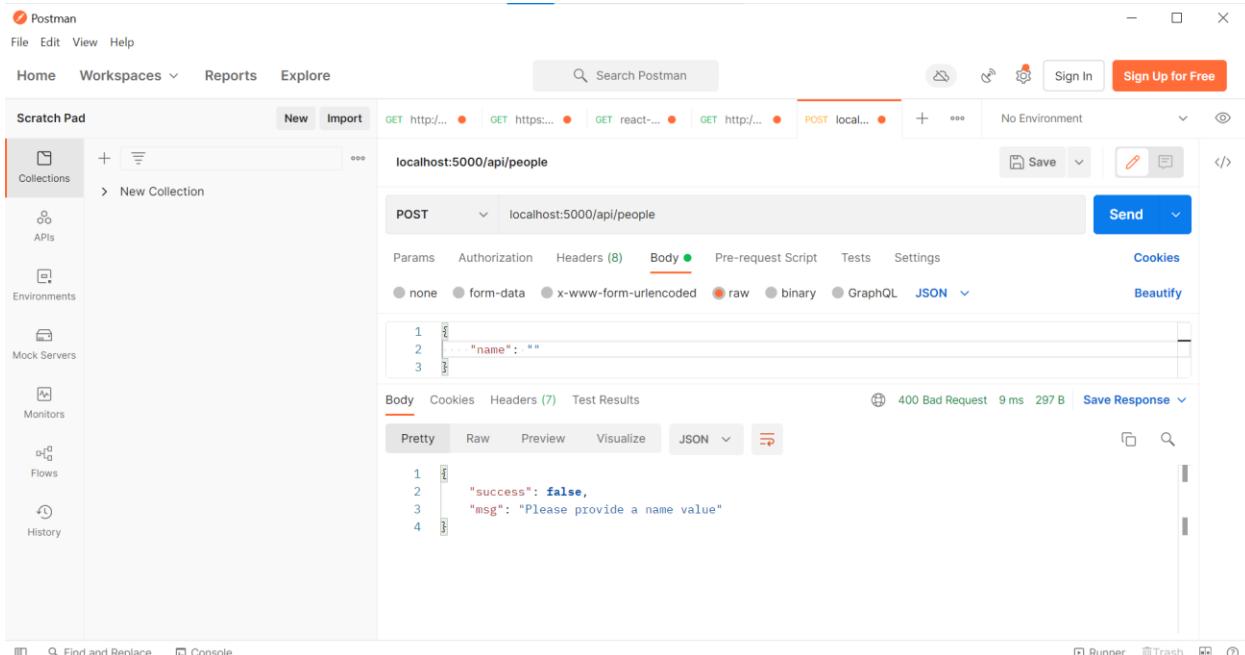
In Postman, localhost:5000/api/people with POST Method but for POST method, we need to send a request body with property named as **name**.



The screenshot shows the Postman application interface. On the left, the sidebar includes 'Scratch Pad', 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. The main workspace displays a POST request to 'localhost:5000/api/people'. The 'Body' tab is selected, showing a JSON payload with a single key 'name' set to 'Sai Krishna'. The response status is '201 Created' with a timestamp of '89 ms' and a size of '278 B'. The response body is shown in JSON format: "success": true, "person": "Sai Krishna".

When we don't send a value to server then we will receive a 400 error.

localhost:5000/api/people with POST Method.



This screenshot shows the same Postman setup as the previous one, but the request body is now empty, containing only the key 'name' without a value. The response status is '400 Bad Request' with a timestamp of '9 ms' and a size of '297 B'. The response body indicates failure: "success": false, "msg": "Please provide a name value".

Let us create an endpoint for postman and test it. ( localhost:5000/api/postman/people with POST method)

app.js

```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

EXPLORER
OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
METHODS-PUBLIC
NAVBAR-APP
PUBLIC
01-HTTP-BASICS.JS
02-HTTP-APP.JS
03-EXPRESS-BASICS.JS
04-EXPRESS-APP.JS
05-ALL-STATIC.JS
06-BASIC.JSON.JS
07-PARAMS.QUERY.JS
08-MIDDLEWARE-BASIC.JS
09-MIDDLEWARE-USE.JS
10-MIDDLEWARE-OPTIONS.JS
11-METHODS.JS
app.js
AUTHORIZE.JS
DATA.JS
INDEX.HTML
LOGGER.JS
PACKAGE-LOCK.JSON
OUTLINE
TIMELINE

app.js M x
02-express-tutorial > app.js > ...
32 |     res.status(200).json({ success: true, data: people });
33 | });
34 |
35 app.post("/api/people", (req, res) => {
36     const userName = req.body.name;
37     // const data = (
38     //     person: userName,
39     // );
40     if (userName) {
41         console.log(req.body);
42         res.status(201).json({ sucess: true, person: userName });
43     } else {
44         res
45             .status(400)
46             .json({ success: false, msg: "Please provide a name value" });
47     }
48 });
49 |
50 app.post("/api/postman/people", (req, res) => {
51     const userName = req.body.name;
52     if (!userName) {
53         res
54             .status(400)
55             .send({ success: false, msg: "please provide a name value" });
56         return;
57     }
58     res.status(201).send({ success: true, people: [...people], userName });
59 });
60 |
61 app.listen(5000, () => {
62     console.log("Server is listening on Port 5000....");
63 });
64 |

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Server is listening on Port 5000....
```

In Postman, localhost:5000/api/postman/people with PUT method with request body.

The screenshot shows the Postman application interface. The left sidebar contains navigation links: Home, Workspaces, Reports, Explore, Scratch Pad, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a collection named "New Collection". A search bar at the top right says "Search Postman". Below it, a row of icons includes "Sign In" and "Sign Up for Free". The main workspace shows a POST request to "localhost:5000/api/postman/people". The "Body" tab is selected, showing the JSON input: 

```
1
2   ...
3   "name": "Sai Krishna"
```

. The "Pretty" tab in the preview section shows the expanded JSON response:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

```
[{"id": 1, "name": "anna"}, {"id": 2, "name": "watson"}, {"id": 3, "name": "Sai Krishna"}]
```

In Postman, localhost:5000/api/postman/people with POST method with no request body.

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, Reports, Explore, and a search bar labeled "Search Postman". On the right side of the header, there are buttons for "Sign In" and "Sign Up for Free". The main workspace is titled "localhost:5000/api/postman/people". A "POST" method is selected. The "Body" tab is active, showing the following JSON payload:

```
1
2   "name": ""
3
```

Below the body, the response status is shown as "400 Bad Request" with a timestamp of "11 ms" and a size of "297 B". The response body is displayed as:

```
1
2   "success": false,
3   "msg": "please provide a name value"
4
```

## Methods – PUT

This method is for editing the data. So, PUT method is for update. If we edit or delete, we always go with the route parameter which gives us the ID to edit the exact specific item from the list.

When we send PUT or DELETE requests, we will send it to a specific path with item id, if item exist then there is body which is used to update.

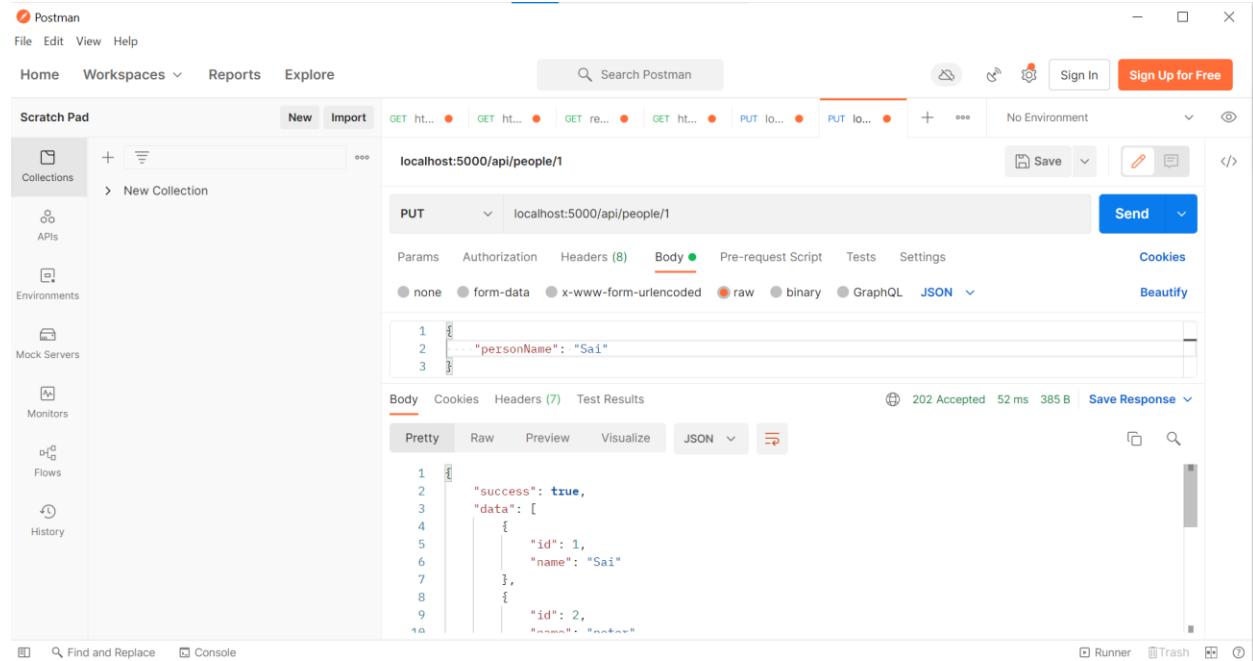
## app.js

The screenshot shows the Visual Studio Code interface with the file "app.js" open. The code handles a PUT request to "/api/people/:id". It checks if the "name" field is provided in the request body. If it is, it updates the person's name. If not, it returns an error message. Finally, it starts the server on port 5000.

```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code
OPEN EDITORS
02-express-tutorial > app.js M > data.js M
02-express-tutorial > app.js M > data.js M
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
  methods-public
    index.html U
    javascript.html U
    normalize.css U
    styles.css U
  navbar-app
  node_modules
  public
    .gitignore
    package.json
    package-lock.json
    public
      01-http-basics.js
      02-http.js
      03-express-basics.js
      04-express-app.js
      05-all-static.js
      06-basic-json.js
      07-params-query.js
      08-middleware-basic.js
      09-middleware-use.js U
      10-middleware-options.js U
      11-methods.js U
    app.js M
    authorize.js U
    data.js M
    index.html
    logger.js
  OUTLINE
  TIMELINE
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
]
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on Port 5000...
Ln 64, Col 55  Spaces: 2  UTF-8  CRLF  ✓ JavaScript  Prettier  R  D
```

In this case, we are working with route /api/people/:id which requires an id to edit/update a particular item and it requires body containing the new value of the item.

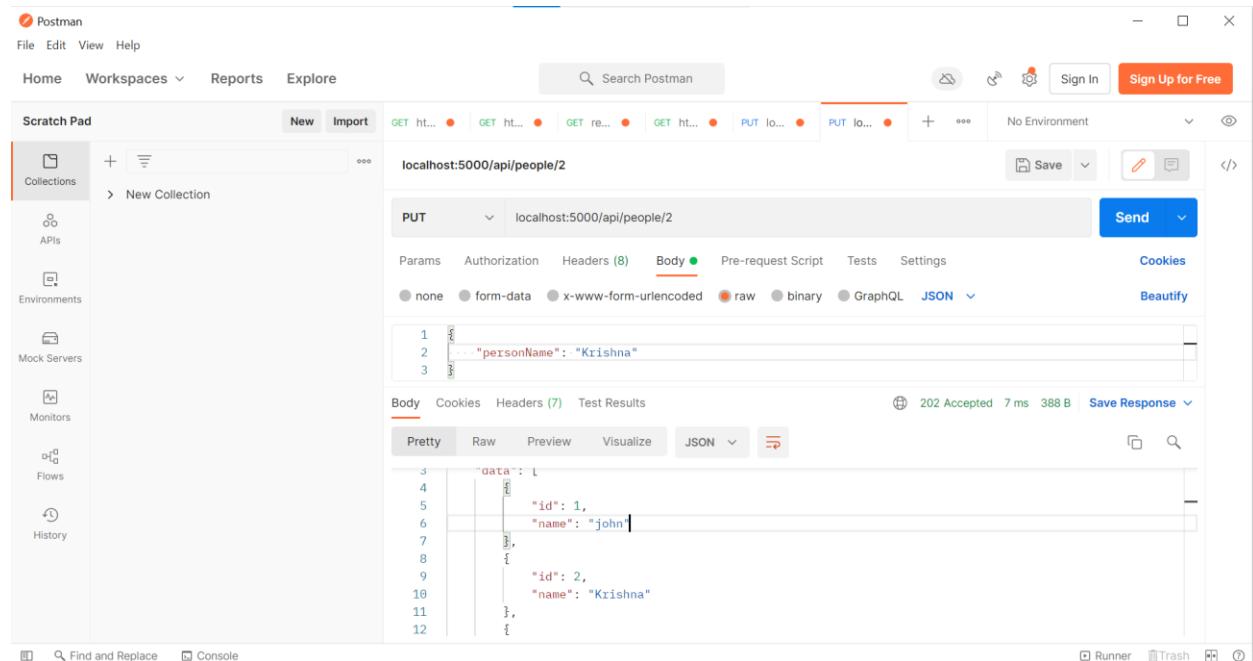
In Postman, localhost:5000/api/people/1. We are entering with Id and passing a request body



The screenshot shows the Postman interface with a PUT request to `localhost:5000/api/people/1`. The request body is a JSON object with `personName: "Sai"`. The response is a 202 Accepted status with a JSON body indicating success and updated data.

```
1 "success": true,
2 "data": [
3   {
4     "id": 1,
5     "name": "Sai"
6   },
7   {
8     "id": 2,
9     "name": "Sai"
10 }
```

In Postman, localhost:5000/api/people/2



The screenshot shows the Postman interface with a PUT request to `localhost:5000/api/people/2`. The request body is a JSON object with `personName: "Krishna"`. The response is a 202 Accepted status with a JSON body indicating success and updated data.

```
1 "success": true,
2 "data": [
3   {
4     "id": 1,
5     "name": "John"
6   },
7   {
8     "id": 2,
9     "name": "Krishna"
10 }
```

In Postman, localhost:5000/api/people/7 when there is no user with id 7

The screenshot shows the Postman interface. A PUT request is made to `localhost:5000/api/people/7`. The request body contains the JSON object `{"personName": "Krishna"}`. The response status is 404 Not Found, with the message "No Resources Found".

```
PUT localhost:5000/api/people/7
{
  "personName": "Krishna"
}
404 Not Found 15 ms 286 B
```

In Postman, localhost:5000/api/people/5 when there is no personName sent in request body

The screenshot shows the Postman interface. A PUT request is made to `localhost:5000/api/people/5`. The request body contains the JSON object `{"personName": ""}`. The response status is 400 Bad Request, with the message "please enter name value".

```
PUT localhost:5000/api/people/5
{
  "personName": ""
}
400 Bad Request 6 ms 303 B
```

## Methods – DELETE

In this method, the setup would be extremely like the PUT method. API route requires an id to delete a specific item.

The only difference from PUT method is that we don't need send a request body for the DELETE request.

app.js

```
File Edit Selection View Go Run Terminal Help app.js - Node JS Tutorial Youtube - Visual Studio Code

EXPLORER ... app.js M x data.js M
OPEN EDITORS app.js 02-express-tutorial M
  data.js 02-express-tutorial M
NODE JS TUTORIAL YOUTUBE
  01-node-tutorial
  02-express-tutorial
    methods-public
      index.html U
      javascript.html U
      normalize.css U
      styles.css U
  navbar-app
  node_modules
  public
    .gitignore
  01-http-basics.js
  02-http-app.js
  03-express-basics.js
  04-express-app.js
  05-all-static.js
  06-basic-json.js
  07-params-query.js
  08-middleware-basic.js
  09-middleware-use.js U
  10-middleware-options.js U
  11-methods.js U
  app.js M
    authorize.js U
    data.js M
    index.html
    logger.js M
  OUTLINE
  TIMELINE

app.js M x data.js M
02-express-tutorial > app.js > app.delete("/api/people/:id") callback
78   }
79   res
80     .status(400)
81     .json({ success: false, msg: "please enter name value", data: [] });
82   } else {
83     res.status(404).json({ success: false, msg: "No Resources Found" });
84   }
85 };
86
87 app.delete("/api/people/:id", (req, res) => [
88   const { id } = req.params;
89   console.log(Number(id));
90   if (peopleIds?.includes(Number(id))) {
91     const updatePeople = people.filter((person) => person.id !== Number(id));
92     res.status(203).json({ success: true, data: updatePeople });
93   } else {
94     res.status(404).json({ success: false, msg: "No Resources Found" });
95   }
96 ];
97
98 app.listen(5000, () => {
99   console.log(`Server is listening on Port 5000....`);
100 });

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[nodemon] starting `node app.js`
Server is listening on Port 5000....
```

In Postman, localhost:5000/api/people/1 deleting the first user.

The screenshot shows the Postman application interface. The left sidebar contains navigation links: Home, Workspaces, Reports, Explore, Scratch Pad, New, Import, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace displays a DELETE request to `localhost:5000/api/people/1`. The request details tab shows the method as `DELETE`, the URL as `localhost:5000/api/people/1`, and the body type as `raw`. The response tab shows a status of `203 Non-Authoritative Information` with a response time of `7 ms` and a size of `384 B`. The response body is displayed as JSON:

```
1   "success": true,
2   "data": [
3     {
4       "id": 2,
5       "name": "peter"
6     },
7     {
8       "id": 3,
9       "name": "anna"
10    }
]
```

In Postman, localhost:5000/api/people/2 deleting the second user.

The screenshot shows the Postman application interface. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible, along with 'Sign In' and 'Sign Up for Free'. Below the navigation is a search bar labeled 'Search Postman'. The main workspace is titled 'localhost:5000/api/people/2' and contains a 'DELETE' request. The 'Body' tab is selected, showing a JSON response with 'success: true' and 'data' containing two user objects with IDs 1 and 3. The status bar at the bottom indicates a 203 Non-Authoritative Information response with 22 ms and 383 B.

In Postman, localhost:5000/api/people/78 when user isn't available, we are providing 404 error.

The screenshot shows the Postman application interface. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible, along with 'Sign In' and 'Sign Up for Free'. Below the navigation is a search bar labeled 'Search Postman'. The main workspace is titled 'localhost:5000/api/people/78' and contains a 'DELETE' request. The 'Body' tab is selected, showing a JSON response with 'success: false' and 'msg: "No Resources Found"'. The status bar at the bottom indicates a 404 Not Found response with 5 ms and 286 B.

## Express Router – Setup

As we are having more routes and more functionality, we have another issue of app.js getting busy. The solution is using **Express Router** where we can group the routes together and then as far as the functionality, we can set them up as **controllers**.

Basically, when we connect with database in the next example, we use a pattern named MVC (Model View Controller). It is a nice pattern when we are setting up with the database. Here we are missing the model part in the MVC pattern because we haven't connected to the database.

In this example we are going to refactor to reduce the burden on app.js and make it lean.

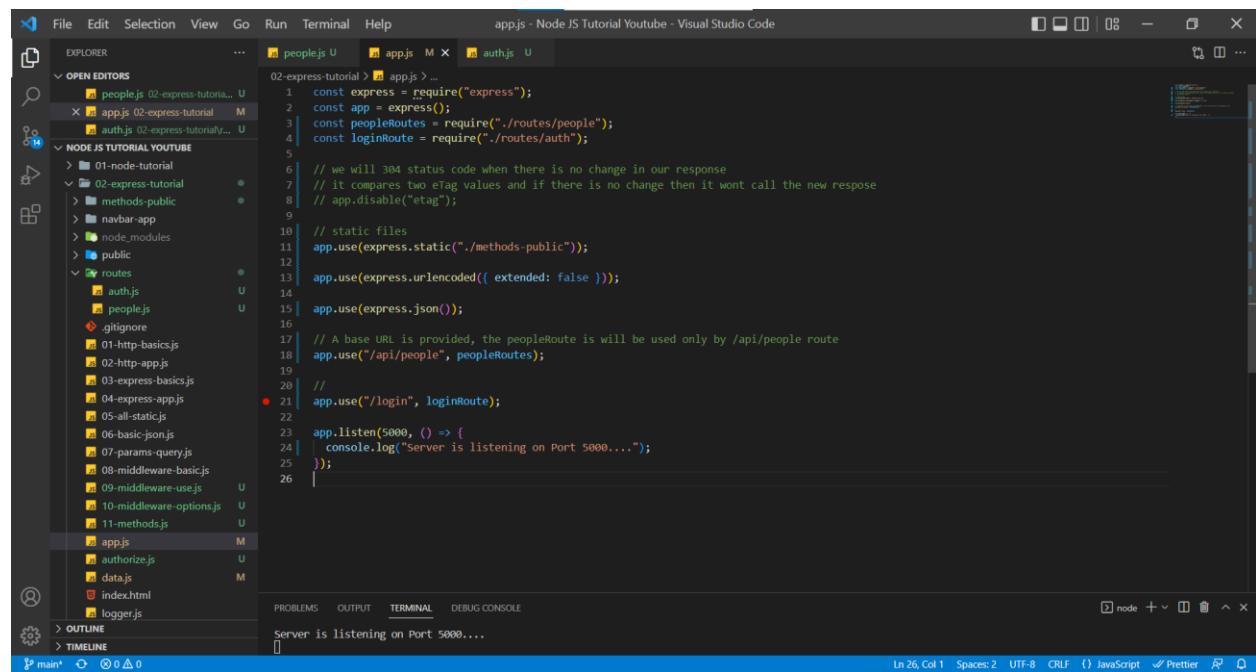
When we have a look at our code, we can most of the routes starting with /api/people and hence it would be a good idea to group those routes together and add one other login route (/login)

Let's create a folder and name it as routes as per common convention.

For login route, we are going with the filename auth.js and /api/people route we are using filename people.js

In people.js instead of setting up the app we go with router, and we explicitly get the router from express.

app.js



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

- File Explorer:** Shows the project structure under "NODE JS TUTORIAL YOUTUBE". The "routes" folder contains "auth.js", "people.js", and "index.html".
- Code Editor:** The active file is "app.js". The code imports express, creates an app, and sets up static files. It then uses the app.use() method to apply the "/api/people" route to the "peopleRoutes" object. Finally, it listens on port 5000.

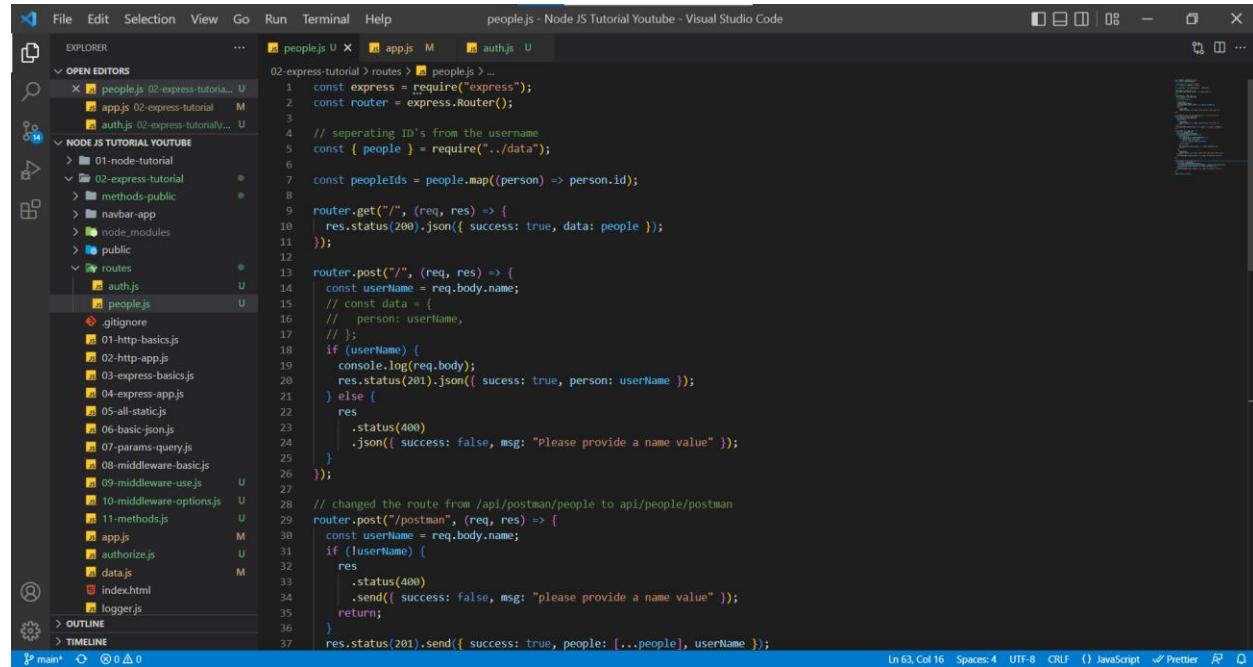
```
1 const express = require("express");
2 const app = express();
3 const peopleRoutes = require("./routes/people");
4 const loginRoute = require("./routes/auth");
5
6 // we will 304 status code when there is no change in our response
7 // it compares two eTag values and if there is no change then it won't call the new response
8 // app.disable("etag");
9
10 // static files
11 app.use(express.static("./methods-public"));
12
13 app.use(express.urlencoded({ extended: false }));
14
15 app.use(express.json());
16
17 // A base URL is provided, the peopleRoute is used only by /api/people route
18 app.use("/api/people", peopleRoutes);
19
20 //
21 app.use("/login", loginRoute);
22
23 app.listen(5000, () => {
24   console.log("Server is listening on Port 5000....");
25 });
26 |
```

- Terminal:** Shows the message "Server is listening on Port 5000...."
- Status Bar:** Shows the file is a JavaScript file (js), has 26 lines, and is in node mode.

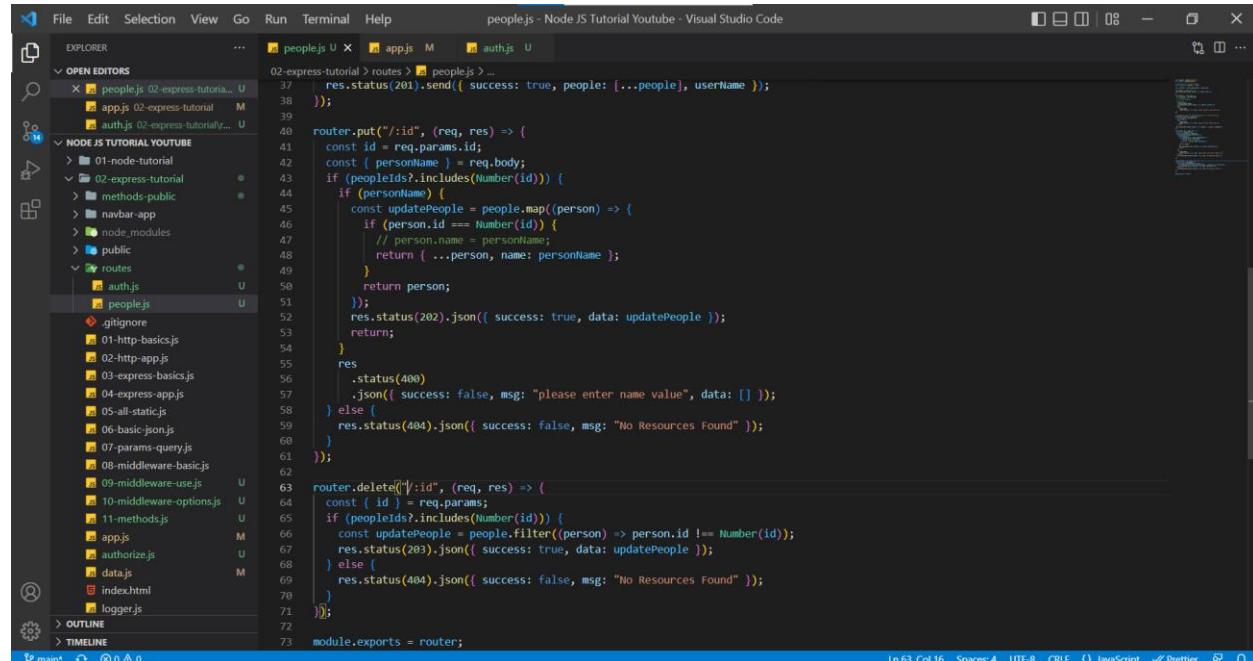
In app.js, we import the routes from each of the base route (/login and /api/public) and then we use the app.use() method to apply those routing routes and functionalities to the base URL.

## people.js

In people.js file we write all the routes starting with /api/people and their functionalities depending on the HTTP method.



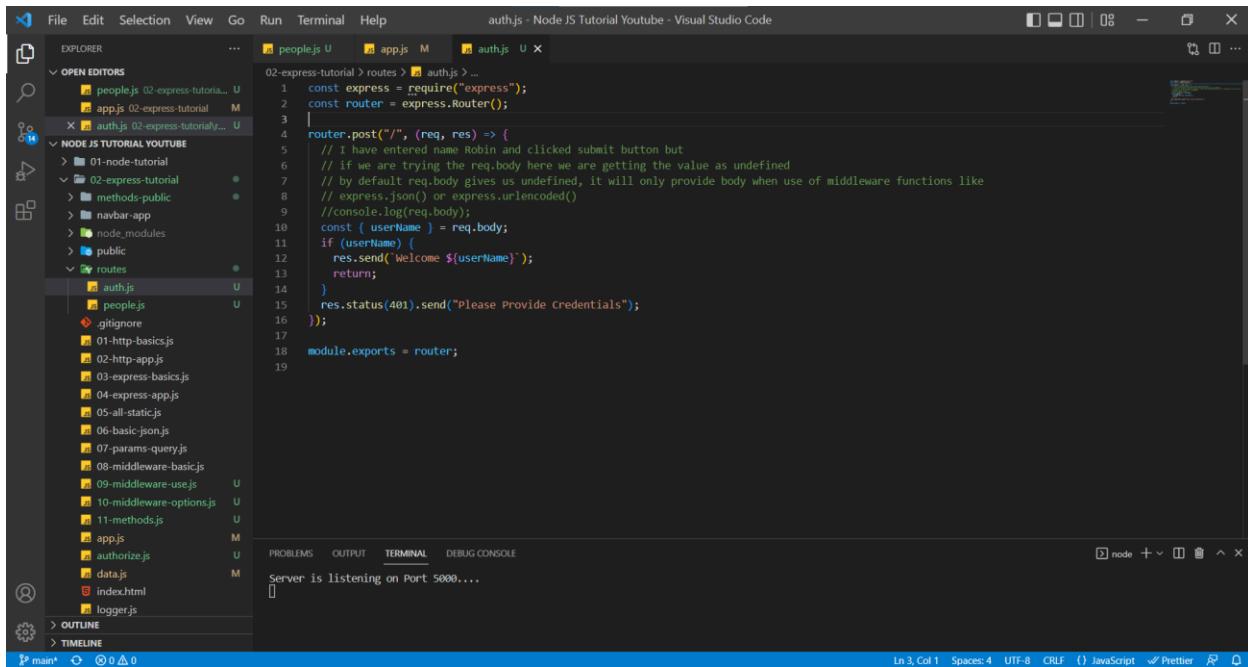
```
File Edit Selection View Go Run Terminal Help people.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS 02-express-tutorial > routes > people.js
1 const express = require("express");
2 const router = express.Router();
3
4 // separating ID's from the username
5 const { people } = require("../data");
6
7 const peopleIds = people.map((person) => person.id);
8
9 router.get("/", (req, res) => {
10   res.status(200).json({ success: true, data: people });
11 });
12
13 router.post("/", (req, res) => {
14   const userName = req.body.name;
15   // const data = {
16   //   person: userName,
17   // };
18   if (userName) {
19     console.log(req.body);
20     res.status(201).json({ sucess: true, person: userName });
21   } else {
22     res
23       .status(400)
24       .json({ success: false, msg: "Please provide a name value" });
25   }
26 });
27
28 // changed the route from /api/postman/people to api/people/postman
29 router.post("/postman", (req, res) => {
30   const userName = req.body.name;
31   if (userName) {
32     res
33       .status(400)
34       .send({ success: false, msg: "please provide a name value" });
35   }
36 }
37 res.status(201).send({ success: true, people: [...people], userName });
Ln 63, Col 16  Spaces: 4  UTF-8  CR/LF  () JavaScript  ✓ Prettier  ⌂  ⌂  ⌂
```



```
File Edit Selection View Go Run Terminal Help people.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS 02-express-tutorial > routes > people.js
37   res.status(201).send({ success: true, people: [...people], userName });
38 });
39
40 router.put("/:id", (req, res) => {
41   const id = req.params.id;
42   const { personName } = req.body;
43   if (peopleIds?.includes(Number(id))) {
44     if (personName) {
45       const updatePeople = people.map((person) => {
46         if (person.id === Number(id)) {
47           // person.name = personName;
48           return { ...person, name: personName };
49         }
50         return person;
51       });
52       res.status(202).json({ success: true, data: updatePeople });
53     } else {
54       res
55         .status(400)
56         .json({ success: false, msg: "please enter name value", data: [] });
57     } else {
58       res.status(404).json({ success: false, msg: "No Resources Found" });
59     }
60   });
61
62 router.delete("/:id", (req, res) => {
63   const { id } = req.params;
64   if (peopleIds?.includes(Number(id))) {
65     const updatePeople = people.filter((person) => person.id !== Number(id));
66     res.status(203).json({ success: true, data: updatePeople });
67   } else {
68     res.status(404).json({ success: false, msg: "No Resources Found" });
69   }
70 });
71
72 module.exports = router;
Ln 63, Col 16  Spaces: 4  UTF-8  CR/LF  () JavaScript  ✓ Prettier  ⌂  ⌂  ⌂
```

## auth.js

In auth.js file, we write the functionality for routes starting with /login.



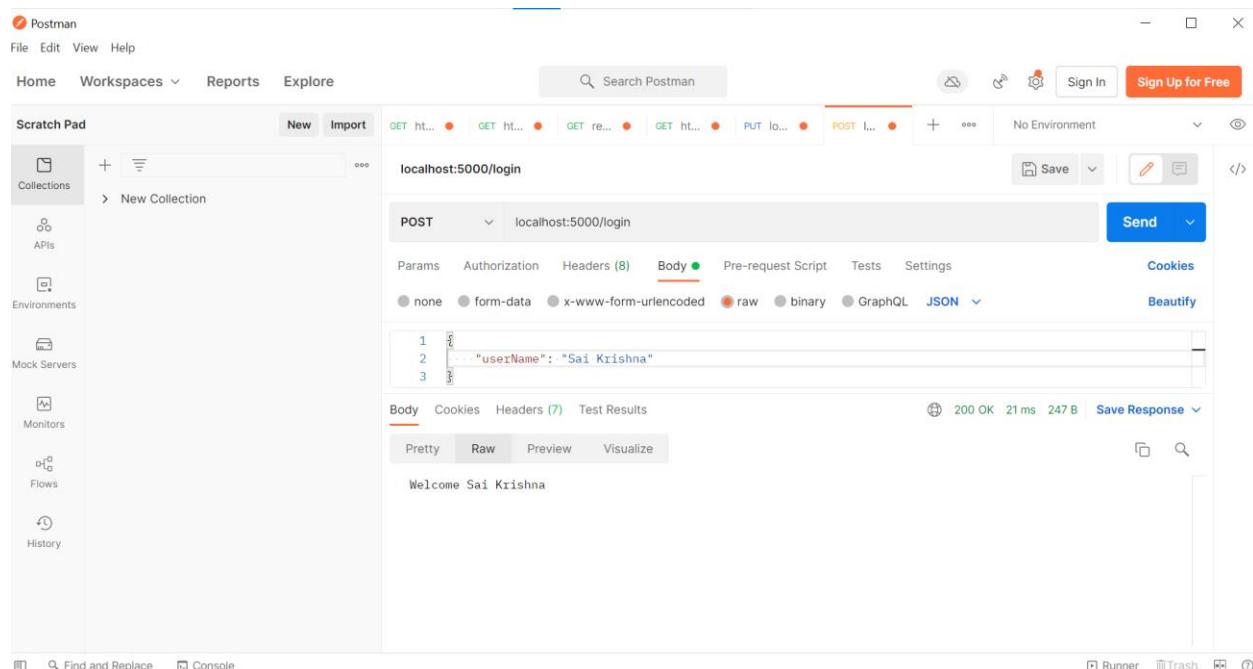
The screenshot shows the Visual Studio Code interface with the auth.js file open in the editor. The code defines a POST route for the root path ('/') that checks if a user has entered a name and sends a welcome message if so, or a 401 error otherwise. The code is as follows:

```
const express = require("express");
const router = express.Router();
router.post("/", (req, res) => {
  // I have entered name Robin and clicked submit button but
  // if we are trying the req.body here we are getting the value as undefined
  // by default req.body gives us undefined, it will only provide body when use of middleware functions like
  // express.json() or express.urlencoded()
  //console.log(req.body);
  const { userName } = req.body;
  if (userName) {
    res.send(`Welcome ${userName}`);
    return;
  }
  res.status(401).send("Please Provide Credentials");
});
module.exports = router;
```

The process of using the routes and MVC pattern makes the similar routes to be grouped together and maintain lean and clean code in app.js file.

After Using Routing,

In Postman, localhost:5000/login



The screenshot shows the Postman application interface. A POST request is made to the URL 'localhost:5000/login'. The request body is set to 'JSON' and contains the key-value pair 'userName': 'Sai Krishna'. The response status is 200 OK, and the response body is 'Welcome Sai Krishna'.

In Postman, localhost:5000/api/people/1

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, Reports, Explore, Scratch Pad, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area shows a collection named 'localhost:5000/api/people/1'. A specific request is selected: a 'DELETE' request to 'localhost:5000/api/people/1'. The 'Body' tab is active, showing a JSON response with two data objects. The response body is:

```
1
2   "success": true,
3   "data": [
4     {
5       "id": 2,
6       "name": "peter"
7     },
8     {
9       "id": 3,
10      "name": "ben"
11    }
12  ]
```

The status bar at the bottom indicates a 203 Non-Authoritative Information response with 9 ms and 384 B.

## Express Router – Controllers

If we have a look at people.js it is still somewhat a mess because we have more routes and bunch of functionalities.

A better setup would be if we could split this into separate file (functions).

To maintain a clean code in routes/people.js we create a new folder named controllers where we refactor the code.

## app.js

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

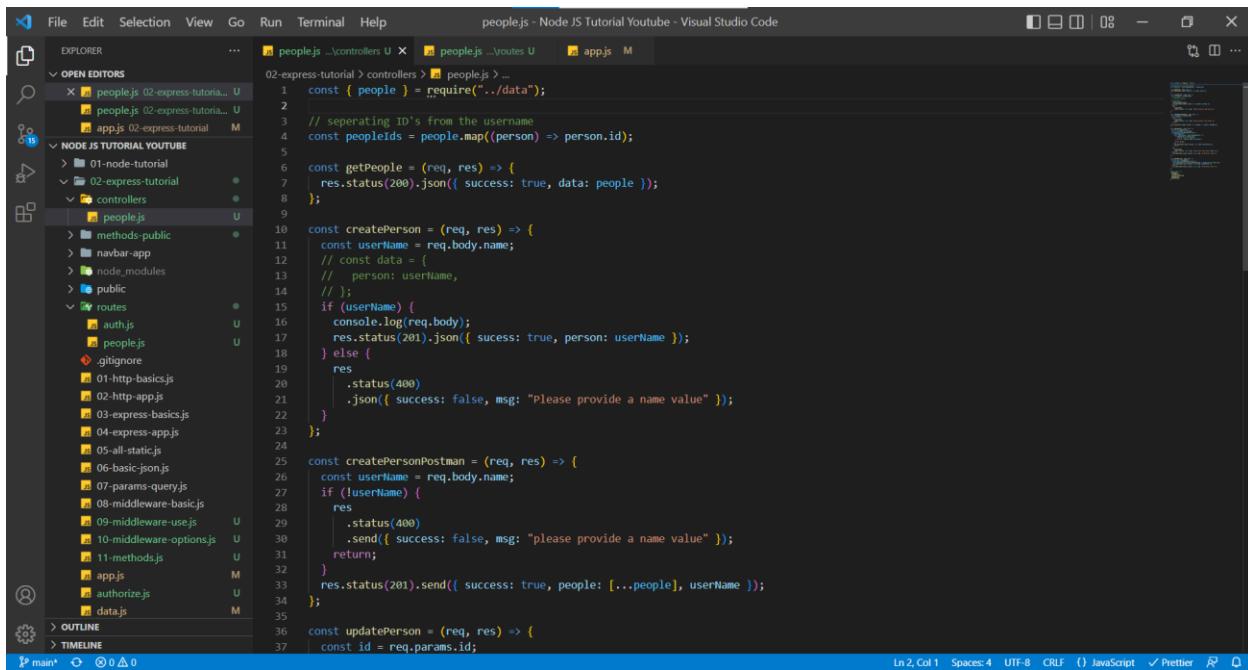
- File Explorer (Left):** Shows a tree view of the project structure under "OPEN EDITORS". The "NODE JS TUTORIAL YOUTUBE" folder contains several subfolders and files, including "01-node-tutorial", "02-express-tutorial" (which is expanded), "controllers", "methods-public", "public", "routes", and various "js" files like "app.js", "people.js", "auth.js", etc.
- Code Editor (Center):** Displays the content of the file "app.js" from the "02-express-tutorial/controllers" folder. The code sets up an Express app, includes middleware for static files and JSON parsing, and defines routes for people and login.
- Terminal (Bottom):** Shows the output of running the application with node, indicating it's listening on port 5000.

routes/people.js (Routes are also made lean and clean) we import the functions from controllers and then use them as per the route.

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

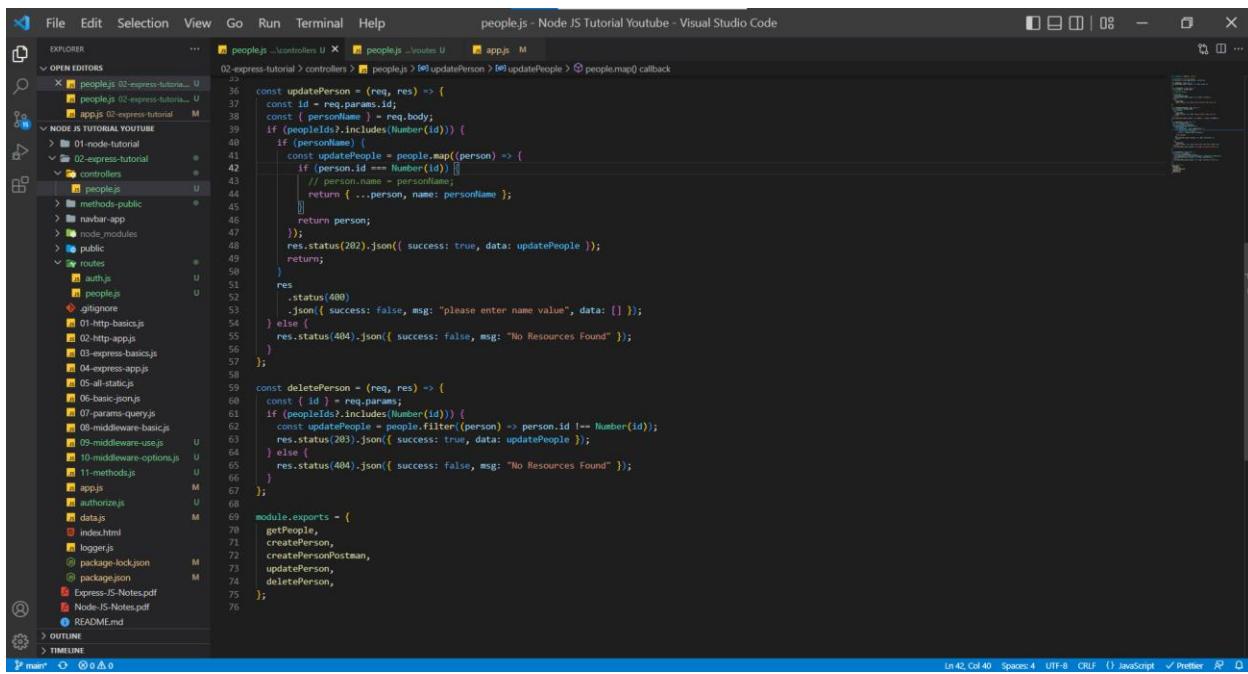
- File Explorer (Left):** Shows the project structure under "OPEN EDITORS".
  - NODE JS TUTORIAL YOUTUBE:** Contains folders like "01-node-tutorial", "02-express-tutorial", and "03-express-apis".
  - 02-express-tutorial:** Contains "controllers" (with "people.js"), "methods-public", "navbar-app", "node\_modules", "public", and "routes" (with "auth.js" and "people.js").
  - people.js:** Opened in the editor, showing code for an Express router.
- Code Editor (Center):** Displays the content of "people.js". The code defines a Router for handling people-related requests using the express module.
- Terminal (Bottom):** Shows the output of running the application with node app.js, indicating the server is listening on port 5000.
- Status Bar (Bottom):** Shows the current file is people.js, the line number is 17, the column number is 1, and the status includes "Spaces: 2", "UTF-8", "CRLF", "JavaScript", and "Prettier".

## controllers/people.js



```
File Edit Selection View Go Run Terminal Help peoplejs ...controllers > people.js > routes app.js M
OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
controllers
people.js
methods-public
navbar-app
node_modules
public
routes
auth.js
people.js
people.js
.peopleignore
01-http-basics.js
02-http-app.js
03-express-basics.js
04-express-app.js
05-all-static.js
06-basic-json.js
07-params-query.js
08-middleware-basic.js
09-middleware-use.js
10-middleware-options.js
11-methods.js
app.js
authorize.js
data.js
OUTLINE
TIMELINE
Ln 2, Col 1 | Spaces: 4 | UTF-8 | CRLF | JavaScript | ✓ Prettier
```

```
02-express-tutorial > controllers > people.js > ...
1 const { people } = require("./data");
2
3 // separating ID's from the username
4 const peopleIds = people.map(person) => person.id;
5
6 const getPeople = (req, res) => {
7   res.status(200).json({ success: true, data: people });
8 };
9
10 const createPerson = (req, res) => {
11   const username = req.body.name;
12   // const data = {
13   //   person: username,
14   // };
15   if (username) {
16     console.log(req.body);
17     res.status(201).json({ success: true, person: username });
18   } else {
19     res
20     .status(400)
21     .json({ success: false, msg: "Please provide a name value" });
22   }
23 };
24
25 const createPersonPostman = (req, res) => {
26   const userName = req.body.name;
27   if (!userName) {
28     res
29     .status(400)
30     .send({ success: false, msg: "please provide a name value" });
31   }
32   res.status(201).send({ success: true, people: [...people], userName });
33 };
34
35 const updatePerson = (req, res) => {
36   const id = req.params.id;
37 }
```



```
File Edit Selection View Go Run Terminal Help people.js ...controllers > people.js > routes app.js M
OPEN EDITORS
NODE JS TUTORIAL YOUTUBE
01-node-tutorial
02-express-tutorial
controllers
people.js
methods-public
navbar-app
node_modules
public
routes
auth.js
people.js
.peopleignore
01-http-basics.js
02-http-app.js
03-express-basics.js
04-express-app.js
05-all-static.js
06-basic-json.js
07-params-query.js
08-middleware-basic.js
09-middleware-use.js
10-middleware-options.js
11-methods.js
app.js
authorize.js
data.js
index.html
logger.js
package-lock.json
package.json
Express-JS-Notes.pdf
Node-JS-Notes.pdf
README.md
OUTLINE
TIMELINE
Ln 42, Col 40 | Spaces: 4 | UTF-8 | CRLF | JavaScript | ✓ Prettier
```

```
02-express-tutorial > controllers > people.js > updatePerson > people.map() callback
const updatePerson = (req, res) => {
  const id = req.params.id;
  const { personName } = req.body;
  if (peopleIds?.includes(Number(id))) {
    if (personName) {
      const updatePeople = people.map((person) => {
        if (person.id === Number(id)) {
          // person.name = personName;
          return { ...person, name: personName };
        }
        return person;
      });
      res.status(202).json({ success: true, data: updatePeople });
      return;
    }
    res
      .status(400)
      .json({ success: false, msg: "please enter name value", data: [ ] });
  } else {
    res.status(404).json({ success: false, msg: "No Resources Found" });
  }
};

const deletePerson = (req, res) => {
  const { id } = req.params;
  if (peopleIds?.includes(Number(id))) {
    const updatePeople = people.filter((person) => person.id !== Number(id));
    res.status(203).json({ success: true, data: updatePeople });
  } else {
    res.status(404).json({ success: false, msg: "No Resources Found" });
  }
};

module.exports = {
  getPeople,
  createPerson,
  createPersonPostman,
  updatePerson,
  deletePerson,
};
```

Controllers folder contains all the functionality code ensuring our code is lean and clean.

We have created controller for /api/people route because there are lot of functionalities but /login route has only one functionality, hence we didn't create a controller for that route.

In Postman, localhost:5000/api/people/2 with DELETE Method.

The screenshot shows the Postman application interface. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible, along with 'Sign In' and 'Sign Up for Free'. Below the navigation is a search bar labeled 'Search Postman'. The main workspace is titled 'localhost:5000/api/people/2'. A 'DELETE' method is selected. The 'Body' tab is active, showing a JSON response with two objects: one for 'john' and one for 'susan'. The response status is 203 Non-Authoritative Information. The bottom section displays the raw JSON response:

```
1
2 "success": true,
3 "data": [
4   {
5     "id": 1,
6     "name": "john"
7   },
8   {
9     "id": 3,
10    "name": "susan"
11 }
```

In Postman, localhost:5000/api/people with GET Method

The screenshot shows the Postman application interface. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible, along with 'Sign In' and 'Sign Up for Free'. Below the navigation is a search bar labeled 'Search Postman'. The main workspace is titled 'localhost:5000/api/people'. A 'GET' method is selected. The 'Body' tab is active, showing a JSON response with five objects: 'john', 'peter', 'susan', 'anna', and 'watson'. The response status is 200 OK. The bottom section displays the raw JSON response:

```
1
2 {"success":true,"data":[{"id":1,"name":"john"}, {"id":2,"name":"peter"}, {"id":3,"name":"susan"}, {"id":4,"name":"anna"}, {"id":5,"name":"watson"}]}
```

We can set the routes in two methods,

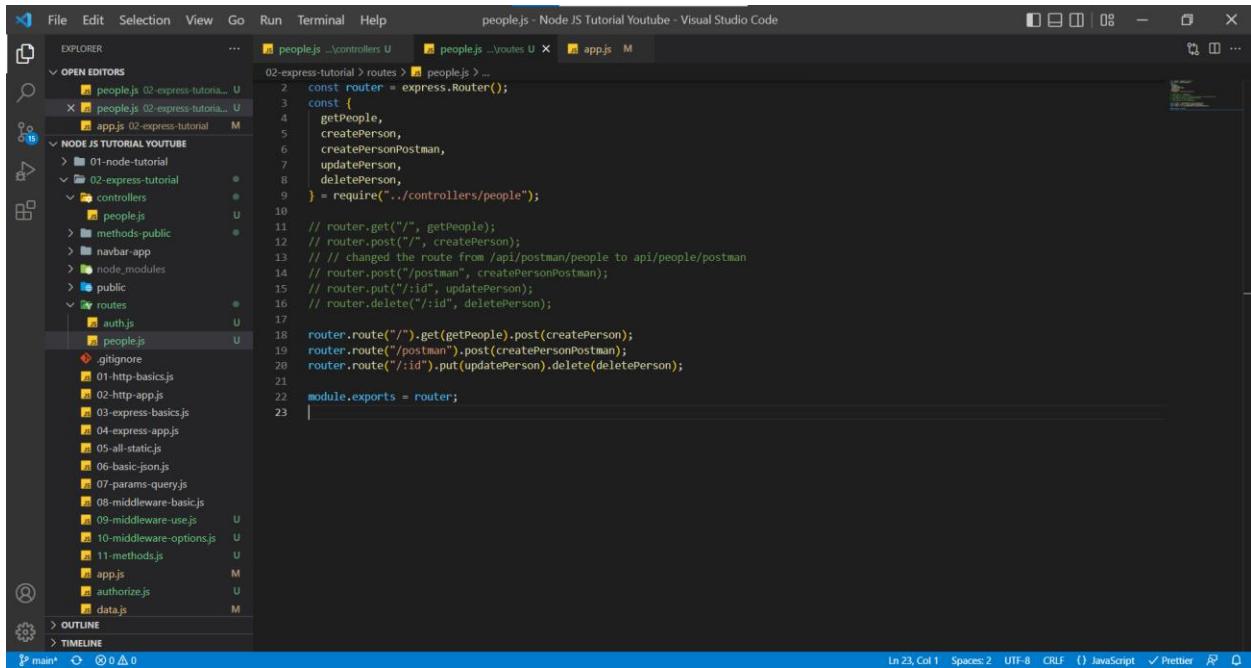
1. `router.get("/", getPeople); // we can set the route for each different HTTP method`

```
router.get("/", getPeople);
router.post("/", createPerson);
// changed the route from /api/postman/people to api/people/postman
router.post("/postman", createPersonPostman);
router.put("/:id", updatePerson);
router.delete("/:id", deletePerson);
```

2. if we have same the URL with different HTTP methods then we can chain them using `router.route()` method.

```
router.route("/").get(getPeople).post(createPerson);
router.route("/postman").post(createPersonPostman);
router.route("/:id").put(updatePerson).delete(deletePerson);
```

## routes/people.js



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

- File Explorer:** Shows the project structure under "NODE JS TUTORIAL YOUTUBE". The "routes" folder contains "people.js". Other files like "auth.js", "methods-public.js", and "navbar-app.js" are also listed.
- Code Editor:** The active file is "people.js" located at "02-express-tutorial > routes > people.js". The code implements a Router and defines various HTTP methods for the "/people" endpoint, including a POST route for "postman" and separate PUT and DELETE routes for individual person IDs.
- Status Bar:** Shows file statistics: Ln 23, Col 1, Spaces: 2, UTF-8, CR LF, JavaScript, Prettier.

```
02-express-tutorial > routes > people.js > ...
1 const router = express.Router();
2 const {
3   getPeople,
4   createPerson,
5   createPersonPostman,
6   updatePerson,
7   deletePerson,
8 } = require("../controllers/people");
9
10 // router.get("/", getPeople);
11 // router.post("/", createPerson);
12 // changed the route from /api/postman/people to api/people/postman
13 // router.post("/postman", createPersonPostman);
14 // router.put("/:id", updatePerson);
15 // router.delete("/:id", deletePerson);
16
17 router.route("/").get(getPeople).post(createPerson);
18 router.route("/postman").post(createPersonPostman);
19 router.route("/:id").put(updatePerson).delete(deletePerson);
20
21 module.exports = router;
22
```

In Postman, localhost:5000/api/people with GET Method

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, Reports, Explore, Scratch Pad, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area is titled "localhost:5000/api/people". A GET request is selected with the URL "localhost:5000/api/people". The Headers tab shows "Content-Type: application/json". The Body tab contains the JSON response: [{"id":1,"name":"john"}, {"id":2,"name":"peter"}, {"id":3,"name":"susan"}, {"id":4,"name":"anna"}, {"id":5,"name":"watson"}]. The status bar at the bottom indicates a 200 OK response with 23 ms and 380 B.

In Postman, localhost:5000/api/people/2 with DELETE Method

The screenshot shows the Postman application interface. The sidebar and main area are identical to the previous GET request, but the method dropdown now shows "DELETE". The URL is "localhost:5000/api/people/2". The Headers tab shows "Content-Type: application/json". The Body tab contains the JSON response: [{"id":1,"name":"john"}, {"id":3,"name":"susan"}, {"id":4,"name":"anna"}, {"id":5,"name":"watson"}]. The status bar at the bottom indicates a 203 Non-Authoritative Information response with 12 ms and 383 B.

**End of Express Tutorial Module**

**Until Here we are done with Express Fundamentals**