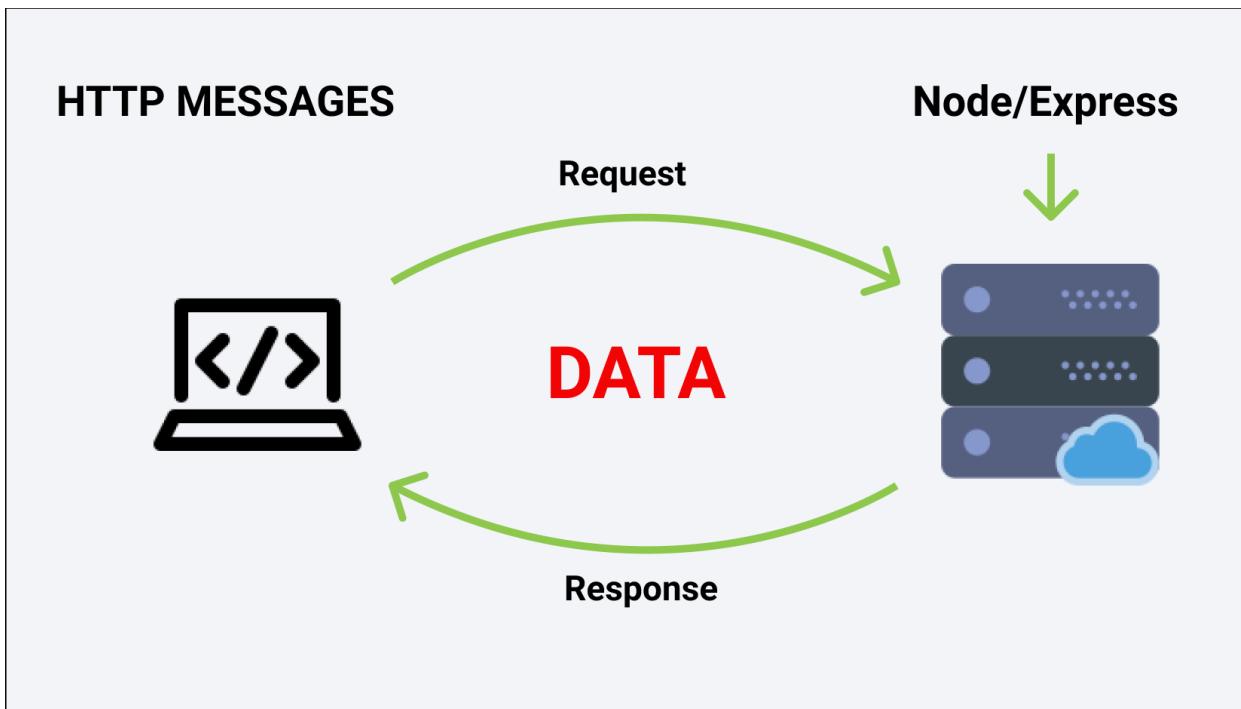


# Express

## 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.

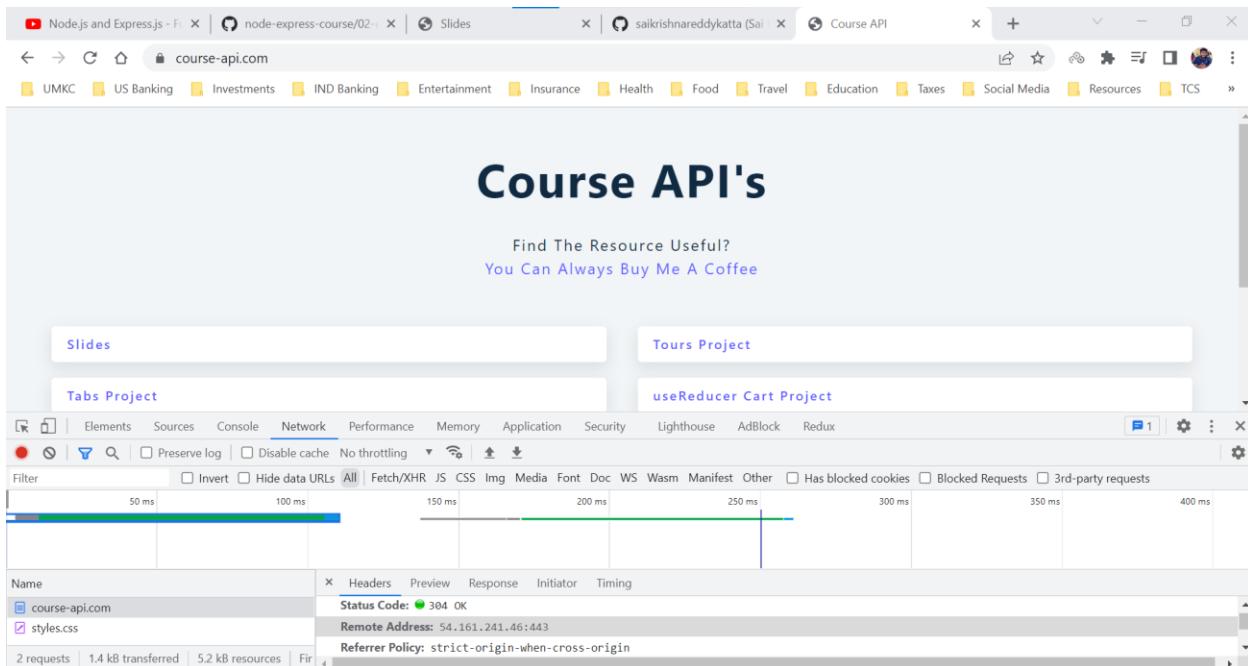
Port Number 21: Used for File Transfer Protocol (FTP) Command Control.

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.**

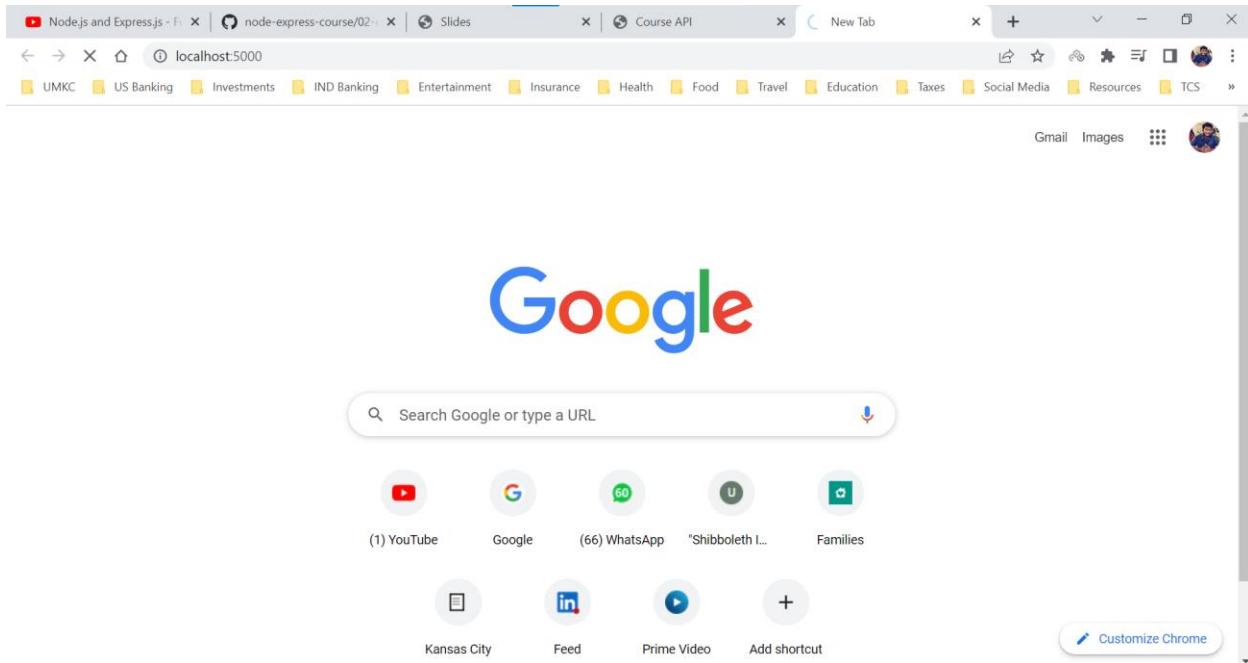
When we don't send any response to the server. We are just console logging the information.

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

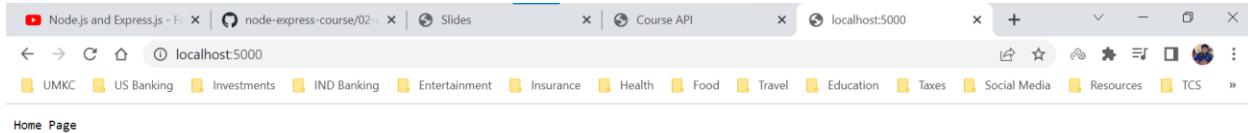
The screenshot shows the Visual Studio Code interface. The left sidebar displays a file tree with files like `app.js`, `package-lock.json`, and `package.json`. The main editor window shows the following code:

```
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.***

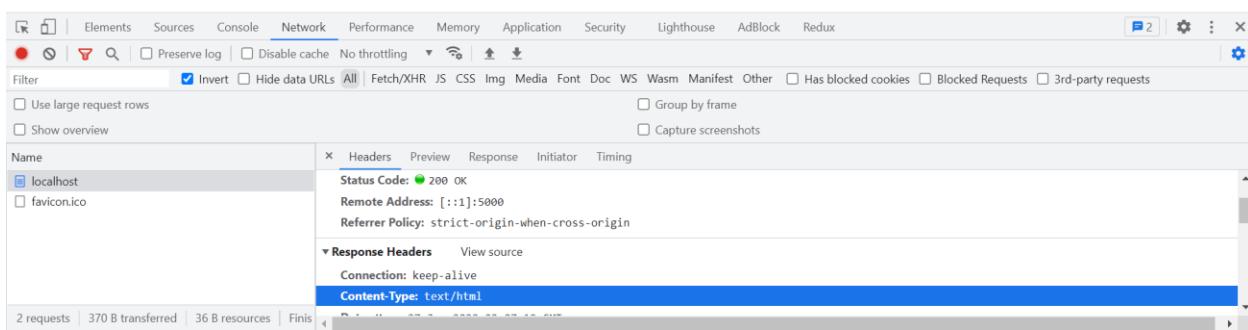
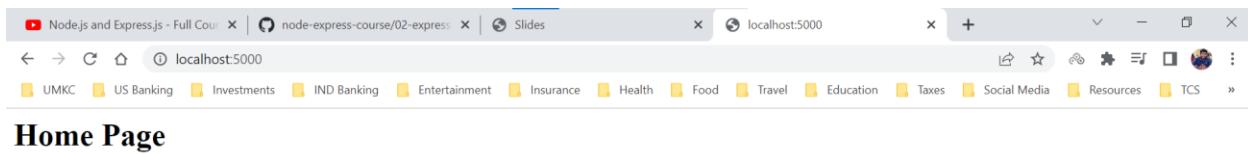
## app.js

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders: 01-node-tutorial, 02-express-tutorial, node\_modules, .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:

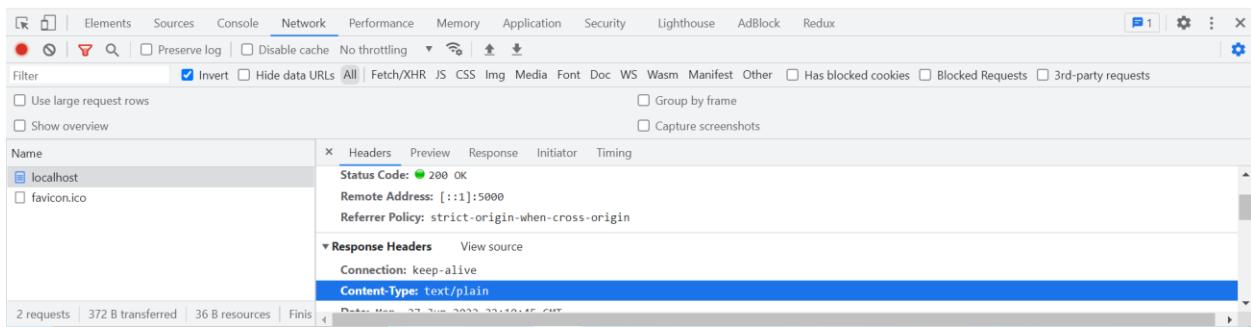
```
User hit the server
User hit the server
User hit the server
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
```

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.

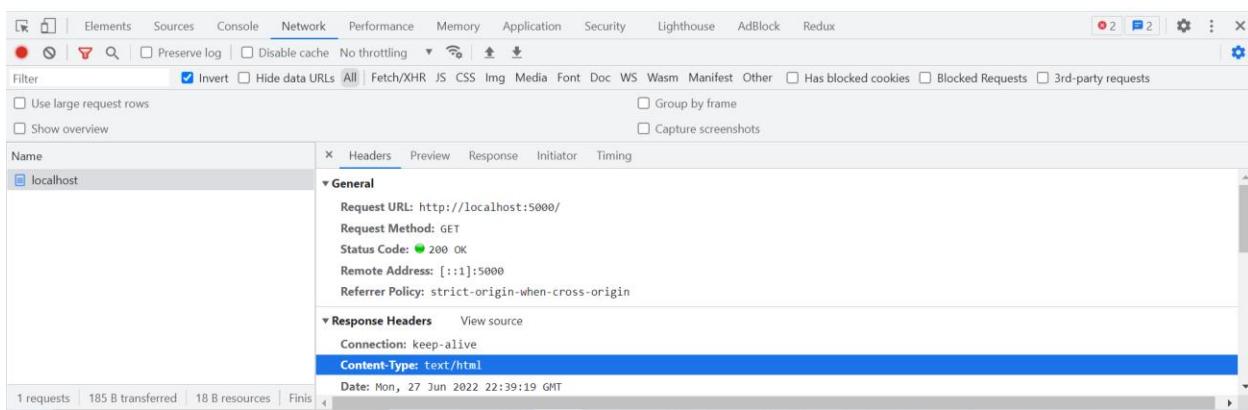
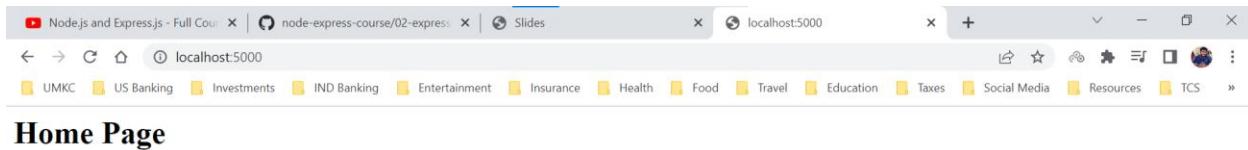
Forward Slash would be the Home Page ("/").

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

## app.js

```
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

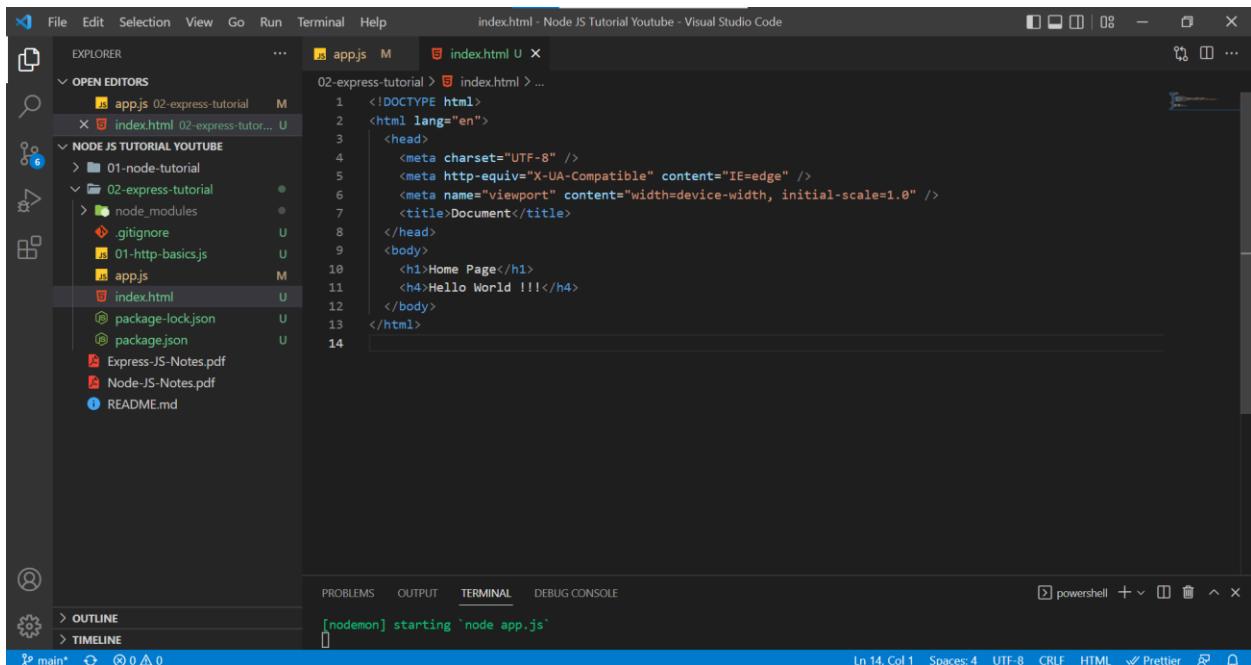
We are not limited to send the HTML directly into `response.write()` method or `response.end()` method. Instead, we can set up a file, request the file using **File System** and just passing it.

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);
```

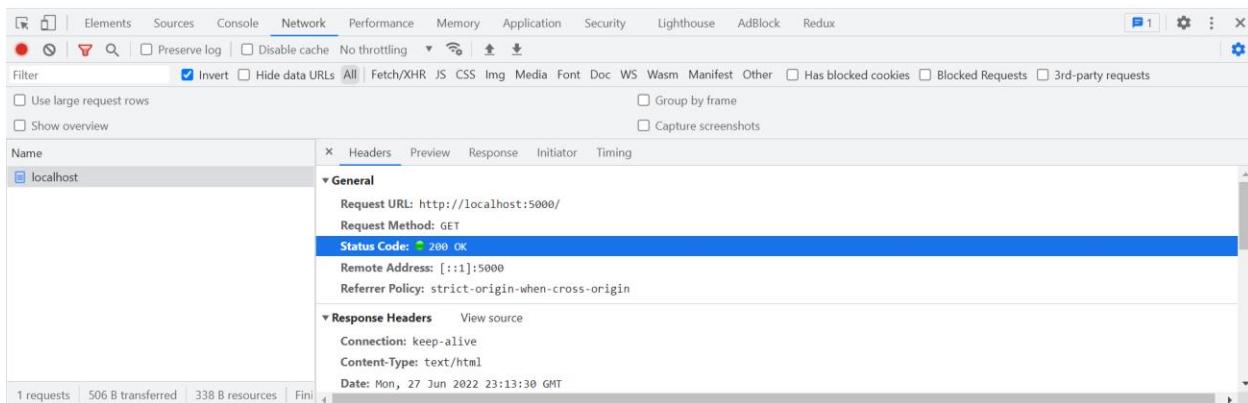
- Terminal:** Shows the command: [nodemon] starting 'node app.js'
- Status Bar:** Shows settings like Ln 22, Col 1, Spaces 4, UTF-8, CRLF, JavaScript, and Prettier.

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 pane is expanded, showing the following headers:

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

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

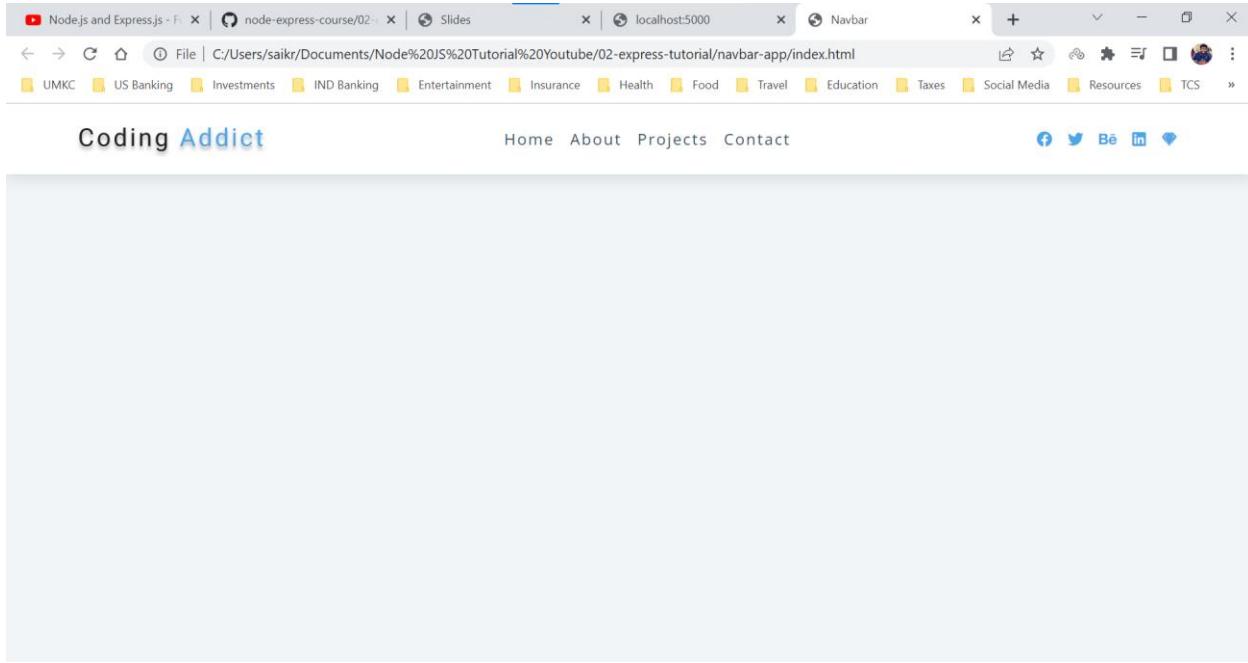
## HTTP – App Example

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 following details:

- File Explorer:** Shows the project structure with files like `app.js`, `index.html`, `logo.svg`, `styles.css`, `01-node-tutorial`, and `02-express-tutorial`.
- Editor:** The `app.js` file is open, displaying Node.js code for an Express application. The code handles requests for the homepage, contact page, and 404 errors.
- Terminal:** The terminal shows output from the nodemon command, indicating it's watching for changes and restarting the node app.js process.
- Status Bar:** Shows the current file is `main*`, has 0 changes, and the status bar includes information about line count, spaces, encoding, and Prettier.

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

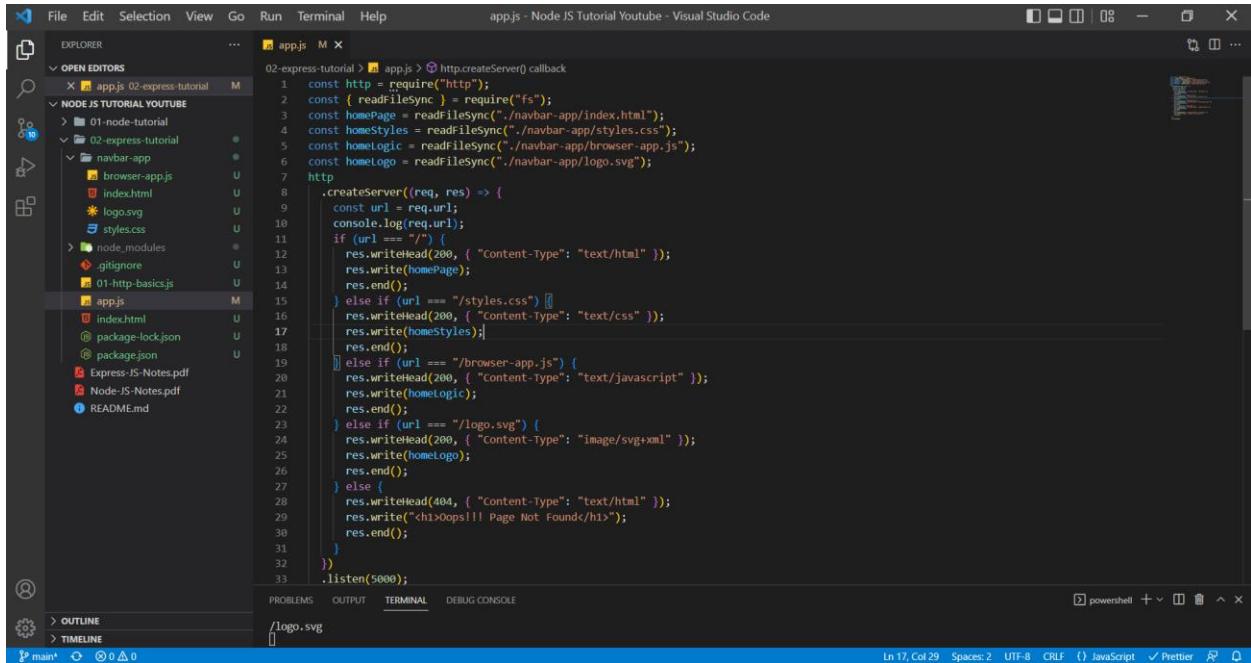
Web Application is requesting resources like (index.html, styles.css, logo.svg, browser-app.js) but as of now we are serving the index.html request, hence other requests are receiving 404 error response. We are not handling those requests (styles.css, logo.svg, browser-app.js) in our server.

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 http = 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 === "/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);
```

We need to handle those requests and provide responses.

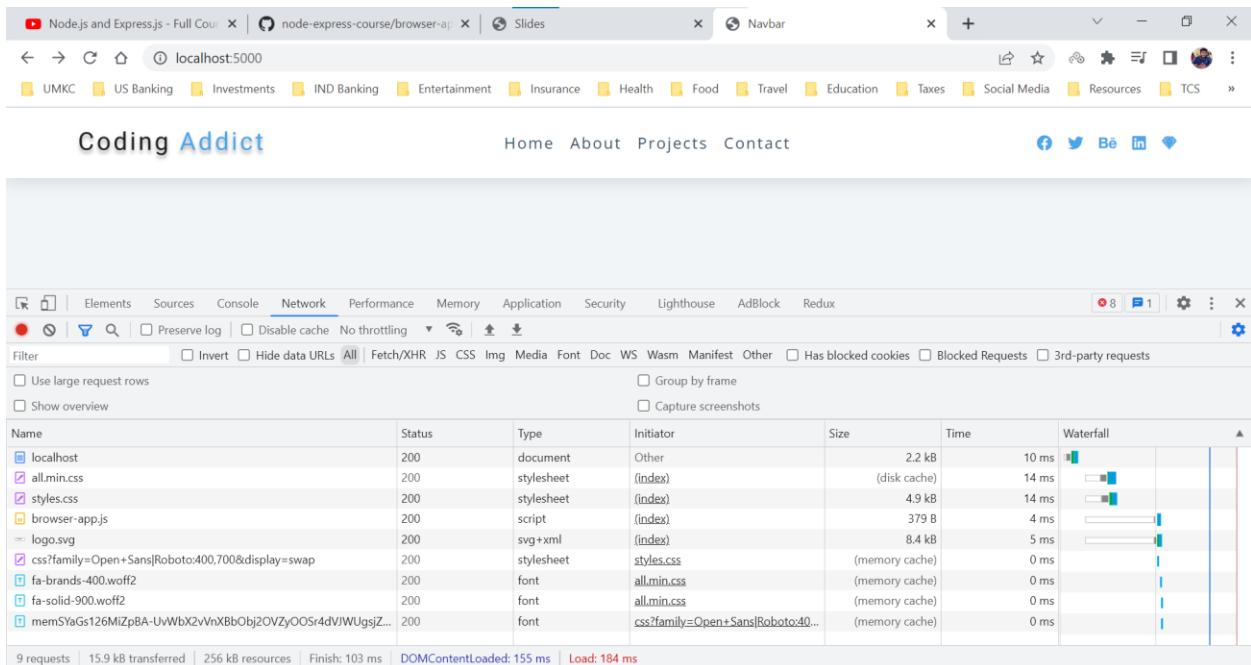
## app.js



```
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 three tabs: 'Node.js and Express.js - Full Cou...', 'node-express-course/browser-a...', and 'localhost:5000/info'. The active tab is 'localhost:5000/info' and displays the error message 'Oops!!! Page Not Found'. Below the browser is the Chrome DevTools Network tab. The table in the Network tab shows one request:

Name	Status	Type	Initiator	Size	Time	Waterfall
info	404	document	Other	205 B	8 ms	

At the bottom of the DevTools, 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

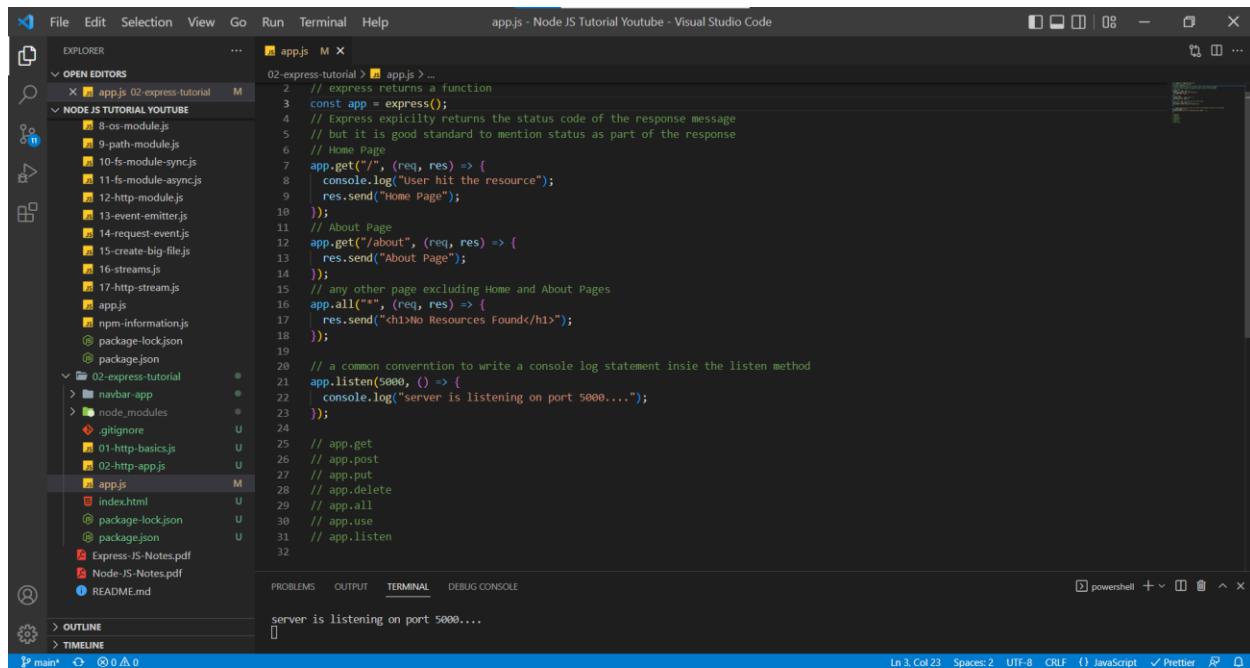
HTTP methods are also known as HTTP verbs, and it is the important part of the HTTP request message that we look for.

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
  OUTLINE
  TIMELINE

app.js M
02-express-tutorial > app.js > ...
2 // express returns a function
3 const app = express();
4 // Express explicitly returns the status code of the response message
5 // but it is good standard to mention status as part of the response
6 // Home Page
7 app.get("/", (req, res) => {
8   console.log("User hit the resource");
9   res.send("Home Page");
10 });
11 // About Page
12 app.get("/about", (req, res) => {
13   res.send("About Page");
14 });
15 // any other page excluding Home and About Pages
16 app.all("*", (req, res) => {
17   res.send("<h1>No Resources Found</h1>");
18 });
19
20 // a common convention to write a console log statement inside the listen method
21 app.listen(5000, () => {
22   console.log("server is listening on port 5000....");
23 });
24
25 // app.get
26 // app.post
27 // app.put
28 // app.delete
29 // app.all
30 // app.use
31 // app.listen
32

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's a footer with links to various categories.

Below the browser window is the developer tools Network tab interface. It shows a single request for 'localhost'. The 'General' section indicates a Request URL of `http://localhost:5000/`, a Request Method of `GET`, and a Status Code of `304 OK`. The 'Response Headers' section includes `Connection: keep-alive` and `Date: Tue, 28 Jun 2022 01:55:40 GMT`. The bottom of the Network tab shows summary statistics: 1 requests, 177 B transferred, and 9 B resources.

## About Page

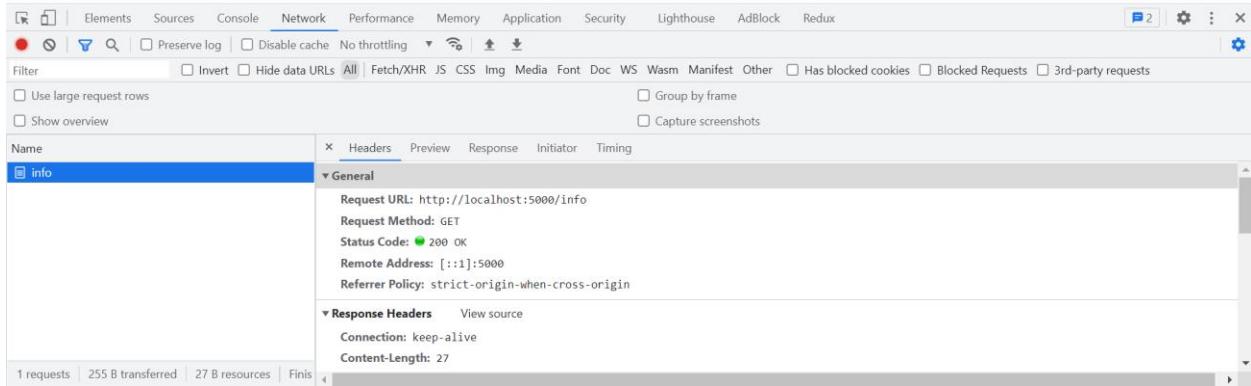
The screenshot shows a browser window with tabs for 'Node.js and Express.js - Full Cou...', 'node-express-course/02-express', and 'Slides'. The active tab is 'localhost:5000/about'. Below the tabs is a navigation bar with links like 'UMKC', 'US Banking', 'Investments', etc. The main content area displays the text 'About Page'. At the bottom, there's a footer with links to various categories.

Below the browser window is the developer tools Network tab interface. It shows a request for 'about'. The 'General' section indicates a Request URL of `http://localhost:5000/about`, a Request Method of `GET`, and a Status Code of `200 OK`. The 'Response Headers' section includes `Connection: keep-alive` and `Content-Length: 10`. The bottom of the Network tab shows summary statistics: 1 requests, 237 B transferred, and 10 B resources.

#### 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

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

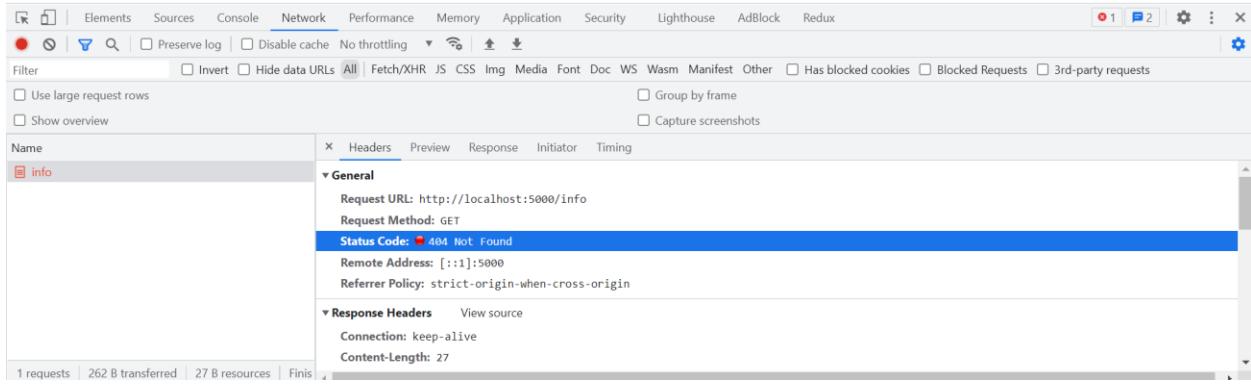
- File Explorer:** On the left, it shows the project structure. The root folder is "02-express-tutorial". Inside, there are subfolders like "02-express-tutorial", "node\_modules", and "02-express-tutorial". The file "app.js" is selected and has a "Modified" status indicator.
- Code Editor:** The main area displays the content of "app.js". The code sets up an Express.js application, defines routes for Home and About pages, and handles other requests by returning a 404 error.
- Terminal:** At the bottom, the terminal window shows the command "node app.js" being run and the message "server is listening on port 5000....".

```
const express = require("express");
// express returns a function
const app = express();
// Express explicitly returns the status code of the response message
// but it is good standard to mention status as part of the response
// Home Page
app.get("/", (req, res) => {
  console.log("User hit the resource");
  res.status(200).send("Home Page");
});
// About Page
app.get("/about", (req, res) => {
  res.status(200).send("About Page");
});
// any other page excluding Home and About Pages
app.all("*", (req, res) => {
  res.status(404).send("<h1>No Resources Found</h1>");
});
// a common convention to write a console log statement inside the listen method
app.listen(5000, () => {
  console.log("server is listening on port 5000....");
});
// app.get
// app.post
// app.put
// app.delete
// app.all
// app.use
// app.listen
```

#### Resources which are not handled



## No Resources Found



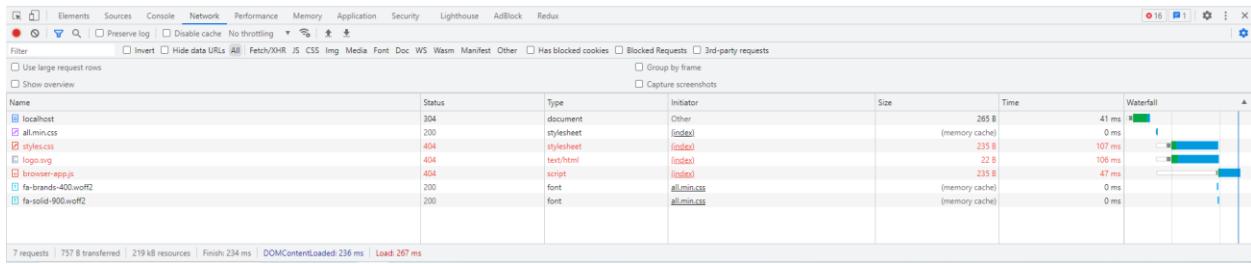
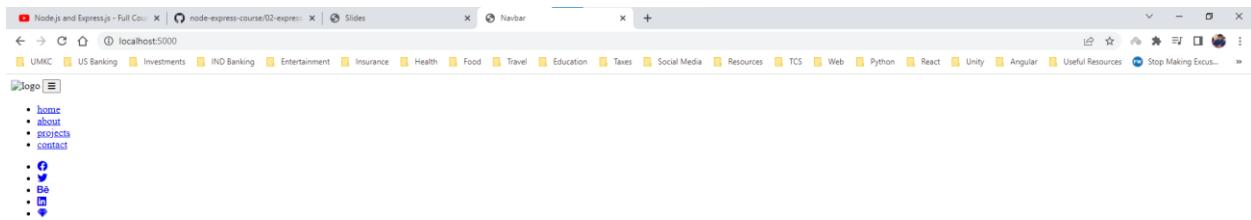
*As we can it is a way less code when compared to creating a server using built-in HTTP module.*

## 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

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

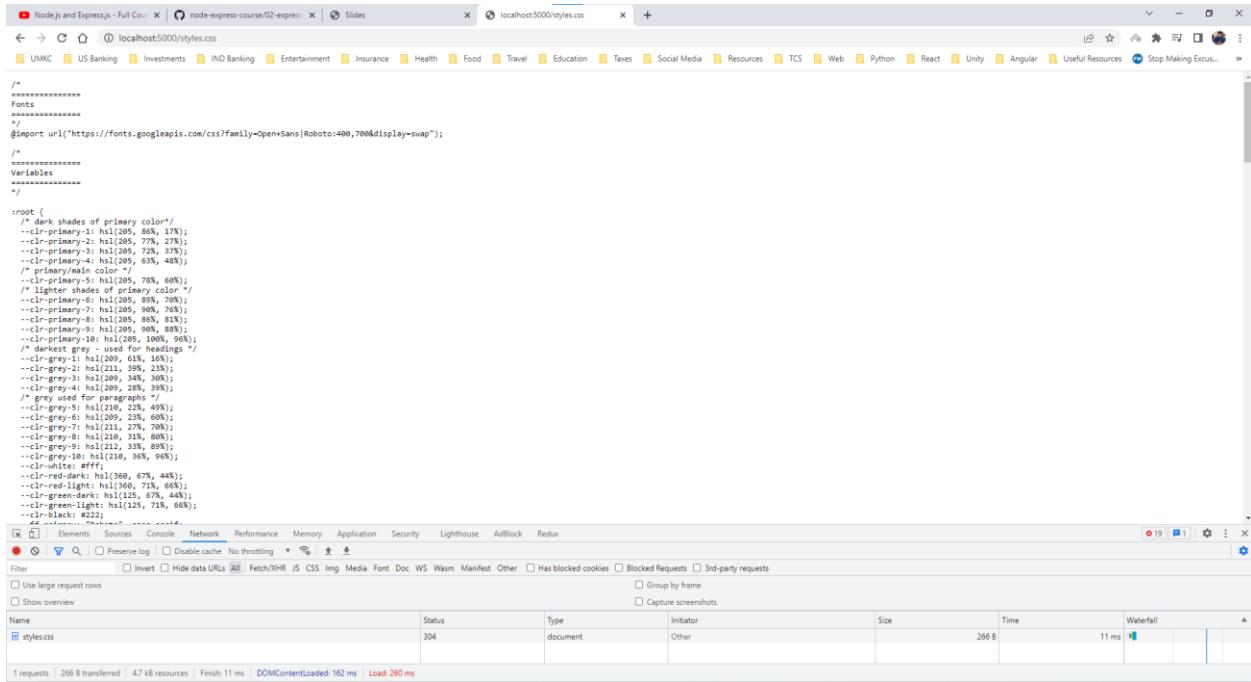
localhost:5000

The screenshot shows a browser window displaying the `Coding Addict` homepage. Below the browser is the developer tools Network tab, which shows a list of requests made to the server. The table includes columns for Name, Status, Type, Initiator, and Size. A detailed timeline for one specific request is shown on the right, indicating a duration of 0.15 ms.

Name	Status	Type	Initiator	Size
localhost	304	document	Other	265
all-min.css	200	stylesheet	(index)	(memory cache)
style.css	200	stylesheet	(index)	5.1 K
logos.png	200	image	(index)	8.5 K
browser-app.js	200	script	(index)	534
czs7am1y-cOpen+SansjRoboto-400,700&display=swap	200	stylesheet	zhiles.css	(disk cache)
fa-brands-400woff2	200	font	all-min.css	(memory cache)
fa-solid-900.woff2	200	font	all-min.css	(memory cache)
mem5t9aG128MZp8A-UwBx2VnX8bOs2Ov2yOC54dWUJgsZ084gaVLuo#2	200	font	czs7am1y-cOpen+SansjRoboto-400,700&display=swap	(disk cache)

Queued at 58.04 ms  
Started at 58.04 ms  
Connection Start  
Stalled  
Request/Response  
Content Download  
Explanation  
0.15 ms  
DURATION  
21 µs  
DURATION  
0.17 ms  
DURATION  
TIME  
During development, you can use [the Server Timing API](#) to add insights into the server-side timing of this request.

## 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. The main content area displays the CSS code for 'styles.css'. At the bottom of the browser window, there is a developer tools interface with tabs for Elements, Source, Console, Network, Performance, Memory, Application, Security, Lighthouse, AdBlock, and Redux. The Network tab is selected, showing a table of requests. The table has columns for Name, Status, Type, Initiator, Size, Time, and Waterfall. There is one entry: 'styles.css' with a status of 304, type document, initiator Other, size 266 B, time 11 ms, and a small waterfall chart.

```
/*
*****
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, 63%, 48%);

    /* primary/main color */
    --cl-primary-5: hsl(205, 78%, 60%);

    /* light shades of primary color */
    --cl-primary-6: hsl(205, 89%, 70%);
    --cl-primary-7: hsl(205, 90%, 76%);

    --cl-primary-8: hsl(205, 86%, 81%);

    --cl-primary-9: hsl(205, 80%, 86%);

    --cl-primary-10: hsl(205, 100%, 96%);

    /* darkest grey - used for headings */
    --cl-grey-1: hsl(211, 3%, 97%);

    --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%, 58%);

    --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%, 98%);

    --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;
}

/*
*****
```

## 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 under 'OPEN EDITORS'. The main editor window contains the code for 'app.js'. The terminal window at the bottom shows the command-line output of running the application.

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

```
PS C:\Users\saikar\Documents\Node JS Tutorial YouTube\02-express-tutorial> npm start
> 02-express-tutorial@1.0.0 start C:\Users\saikar\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]  - C:\Users\saikar\Documents\Node JS Tutorial YouTube\02-express-tutorial\app.js
[nodemon] starting `node app.js`
Server is listening on port 5000....
```

localhost:5000

The screenshot shows a browser window displaying a website titled 'Coding Addict'. Below the page content, a detailed network analysis tool is overlaid, showing the list of resources loaded by the page and their performance metrics.

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	
cssfamily=Open+Sans@Roboto:400,700&display=swap	200	stylesheet	styles.css+Infinity		(memory cache)	0 ms
fa-brands-400.woff2	200	font	allmin.css		(memory cache)	0 ms
fa-solid-900.woff2	200	font	allmin.css		(memory cache)	0 ms
mem5YeGz126MjZp8A-UwBx2vNxKBbObj2Ov2yOOS4dVWUjsz2084gaVlwoF2	200	font	cssfamily=Open+Sans@Roboto:400,700&display=swap		(memory cache)	0 ms

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)

The slide has a light gray background. At the top left, it says "Express.js". On the right is a green hexagonal logo with the letters "JS" inside. Below the title, the words "API VS SSR" are centered in large, bold, black capital letters. Underneath, there are two columns of three items each, each preceded by a small square checkbox:

<input type="checkbox"/> <b>API - JSON</b>	<input type="checkbox"/> <b>SSR - TEMPLATE</b>
<input type="checkbox"/> <b>SEND DATA</b>	<input type="checkbox"/> <b>SEND TEMPLATE</b>
<input type="checkbox"/> <b>RES.JSON()</b>	<input type="checkbox"/> <b>RES.RENDER()</b>

In the bottom right corner of the slide, there is a small button with a fire icon and the word "SUBSCRIBE".

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 response. The response is a list of two objects, each representing a job entry:

```
[{"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": "recI6mJNfuObonls", "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"}]
```

Below the JSON response, the browser's developer tools Network tab is open, showing a request to "https://course-api.com/react-tabs-project". The request details are as follows:

- Request URL: https://course-api.com/react-tabs-project
- Request Method: GET
- Status Code: 200 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:

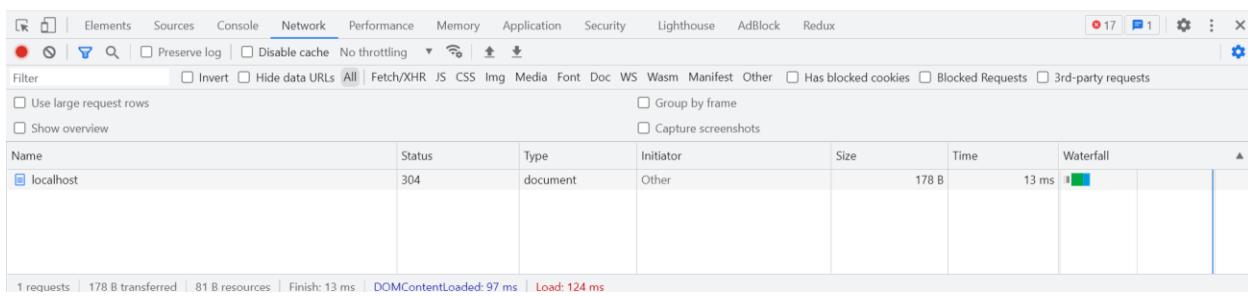
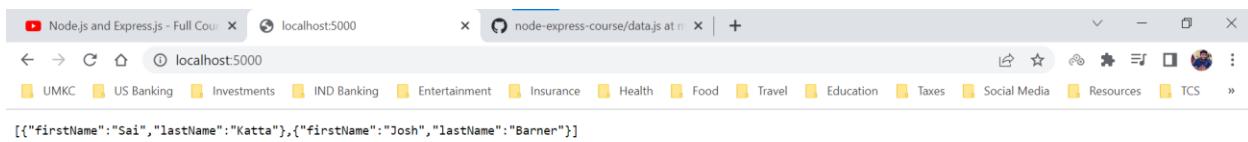
```
02-express-tutorial > app.js > app.get("/") callback
1 const express = require("express");
2 const app = express();
3
4 app.get("/", (req, res) => {
5   res.json([
6     { firstName: "Sai", lastName: "Katta" },
7     { firstName: "Josh", lastName: "Barner" },
8   ]);
9 });
10
11 app.listen(5000,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

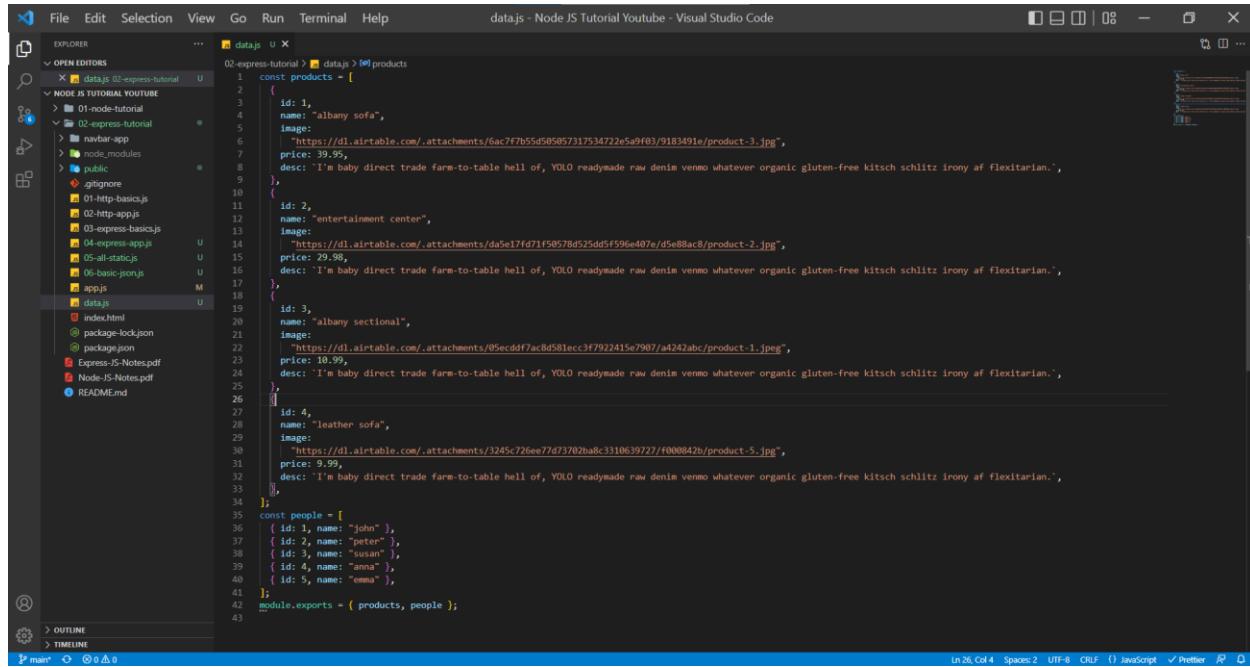
[nodemon] 2.0.18
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node app.js
Server is listening on Port 5000....
```

localhost:5000



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

data.js

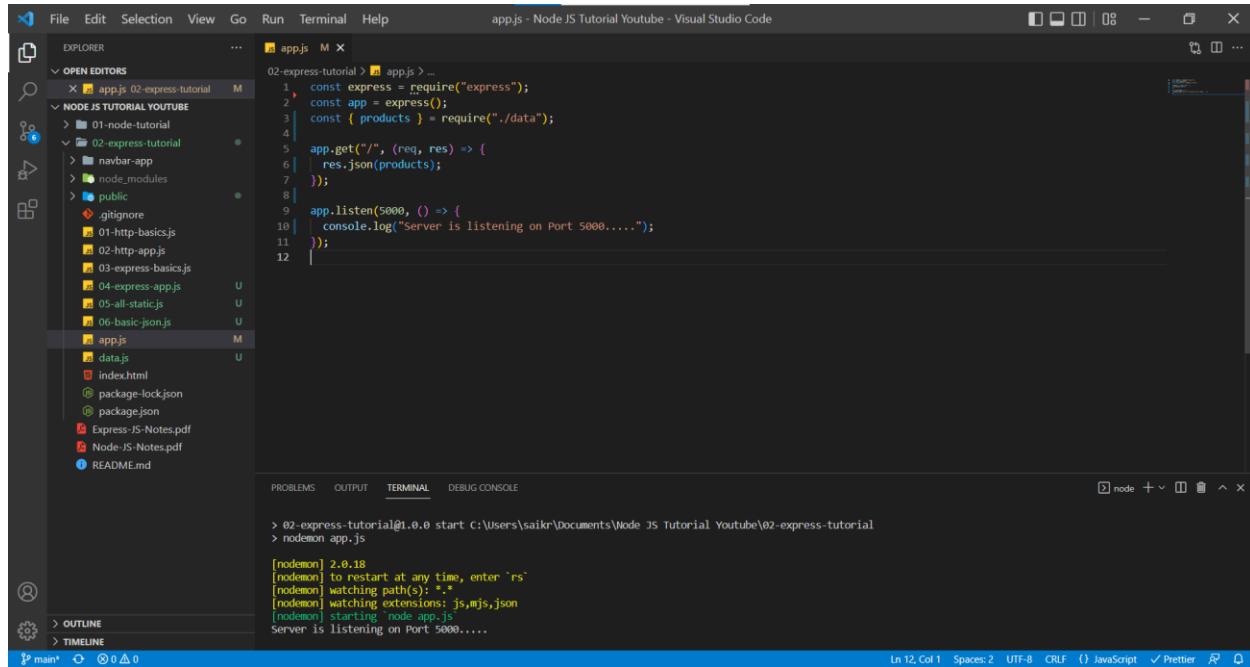


```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS data.js U
02-express-tutorial > data.js > products
1 const products = [
2   {
3     id: 1,
4     name: "albany sofa",
5     image:
6       "https://d1.airtable.com/.attachments/6ac7f755d50557317534722e5e9f03/9183491e/product-3.jpg",
7     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 };

```

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
OPEN EDITORS app.js M X
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

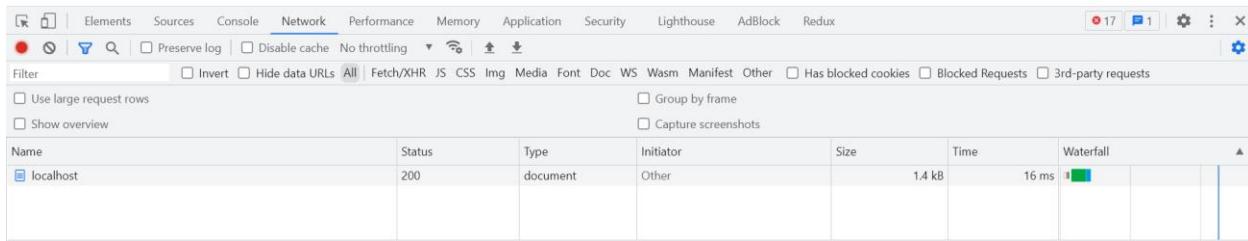
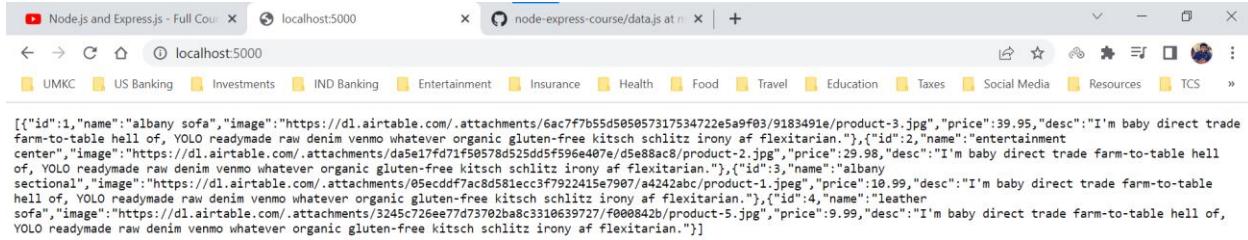
```

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.

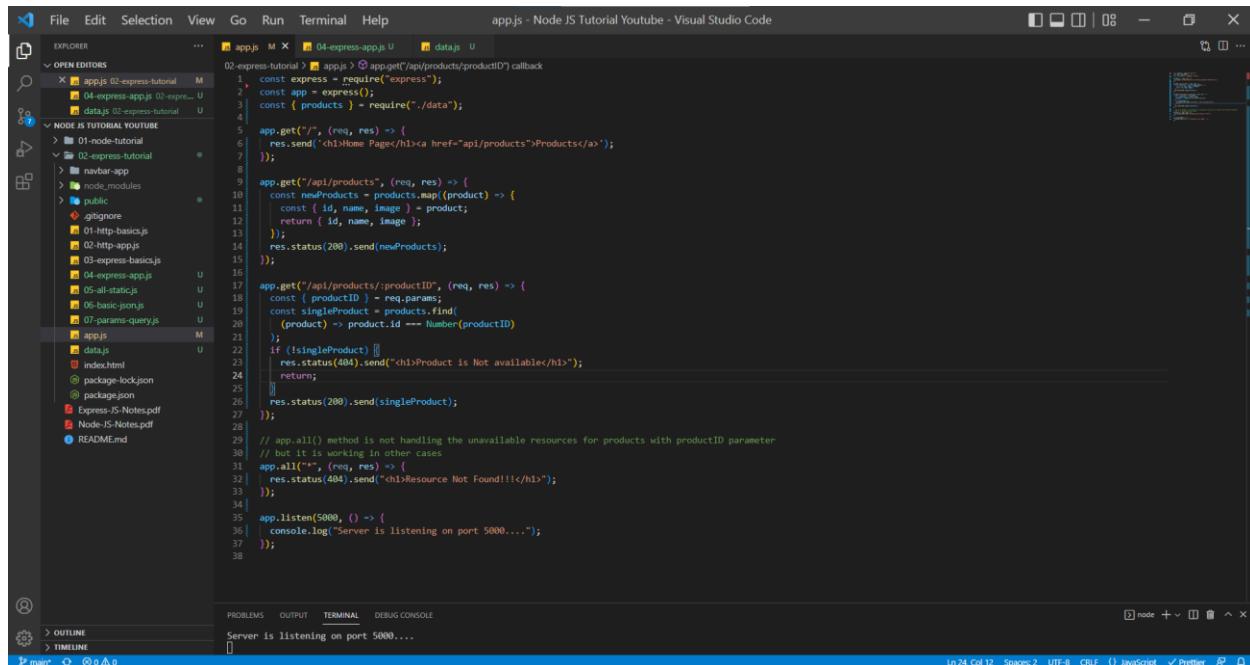
Ex: localhost:5000/api/products/hgjh

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");

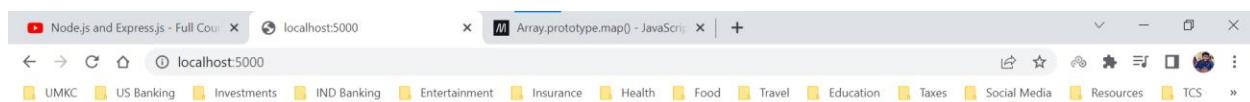
app.get("/api/products/:productID", (req, res) => {
  const product = products.find((product) => product.id === Number(req.params.productID));
  if (!product) {
    res.status(404).send("<h1>Product is Not available</h1>");
    return;
  }
  res.status(200).send(product);
});

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);
});

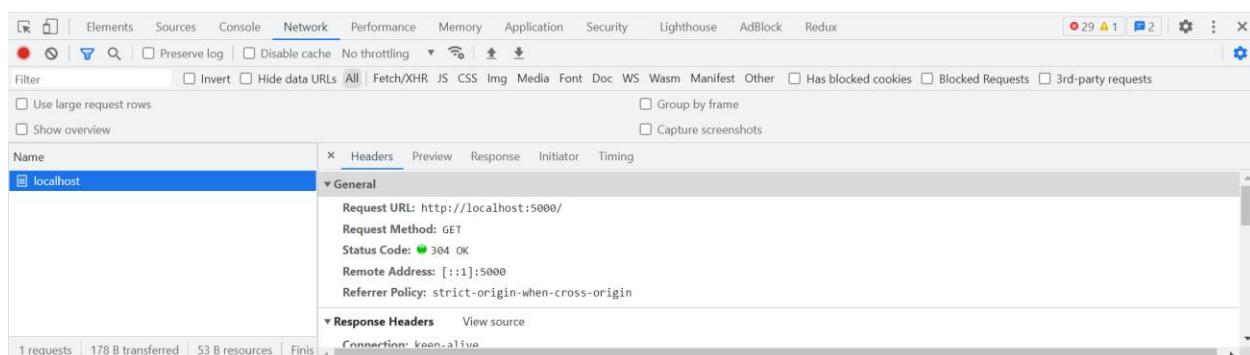
app.get("/", (req, res) => {
  const newProducts = products.map((product) => {
    const { id, name, image } = product;
    return { id, name, image };
  });
  res.send(`<h1>Home Page</h1><a href="/api/products">Products</a>`);
});

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

localhost:5000 – Home Page

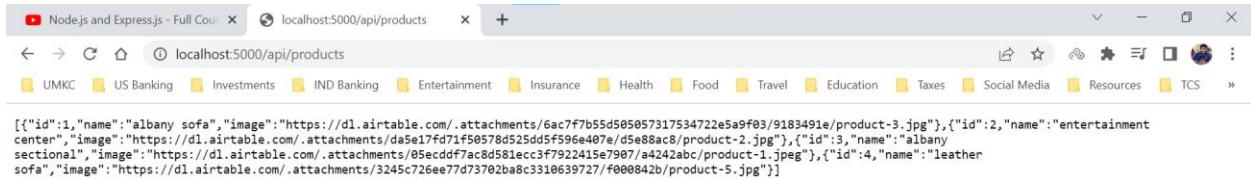


## Network Tab in DevTools

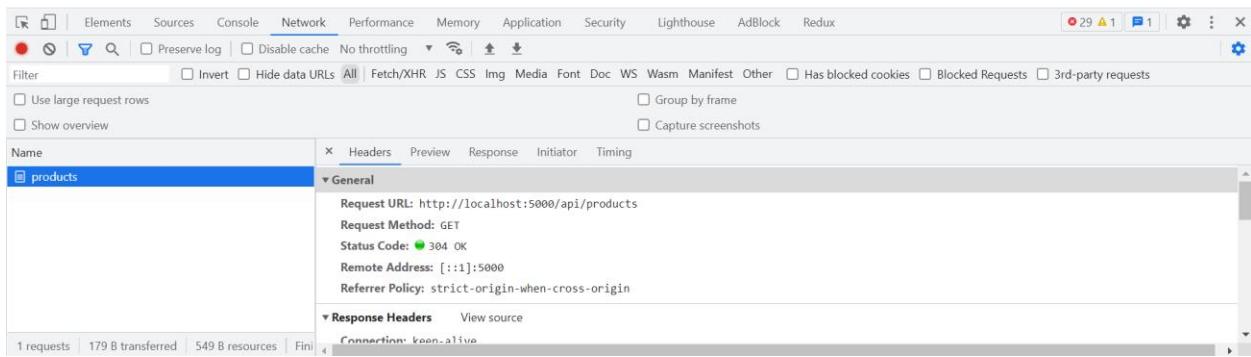


Name	Headers	Preview	Response	Initiator	Timing
localhost	Request URL: http://localhost:5000/ Request Method: GET Status Code: 304 OK Remote Address: [::1]:5000 Referrer Policy: strict-origin-when-cross-origin				
	Response Headers	View source	Connection: keep-alive		

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

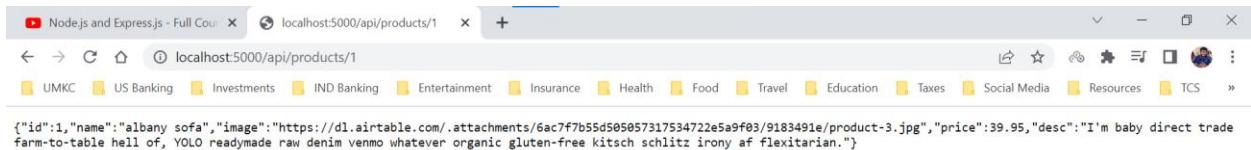


```
[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b55d505057317534722e5a9f03/9183491e/product-3.jpg"}, {"id":2,"name":"entertainment sectional","image":"https://dl.airtable.com/.attachments/dae17fd71f50578d525dd5f596e407e/d5e88ac8/product-2.jpg"}, {"id":3,"name":"albany 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"}]
```

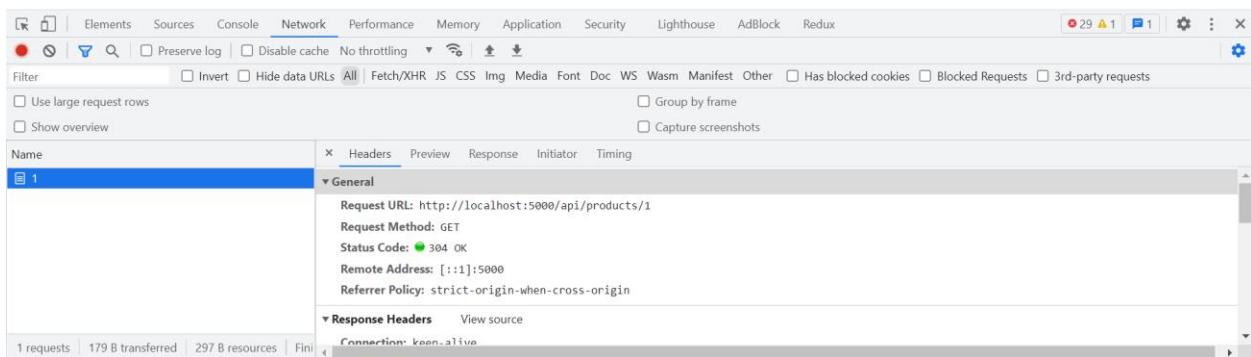


Request URL: http://localhost:5000/api/products  
Request Method: GET  
Status Code: 200 OK  
Remote Address: [::1]:5000  
Referrer Policy: strict-origin-when-cross-origin

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



```
{"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."}
```



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!!!

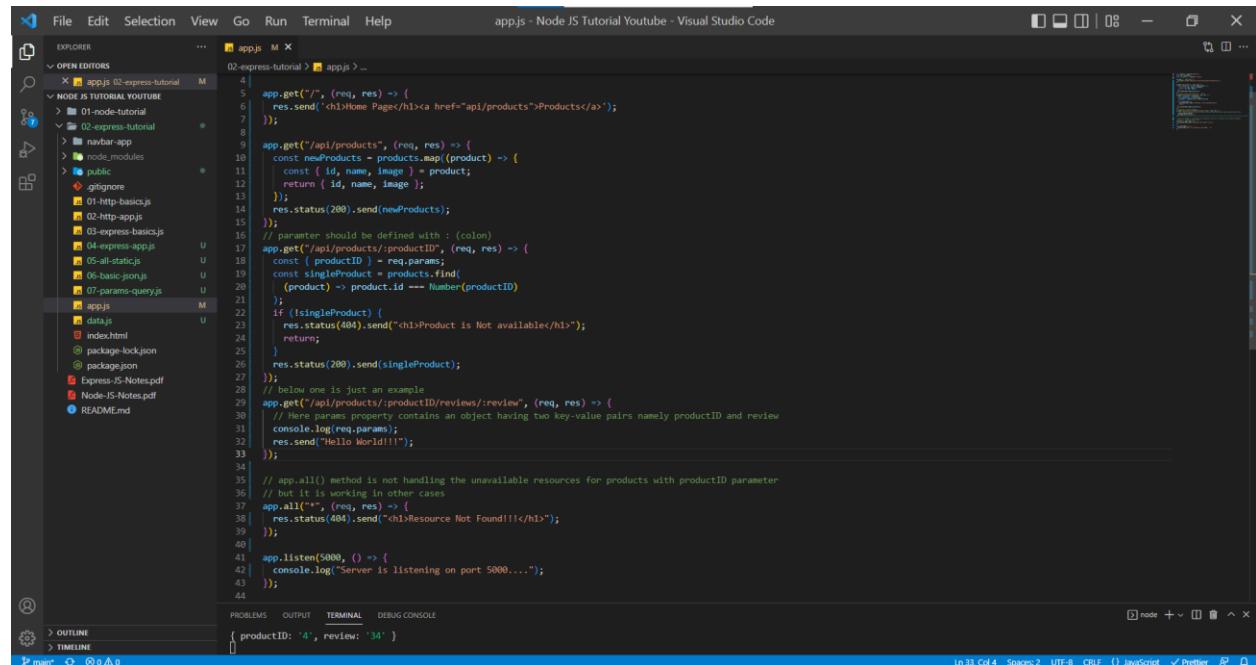
A screenshot of the Network tab in Chrome DevTools. A request for 'qwerty' is selected. The General section shows:

- 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.js
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 M
  datajs U
  index.html
  package-lock.json
  package.json
  Express-JS-Notes.pdf
  Node-JS-Notes.pdf
  README.md

app.js
express-tutorial > app.js > ...
4
5  app.get("/", (req, res) => {
6    res.send(`<h1>Home Page</h1><a href="/api/products">Products</a>`);
7  });
8
9  app.get("/api/products", (req, res) => {
10   const newProducts = products.map(product) => {
11     const { id, name, image } = product;
12     return { id, name, image };
13   };
14   res.status(200).send(newProducts);
15 });
16 // parameter should be defined with : (colon)
17 app.get("/api/products/:productId", (req, res) => {
18   const { productId } = req.params;
19   const singleProduct = products.find(
20     (product) => product.id === Number(productId)
21   );
22   if (!singleProduct)
23     res.status(404).send(`<h1>Product is Not available</h1>`);
24   return;
25 }
26 res.status(200).send(singleProduct);
27 });
28 // below one is just an example
29 app.get("/api/products/:productId/reviews/:review", (req, res) => {
30   // Here params property contains an object having two key-value pairs namely productId and review
31   console.log(req.params);
32   res.send("Hello World!!!");
33 });
34
35 // app.all() method is not handling the unavailable resources for products with productId parameter
36 // but it is working in other cases
37 app.all("*", (req, res) => {
38   res.status(404).send("<h1>Resource Not Found!!!</h1>");
39 });
40
41 app.listen(5000, () => {
42   console.log("Server is listening on port 5000....");
43 });
44
```

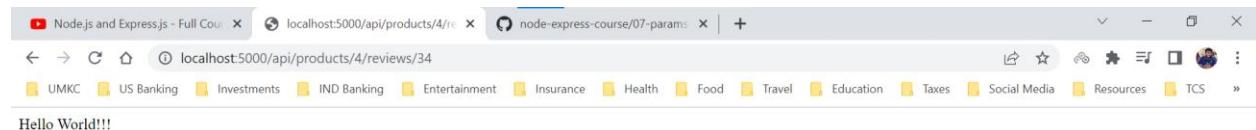
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

{ productId: '4', review: '34' }

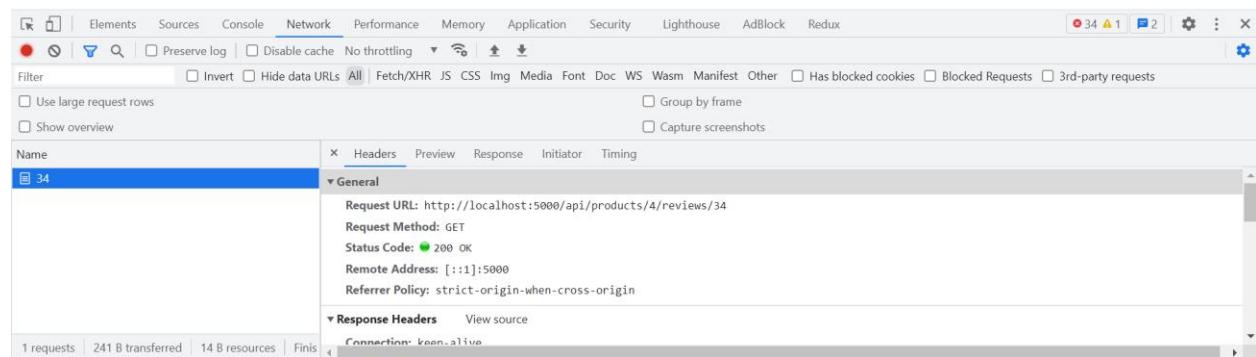
In 33 Col 4 Spaces 2 UTF-8 CRLF ↻ JavaScript ✓ prettier ↻

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, which shows a single request for '34' with a status of 404, type document, and initiator Other. The size is 265 B and the time is 9 ms. The Waterfall section is empty.

## 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>

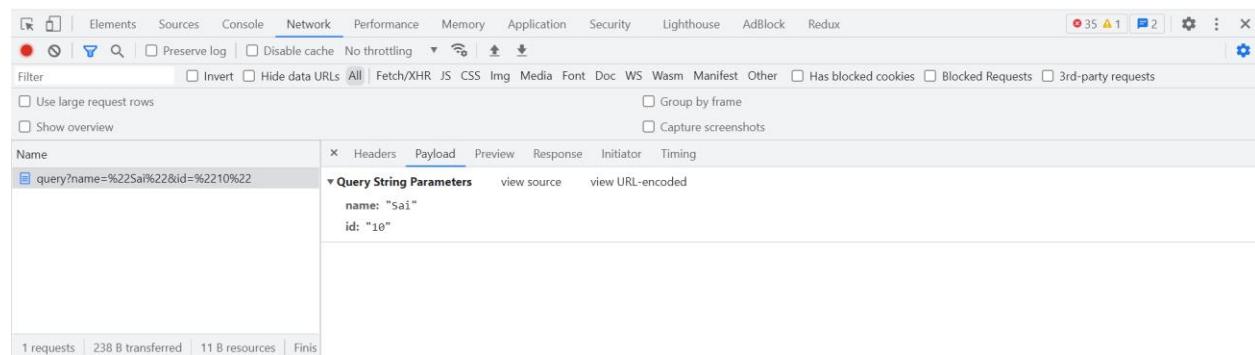
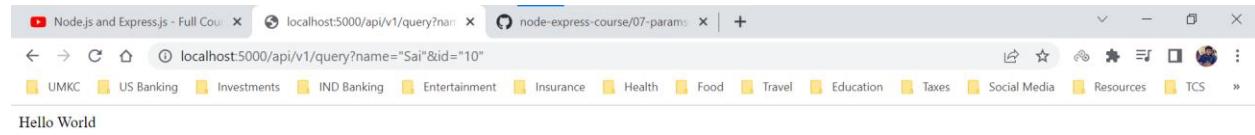
Whatever is after the question mark is not technically part of the URL meaning it is just a way for us to send that data to the server and then server decides what to do with this data.

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

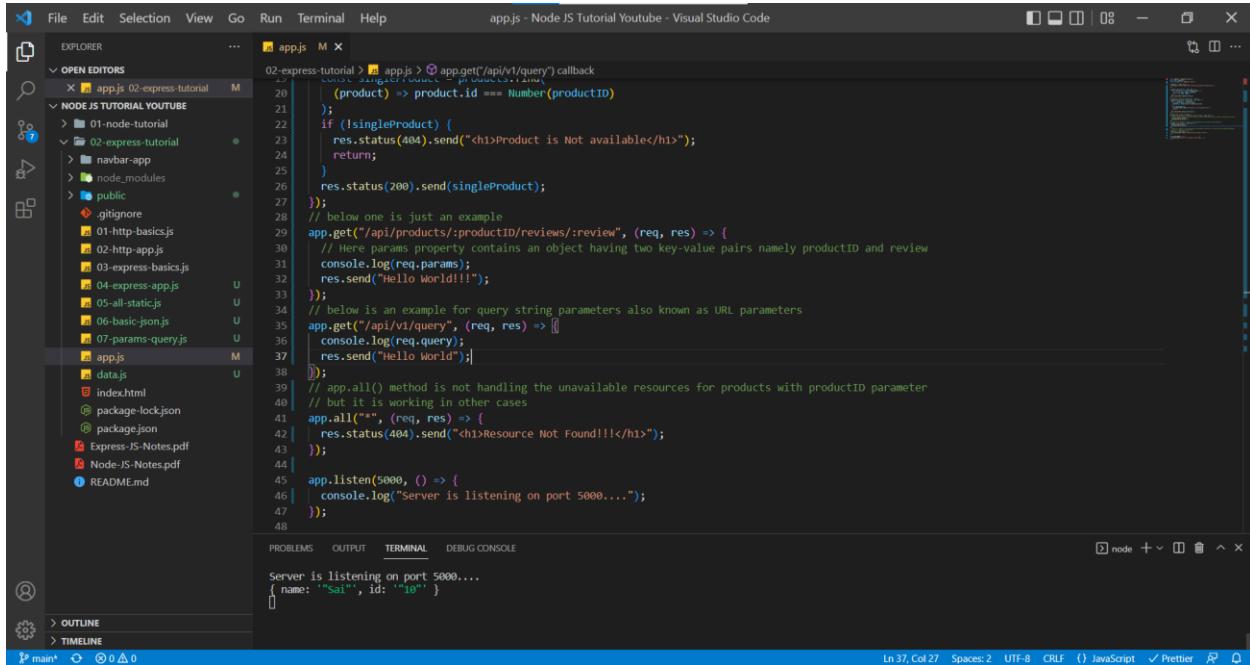
We can add how many query parameters, but they need to start after question mark (?) and all of them need to be combined with ampersand (&), based on the query parameters we can develop the functionality and provide the response.

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.

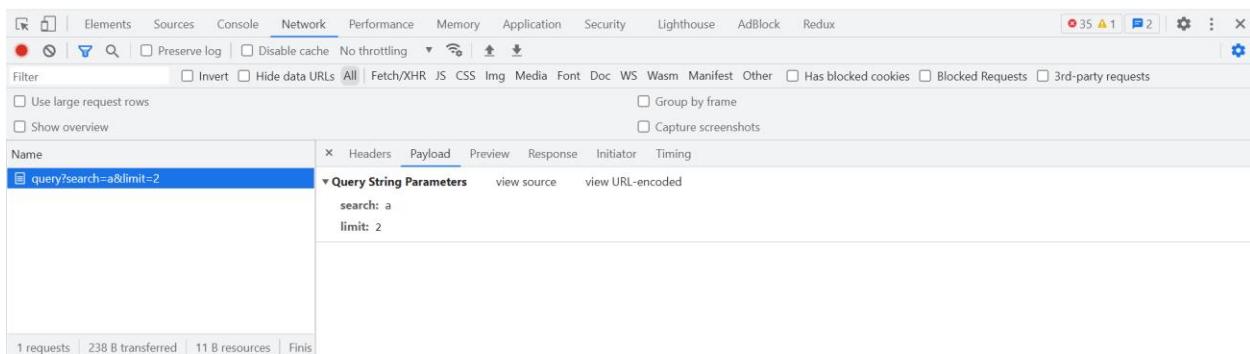


```

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 > app.get('/api/v1/query') callback
  const singleProduct = products.find((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....");
});
48
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 sent via 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   const { search, limit } = req.query;
 38   console.log(req.query);
 39   res.send("Hello World");
 40 ]);
 41 // app.all() method is not handling the unavailable resources for products with productId parameter
 42 // but it is working in other cases
 43 app.all("*", (req, res) => {
 44   res.status(404).send("<h1>Resource Not Found!!!</h1>");
 45 });
 46 app.listen(5000, () => {
 47   console.log("Server is listening on port 5000....");
 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

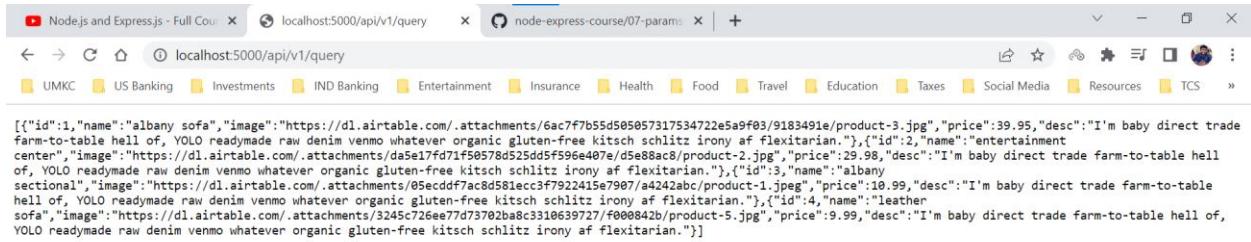
The screenshot shows the Visual Studio Code interface with the following details:

- 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 sent via 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 });

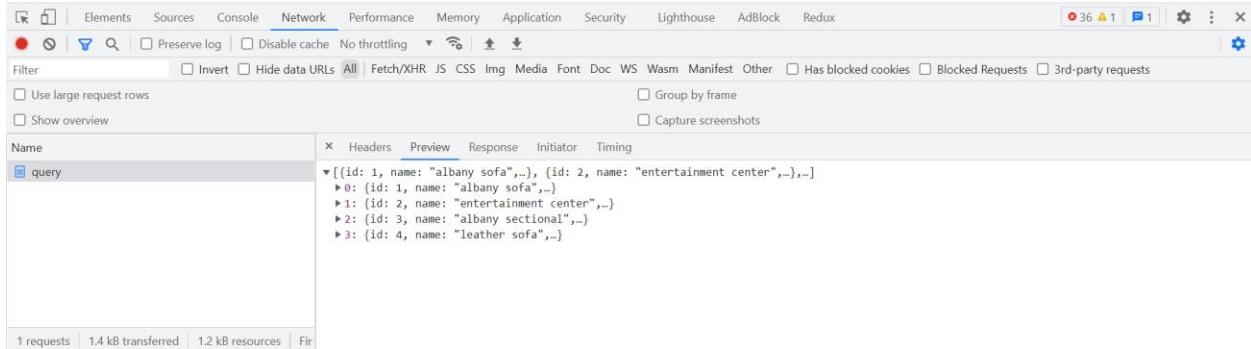
[nodemon] starting 'node app.js'
Server is listening on port 5000...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on port 5000...
```

## 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 the JSON response from the API endpoint. The response is a single array containing four objects, each representing a sofa with its ID, name, image URL, price, and description.

```
[{"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/05ecddf7ac8d581ecc3f7922415e7907/44242abc/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."}]
```



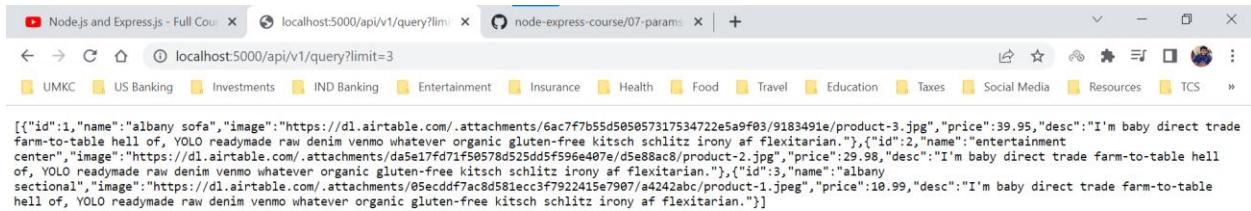
The screenshot shows the Network tab in Chrome DevTools. A single request is listed under the "query" entry. The Headers section shows the following:

```
Name: query
```

The Response section shows the same JSON array as the previous screenshot.

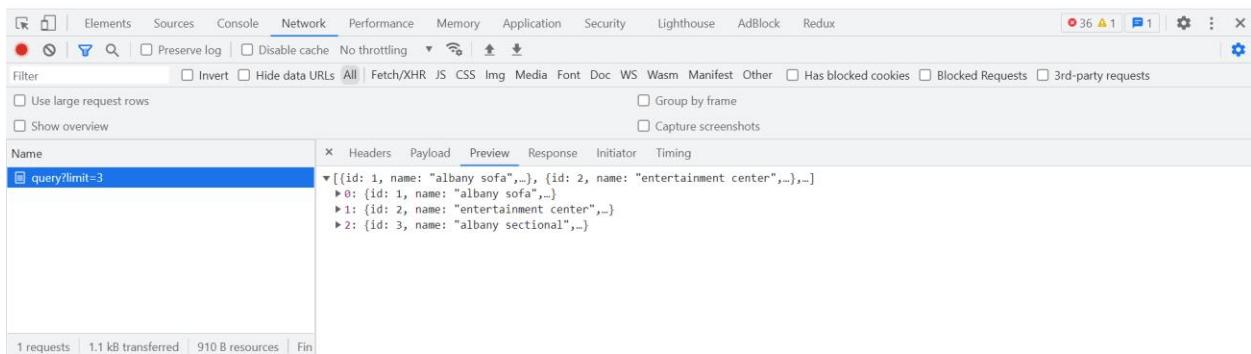
```
1 requests | 1.4 kB transferred | 1.2 kB resources | Fin
```

## 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 the JSON response from the API endpoint. The response is a single array containing only the first three objects from the original list, as specified by the limit query parameter.

```
[{"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/05ecddf7ac8d581ecc3f7922415e7907/44242abc/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 is listed under the "query?limit=3" entry. The Headers section shows the following:

```
Name: query?limit=3
```

The Response section shows the same JSON array as the previous screenshot.

```
1 requests | 1.1 kB transferred | 910 B resources | Fin
```

<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 Network tab in the Chrome DevTools. It shows a single request labeled "query?search=a". The "Payload" section of the request pane shows the search query, and the "Preview" section of the response pane shows the JSON array returned by the API.

<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, as specified by the limit parameter.

```
[{"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 Network tab in the Chrome DevTools. It shows a single request labeled "query?search=a&limit=1". The "Payload" section of the request pane shows the search and limit query, and the "Preview" section of the response pane shows the JSON array returned by the API, which contains only the first item from the original response.

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

The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "localhost:5000/api/v1/query?search=jhgjhg", and "node-express-course/07-params". The middle tab displays the URL "localhost:5000/api/v1/query?search=jhgjhg". Below the tabs is a navigation bar with various categories like UMKC, US Banking, Investments, IND Banking, Entertainment, Insurance, Health, Food, Travel, Education, Taxes, Social Media, Resources, and TCS. The main content area is empty, showing only a small icon and the text "[ ]".

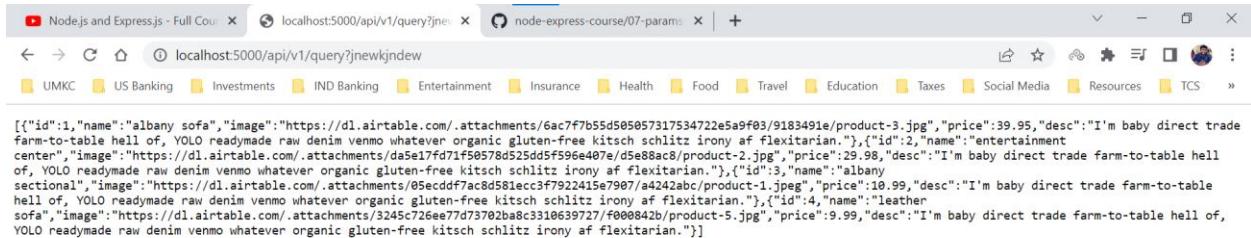
Below the browser is a screenshot of the Network tab in the Chrome DevTools. The request "query?search=jhgjhg" is listed under "Name". The "Payload" section shows an empty array "[ ]" with the message "No properties". At the bottom of the DevTools interface, there are buttons for "1 requests", "235 B transferred", "2 B resources", and "Finish".

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

The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "localhost:5000/api/v1/query?limit=abc", and "node-express-course/07-params". The middle tab displays the URL "localhost:5000/api/v1/query?limit=abc". Below the tabs is a navigation bar with categories like UMKC, US Banking, Investments, IND Banking, Entertainment, Insurance, Health, Food, Travel, Education, Taxes, Social Media, Resources, and TCS. The main content area is empty, showing only a small icon and the text "[ ]".

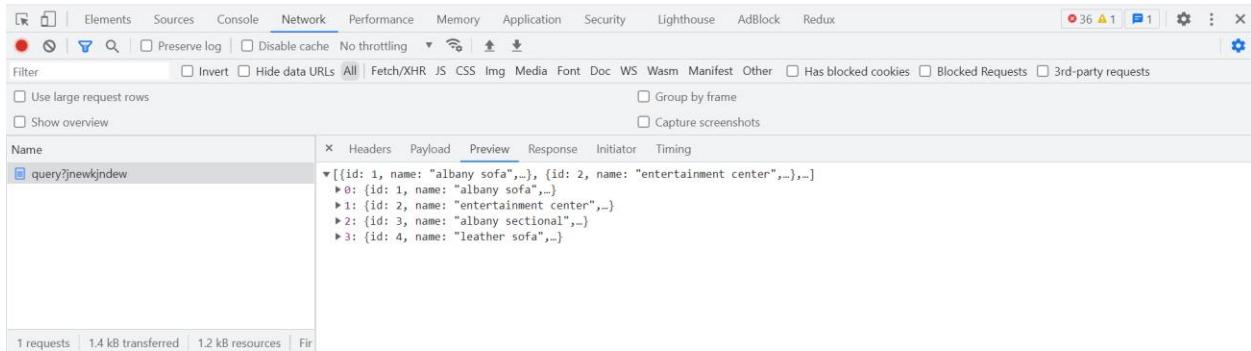
Below the browser is a screenshot of the Network tab in the Chrome DevTools. The request "query?limit=abc" is listed under "Name". The "Payload" section shows an empty array "[ ]" with the message "No properties". At the bottom of the DevTools interface, there are buttons for "1 requests", "235 B transferred", "2 B resources", and "Finish".

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



The screenshot shows a browser window with three tabs: "Node.js and Express.js - Full Course", "localhost:5000/api/v1/query?jnewkjndew", and "node-express-course/07-params". The active tab displays a JSON array with four objects:

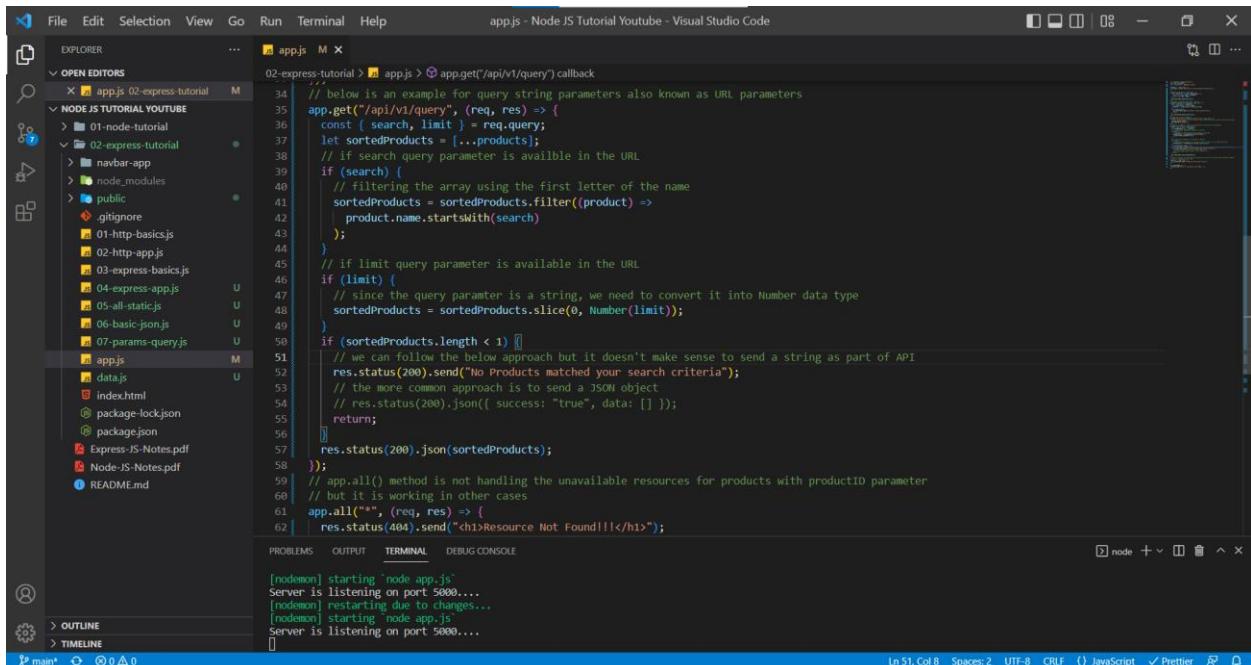
```
[{"id":1,"name":"albany sofa","image":"https://dl.airtable.com/.attachments/6ac7f7b5d50505731753472e5a9f03/9183491e/product-3.jpg","price":39.95,"desc":"I'm baby direct trade farm-to-table hell of, YOLO readymade raw denim venmo whatever organic gluten-free kitsch schlitz irony af flexitarian."}, {"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/05ecdddf7ac8d581ecc3f7922415e7907/44242abc/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."}]
```



The screenshot shows the Network tab in the Chrome DevTools. A single request is listed: "query?jnewkjndew". The response payload shows the same JSON array as the previous screenshot.

```
[{"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/05ecdddf7ac8d581ecc3f7922415e7907/44242abc/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.



The screenshot shows the Visual Studio Code interface with the "app.js" file open. The code handles query parameters and includes a check for an empty array:

```
if (sortedProducts.length < 1) {
    res.status(200).send("No Products matched your search criteria");
}
```

<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
  ...
  // below is an example for query string parameters also known as URL parameters
  app.get('/api/v1/query', (req, res) => {
    const { search, limit } = req.query;
    let sortedProducts = [...products];
    // 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 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("<h1>Resource Not Found!!!</h1>");
  })
}

[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...
```

The terminal at the bottom shows the Node.js server starting and 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' (active), and 'node-express-course/07-params' (closed). The active tab displays the JSON response: {"success": "true", "data": []}.

The screenshot shows the Network tab in Chrome DevTools. A single request is listed: 'query?search=b'. The 'Preview' section shows the JSON response: {"success": "true", "data": []}.

## 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

The screenshot shows the Visual Studio Code interface with the file 'app.js' open. The code implements a search functionality for products. It checks if a 'search' query parameter is present in the URL. If so, it filters the products array based on the first letter of the name. It also handles a 'limit' parameter by slicing the array. If no products are found, it returns a JSON object with 'success: true' and an empty 'data' array. Otherwise, it returns the sorted products. The code uses Node.js's built-in 'http' module and the 'express' framework.

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

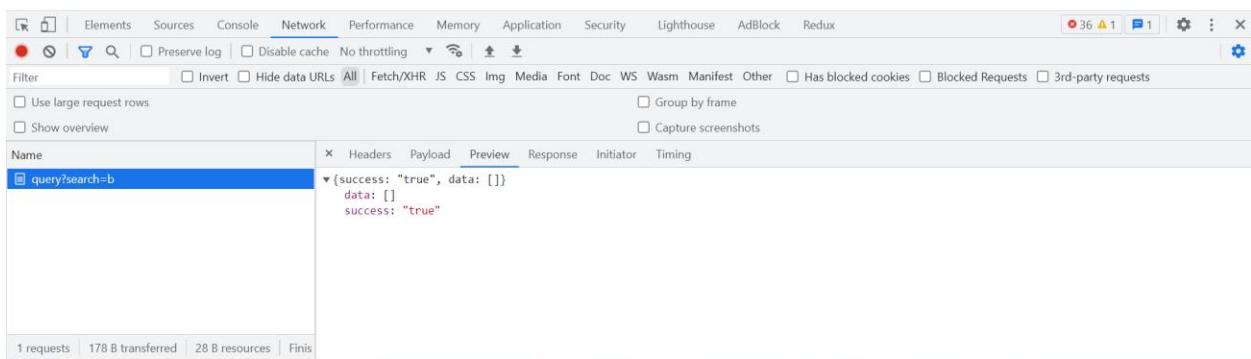
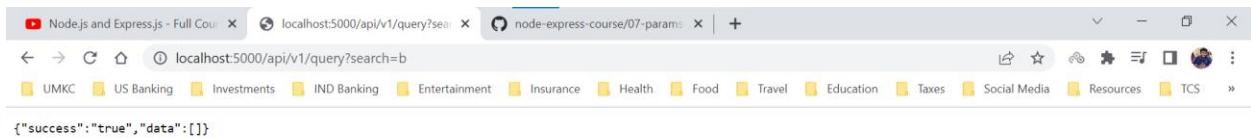
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ln 54, Col 57  Spaces: 2  UTF-8  CHRF  { JavaScript  ✓ Prettier  R  D

node + v  □  ^  ×

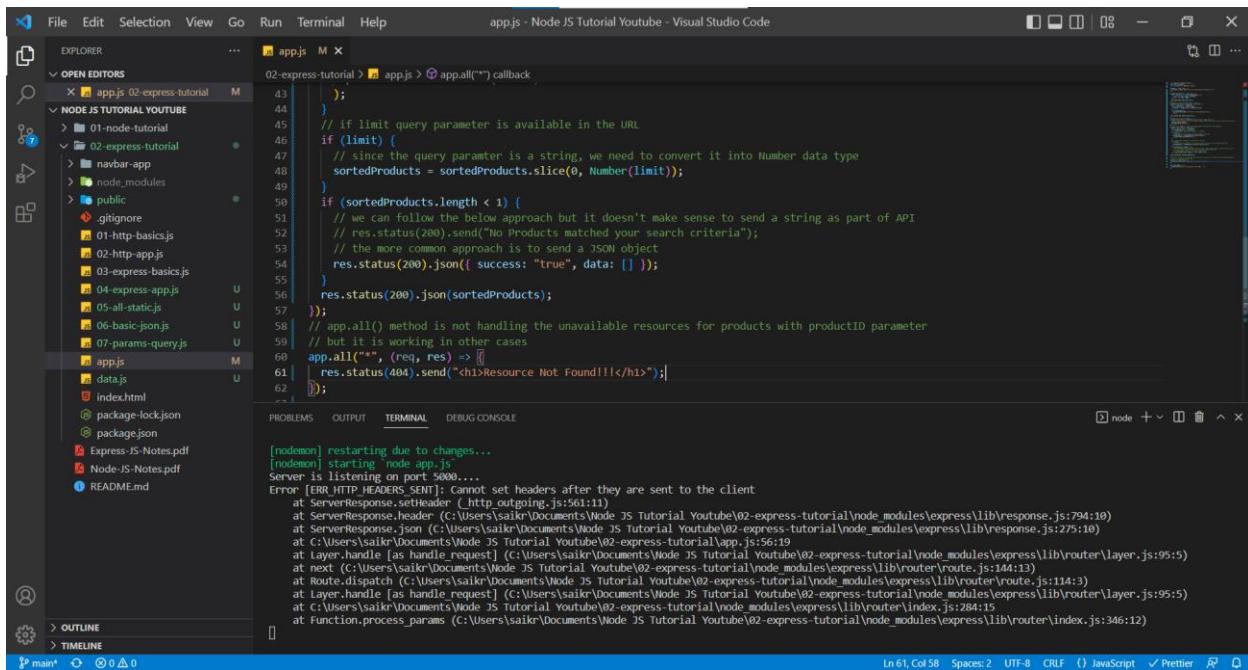
[nodemon] starting 'node app.js'
Server is listening on port 5000...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Server is listening on port 5000...
[]

HTTP://localhost:5000/api/v1/query?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.



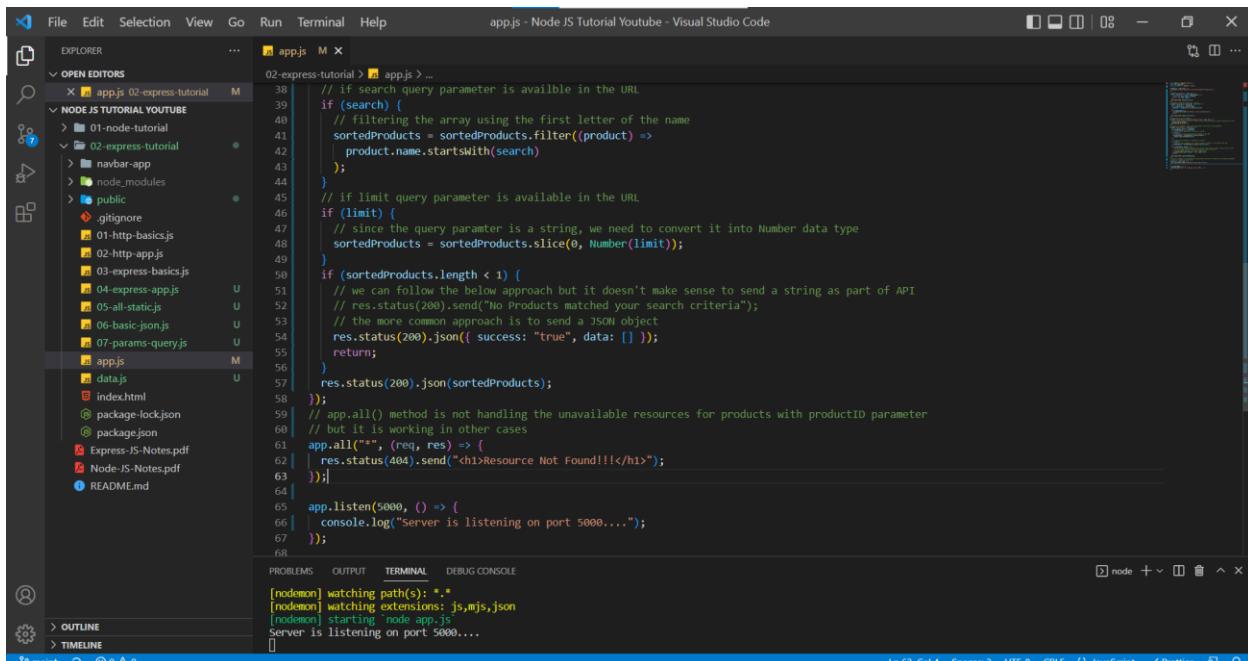
The screenshot shows the Visual Studio Code interface with the file `app.js` open. The code contains several errors, notably at lines 40, 41, and 42, which are highlighted in red. The terminal window below shows the following error message:

```
[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 (http_outgoing.js:561:11)
at ServerResponse.header (<file>:275:10)
at ServerResponse.json (<file>:275:19)
at <file>:56:19
at Layer.handle [as handle_request] (<file>:144:13)
at next (<file>:144:13)
at Route.dispatch (<file>:144:13)
at Layer.handle [as handle_request] (<file>:144:13)
at Layer.handle [as handle_request] (<file>:144:13)
at Layer.dispatchRequest (<file>:144:13)
at Function.process_params (<file>:284:15)
at Function.next (<file>:284:15)
at Function.handle (<file>:284:15)
at Layer.handle [as handle_request] (<file>:284:15)
at Layer.handle [as handle_request] (<file>:284:15)
at Layer.dispatchRequest (<file>:284:15)
at Function.process_params (<file>:346:12)
```

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

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. The code has been modified to include a `return;` statement at line 55, which resolves the error. The terminal window below shows the server is now listening correctly on port 5000.

```
// 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 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....");
});
```

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

## Middleware - Setup