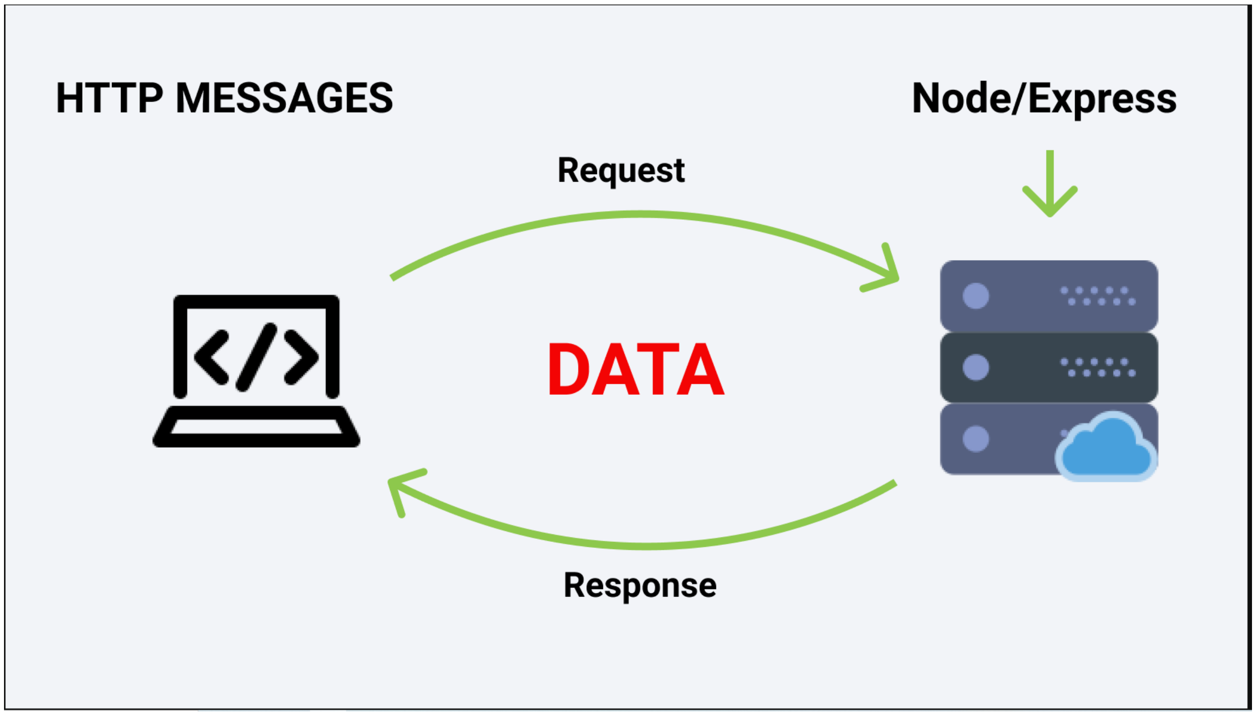
Express

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.

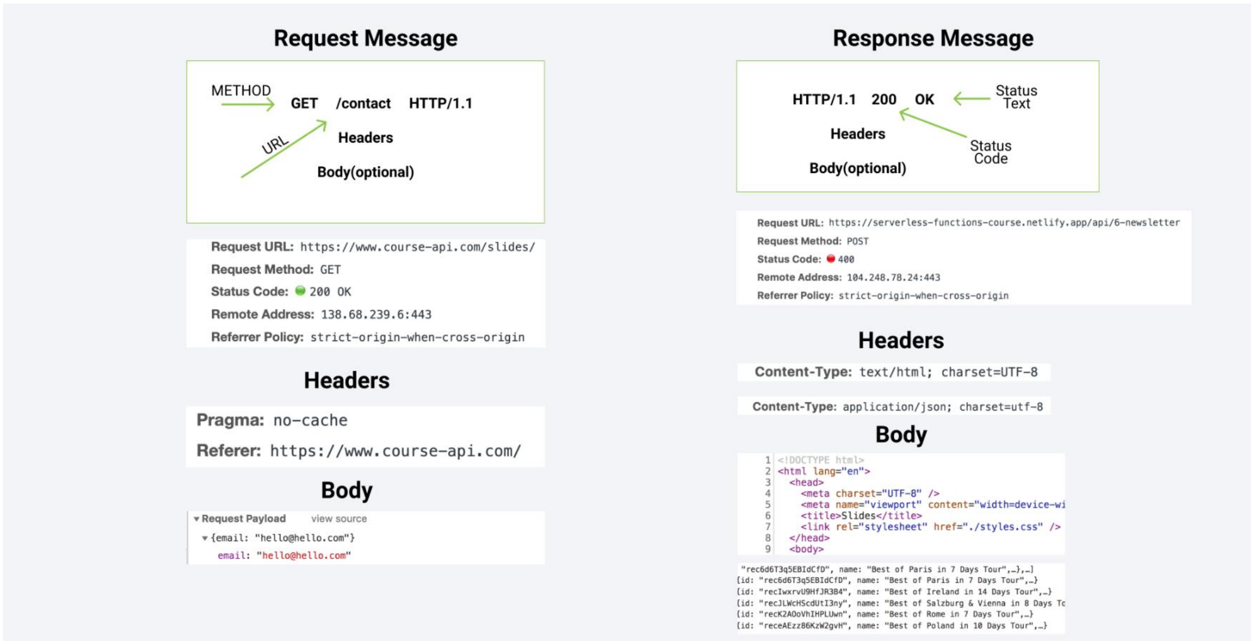


Suer sends a HTTP request message and then server sends an HTTP response message and that's how we exchange data on web.

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.



```
Request URL: https://www.course-api.com/slides/
Request Method: GET
Status Code: ● 200 OK
Remote Address: 138.68.239.6:443
Referrer Policy: strict-origin-when-cross-origin
```

```
Pragma: no-cache
Referer: https://www.course-api.com/
```

```
▾ Request Payload    view source
  ▾ {email: "hello@hello.com"}
     email: "hello@hello.com"
```

```
Request URL: https://serverless-functions-course.netlify.app/api/6-newsletter
Request Method: POST
Status Code: ● 400
Remote Address: 104.248.78.24:443
Referrer Policy: strict-origin-when-cross-origin
```

```
Content-Type: text/html; charset=UTF-8
```

```
Content-Type: application/json; charset=utf-8
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-wi
    <title>Slides</title>
    <link rel="stylesheet" href="./styles.css" />
  </head>
  <body>
```

```
 "rec6d6T3q5EBIdCfD", name: "Best of Paris in 7 Days Tour",…},…]
{id: "rec6d6T3q5EBIdCfD", name: "Best of Paris in 7 Days Tour",…}
{id: "recIwxrvU9HfJR3B4", name: "Best of Ireland in 14 Days Tour",…}
{id: "recJLWcHScdUtI3ny", name: "Best of Salzburg & Vienna in 8 Days To
{id: "recK2AOoVhIHPLUwn", name: "Best of Rome in 7 Days Tour",…}
{id: "receAEzz86KzW2gvH", name: "Best of Poland in 10 Days Tour",…}
```

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 in 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.

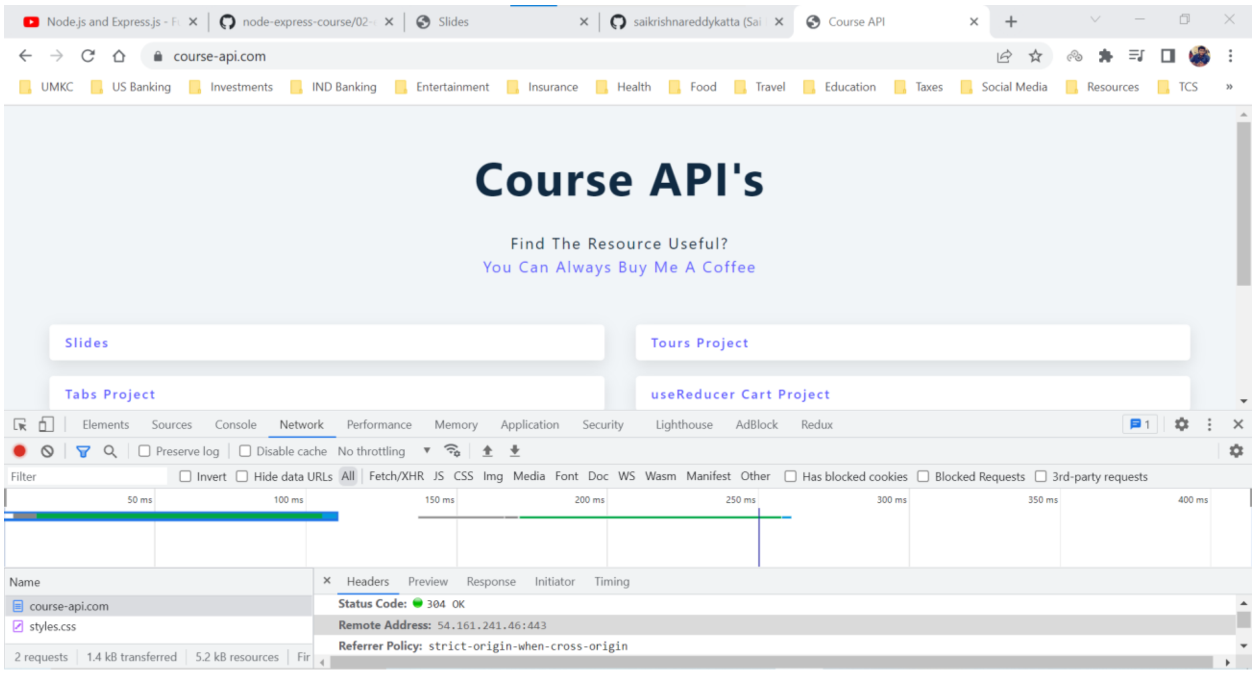
Port Number 21: Used for File Transfer Protocol (FTP) Command Control.

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.

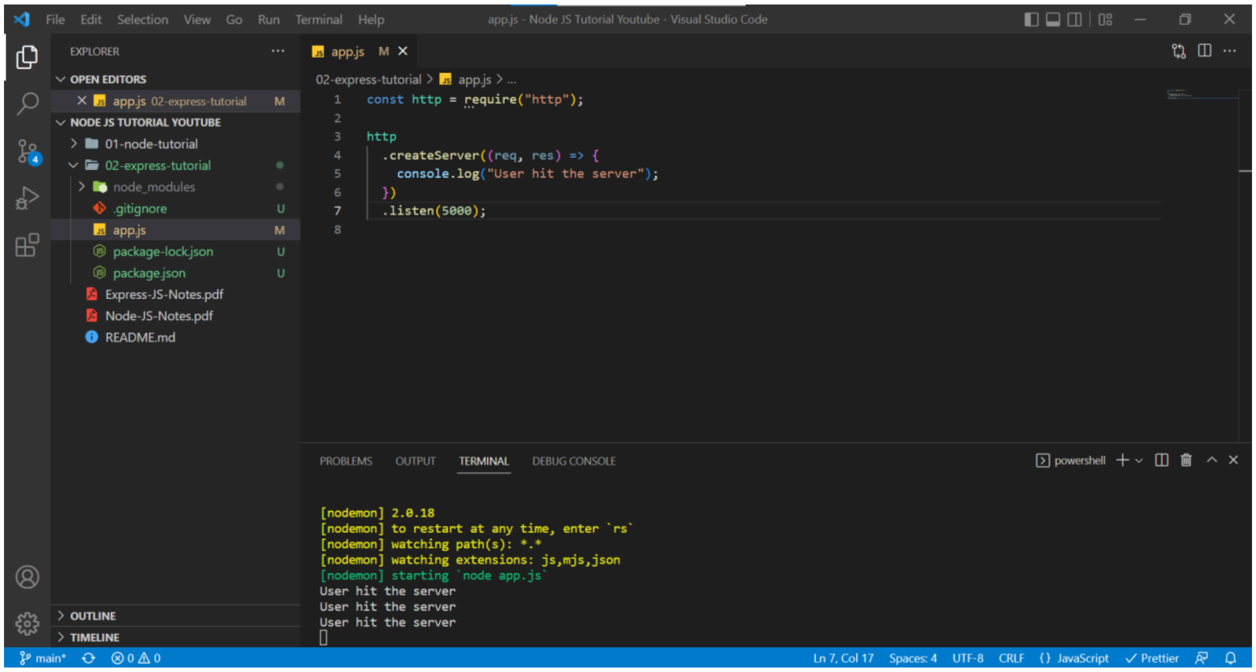

course-api.com
Course API's
Find The Resource Useful?
You Can Always Buy Me A Coffee
Slides
Tours Project
Tabs Project
useReducer Cart Project
Elements Sources Console Network Performance Memory Application Security Lighthouse AdBlock Redux
Name
course-api.com
styles.css
Headers Preview Response Initiator Timing
2 requests 1.4 kB transferred 5.2 kB resources

Status Code: 304 OK
Remote Address: 54.161.241.46:443
Referrer Policy: strict-origin-when-cross-origin

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.

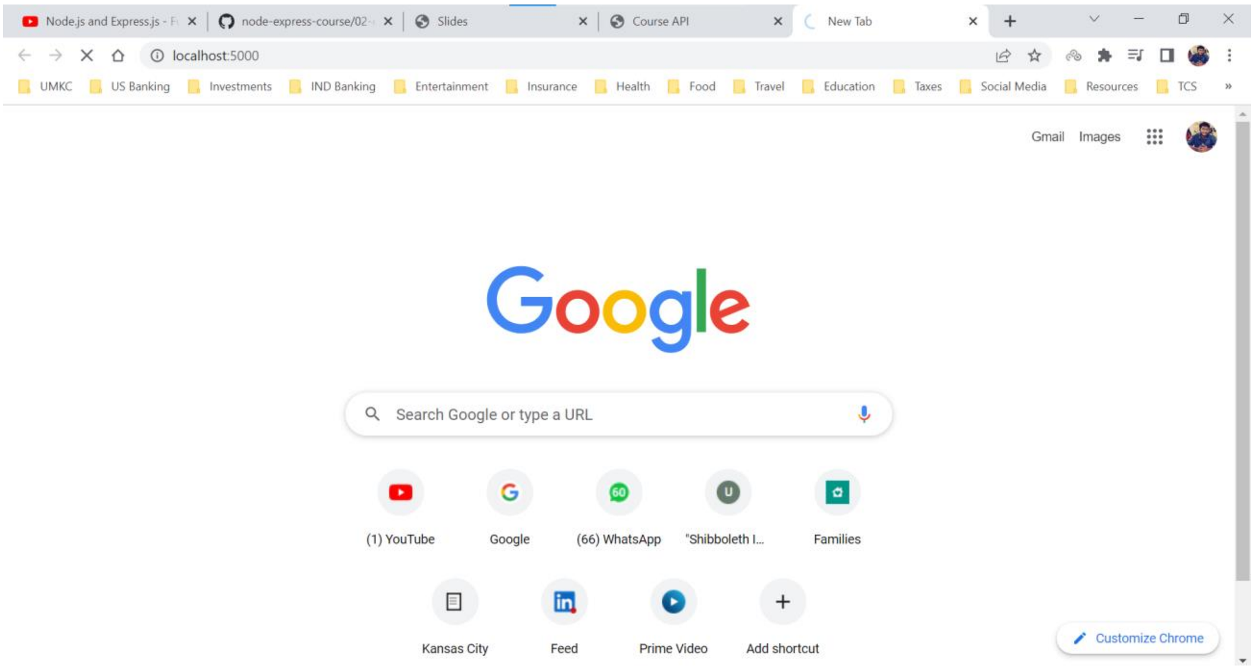


```
const http = require("http");

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

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

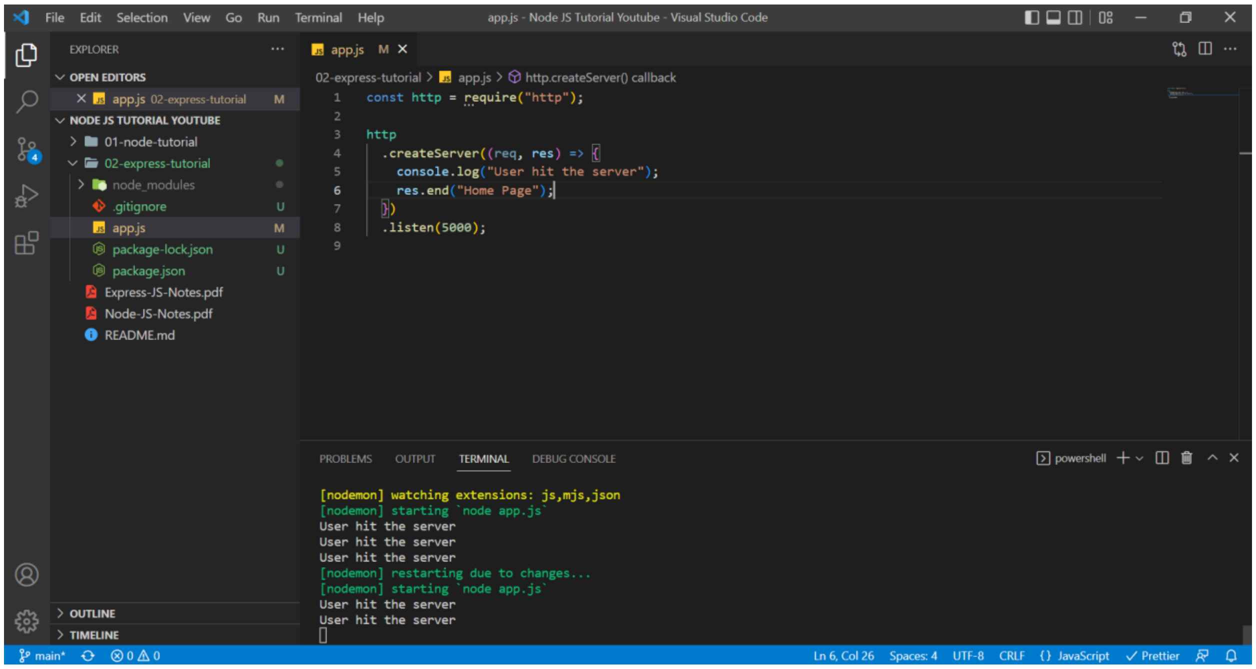
Server is waiting for the response, and it is still loading.



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

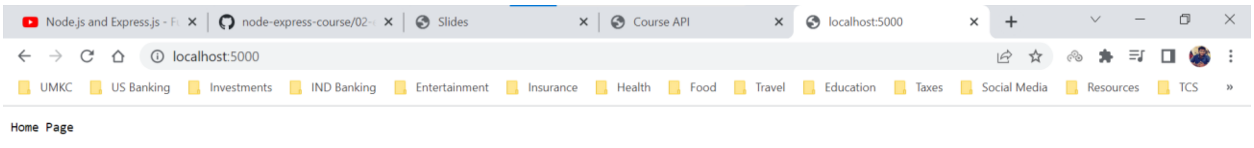


```
const http = require("http");

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

```
[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



```
Home Page
```

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.

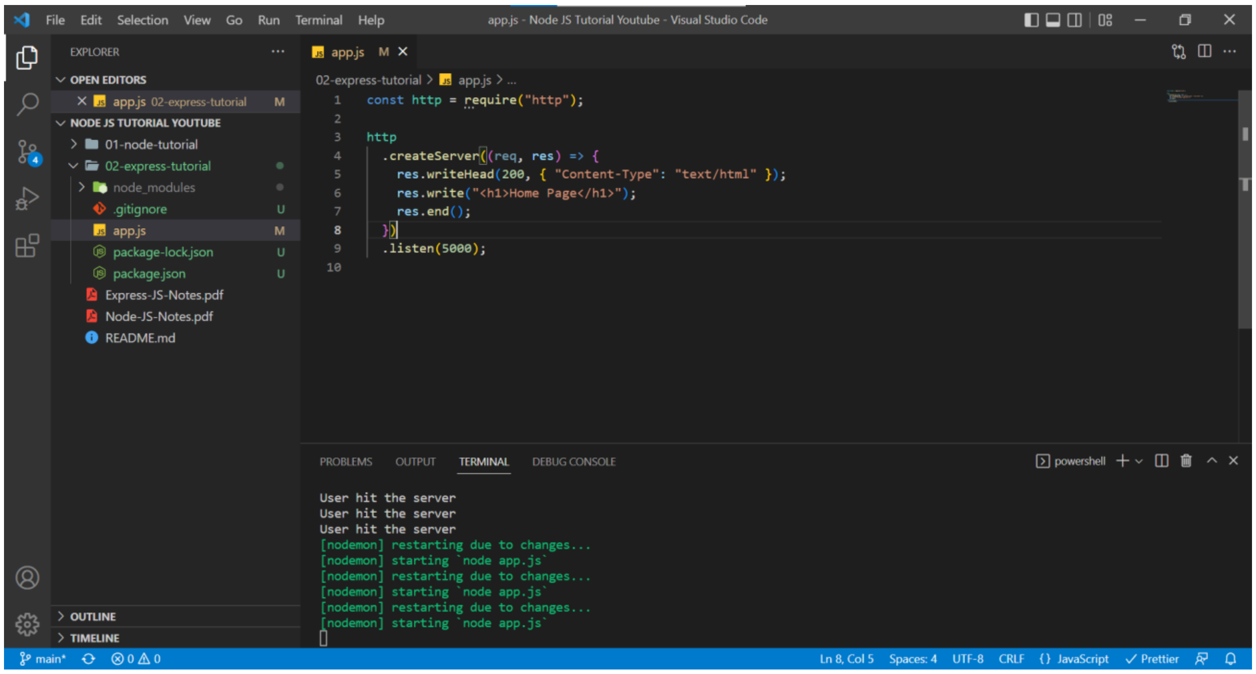
Ex: type/subtype; parameter = value

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

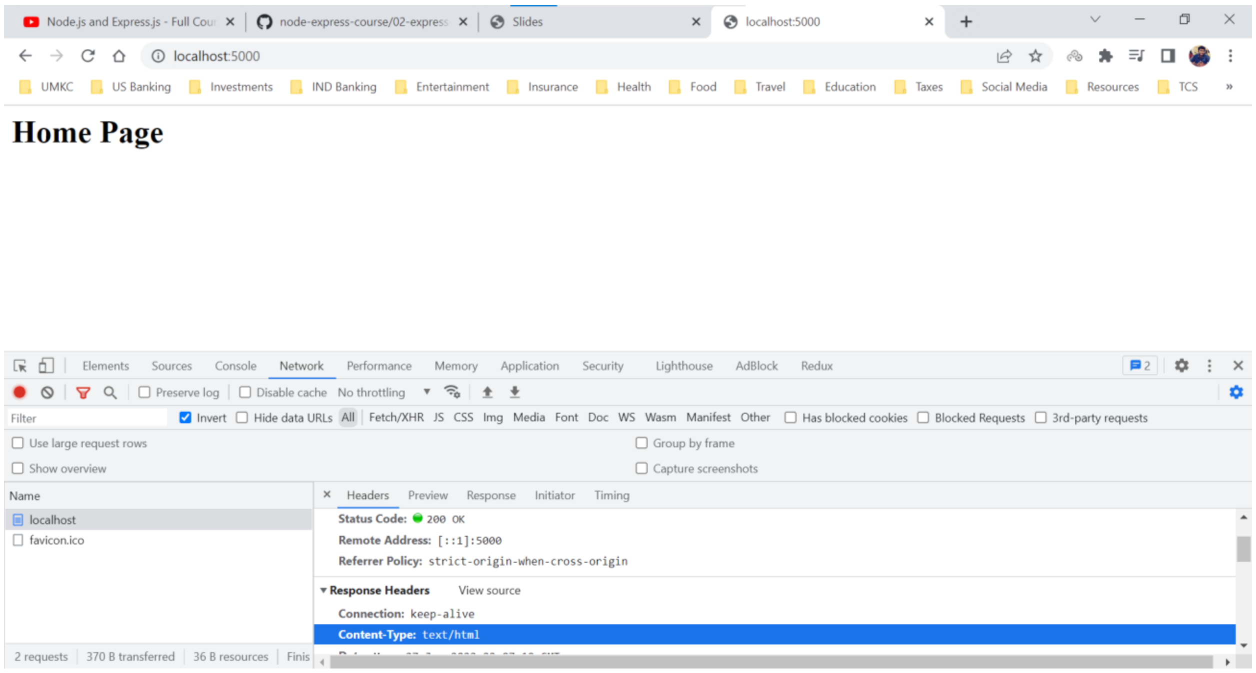


```
const http = require("http");

http
  .createServer((req, res) => {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write("<h1>Home Page</h1>");
    res.end();
  })
  .listen(5000);
```

```
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`
```

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

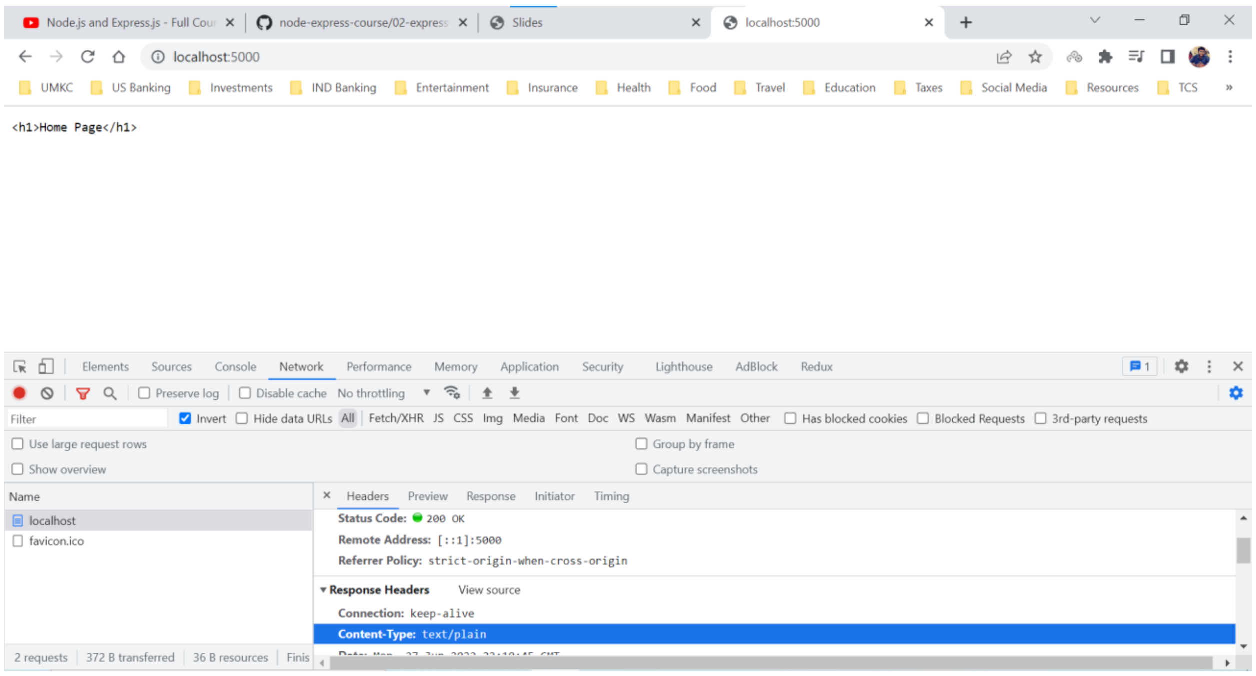


Home Page

```
Status Code: 200 OK
Remote Address: [::1]:5000
Referrer Policy: strict-origin-when-cross-origin

Response Headers
Connection: keep-alive
Content-Type: text/html
```

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



```
<h1>Home Page</h1>
```

```
Status Code: 200 OK
Remote Address: [::1]:5000
Referrer Policy: strict-origin-when-cross-origin

Response Headers
Connection: keep-alive
Content-Type: text/plain
```

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.

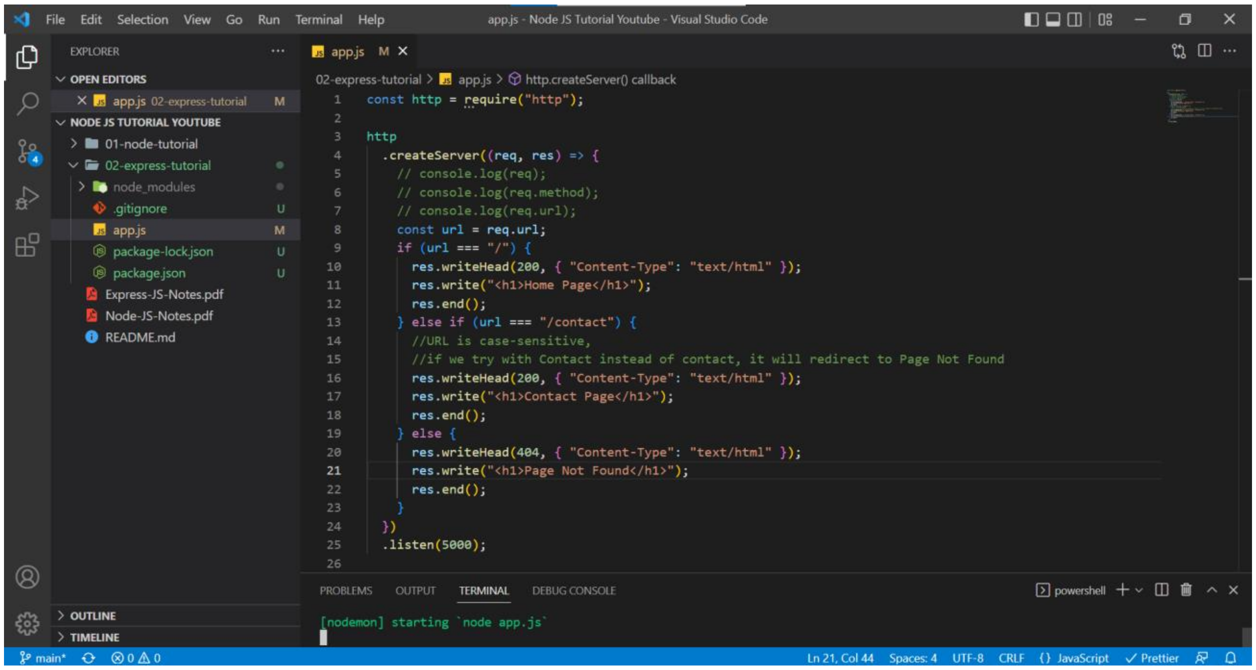
request.method – it is one of the properties which provides information about the HTTP method.

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

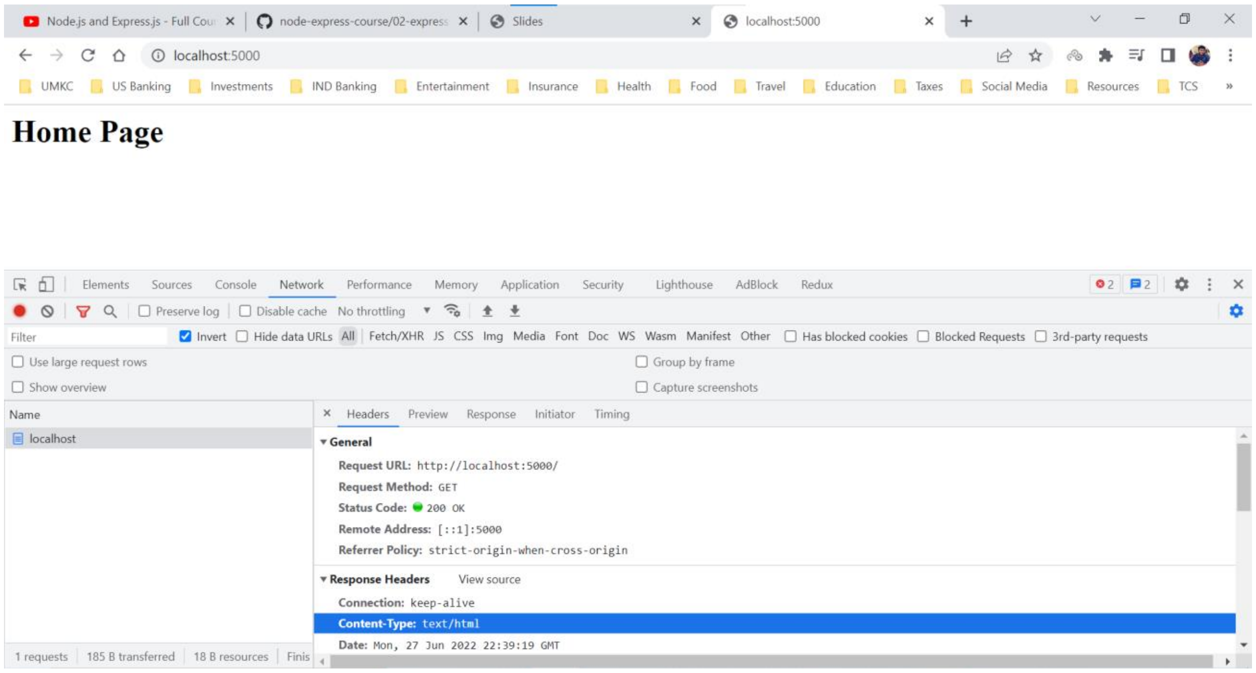


```
const http = require("http");

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);
```

```
[nodemon] starting `node app.js`
```

Home Page

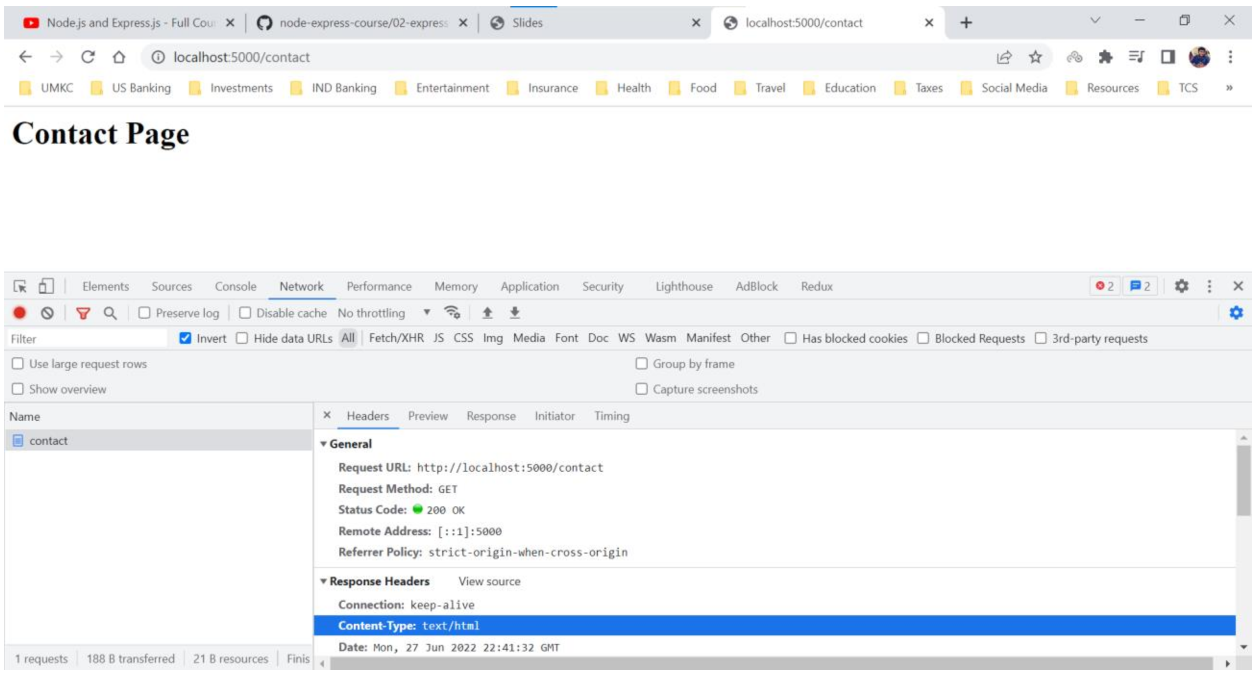


Home Page

```
General
  Request URL: http://localhost:5000/
  Request Method: GET
  Status Code: 200 OK
  Remote Address: [::1]:5000
  Referrer Policy: strict-origin-when-cross-origin
Response Headers    View source
  Connection: keep-alive
  Content-Type: text/html
  Date: Mon, 27 Jun 2022 22:39:19 GMT
```

```
1 requests | 185 B transferred | 18 B resources
```

Contact Page



localhost:5000/contact

Contact Page

contact

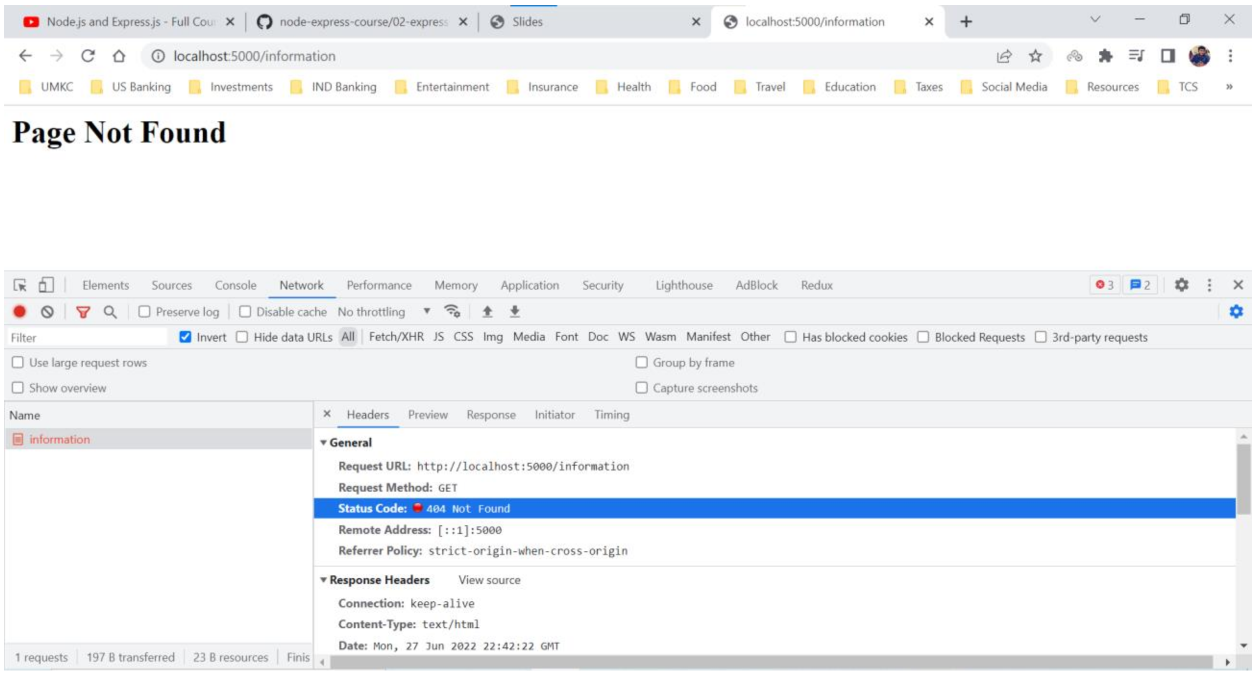
▾ General

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

▾ Response Headers

Connection: `keep-alive`
Content-Type: `text/html`
Date: `Mon, 27 Jun 2022 22:41:32 GMT`

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



localhost:5000/information

Page Not Found

information

▾ General

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

▾ Response Headers

Connection: `keep-alive`
Content-Type: `text/html`
Date: `Mon, 27 Jun 2022 22:42:22 GMT`

HTTP – HTML File

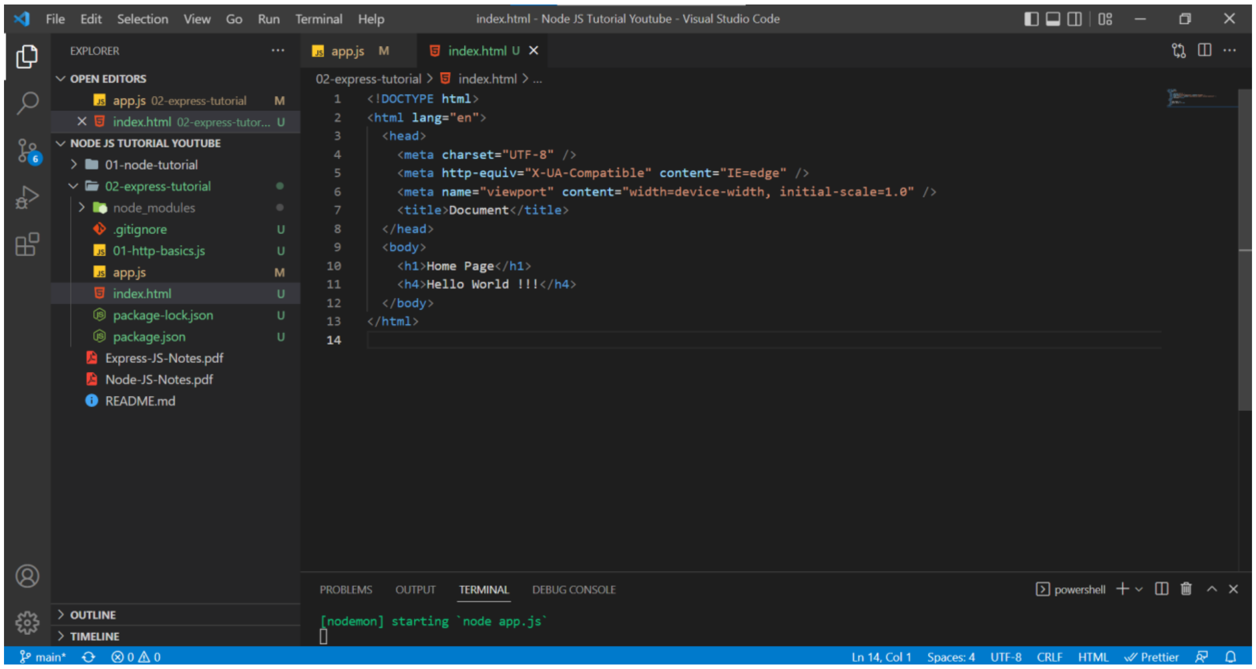
We are not limited to send the HTML directly into ***response.write()*** method or ***response.end()*** method. Instead, we can set up a file, request the file using ***File System*** and just passing it.

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.

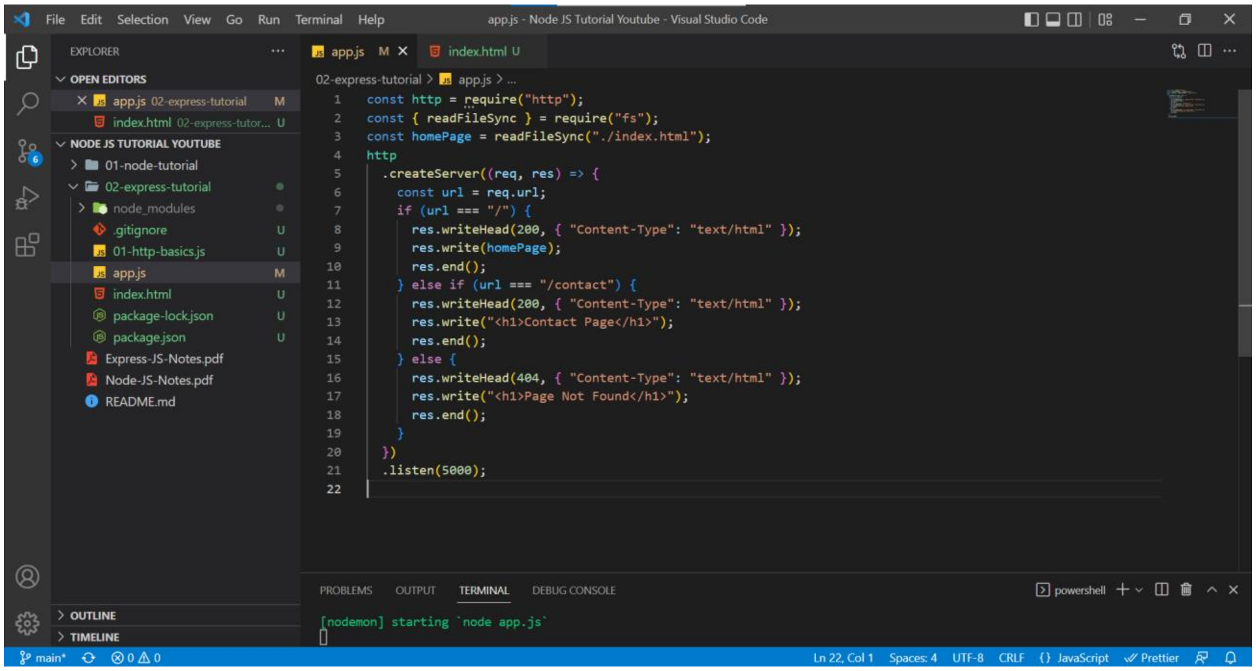
index.html



```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>
```

```
[nodemon] starting `node app.js`
```

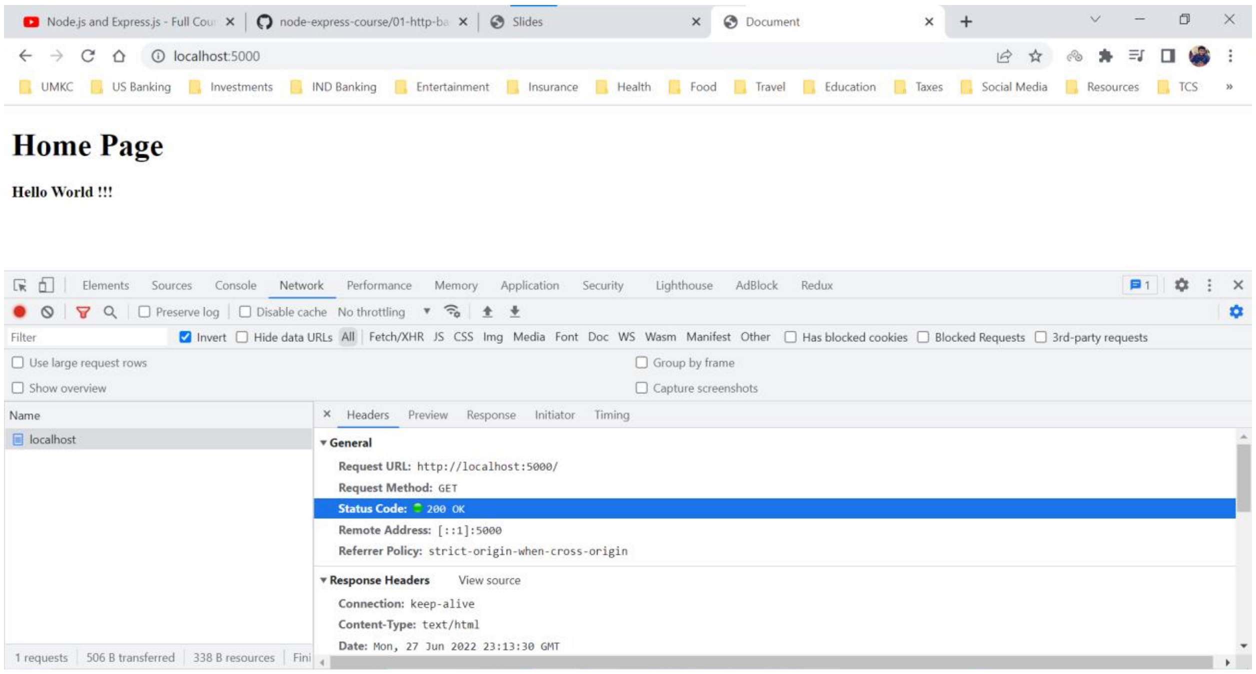
app.js



```js
const http = require("http");
const { readFileSync } = require("fs");
const homePage = readFileSync("./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);
```

```
[nodemon] starting `node app.js`
```

Home Page when Content Type is text/html.



localhost:5000

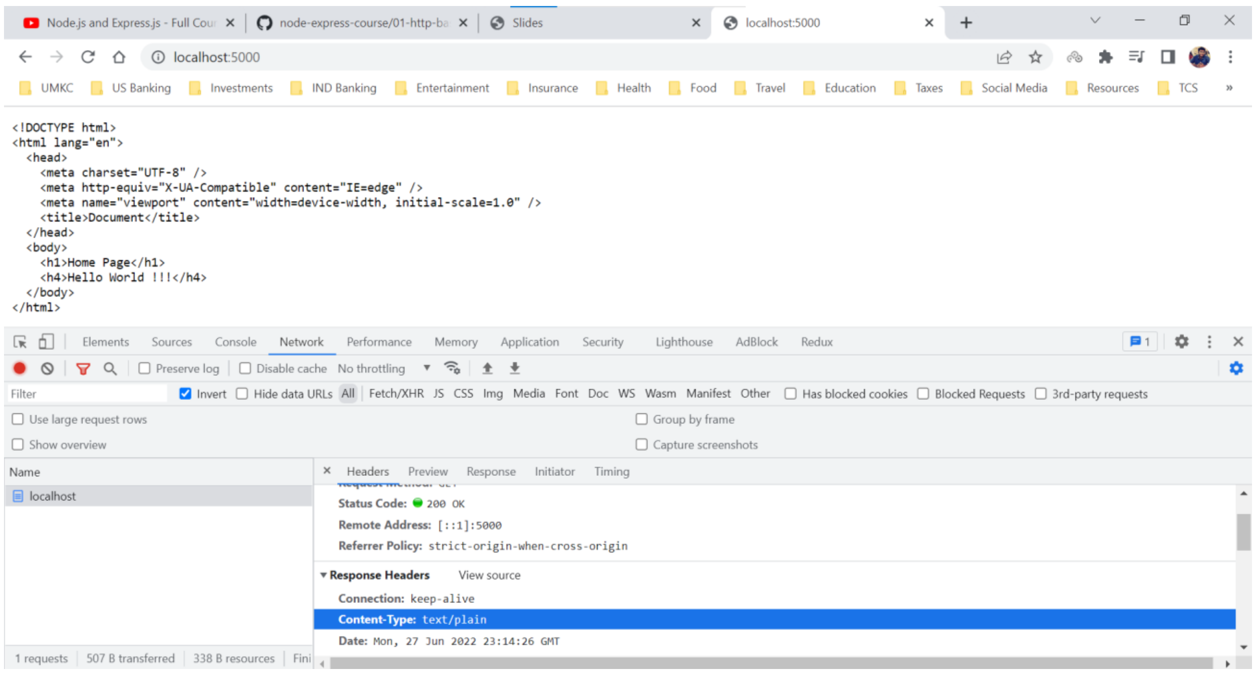
Home Page

Hello World !!!

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

Response Headers
Connection: keep-alive
Content-Type: text/html
Date: Mon, 27 Jun 2022 23:13:30 GMT
```

Home Page when Content Type is text/plain.


```
<!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>
```

```
Status Code: 200 OK
Remote Address: [::1]:5000
Referrer Policy: strict-origin-when-cross-origin

Response Headers
Connection: keep-alive
Content-Type: text/plain
Date: Mon, 27 Jun 2022 23:14:26 GMT
```

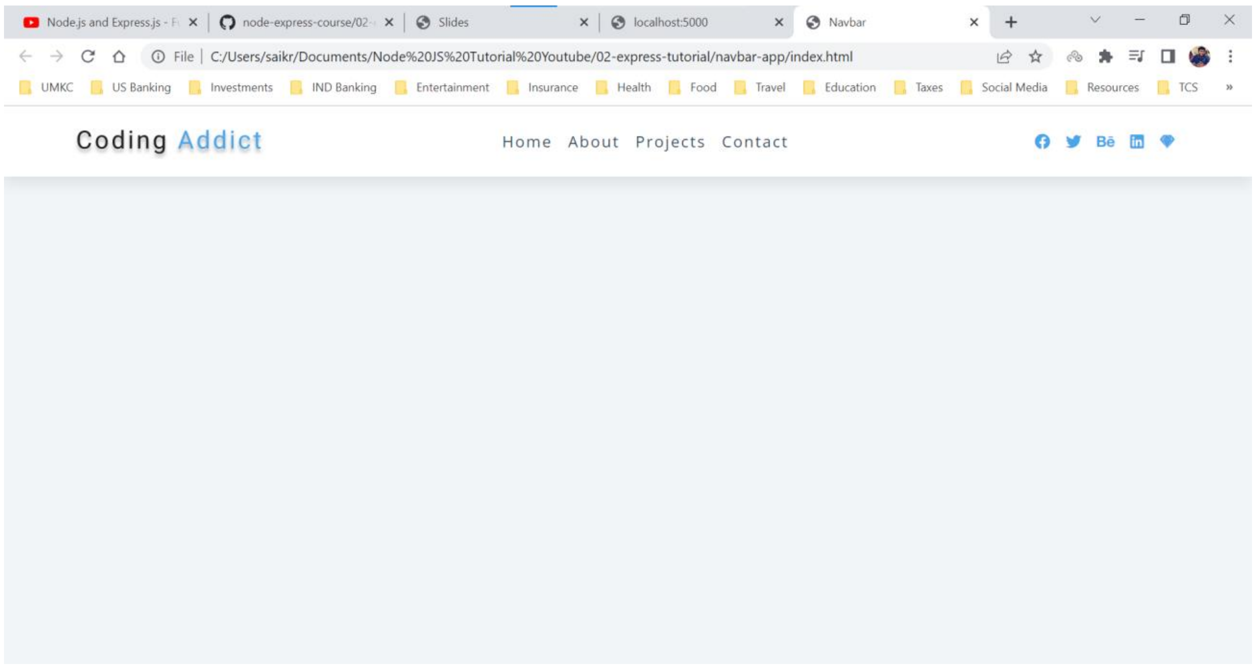
HTTP – App Example

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.

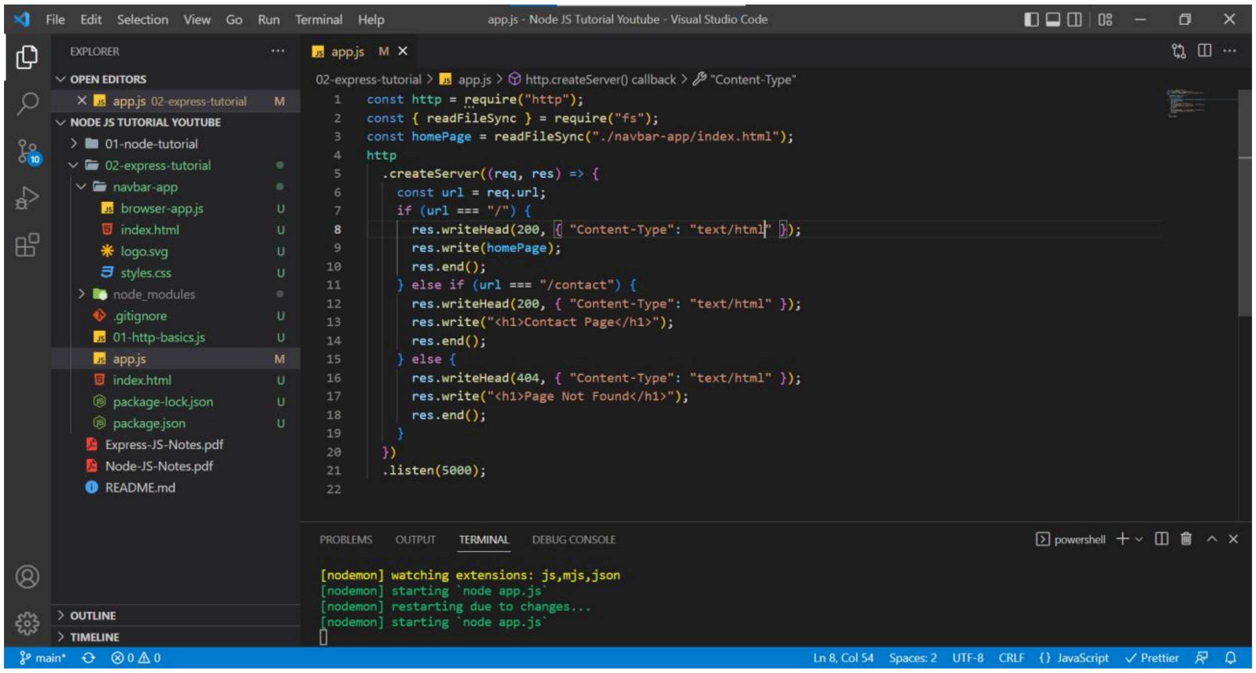
We try to run our server and connect to this application and provide responses based on the request.

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

Web Application


C:/Users/saikr/Documents/Node%20JS%20Tutorial%20Youtube/02-express-tutorial/navbar-app/index.html
Coding Addict
Home
About
Projects
Contact

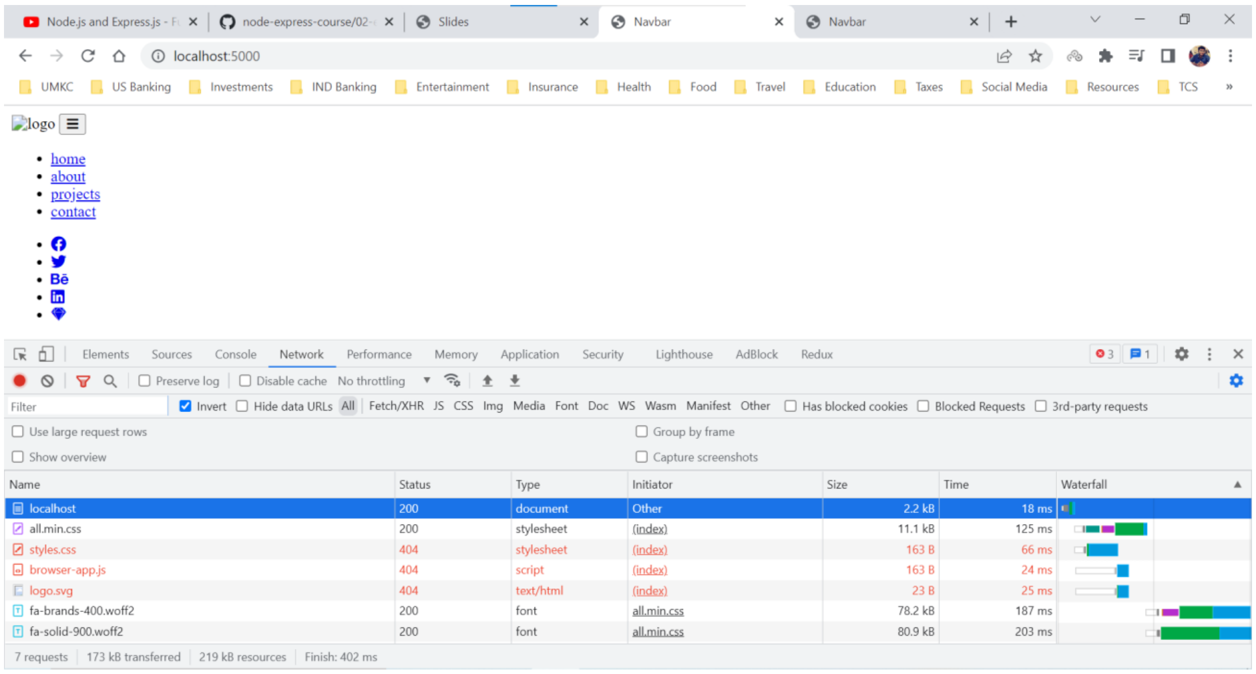
app.js



```
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);
```

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

localhost:5000



- home
- about
- projects
- contact

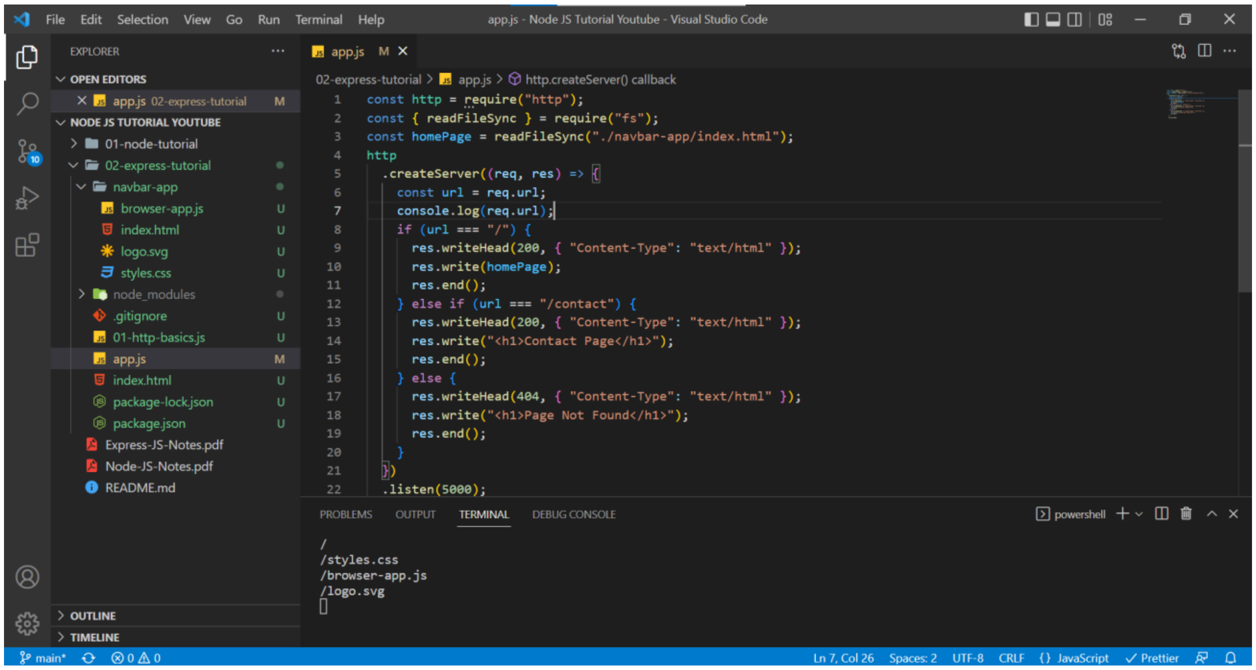
Name	Status	Type	Initiator	Size	Time
localhost	200	document	Other	2.2 kB	18 ms
all.min.css	200	stylesheet	(index)	11.1 kB	125 ms
styles.css	404	stylesheet	(index)	163 B	66 ms
browser-app.js	404	script	(index)	163 B	24 ms
logo.svg	404	text/html	(index)	23 B	25 ms
fa-brands-400.woff2	200	font	all.min.css	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 ae not handling those requests (styles.css, logo.svg, browser-app.js) in our server.

We can console log the requests by the web application

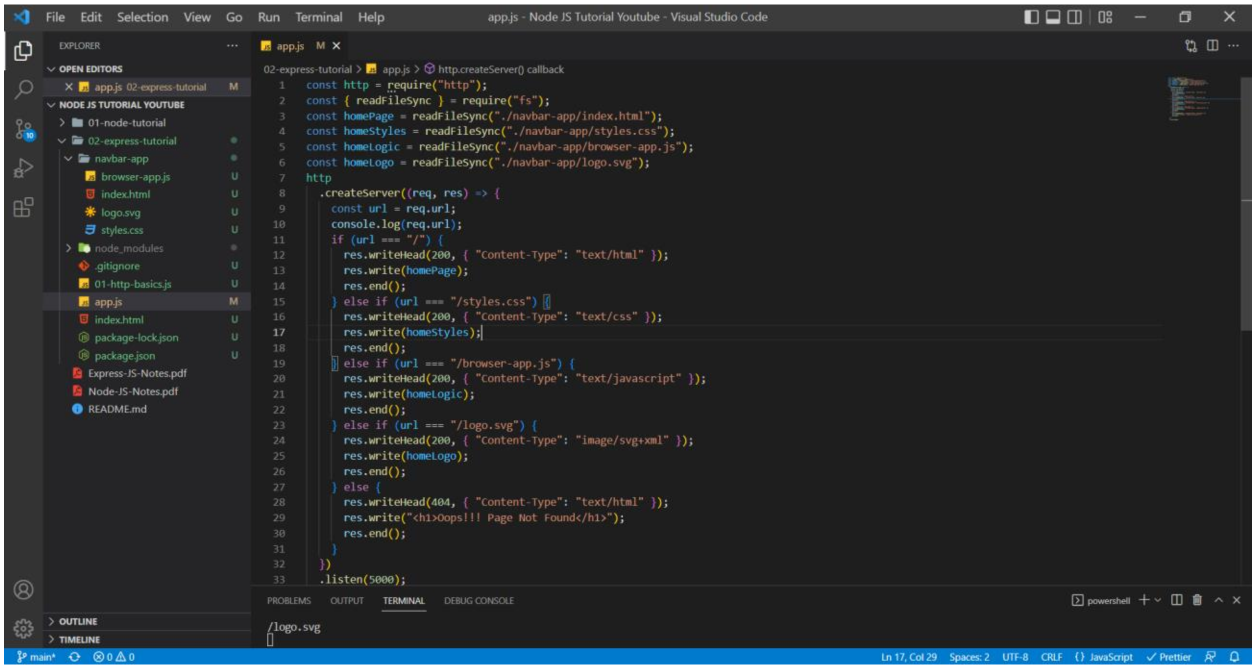


```javascript
const http = require("http");
const { readFileSync } = require("fs");
const homePage = readFileSync("./navbar-app/index.html");
http
  .createServer((req, res) => {
    const url = req.url;
    console.log(req.url);
    if (url === "/") {
      res.writeHead(200, { "Content-Type": "text/html" });
      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);
```

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

We need to handle those requests and provide responses.

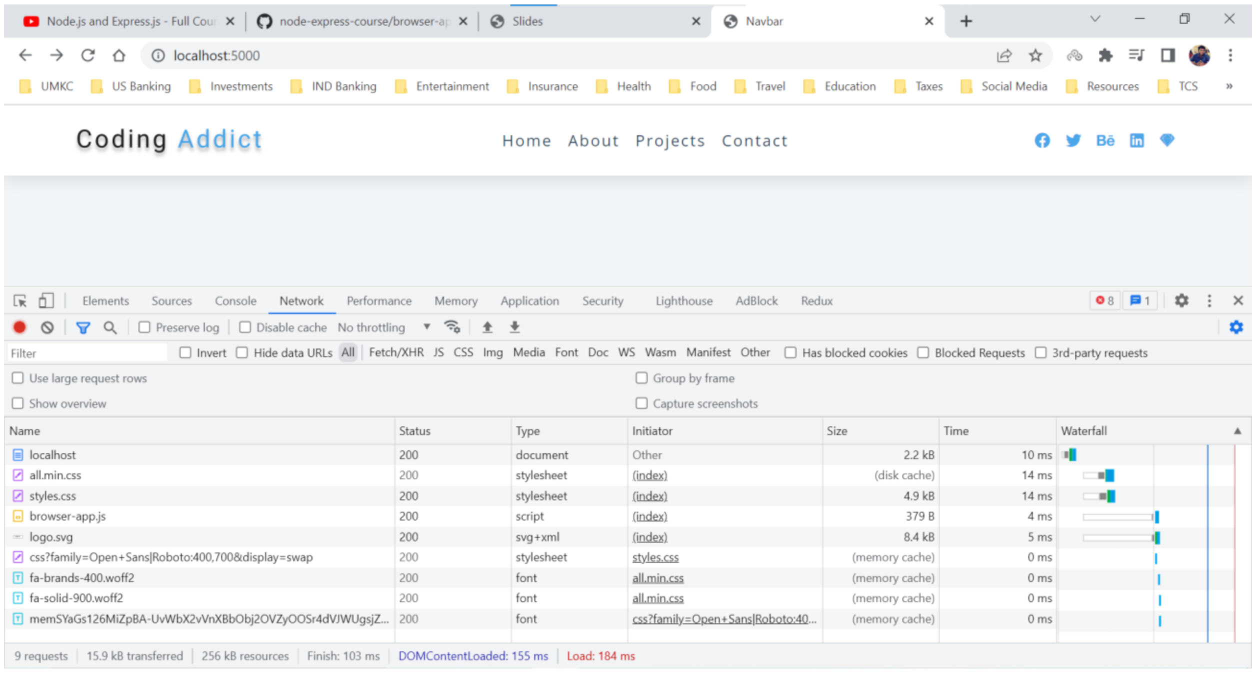
app.js



```
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 homeLogo = 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(homeLogo);
      res.end();
    } else {
      res.writeHead(404, { "Content-Type": "text/html" });
      res.write("<h1>Oops!!! Page Not Found</h1>");
      res.end();
    }
  })
  .listen(5000);
```

```
/logo.svg
```

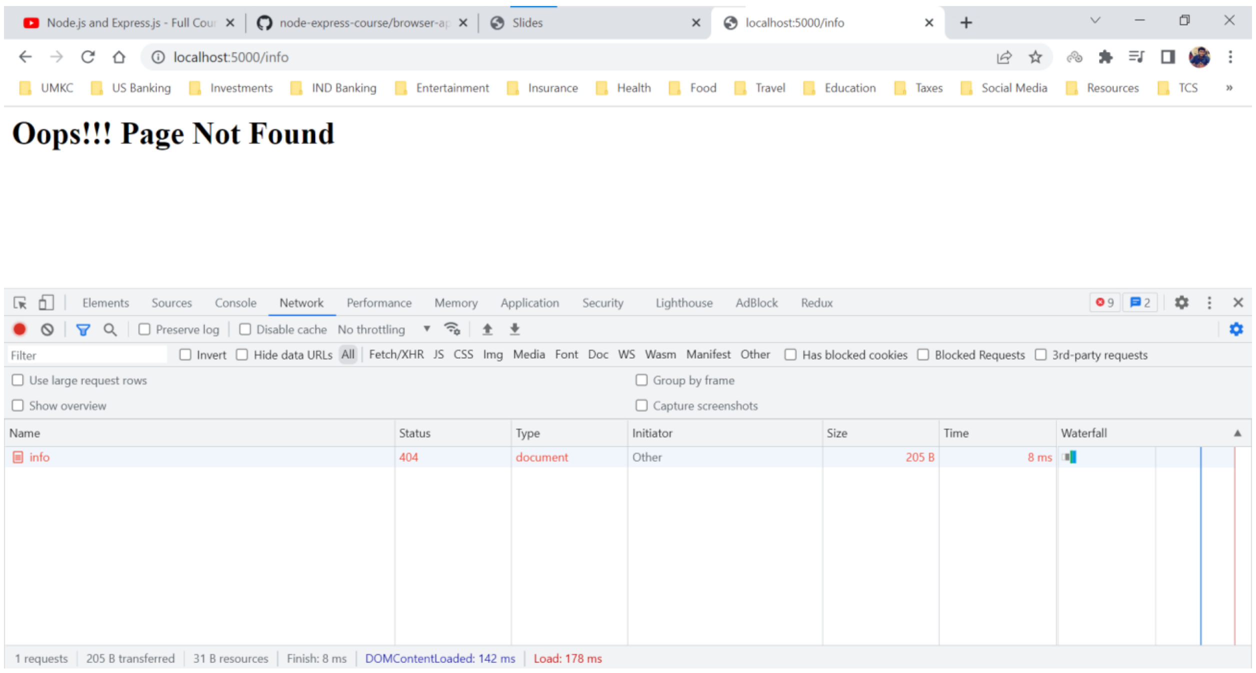
We are serving all the requests and let us see the response in browser.



Name	Status	Type	Initiator	Size	Time
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-UvWbX2vVnXBbObj2OVZyOOSr4dVJWUgsjZ...	200	font	css?family=Open+Sans|Roboto:40...	(memory cache)	0 ms

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.



```
localhost:5000/info
```

Oops!!! Page Not Found

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

```
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

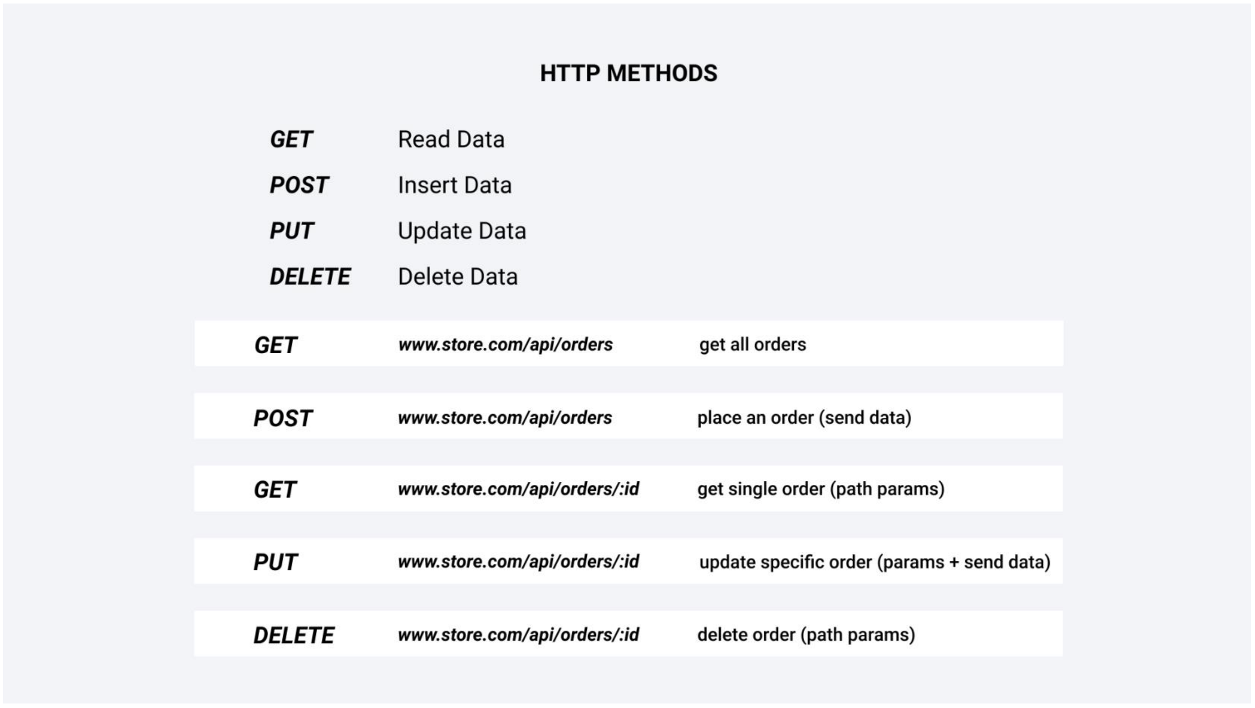
npm install express@4.17.1 - -save

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.

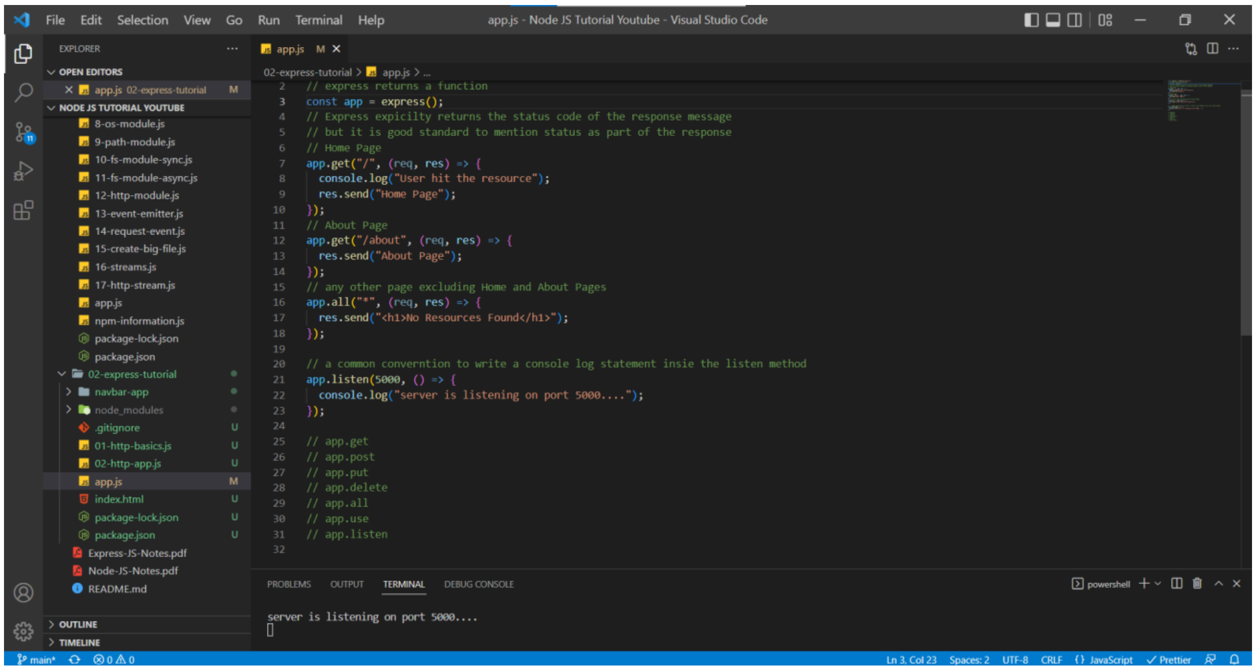


HTTP METHODS

GET	Read Data
POST	Insert Data
PUT	Update Data
DELETE	Delete Data

GET	***www.store.com/api/orders***	get all orders
POST	***www.store.com/api/orders***	place an order (send data)
GET	***www.store.com/api/orders/:id***	get single order (path params)
PUT	***www.store.com/api/orders/:id***	update specific order (params + send data)
DELETE	***www.store.com/api/orders/:id***	delete order (path params)

app.js



```javascript
// express returns a function
const app = express();
// Express expicilty 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.send("Home Page");
});
// About Page
app.get("/about", (req, res) => {
  res.send("About Page");
});
// any other page excluding Home and About Pages
app.all("*", (req, res) => {
  res.send("<h1>No Resources Found</h1>");
});

// a common converntion to write a console log statement insie 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

```

```
server is listening on port 5000....
```

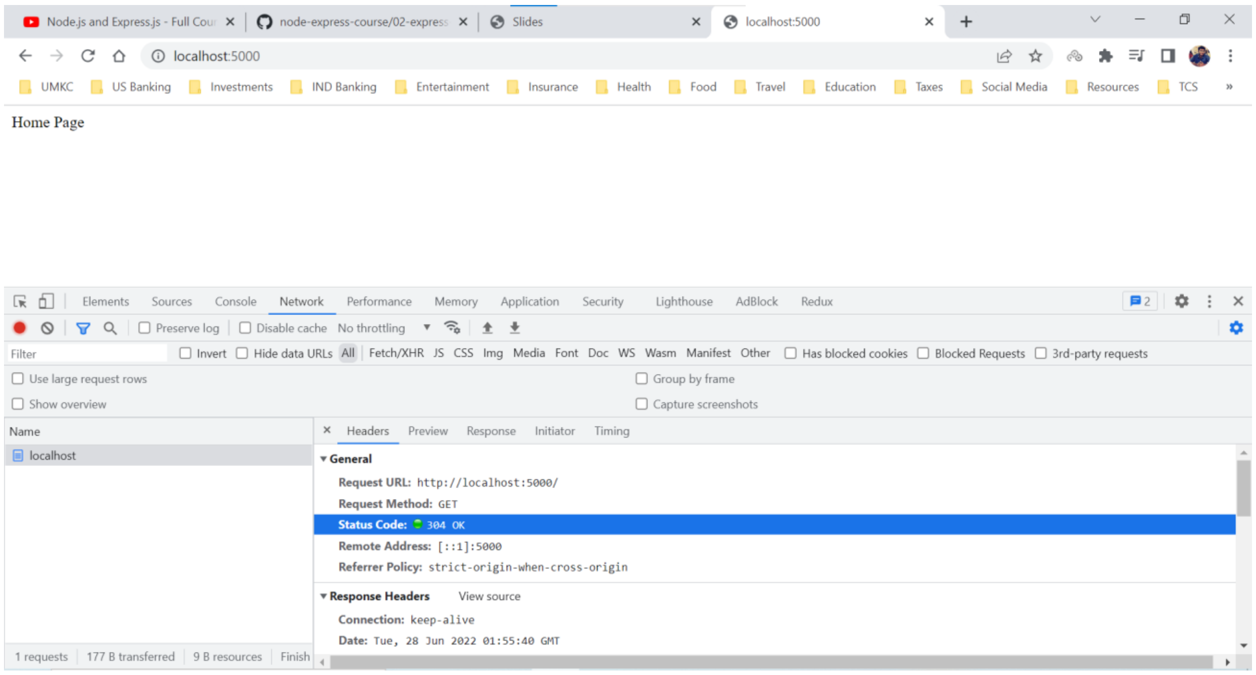
app.all – this method works with all of them. It is used handle to requests which doesn’t have any resources.

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

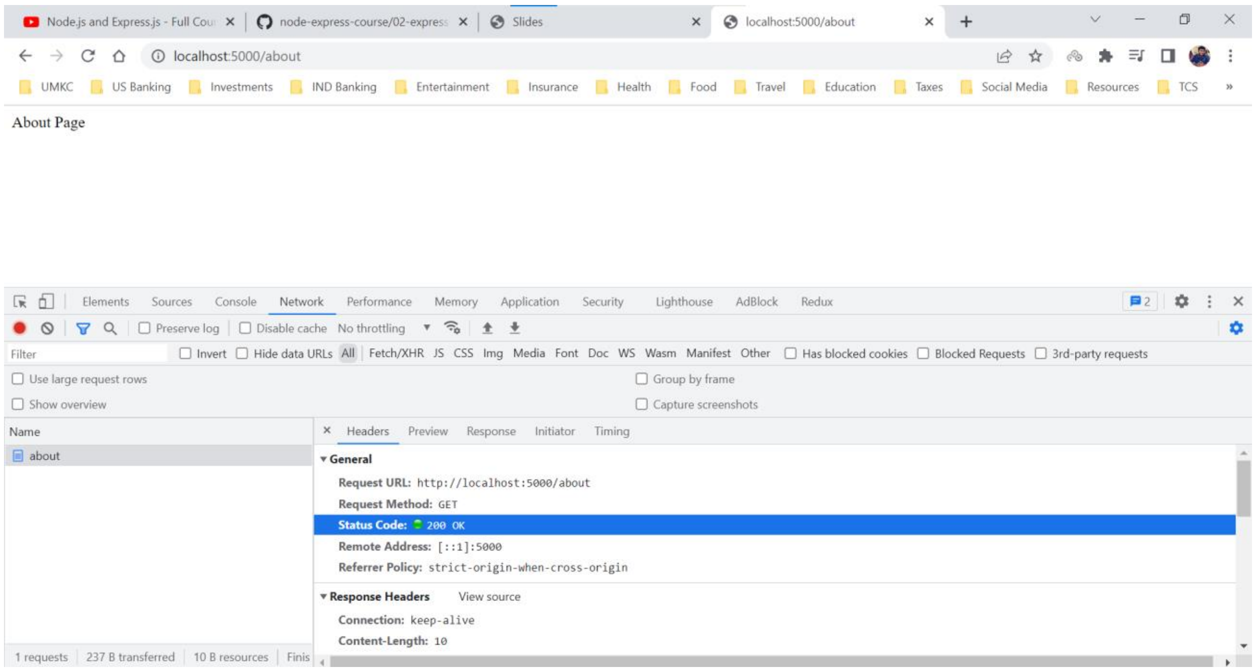


Home Page

```
▾ General
    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
    Date: Tue, 28 Jun 2022 01:55:40 GMT
```

1 requests | 177 B transferred | 9 B resources | Finish

About Page

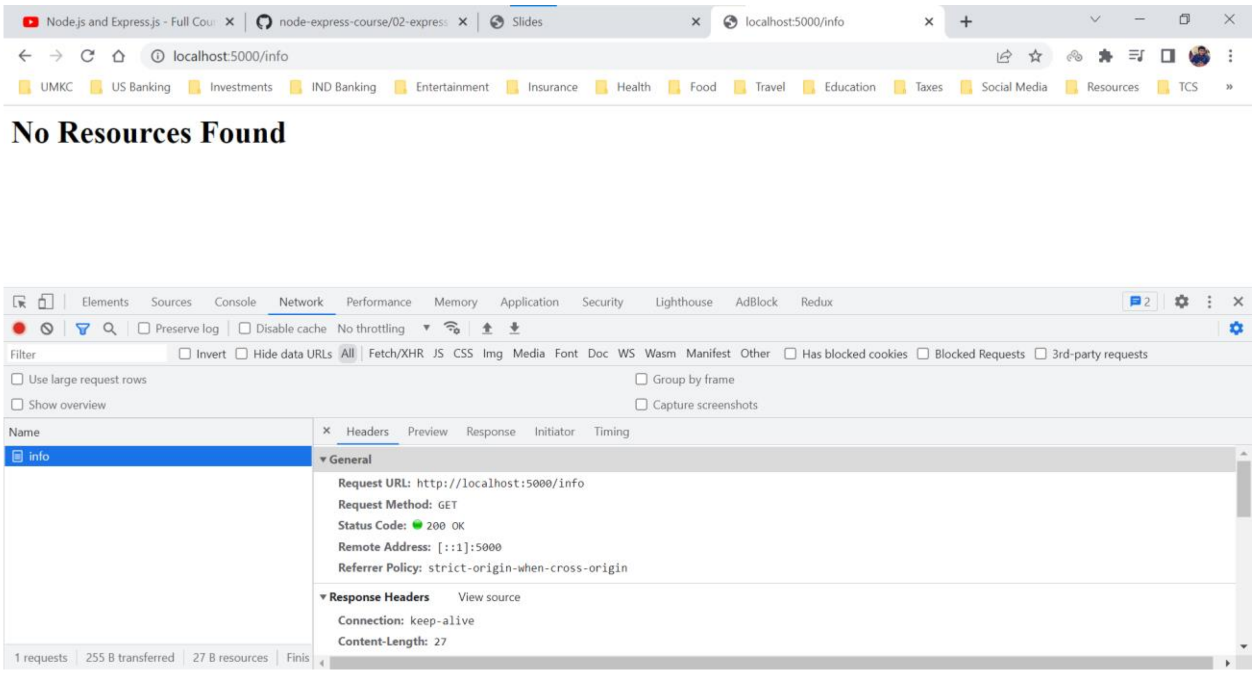


About Page

```
▾ General
    Request URL: http://localhost:5000/about
    Request Method: GET
    Status Code: ● 200 OK
    Remote Address: [::1]:5000
    Referrer Policy: strict-origin-when-cross-origin
▾ Response Headers    View source
    Connection: keep-alive
    Content-Length: 10
```

1 requests | 237 B transferred | 10 B resources | Finis

Resources which are not handled



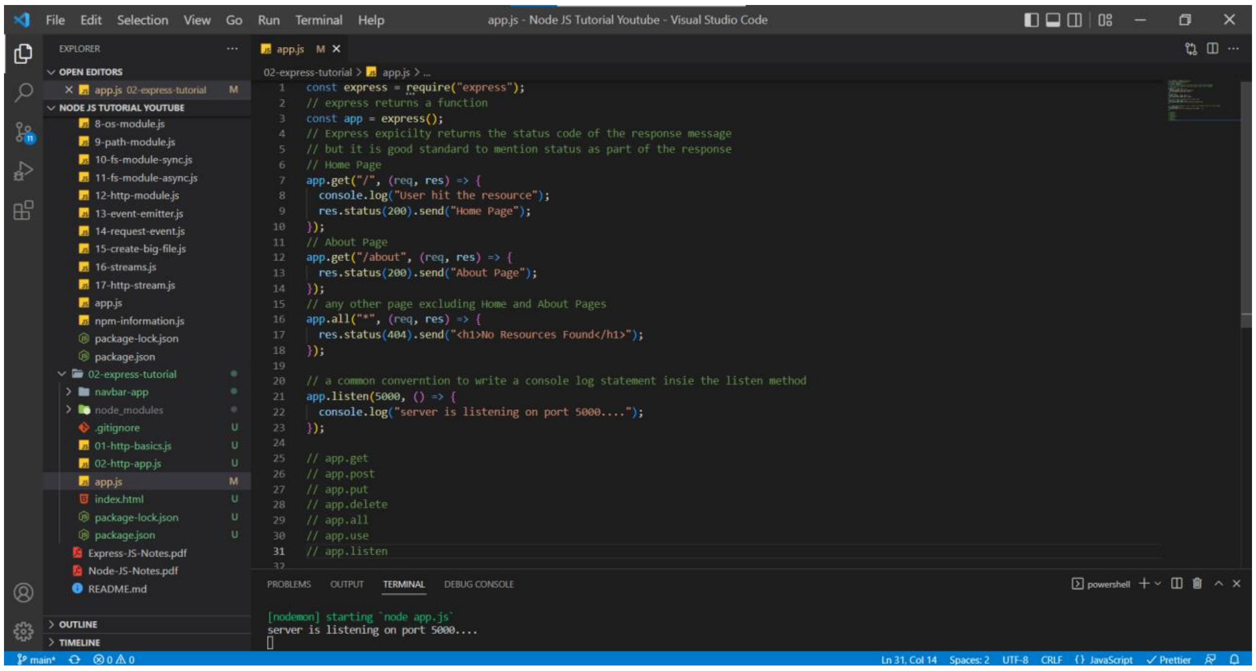
No Resources Found

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

Response Headers
Connection: keep-alive
Content-Length: 27
```

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



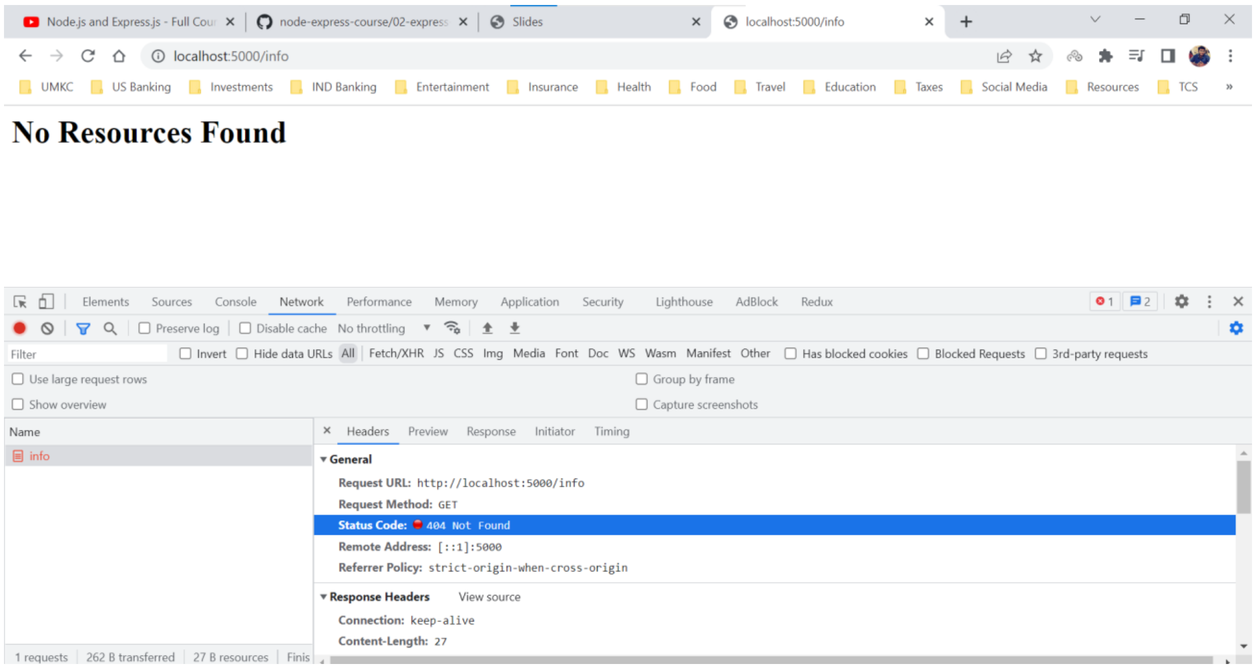
```
const express = require("express");
// express returns a function
const app = express();
// Express expicilty returns the status code of the response message
// but it is good standard to mention status as part of the response
// Home Page
app.get("/", (req, res) => {
  console.log("User hit the 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 converntion to write a console log statement insie 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
```

```
[nodemon] starting `node app.js`
server is listening on port 5000....
```

Resources which are not handled


localhost:5000/info

No Resources Found

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

Response Headers
Connection: keep-alive
Content-Length: 27

1 requests | 262 B transferred | 27 B resources

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

Express – App Example