**Steps to create Backend :-**

Do everything in a “server” folder.

1. Open the visual studio, make a folder named “**mernstackproj**” and open it, inside it make a “**server**” folder.
2. Go to the terminal, inside the server folder, using command **cd server**
3. initializing our project as a node project, by using “**npm init -y**”, it will make a package.json file.
4. Install Express & nodemon using, “**npm i express**”
5. Install nodemon globally using “npm install -g nodemon”
6. install mongoose using, “**npm i mongoose**”
7. After making the backend server and routing, we will connect our database MongoDB Atlas with the backend server.
8. Create database, cluster, and copy the connection link :

mongodb+srv://shiv:<password>@cluster0.ci7ti.mongodb.net/?retryWrites=true&w=majority

=>add new project

=>create new database

=>go to that database

=>Click on create button to create a cluster

OR

=>After clicking on add project, provide username & password for the database

username : shiv

password : shiv123

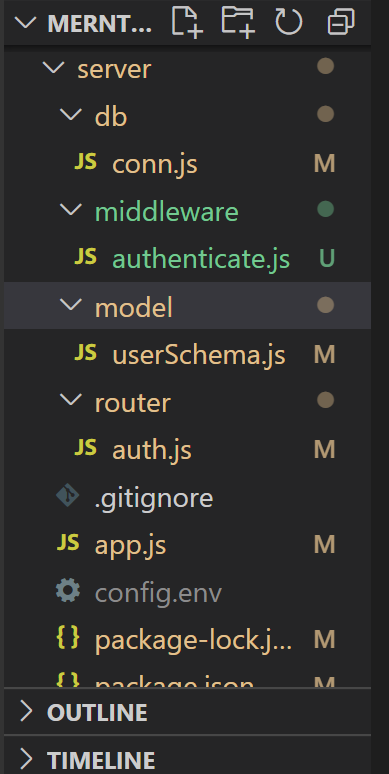
Database

=>Allow access to all ip addresses, through the Network access tab.

=>copy the connection link, and replace the password field with your actual password, and add the name of database, between / and ?

**mongodb+srv://shiv:shiv123@cluster0.oudfe.mongodb.net/demo?retryWrites=true&w=majority**

1. Create a folder called db , inside it make a file called conn.js (db/conn)



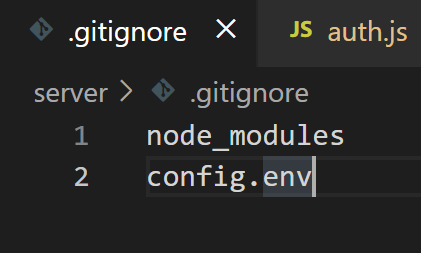
1. Click on app.js, then make a file inside the server folder called **.gitignore & config.env**. All the file names that we add in **.gitignore** file, will be ignored, and it will be hidden when uploading to git, and **config.env.**
2. <https://www.npmjs.com/package/dotenv>
3. **What is dotenv ?**

dotenv allows you to separate secrets from your source code. This is useful in a collaborative environment (e.g., work, or open source) where you may not want to share your database login credentials with other people. Instead, you can share the source code while allowing other people to create their own.

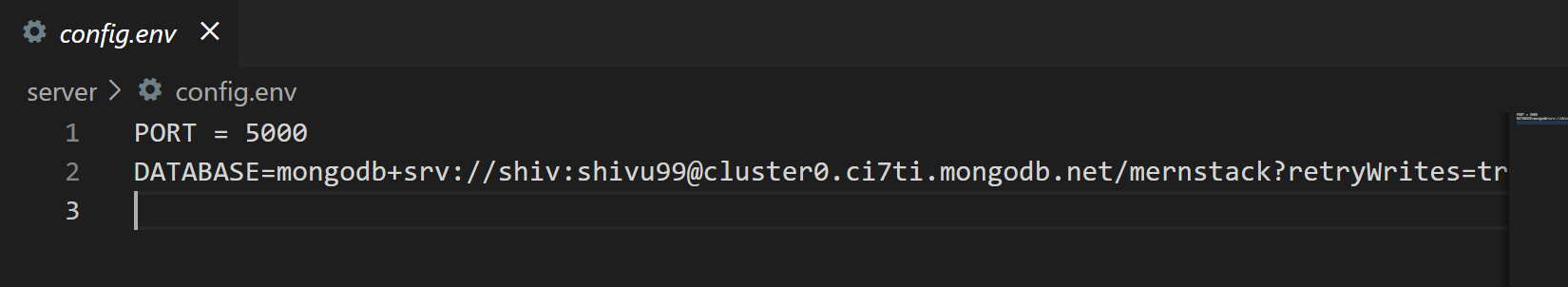
1. Install npm package called “**npm i dotenv**”

Add PORT and DATABASE in the config.env file. (these 2 values will be hidden from others).

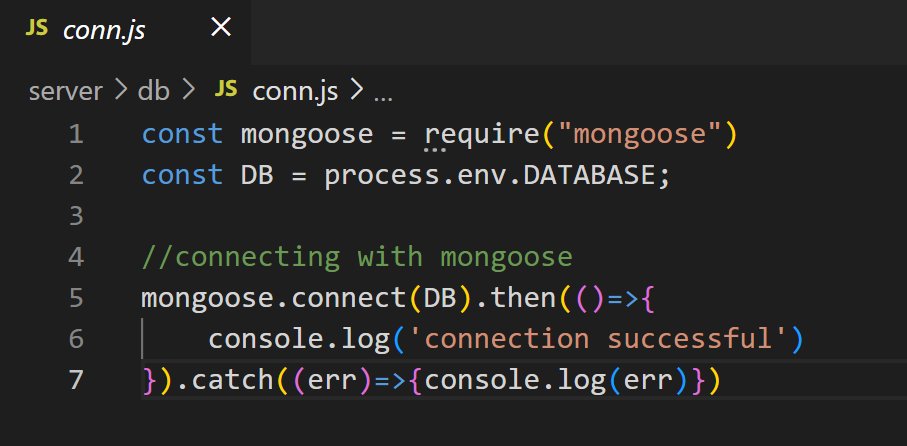
1. Add the file names of “node\_modules” & “config.env” into **.gitignore** file



1. Add PORT & DATABASE values inside the config.env file to keep them protected.

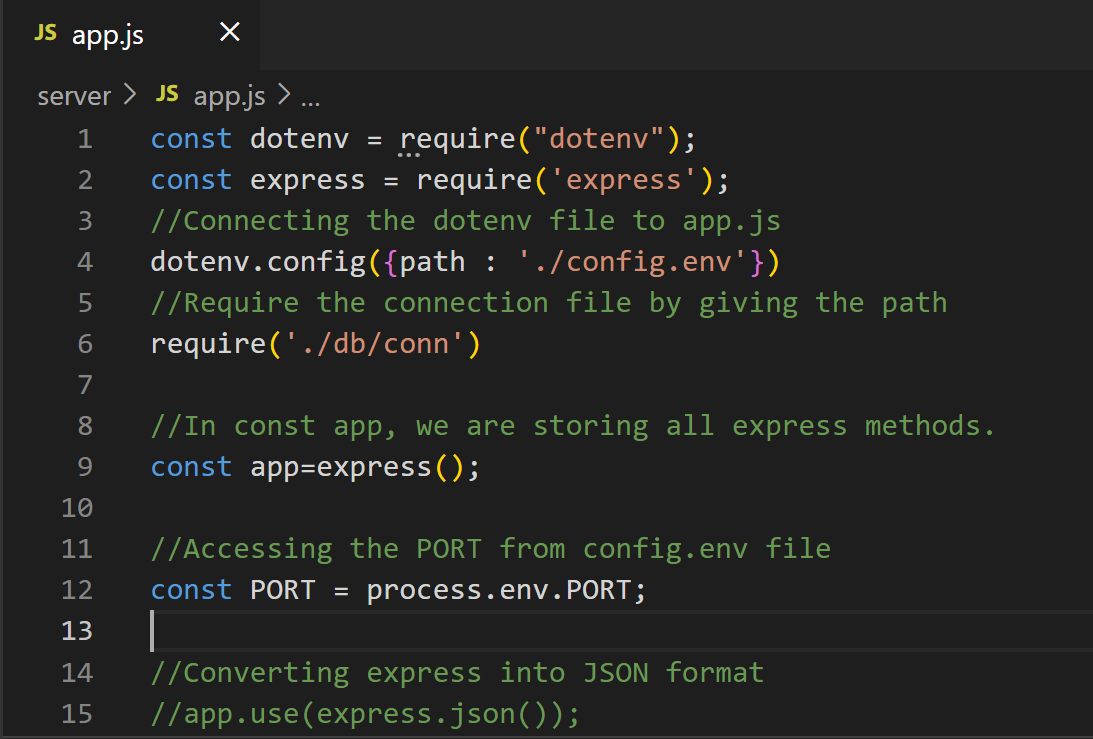


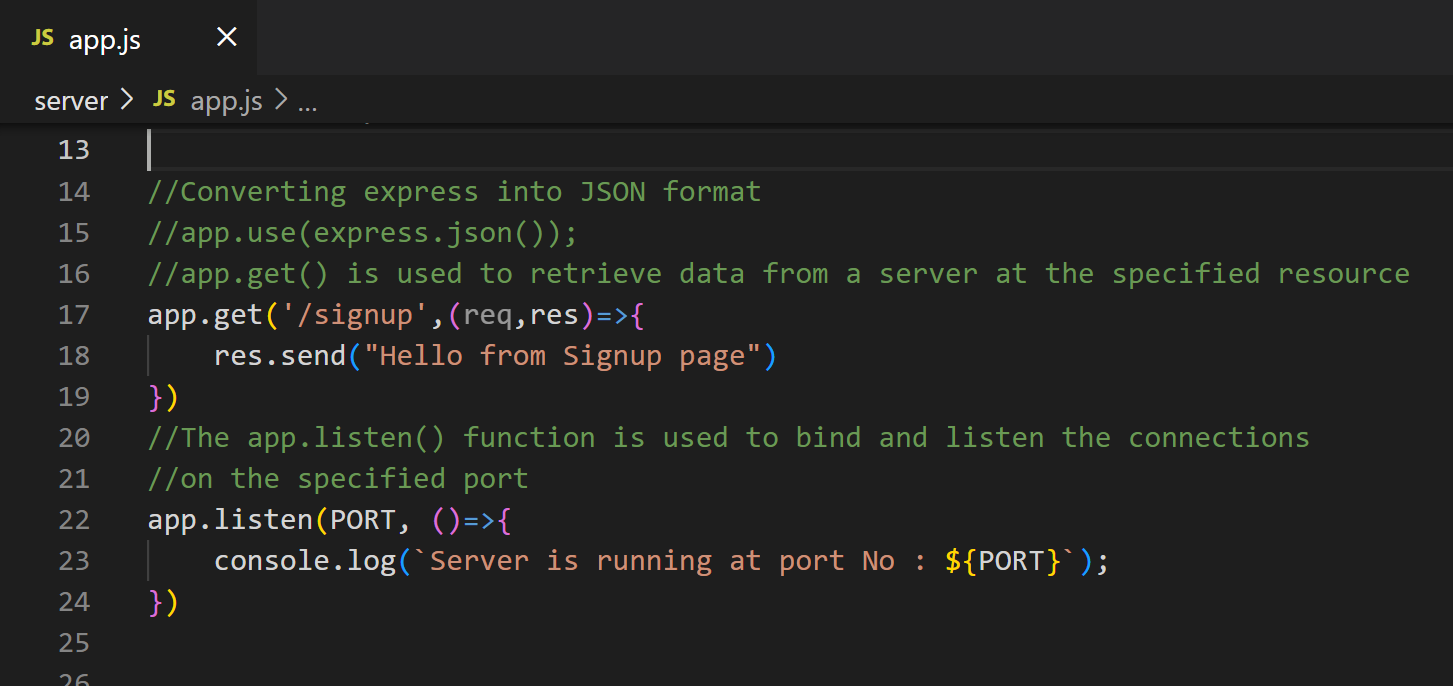
1. Go to conn.js , using promises, create a connection



1. Create an app.js then we will run basic server code
2. **What is Express ?**

Express is a node js web application framework that provides broad features for building web and mobile applications. It is used to build a single page, multipage, and hybrid web application. It's a layer built on the top of the Node js that helps manage servers and routes.

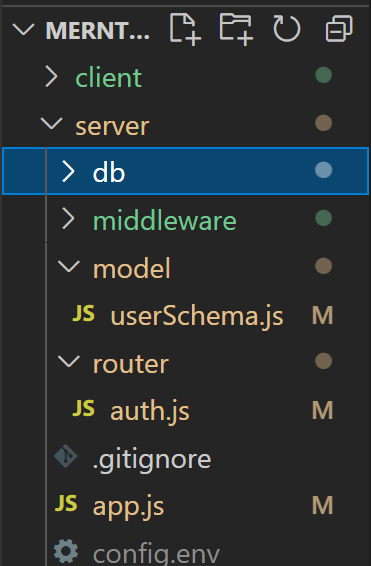




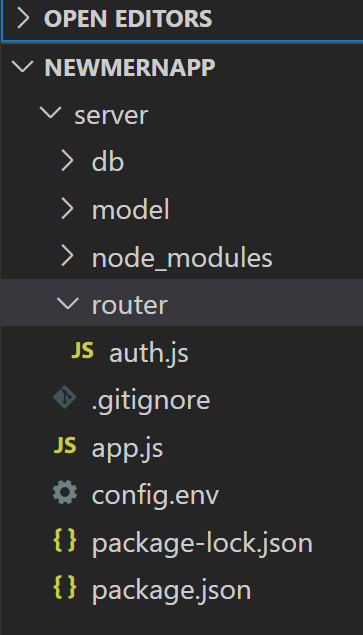
**GET METHOD : -**

GET requests are the most common and widely used methods in APIs and websites. Simply put, the GET method is used to **retrieve data from a server at the specified resource**. For example, say you have an API with a /users endpoint. Making a GET request to that endpoint should return a list of all available users.

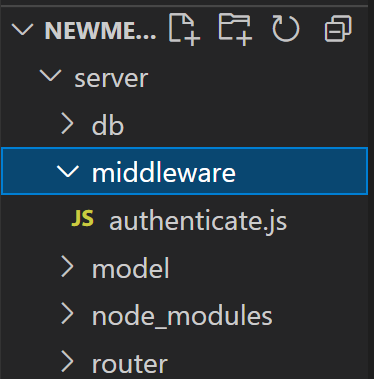
1. We will make Schema & models for our database (mongoose)
2. Create a folder called “**model**” inside that make a file called “**userSchema.js**”



1. Create a folder called “**router**” inside that make a file called “**auth.js**” ( we will give all the routes in this file)



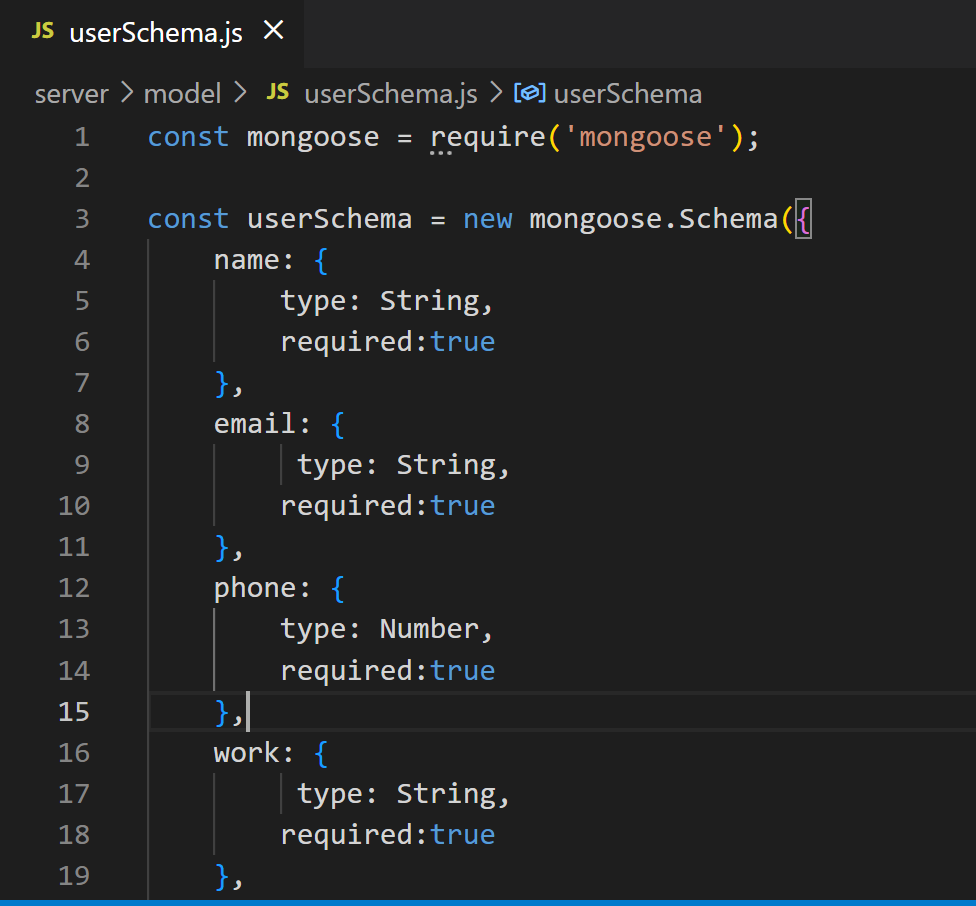
1. Create a folder called “**middleware**” inside that make a file called “**authenticate.js**”

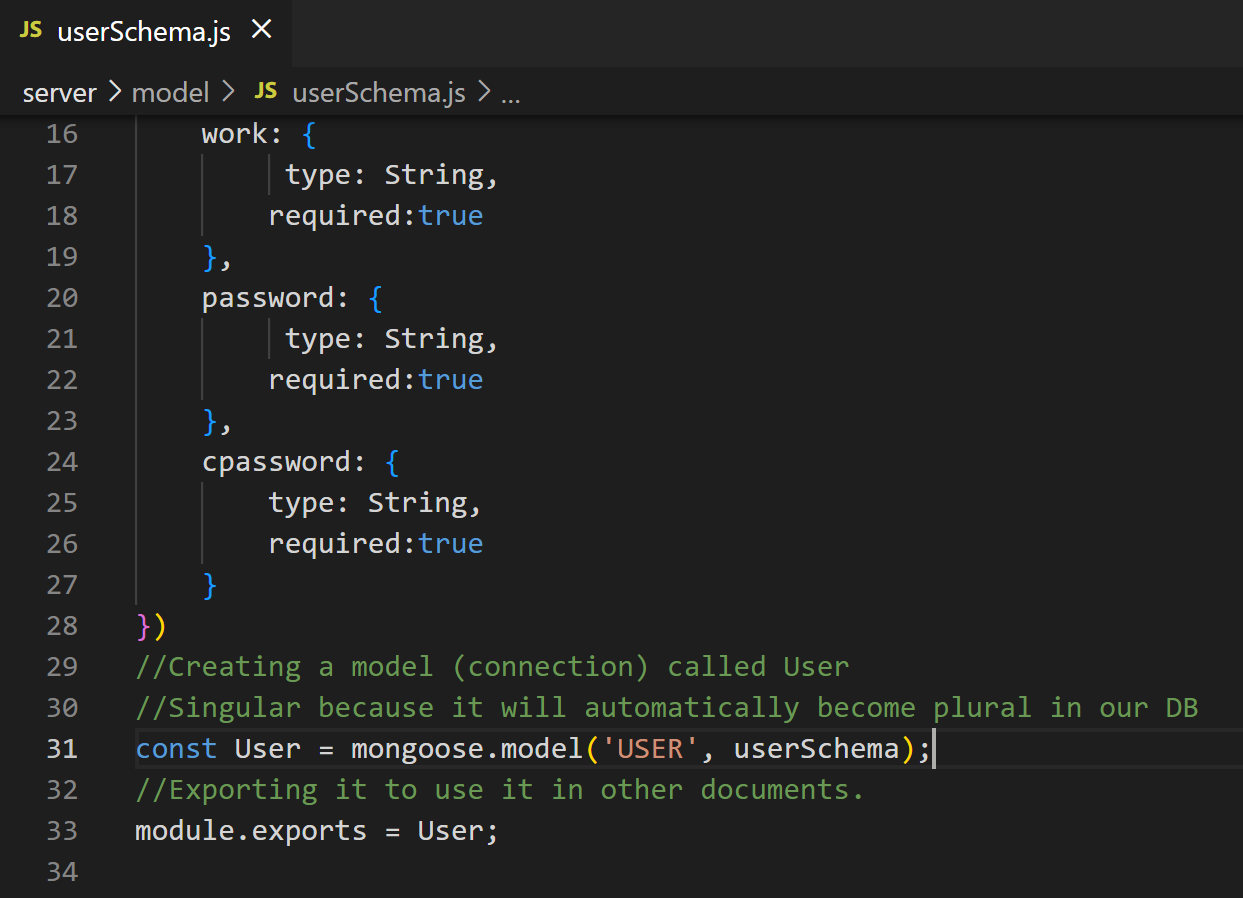


**Q. What is middleware?**

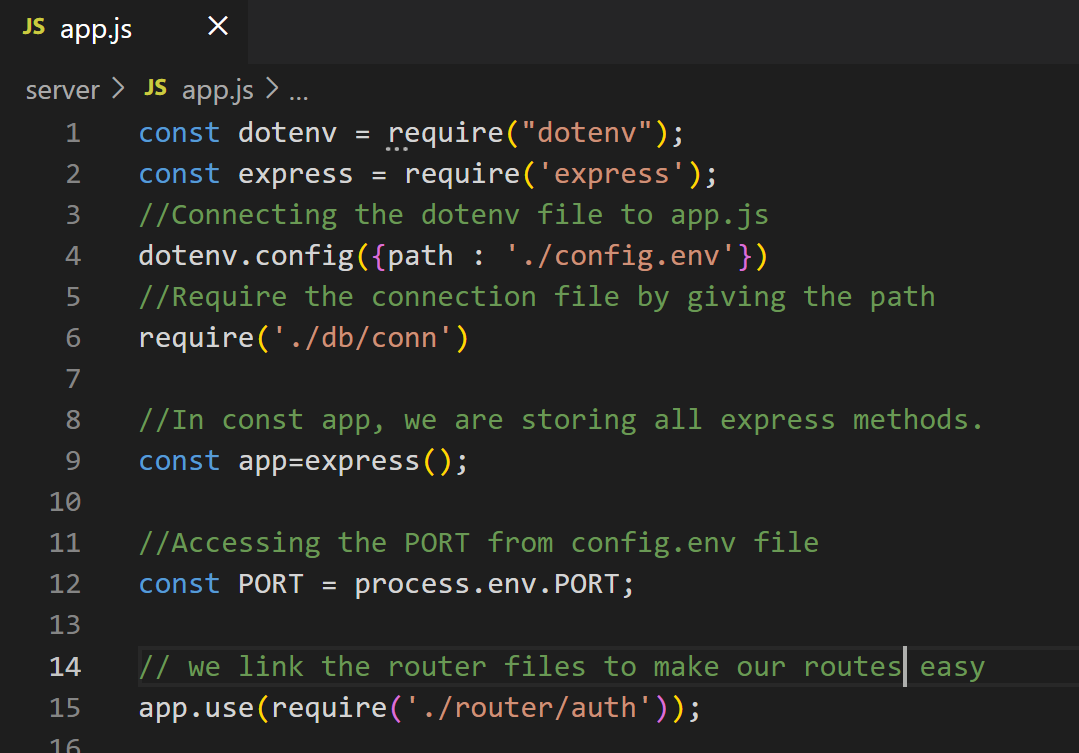
Middleware functions are functions that have access to the request object ( req ), the response object ( res ), and the next middleware function in the application's request-response cycle.

1. We will go to model/userSchema.js, and make the Schema & model for our collections.(Copy the Code from other Doc)

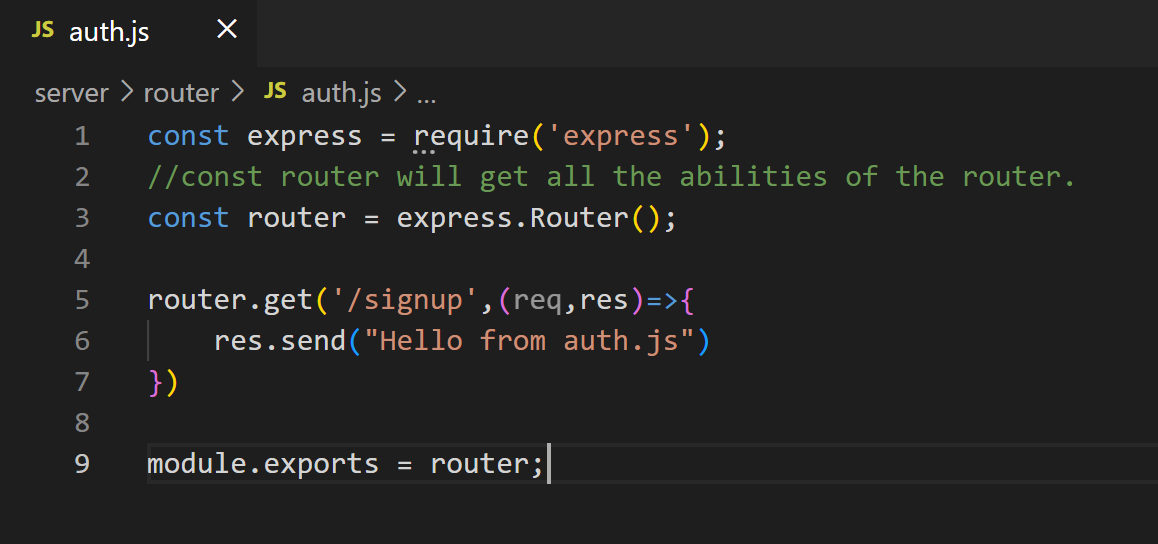




1. We will set up all the routes of our application, inside **router/auth.js**, and we will require the same file inside our app.js, using the command **app.use(require('./router/auth'));**

****

1. **Now we will write the code in router/auth.js file, to separate the routes.**

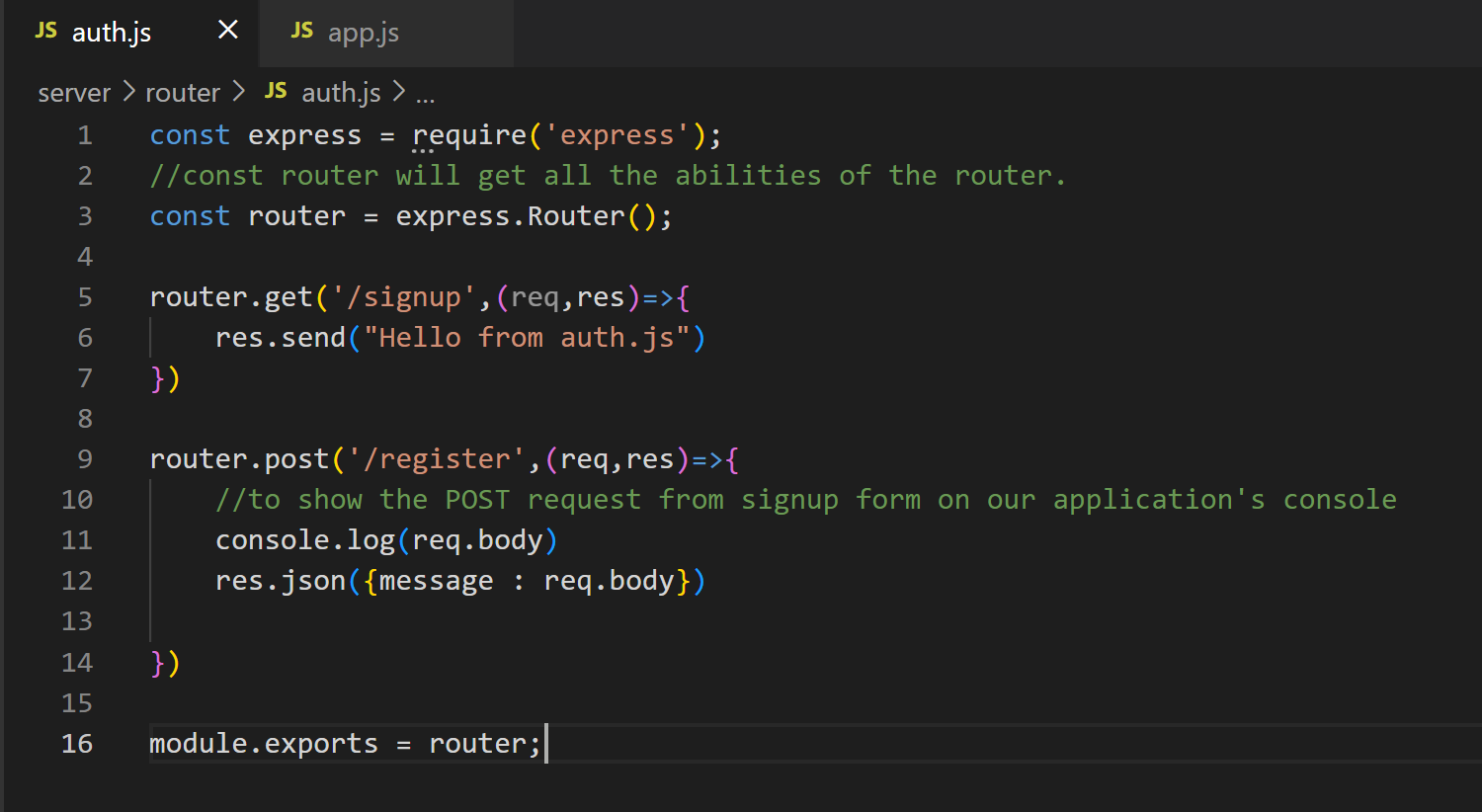
****

1. **What is POST method?**

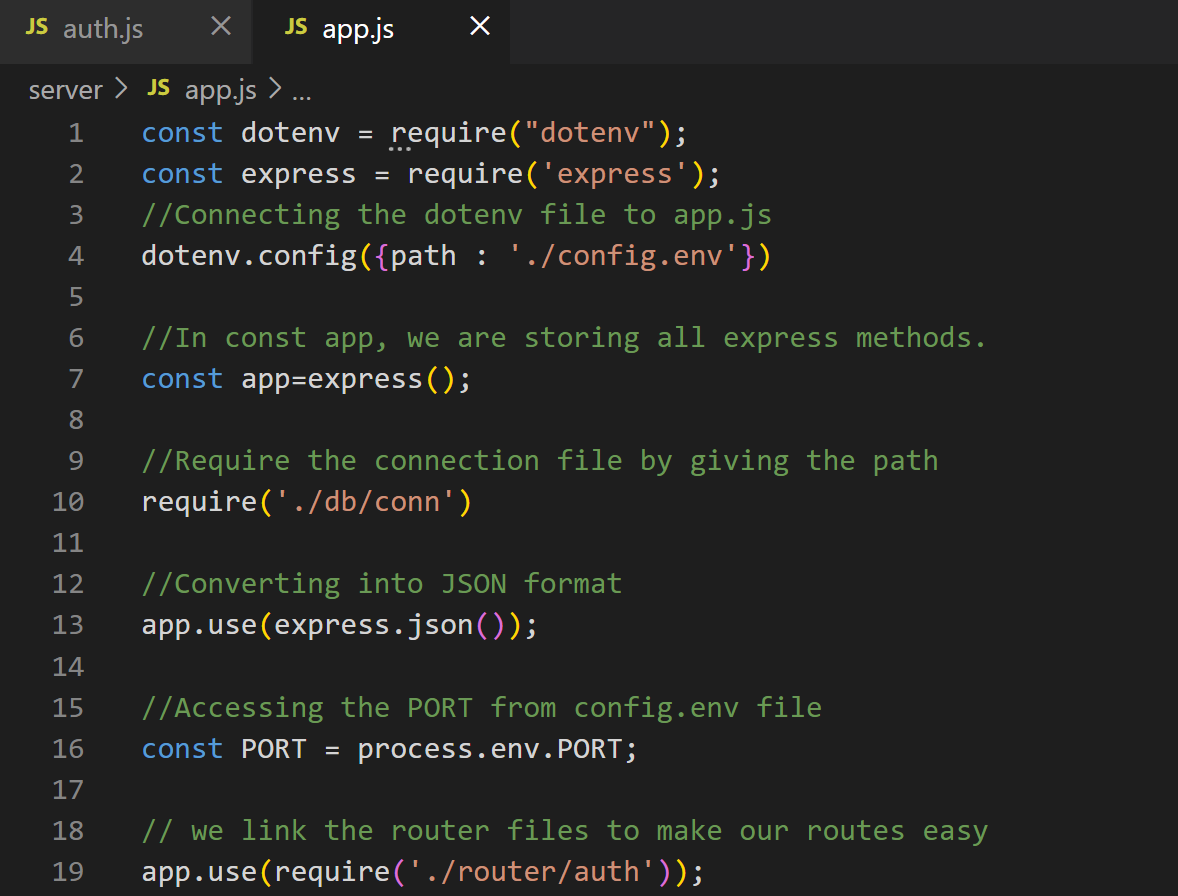
POST request submits data to a specified resource to be processed, we need to submit the data from our register form, to our MERN app. Whereas GET request, is requesting data from a specified source.

Make a POST request to the server, to accept the data, that we would normally submit by clicking the submit button in our register form.

Sending data from register form and accepting it in our MERN application. (sent through POST method).



1. Our application does not understand JSON, so we need to add **app.use(express.json());** in app.js ( so that it will convert the messages into JSON format), make sure it is right after requiring the Connection file.



1. **How to generate POST requests using Postman?**

1. Create a new collection

2. Create a new Request, select the type of request (GET/POST)

3. Enter your path in the space provided (<http://localhost:3000/register>)

4. Go to the “Headers” section,

Enter key as “Content-Type”

Enter value as “application/json”

5. Go to the “Body” section

Select “raw” format for entering the data.

6. Enter your data in JSON format

**{**

**"name" : "Shivangi",**

**"email" : "abc@gmail.com",**

**"phone" : 9911223344,**

**"work" : "web developer",**

**"password" : "shiv123",**

**"cpassword" : "shiv123"**

**}**

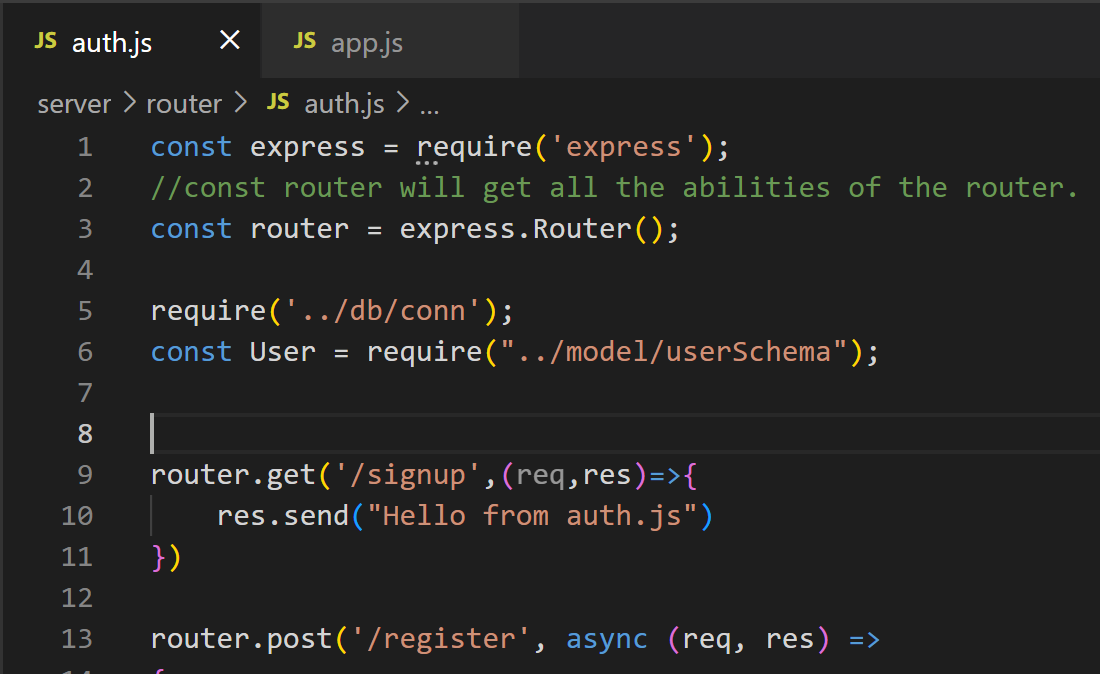
7. Click the “Send” button, to make the POST request (on the given URL)

1. Run the code from the terminal, and go to the given “[**http://localhost:3000/register**](http://localhost:3000/register)”, and send the POST request from POSTMAN. You will observe the message on the Postman Body & JSON object in the console of our MERN Application.
2. **Code to Store the Data into DB :**

**Require the connection file, and require the model & Schema into a constant.**

require('../db/conn');

const User = require("../model/userSchema");



1. Write the code to Save the data into database from Postman: (copy the code from Code Document)

**router.post('/register', async (req, res) =>**

**{**

**const { name, email, phone, work, password, cpassword} = req.body;**

**if (!name || !email || !phone || !work || !password || !cpassword)**

**{**

**return res.status(422).json({ error: "Please fill the field properly" });**

**}**

**try**

**{**

**//Check if the user exists in our database.**

**const userExist = await User.findOne({ email: email });**

**if (userExist)**

**{**

**return res.status(422).json({ error: "Email already Exist" });**

**}**

**else if (password != cpassword)**

**{**

**return res.status(422).json({ error: "passwords are not matching" });**

**}**

**else**

**{**

**const user = new User({ name, email, phone, work, password, cpassword });**

**await user.save();**

**res.status(201).json({ message: "user registered successfuly" });**

**}**

**}**

**catch (err)**

**{**

**console.log(err);**

**}**

**});**

1. **How can we secure our password? (using hashing & Salting)**

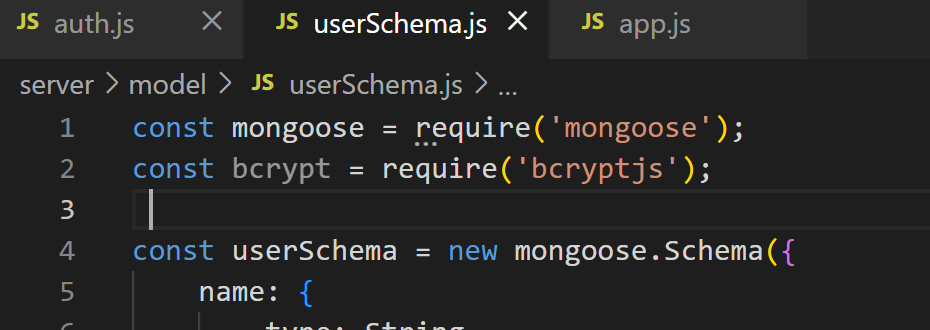
**Q. What is Password Hashing?**

Password hashing is defined as putting a password through a [hashing algorithm](https://www.okta.com/identity-101/hashing-algorithms/) (bcrypt, SHA, etc) to turn plaintext into an unintelligible(impossible to understand.) series of numbers and letters.

**Q. What is Password Salting?**

Salting is the act of adding a series of random characters to a password before going through the hashing function.

1. Install the bcryptjs package using the command on the terminal, “**npm i bcryptjs**”,check your package.json file to confirm if it’s properly installed.
2. We will use the hashing before saving the document, and after taking the input.
3. Go to the userSchema.js and first require bcrypt, using **const bcrypt = require('bcryptjs');**



1. Now after defining userSchema, and before creating the model and exporting the model, we will make a new function, which will act like a middleware function, and do all the password Hashing.

// we are hashing the password

userSchema.pre('save', async function (next)

{

if (this.isModified('password'))

{

this.password = await bcrypt.hash(this.password, 12);

this.cpassword = await bcrypt.hash(this.cpassword, 12);

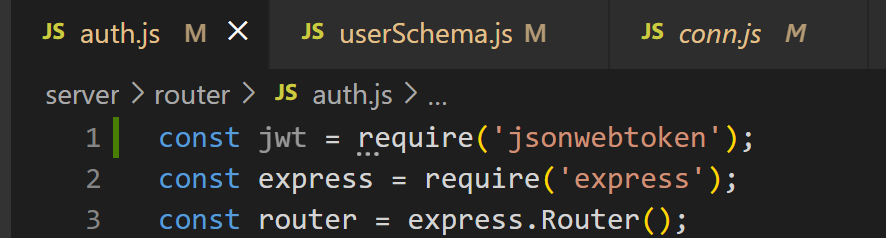
}

next();

});

1. **Authenticating users with the help of JWT (JSON Web token), that the same user that has logged in should be able to access the About me page.**
2. Install jsonwebtoken through terminal, using **npm i jsonwebtoken**

Use the command **const jwt = require('jsonwebtoken'); in both the files, userSchema.js & auth.js**

****

1. **Creating the Login Route in auth.js**

**//Login Route**

**router.post('/signin', async (req, res) =>**

**{**

**try**

**{**

**let token;**

**const { email, password } = req.body;**

**if (!email || !password)**

**{**

**return res.status(400).json({error:"Plz Filled the data"})**

**}**

**const userLogin = await User.findOne({ email: email });**

**// console.log(userLogin);**

**if (userLogin)**

**{**

**const isMatch = await bcrypt.compare(password, userLogin.password);**

**if (!isMatch)**

**{**

**res.status(400).json({ error: "Invalid Credientials " });**

**}**

**else**

**{**

**// need to genereate the token and stored cookie after the password match**

**token = await userLogin.generateAuthToken();**

**console.log(token);**

**res.cookie("jwtoken", token, {**

**expires: new Date(Date.now() + 25892000000),**

**httpOnly:true**

**});**

**res.json({ message: "user Signin Successfully" });**

**}**

**}**

**else**

**{**

**res.status(400).json({ error: "Invalid Credientials " });**

**}**

**}**

**catch (err)**

**{**

**console.log(err);**

**}**

**});**

1. Adding a function in userSchema.js for generating tokens :

// we are generating token

userSchema.methods.generateAuthToken = async function () {

try {

let token = jwt.sign({ \_id: this.\_id }, process.env.SECRET\_KEY);

this.tokens = this.tokens.concat({ token: token });

await this.save();

return token;

} catch (err) {

console.log(err);

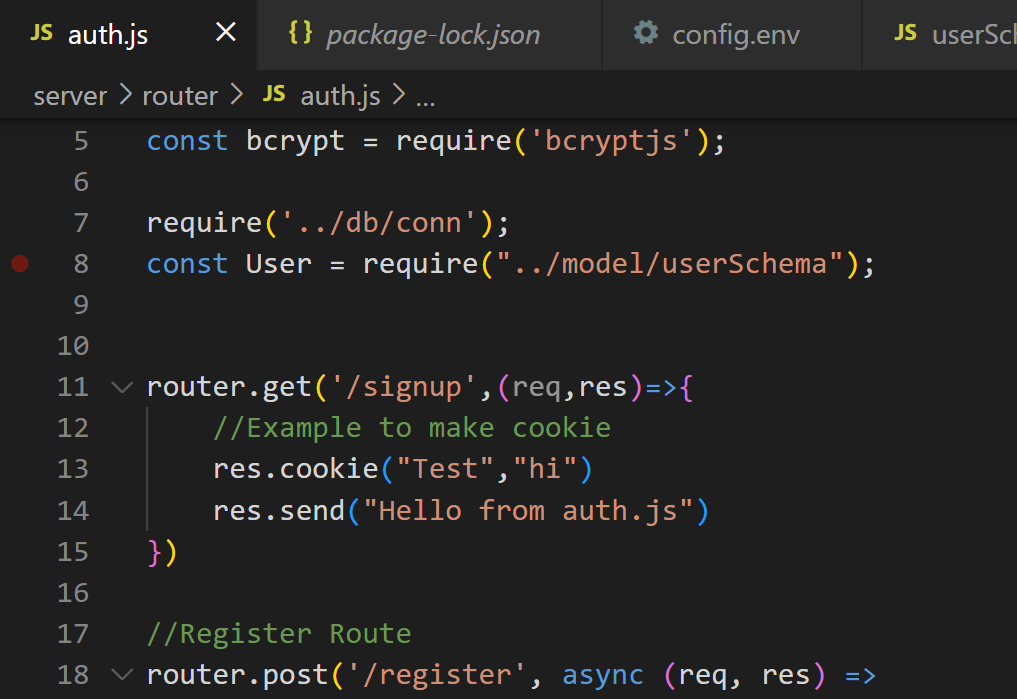
}

}

1. In config.env , enter the value of your secret key with the help of the following code:

**SECRET\_KEY=SHIVANGI1234567SHIVANGI123456789 (Any 32 Characters)**

1. **Now if we enter the login details (Email & password) of the registered user, User signin successful.**
2. **How to generate your own cookie?**

****

You can check in the browser as well.

To check the cookie that we created, we will see it in POSTMAN under cookies section.

**What is REST API?**

A REST API is an application programming interface that adheres to the constraints of REST architectural style and enables interaction with RESTful web services. Interconnected networks make up the web. A web service is a set of open protocols and standards used for exchanging data between client-server applications

**POST**

In web services, POST requests are used to send data to the API server to create or update a resource. The data sent to the server is stored in the [request body](https://stackoverflow.com/questions/22034144/what-does-it-mean-http-request-body) of the HTTP request.

that data is put in the **response body** of the request and sent to the server.

**Understanding Express Router.**

Go to postman, make a collection, add request, select type as POST, go to headers -> add key as “Content-Type” and set value as application/json

Go to body and then select “RAW” option , check if it’s in JSON

Getting Data Back From POSTMAN to Our MERN Application

We will get the data that we sent in the form (basically by post method through postman), and we will see it on console.log in our MERN Application.

After getting the postman, we will learn how to store the data in the database.

**STORING DATA IN DATABASE :**

Command to use is your port Number is occupied :

To stop the occupation of the port, which is already in use, run cmd as administrator, type this on cmd.

**taskkill /f /im node.exe**

Successfully wrote code to save data in database, directly, using the async await function

**STARTING WITH FRONT END: (With react) :-**

React is a JS library for building user interfaces.

1. How to make a react project:
2. Create a folder named “**client**”, go to the folder, using the “**cd client”** command and build a react project, by running the command “**npx create-react-app client**”.
3. You can create a React application by using create-react-app. Create React App doesn't handle backend logic or databases; it just creates a frontend build pipeline, so you can use it with any backend you want.
4. Now we need to start our react app by using the command “**npm start**”, because “**yarn start**” has been deprecated.
5. Download the extension in VSC called “es7 react”(ES7 React/Redux/React-Native/JS snippets) to write the shortcuts.
6. In the file “App.js”, remove all the existing contents & Write “rafce” (react arrow function component with export, to give u a default code. Then just write

<div>

<h1>

Hello World from Shivangi

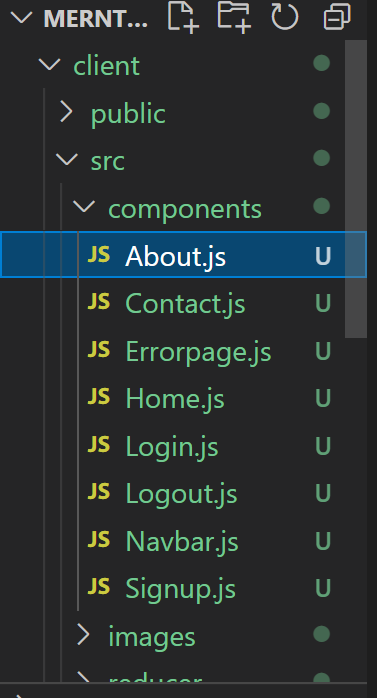
</h1>

</div>

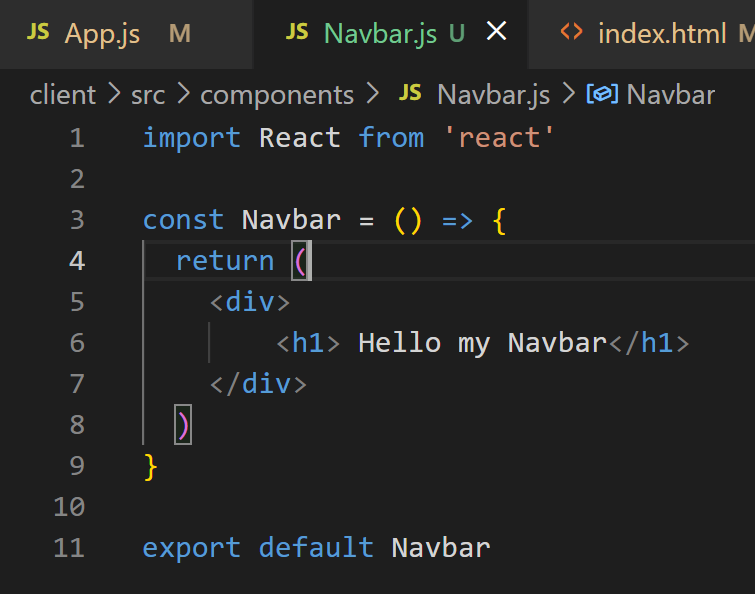
To check the output, in cmd, navigate to the client folder, using “cd client”, then run the command “npm start” , you will be able to see “Hello world from Shivangi” as output

**Creating a navbar( Navigation bar):**

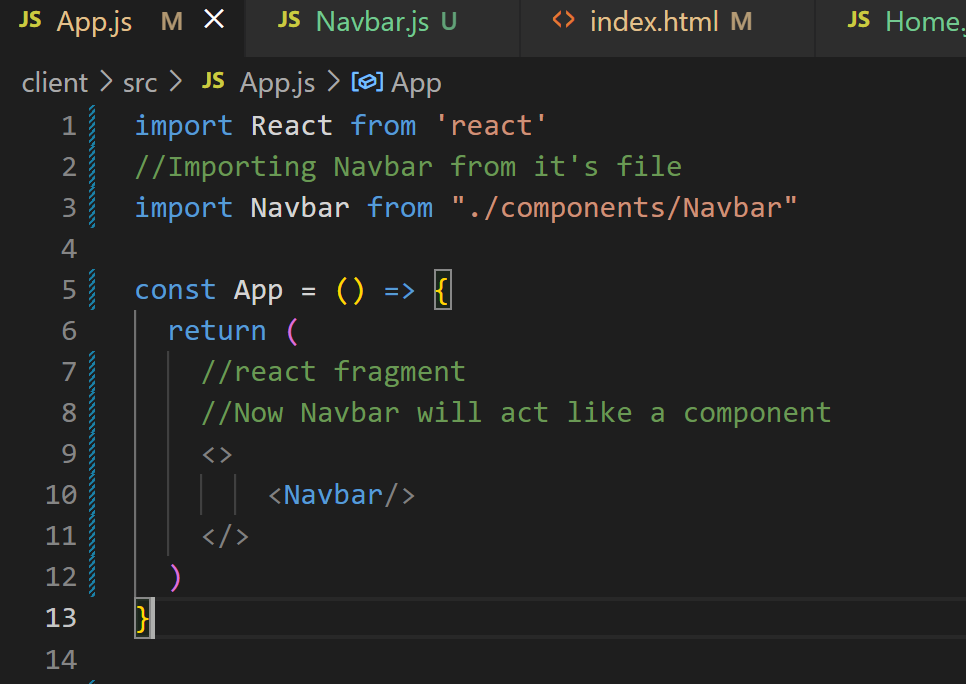
1. Install bootstrap first, by running the “**npm i bootstrap**” command inside the client folder.
2. <https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp>
3. Visit this website and copy the entire dependency contents & go to the folder client/public/index.html and paste it just before the ending </body> tag.
4. Now create a new folder called “**components**” inside client/src folder, now inside the components folders we will create JS files called : Navbar.js, Home.js, Login.js, Contact.js, Signup.js, About.js



1. Use the command , “**rafce**”, to create basic components in all the files
2. In Navbar.js, print a normal message.

.

1. Import it inside the “App.js” file, using the command : “**import Navbar from ./components/Navbar**” (In Line2)
2. With the help of react fragment(a component to return multiple elements), we will use Navbar as a self closing component.



1. Go to localhost output, and we should see the message written inside Navbar.
2. Bootstrap : Bootstrap is the most popular CSS Framework for developing responsive and mobile-first websites.
3. <https://getbootstrap.com/docs/4.0/components/navbar/>
4. Copy the entire navbar bootstrap code, from the above link and paste it inside **Navbar.js**, by changing the div tag into fragment, and removing the h1 tag , and paste your entire copied code.
5. There will be a red error, because React would work with JSX, which is like javascript but it is more powerful and it has some rules, like every opening bracket should have a closing bracket.
6. In our code, the input tag is not closing, so we will close it by putting a backward / slash. The error will go away.
7. Check the output.
8. Go to pkg,json , and check, now we will use the import command, to include the bootstrap, that is added to our package.json. Use the command “**import 'bootstrap/dist/css/bootstrap.css**'” in **navbar.js** (It will give a path that is inside our node\_modules.)
9. Check the output, we get the default navbar
10. UL means unordered list, LI means items inside the list.
11. We remove the unnecessary components, and copy-paste the **<li class="nav-item active">** tag multiple times, and change their names to Home, About, Contact, Login, Registration.
12. We change the ul class to **<ul class="navbar-nav ml-auto">** ml - margin left (we want margin from the left side)
13. Go to **Navbar.js** and import NavLink component using the command **import { NavLink } from 'react-router-dom'**,
14. Replace all (because in react class is a reserved keyword) : (Navbar.js)
    1. anchor tags into NavLink (**import { NavLink } from "react-router-dom"**;)
    2. class into className
    3. href into to

**CODE : (navbar.js)**

import React from 'react'

import 'bootstrap/dist/css/bootstrap.css'

import { NavLink } from "react-router-dom";

const Navbar = () => {

return (

<>

<nav className="navbar navbar-expand-lg navbar-light bg-light">

<NavLink className="navbar-brand" to="#">Navbar</NavLink>

<button className="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span className="navbar-toggler-icon"></span>

</button>

<div className="collapse navbar-collapse" id="navbarSupportedContent">

<ul className="navbar-nav ml-auto">

<li className="nav-item active">

<NavLink className="nav-link" to="/">Home <span className="sr-only">(current)</span></NavLink>

</li>

<li className="nav-item">

<NavLink className="nav-link" to="/about">About</NavLink>

</li>

<li className="nav-item">

<NavLink className="nav-link" to="/contact">Contact</NavLink>

</li>

<li className="nav-item">

<NavLink className="nav-link" to="/login"> Login</NavLink>

</li>

<li className="nav-item">

<NavLink className="nav-link" to="/signup">Registration</NavLink>

</li>

</ul>

</div>

</nav>

</>

)

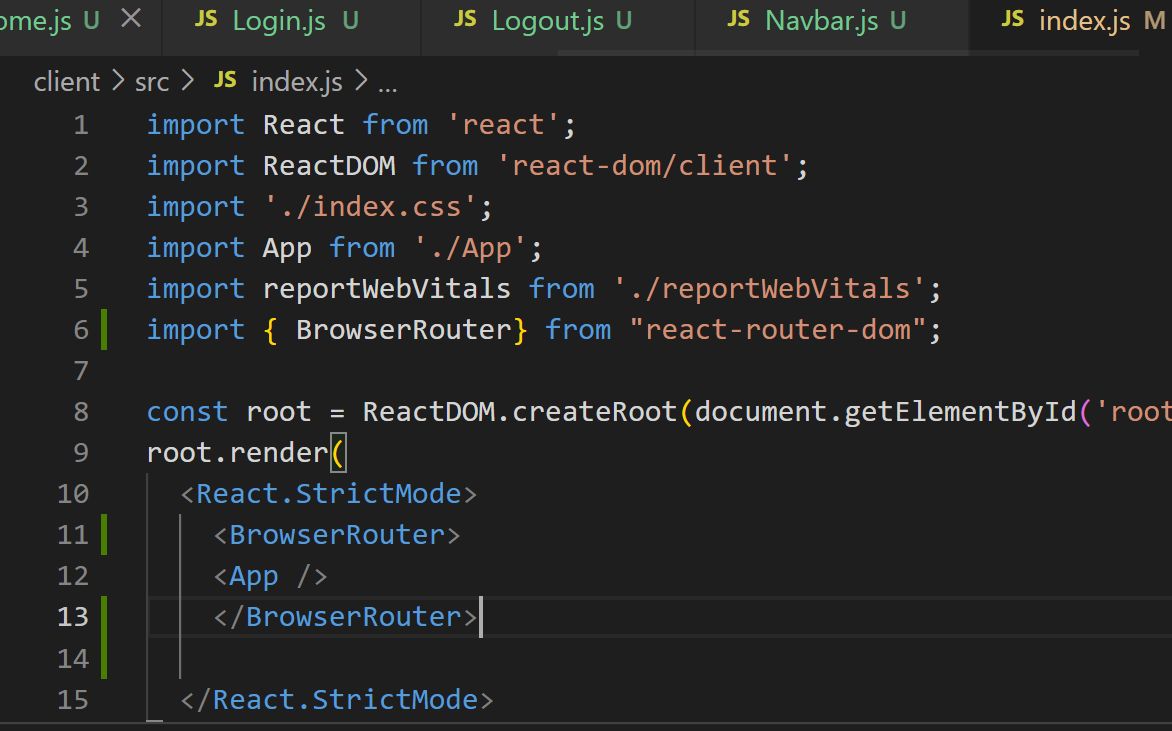
}

export default Navbar

**SINGLE PAGE APPLICATION, How to open the new page without reloading/refreshing with REACT-ROUTER-DOM**

1. In terminal of client, we need to install react-router-dom using “**npm i react-router-dom**”
2. Run the code, using “**npm start**” in the terminal (make sure you are in the client folder/directory).
3. Go to **index.js** file, and use **import { BrowserRouter} from "react-router-dom**";
4. In **index.js** enclose the </App> tag inside opening and closing **<BrowserRouter>**

**IMAGE : (index,html)**



**CODE: (index.html)**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { BrowserRouter} from "react-router-dom";

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

1. Go to App.js file and import Routes & Routes using **import { Routes, Route} from 'react-router-dom' , import all components Navbar, Home, Aboutus, Contact, Login, Signup from respective path.**

**IMAGE :**

1. Go to App.js, will specify the routes and open respective pages using the following code :

**CODE : (App.js)**

**import React from 'react'**

**import { Routes, Route,Switch} from 'react-router-dom'**

**import 'bootstrap/dist/css/bootstrap.css';**

**import './App.css'**

**import Navbar from './components/Navbar'**

**import Home from './components/Home'**

**import About from './components/About'**

**import Contact from './components/Contact'**

**import Login from './components/Login'**

**import Signup from './components/Signup'**

**import ErrorPage from './components/Errorpage'**

**import Logout from './components/Logout'**

**const App = () => {**

**return (**

**<>**

**<Navbar/>**

**<Routes>**

**<Route exact path="/" element={<Home/>} />**

**<Route path="/about" element={<About/>} />**

**<Route path="/contact" element={<Contact/>} />**

**<Route path="/login" element={<Login/>} />**

**<Route path="/signup" element={<Signup/>} />**

**<Route path="/logout" element={<Logout/>} />**

**</Routes>**

**</>**

**)**

**}**

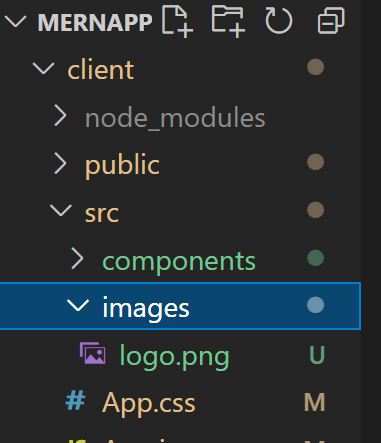
**export default App**

1. **import './App.css'** add this import statement in App.js file to import the css file
2. Our Navbar is working perfectly now.

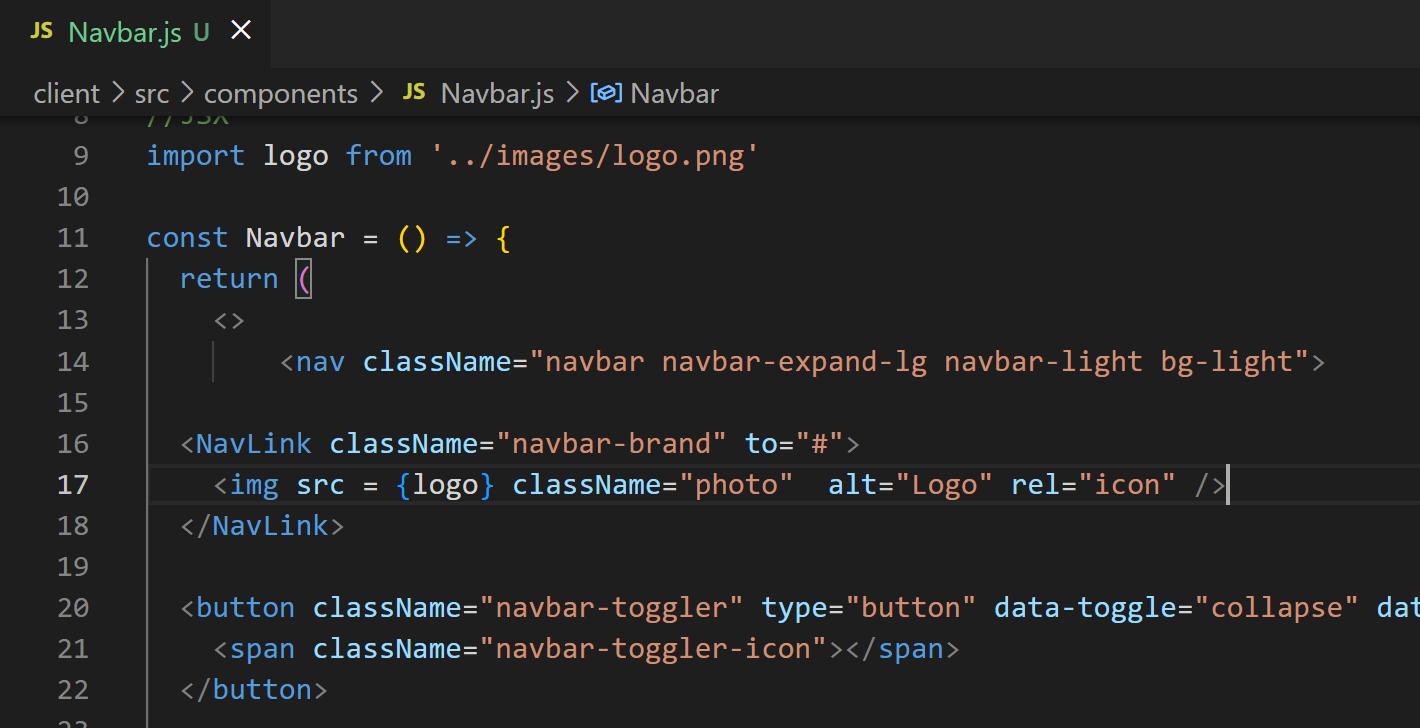
**How to add images in our Project :** .

Adding a logo in our Navbar.

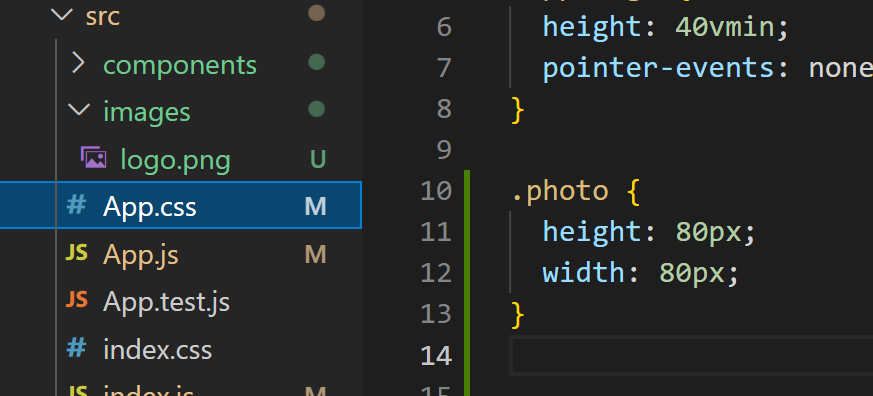
1. Copy paste your images folder inside src folder.



1. Go to Navbar.js , import the logo using the following command **import logo from '../images/logo.png'**
2. Use the img tag inside the NavLink, and give appropriate className, which should also be added in your css file. Alternate code for giving width and height : **<img src='imagepath.jpg' width={250} height={250} alt='Large Pizza' />**

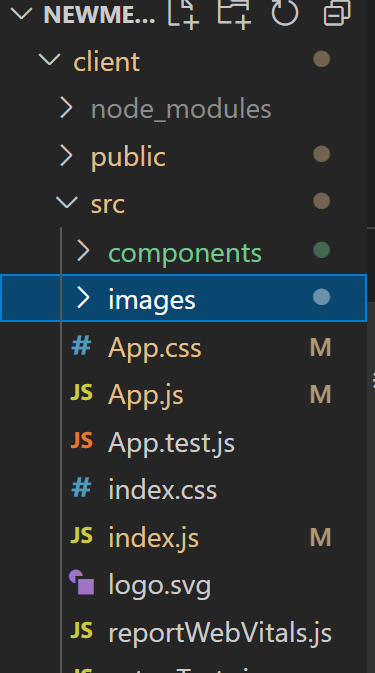


1. Make sure that the className is defined in your App.css file.

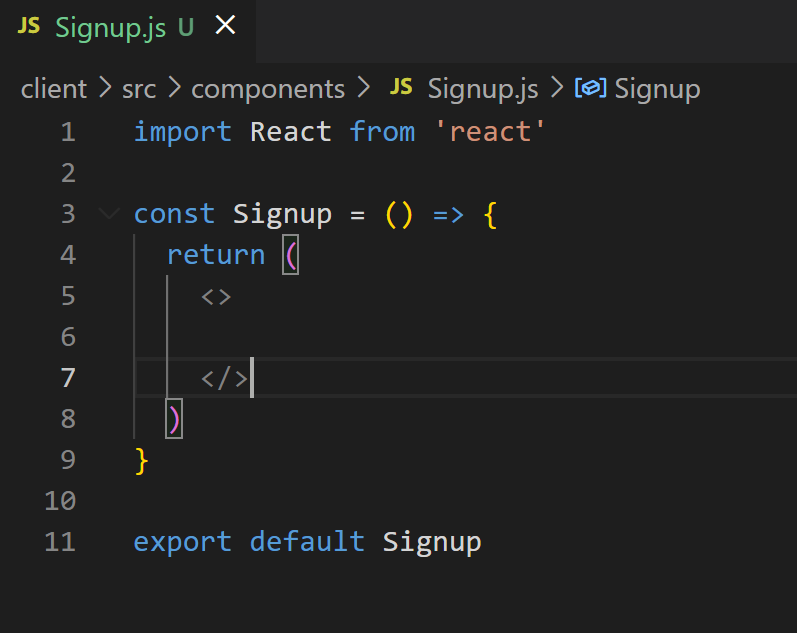


**Creating Registration Form/Signin in our project :**

1. Add the images folder in the src folder.



1. Go to signup.js, remove the div tab, and change it into fragment.



Write the UI code, don’t forget to add the images folder.

1. <https://zavoloklom.github.io/material-design-iconic-font/>

Visit the above given link to add icons into your project,

Make sure you add it in the <head> tag of your index.html file.

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/material-design-iconic-font/2.2.0/css/material-design-iconic-font.min.css">

**How To Store User Registration Form Data in Front-End React State**

1. **What are Hooks in React?**

Hooks are functions that let you “hook into” React state and lifecycle features from function components.

What is useState Hook?

const [state, setState] = useState(initialstate)

The first element is the initial state and the second one is a function that is used for updating the state.

1. Spread operator (...)

The JavaScript spread operator ( ... ) allows us to quickly copy all or part of an existing array or object into another array or object.

**STORING USER REGISTRATION FORM IN FRONT END : (Using hooks)**

**import React,{useState} from 'react'**

**(Inside Signup)**

const [user, setUser] = useState({

name: "", email: "", phone: "", work: "", password: "", cpassword: ""

});

let name, value;

const handleInputs = (e) => {

console.log(e);

name = e.target.name;

value = e.target.value;

setUser({...user, [name]:value});

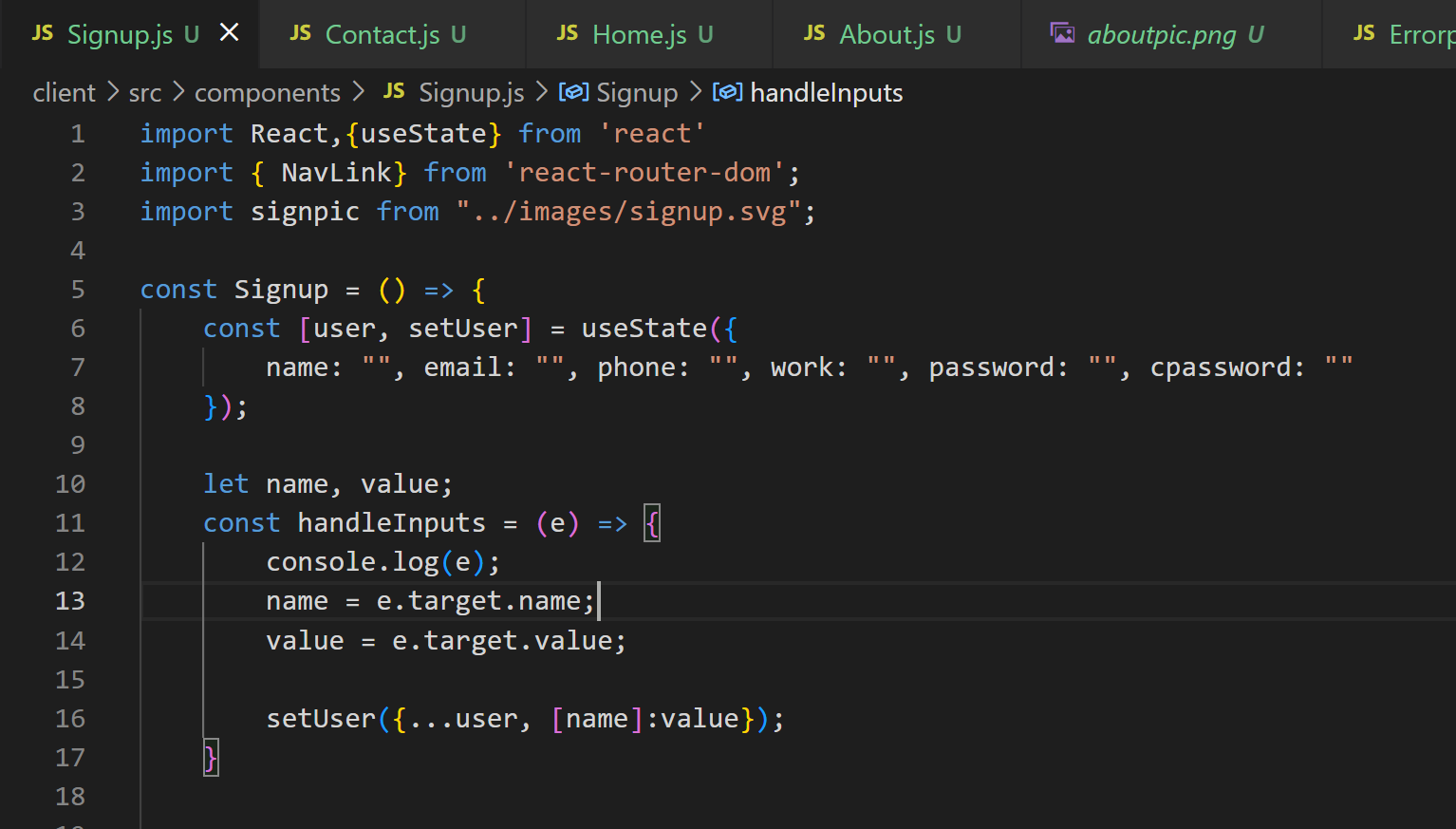
}

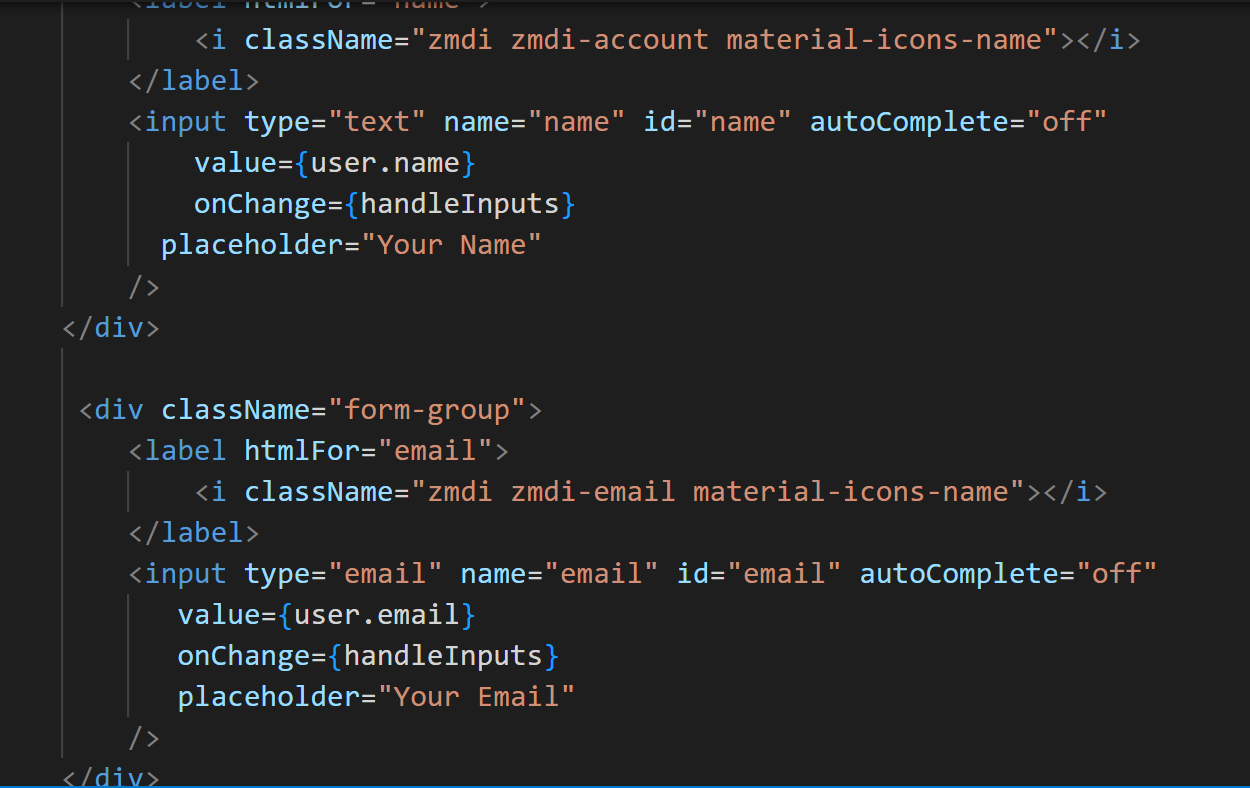
Here, the initial value of the user, will go into useState. (which is empty initially), and setUser, will be the method which will be called when values are updated.

In input tags, we will add the value and onChange attributes ,

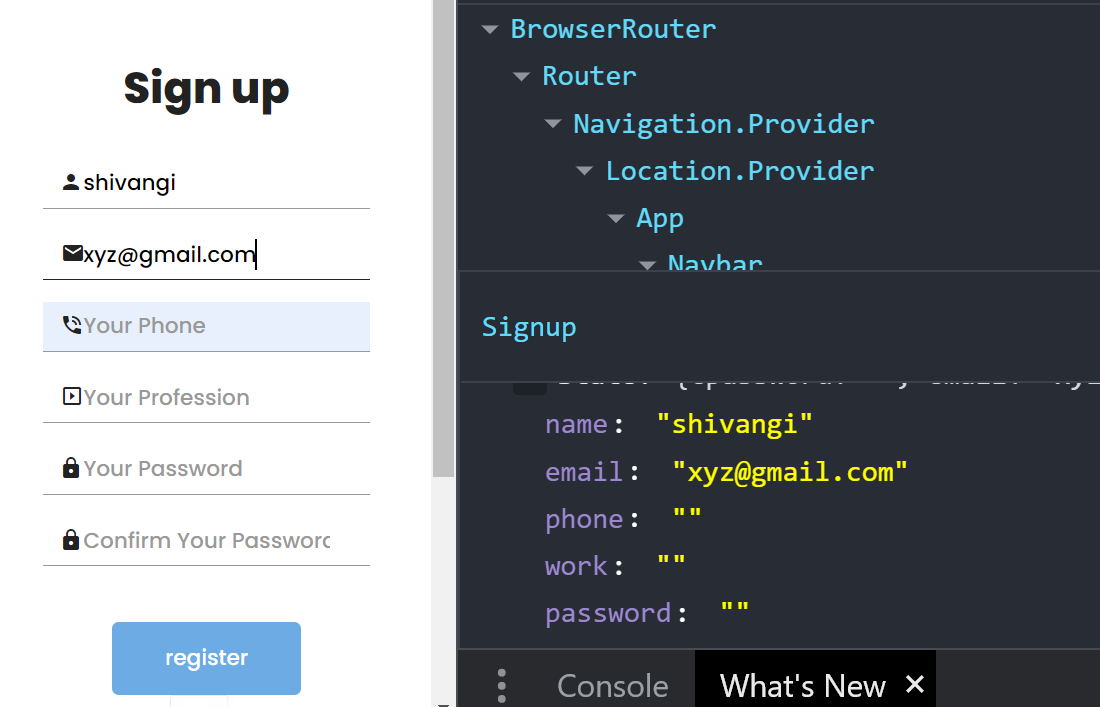
value={user.name}

onChange={handleInputs}





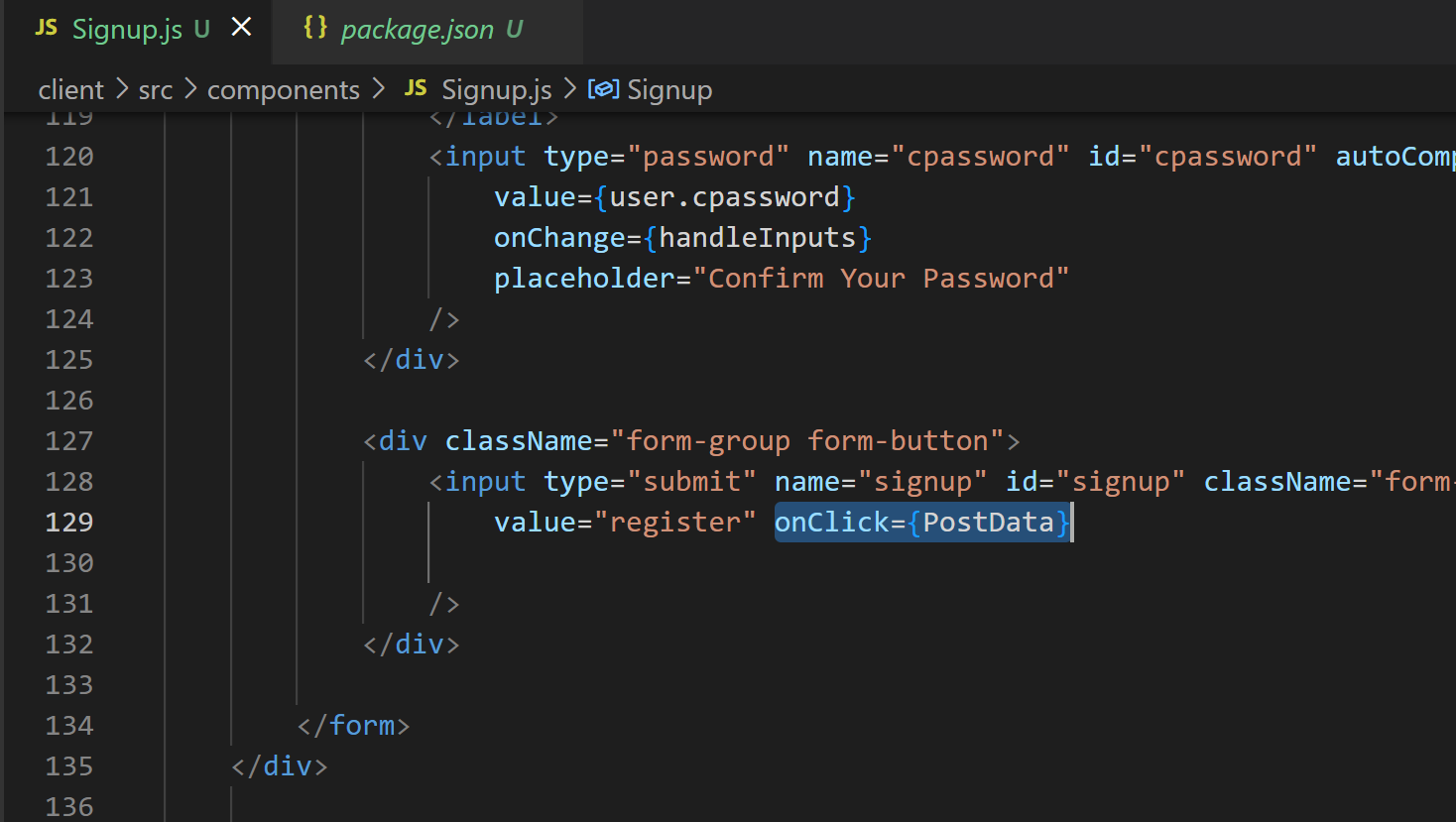
**Check it on the console, in the components section.**



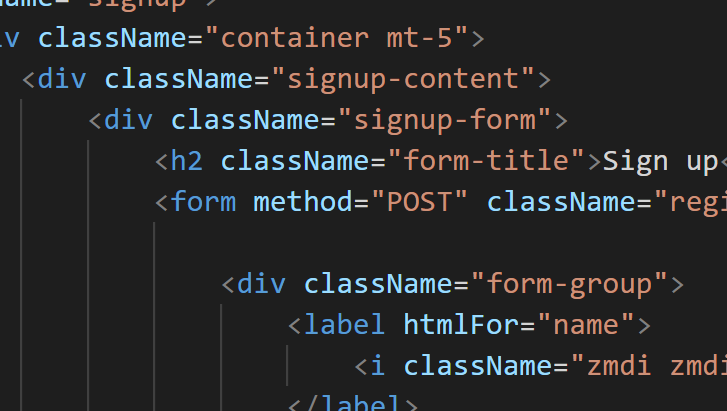
**Storing data inside database through Application :-**

**Go to Signup.js**

Add **onClick={PostData}**

****

Write form **method=”POST”**

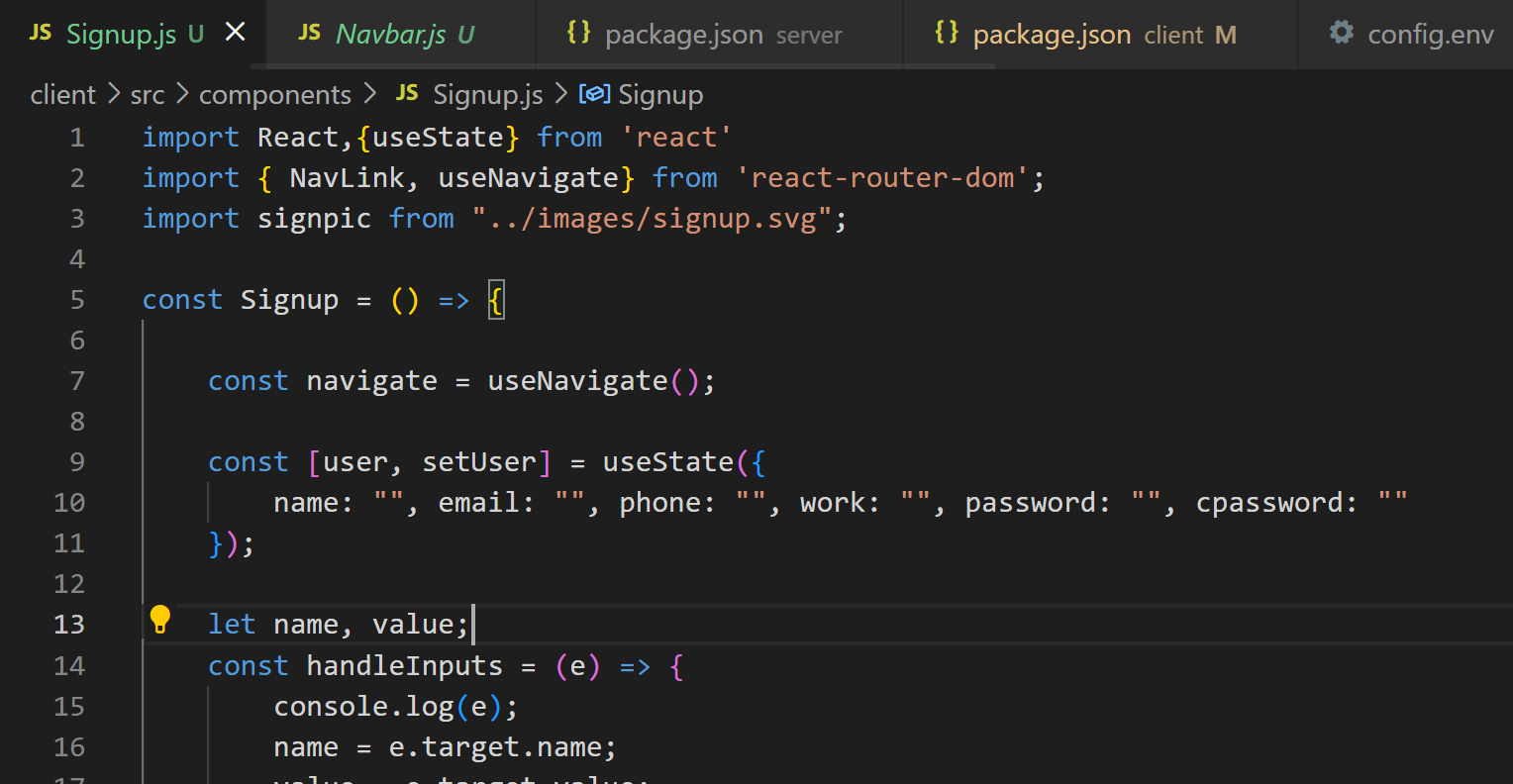
****

**Now we just need to store the data from the state, into database, using fetch API.**

**FETCH API:-**

The Fetch API through the fetch() method allows us to make an HTTP request to the backend. With this method, we can perform different types of operations using HTTP methods like the GET method to request data from an endpoint, POST to send data to an endpoint, and more.

**Overall picture :(signup.js)**



Import useNavigate to open the login page after registration :

import { NavLink, useNavigate} from 'react-router-dom';

Make a constant to store it’s methods inside it :

const navigate = useNavigate();

**Make a function called PostData :-**

**//PostData Function**

**//It will post all our data to database through fetch API**

**const PostData = async (e) => {**

**e.preventDefault();**

**const { name, email, phone, work, password, cpassword } = user;**

**const res = await fetch("/register", {**

**method: "POST",**

**headers: {**

**"Content-Type": "application/json"**

**},**

**body: JSON.stringify({**

**name, email, phone, work, password, cpassword**

**})**

**});**

**const data = await res.json();**

**// I need to change the data to res**

**if (data.status === 422 || !data) {**

**window.alert("INvalid Registration");**

**console.log("INvalid Registration");**

**} else {**

**window.alert(" Registration Successfull");**

**console.log("Successfull Registration");**

**//history.push("/login");**

**navigate('/login');**

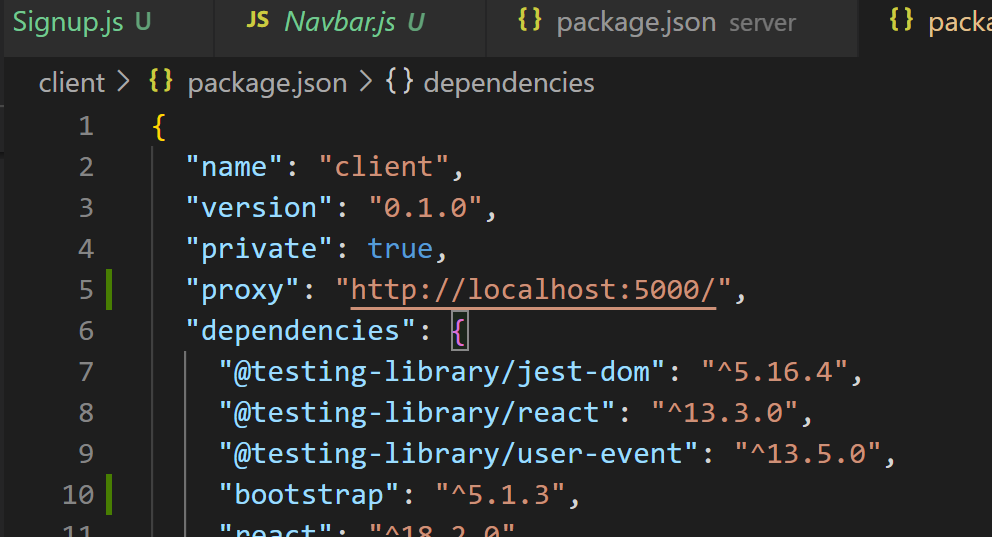
**}**

**}**

**Set proxy in package.json (on client side), on your port number , using the following code.(below private : true)**

**"proxy": "**[**http://localhost:5000/**](http://localhost:5000/)**",**

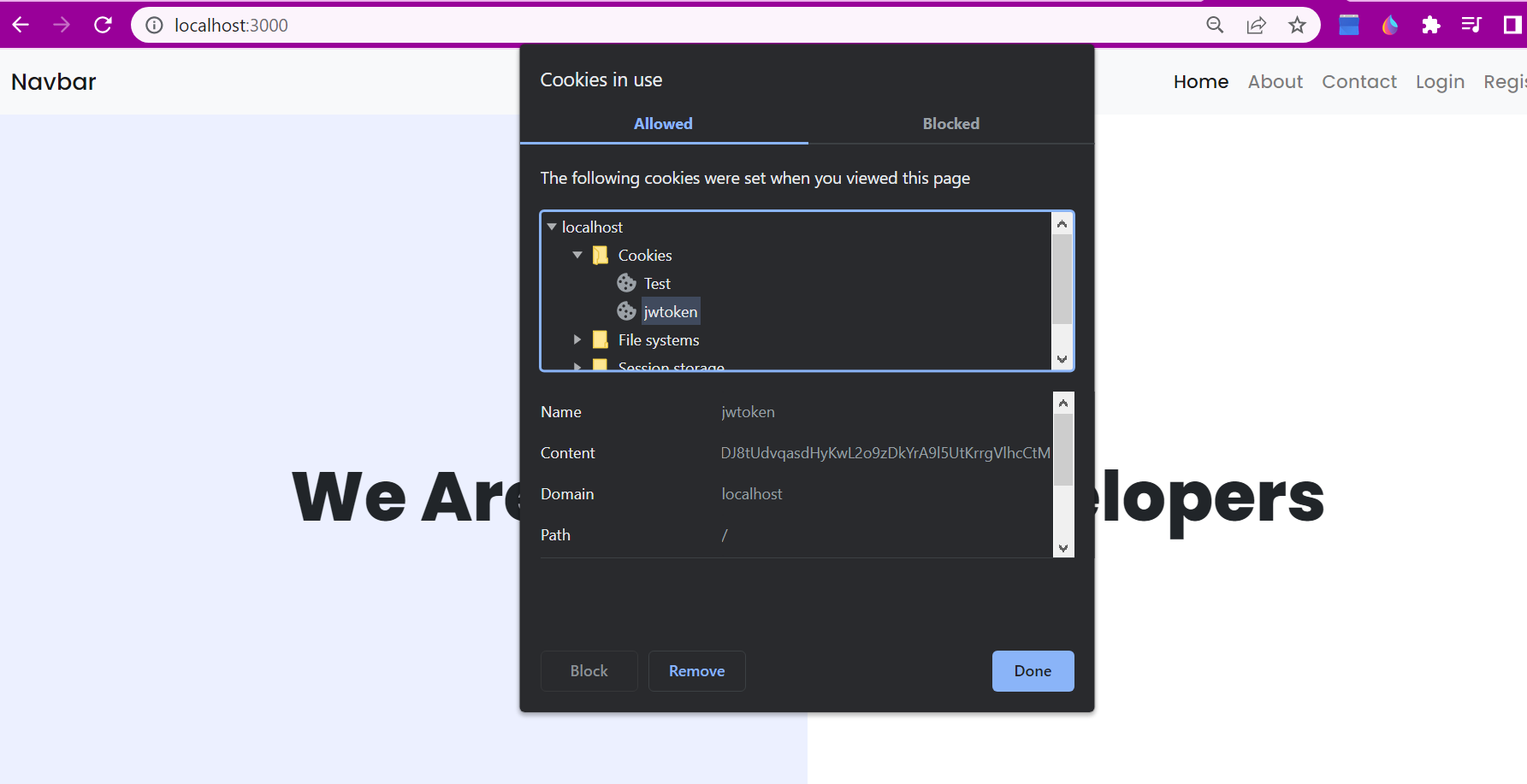
**IMAGE**

****

**Run the code, and enter details in the form and hit the submit button, and check the values in the database.**

**LOGIN PAGE : (similar to signup, so just copy paste)**

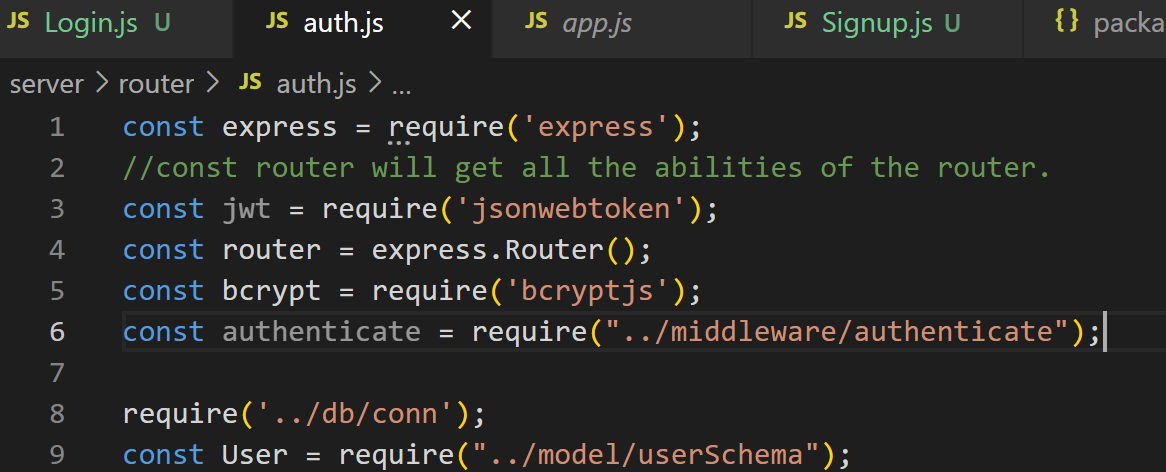
**Cookies after logging the user in and redirecting them to the home page**

****

**CREATING AUTHENTICATION MIDDLEWARE for about us page**

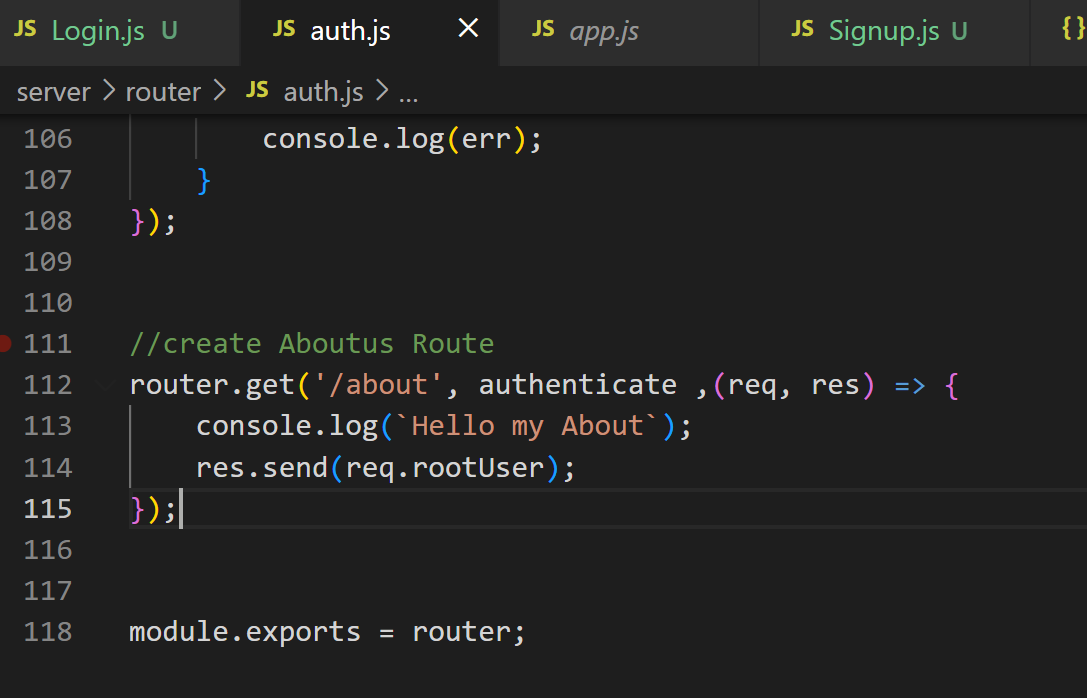
1. Go to server/router/auth.js

**const authenticate = require("../middleware/authenticate");**

****

**2. Create aboutus Route, under the Login route.**

**IMAGE : (server/router/auth.js)**

****

**CODE:**

**//create Aboutus Route**

**router.get('/about', authenticate ,(req, res) => {**

**console.log(`Hello my About`);**

**res.send(req.rootUser);**

**});**

**3. Go to server/middleware/authenticate.js**

**CODE : (server/middleware/authenticate.js)**

**const jwt = require("jsonwebtoken");**

**const User = require("../model/userSchema");**

**const Authenticate = async (req, res, next) => {**

**try {**

**const token = req.cookies.jwtoken;**

**const verifyToken = jwt.verify(token, process.env.SECRET\_KEY);**

**const rootUser = await User.findOne({ \_id: verifyToken.\_id, "tokens.token": token });**

**if (!rootUser) { throw new Error('User not Found') }**

**req.token = token;**

**req.rootUser = rootUser;**

**req.userID = rootUser.\_id;**

**next();**

**} catch (err) {**

**res.status(401).send('Unauthorized:No token provided');**

**console.log(err);**

**}**

**}**

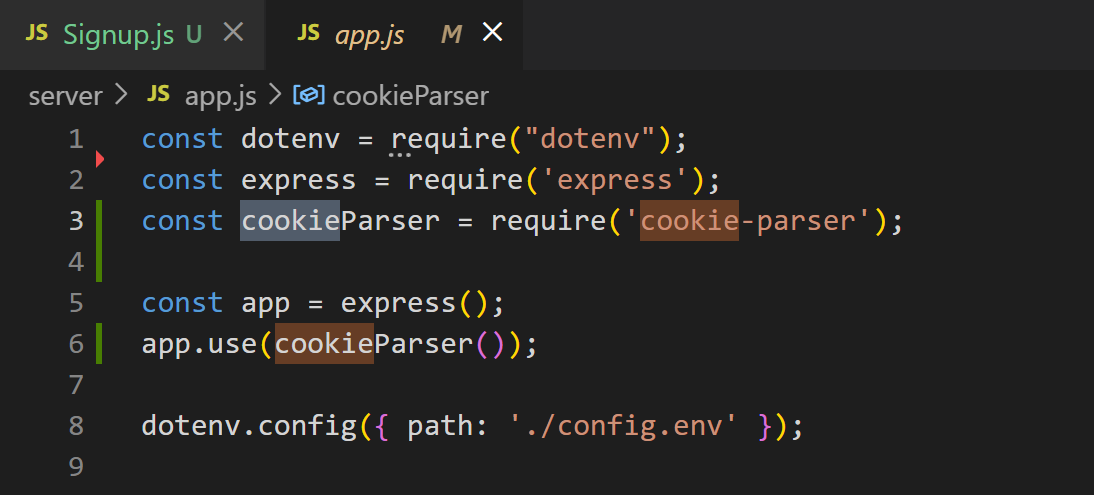
**module.exports = Authenticate;**

**4. Install cookie parser on server folder, using the code**

npm i cookie-parser

Check whether it’s installed or not in the package.json file(of server)

**IMAGE :**



**CODE:**

**const cookieParser = require('cookie-parser');**

**app.use(cookieParser());**