React and Node.js Notes

1. React.js
2. Node.js
3. SQL & NoSQL database

Pre-requisites

1. HTML
2. CSS
3. Javascript (ES6 & latest)

Topics to be covered

1. HTML - basic HTML elements, Tables, Lists, Forms
2. CSS - classes, id, types of CSS
3. Javascript - basics to intermediate
4. React.js
5. Node.js
6. SQL & NoSQL

HTML: Hyper Text Markup Language, it is mainly used to display the content on the web page using HTML elements

CSS: Cascading Style Sheet, it is mainly used to style the HTML

JS: Javascript, it is mainly used to add interactivity to the web pages like dynamically changing the content, updating the color and so on.

Software requirement - for React.js & Node.js including HTML, CSS & Javascript

1. VS Code - Editor
2. Browser - Google Chrome
3. Node.js - node & npm commands
4. Permission to download the libraries from the internet

HTML:

It provides HTML elements like html, head, body, div, p, table, and etc.

Heading tags: These are used to display the headings in the web page, there are totally 6 heading tags like h1, h2, h3, h4, h5 and h6

<h1>Some Content</h1>

<h2>Some Content</h2>

Image tag: <img src = “url” width = “100p” height = “100” />

Anchor tag: It is used to create hyper links

<a href = “url”>Link Name</a>

Lists: These are the contents which are displayed in the form of lists either in the order or bullets, there are two types

1. ordered list <ol>
2. unordered list <ul>

both of these tags can display the items using <li>

<ol>

<li>Apples</li>  
 <li>Grapes</li>

</ol>

Tables: When you want to display the contents in a table form you can use <table> tag, along with <tr>, <th> & <td>

<tr>: It creates row

<th>: It creates columns in a bold font

<td>: It creates columns in a normal font

<table border = “1”>  
 <thead>  
 <tr>  
 <th>Heading1</th><th>Heading2</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr>  
 <td>Content1</td><td>Content2</td>  
 </tr>  
 <tr>  
 <td>Content1</td><td>Content2</td>  
  
 </tr>  
 </tbody>  
</table>

Forms:

When user wants to provide the inputs you can use the form

<form>  
 Enter Name <input type = ‘text’ /> <br />  
 Enter Age <input type = ‘number’ /> <br />  
 Select DOB <input type = ‘date’ /> <br />  
 <input type = ‘submit’ value = ‘Submit’ />  
 <input type = ‘reset’ value = ‘Reset’ />  
</form>

div:

It is a container tag, that can wrap multiple HTML elements together, so that you can style the container which applies to all the children’s of the div

CSS: Cascading Style Sheet

3 types of CSS

1. inline css: You can apply the styles to a particular element
2. internal css: You can apply to the entire HTML document
3. external css: You can apply style to multiple HTML documents, by creating a css file & referencing that css file

How to comment & uncomment: Highlight the lines and press Control and / together

3rd party css

bootstrap.css is one of the widely used external css, it provides 10000’s of inbuilt classes like .btn-primary, alert-danger, alert-primary, text-success, alert-success, container, row, col, table, table-striped and so on

Javascript:

It is a programming language used to provide effects to the HTML, it allows you to dynamically access HTML & CSS so that you can modify the DOM (Document Object Model) which is a tree structure formed when the HTML loaded on the browser

It provides many fundamentals like

* variables
* functions
* event handling
* objects
* arrays

Example on how to access and update HTML



let & const keyword: It is a new keyword introduced in ES6(new feature of Javascript), ES stands for ECMA Script, let & const are the keywords you can use to declare variables, you should avoid using var keyword, because it doesn’t have a scope.

let variables can be modified, however const variables can’t be modified.

How to use arrays in Javascript

Array is a container to store multiple values, in Javascript you use [] to create an array.

let numbers = [20, 30, 10, 50, 40];

let fruits = [“Apple”, “Mango”, “Grapes”];

How to use objects

let employee = {id:100, name:”Rajesh”, salary:35000};

Array of Objects

let employeeArray = [{id:200, name:”Raj”}, {id:300, name:”Vijay”}, {id:400,name:”Ajay”}];

Array iterations

1. It can be done using for loop.
2. It can be done using an inbuilt forEach function, that does the internal iteration
3. It can be done using an inbuilt map function, that is similar to forEach, but it returns a new array after iterating the elements, which can be used to transform the iterated elements.

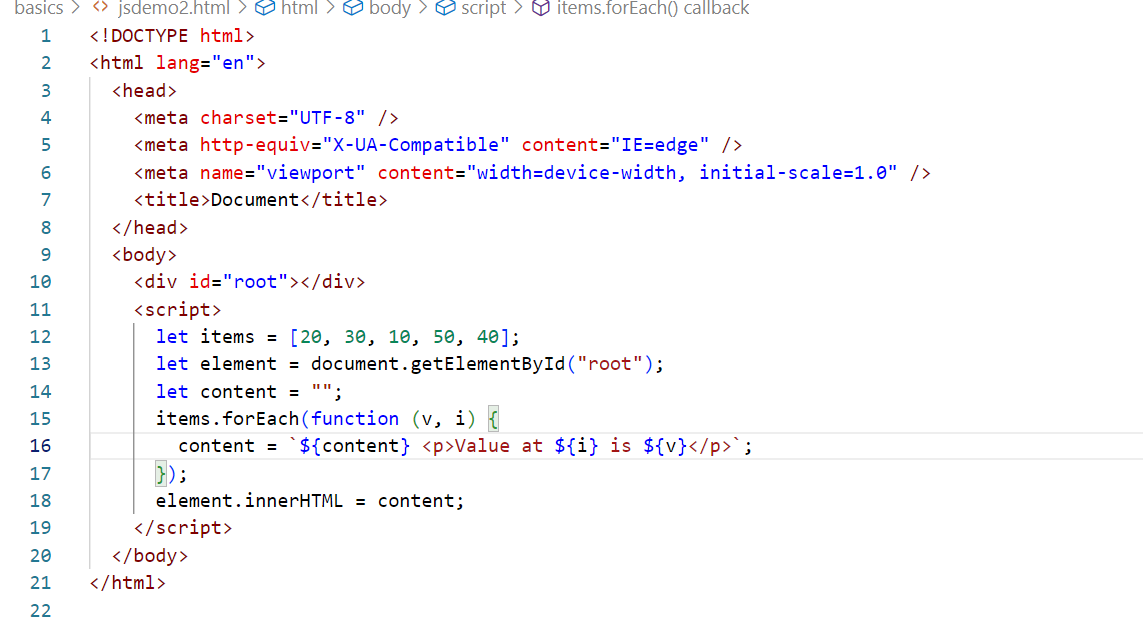
let items = [20, 30, 40, 10, 50];

items.forEach(callbackFn);

callbackFn: it is a callback function, takes 2 arguments which is an element that is iterated and the index of the iterated element.

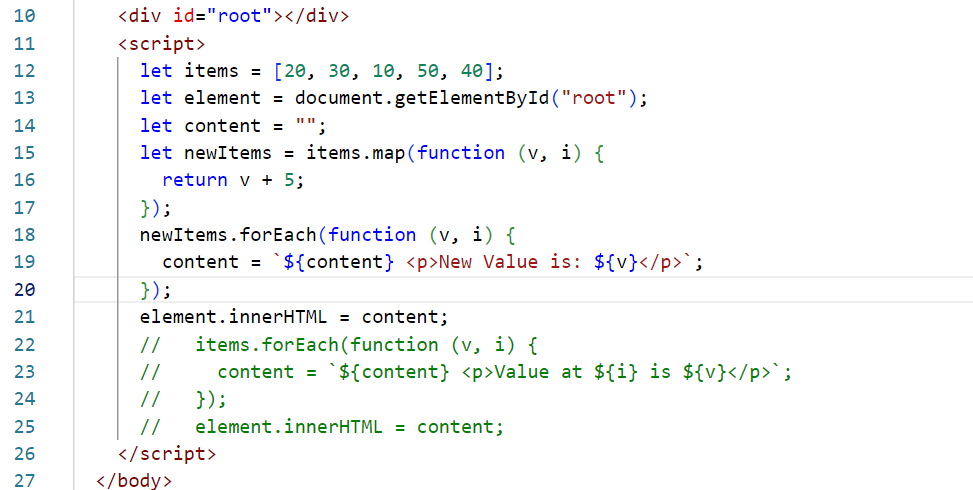
items.forEach( function(v, i) {  
 v : value of the iterated element  
 i : index of the iterated element  
});

Array iteration using forEach



Map: It is an inbuilt function to iterate an array & return each element into another array after the transformation

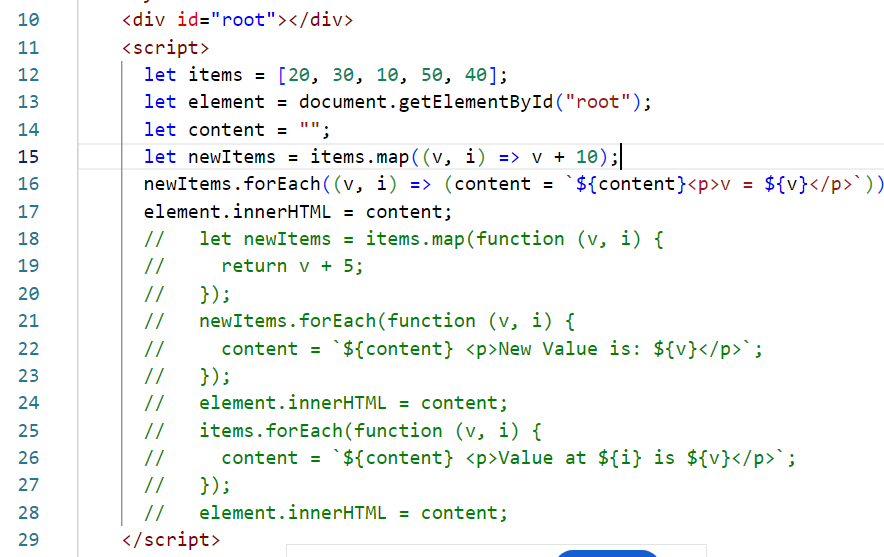
let newItems = items.map(function(value, index) { } );



arrow functions

ES6 introduced arrow functions to simplify writing the anonymous functions/callback functions

|  |  |
| --- | --- |
| Callback Function | Arrow Function |
| function(x, y) {   stmt1;  return value; } | (x, y) => {  stmt1;  return value; } |
| function (x, y) {   return value; } | (x, y) => value;  [or]  (x, y) => { return value; } |
| function(x, y) { stmt1; } | (x, y) => stmt1; |
| function(x) {  stmt1;  } | x => stmt1; |
| ex: items.forEach(function(v, i) { } ); | ex: items.forEach( (v, i) => {} ) |
| ex: items.map( function(v, i) { return v+5; }); | ex: items.map((v, i) => v + 5; ); |



Destructuring objects & arrays

Suppose you have an object as below:

let employee = { id : 100, name : “Raj”, salary:25000, desig: “Sales”, address: {state: “KA”, city:”BLR”} };

If you want to assign the property to a variable you need to access using the dot operator

let id = employee.id;  
let name = employee.name;  
let salary = employee.salary;  
let desig = employee.desig;  
let state = employee.address.state;

Using employee.name at many places would be difficult if there are too many nested properties   
With destructuring its much more simpler

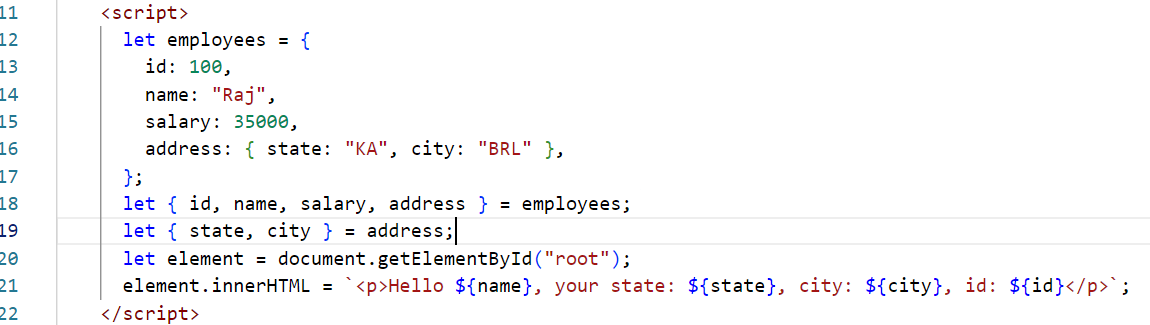
let {id, name, salary, address} = employee;  
let {state, city} = address;

you can use id, name, salary, address directly without using employee.name or employee.address

You can also destructure arrays, but use [ ].

let items = [30, 40, 10]

let [x, y, z] = items; // x = 30, y = 40, z = 10;



React.js

It is a Javascript library used to develop rich UI’s, it helps you to create reusable UI’s which are called as components to build SPA (Single Page Applications).

React.js uses a special syntax i.e., JSX

JSX: It stands for Javascript XML or Javascript Extension, this supports all the JS features along with other features to simplify writing the HTML inside the Javascript

How JSX is written

let content = <p>Some content</p>

To print the content use { content }

let employee = { id : 100, name : “Raj”, salary : 35000 }

<div>{ employee.id } { employee.name } {employee.salary}</div>

Using expressions & JS function in JSX

<p>{ 2 + 3 }</p>

<p>{ Math.pow(3, 2) } </p>// print 9

React.js uses 3 libraries to develop the application

1. React
2. React DOM
3. Babel: Converts JSX to Javascript, so that browser understands

Components: These are the visible part in the page, you can independently create the components & use with other component to create a complex component & build an application

Root component: This is the main component which represents the entire page, it will have other components in it, you need to use all the components inside the root component, because this is the only component loaded to the HTML

Online website to quickly develop create components

Codepen is the website that helps you to quickly create react components & use in the HTML.

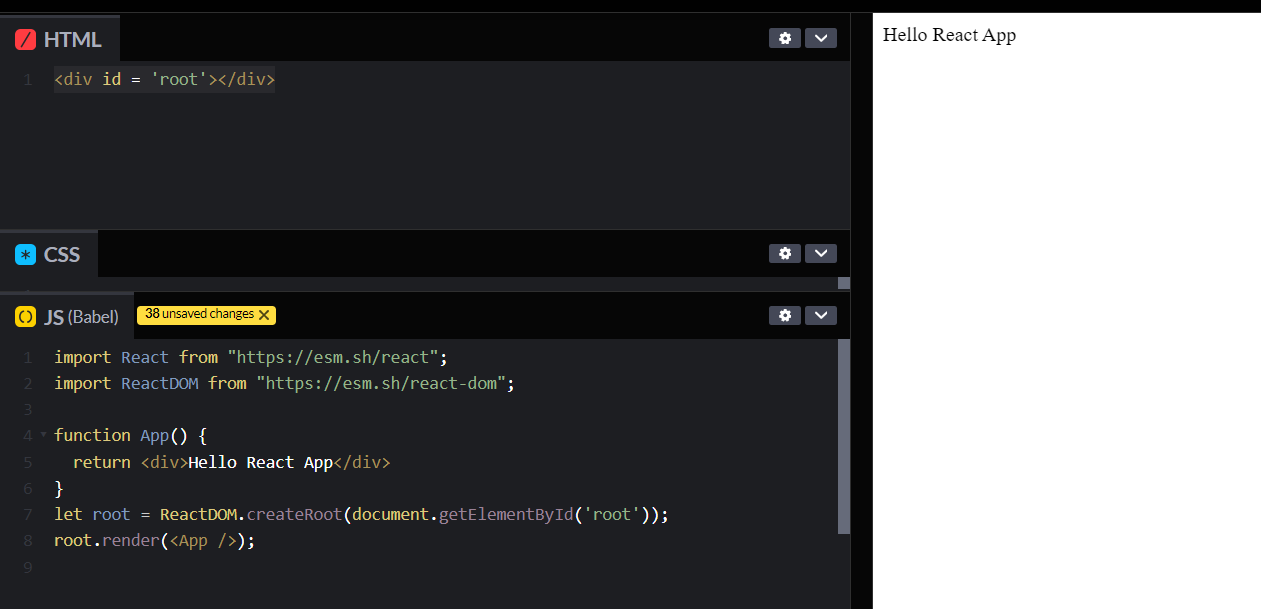
ReactDOM: It is a library that accesses the HTML element in the DOM to map the component developed in the React

let root = ReactDOM.createRoot( document.getElementById(“root”) );

root.render( <RootComponentTag /> )

function RootComponentTag() { return complex components }

Hello World program in React



React Tool Kit:

It is used to create a ready to run & deploy react applications, it follows industry standard structure & provides many automated features like

1. Live server to launch the application
2. Webpack to bundle the application into a build file that can be deployed in any server
3. Compiler to compile react applications
4. Auto-compilation & Live reload features while making changes in the application
5. Git tracking feature

create-react-app: it is the tool that used to create the react applications

npx create-react-app project-name: This command directly downloads the react project using the latest version of the tool

There’s another way also to download the react project i.e., by installing the toolkit using

npm install create-react-app: Installs the toolkit

create-react-app project-name: creates the project

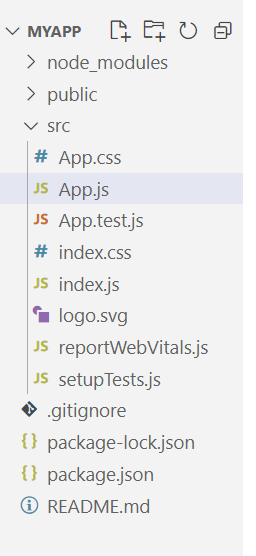
Verifying the commands



create-react-app with npx downloads the project from the latest version of the toolkit, once it downloads the project you need to navigate to the project and can type following commands

1. npm start: This runs the react application on a live server in 3000 port
2. npm run build: This creates a build file for the project using webpack and other libraries, build is mainly used in the production environment which will have optimized code i.e, if in the development environment the project is around 300MB then the build file will be in KB’s like 10KB or 20KB

Project Structure



Types of components

You can create components in 2 ways

1. Using class
2. Using functions

Before React.js 16 there were many differences in class based and function based components, but afterwards they provided same features in both

Creating class based components.

class Hello extends React.Component {  
 render() {  
 return <div>Some Content</div>  
 }  
}  
Function based components

function Hello() {  
 return <div>Some Content</div>  
}

Note: Before React V16, certain things were not able to do with functions hence classes had to be used, but the latest version of React has everything that can be done in function based components, you can avoid using class based components

Rules of JSX, usually its treated limitations of JSX by HTML developers

1. A component must have only one root tag
2. Every tag must be closed, even the self closing tags

ex: <input> is an error

ex: <input></input> is not an error

1. It is strict about the contents, if some tags doesn’t have content you can’t have contents in that tag, JSX throws error

ex: <input type = ‘text”>HELLO</input> this leads to error  
ex: <input type = “text”> </input> this also leads to error because <input> can’t have a space

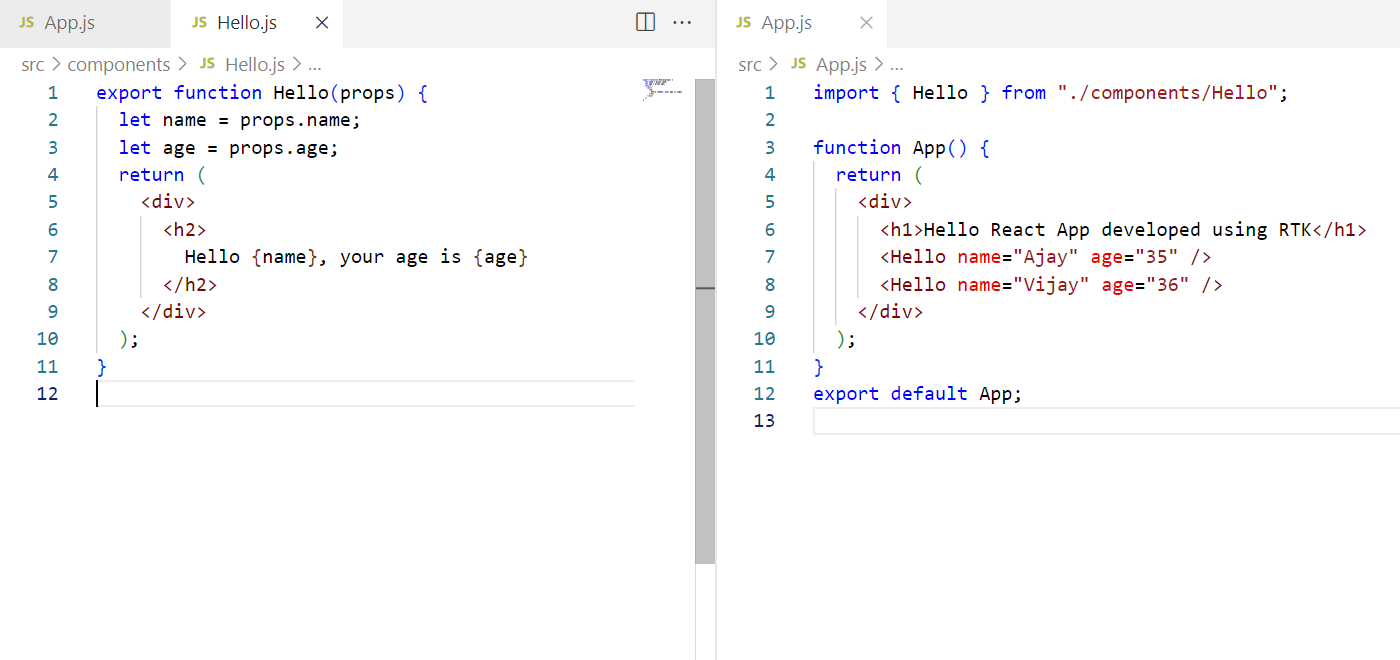
How to display the data in the component

You can use { expression } to display the data

props: You can pass the data from parent to child component

function App() {  
 return (<div>  
 <Hello name = “Ajay” /> // props = { name : “Ajay” }  
 <Hello name = “Vijay” /> // props = { name : “Vijay” }  
 </div>);  
}

function Hello(props) {  
 return {props.name}  
}



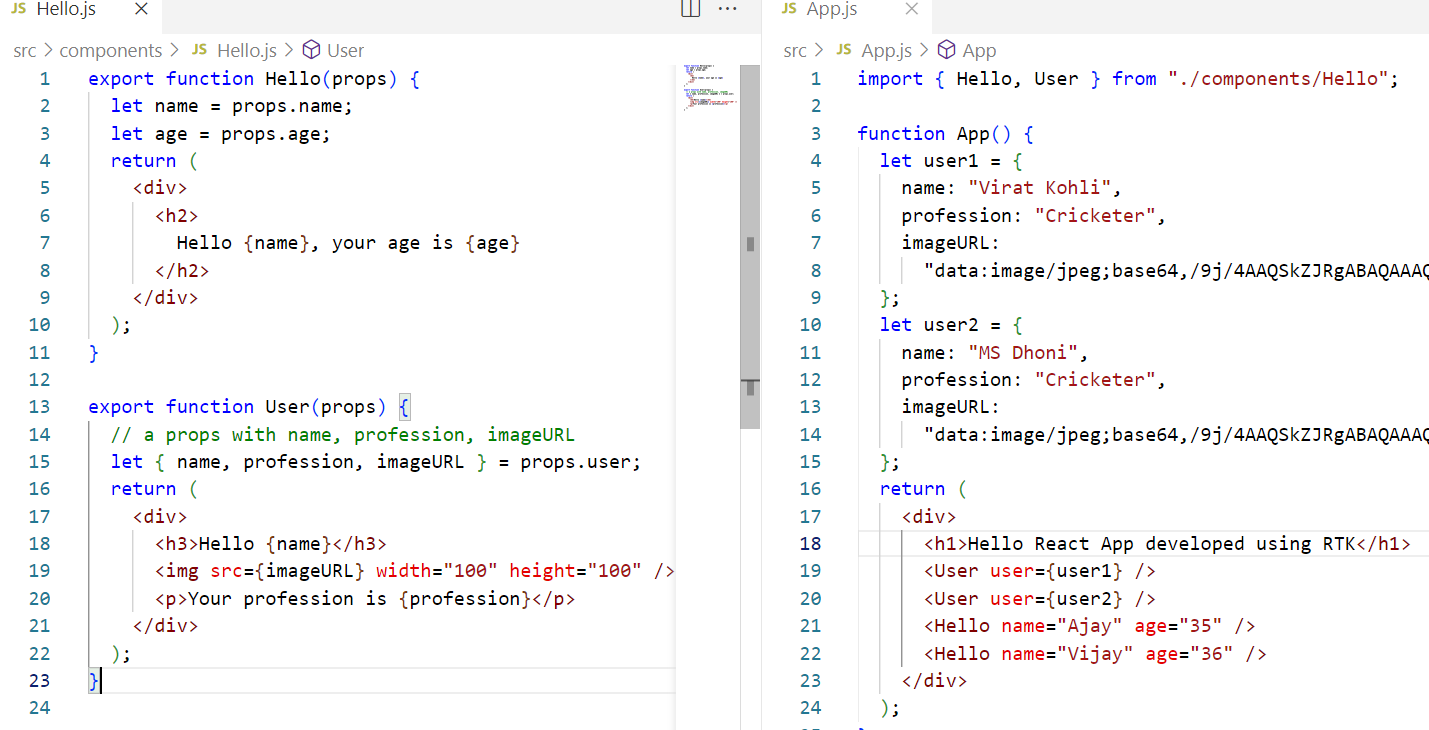
How to pass the complex data i.e., objects

You can have an object and pass that to the components, so that components can use props to access those data.

let user1 = { name : “Sachin”, profession: “Cricketer”}

<User user = {user1} />  
function User(props) {  
 let name = props.user.name;  
 let profession = props.user.profession;  
 [or] let {name, profession} = props.user; // because user has name & profession property - destructure  
}

Example on passing the object



Rules while creating the components

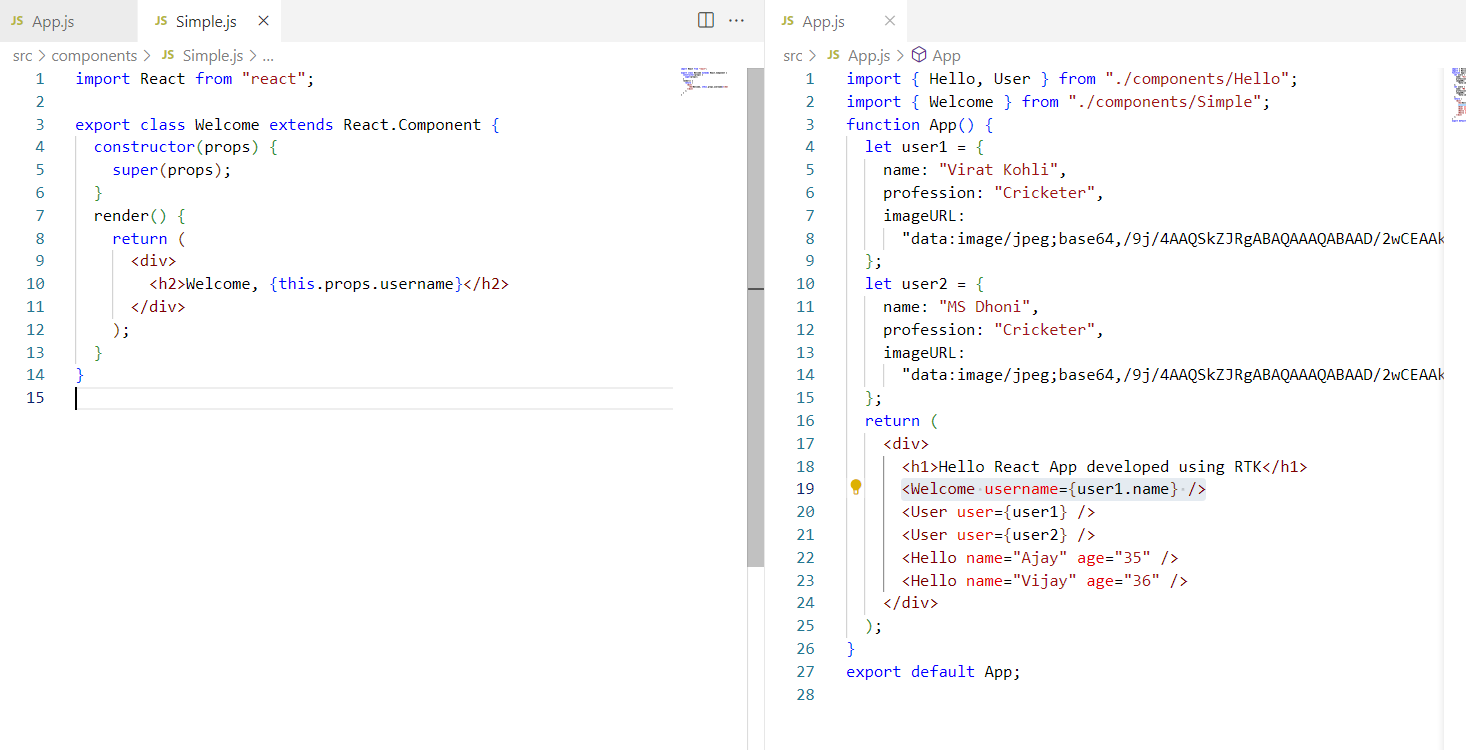
1. Your component names must begin with uppercase & can follow the camel case i.e., HelloWorld, EmployeeList and so on.
2. Your components must have only one root tag or parent tag else JSX error will occur
3. Create components in separate files when they are independent, and the components related to other components in the same file

ex: Profile component may want to perform crud operations, then UpdateProfile, DeleteProfile, DisplayProfile all can be kept in a single js file, then other related components can be another js file

How to create components using classes

You need to extend React.Component and use constructor to initialize the props

class Welcome extends React.Component {  
 constructor(props) { super(props); }  
 render() {  
 return {this.props};  
 }  
}



How to add styles in React.js

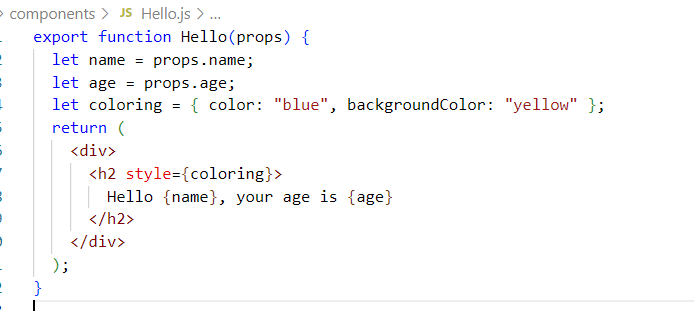
1. We can create our own CSS files and import in the JS file
2. We can create inline CSS using style attribute
3. We can include 3rd party CSS also

Inline CSS: It is used inside the JS code, hence you must access the property and the value like a Javascript object

In CSS you write { “color”:”red”, “background-color”:”yellow” }

In Javascript / React you write let styles = { color : “red”, backgroundColor : “yellow” }

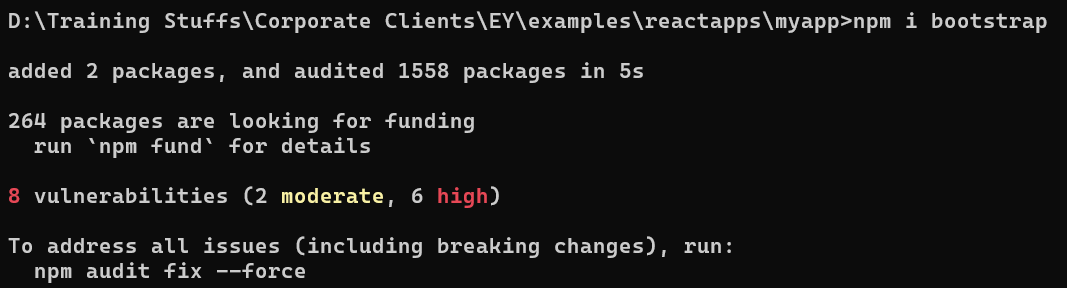
<p style = { styles } >Some content</p>



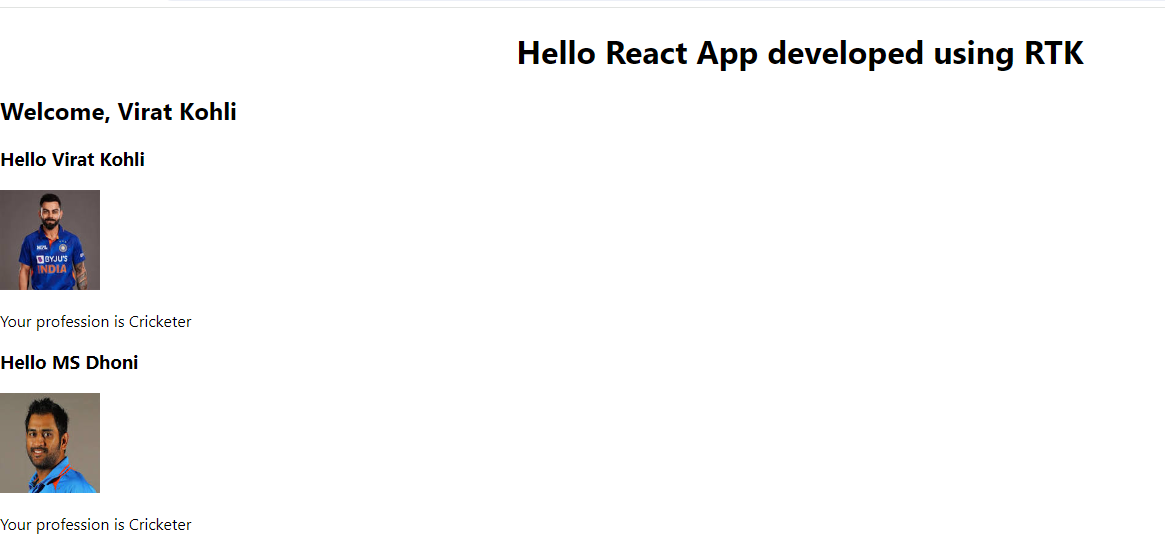
Adding the bootstrap in the react application

1. Install the bootstrap using npm install bootstrap in the project
2. You need to import the css of bootstrap in the index.css file (global stylesheet), using @import url(path)
3. You can use their classes in any component.

Installing bootstrap



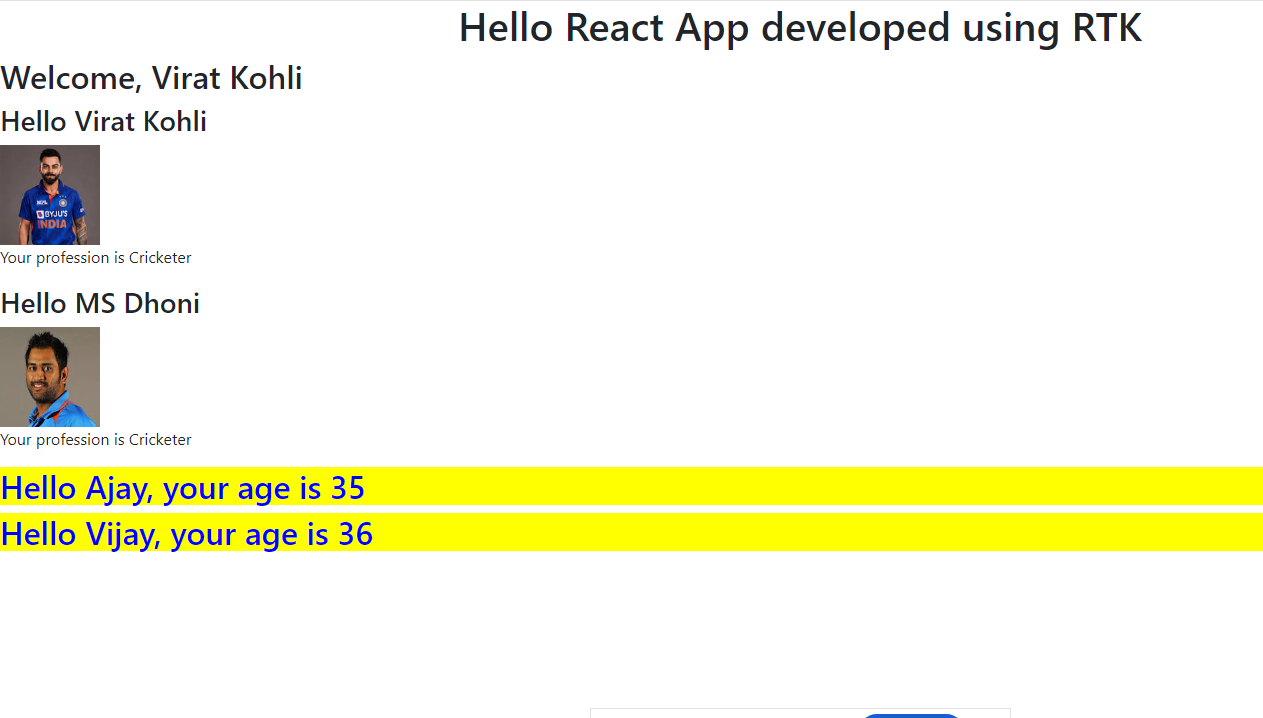
Before adding bootstrap in the index.css the output looks as below



Adding the bootstrap in the index.css



After importing the bootstrap the output looks as below



Difference in using CDN link vs Downloading the bootstrap

CDN must be avoided because it needs network connectivity and accessed over the internet, whereas downloading is better because it is always part of the project.

Rendering arrays

We must use List & Keys to iterate the arrays, list is a collection of elements which is an array & key is used to uniquely identify the element in the List

While rendering the array we must generate HTML elements for each iteration hence we must use the inbuilt map function as it transforms the data, it can be used to transform the array element into array of HTML element.

let iplTeams = [“KKR”, “DC”, “RCB”, “CSK”, “RR”];

{   
 iplTeams.map( (v, i) => <p key = {i}>{v}</p> );  
}

You can display the elements in the <ol> or <ul> or in the table using <tr>

<ol>  
 { iplTeams.map((v, i) => <li key = {i}>{v}</li>) }  
</ol>

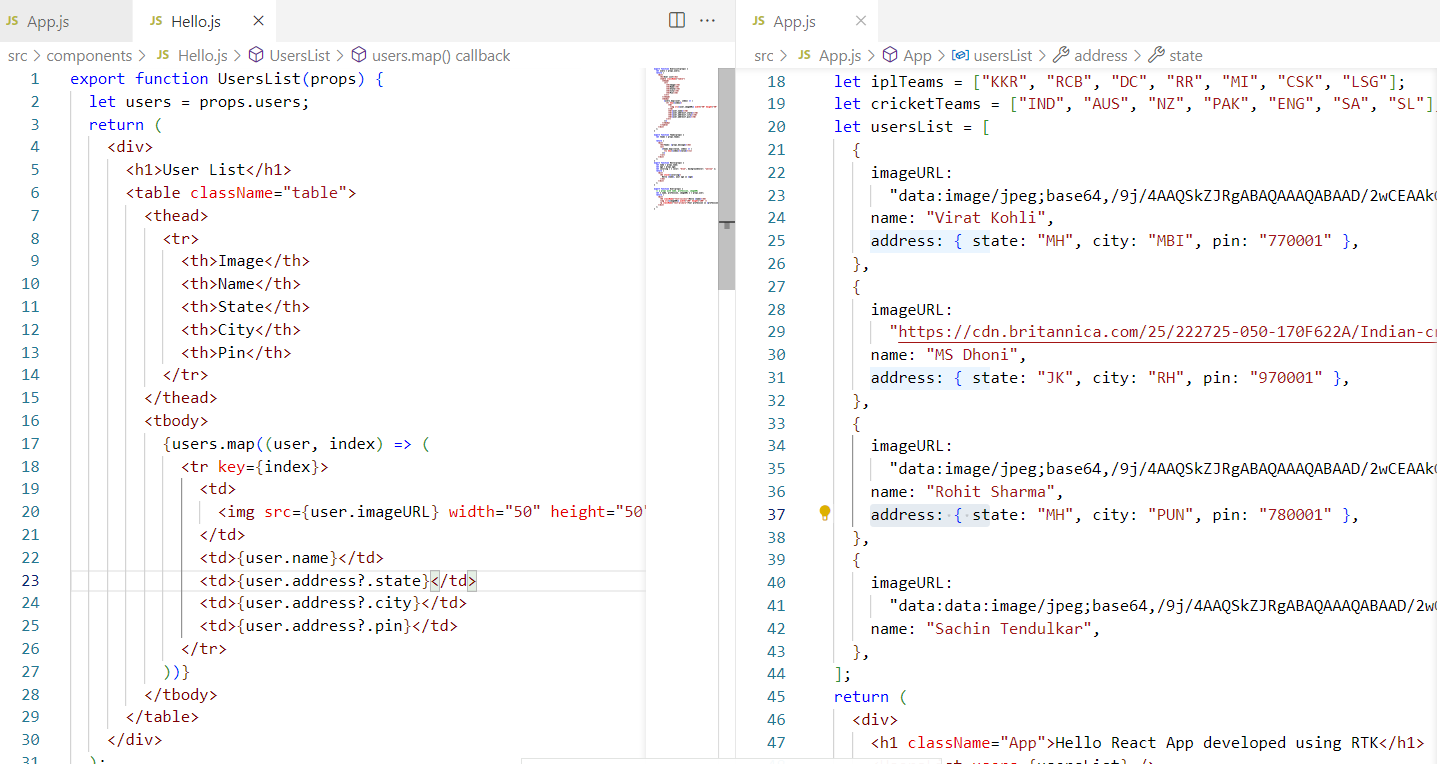
Activity:

Pass an array of objects to a component and display the result in a table format, use the array as below

let users = [   
 { imageUrl : “googleImageURL”, name : “..”, address : {state:”…”, city:”…”, pin: “…”},  
 { imageUrl : “googleImageURL”, name : “..”, address : {state:”…”, city:”…”, pin: “…”},  
 { imageUrl : “googleImageURL”, name : “..”, address : {state:”…”, city:”…”, pin: “…”},  
 { imageUrl : “googleImageURL”, name : “..”, address : {state:”…”, city:”…”, pin: “…”},  
] ;

In App component pass users to a component  
ex: <UsersList usersList = {users} />

Display the result in table which will have headings like Profile Pic, Name, State, City & Pin



Optional Chain (?.):   
It is one of the Javascript feature which enables you to access the nested property only if its present, else it stops accessing the nested properties, this avoids runtime error

ex: { user.address.state } may lead to an error compare to {user.address?.state}.

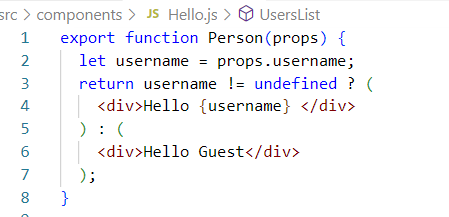
Conditional Rendering

Rendering the content based on the conditions, you can use if else logics or ternary operator if only two conditions need to be evaluated.

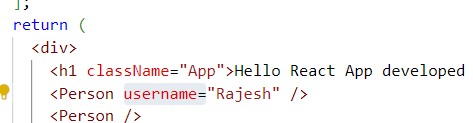
if( x > y ) { return <div>Hello</div> }   
else { return <div>Welcome</div> }

Using ternary operator

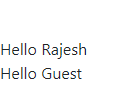
return ( x > y ) ? <div> Hello </div> : <div>Welcome</div>



App.js



Output:



States: These are components data that can be updated, as well as can be displayed and also can be passed as props to other components.

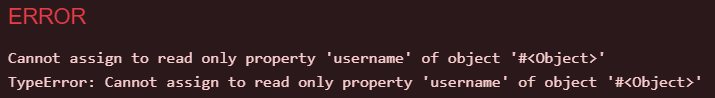
Components will have two types of data

1. props: read-only data
2. state: read and write data

Trying to modify the props gives error



Output:



State: This is a modifiable data in the component, you must first create the state with an initial value and update the state based on some actions like

* User enters an input
* When backend data is ready
* When some other actions need to make changes

How to create a state

Before React V16, function components were used only for props and can’t be used for states, at that time developers had to use classes to use the state, because React.Component was providing a state property for every components created with class.

V16 onwards React.js introduced some special functions which are called as React Hooks which made functional components to get all the features what class components get.

Before V16, you can use state as below

class User extends React.Component {   
 constructor(props) {   
 super(props);  
 this.state = initialValue; // state is inherited from the React.Component  
 }  
 // to update the state you need to use a setState() function  
 update() {   
 this.setState( newValue );  
 }  
}  
React released hook functions to create states in the function components from V16 onwards, to create states there’s a hook function called useState(initialValue), it must be imported from the React library.

import { useState } from ‘react’;  
  
function User(props) {   
 let [name, setName] = useState(initialValue);  
 let update = () => {   
 setName(newValue);  
 }   
 return { name }   
}

In Class you can’t give your own state names or set methods to modify you need to work around state & setState

ex:

In class to initialize:   
this.state = {name : “Alex”, age : 30}

To modify the state you must use setState({ });

this.setState( { name : “Raj”, age : 31 } );

In function to initialize

let [age, setAge] = useState(30); // initialize value is 30  
let [name, setName] = useState(“Alex”); // initialize value is Alex

To modify the age & name you use setAge & setName

setAge(35);  
setName(“Raj”);

Databinding:

binding the data or assigning the data from UI to the component function or vice versa

props is a data binding between the components i.e., from parent to child

We can read the input from the front-end and assign its value to the state using events, in React.js you need to handle events slightly different compare to the normal Javascript event handling.

In Javascript

<button onclick = “update()”>MyButton</button>  
function update() {   
 read inputValue using some code & assigns value to some variable  
}

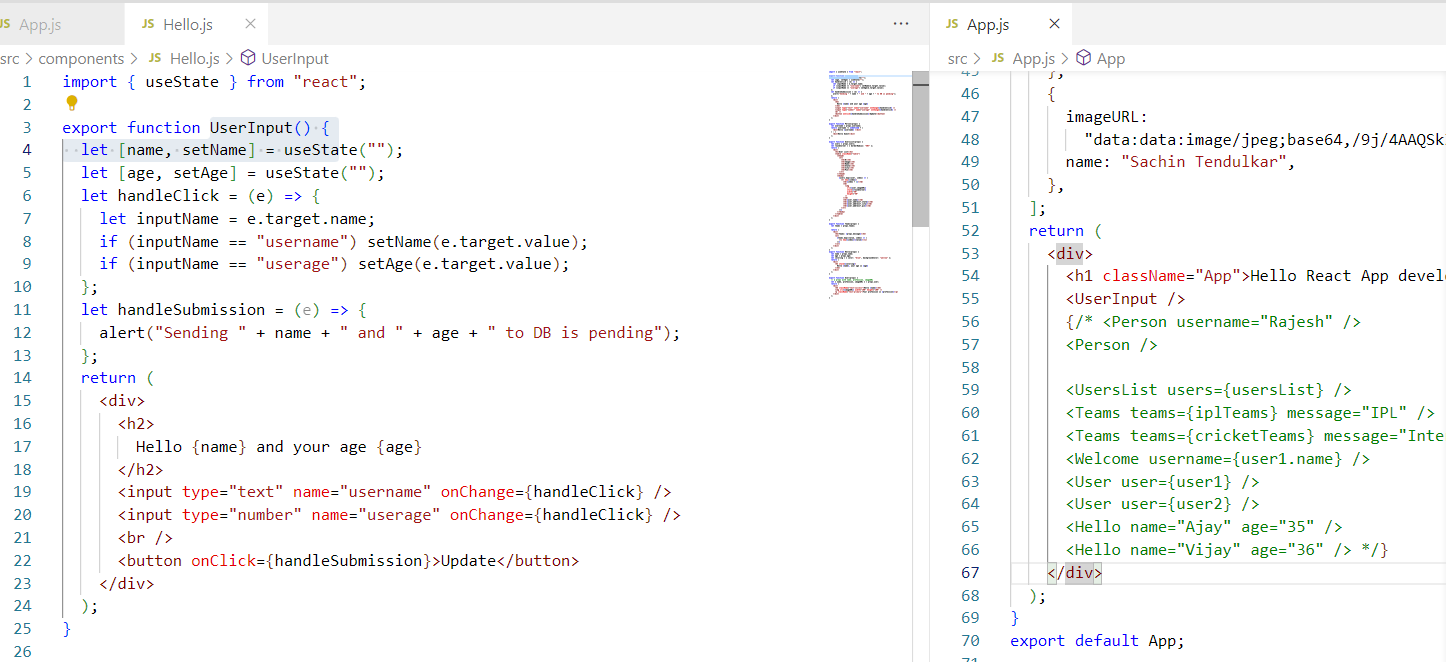
In React.js: You always start the event names in lower case followed by camel case for each new words

<button onClick = {update}>MyButton</button>

let update = () => {   
 read the value & assign value to the state  
}

This is one way, but you can also write callback in the same line, i.e., called as inline callback

<button onClick = { () => { read value and assign value to the state } } >MyButton</button>



Activity

Create a form with below inputs and a submit button

1. name
2. password
3. dob
4. phone
5. email

Store these inputs in the state and when you click on the submit button then display the state in the web page

Summary:

1. HTML
2. CSS
3. Javascript
4. SPA
5. Components
6. Props
7. Nested Components
8. State
9. Event Handling
10. List & Keys

React Forms: These are used to submit the data to the application, whenever you have a form and submit the form an event onSubmit occurs where you can handle that event and pass the data to the backend

<form onSubmit = { handleSubmit }>  
 <input type = “text” name = “username” />  
 <input type = “password” name = “password” />  
 <input type = “submit” value = “Login” />  
</form>

Vite: It is a tool provided by a React community to quickly download the project & you can install the dependencies later, it runs the project in a different port and also the command used to run the project is also different.

npm create vite@latest app-name

Commands used to run the react project

React Toolkit: npm start

Vite: npm run dev

Installing Bootstrap in the project

Note: Keep the running server command prompt as it is, and open a new command in the project location & install other libraries

Adding bootstrap to the project

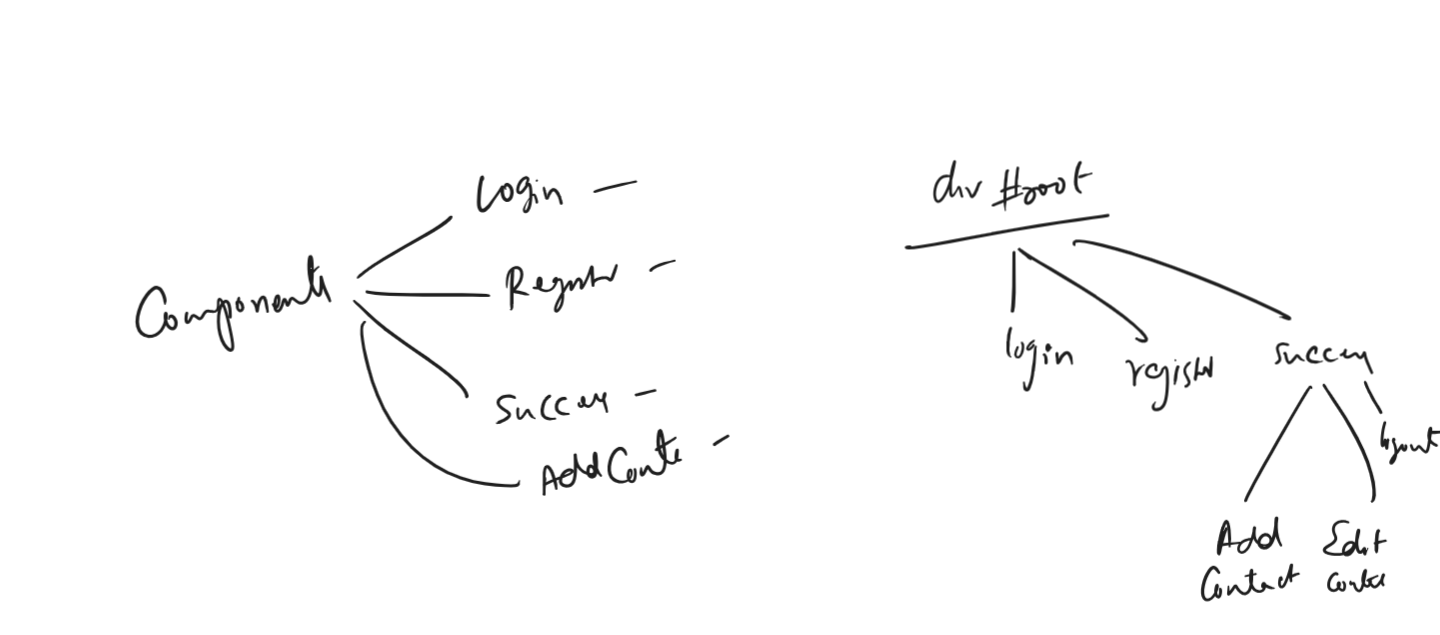
open index.css and copy the bootstrap/dist/css/bootstrap.min.css path in the @import

Case Study

Contact Management Application: Here as a user you must able register yourself and login, after you successfully login you must able to perform following tasks

1. Add Contact Name & Phone number of a your friends
2. View all contacts
3. Delete contacts
4. Edit contacts & Edit your details
5. Logout

Note: We are going to interact with the Database with Node.js later



First thing to implement

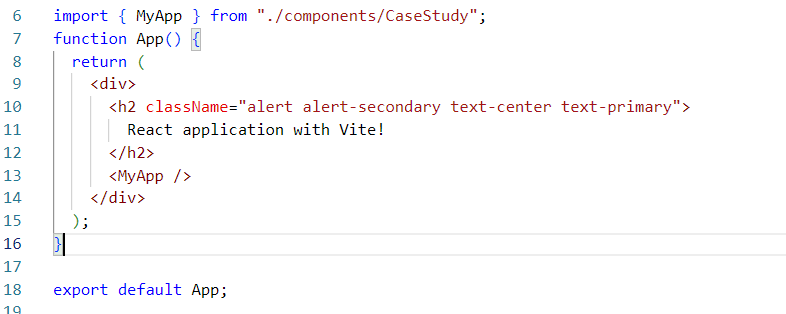
1. Registration Component
2. Login Component
3. Success component

Registration Component: It must accept profileId, name, password, dob, phone, email

components/CaseStudy.jsx



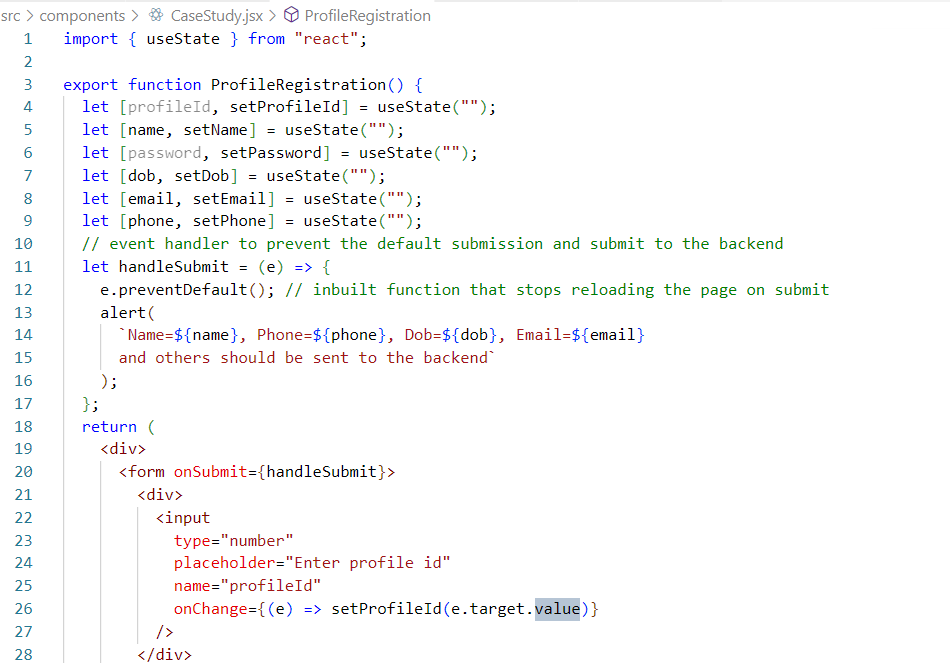
App.jsx



Implementing the Registration form:

Enter profileId, name, password, dob, phone & emailId

CaseStudy.jsx





Two way data binding :

Data that is shared from UI to component function and component to the UI is two data binding

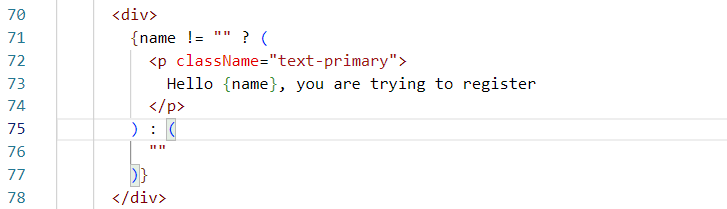
<input type = “text” onChange = {e => setName(e.target.value) } />   
The above code takes data from the UI and passes to the component function

{ name }

The above code takes the data from the component state and passes to the UI

Conditional rendering

Rendering the output based on some conditions



Above code shows the message Hello {name}, you are trying to register if the name is not empty, else it doesn’t show any message.

Form Validation:

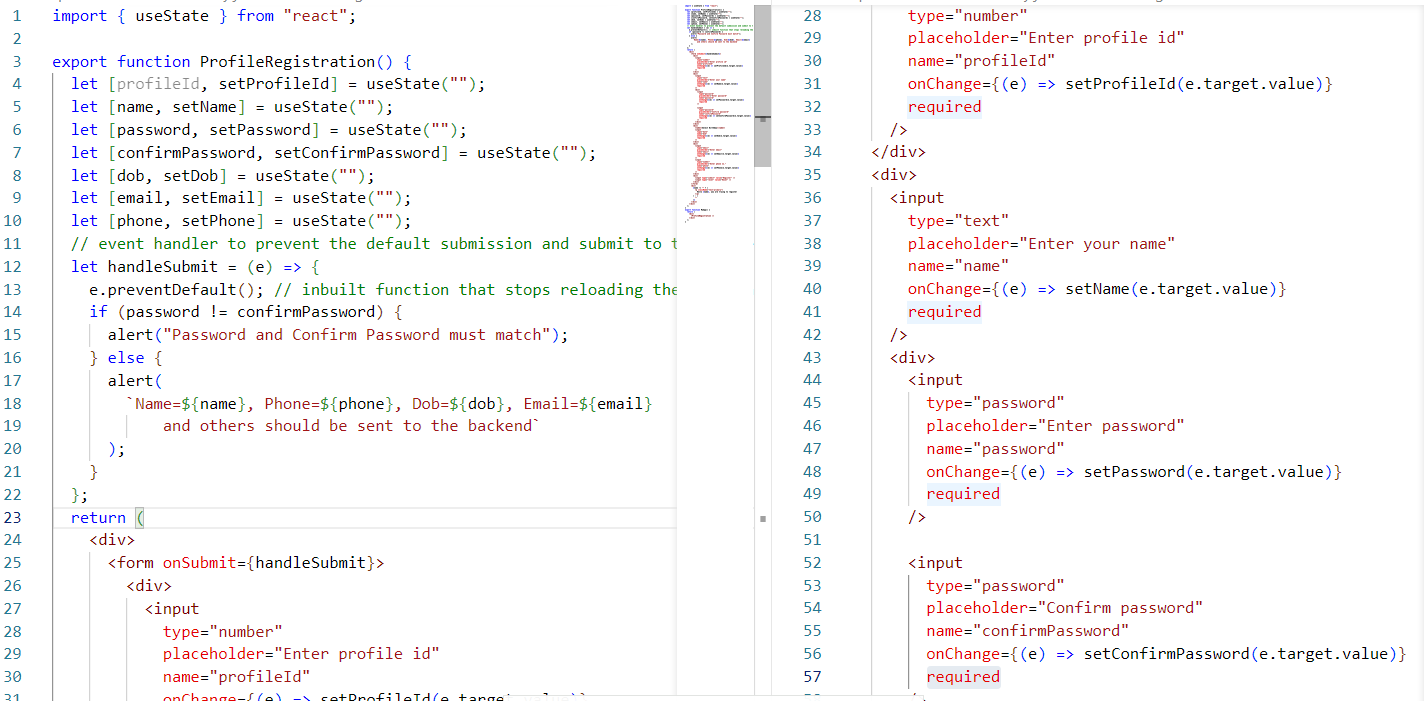
When user tries to enter some invalid input which you don’t want to send to the backend then you can validate the form and show the user the appropriate error message.

By default HTML itself has inbuilt validators like email validator, required validation however we can also create custom validators based on our requirement.

Inbuilt HTML validators

required attribute we can use.

Custom validators: We can create our own validators and display the error message.



useEffect hook method

This is one of the life cycle method in React component, it is called everytime a state changes or when the component is rendered/re-rendered, you need to write this method at the top level of the component, it takes an arrow function in the 1st parameter, and an array of state in the 2nd parameter

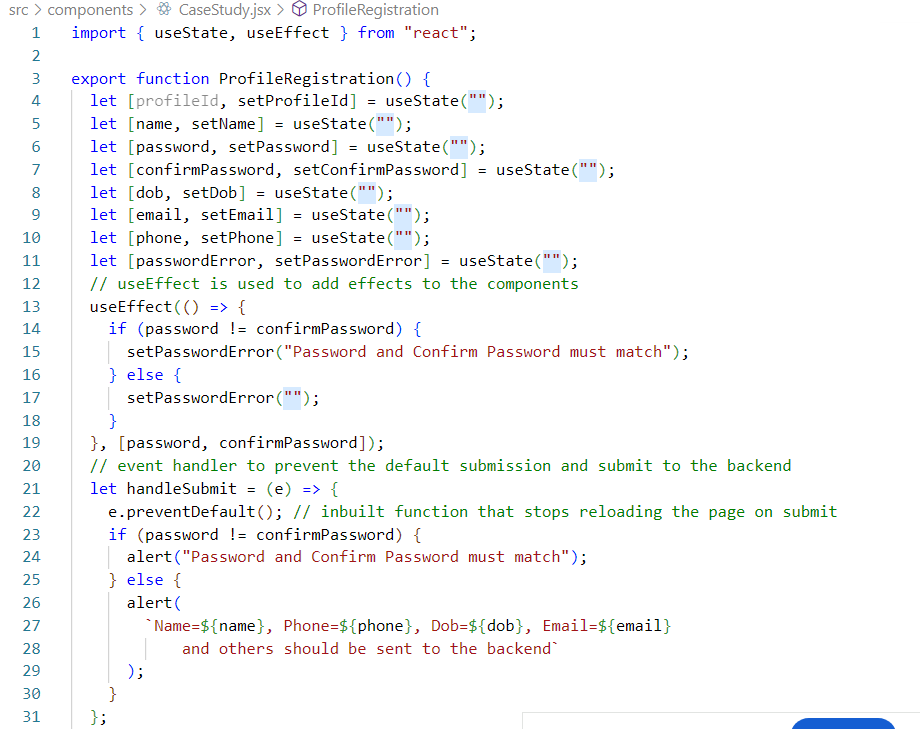
i.e., useEffect(callbackFn, [])

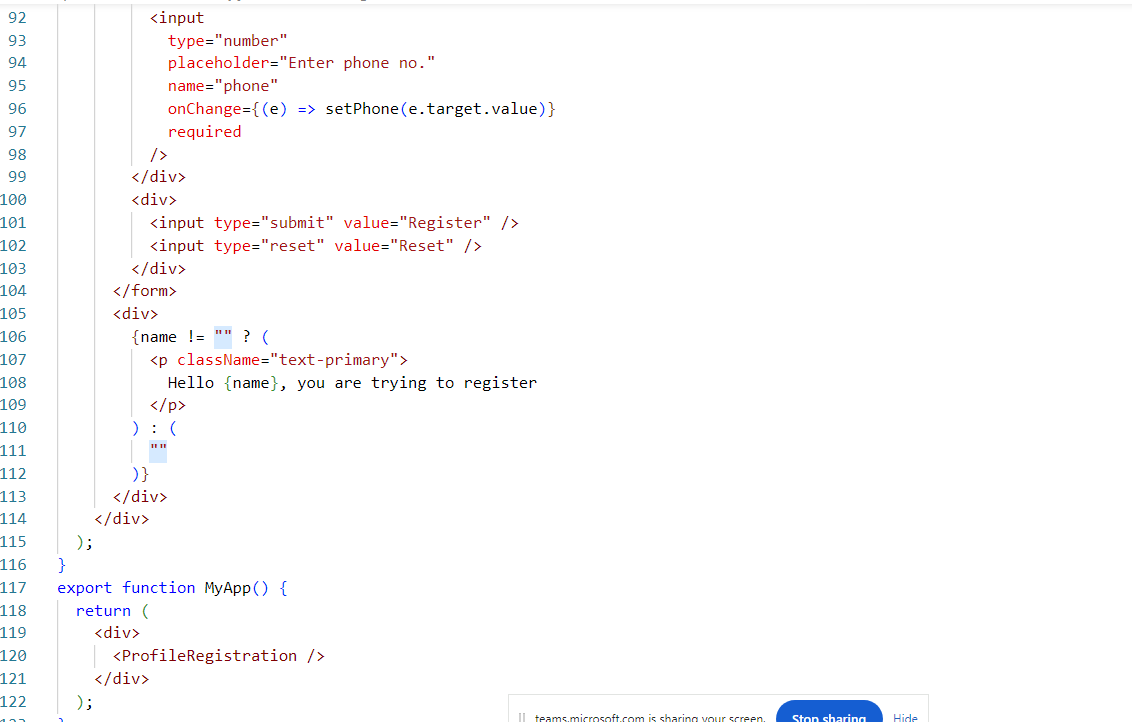
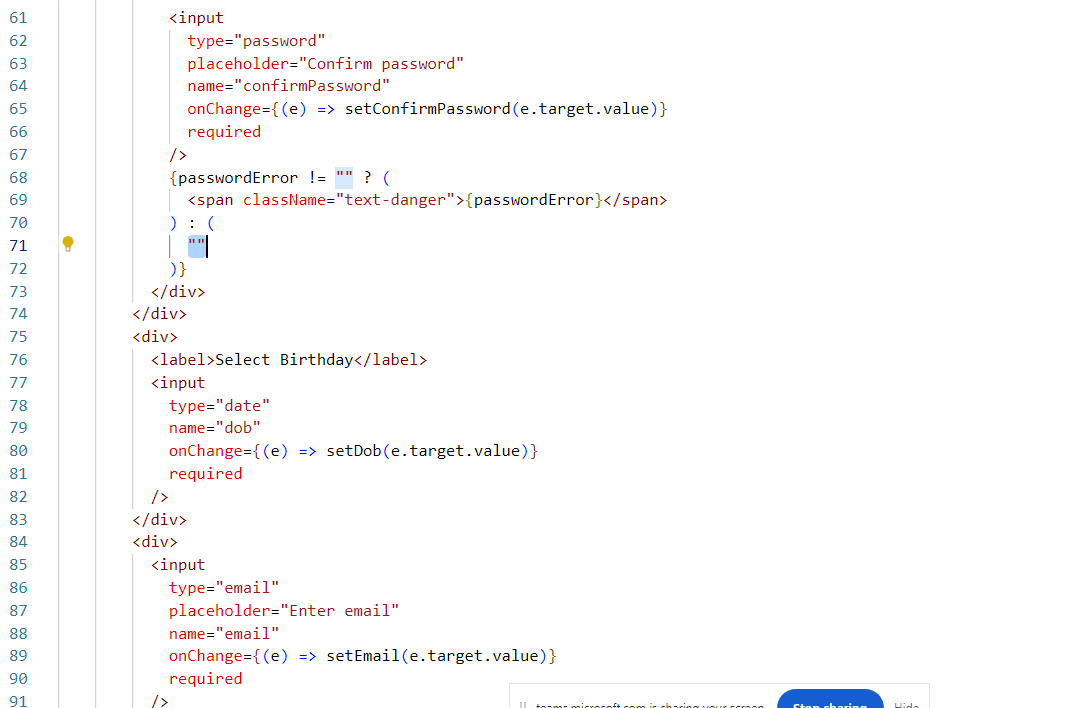
callbackFn: It called when the component is rendered and also when the state mentioned in the 2nd argument changes

[]: you can mention the list of states that need to be detected for the changes to call the callbackFn.

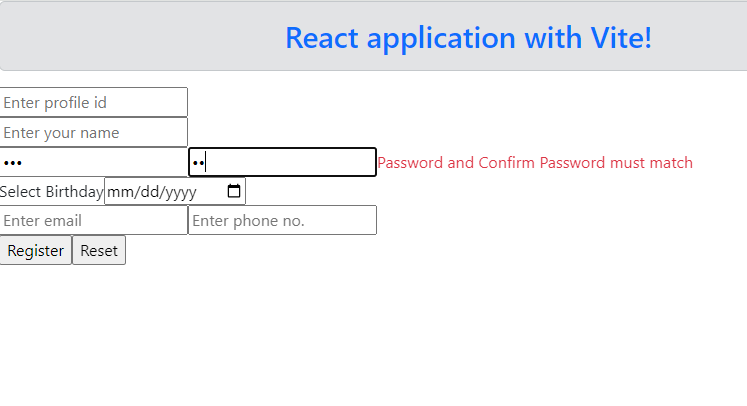
i.e., useEffect( () => { }, [password, confirmPassword]); the callback is called only if password or confirm password changes.

Note: it must be imported from the react library





Output:



Activity:

Create a ProfileLogin component that shows user to enter profile id and password, validate the input box so that both the input fields are required, show the error messages beside the input box if the input field is empty, display this component just below the registration form for the time being.

Note: Use bootstrap form-control classes by referring the bootstrap website to give a good look & feel

Note: Create the ProfileLogin component in the same file in which you created the ProfileRegistration

React Routers

In order to navigate from one view to another view React has a core library called React Router which will provide some predefined components to navigate to the view based on the link we click or programmatically, this react router is a core library that can be used in the browser or native applications (mobile applications), on top of this you get some libraries which specifically works on either browser or native applications, which are

1. React Router DOM: Works in web browser
2. React Router Native: Works in mobile applications

React Router DOM: It is a library that is used to create navigation features for web browsers, using this we can load any components we want programmatically or by clicking on some links

Note: React Router, React Router DOM and React Router Native are not available by default in the react projects, we must download using npm install

When you download React Router DOM automatically React Router will be downloaded.

Following are the predefined components & hook functions we get from the React Router DOM

1. <BrowserRouter>: This is the wrapper of all the components, we must keep all the components including the root component inside this component, this will load only part of the DOM instead of loading the entire browser

ex:   
<BrowserRouter>  
 <App />  
</BrowserRouter>

1. <Link>: This is used to create hyper links which will have URL configurations that is sent to the router to know what is the URL so that it can load the component matches to the URL.

ex:

<Link to = “/register”>Registration</Link>  
<Link to = “/login”>Login</Link>

1. <Routes> & <Route>: These are used to switch from one components to another components, <Routes> will have collection of <Route>, The <Route> is going to have the component and URL configurations

ex:

<Routes>  
 <Route path = “/register” element = { <ProfileRegistration /> } />  
 <Route path = “/login” element = { <ProfileLogin /> } />  
 <Route path = “/success” element = {<Success /> } />  
</Routes>

1. useNavigate(): A hook function that can provide a programmatic navigation, so that you can call it by passing the URL

let nav = useNavigate();  
if(…) { nav(“/success”) }  
else { nav(“/login”) }

1. useParams(): A hook function that allows you to read route parameters

router parameters: These are dynamic paths which can take any value while routing,

ex: we want to provide 100 URLs to specify which user has logged in then we can’t have 100 <Route> inside the <Routes> like  
<Route path = “/success/1” element = {SuccessComponentOf1} />  
<Route path = “/success/2” element = {SuccessComponentOf2} />  
 and so on till  
<Route path = “/success/100” element = {SuccessComponentOf100 } />

Solution is to create Route parameters which can load the same component for any sub-paths

i.e.,  
<Routes>  
 <Route path = “/login” element = {<ProfileLogin /> } />  
 <Route path = “/success/:profileId” element = {<Success /> } />  
 <Route …. />  
</Routes>

/:profileId can accept any value like /1, /2, /3 to /n, we need to read this using useParams

i.e.,   
let {profileId}= useParams(); // returns an object with { profileId : value }

another example : if /success/:profileId/:name, then we must use  
let {profileId, name} = useParams():   
//above returns an object with { profileId: value, name: value}

/success/1/Raj, then useParams returns an object like {profileId:1, name:Raj}  
/success/100/Kiran: then useParams returns an object like {profileId:100, name:Kiran}.

Note: React Router DOM version 6 or later has changed the name of the components and provided many changes like

<BrowserRouter>  
<Routes>  
<Route>

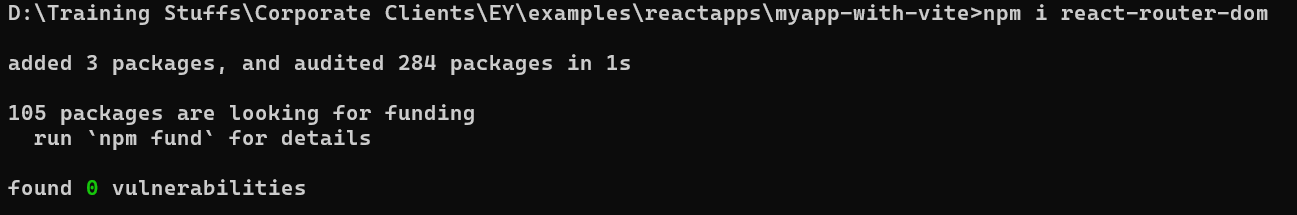
Before React Router DOM version 5, you will see different names in place of <BrowserRouter>, <Routes> & <Route> like withRouter, Switch and so on

Steps to add routers in react application

1. Installing the react-router-dom library using npm install
2. Wrapping <App> inside the <BrowserRouter>
3. Creating the <Routes> & <Route> for the components
4. Providing the url’s using <Link> or useNavigate() hook

Note: All the above components & hooks are present in the library ‘react-router-dom’ which you must import

Step1:



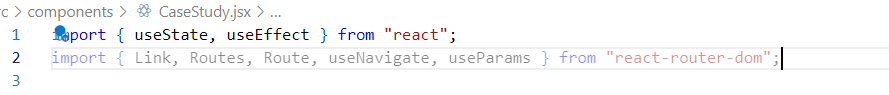
Step2:



Step3:

Writing the <Link>, <Routes>, <Route> inside some components

Importing all the modules like Link, Routes, Route and others in CaseStudy.jsx



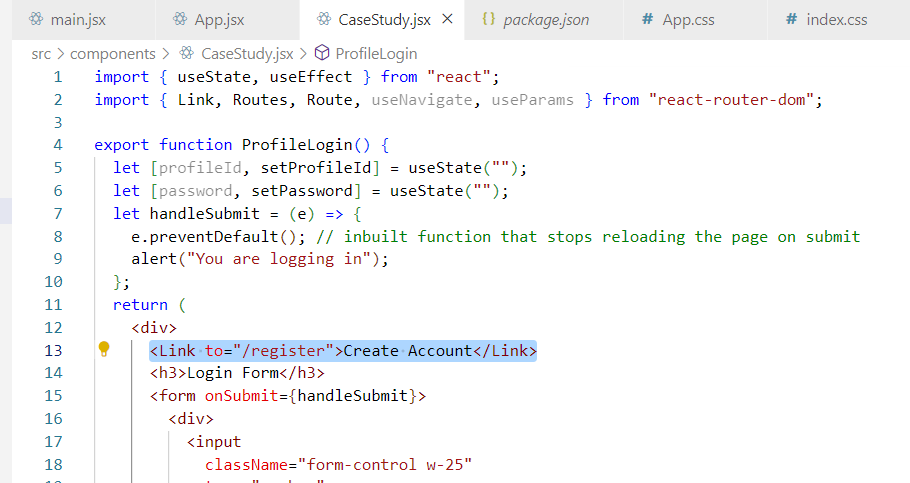
Adding Routes & Route to navigate to Success, ProfileLogin, ProfileRegistration and so on (like add contacts, view contacts and etc) in CaseStudy.jsx>>MyApp



Note: /:profileId can take any value like /1, /2, /3, /N, however /:profileId/\* is to specify that child routes, which will have child components that are loaded inside the success component

Step4:

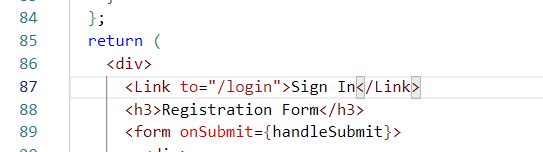
Using <Link> or useNavigate() to navigate to the components



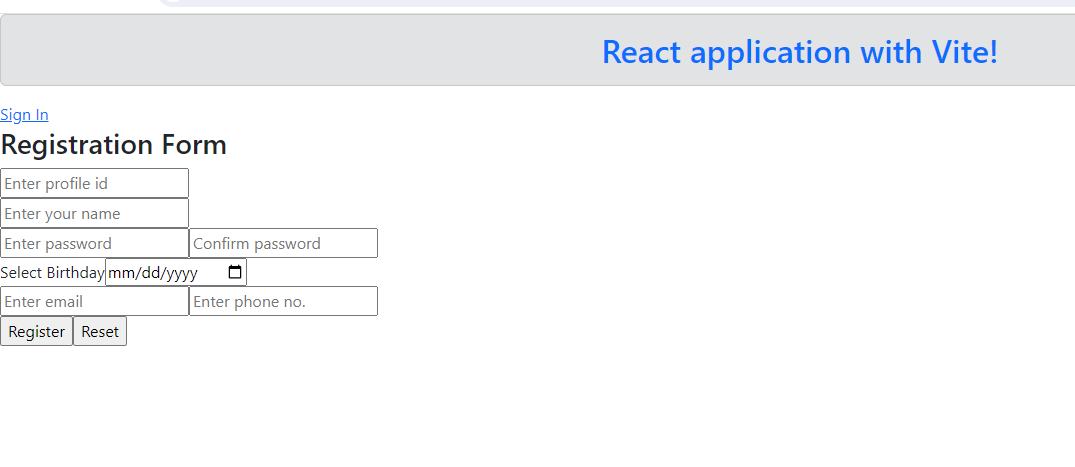
Output:



Creating <Link> in the Registration component



Output:



Programmatic Navigation using useNavigate()

let navigate = useNavigate();

navigate(“/success”);

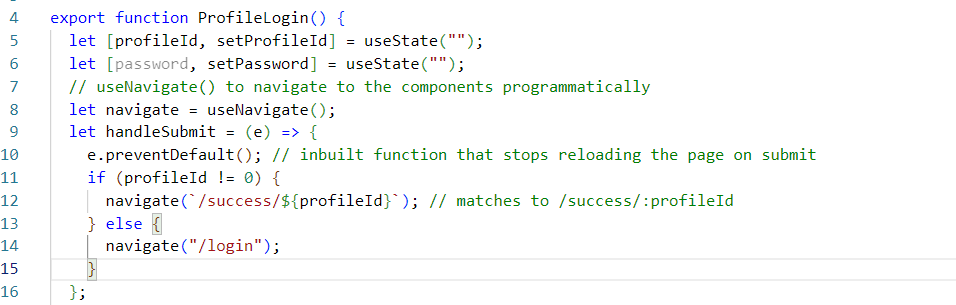
when id = 1 or 2 or 3 or N, then

navigate(“/success/”+id); this sends request to /success/:profileId

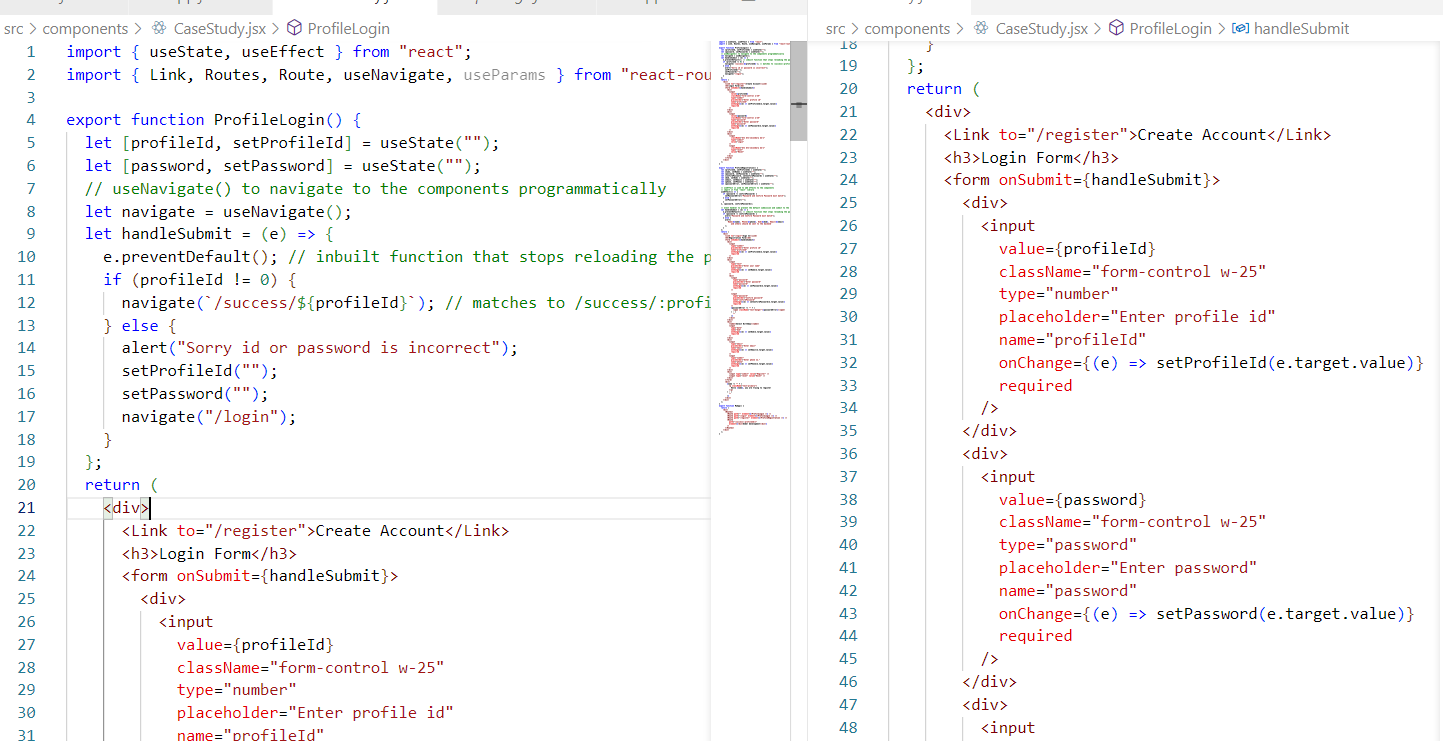
How to read the :profileId in a component

let {profileId} = useParams(); // read the :profileId and assigns its value to the variable

Note: We must use the same name what is used in the path variable i.e., /:profileId is used hence we must use let {profileId} = useParams();



Now we can navigate to the /success/anyValue, however we need to still create success component that is rendered on /success/:profileId/\*

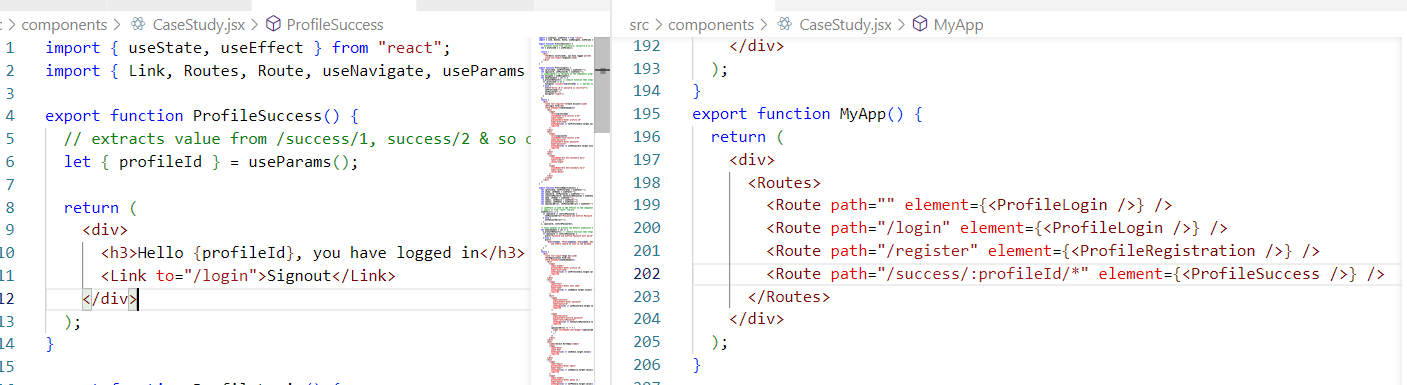


Currently we are seeing the Under development page, but we need to go to the Success component for the path /success/anyValue, the anyValue can be extracted using useParams()

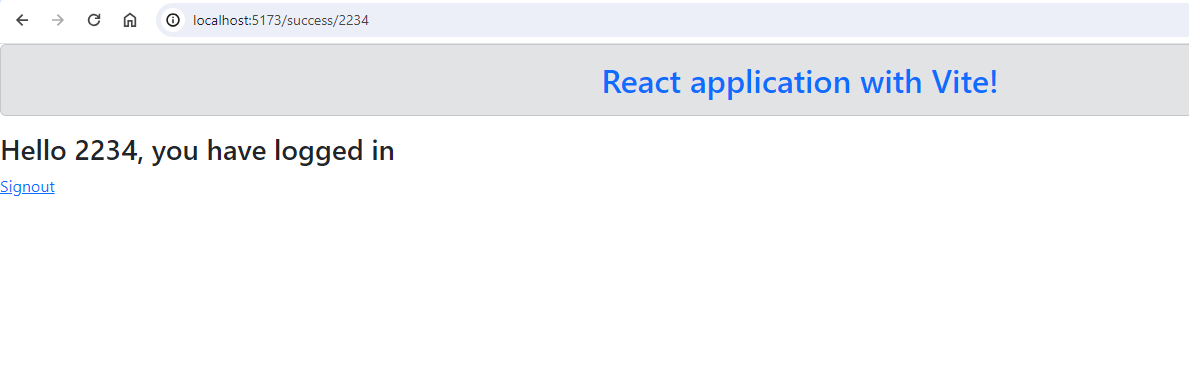
i.e.,

let { profileId } = useParams(); // we need to mandatorily use profileId, because the path parameter name is profileId because of /success/:profileId

CaseStudy.jsx



Output:



What is backend api

These are the services that can accept the data or return the data to any type of applications, usually they are web services with URLs and HTTP methods, when any front-end libraries/frameworks like JS, React.js, Vue.js, Angular, Native apps, mobile apps or any front end devices like swiping machine, atm machines tries to access these backend they either get the data or they need to send the data using 2 things

1. URL
2. HTTP methods: Specifies the type of request / operations you want to make, there are 4 http methods like GET, POST, PUT, DELETE

Rules of these HTTP methods

1. GET: Use when you want to retrieve the data
2. POST: Use when you want to create a new resource or store a new resource
3. PUT: Use when you want to update the existing resource
4. DELETE: Use when you want to delete the existing resource.

Currently we are using a Fake API to get the JSON data i.e., jsonplaceholder.com

axios library

This is the library we must use to access the backend, it provides methods which are http methods to access the webservices mapped to these http methods, each method takes URL as the parameter

import axios from ‘axios’; // we must download this using npm install axios

then make a GET request like, GET cannot submit data to the backend via request body, but it can send data via URL, but rest of the HTTP methods can

axios.get(URL)

then make a POST request like

axios.post(URL, data)

then make a PUT request like

axios.put(URL, data)

then make a DELETE request like

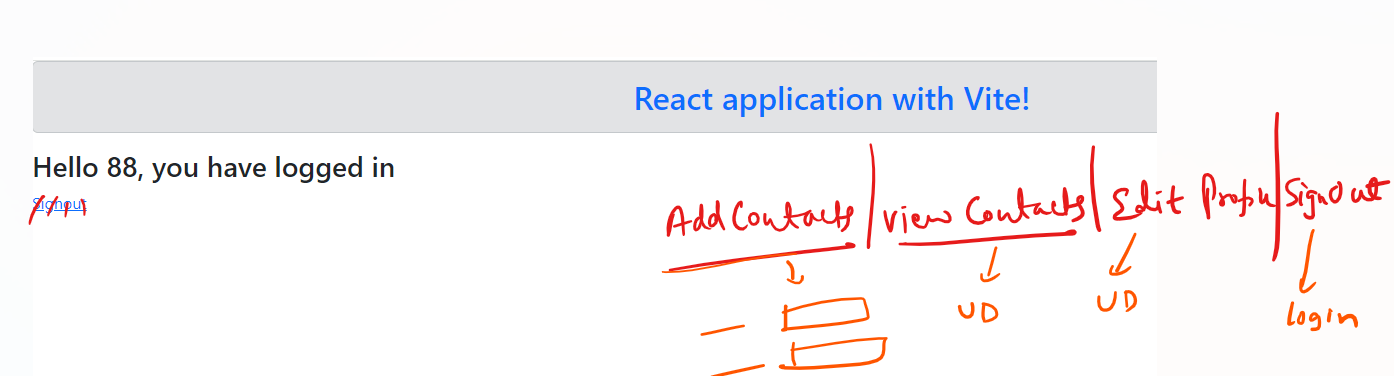
axios.delete(URL, data)

Activity:

Enhance the case study by creating child routes in the success component, from the success you need to create links to navigate to below components

1. Add Contacts: When you click on this link a component with a form that takes Name & Phone no must appear
2. View Contacts: When you click on this link just show under development
3. Edit Profile: When you click on this link just show under development
4. Sign Out: When you click on this link it must redirect to the Login component.

Note: All these must happen inside the Success & the above links must not visible in the parent route(i.e., Login/Register), it must be visible only inside the Success component



When you click on any link it must show the components related to the link, all the output you must not display at the same time

axios

It is the library used in the react application to make HTTP calls to the backend using HTTP methods like get, post, put & delete, all these methods gives a Promise object, which can result in 2 forms

1. promise resolved or success : 2xx series status codes
2. promise rejected or failed: 4xx series status codes

these promises you can handle using 2 methods

1. .then(callbackFn)
2. .catch(callbackFn)

The callbackFn are executed based on the promise result

.then() invokes the callbackFn if the promise is resolved

.catch() invokes the callbackFn if the promise is rejected

Usage of axios will be like this

axios.get(“http://abc.com”).then( res => {…} ).catch( err => {… })

Front end & Back end uses JSON as common datastructure to share the data

JSON looks like this

{ “id”:1234, “name”:”Kishor”, “salary”:45000 } : Simple JSON data

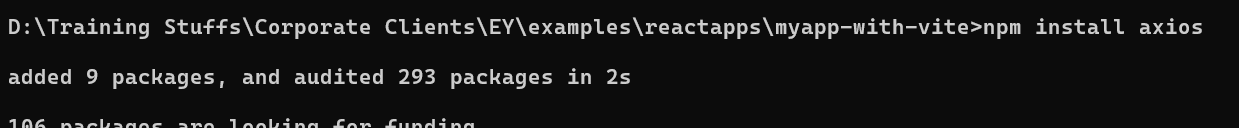
[ { “key” : value, “key” : value }, { “key” : value, “key” : value } ]: JSON array

axios takes care of converting JSON to JS and vice versa.

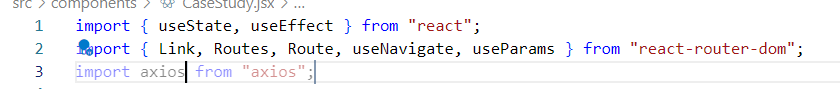
Steps to use axios to make HTTP calls

1. install axios using npm install axios
2. import axios and call http methods
3. http methods is promise based which can be fulfilled by calling then() and catch()
4. then & catch accepts an arrow function which are called based on the promise status, if promise is resolved then the arrow function of .then() is invoked, if promise is rejected then the arrow function of .catch() is invoked.

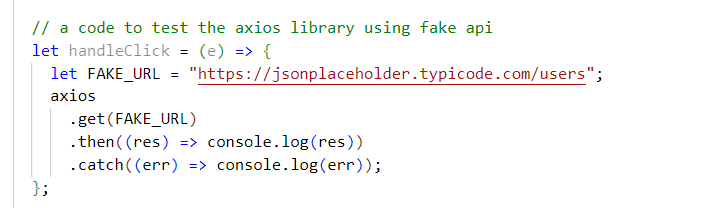
Step1:



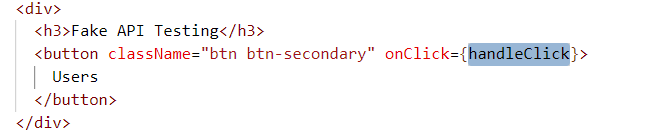
Step2:



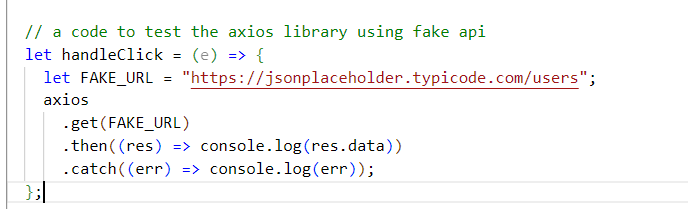
Step3: Making HTTP calls



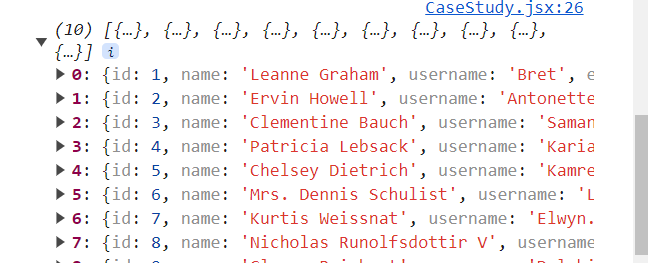
Create a button and attach the handleClick to the onClick event



We can use “data” property on the callback parameter of the .then() to access the response content.



Output:



Note: We need to use this case-study to store the profile registration in the database & login by using username & password by interacting with the DB, for that we need a real backend service, that we can implement using any technology like Node.js, Spring Boot, C# and so on.

Node.js

It is a javascript runtime environment that lets you run the javascript outside the browser, so that you can use javascript to write backend programs to perform various tasks just like other languages Java, C#, C++, Python supports

Node.js allows javascript to be used to perform any backend tasks like

1. Developing full fledged web apps
2. Accessing files
3. OS resources
4. Accessing DB’s
5. Network programs

Advantage of Node.js

1. Developers who already have idea on Javascript doesn’t need to learn a new technology for backend programs
2. It has NPM registry where lot of libraries are made available to build any kind of applications could be front-end or back-end, ex: axios, bootstrap, react-router-dom and many other libraries are available from these npm registry
3. JSON is the common format used to interchange the data between the client & the server, JSON is native to Node.js and it doesn’t need any kind of interpreter

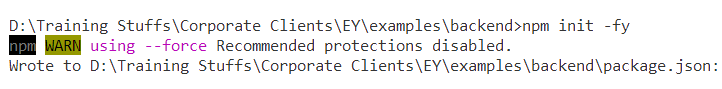
Architecture of Node.js

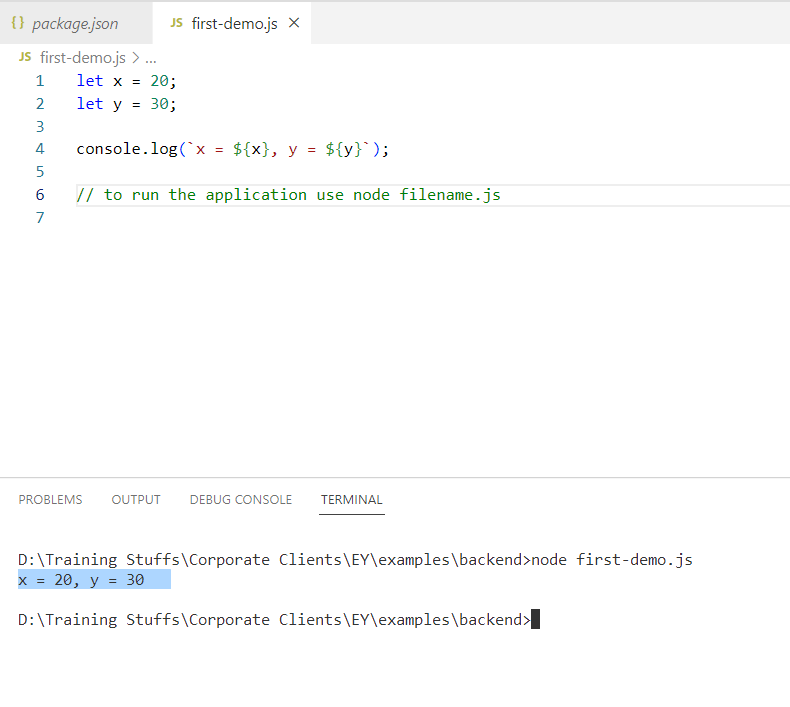
Node.js is a single thread model, which uses only one thread to run the javascript code, it doesn’t block the threads to perform any kind of complex computations because those are done in an asynchronous way.

package.json

Every node.js project need package.json which is an heart to any node project, it will have all the information’s of the project

npm init -fy: Creates a package.json file for us





Note: In node.js you can’t use inbuilt objects & functions of browsers like

* document
* alert(), confirm(), prompt()
* sessionStorage
* localStorage

Understanding the node.js architecture

Node.js uses single thread to execute the script, these scripts are executed by callstack one by one once the script is loaded.

Event loop: it runs to check the event queue if there are any scripts for execution by removing the scripts from the queue and adding to the stack for execution.

Event Queue: it keeps the scripts that needs to be executed later after completing the asynchronous tasks

LIBUV: it is a thread pool that runs the asynchronous scripts, like setTimeout, db connections & operations, network operations and so on.

Node Modules

Modules are reusable functions, objects, classes or variables that are of 3 types.

1. Core Module: Part of Node ex: fs, os, http
2. Local Module: Created within the project
3. Third Party Module: Available from the npm registry over the internet ex: axios, bootstrap, react-router-dom, express, cors, mongodb, mysql

These reusable modules can be imported using ES5 require statements or ES6 import statement

ES5 require statement

let axios = require(“axios”); // default module  
let BrowserRouter = require(“react-router-dom”).BrowserRouter; // named module

ES6 import statement

import axios from “axios” // default module

let { BrowserRouter } from “react-router-dom”

Note: By default node.js doesn’t support ES6 module imports/exports, we must add a property in package.json as “type”:”module”

package.json



Core Modules

These modules are available inbuilt like

* os
* fs
* http

OS Module: This is a module used to get OS related information’s, it is used by many toolkits to show the user OS related information’s



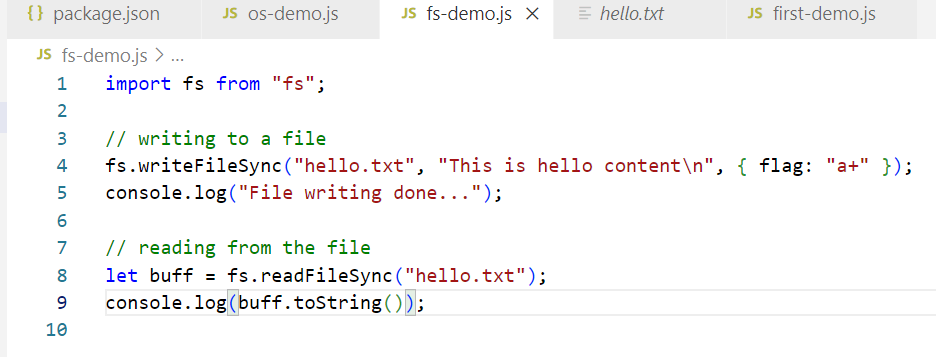
fs module

It is used to read/write files

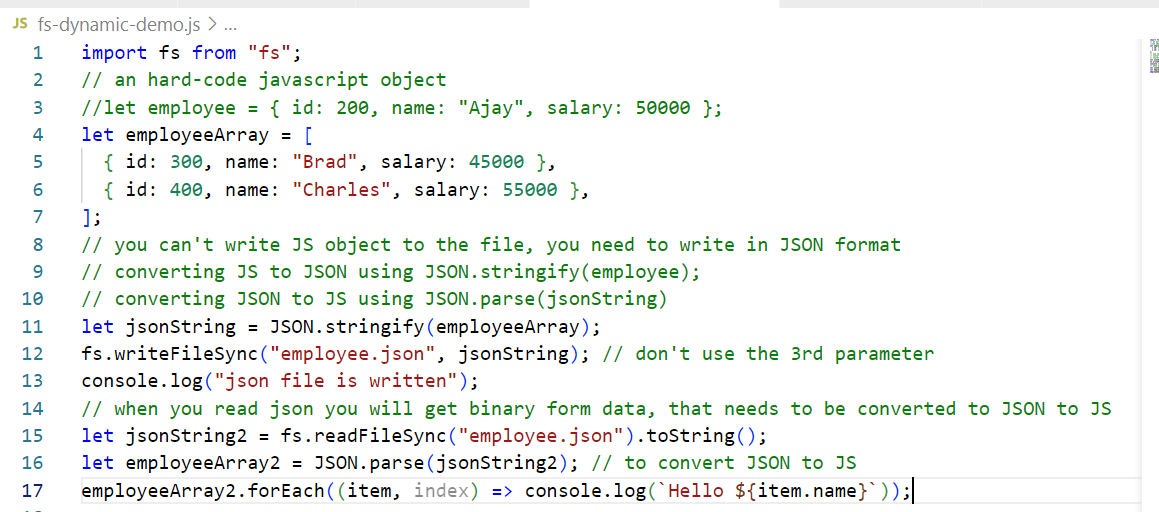
import fs from ‘fs’;

fs.writeFileSync(filename, content, appendOption);

fs.readFileSync(filename): It returns the data in Buffer format/Binary format that needs to be converted to the text using toString() if the file is text file



read & write json



Third party module: we must use npm install to download the third party modules

Taking input from the keyboard: readline-sync is the library that helps you to take input from the keyboard

npm install readline-sync

import readline from ‘readline-sync’

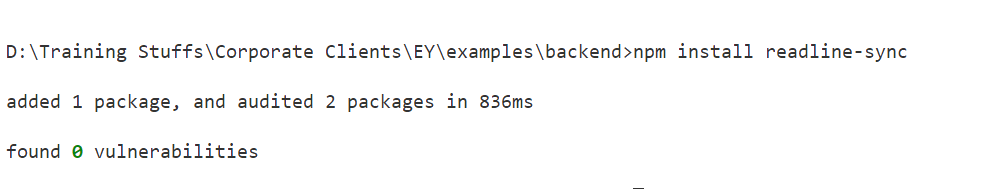
let input = readline.question(“Enter some input”);

let inputInt = readline.questionInt(“Enter some input):

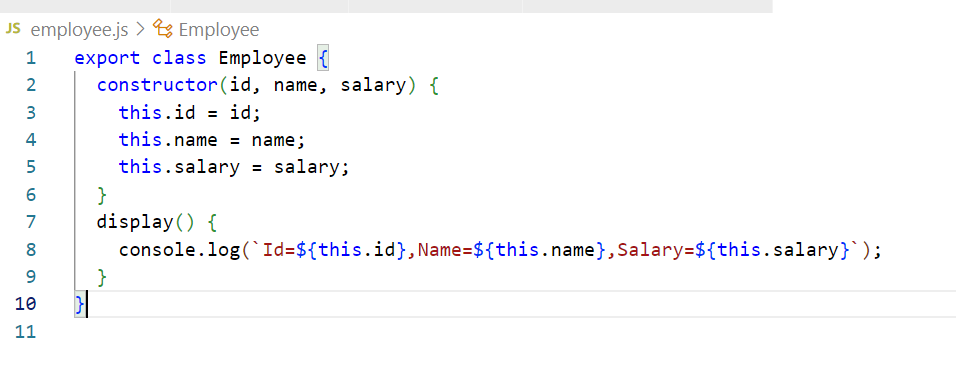
Local module: These are the modules which we create in our project like variables, classes, functions,

export class Employee { … }   
export class Customer { … }

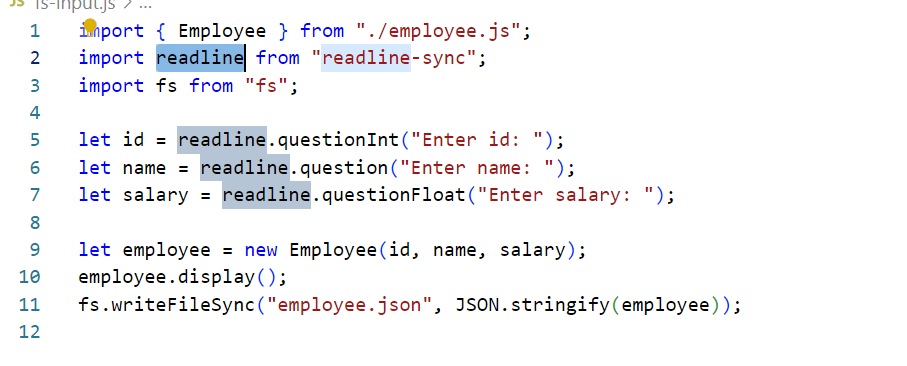
Installing readline-sync



employee.js



fs-input.js



Note: This code re-writes the new json with the older one

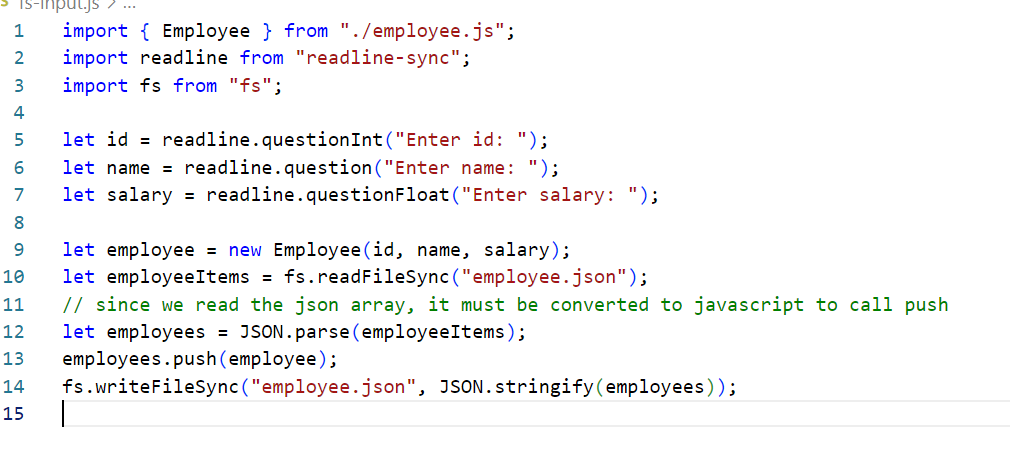
Activity: Write the program that can keep every employee object you store in the json file without erasing the old one

Array acts like Stack in JS & Node.js

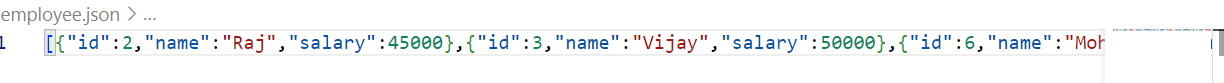
it has methods like

push(): to store the data

pop(): to remove the data in LIFO order

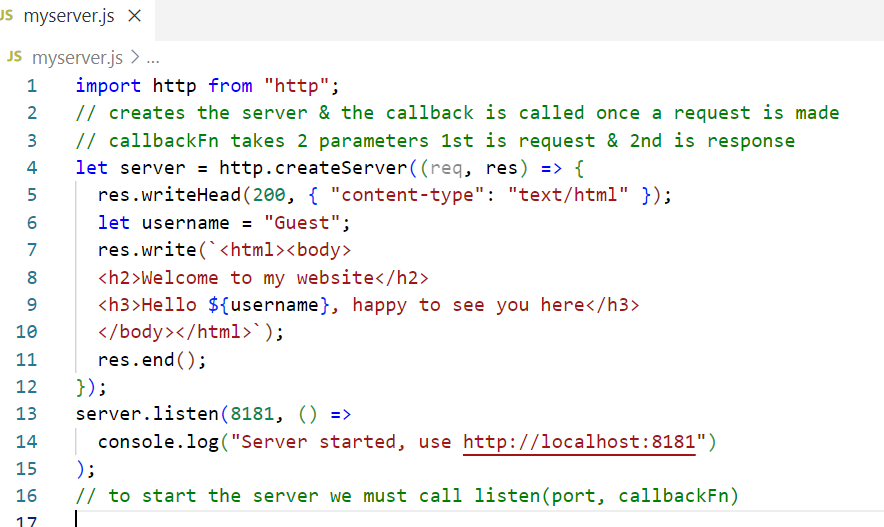


Output:



http module

This is an inbuilt module which helps you to create your own server and also develop a website that can have html, css & js files



Limitations of just using http

1. it helps you to create simple websites
2. it becomes complex when it has to serve different types of URL’s, because you need to categorize each URL’s inside the callback functions

Express Server

It internally uses http module, it is mainly used to develop REST based applications or REST webservices.

Rest Webservices:

It enables multiple technologies to exchange the data in a common format called JSON, these JSON are converted to the structure the application understands

ex:

Zomato can make payment through phone pay, however phone pay sends that amount to various bank webservices

So Zomato sends in JSON, phone pay takes that and converts to the structure it understands and passes the data and other related informations to the registered bank in JSON format

Rules while developing webservices

1. A webservice must have an URL
2. A webservice must be mapped to HTTP methods like GET, POST, PUT & DELETE, so that the consumers would use the same HTTP methods to send request.

GET: for retrieving the existing resource

POST: for creating a new resource

PUT: for updating the existing resource

DELETE: for deleting the existing resource

Note: GET request cannot send the data via the request body, it can send the data either in the URL or in the request headers.

Express is not available inbuilt, we must download using npm install express

We can import and create webservices mapped to various http methods

import express from ‘express’;

let app = express();

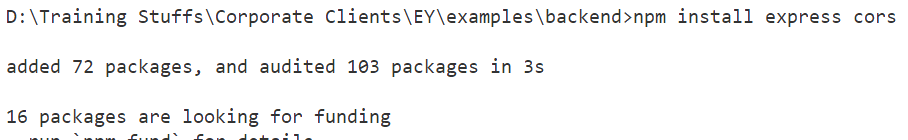
app.get(‘/profiles’, (req, res) => { handler for HTTP GET /profiles } );  
app.post(‘/profiles’, (req, res) => { handler for HTTP POST /profiles } );  
app.put(‘/profiles’, (req, res) => { handler for HTTP PUT /profiles });  
app.delete(‘/profiles’, (req, res) => { handler for HTTP DELETE /profiles });

You can also pass path parameters / query parameters using /:name

app.get(‘/profiles/:id’, (req, res) => { hander for HTTP GET /profiles/value });

Libraries to download

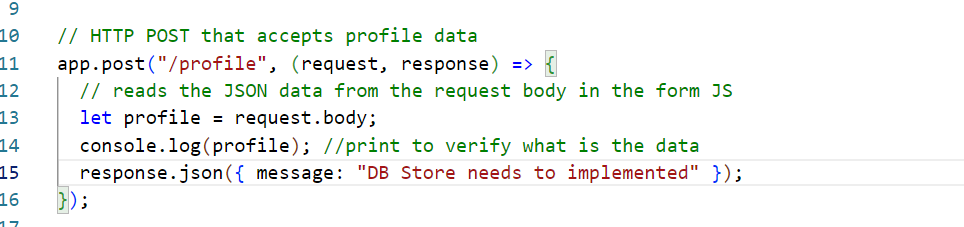
1. express
2. cors: Cross Origin Resource Sharing



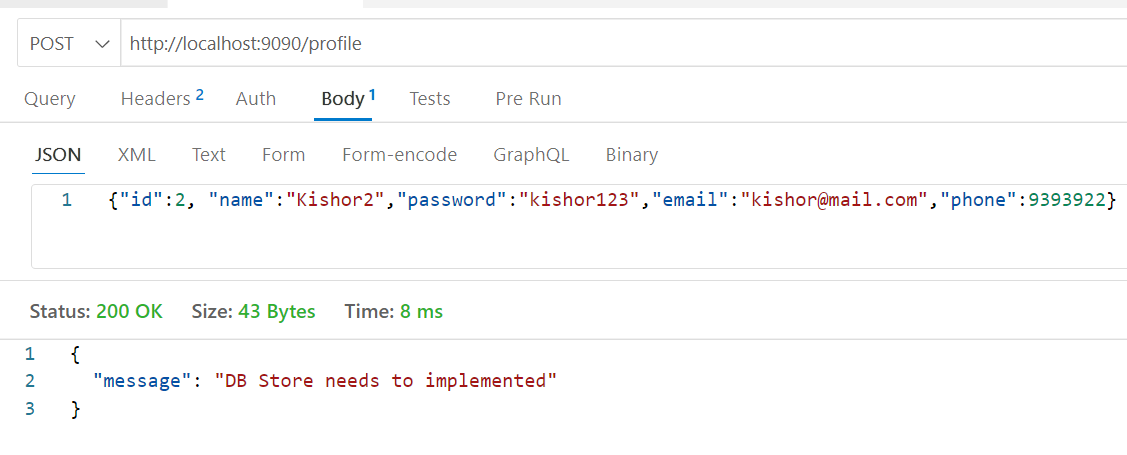
main.js



Webservice that accepts HTTP post with JSON data



Output:



Database

It is used to maintain the data, there are two types of database

1. Structured: RDBMS - table form - SQL (MySQL, Oracle, Sybase, Postgres & etc)
2. Non-Structured: NoSQL - Document form that is usually JSON format (MongoDB, Casandra & etc)

MongoDB: It is a NoSQL database which maintains the data in a document format usually a JSON, all these documents are stored in a table called collection, mongodb uses Javascript extensively, hence the mongo-terminal or mongo-shell supports javascript functions

Collection: It is a container like a table, that stores all the documents

Document: It is a JSON object

MongoDB provides lot of inbuilt Javascript functions to manipulate the database using the terminal, this mainly for administrator’s, however programmers can use mongodb library in different technologies to perform CRUD operations

Things to download

1. mongodb
2. mongo shell

Configurations

All the data is maintained in a folder data/db of your root directory be it C: drive or D: drive or root directory if its Unix variant, however if its windows we must specify the mongodb that it must maintain in the C or D drive of data/db folder.

How to create a collection

db.createCollection(“users”); // creates a users collection

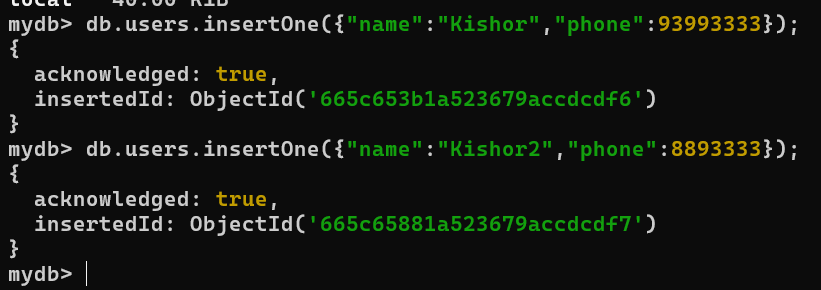
How to insert a document

db.users.insertOne( { json structure } ); // stores a document in users collection

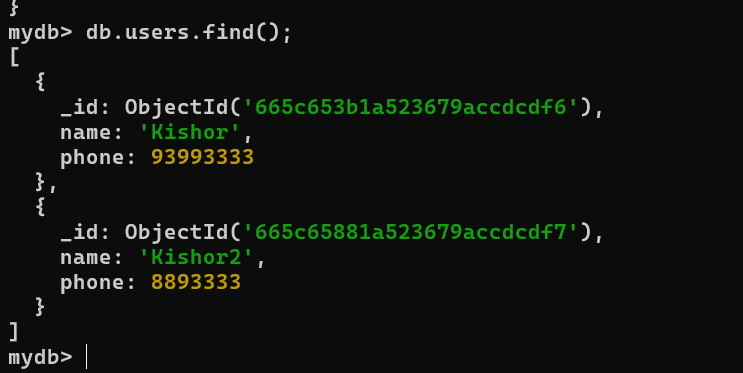
How to find a document

db.users.find(); // retrieves all the documents form the users

Storing the documents



Find all the documents



MongoDB provides inbuilt javascript functions which you can use even in the terminal/shell, those functions are

1. insertOne( { json documents } ): to store
2. find( { json filter } ): to read 0 or more documents
3. updateOne( { json filter }, { json document } ): to update 0 or more documents
4. deleteOne( {json filter}): to delete a document

To start the database: mongod

To connect to the database from the shell: mongosh

Create a collection called profiles to use in the case-study

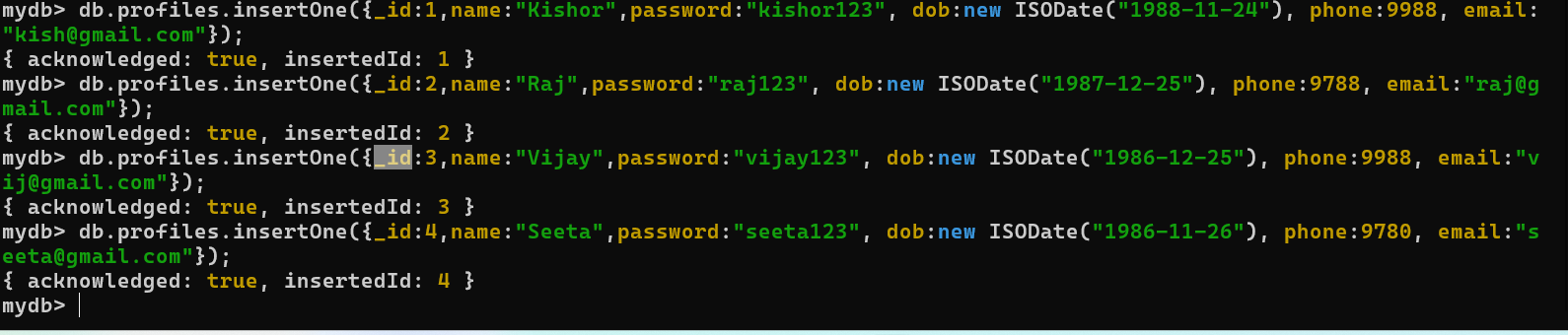
Since in the registration you are passing id, name, password, dob, phone, email, we must understand in mongodb \_id is used to uniquely recognize the primary key, so in React.js you can send the json with \_id or in node.js you can convert the id to \_id before storing

db.createCollection(“profiles”) : profiles will be created at the time of insertion if collection is not present in the db

db.profiles.insertOne( { json } ): profiles will be created if not present & will store the JSON

Performing CRUD operations using insertOne, deleteOne, updateOne, find() on profiles

Note: for Date you must follow ISO standard, hence you must use the following js function



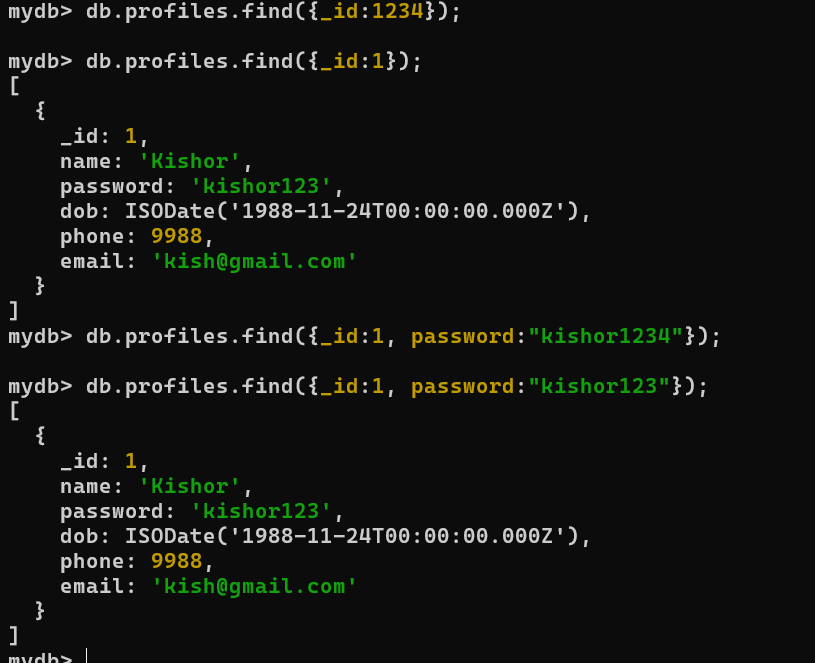
Note: You can use insertMany([ { }, { }, { }, { } ]) to insert multiple documents at once, you must wrap all the documents inside a [ ]



Finding a document using a filter

db.profiles.find( { \_id : 1 } ): Gets the document based on the matching \_id value, \_id is not present returns empty

db.profiles.find( { \_id : 1, password : “kishor123” } ): You can use these when you want to get the document from multiple filters

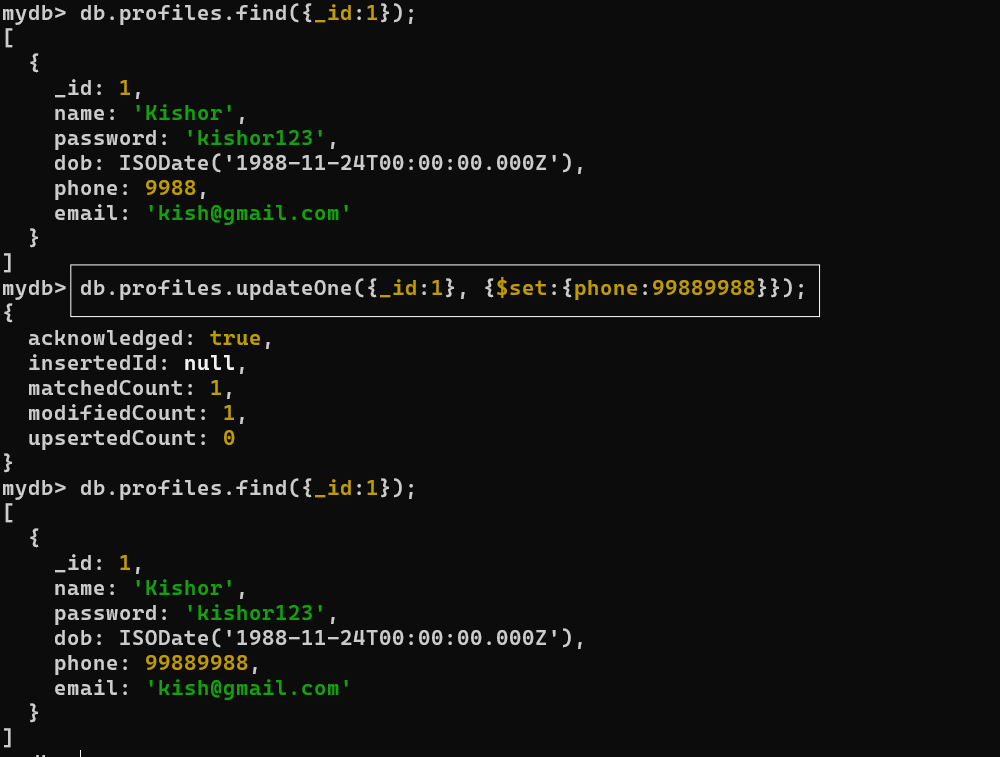


Updating the document based on the condition

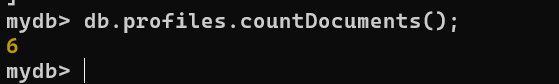
updateOne({filter}, {$set: {document} }):

example:

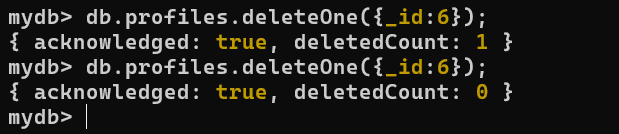
db.profiles.updateOne({\_id:1}, {$set : {phone:998877793} });



How to count documents

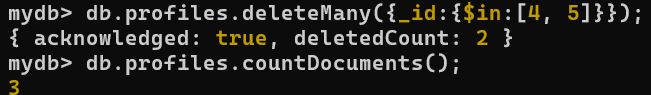


How to delete the documents

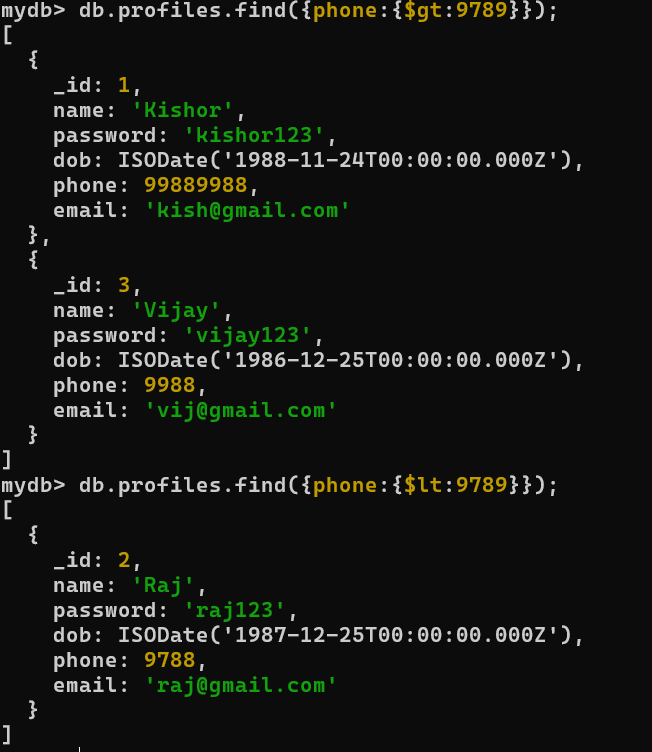
db.profiles.deleteOne({\_id:6});  


How to delete multiple documents based on id: you must use $in to choose multiple values

db.profiles.deleteMany({ \_id : { $in:[ …. ] }} );



$gt, $lt: These are used to mention conditions with greater or lesser values



You can see other operators in mongodb like $and, $or

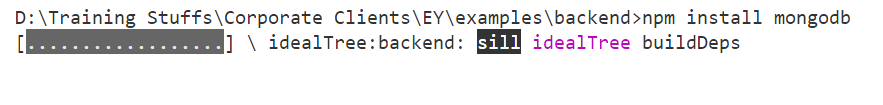
How to access the Mongodb using node.js

NPM registry has a mongodb library that provides inbuilt functions to perform CRUD operations, these functions are almost similar to the mongodb functions, however the application must first connect to the database and then perform the CRUD operations

Steps:

1. Install the mongodb library: npm install mongodb
2. You must connect to the mongodb using mongodb library
3. CRUD operations can be done using inbuilt functions provided by mongodb library

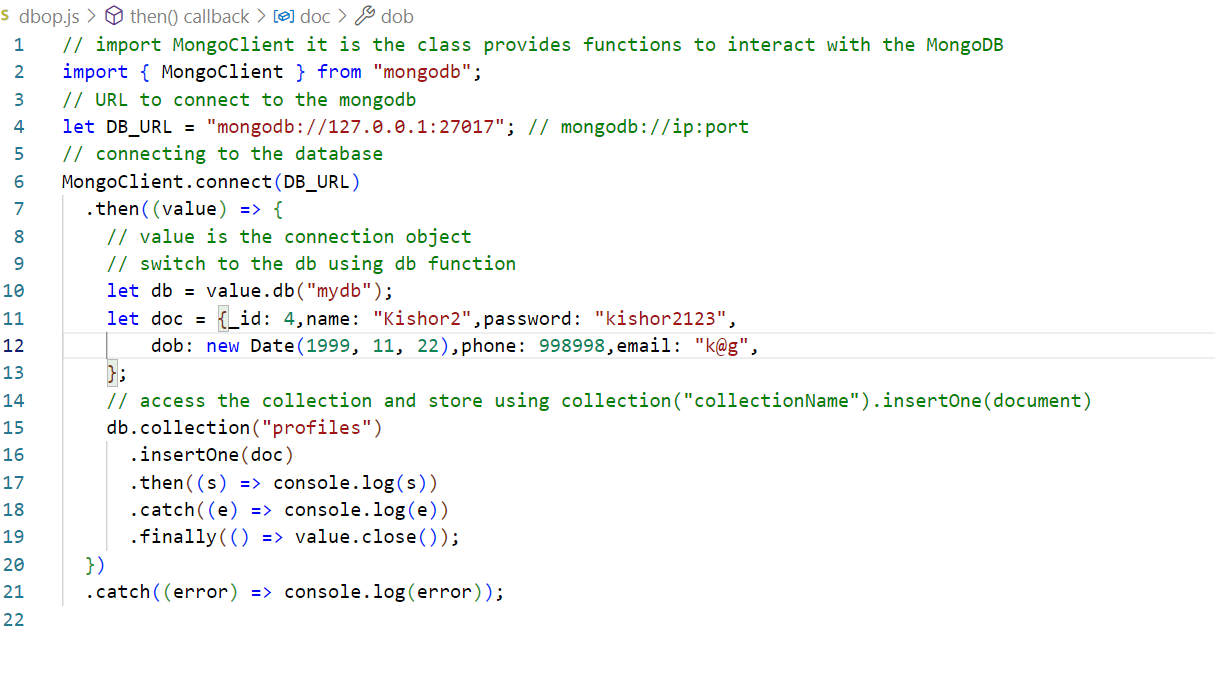
Installing mongodb library



MongoClient: It is a class that provides following functions

connect(URL): To connect to the database, it returns a Promise because it can be successful or rejected

Storing the document



Using async / await: Whenever you want to handle the promise by waiting for the promise to be resolved/rejected you can use async / await

With Promise

function dbConnect(url) {   
 MongoClient.connect(url).then( callback ).catch(callback);  
}

With Async / await

async function dbConnect(url) {   
 return await MongoClient.connect(url)  
}

With Promise performing insertOne

function dbConnect(url) {   
 MongoClient.connect(url)

.then( (value) => {  
 let db = value.db(“mydb”);  
 db.collection(“profiles”).insertOne(doc).then(callback).catch(callback)

} ).catch( err => { … } );  
}

With Async/Await performing insertOne

async function dbConnect(url) {   
 return await MongoClient.connect(url);  
 }  
async function store(doc) {  
 let value = dbConnect(url);  
 let db = value.db(“mydb”);  
 return await db.collection(“profiles”).insertOne(doc);  
}

Source Code to connect & store

import { MongoClient } from "mongodb";

let DB\_URL = "mongodb://127.0.0.1:27017";

// to connect to the db, returns the Promise

async function dbconnection(url) {

  let con = null;

  try {

    con = await MongoClient.connect(url);

  } catch (e) {

    throw "DB Connection failed";

  }

  return con;

}

// CRUD operations : store, find, delete & update

// storing the document by writing a reusable store

export async function store(doc) {

  // dbconnection(DB\_URL).then().catch() [or]

  // try { await dbconnection(DB\_URL)} catch(err) { }

  let value = null;

  try {

    value = await dbconnection(DB\_URL);

    let db = value.db("mydb");

    return await db.collection("profiles").insertOne(doc);

  } catch (e) {

    throw e;

  } finally {

    value.close();

  }

}

// now invoke store like store({}).then().catch() or try { store({}) }

Source code to test

import { store } from "./dbprofile.js"; // extracting the db operations

store({

  \_id: 10,

  name: "Alex",

  phone: 9922334,

  password: "alex123",

  dob: new Date("1990-10-15"),

  email: "alex@g",

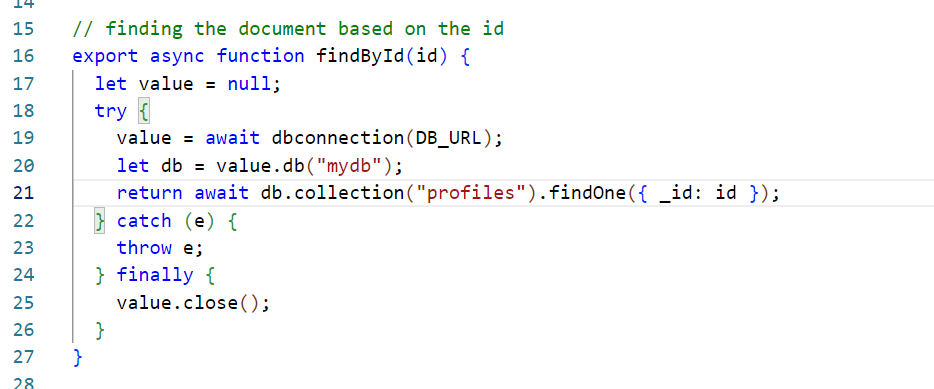
})

  .then((result) => console.log(result))

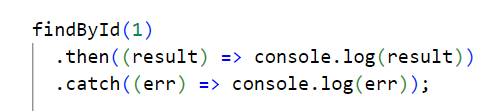
  .catch((err) => console.log(err));

Reading the document based on id

In mongodb library you get findOne function where you pass filter



Testing



Activity:

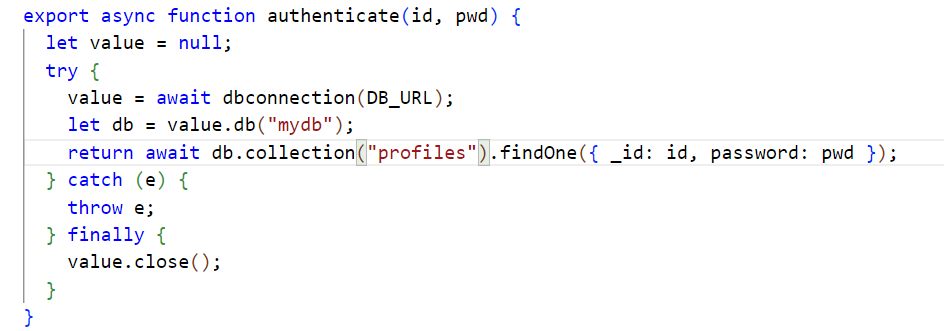
1. Create a function that finds all the documents & returns the promise
2. Create a function that accepts id and password and returns the promise
3. Create a function that updates the phone based on the \_id value & returns the promise
4. Create a function that deletes the document based on the \_id value & returns the promise

Call all the 4 methods from testingCRUD.js

Finding all the documents



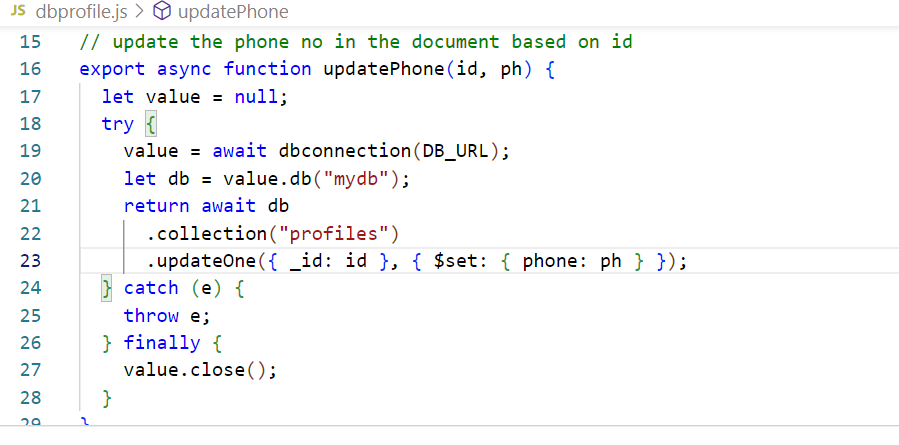
Authenticate based on id & password



Output:



Updating the document



Similar to the Mongodb you can also interact with SQL databases, for that you need to download the specific libraries like mysql2, oracledb, postgres

SQL will have insert, update, delete queries, these libraries give the methods with the same names.

Installing mysql2



Source code of insert and select on MySQL database

import mysql from "mysql2/promise";

let connection = undefined;

try {

  connection = mysql.createPool({

    host: "localhost",

    user: "root",

    password: "root",

    database: "mydb",

    port: 3306,

    waitForConnections: true,

    connectionLimit: 10,

    maxIdle: 10, // max idle connections, the default value is the same as `connectionLimit`

    idleTimeout: 10000, // idle connections timeout, in milliseconds, the default value 60000

    queueLimit: 0,

    enableKeepAlive: true,

    keepAliveInitialDelay: 0,

  });

} catch (err) {

  console.log(err);

}

async function store(id, name) {

  const [result, fields] = await connection.execute(

    "insert into test(id, name) values(?,?)",

    [id, name]

  );

  return result;

}

async function findById(id) {

  const [result, fields] = await connection.execute(

    "select \*  from test where id = ?",

    [id]

  );

  return result;

}

findById(2).then((r) => console.log(r));

// store(6, "Six")

//   .then((r) => console.log(r))

//   .catch((e) => console.log(e));

To see more operations on mysql follow the instructions in the official document

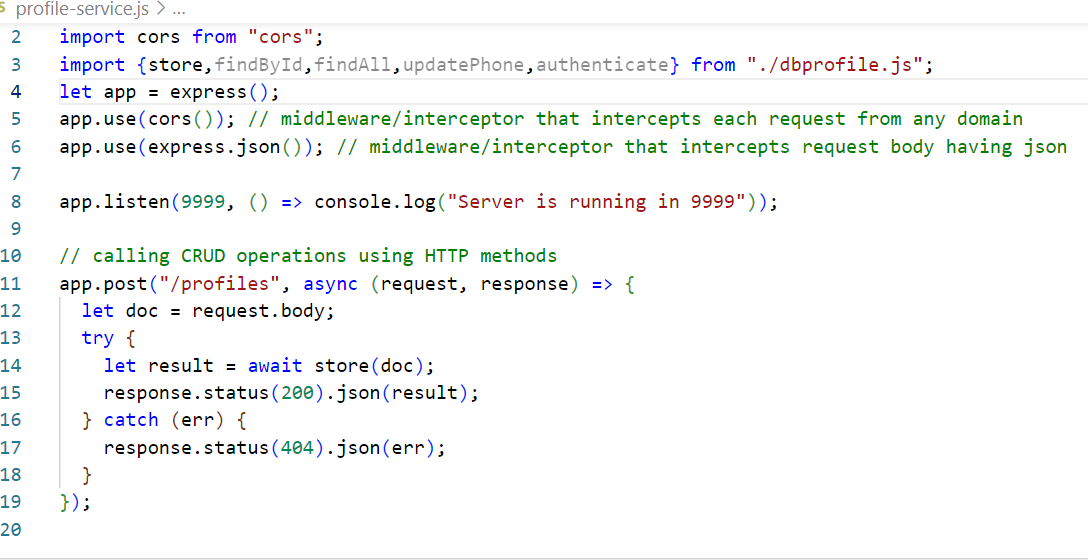
<https://sidorares.github.io/node-mysql2/docs>

For mongodb

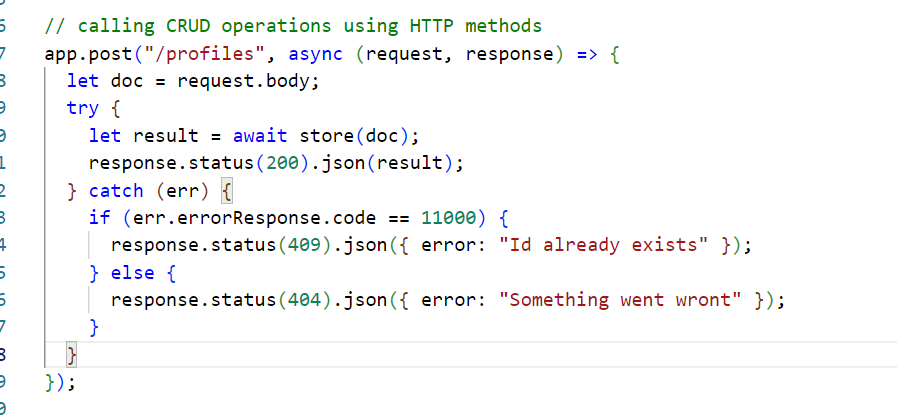
<https://www.mongodb.com/docs/drivers/node/current/quick-start/>

How to access database from the react

1. We must create webservices at the backend
2. These webservices must reuse the database logics i.e., extract the db logics and respond to the clients request.
3. Then React can use axios to access the webservice



Error responses for duplicate keys



From react how to post the data

axios.post(URL, jsonData).then(…).catch(…)



Login to the dashboard by entering id & password

When you enter id & password to query the database then we need to send the request using HTTP GET, however GET doesn’t support body, so we must pass the id & password either in the

1. URL: visible to the client
2. Authorization header (base64 encoded format)

To encode we need a Buffer library that provides a method to convert the text into base64 format and at the backend we will be using same base64 to decode.

\_id=1, password=alex123

Converting this to base 64

import {Buffer} from ‘buffer’;  
const encoded = Buffer.from(\_id + ‘:’+password).toString(“base64”)

Passing this in the Authorization header: We must use a Basic type Authorization, it is understood by HTTP, this is a header in the request

headers : { Authorization: “Basic “+encoded }

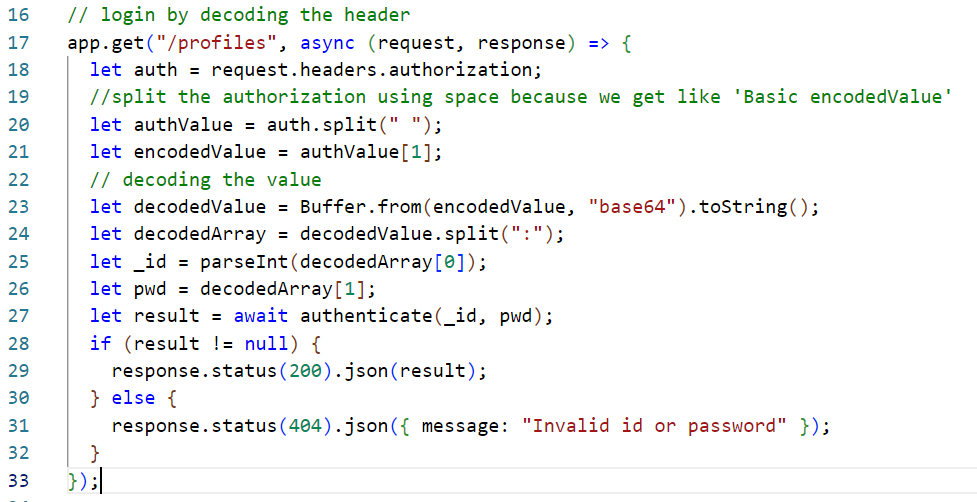
axios.get(URL, headers).then(..).catch(…)

In the backend the authorization header will be received, that needs to be extracted and decoded to get the original \_id & password

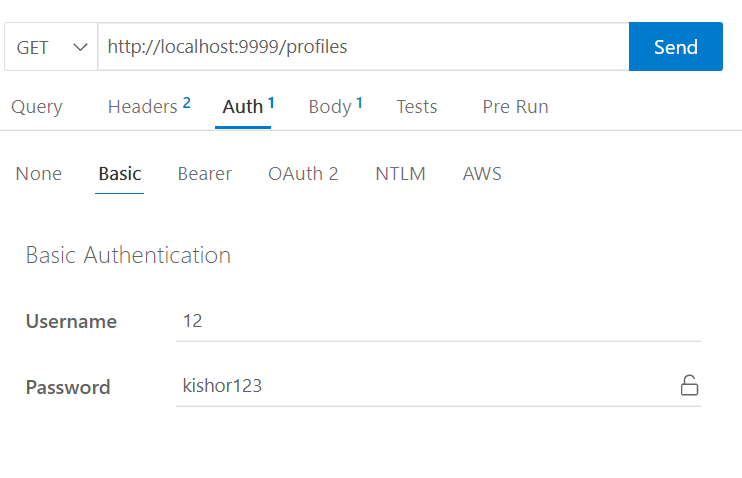
Things to implement

1. Login method at the backend that reads the authorization header
2. From the front-end encode the data and send in the authorization header

Backend code to print the extract header authorization



How to send authorization header in thunder-client



From react you must pass the authorization header using the header property like

axios.get(URL, { headers : { Authorization: “Basic “+encodedValue}})

To encode we must download Buffer library which is not present in react



In the ProfileLogin component add the Authorization header by reading the \_id & password

Import the Buffer in the CaseStudy.jsx



ProfileLogin code



Source code

export function ProfileLogin() {

  let [profileId, setProfileId] = useState("");

  let [password, setPassword] = useState("");

  // a code to test the axios library using fake api

  let handleClick = (e) => {

    let FAKE\_URL = "https://jsonplaceholder.typicode.com/users";

    axios

      .get(FAKE\_URL)

      .then((res) => console.log(res.data))

      .catch((err) => console.log(err));

  };

  // useNavigate() to navigate to the components programmatically

  let navigate = useNavigate();

  let handleSubmit = (e) => {

    e.preventDefault(); // inbuilt function that stops reloading the page on submit

    // encode the profileId & password using Buffer in base64

    let encodedValue = Buffer.from(profileId + ":" + password).toString("base64");

    let URL = "http://localhost:9999/profiles";

    // call the webservice by passing this encodedValue in the authorization header

    // get(URL, headers)

    axios

      .get(URL, { headers: { Authorization: "Basic " + encodedValue } })

      .then((r) => navigate("/success/" + profileId))

      .catch((e) => alert(e.response.data.message));

  };

  return (

    <div>

      <Link to="/register">Create Account</Link>

      <h3>Login Form</h3>

      <form onSubmit={handleSubmit}>

        <div>

          <input

            value={profileId}

            className="form-control w-25"

            type="number"

            placeholder="Enter profile id"

            name="profileId"

            onChange={(e) => setProfileId(e.target.value)}

            required

          />

        </div>

        <div>

          <input

            value={password}

            className="form-control w-25"

            type="password"

            placeholder="Enter password"

            name="password"

            onChange={(e) => setPassword(e.target.value)}

            required

          />

        </div>

        <div>

          <input

            className="btn btn-secondary me-1"

            type="submit"

            value="Login"

          />

          <input

            className="btn btn-secondary me-1"

            type="reset"

            value="Reset"

          />

        </div>

      </form>

      <div>

        <h3>Fake API Testing</h3>

        <button className="btn btn-secondary" onClick={handleClick}>

          Users

        </button>

      </div>

    </div>

  );

}

Topics that needs to be learnt in React.js

HOC: Higher Order Components : A component that gives another enhanced component

Use Context

Use Ref

Debugging

Dynamic Input

Life cycle methods

Topics that needs to be learning in node.js

1. View Engine i.e., Jade Template
2. Buffer & Streams

Enhancing the case - study

When you login you will go to success page, from the success component you must perform following operations

1. Add Contacts: You must add your friends contact name & number to your profile document
2. View Contacts: you must view all the contacts of a logged in profile
3. Delete Contacts: you must able to delete a particular contact present in the profile
4. Edit Profile - change password
5. Sign out