Javascript & React.js

Software’s requirement

* VS Code
* Node.js

Javascript: To add interactivity to your website, things you can do with javascript

* Input validation
* Event Handling
* Pop-up boxes

New features of Javascript (ES6 new features)

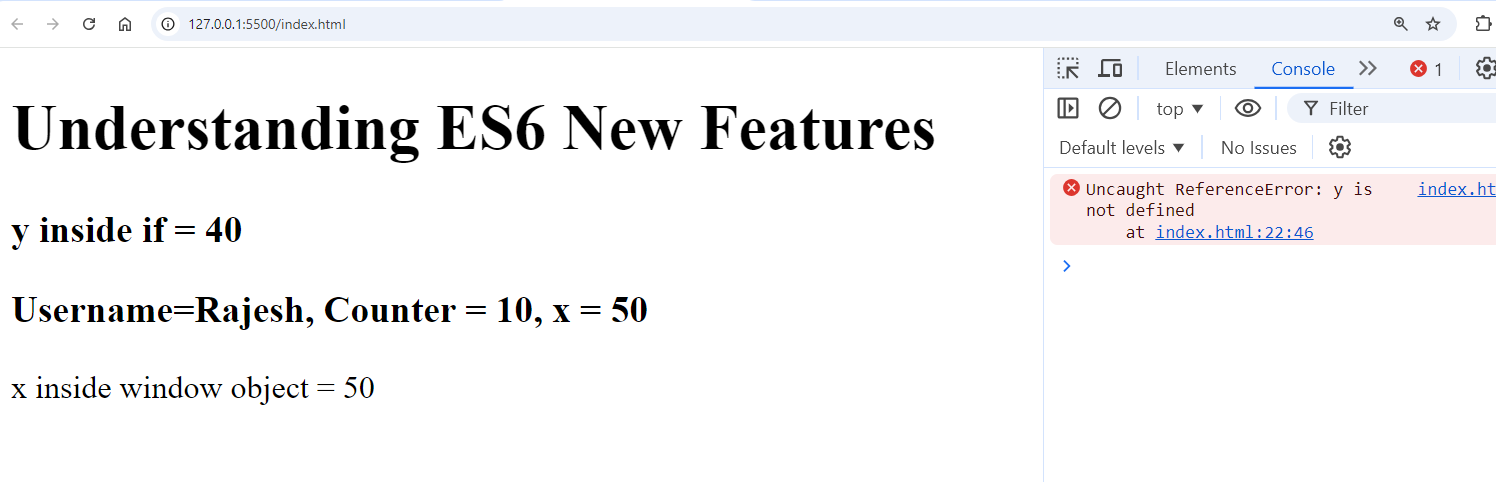
Helps to easily write the javascript code, it added lot of modern language syntax

* Keywords like let, const, class, super, extends
* Template strings : ``
* Arrow functions
* Spread & Rest Operators
* Promises
* Async / Await
* Optional Chain(?.)

let & const: They are used to create variables, but earlier javascript used var



Output:



let variables you can modify, const variables you can’t modify

let x = 20;

x = 30; // ok

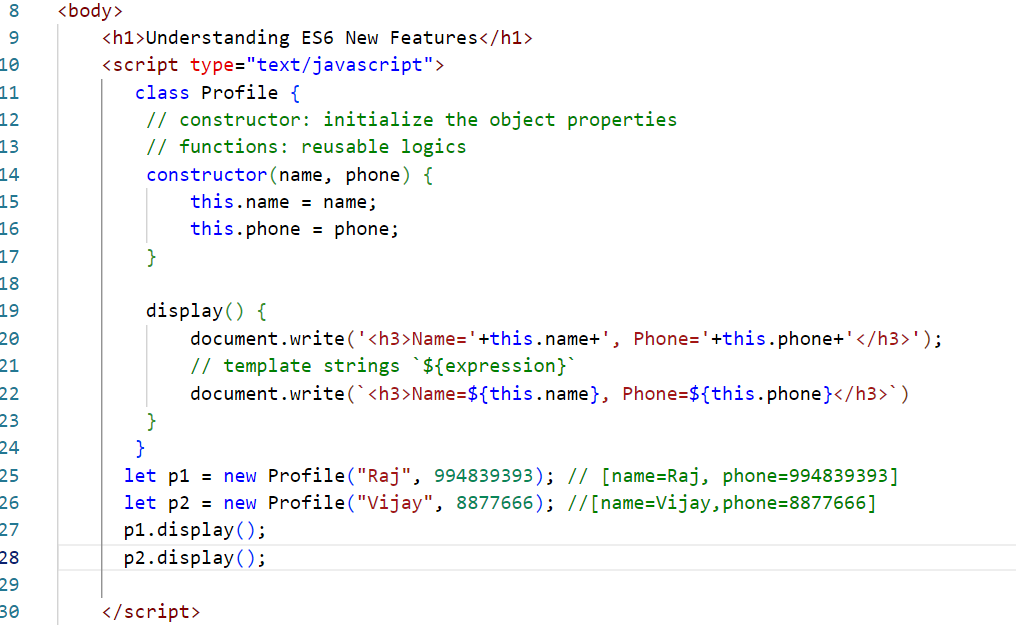
const y = 50;

y = 55; // error

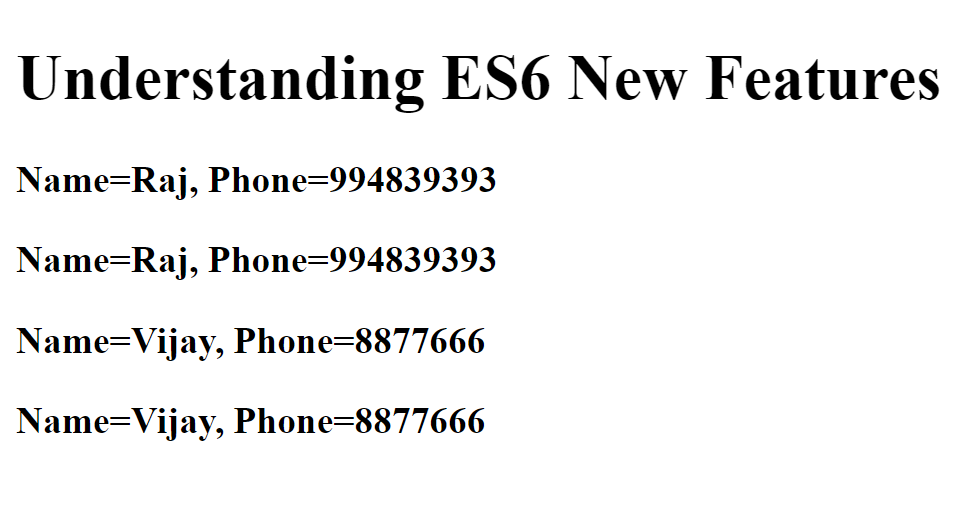
class:

It is a blueprint of an object, which tells the structure of the object.

Understanding the classes & objects

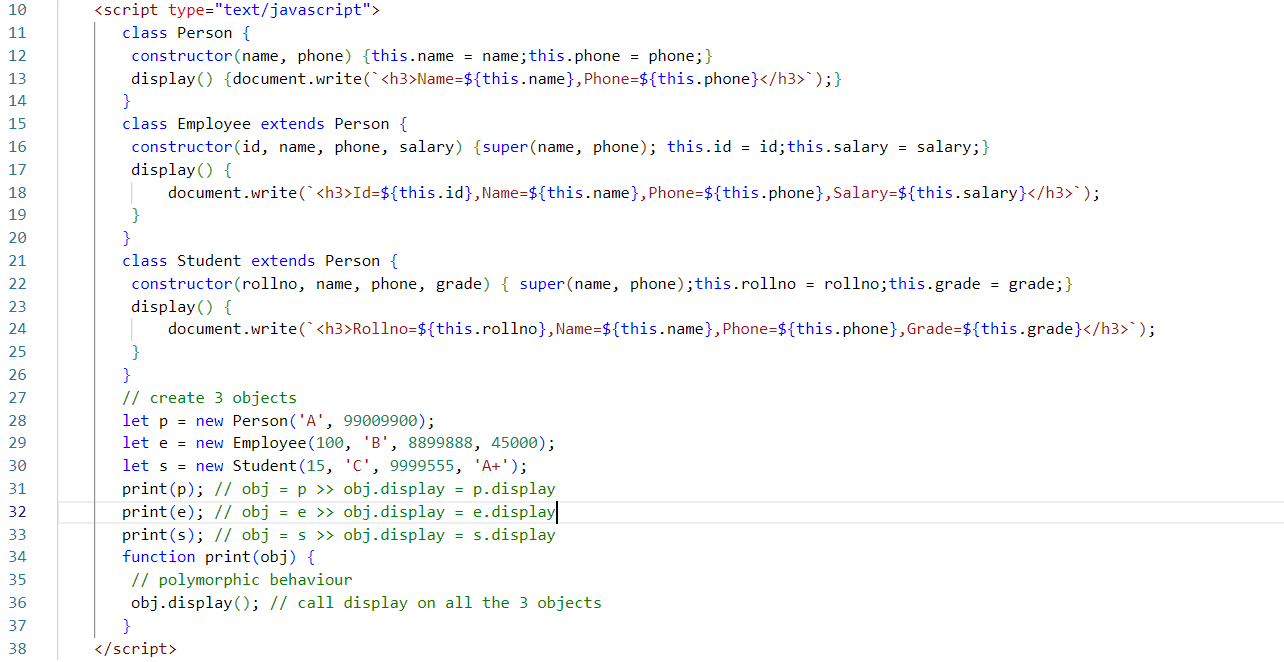


Output:



Inheritance

Process of acquiring the properties & behaviors from one object to another object.

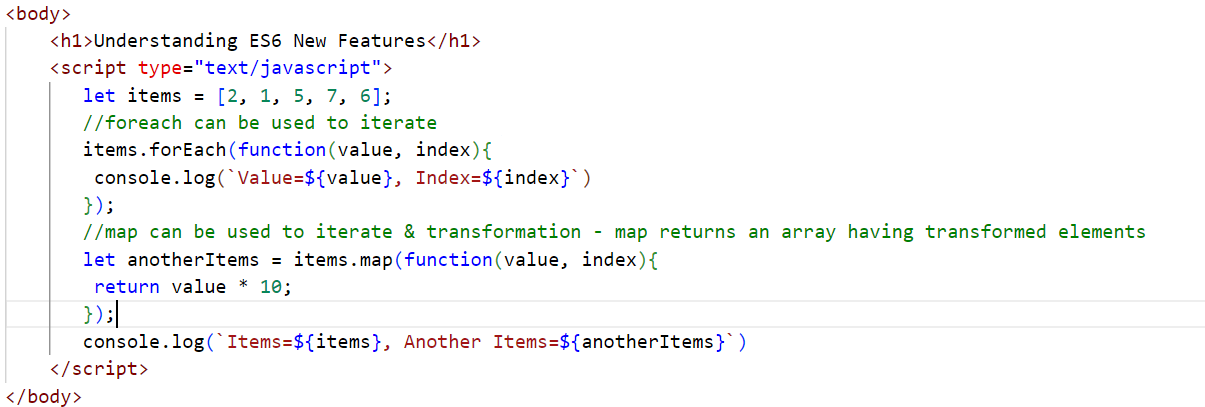


Output:

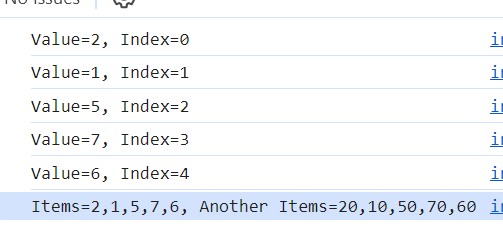


Callback functions

These are the functions which are invoked later depending on some events, like response from the server, events from user interaction, when timeout happens



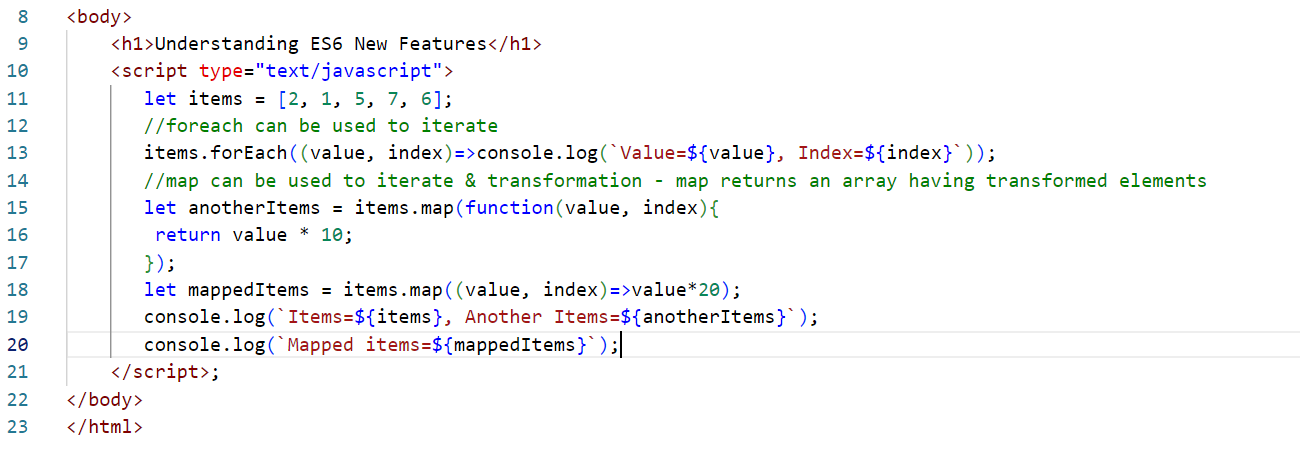
Output:



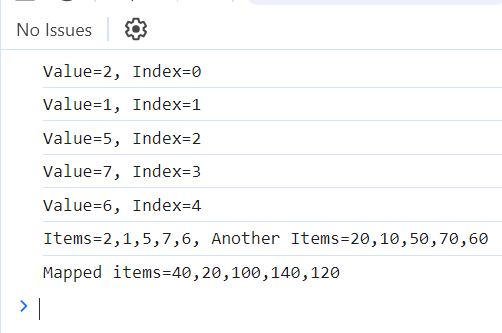
Arrow functions: These are the alternate form for the callbacks, it also simplifies the syntax

Callback:  
function(x, y) {   
 return x \* y;  
}  
Arrow function:  
(x, y) => { return x \* y ; }  
(x, y) => x \* y;

(x, y) => console.log(x, y); // this doesn’t return any thing



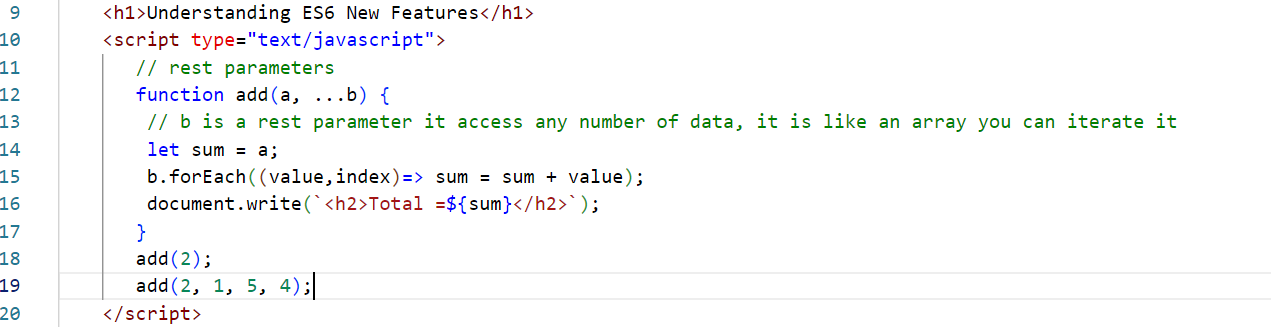
Output:



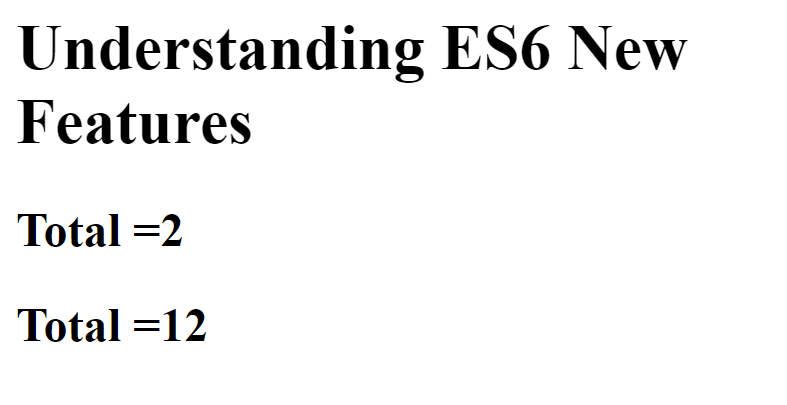
Rest & Spread operators

function add(a, …b) {   
}

Rest parameter can accept 0 or more elements



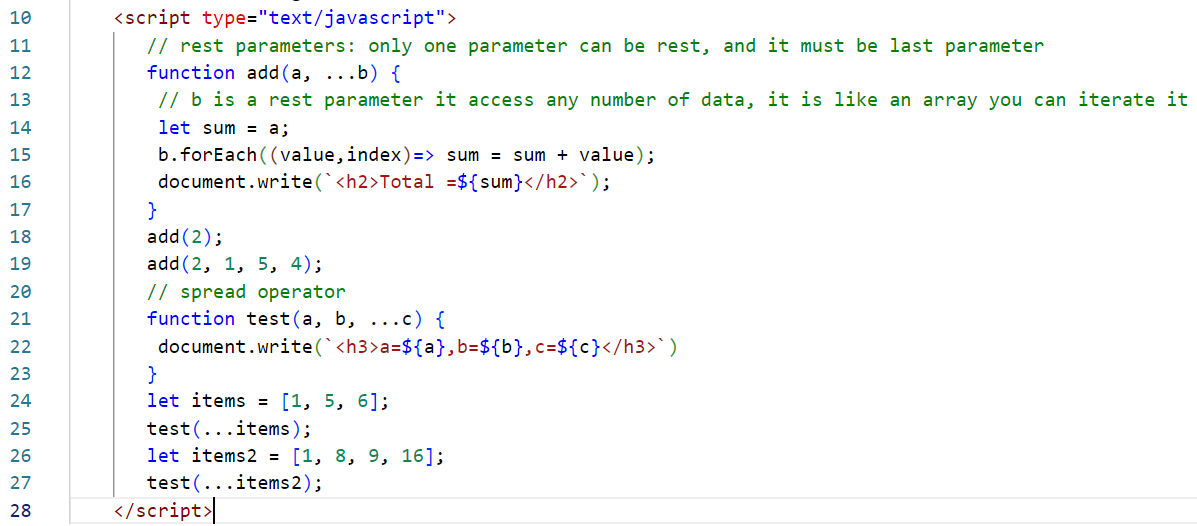
Output:



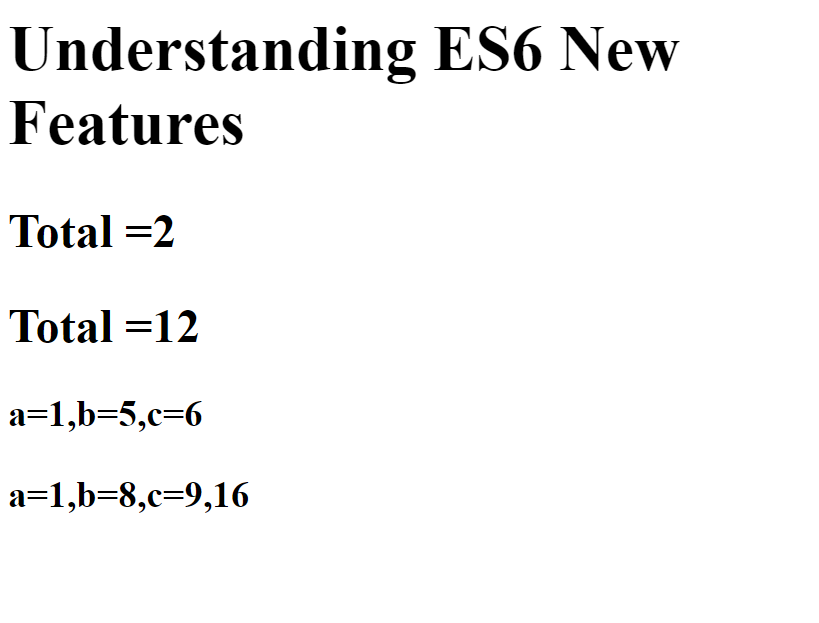
Spread parameter:  
It helps in distributing the elements into multiple parameters

let items = [2, 5, 6];

function test(a, b, c) { }  
test(…items); // a = 2, b = 5, c = 6



Output:



Destructuring: This helps in unwrapping the elements in the objects or arrays into another variables.

* Object Destructuring
* Array Destructuring

old approach

employee = {id:100, name: ”Rajesh”, salary:45000, email:”raj@g”, address : {st:”ka”,ct:”bl”} } ;

let id = employee.id;  
let name = employee.name;  
let salary = employee.salary;  
let email = employee.email;

New approach: Object destructuring

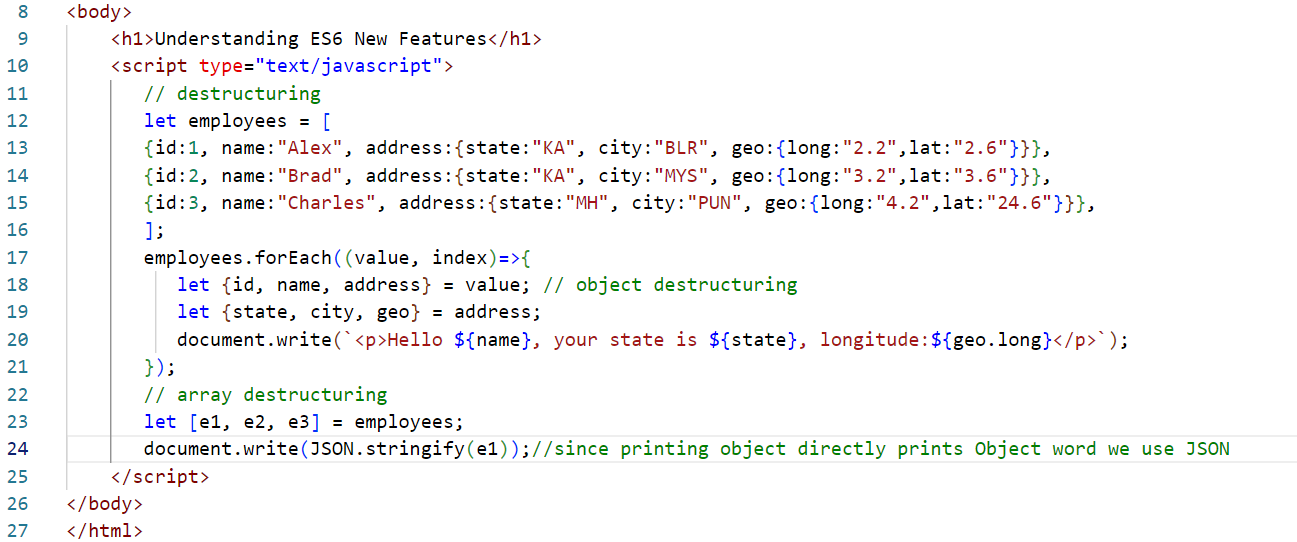
let { id, name, salary, email, address } = employee;   
let {st, ct } = address;  
console.log(st); // ka

Array Destructuring

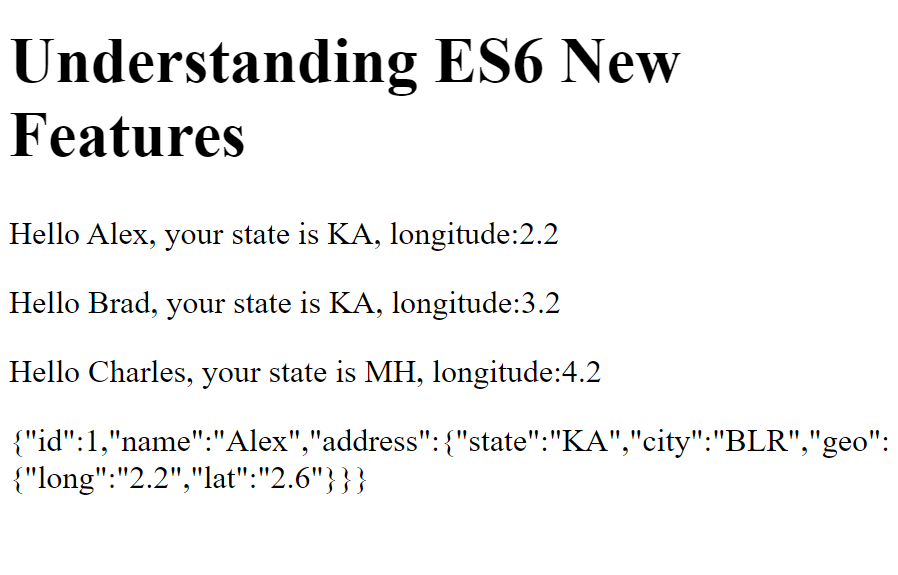
Old approach uses index

let iplTeams = [“RCB”, “CSK”, “KKR”, “GT”];  
let r = iplTeams[0];  
let c = iplTeams[1];

New approach uses Destructuring  
let [a, b, c, d] = iplTeams;

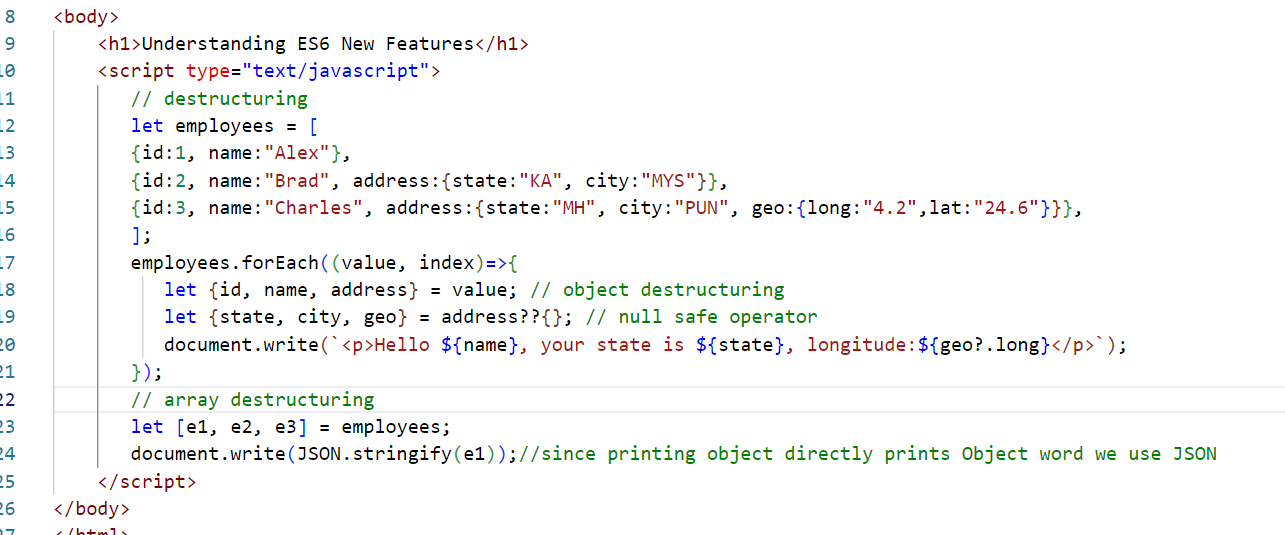


Output

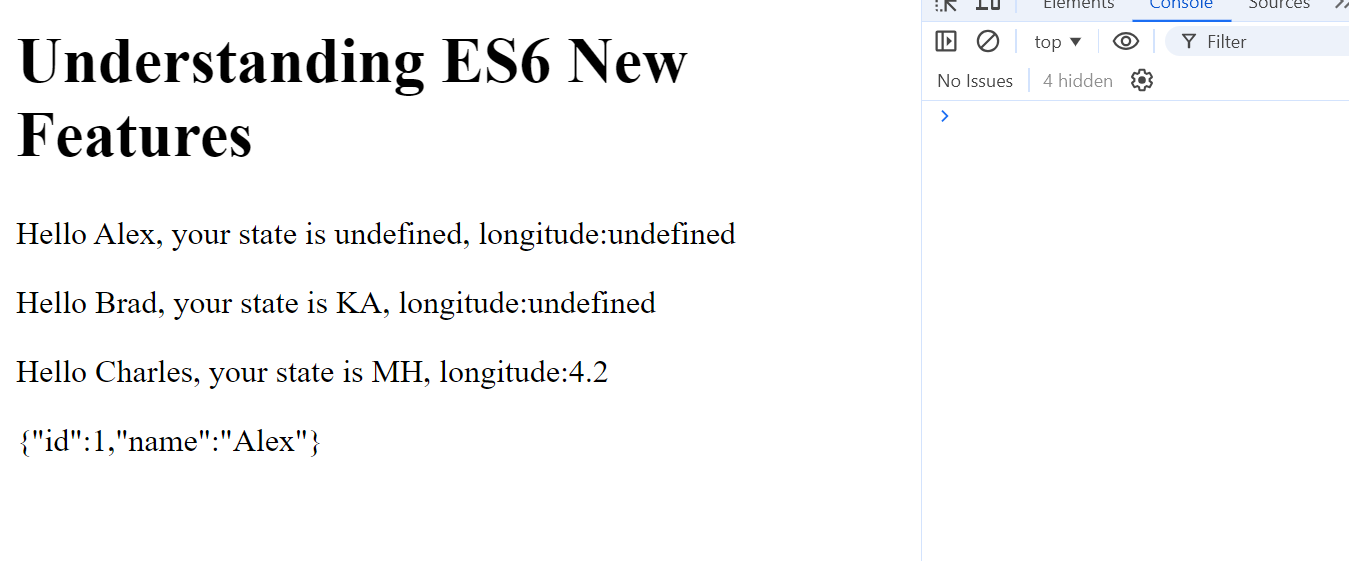


Optional Chain

Whenever you access nested properties you need to be careful because there could be chance of type errors

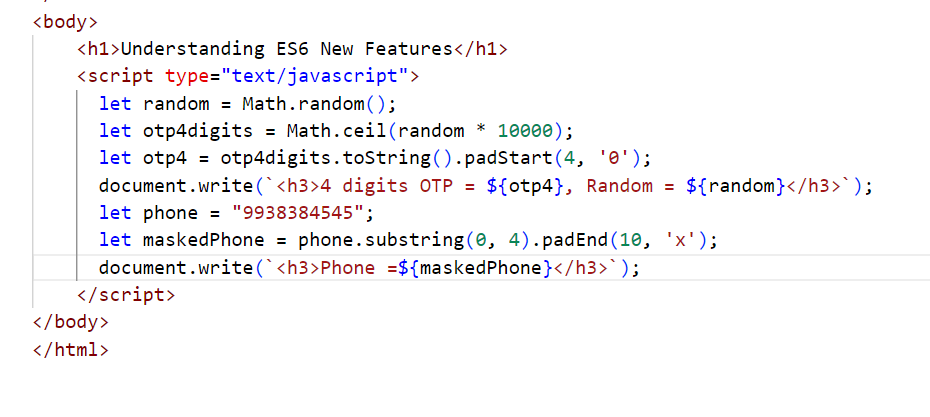


Output:



String padding start & end

9998889xxxx  
xxxx88989



Output:



canvas element:

It is mainly to create 2d shapes

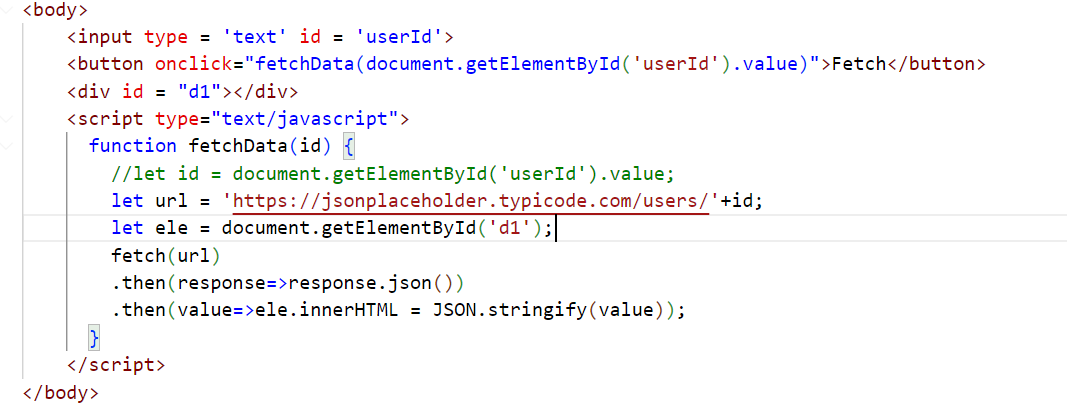
<canvas width=”400” height=”400” id = “c1”></canvas>

Promises

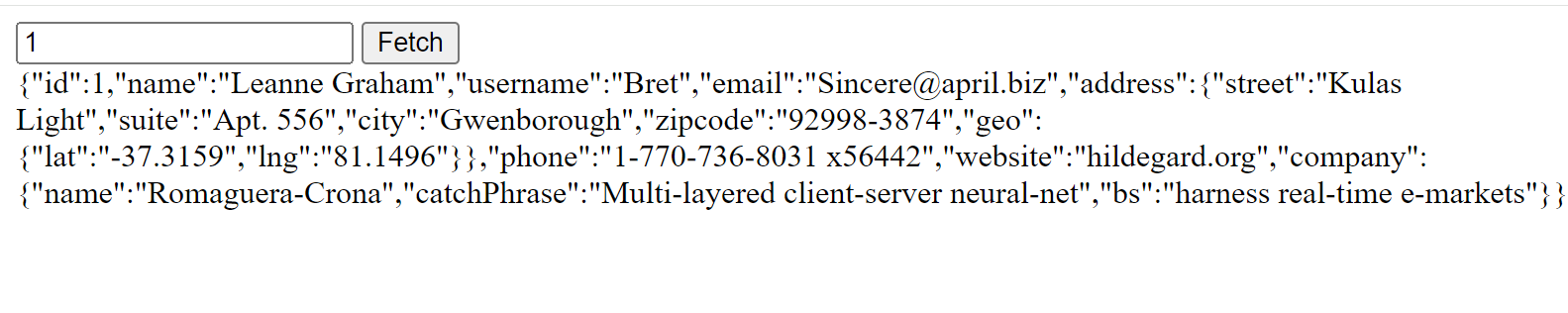
These are objects that make asynchronous requests and gets the response which can be either success or errors, if success its called promise is resolved, if error its called promise is rejected

You will use two methods of promise that takes callback functions as parameters

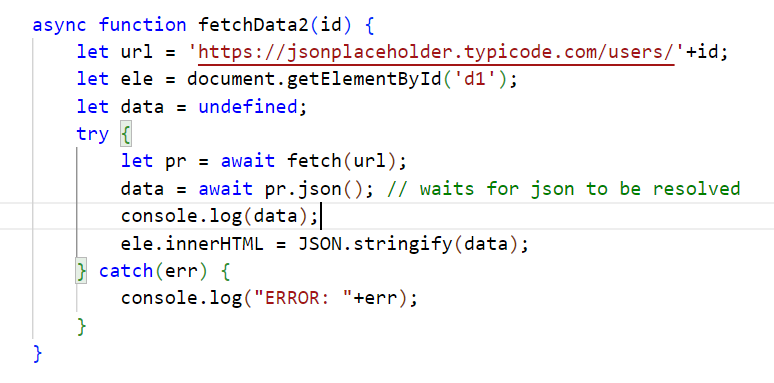
.then(callback)  
.catch(callback)



Output:



Async/Await



setters & getters

set & get keyword is used to create write & read functions

React.js

It is used to develop Single Page Applications(SPA), it is a Javascript library that helps you to create rich UI’s for your web page.

SPA:

Everything happens in one page, any changes you make that modifies only part of the page without reloading entire page

Components: These are independent UI’s which you can reuse in any part of the page.

Technologies react uses

1. HTML
2. JSX: Javascript XML: It makes you write HTML easily in the Javascript

Software’s required

1. VS Code
2. Babel toolkit -> Converts JSX to Javascript
3. Node.js & NPM

Libraries

Below are the libraries React.js uses

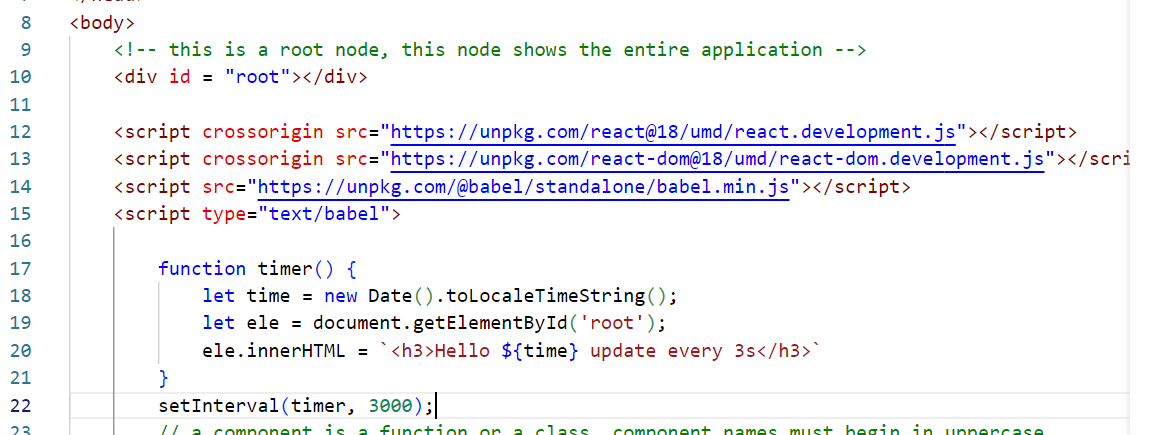
1. React: It is to create components & other react related features
2. React DOM: It is a virtual DOM
3. Babel: It is a transpiler to convert JSX to Javascript



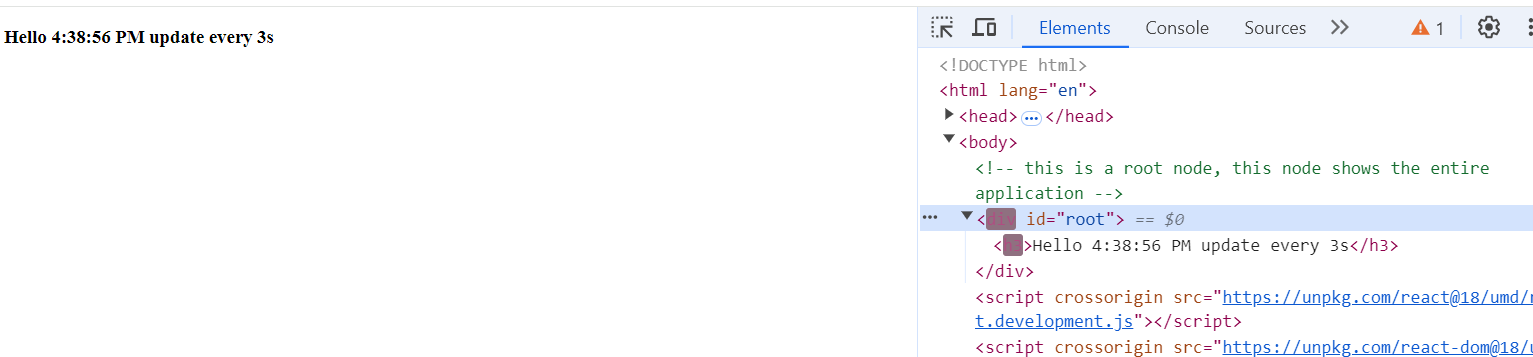
Virtual DOM

React uses this Virtual DOM to compare the content with the Real DOM of the browser and update only the changes required without reloading the DOM tree.

Without Virtual DOM



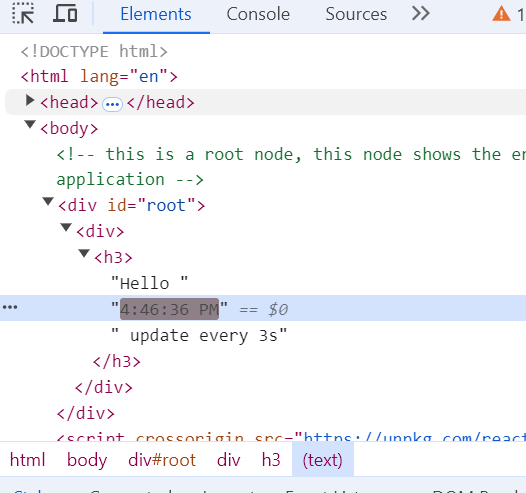
Output: You see entire DOM tree getting reloaded



With Virtual DOM



Output: You will see only time part is updating not the entire div or h3

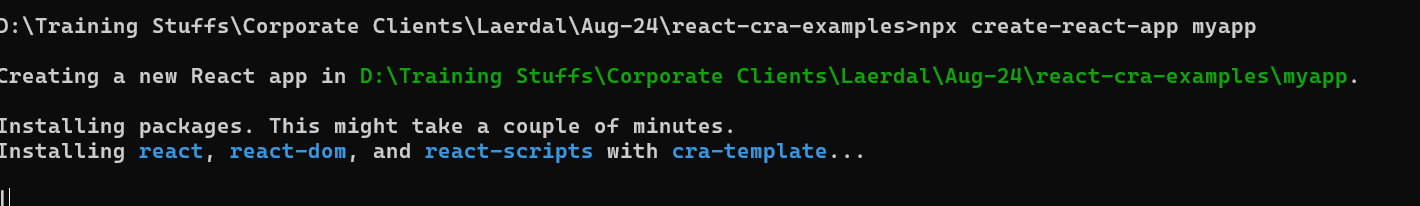


React Tool Kit

It is used to create an industry standard react application, it gives inbuilt structure like

* node\_modules: to maintain all the libraries
* src: to keep all the source code
* package.json: to keep the project dependencies & commands to maintain the application

npx create-react-app myapp



React Tool Kit

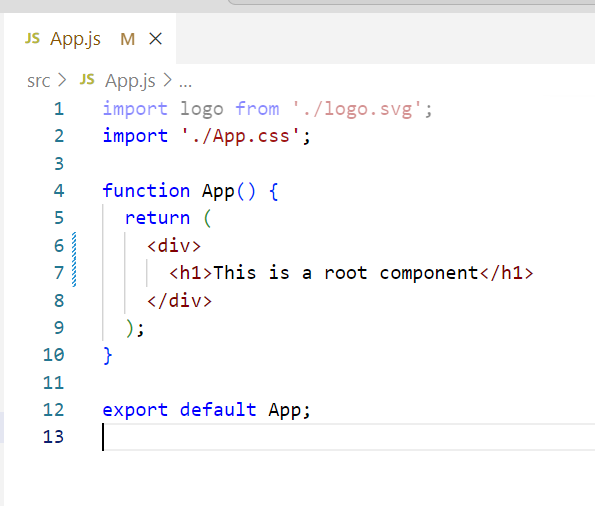
* It helps you to quickly create industry standard react applications
* It will have inbuilt live server which can auto-detect the changes
* It will have transpiler to convert JSX to Javascript
* It will be having GIT configuration as well, it will be in version control
* It will have project structure that separates HTML, CSS, Javascript, Libraries
* It will have package.json file that maintains application configurations

Command:

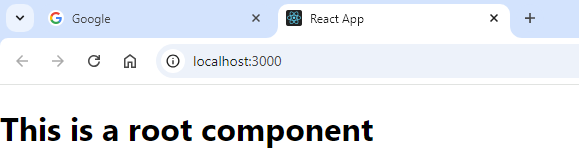
npx create-react-app myapp1

npm start: This starts the react application

Change in App.js



Output:

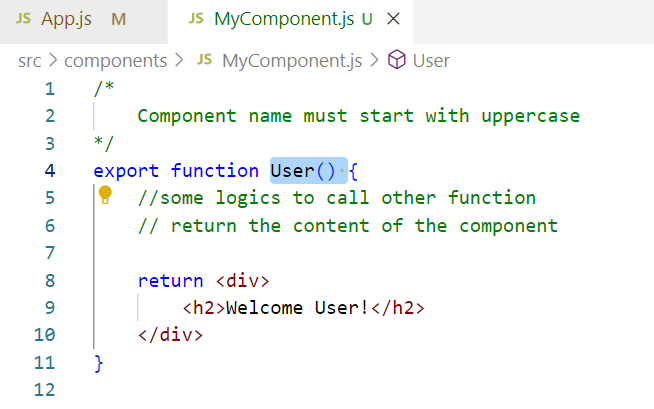


Let use custom components

There are two ways you can create components

1. Through classes that extends React.Component
2. Through function

src/MyComponent.js



src/App.js

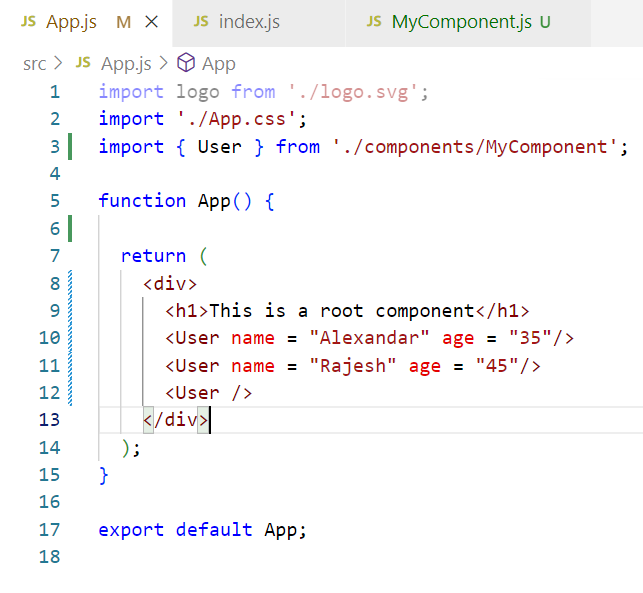


Props: These are input given to the components from another component, i.e., passing data from one component to another component

<User name = “Alex” age = 35 /> // props = {name:Alex, age:35 }  
<User name = “Rajesh” age = 45 /> // props = {name:Rajesh, age:45}  
 <User /> // props = undefined

function User(props) {   
 let name = props.name;  
 let age = props.age  
}

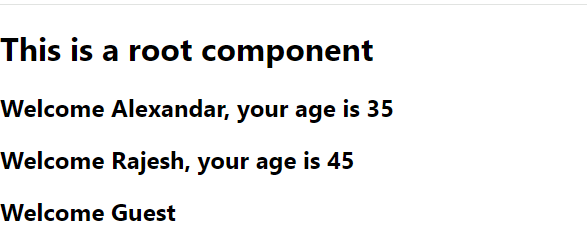
Passing props



Conditional rendering



Output:



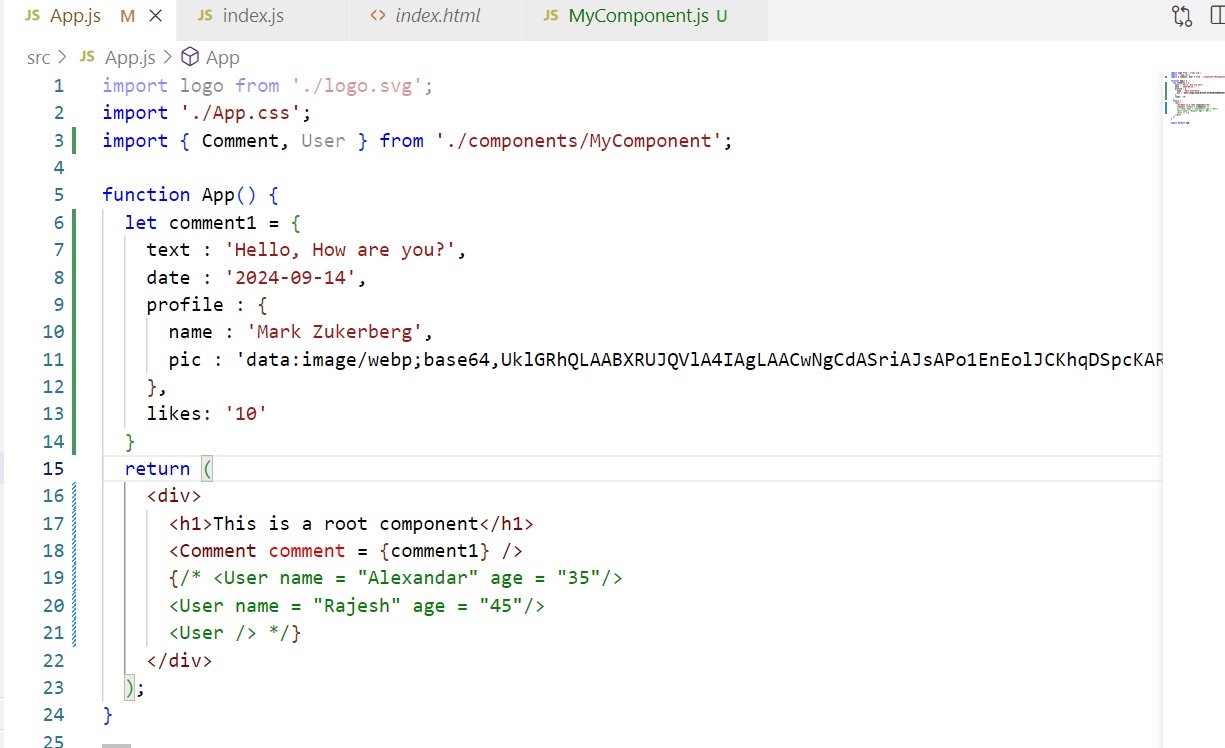


Splitting components into smaller components

MyComponent.js



App.js



How to add styles in the react applications

You have css files like

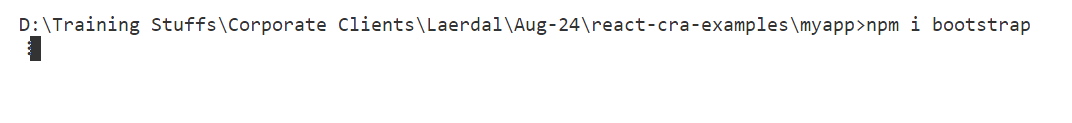
App.css: It is for Root component i.e., App

index.css: It is a global style that is available for all the components, this is imported in index.js

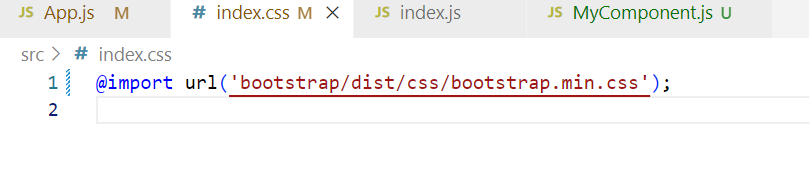
We can add 3rd party css into our project

Note: It’s not recommended to add the CDN link, it is always better to download the css files and import in your project

npm install bootstrap

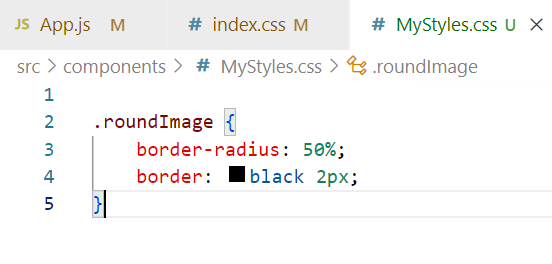


index.css



Can we create a css for a particular component

Answer: Yes



MyComponents.js



List & Keys

List is a collection of data, which you can render using map function, whenever you iterate the list you must use an unique key which helps in identifying any changes happening in the list.

listOfPlayers = [ “Virat”, “MS Dhoni” ,”Rohit”, …] ;  
employees = [ { ] ;

<ol>

listOfPlayers.map( (value, index) => { <li key = {index}>{value}</li>} );

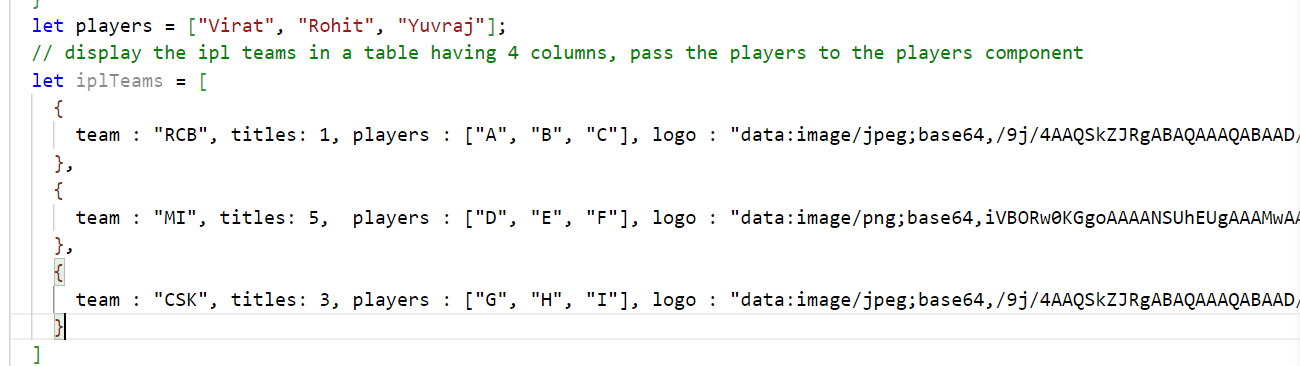
</ol>



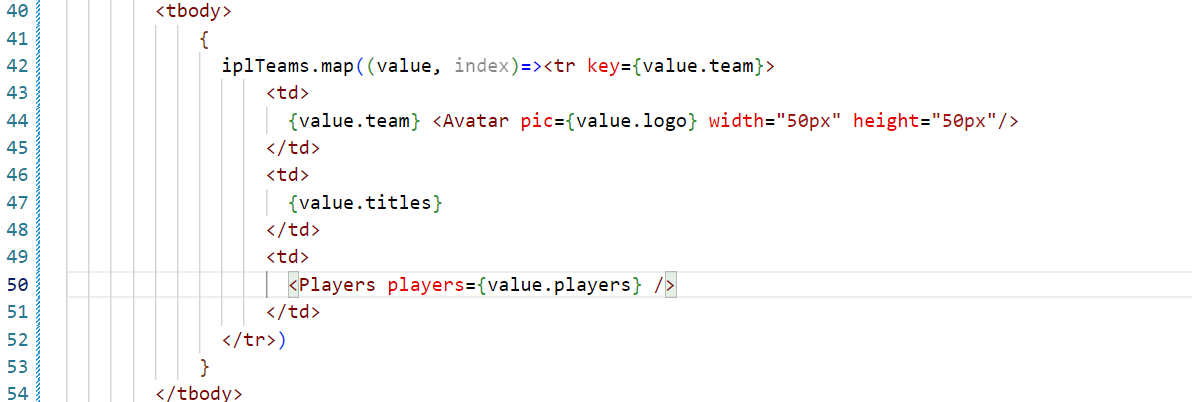
App.js



Array of complex objects



Rendering with nested components

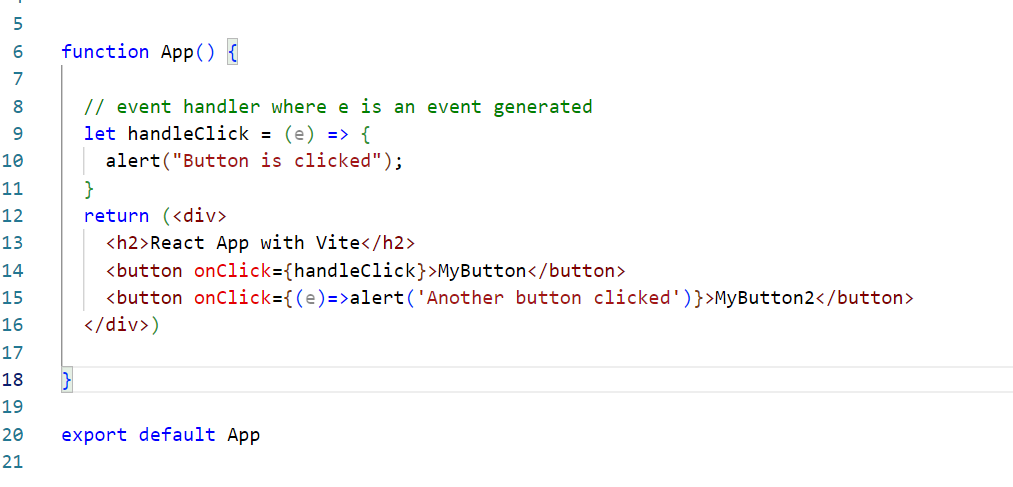


Event Handling & States in React

We can handle events in react similar to the way you handle in javascript, but the name the events should be camel case in react

|  |  |
| --- | --- |
| Event Names | React Event Names |
| onclick, ONCLICK, onClick | onClick |
| oninput, ONINPUPT | onInput |
| onchange | onChange |
|  |  |





Inline & Named event handlers



States:

Components can maintain the data in two forms

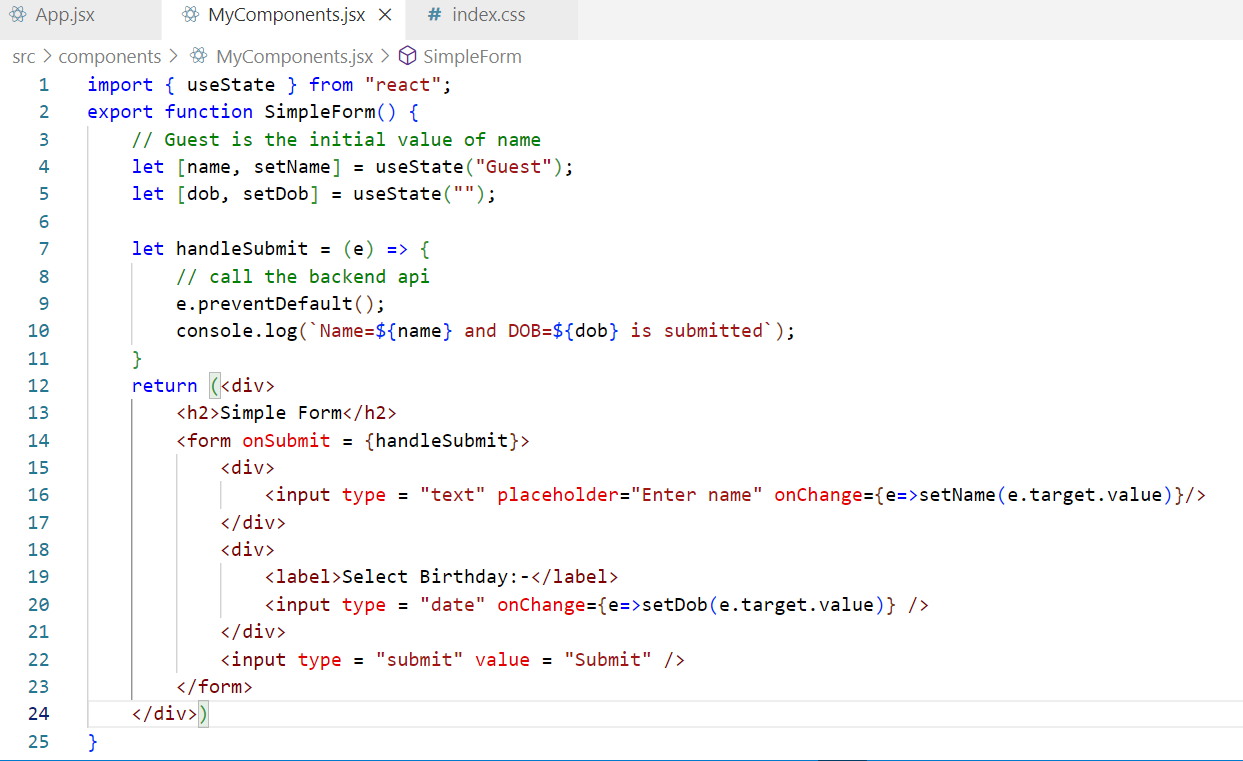
1. props: Read-only which you can’t modify, it is an input comes from different component
2. states: Read & Write both, it is used to store the component data like form data, backend data

Before React 16.8 state was present only in the class components & function components were just read-only, but after 16.8 react community added hooks which are some special function to support states in the functions.

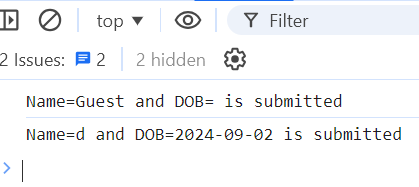
useState: It is a hook function used to create states in the function component

let [name, setName] = useState(“”); // name = “” by default

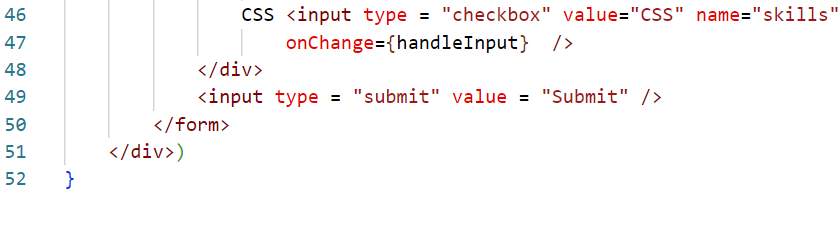
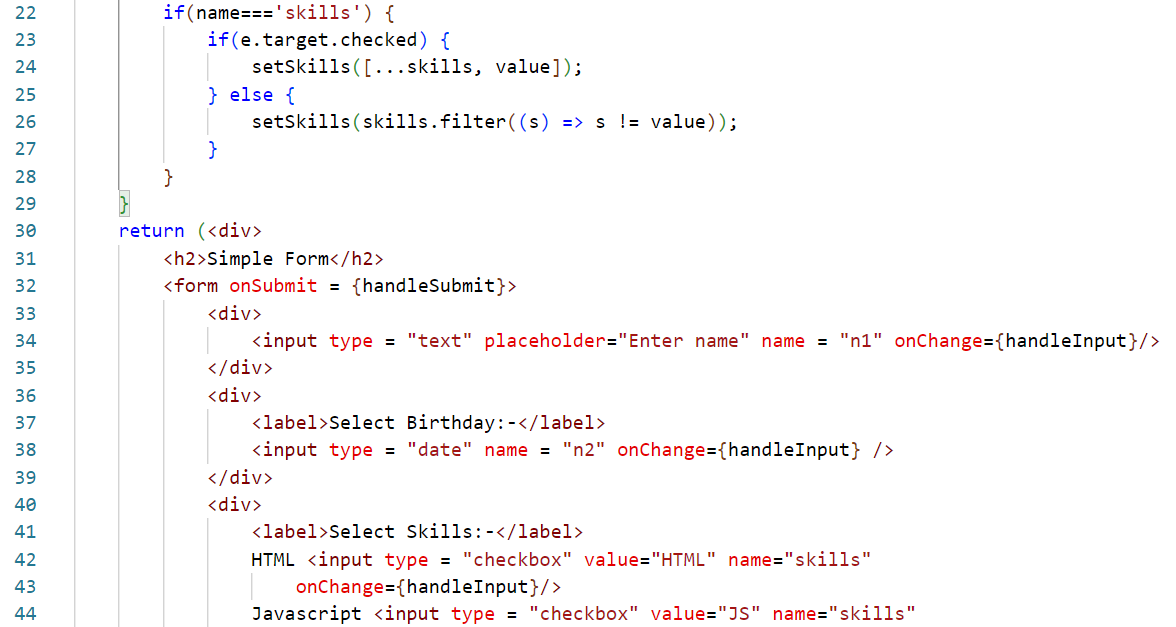
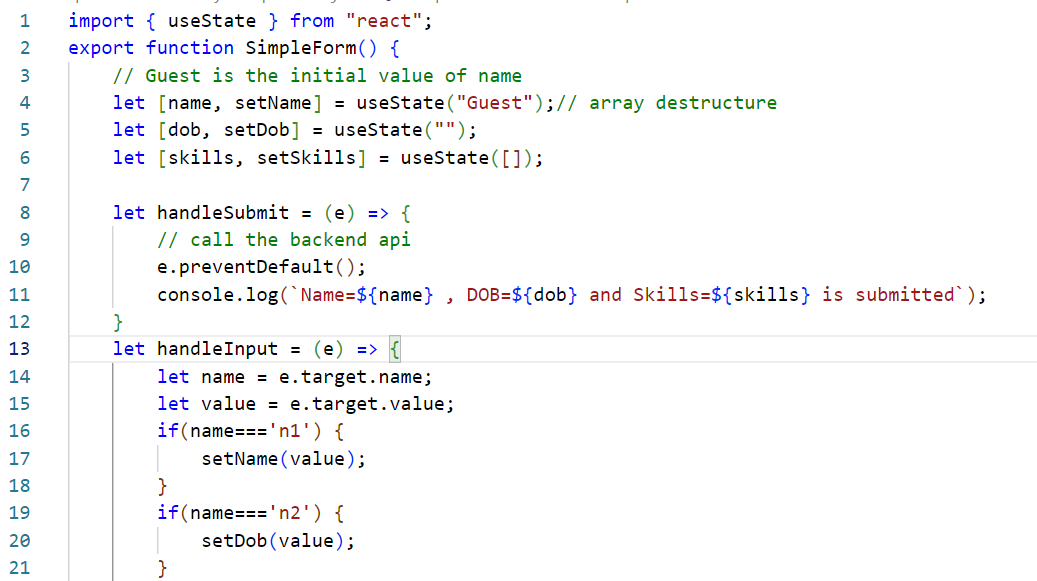
setName(“Raj”); // name=”Raj”;



Output:



For checkbox

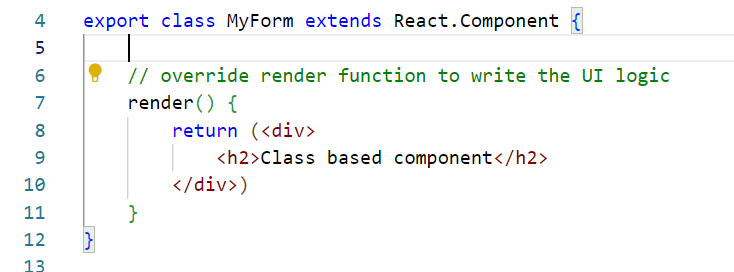


Class based components

React introduced two ways to create components

1. Class based
2. Function based

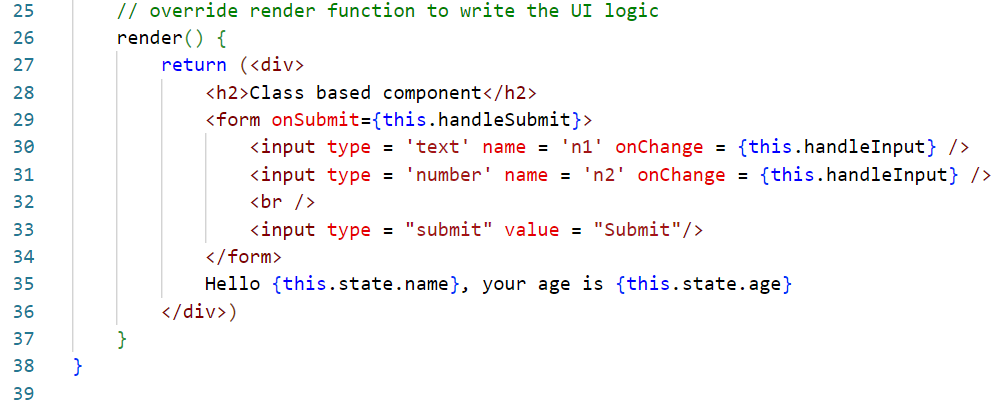
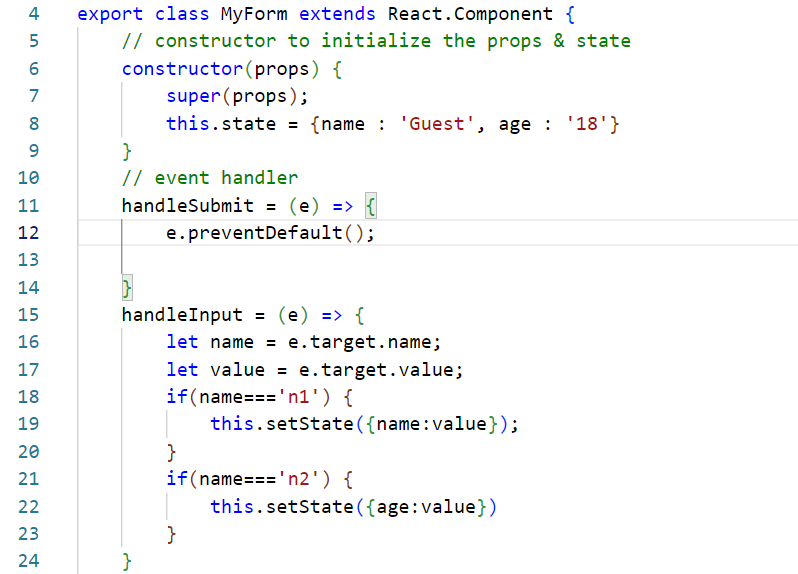
Components via Classes



In class you get a state property inherited that can be used to maintain the state & setState() method must be used to modify the states

* You must initialize the state in the constructor
* Constructor must accept the props and pass to the super class to initialize the props

constructor(props) { super(props); this.state = {name:””, age: “” }}



Differences between class & function based components

Before React 16.8 function components didn’t have state, life-cycle methods features, to use these features you must use class-based components

|  |  |
| --- | --- |
| function | class |
| Stateless components before 16.8 | Stateful components |
| functions support states from 16.8 onwards through useState | classes supports states using state & setState methods that are inherited |
| function supports life cycles methods through hook functions from 16.8 onwards ex: useEffect(), useContex() | class supports life cycle methods through the inherited methods |
| function will have the UI logic | render() will have the UI logic |

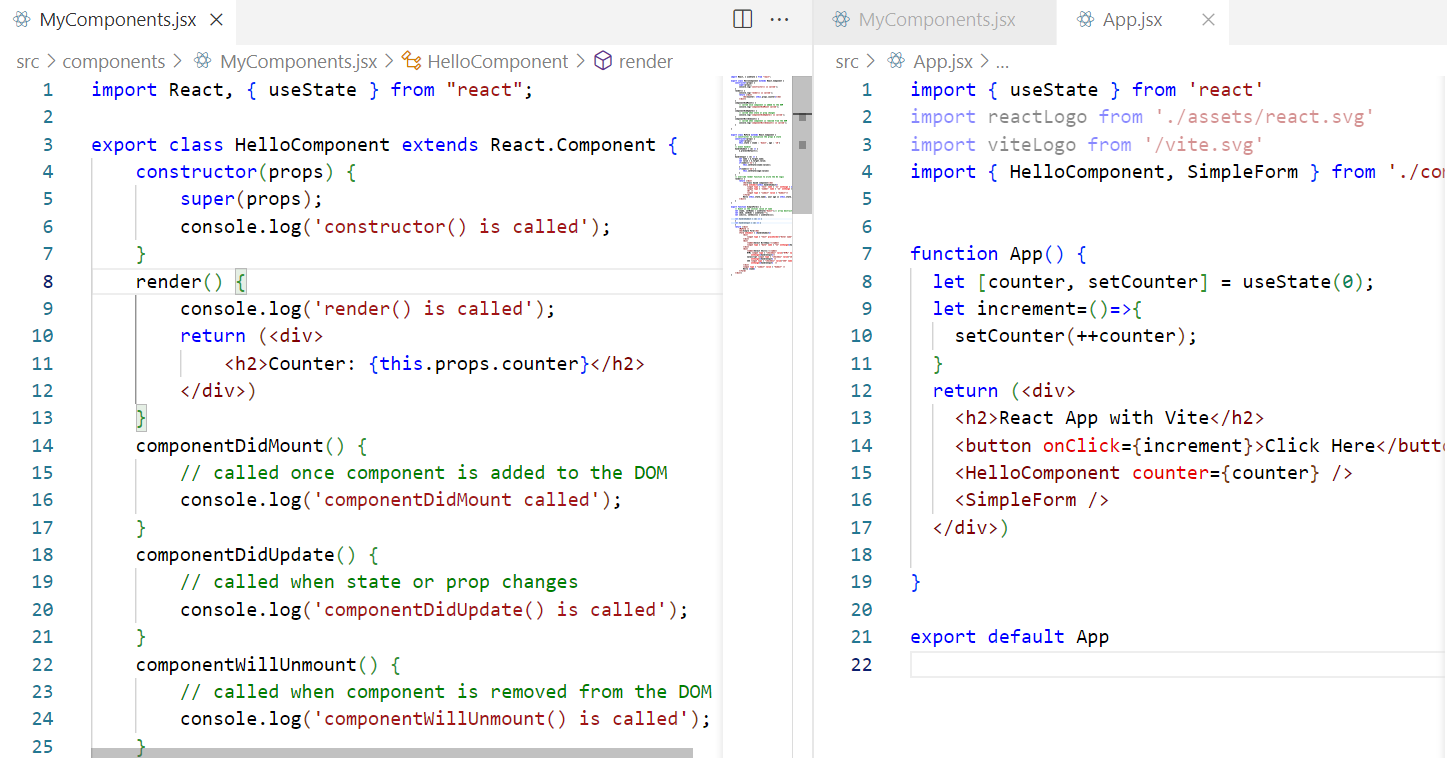
Life cycle methods of components

These are automatically called by react whenever the component is loaded or props or state is updated or whenever component is removed

Since we have class or function based components theses life cycle methods are implemented differently in class and functions.

class based components have following life cycle methods

1. componentDidMount: Called when component is added to the DOM
2. componentDidUpdate: Called when props or state changes
3. componentWillUnMount: Called when component is removed from the DOM



Sequence:

1. constructor
2. render
3. componentDidMount
4. componentDidUpdate or componentWillUnMount

Lifecycle methods for functional components

In React we have an effect hook to handle life cycle of the components, but in function based component, a single hook can be used to achieve did mount, did update & unmount

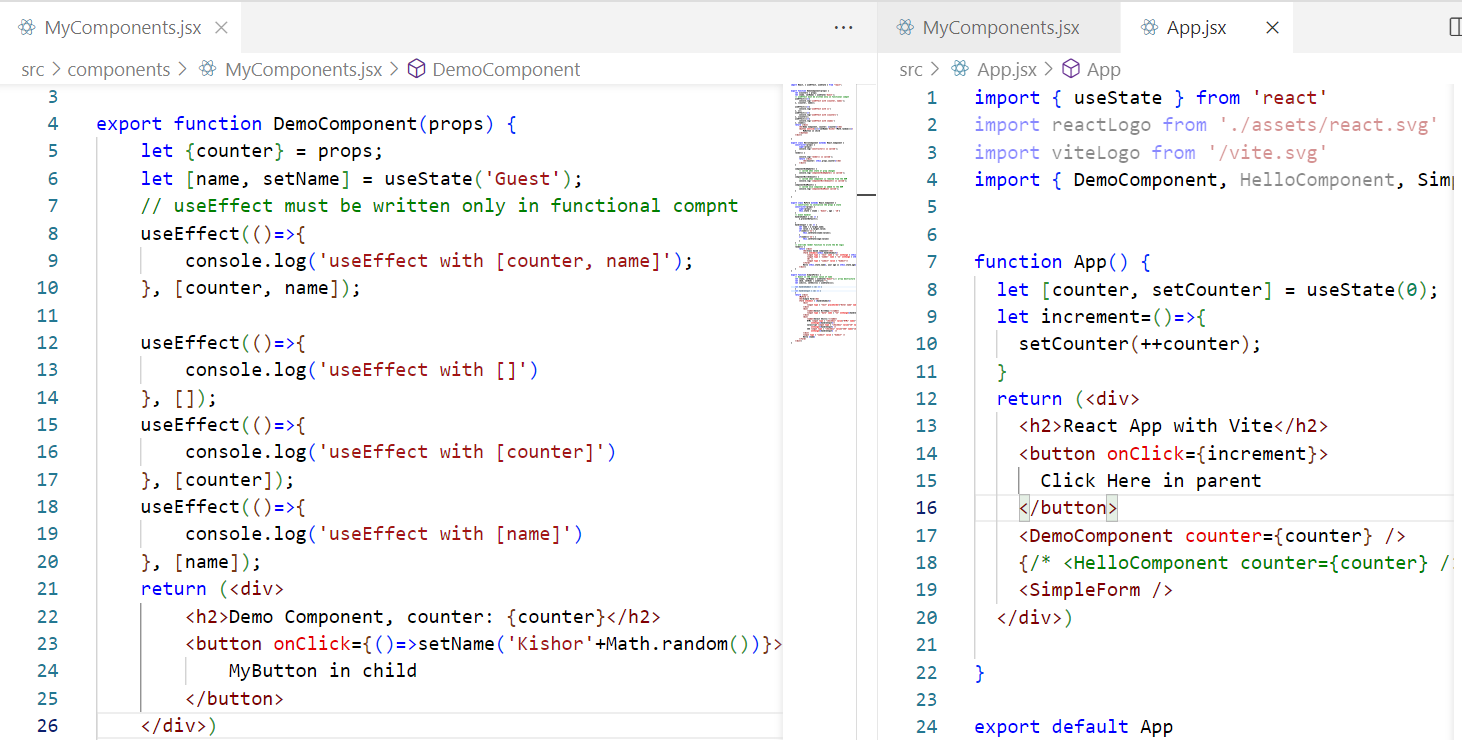
The hook function is useEffect( callback, [dependencyParameterLikePropsOrStates]);

useEffect( () => { /\* did mount \*/}, [ ])

useEffect( () => { /\* did mount + update \*/ }, [x, y] }

useEffect( () => { return () => { /\* unmount \*/ }, [] }

Note: You can’t use hook functions in the class components, it must be used only in the function components



Output:

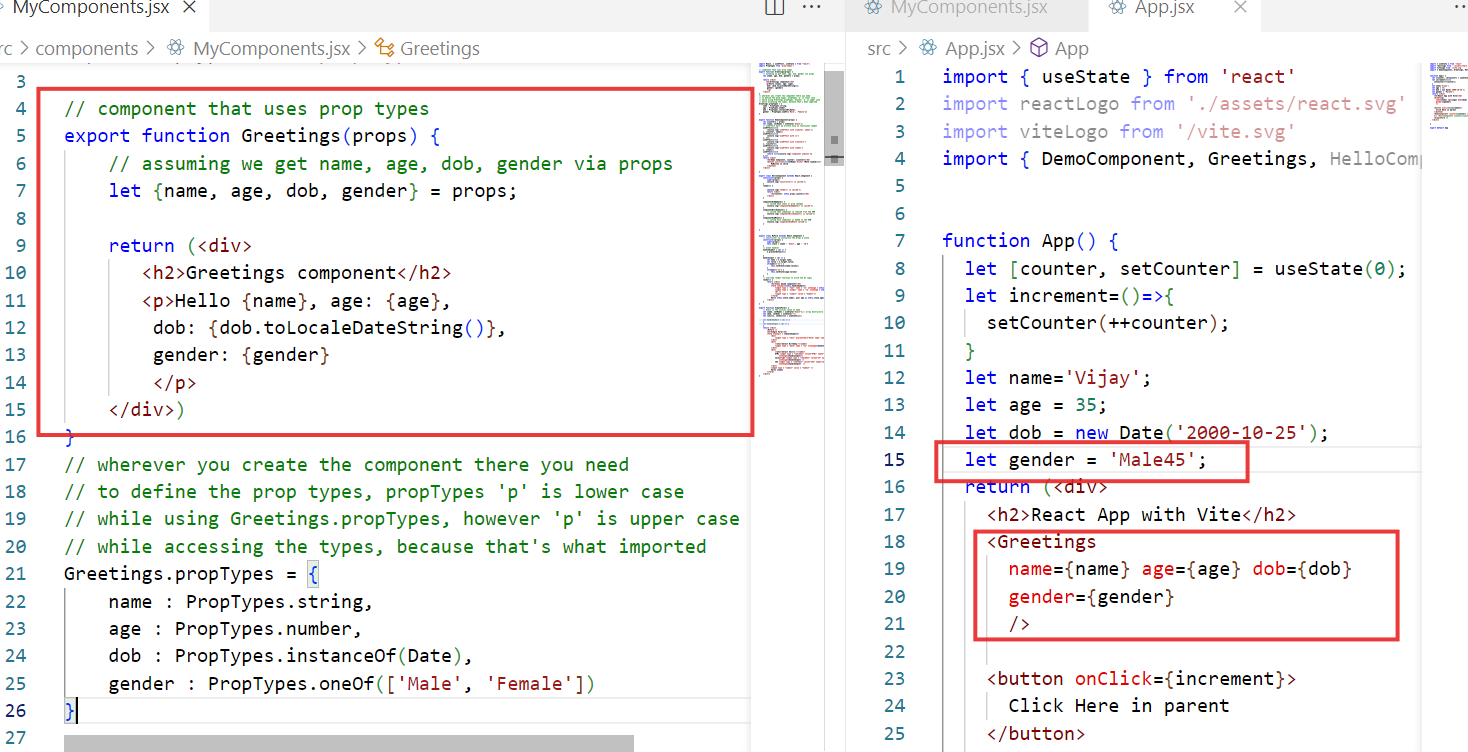
useEffect with [] called only once when component is added  
useEffect with [name] called when state changes  
useEffect with [counter] called when prop changes  
useEffect with [name, counter] called when prop or state changes

Prop Validations

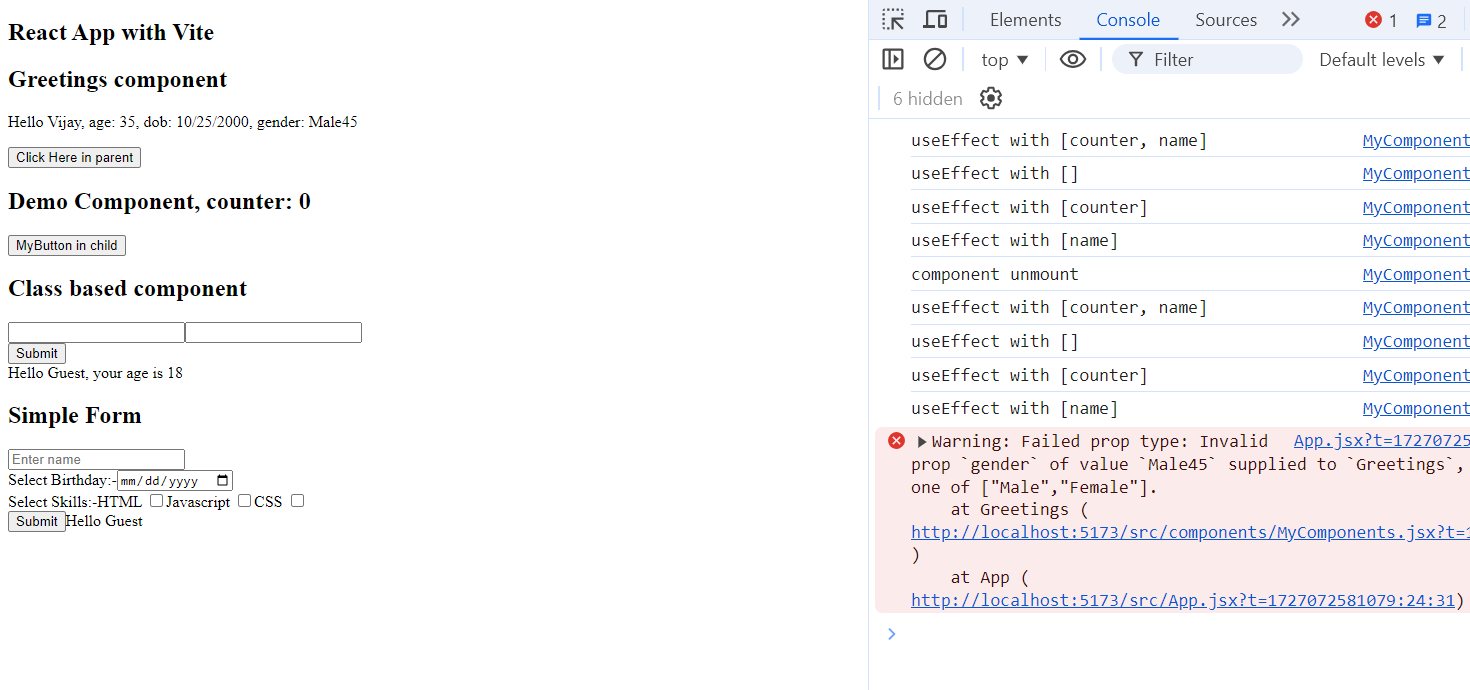
React provides prop-types library to specify the types to the props, in case the types don’t match you will get the warnings in the console

Below are the types provided by PropTypes

1. string
2. number
3. bool
4. instanceOf
5. oneOf



Output:

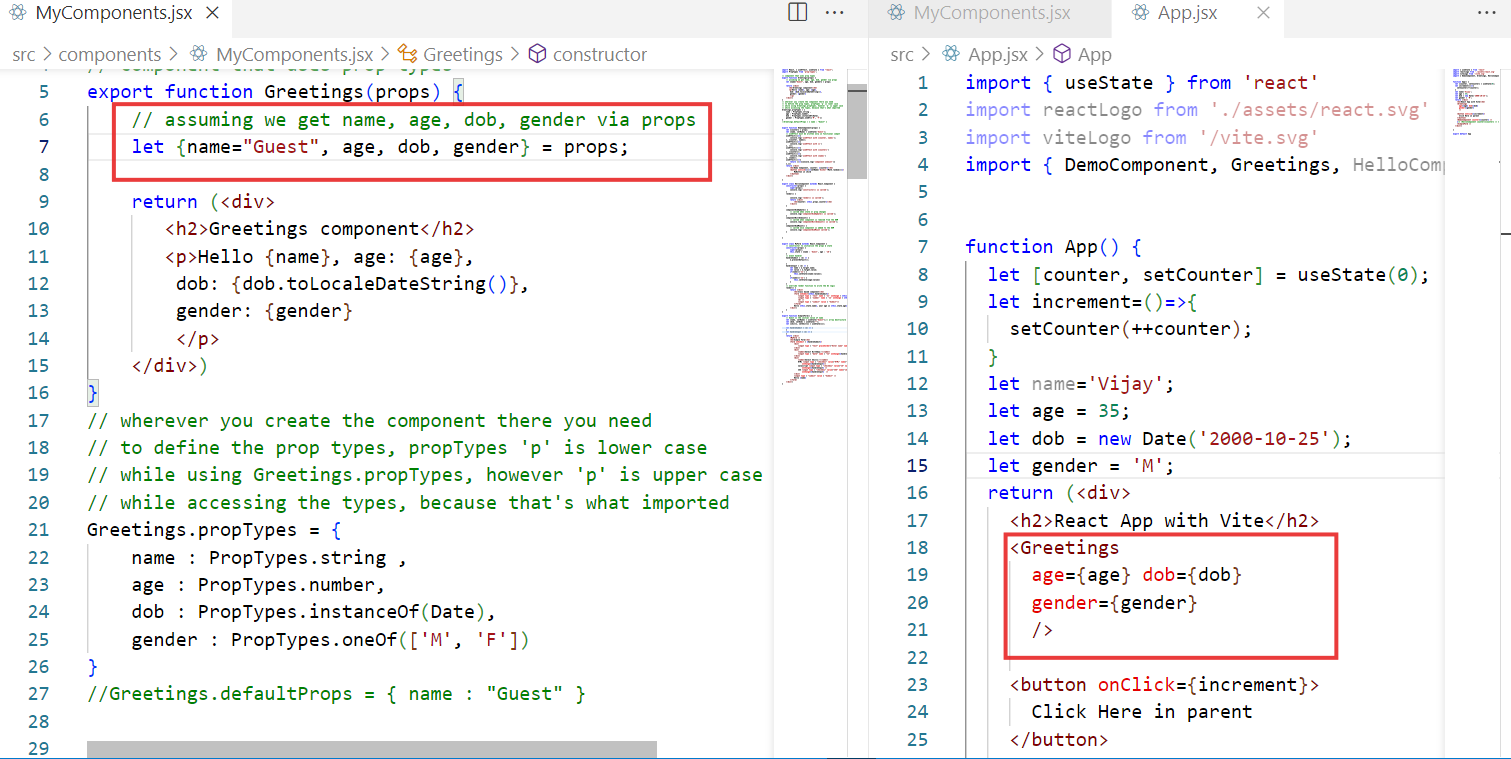


React also provides default props to give the values when the prop doesn’t get the value, but it is deprecated because react suggests to use the Javascript default parameters in place of default props

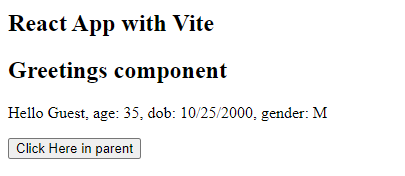
Greetings.defaultProps = {  
 name : “Guest”, age : 18, gender : ‘Male’  
}



Using the default parameter of javascript instead of default props

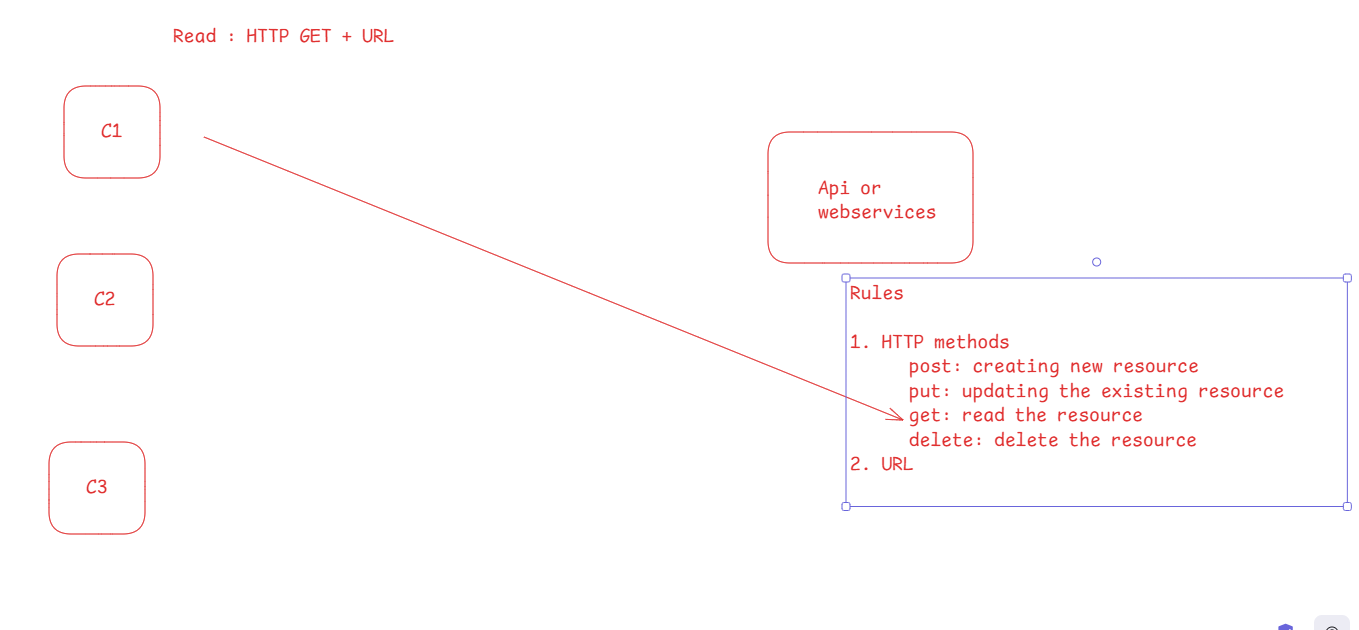


Output:



How to access API’s in React.js

API’s are webservices which consumes or produces data from/to the client programs, here client programs can be React, Angular, Vue, Java, Javascript, Python



We can use Online JSON placeholder website to access a fake API through HTTP GET

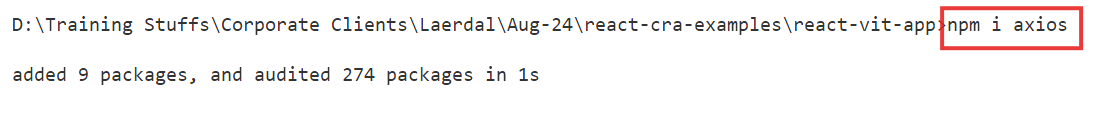
React uses a library called axios to access API’s

npm install axios

axios provides the functions with http method names that takes URL & data if required, it returns a Promise that can be resolved or rejected by then() & catch

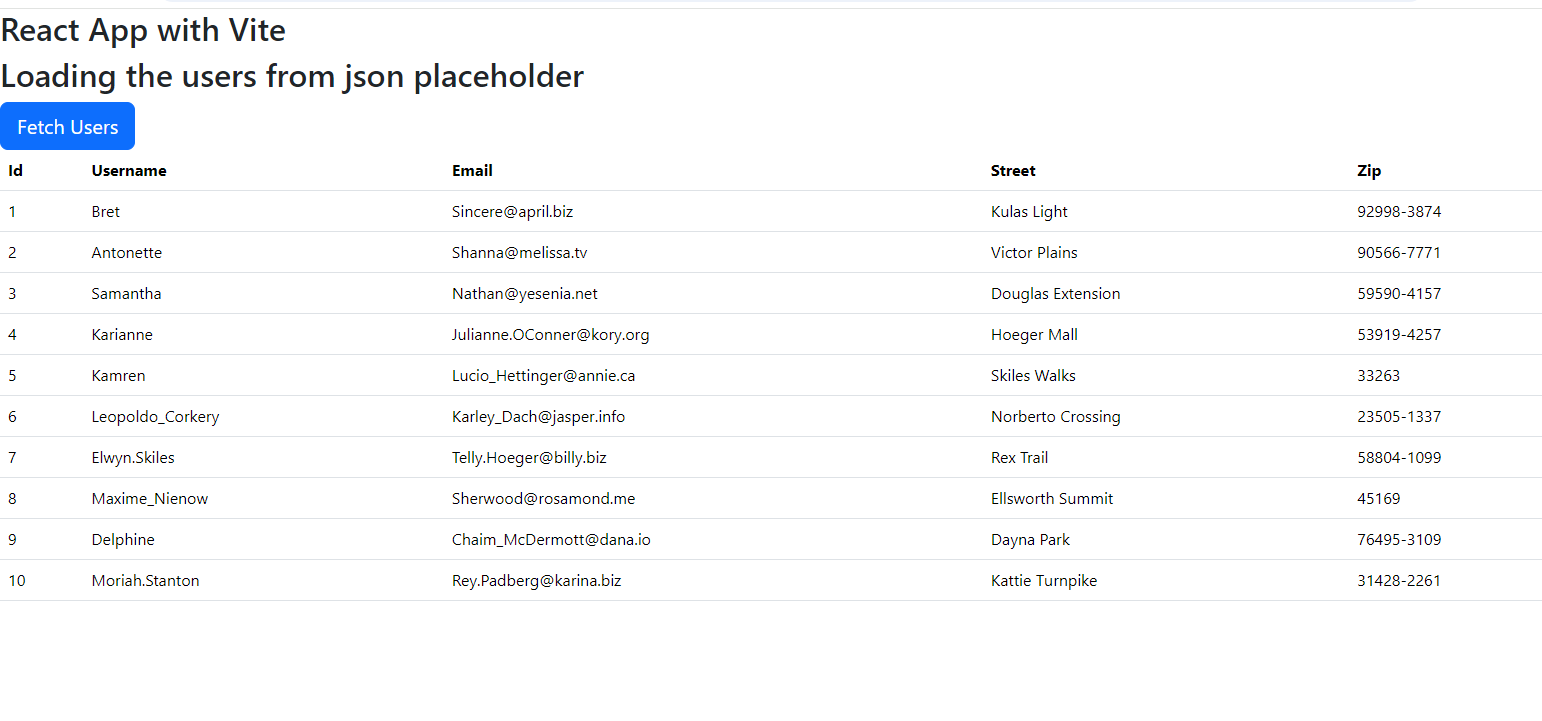
ex:

axios.get(URL).then( callbackFn).catch( callbackFn )  
axios.post(URL, data).then( cbFn).catch( cbFn )  
axios.put(URL, data).then( cbFn).catch( cbFn )  
axios.delete(URL, data).then( cbFn).catch( cbFn )





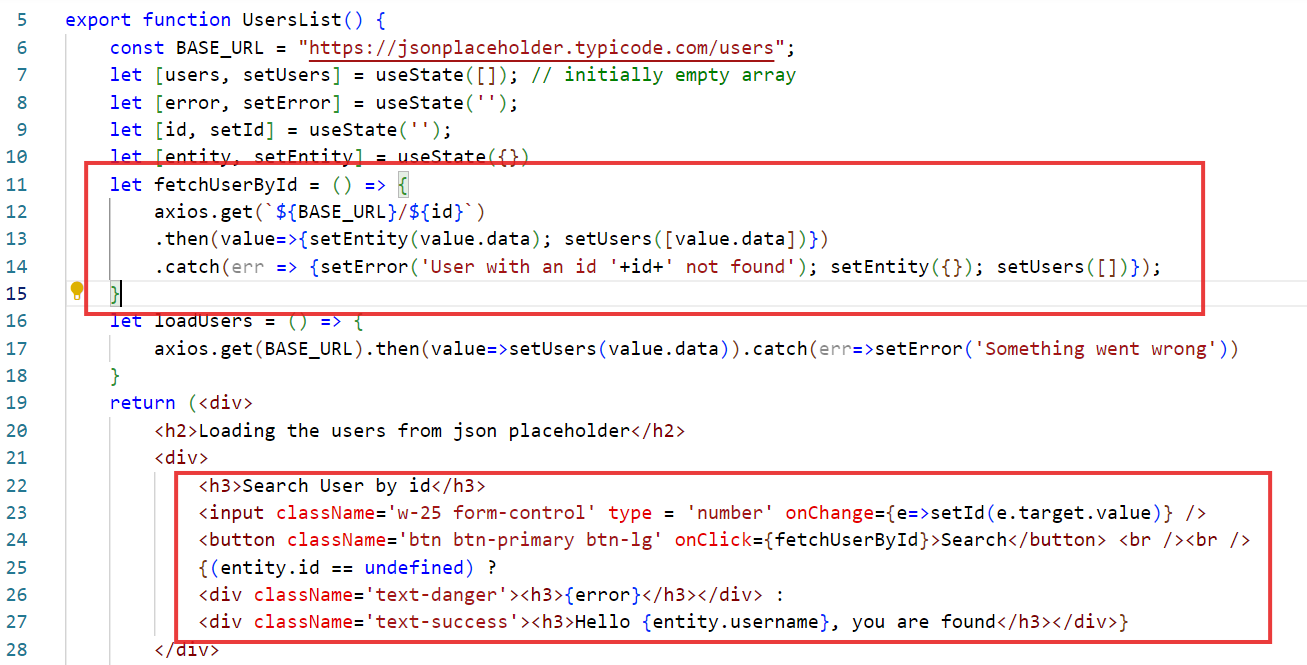
Output:



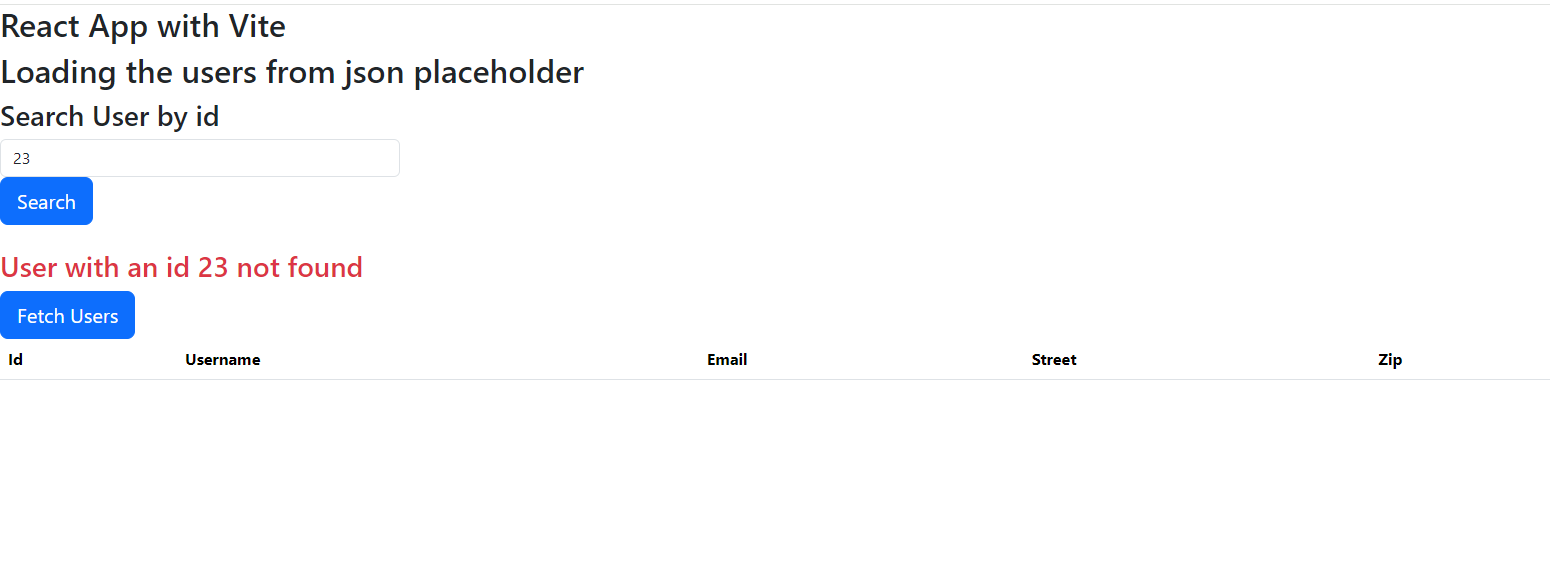
Path parameters:

When you have dynamic value that needs to be part of the URL path, then you can use the same URL that can be parameterized

/users/1  
/users/2  
/users/3



Output:



Routers

It helps in developing Single Page Application look and feel between the components, where you can navigate from one component to another

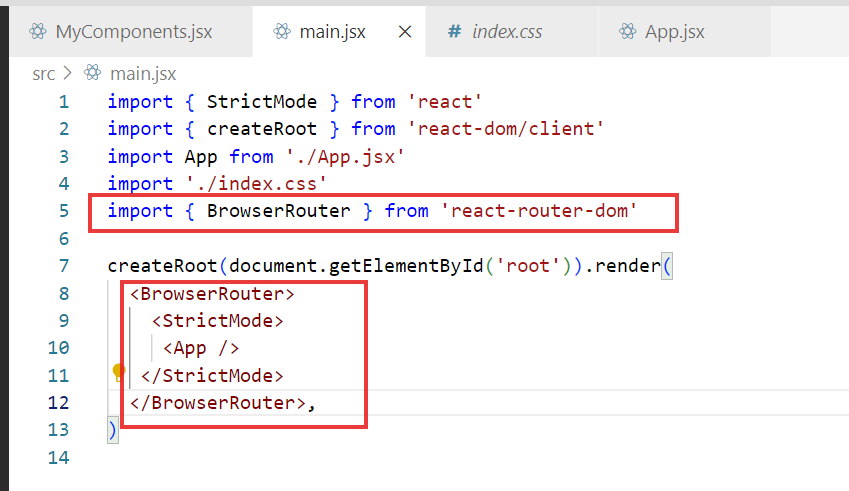
We must install Router library called react router dom

npm install react-router-dom

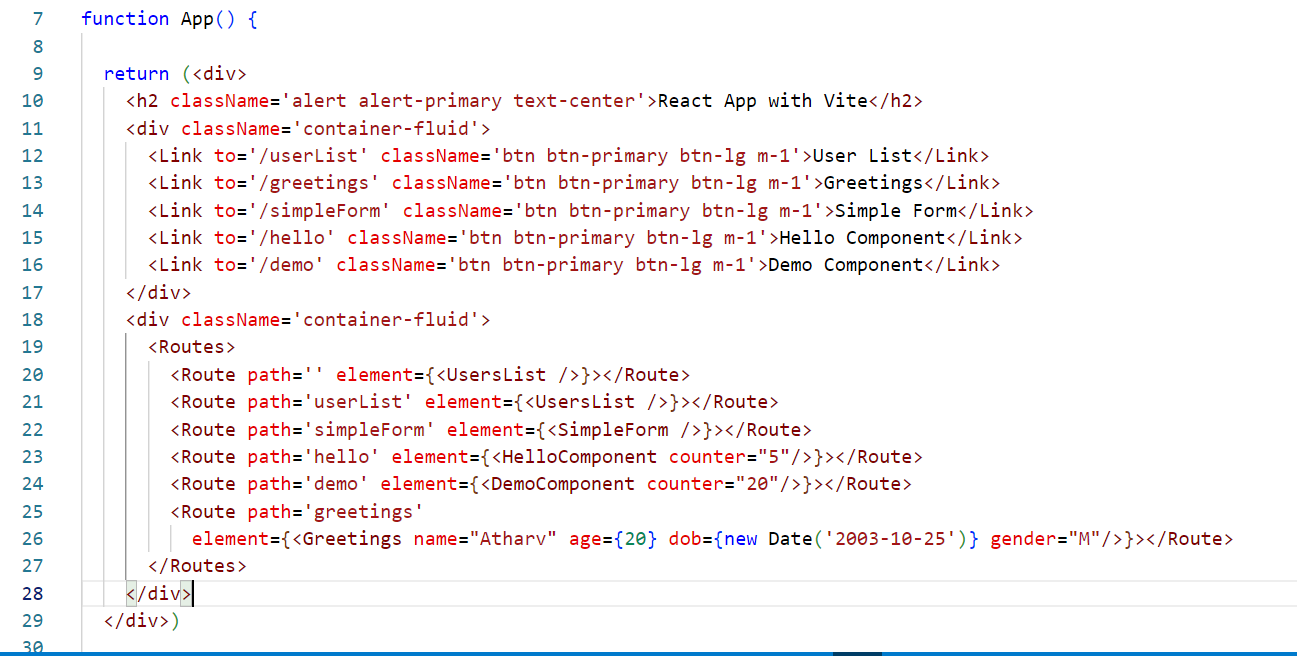
React Router DOM provide following components & hook functions

1. <BrowserRouter>: It must be the parent of all the components
2. <Link>: It is used to create links which you can click
3. <Routes> & <Route>: These are like switch cases to load the component based on the link you click
4. useNavigate(): This is a hook which acts like <Link> but its used for programmatic navigation
5. useParams(): This is a hook used to extract the path parameters while routing

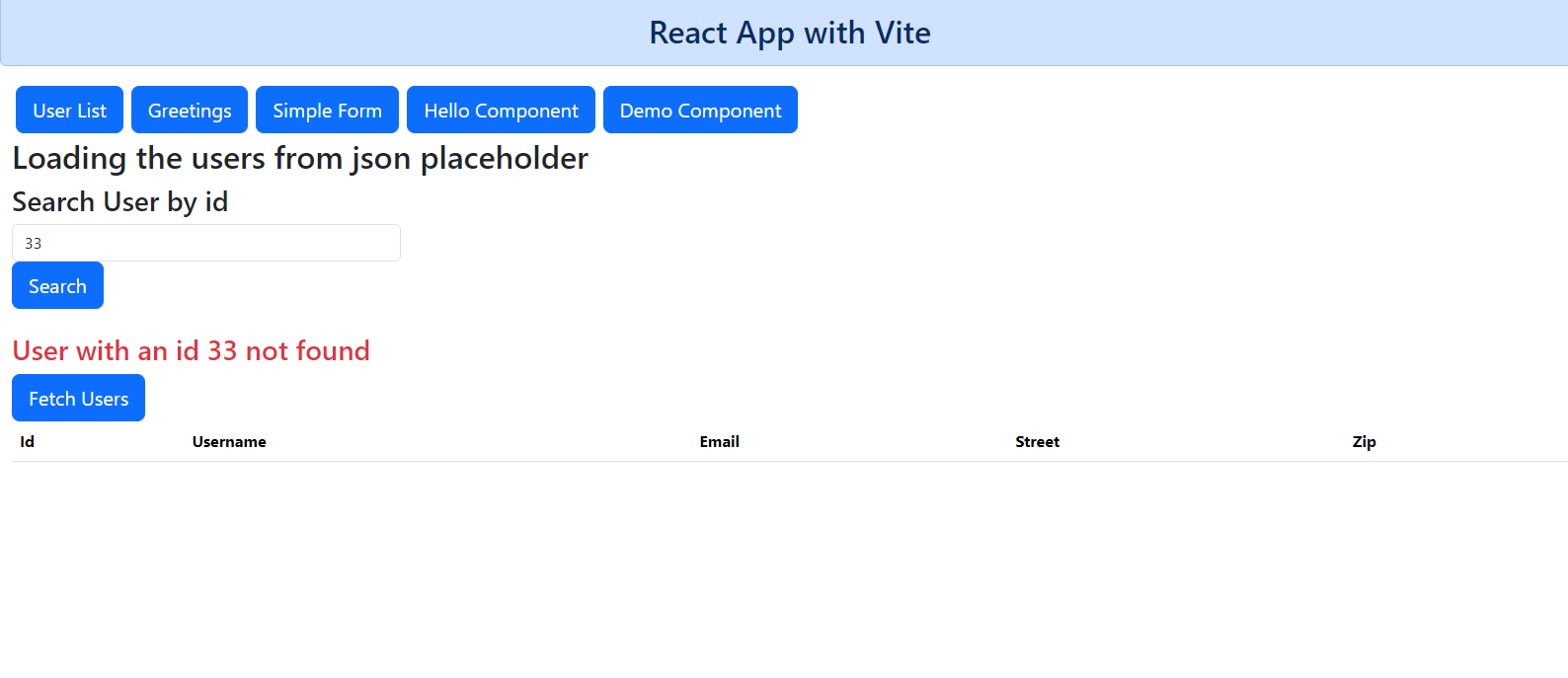
Adding <BrowserRouter>



Now you can use <Link>, <Routes>, <Route> in any component you want



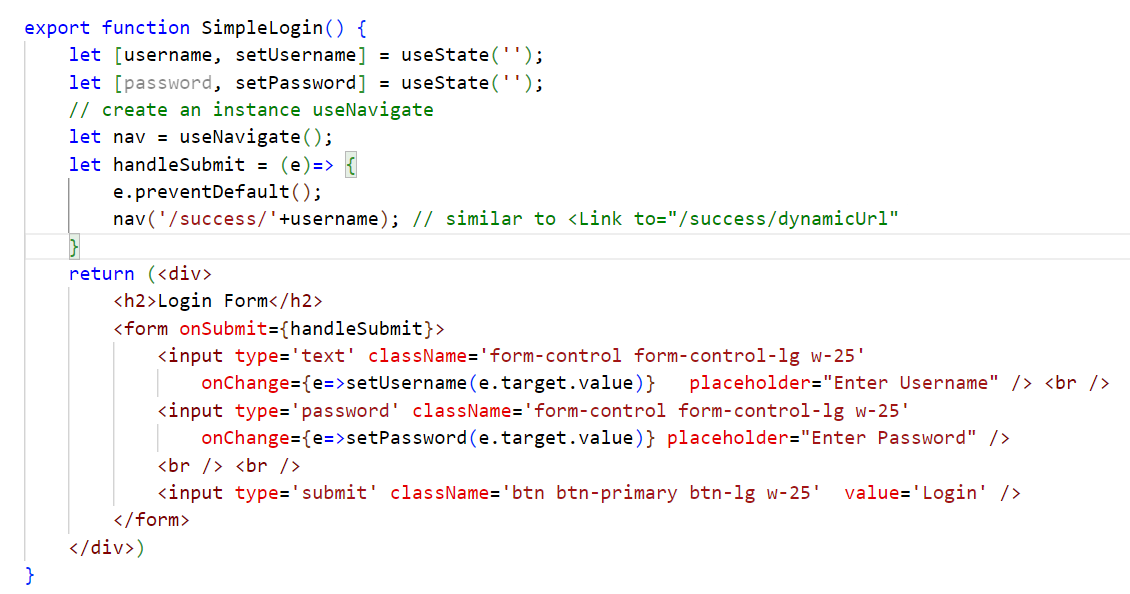
Output:



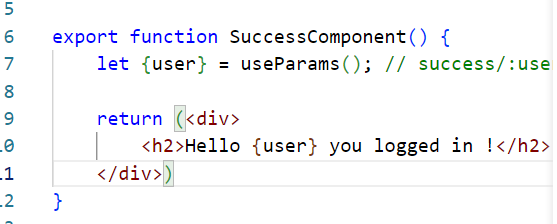
Using path parameters / route parameters and programmatic routing

Route parameters can be extracted using useParams() hook

Programmatic routing is done with the help of useNavigate()



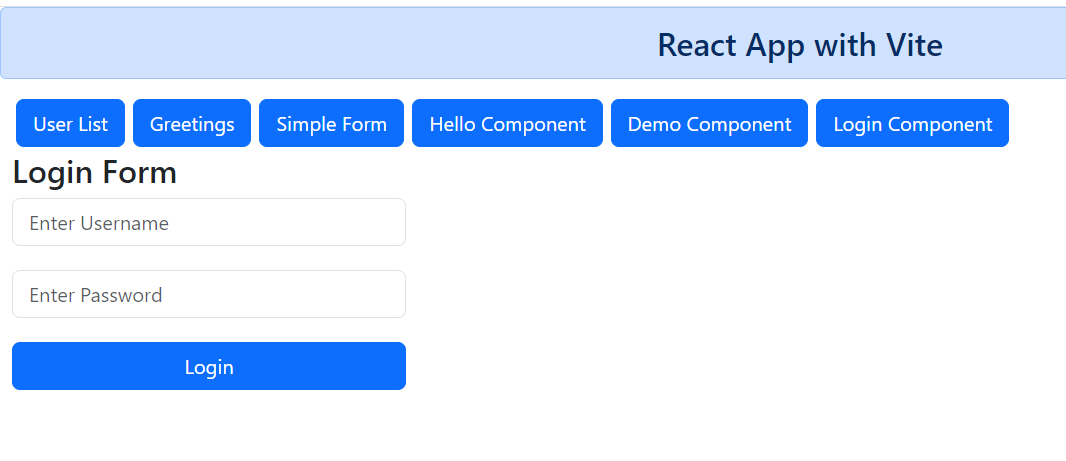
Success Component



Defining the routes for the /success/dynamicUser



Output:



useContext()

This is used when you want to pass the data across the components without using props drilling

It is like a global state that needs to be shared across the multiple components

import { createContext} from ‘react’;

export const NameContext = createContext();  
  
let [name, setName] = useState();  
let [counter, setCounter] = useState();

<NameContext.Provider value = {name}> or value = { [name, counter] }>  
 <Component1 />  
</NameContext.Provider>

<One.Provider>  
 <Compnent 1 />  
</One.Provider>

Component1 -> Component2 -> Component3

In Component3

let ctx = useContext(NameContext)

{ ctx }

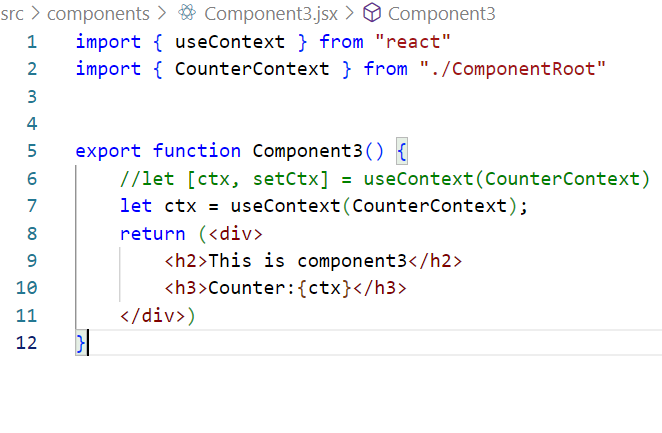
ComponentRoot.jsx



Component1.jsx

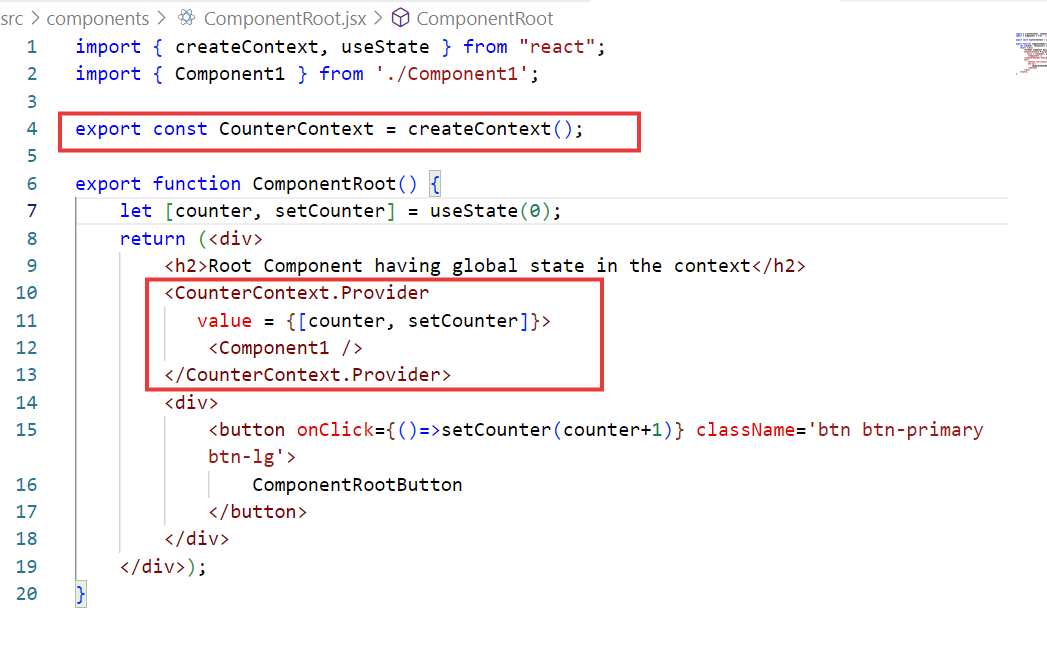


Component3.jsx



Changing the context by lifting the state up

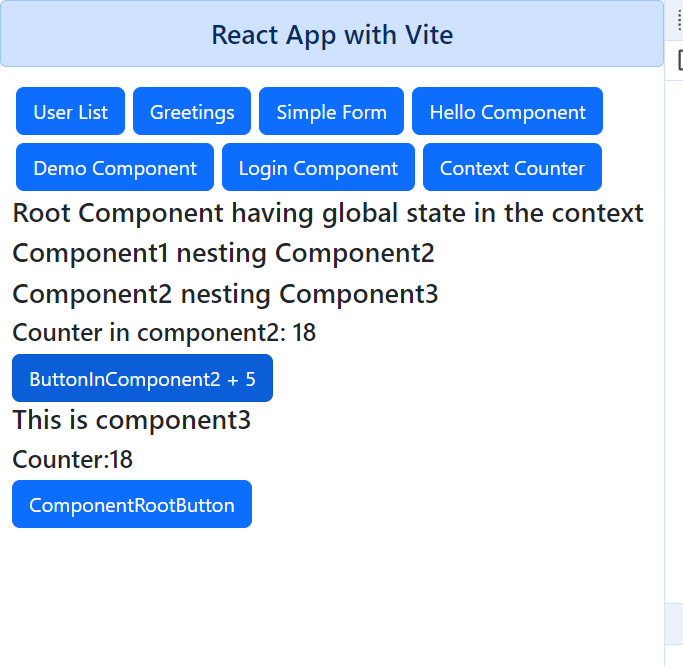
ComponentRoot.jsx



Component2.jsx



Output:



Real API’s can be developed using node.js or java or PHP or python

If use Node.js it is similar to writing javascript, it provides third party libraries like express.js to create RESTful webservcies and mongodb library to interact with the mongodb.

Download mongo-db software’s

1. mongodb-community server – zip/tgz - database
2. mongo shell – terminal to start & query the database

Mongo DB is a NoSQL database, it maintains the data in JSON format, MongoDB inbuilt supports Javascript functions

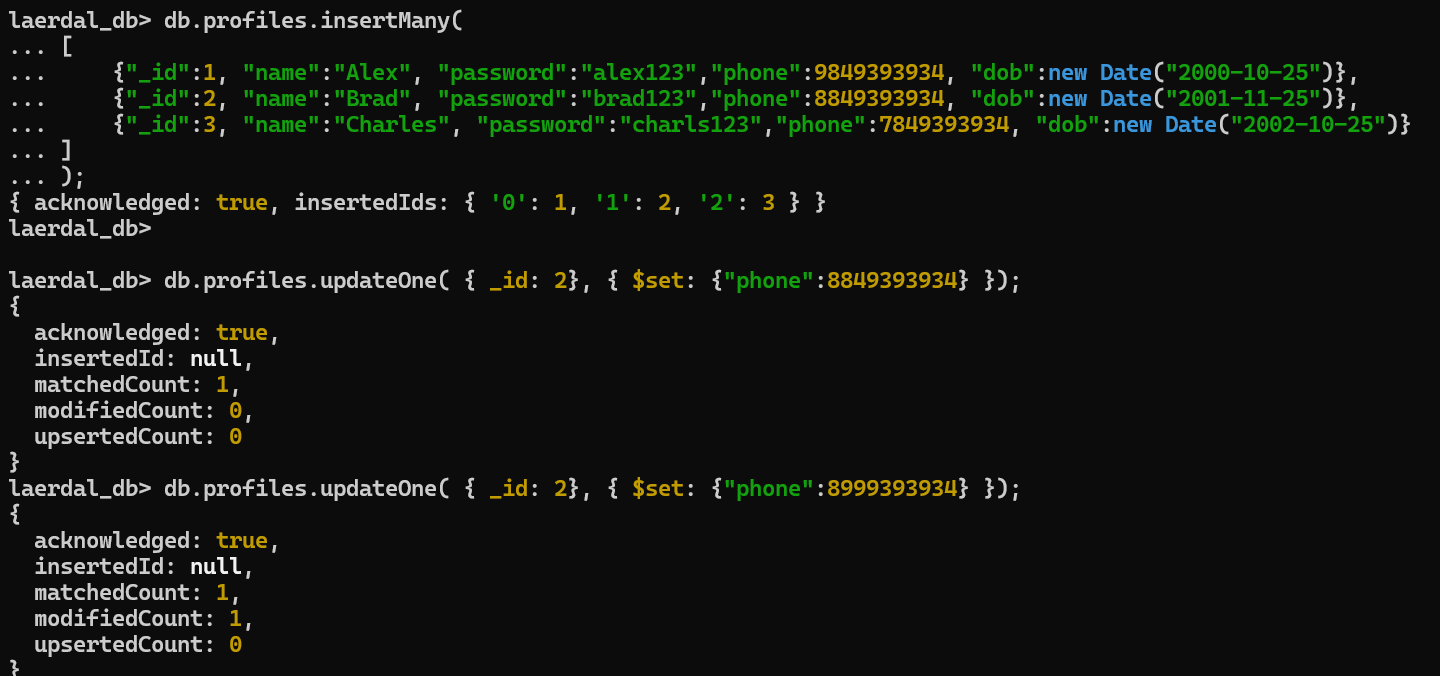
mongod: It is a command to start the database, by default it looks for data/db folder to maintain the data

Case-Study

A Profile/User who maintains his/her friends contact in the application

Profile – id, name, password, dob, phone, email – { contacts list }

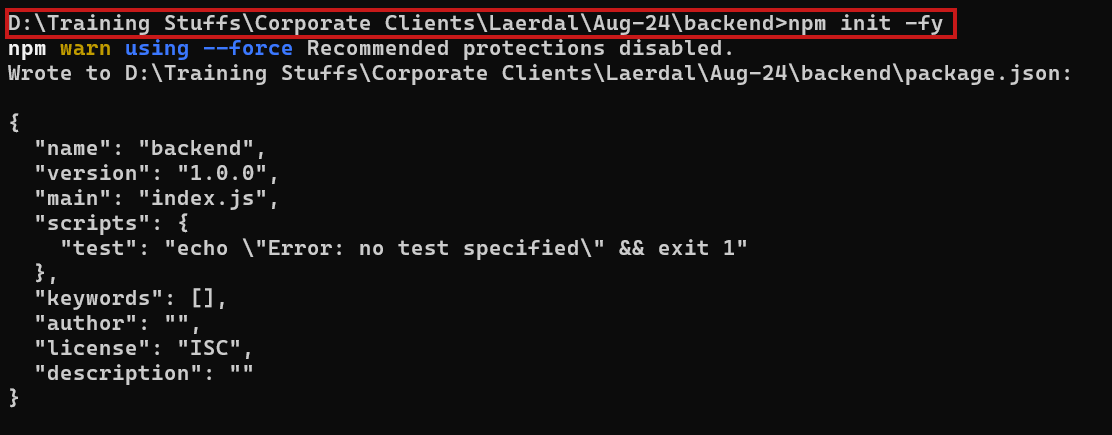
insertOne, deleteOne, updateOne, findOne, find: These are the inbuilt functions available in the MongoDB



How to interact with the DB using Node, Express

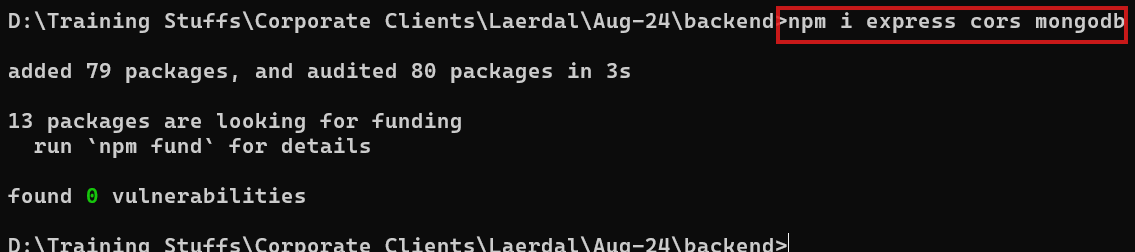
You must install the library

* mongodb: provides functions that return promise

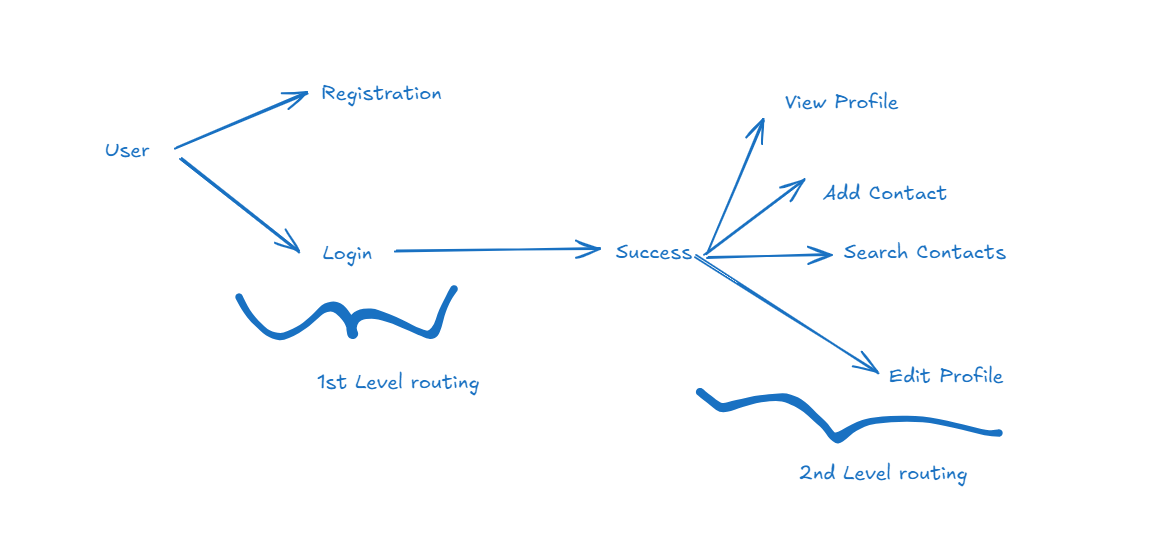


We must install following libraries to create API & backend programs

1. express: Create backend webservice
2. cors: To enable Cross Origin Resource Sharing
3. mongodb: To interact with the mongodb

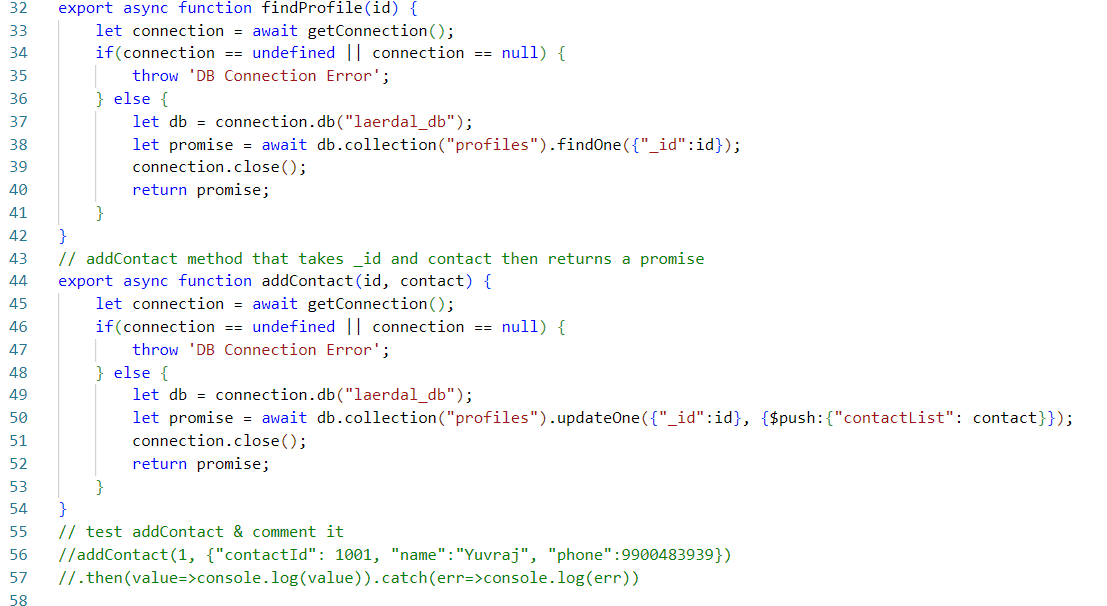
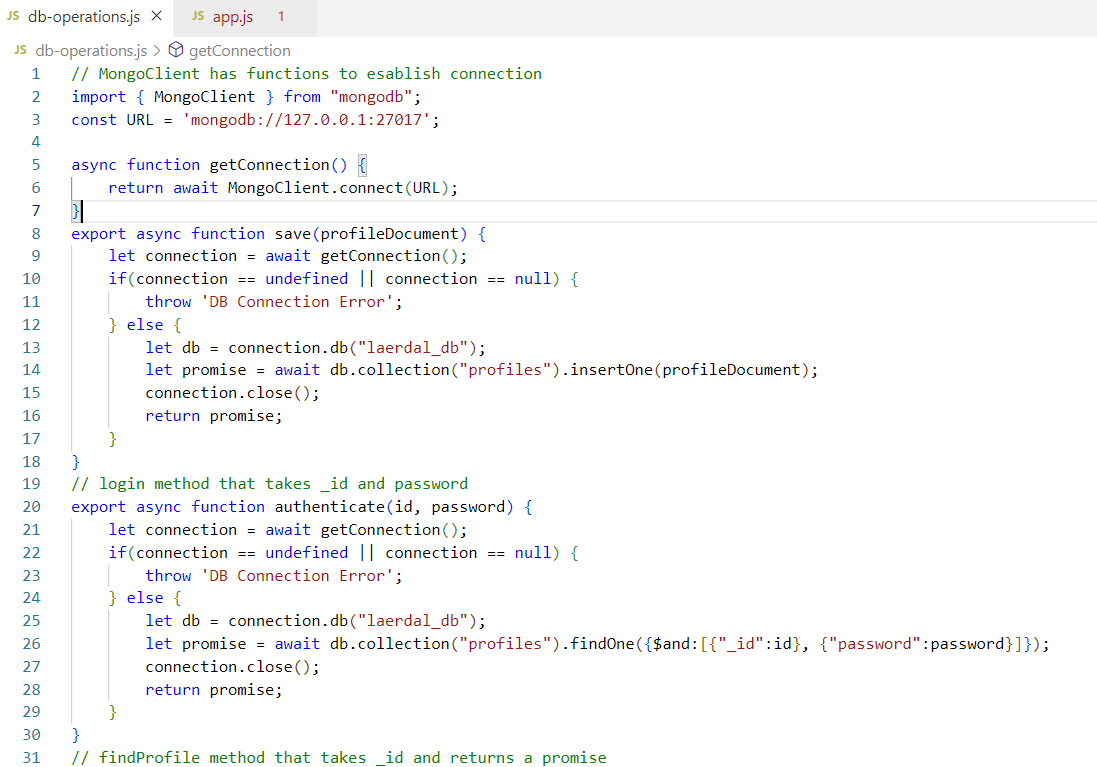


Flow chart of contact management application

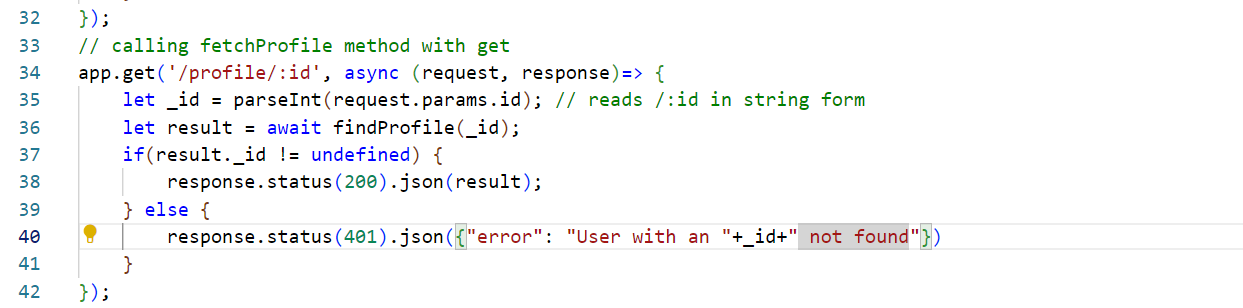
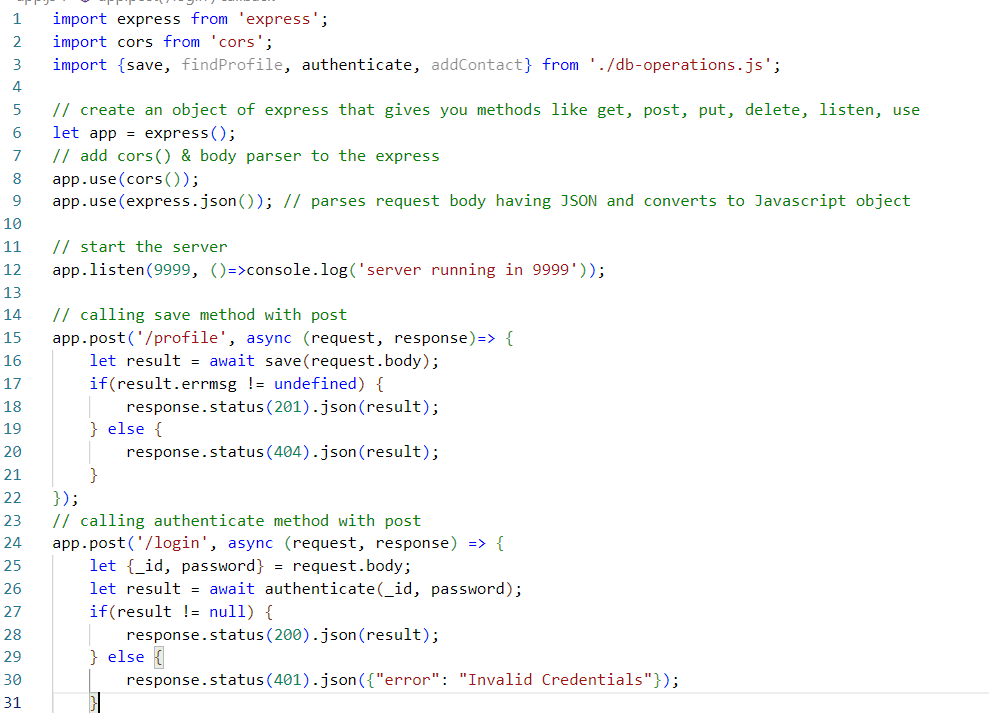


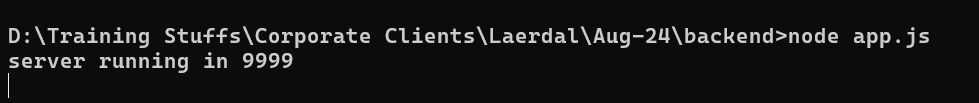
Note: We must create a db-operation that adds contact to the existing profile

db-operations.js

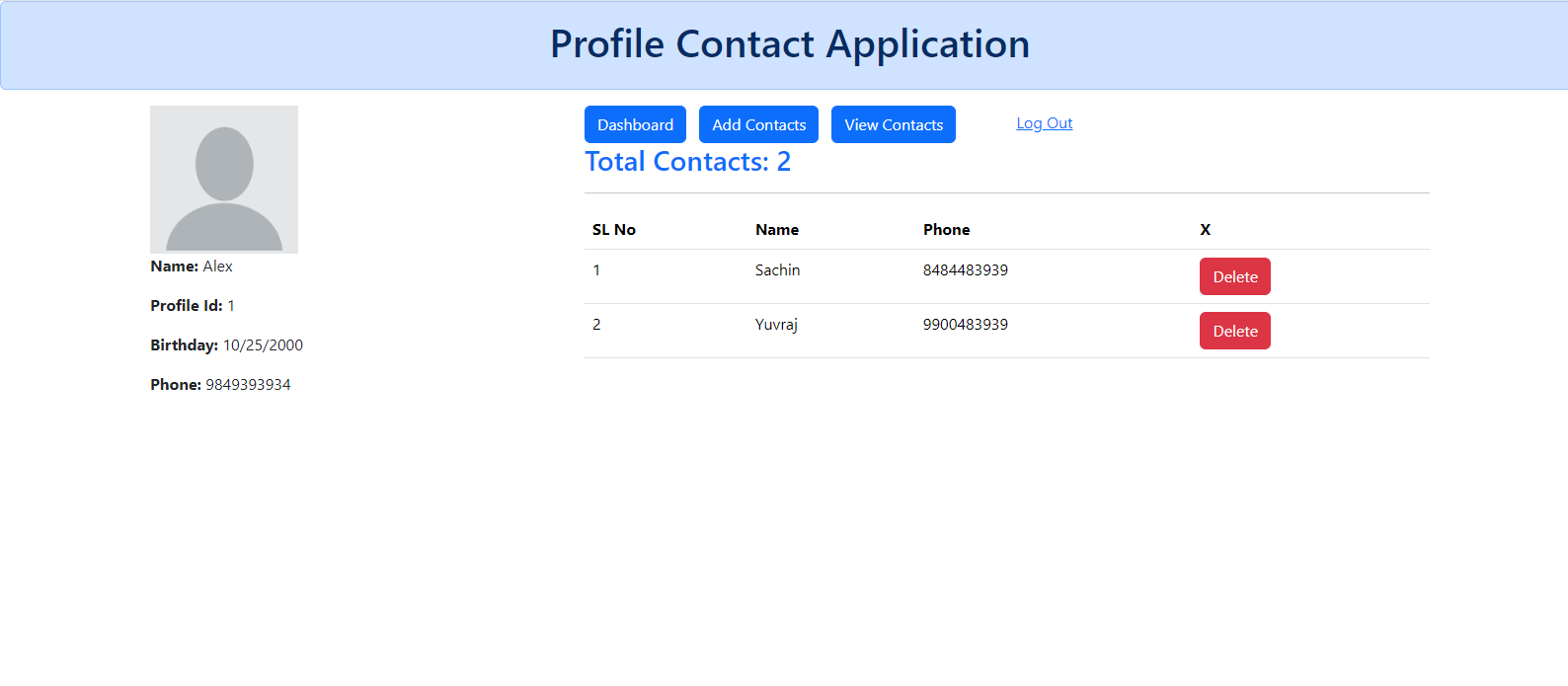


Creating REST APIs to access the db-operations.js functions





Create the UI as below



How to add typescript to the existing react project

In Javascript

let x = 20;  
x = “hello world”;  
x = { id : 20, name : “Raj” }

In Typescript

let x : number = 20;  
x = “hello world” ; // typescript throws the error

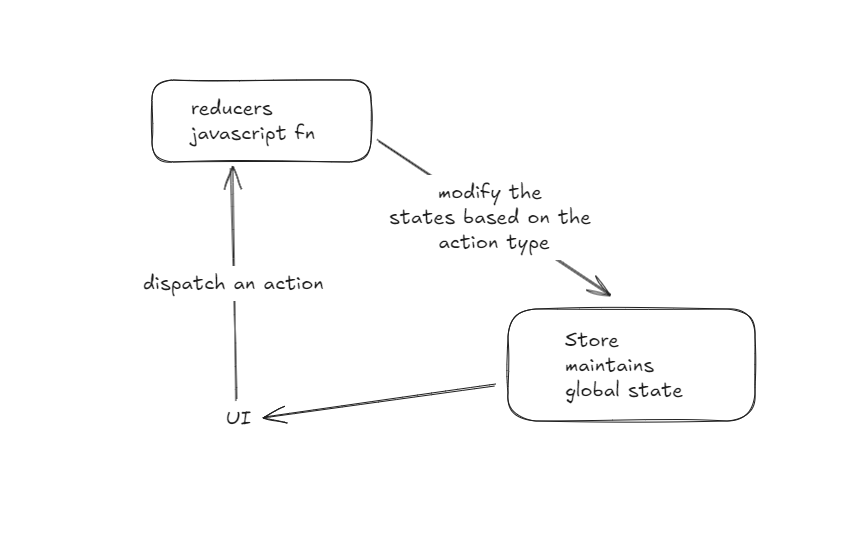
let x : number | string = 20;  
x = “hello world”;

Types in Typescript

void  
string  
any  
number  
boolean  
[]  
undefined

Redux

It centralizes the state so that multiple components can update / read



Redux modifies the state in a different way compare to the latest redux version and old redux version

Redux with older version

It uses two libraries

1. react-redux
2. redux

Redux with latest version

Redux has given a toolkit where it has simplified writing actions, reducers, it uses two libraries

1. reduxjs/toolkit
2. react-redux

Store: It is used to maintain the global state

Reducer: It is a javascript function that will have action creators which can modify the state

reducer.ts

let states ;   
function reducer(state, action) { … } //

export reducer;

store.ts

let state = createStore( reducer );  
export state;

main.tsx

<Provider state = {state}>  
 <App />  
</Provider>

let value = useSelector(state => state.value); // the components can read the state

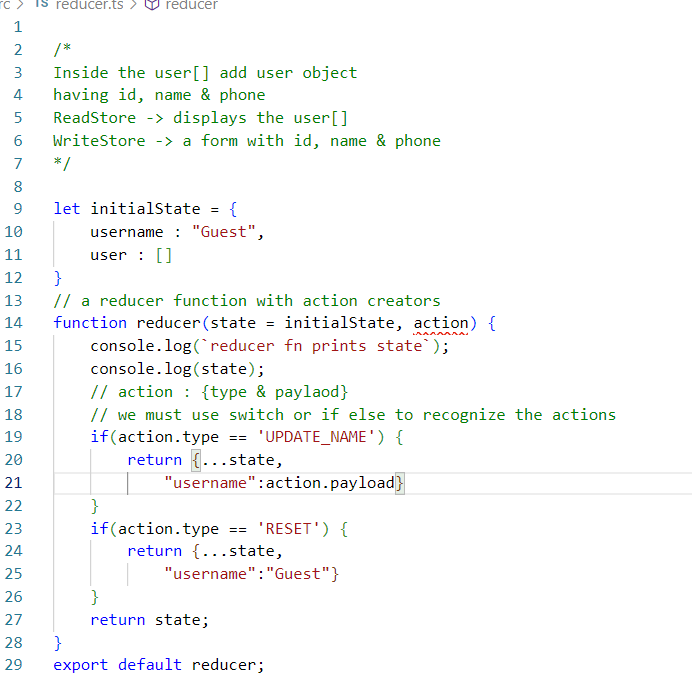
let dispatch = useDispatch();

dispatch( { type : “ADD”, payload : 25 } ) // dispatch an action to the reducer with a value 25, so that reducer must pass 25 to the store to update the state

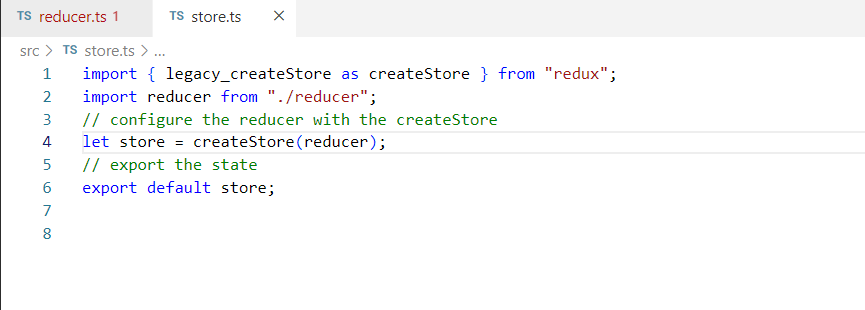
Program to create 2 components one reads the store and another writes the store

Full Program

reducer.ts



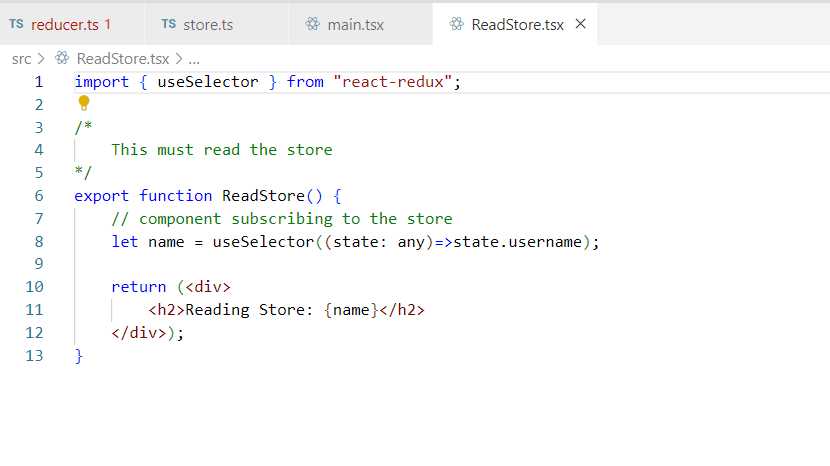
store.ts



main.tsx



ReadStore.tsx



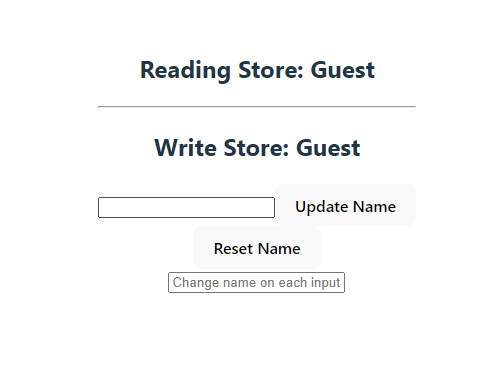
WriteStore.tsx



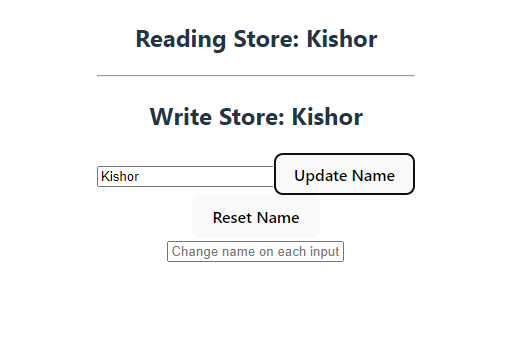
App.tsx



Output:



Enter some input and invoke update name button

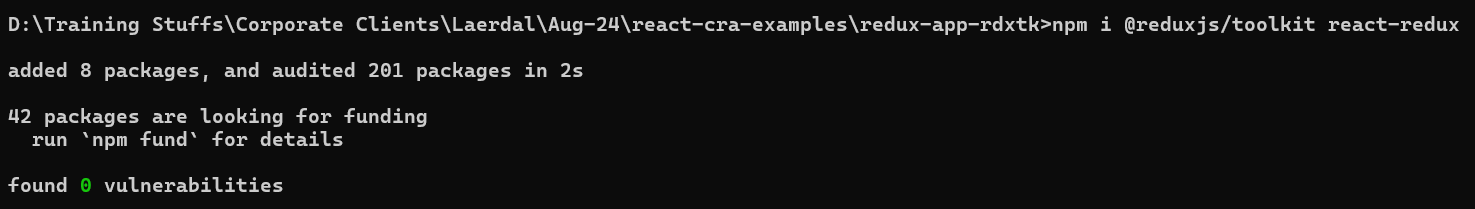


React Redux with Redux toolkit

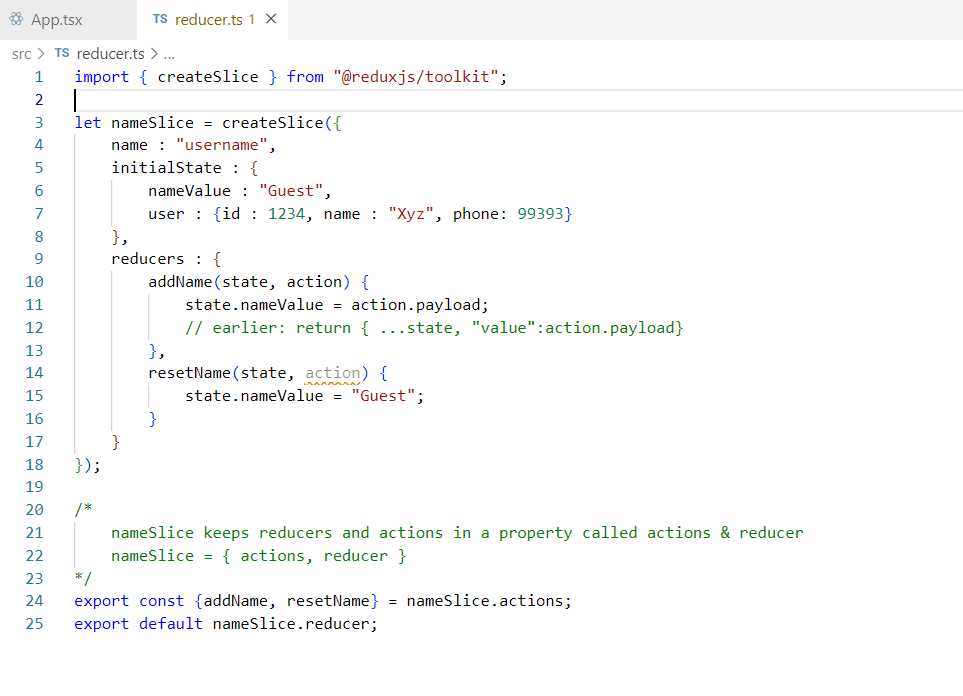
This simplifies writing the redux logic by eliminating the action types, createStore

You must download two libraries

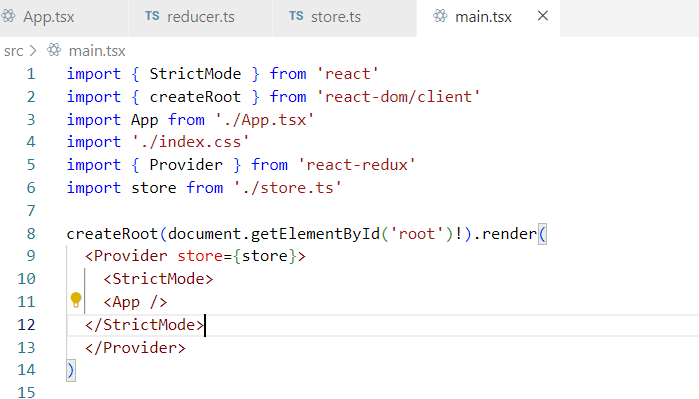
* @reduxjs/toolkit
* react-redux



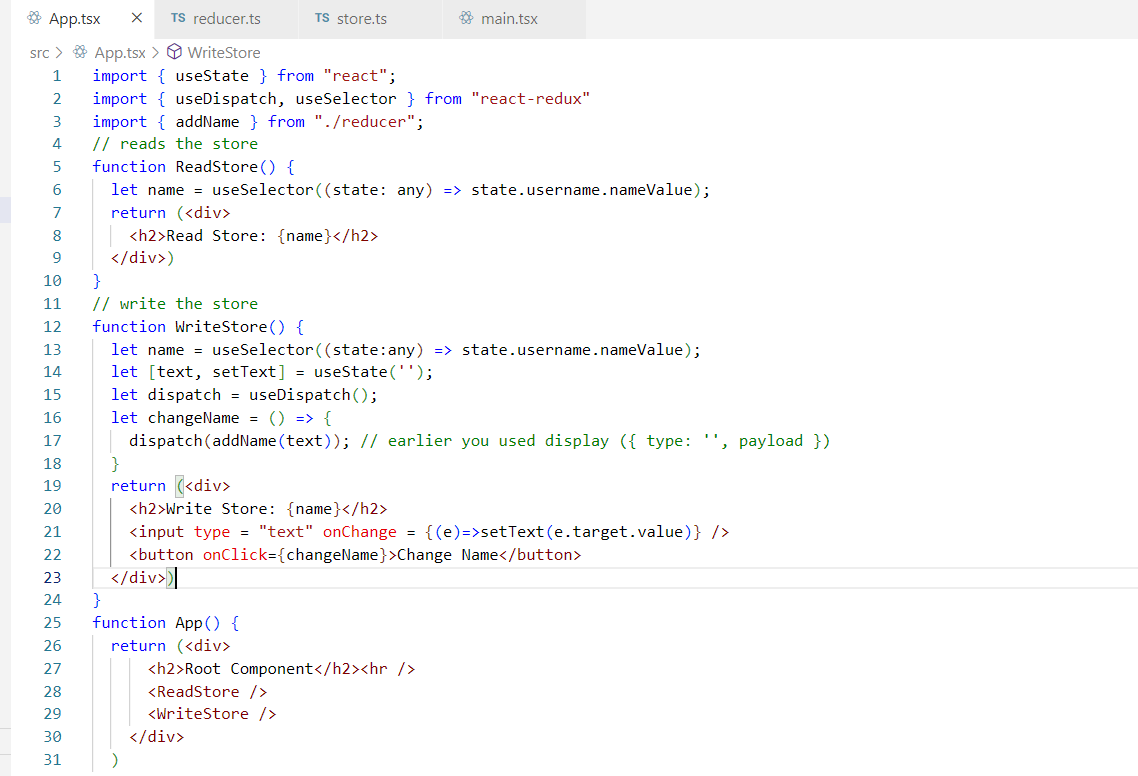
With Redux toolkit creating reducers is simplified with the help of createSlice



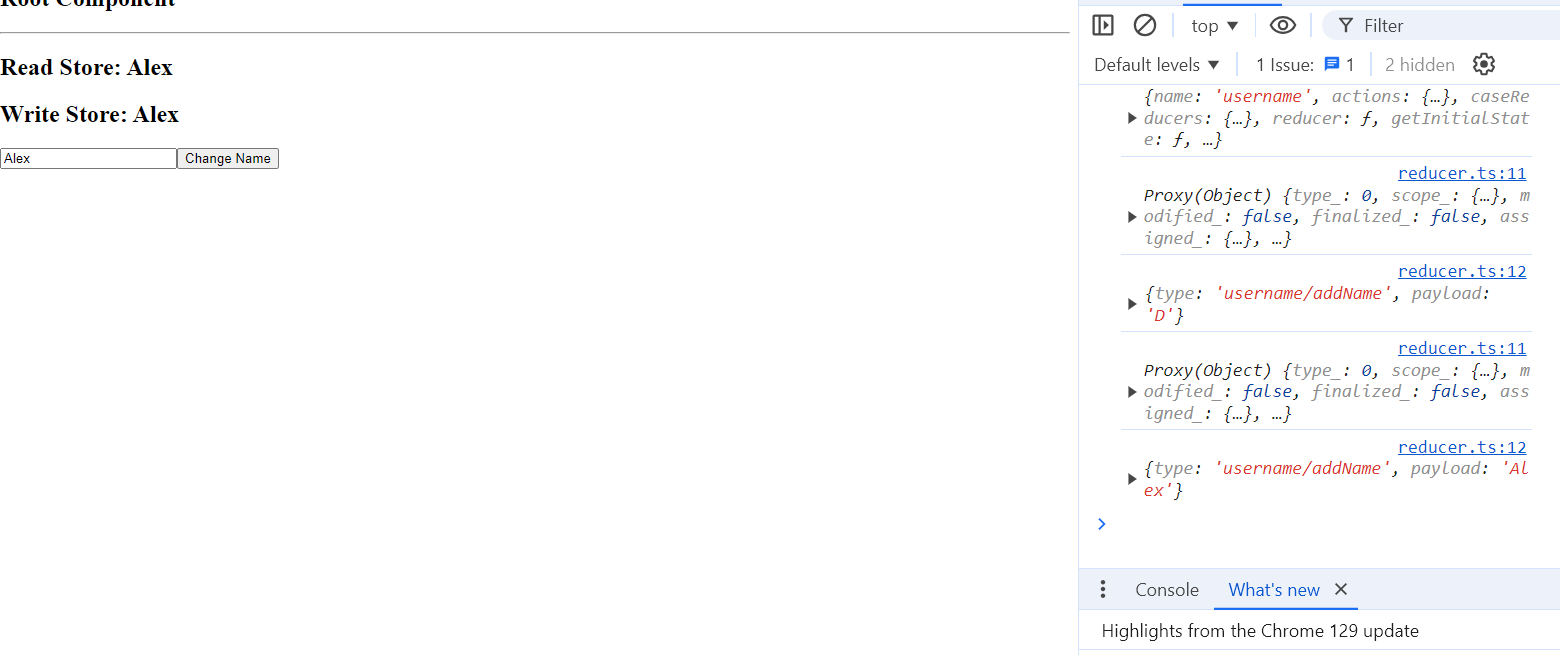
Adding the <Provider> in main.tsx



App.tsx



Output:



Redux official document

<https://react-redux.js.org/introduction/getting-started>

Unit test cases: This can test each component whether or not implemented as per the requirement

JEST: It is a testing library, by default it is not available in vite, it needs some configurations like npm start, npm test must run all the test cases

Using create react toolkit we will get inbuilt test environment

You can visit this official document to learning Testing in JEST & React

<https://legacy.reactjs.org/docs/testing-recipes.html>

<https://jestjs.io/docs/tutorial-react>

How to build & deploy

npm run build