React.js

* HTML
* CSS
* Javascript

HTML: It is mainly to display the content

CSS: It is to style the HTML

Javascript: It is to add effects to the web page by changing HTML & CSS at runtime

Software requirement

1. VS Code
2. Live server plugin: To auto-reload the changes and give a live preview

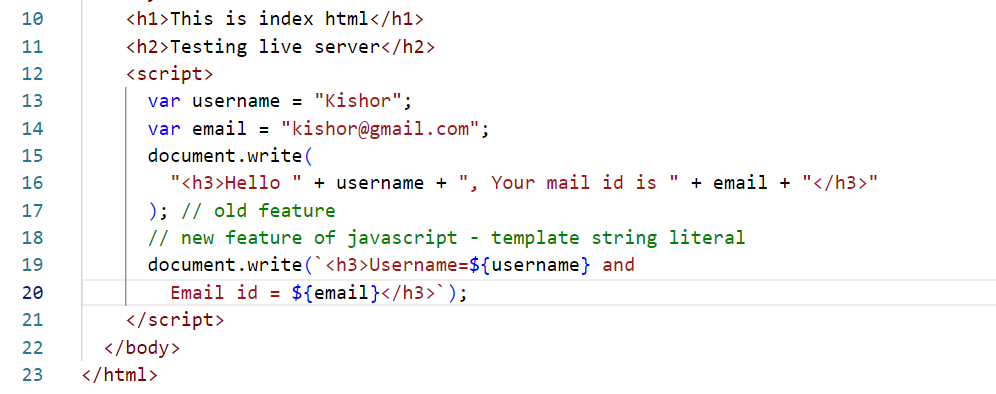
New Features of ES6 or Javascript

Template String literal:

It avoids you to break the string i.e., using + operator to concatenate string and the expression

var name = “…”;  
var email = “…”;

`<h3>Your name is ${name}, email id ${email}</h3>`



Classes & Constructors to create object

ES6 introduced class & constructor keyword to create the objects, earlier developers had to use functions as a constructor to create the object.

Old approach

function Employee(id, name, salary) {  
 this.id = id;  
 this.name = name;  
 this.salary = salary;  
}

var obj = new Employee(123, “Raj”, 45000);

obj.id, obj.name, obj.salary will access the properties

Suppose you want to add a function to the object, then you must use prototype.

Employee.prototype.display = function() {   
 document.write(….);  
}

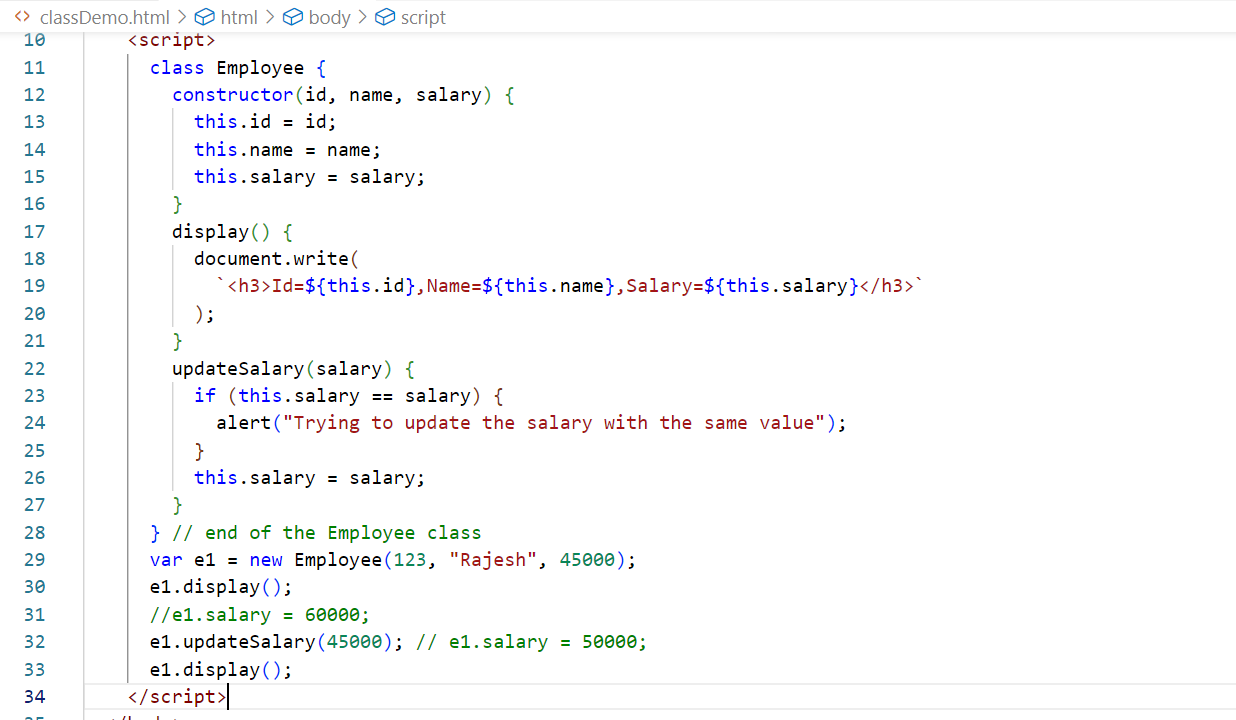
obj.display();

New approach: ES6 introduced classes & constructors to simplify creating objects

class Employee {   
 constructor( id, name, salary) {   
 this.id = id;   
 this.name = name;  
 this.salary = salary;  
 }  
 display() {   
 document.write(….);  
 }  
}

let obj = new Employee(2345, “Rajesh”, 42000);  
obj.display();

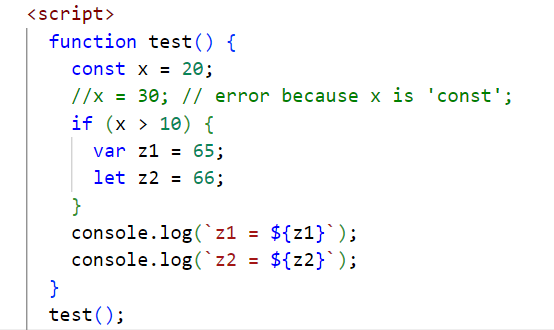
example



let & const keywords

They are used to create block scope variables, var was the keyword which was earlier used to create variables it is not a block scoped variable, it must be avoided

if(…) {  
 var cal = x + y;  
 let cal2 = x + y;  
 const cal3 = x + y;  
}  
you can access cal variable outside the if block



Event Handling: This was there even before ES6, it is used to detect the element generated the event & perform some action (executing some code)

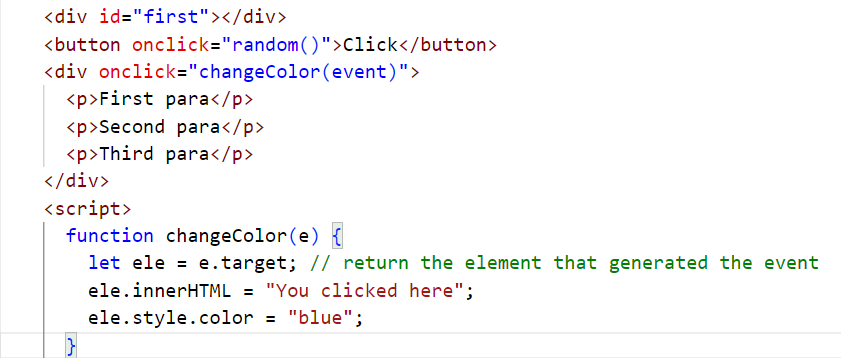
In Javascript an event is referenced using an event property, list of events that an element can generate

1. onsubmit
2. onclick
3. onmouseover
4. onmouseout
5. onfocus
6. onblur
7. onchange

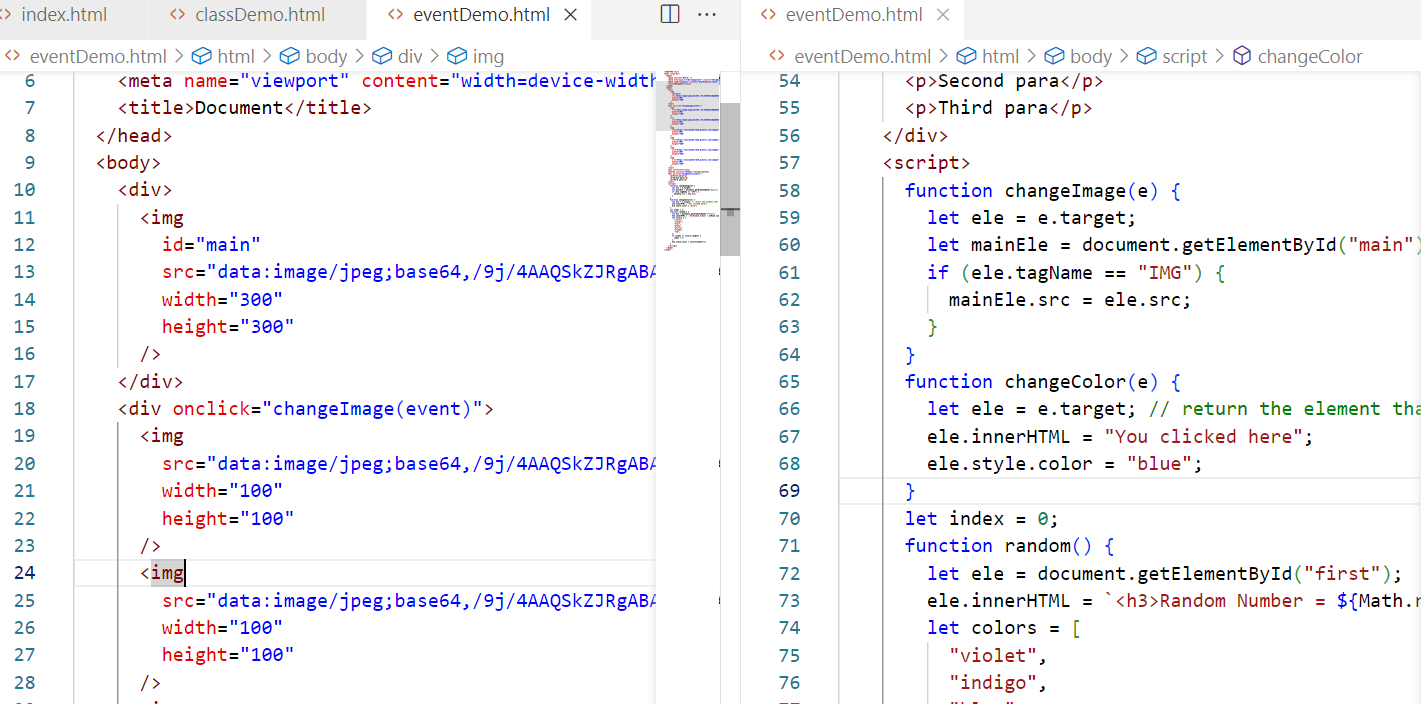


Accessing the HTML element using event object

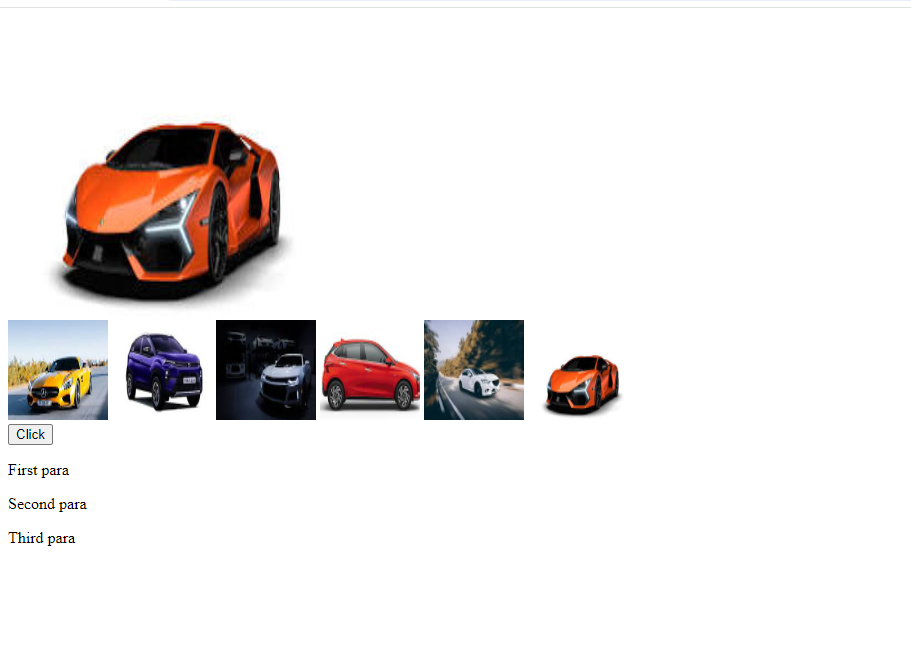
Every HTML element that generate the event can be identified by using the event.target property



Accessing the attributes using the events



Output:



Callback functions: These are the functions that are initiated now but executed later, i.e., when certain event occurs or when a response arrives

Callback functions are generally passed as a parameter to another function

ex: when you iterate an array you can use some inbuilt functions like forEach(), map(), sort() and so on.

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

for(let index = 0; index < items.length; index++) { … } // traditional approach

You can also iterate using forEach function, which is part of an array

items.forEach( callbackFn(value, index){} )

Note: forEach automatically calls the callbackFn by passing the iterated value to the 1st parameter & index to the 2nd parameter

Note: You can give any name to the parameters.

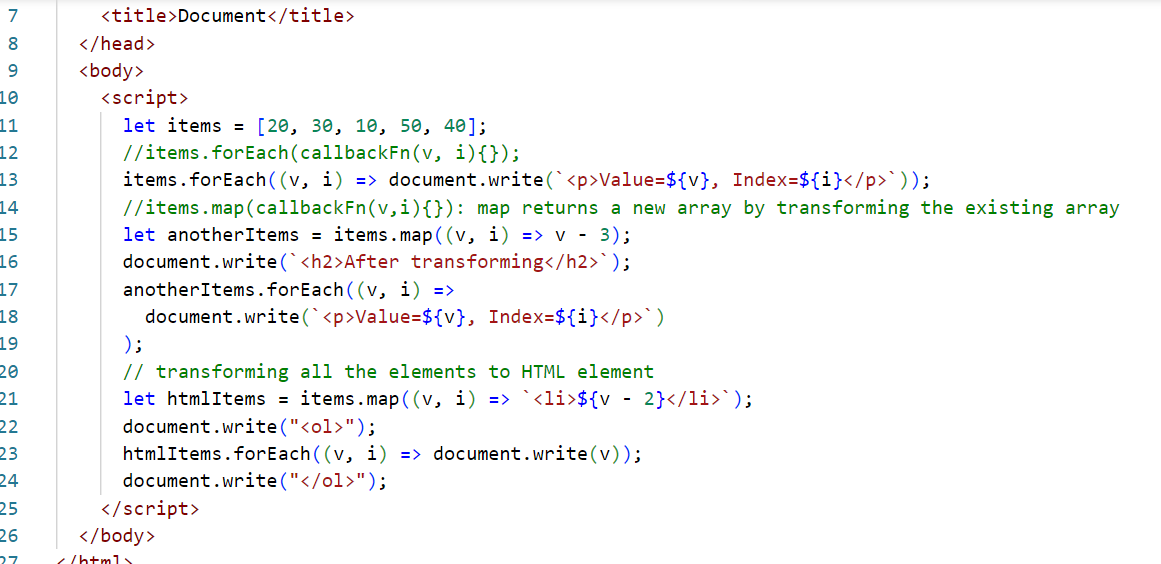
Arrow function:

It is introduced in ES6 to simplify writing callback functions, if the callback is only one line then no need to use { } or return if in case callback returns value, and also no need to use function keyword

Note: { } & return is required only if the callback has more than one line

|  |  |
| --- | --- |
| Callback | Arrow Function |
| function(v) { one-statment } | (v) => one-statement  [or]  v => one-statement |
| function(v) { return value; } | v => value; |
| function(x, y) { return x + y; } | (x, y) => x + y;  [or]  (x, y) => { return x + y; } |
| function(x, y) { console.log(x+y) } | (x, y) => console.log(x+y); |
| function(x, y) { 1st line; 2nd line } | (x, y) => {1st line; 2nd line } |
| function(x, y) { 1st line; 2nd line; return value } | (x, y) => {1st line; 2nd line; return value; } |
| ex: forEach ( function(v, i) { } ) | ex: forEach( (v, i) => { } ) |
| ex: map(function(v, i) { return v + 2; } | ex: map( (v, i) => v + 2 ); |

Replacing all the callbacks to arrow function



Activity:

Assuming you have an array of objects, print those items in an HTML table, below is the array  
Input:

[

{id:1, name:”Vijay”, address : {state:”KA”, city:”BLR”, pin:560001 } },

{id:2, name:”Ajay”, address : {state:”KA”, city:”MYS”, pin:560061 } },

{id:3, name:”Ravi”, address : {state:”MH”, city:”MBI”, pin:760001 } },

{id:4, name:”Siddharth”, address : {state:”TN”, city:”CHN”, pin:660001 } }

]

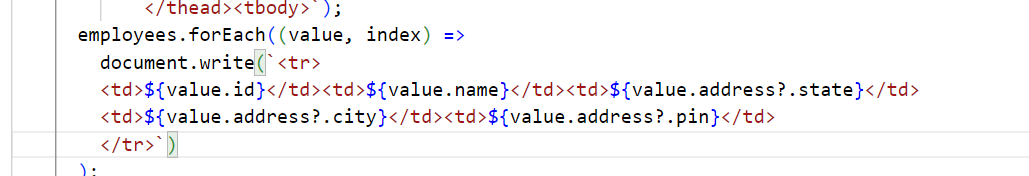
Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | State | City | Pin |
| 1 | Vijay | KA | BLR | 560001 |
| 2 | Ajay | … | .. | .. |
| … | .. | … | .. | … |
| …. | .. | . | .. | . |

TypeError: Whenever we try to access the nested properties there could be chance that nested properties may not be present, in that case you get type-error, to avoid this developers used if(value.address != undefined) { then access value.address.state }

Optional Chain(?.): ES8 added new feature called Optional chain that will access the nested property only if the property exists

value.address?.state // if address is not undefined then access state.



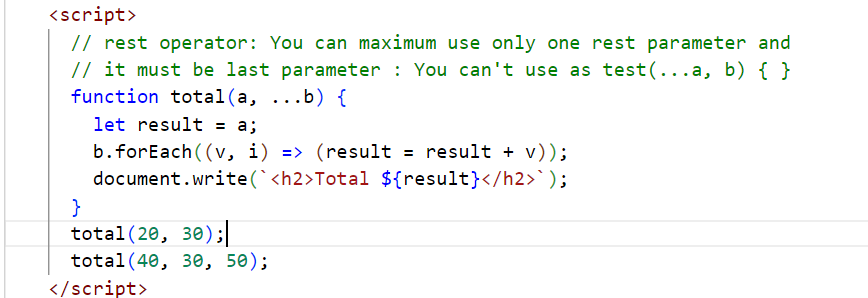
Rest & Spread operators

Rest operator: When a function needs to accept an optional parameter or 0 or more parameters, then you can use rest parameter

Earlier data’s were lost when there are no parameters to accept some values

ex: function test(x, y) { }   
now you can call test by passing only 2 parameters, but if you pass more than 2 parameters then all those data will be lost

function test(x, …y) { } // here y can accept 0 or more parameters, y also acts like an array



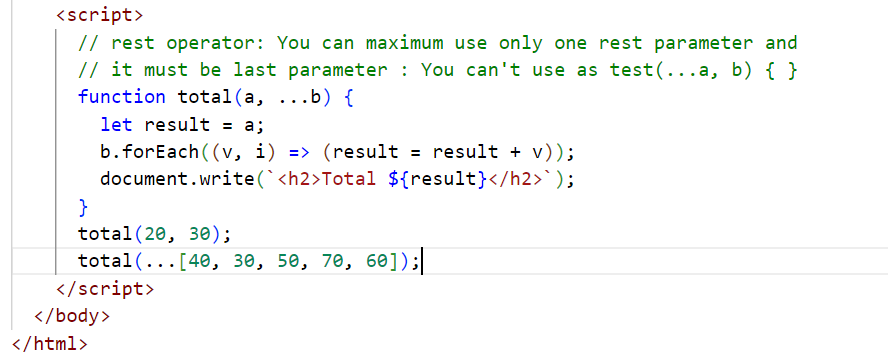
Spread operator: It is similar to rest parameter, however used to distribute the list of values to multiple parameter of a method, used while calling the method

function test(x, y, z) { }

let items = [20, 10, 30];  
test(items); // x = [20, 10, 30], y = undefined, z = undefined

// spread operator  
test(…items); // x=20, y=10, z=30;

Note: If the number of parameters in the function is less, then the last parameter must be the rest parameter, so that the last parameter accepts rest of the values



Destructuring: It is used to assign values of the array or an object to a separate variable in a single line.

Earlier:   
let items = [20, 10, 30, 50];

If each item needs to be assigned to separate variables

let a = items[0];   
let b = items[1];  
let c = items[2];  
let d = items[3];

let obj = { id : 1, name : “Raj”, salary: 45000 };  
let id = obj.id;  
let name = obj.name;  
let salary = obj.salary;

From ES6 onwards you can use destructuring

Array destructuring  
let items = [20, 10, 30, 50];

let [a, b, c, d] = items; // each item is assigned to the variables in the order they appear

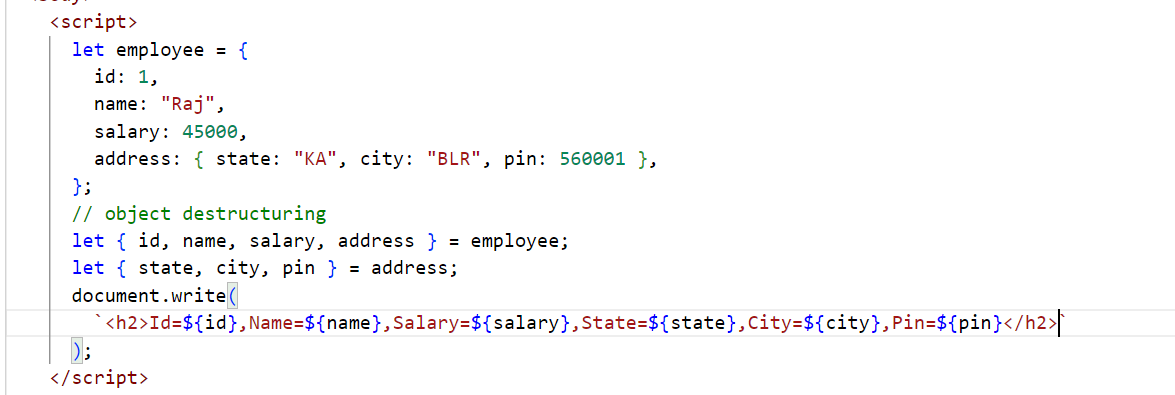
Object destructuring

let obj = { id : 1, name : “Raj”, salary: 45000 };  
let { id, name, salary } = obj;

function test([a, b, c, d]) // array destructuring  
test(items);

function display( {id, name, salary} ) { } // object destructuring

display( obj );



Default values: When a parameter doesn’t get any value this default value will be given

function test(x = 0, y = 0) {   
  
}  
test(); // x & y will be 0  
test(2); // x = 2, y = 0

How to access backend services from javascript

Whenever front-end sends the request to the backend services it gets the data in the form responses, there are few approaches javascript uses

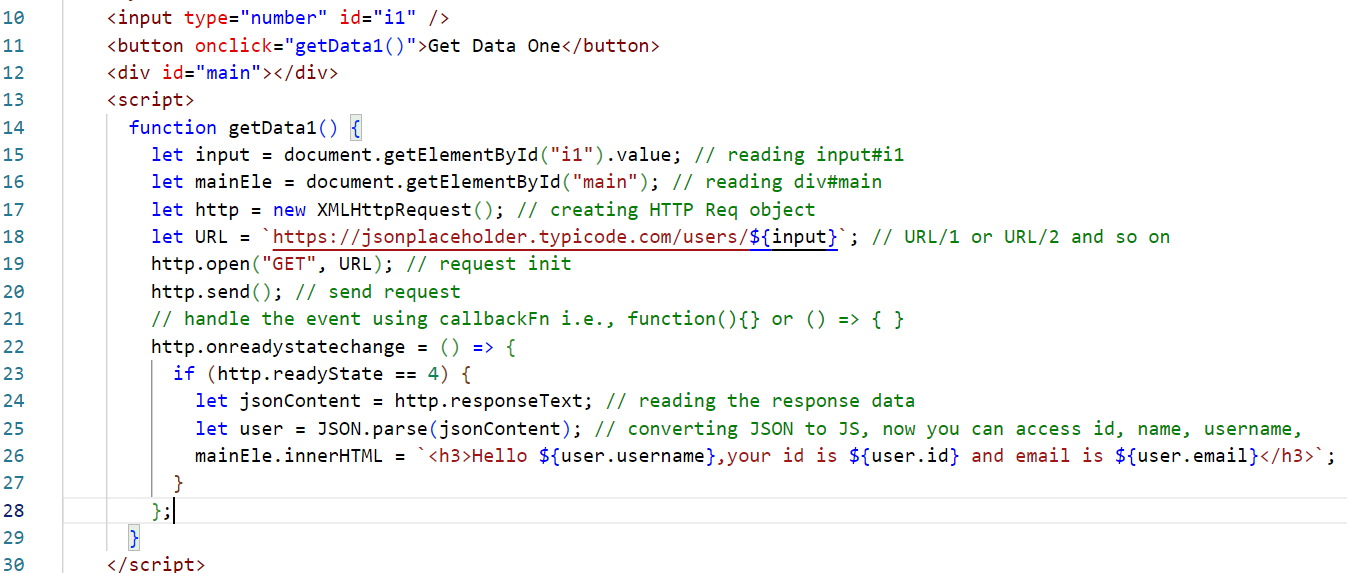
1. Old approach : XMLHttpRequest which is callback based
2. New approach: fetch which is Promise based
3. Enhancement to the Promise: async/await

XMLHttpRequest: It is an object used earlier to make HTTP calls(HTTP requests) to the backend, it gets the response through some events and invokes the callback based on those events, list of properties & functions in XMLHttpRequest

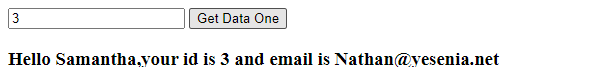
1. readyState: it maintains the value ranging from 1 to 4, if 1 then request is initialized, if 2 then request is sent, if 3 then half of the response is ready, if 4 then full response is ready means request is complete
2. responseText: it maintains the response data
3. onreadystatechange (everything in lowercase): this generates an event each time the readyState value changes & it invokes the callback attached to this
4. open(httpMethod, URL): to initialize the request with HTTP methods & URL
5. send(): to send the request

In order to send HTTP request to the json placeholder we must use HTTP GET method & an URL

<input id = “i1” > <button onclick=”getData()”>Get Data</button>  
function getData() {  
 let id = read the #i1.value  
 let url = “url/”+id;   
 let http = new XMLHttpRequest();  
 http.open(“GET”, url);  
 http.send();   
 http.onreadystatechange = callbackFn() { …when readState==4 read responseText. }  
}



Output:



Promise based:

Promise is an object that will have two status based on the request

1. Resolved: If the promise is fulfilled / success then it is treated as resolved
2. Rejected: If the promise is rejected/failed then it is treated as rejected

Promise provides two methods to handle success & failure, those are then(callback) and catch(callback)

then(callback): it is invoked when the promise successfully resolved

catch(callback): it is invoked when the promise is failed or when there’s any error

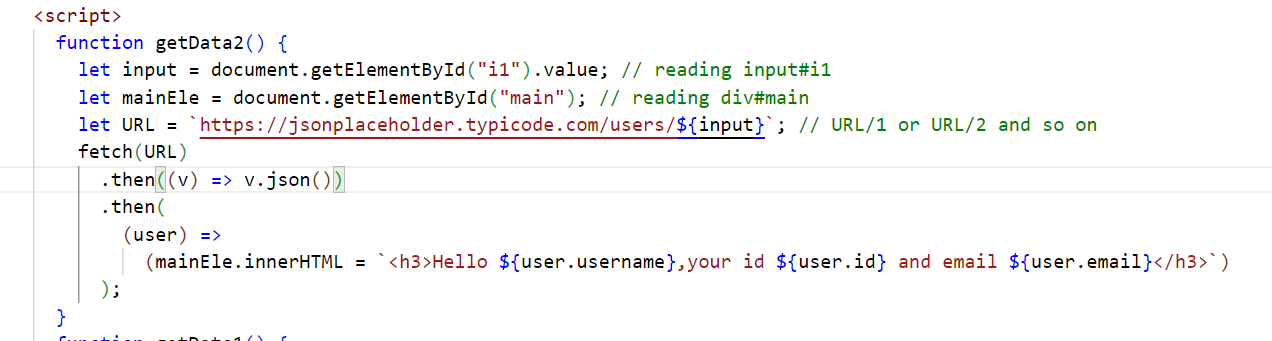
both of them takes callback which is executed based on the promise successful or reject status

fetch(URL): It is an inbuilt method that returns a Promise when the request is made, this can be resolved or rejected

fetch(URL).then( callbackFn ).catch( callbackFn )

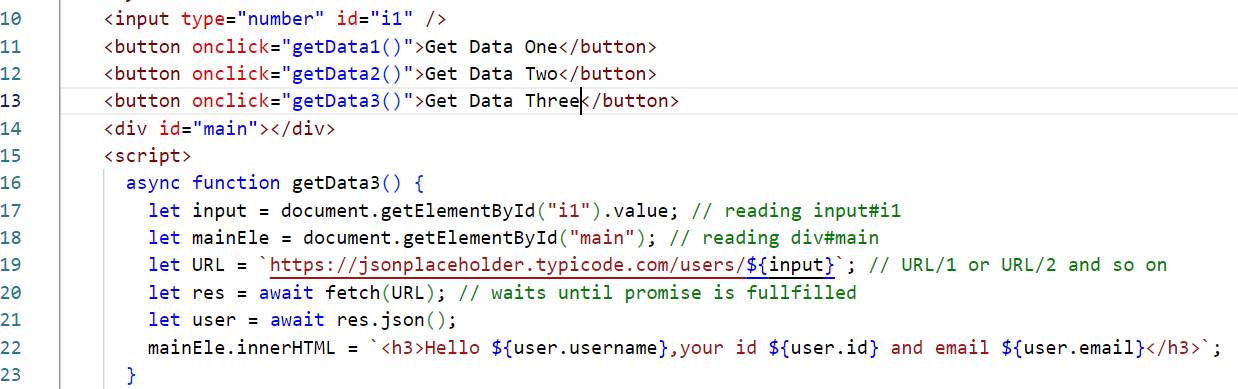
getData2() {  
 let id = read the #i1.value  
 let url = “url/”+id;   
 fetch(url).then( (res) => { } ).catch( (err) => { } )

}



Async/Await: It is used whenever you need to wait for the asynchronous operations to complete and then you want to proceed

async function getData3() {  
 let res = await fetch(URL) // no need to use .then() or .catch  
 print res; // executed only if the Promise is either rejected/resolved  
}



Getters & Setters in Javascript

These are functions that can read and write the data of an object

class Employee {   
 constructor(id, name, salary) { … }  
 set empId(id) { … }   
 get empId() { return this.id }  
 set empName(name) { … }   
 get empName() { … return this.name.toUpperCase(); }   
 set empSalary(salary) { if(salary == this.salary) alert(‘trying to update salary with same value’)  
}  
let e = new Employee(1234, “Rajesh”, 45000);  
e.empId = 2545; // calls empId(2545) >> id = 2545  
console.log(e.empId) // returns 2545 & prints in the console

React.js: It is a Javascript library used to develop single page applications (SPA).

SPA: Everything happens in one single page, it updates part of the page without reloading the entire page

Components: These are reusable UI’s which you can develop and use independently wherever is necessary

ex: Profile is a component that will have image & name that is used in comment, like, post, friends list etc in Facebook

React.js uses following languages

1. HTML
2. JSX - Javascript XML - Extended form of Javascript - It simplifies writing HTML code in Javascript

Note: Browser doesn’t understand JSX, hence React.js uses babel which is a transpiler that converts JSX to Javascript

HTML in Javascript

content = ‘<div><p>Hello ‘+name+’</p></div>’

or

content = `<div><p>${name}</p></div>`

HTML in JSX

content = <div><p>{name}</p></div>

React.js uses 3 libraries to develop the application

1. React: It is to create components & other features
2. React DOM: It updates the Real DOM i.e., browser DOM
3. Babel: It compiles/transpiles JSX to Javascript

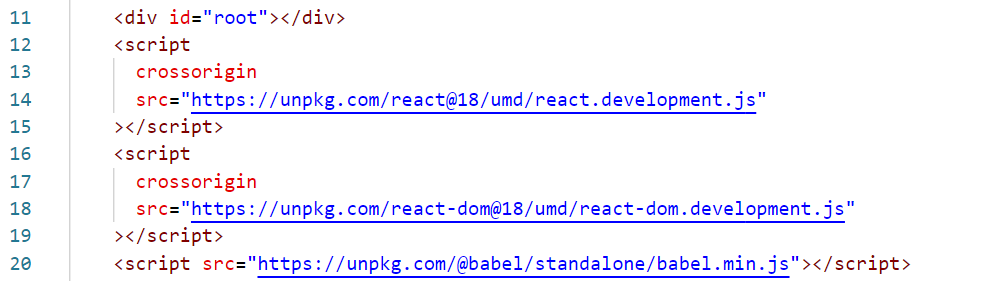
React applications can be created either by

1. Using Online CDN Link - this is only for learning purpose
2. Using React Tool Kit - provided by react community - this is used in real time

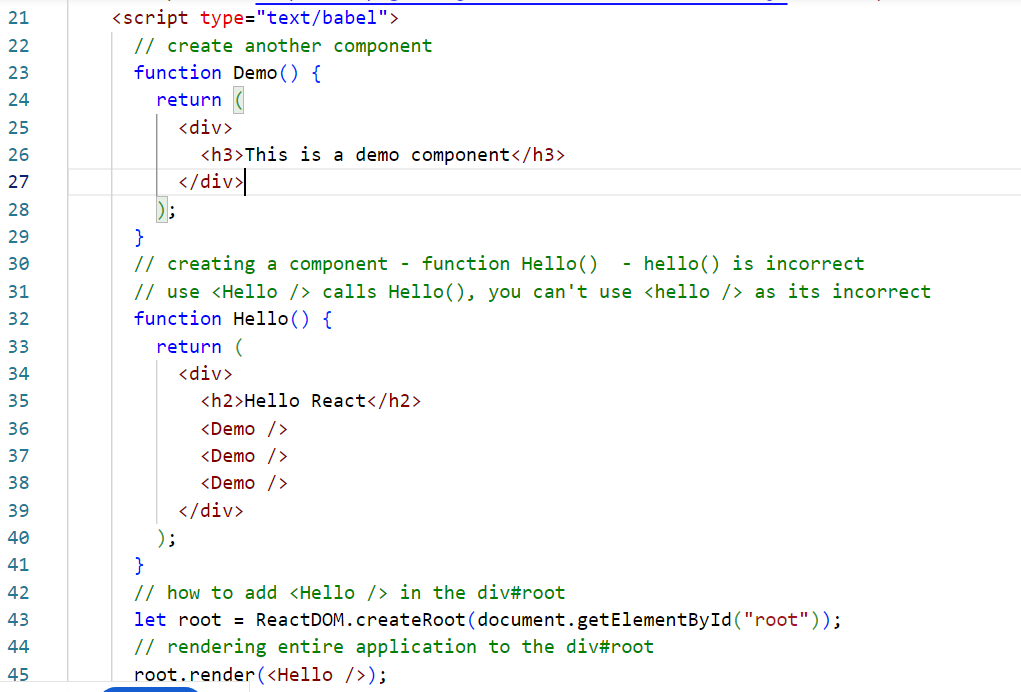
Root node: It is an HTML element which loads all the components

Root component: This is the component that is going to nest all other components, and it is loaded into the browser DOM

Adding CDN links

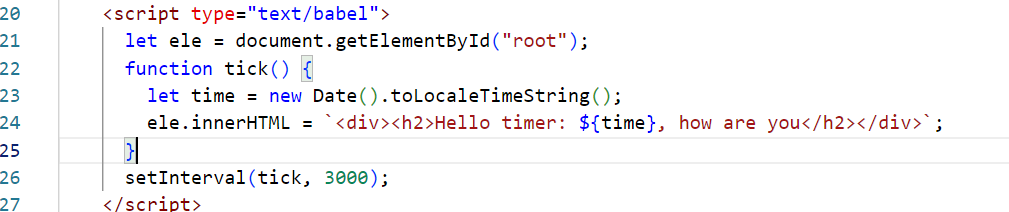


Script with text/babel

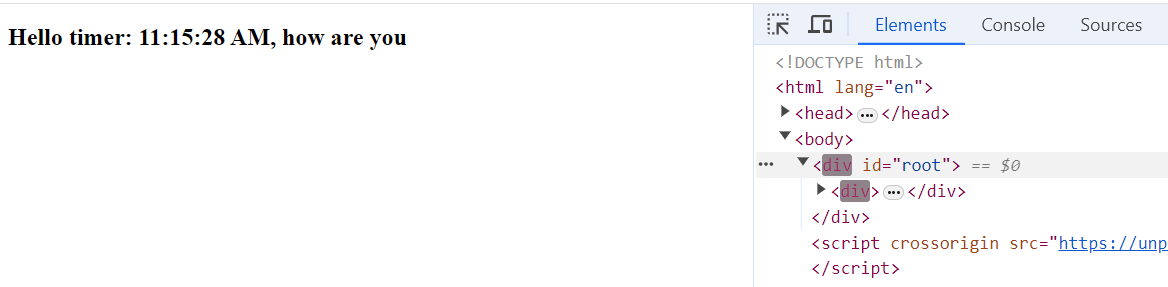


ReactDOM: It is a virtual DOM that updates the original DOM by comparing the original DOM content with virtual DOM content, it only doesn’t update the DOM if the contents are same & does changes only to the content that needs

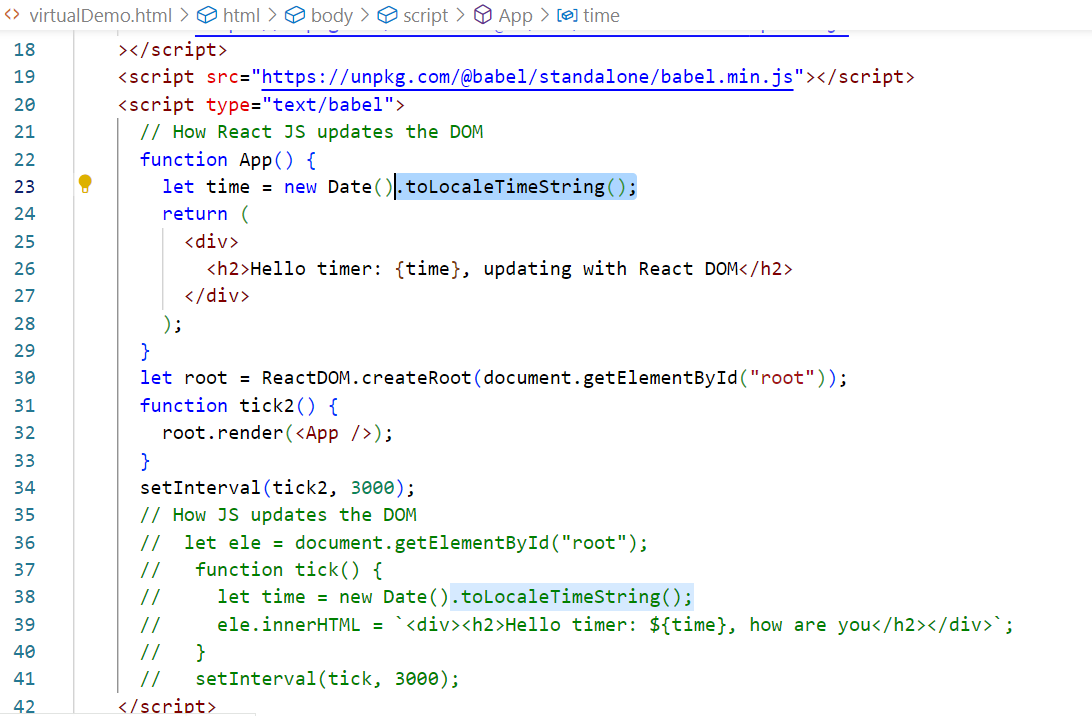
Javascript doesn’t compare the previous content while updating the DOM



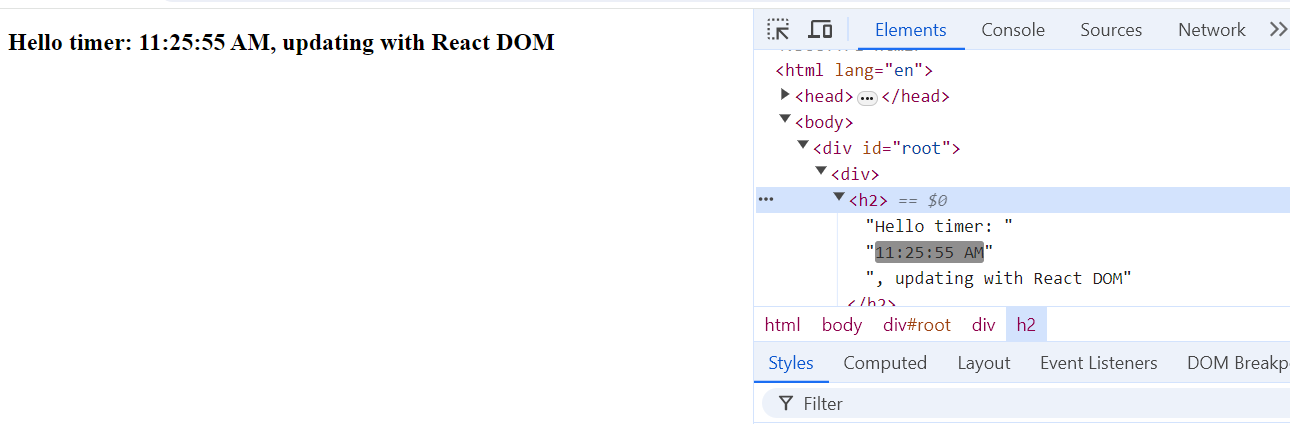
Output



It updates the entire DOM, just to modify the time, however React.js resolves these issues with Virtual DOM



Output:



React Tool Kit: It is provided by React community to quickly create a ready to run react application

Benefits

1. It organizes all the files and folders in a standard way

ex: Keeps all the JS in src, html in public folder

1. It comes with inbuilt liver server
2. It automatically compiles the code
3. It provides commands to launch, test and build the application

Ways to use react tool kit

There are 2 ways

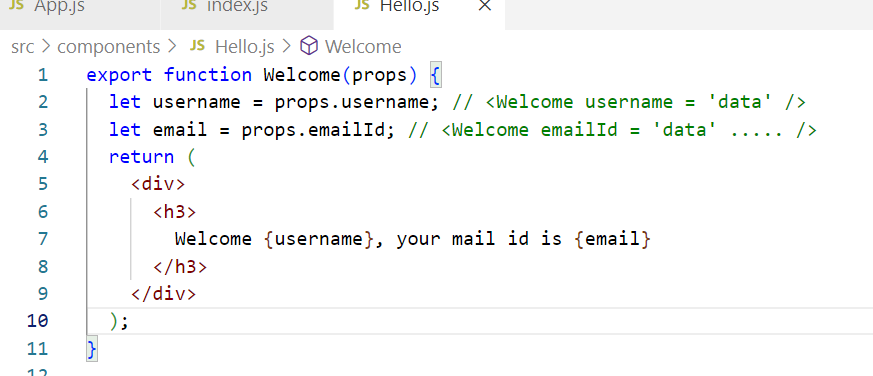
1. Installing & creating the project: npm install -g create-react-app, then download the project using create-react-app app-name
2. Directly downloading the project without installation: npx create-react-app app-name

Props: These are components data that can be supplied from one component to another component

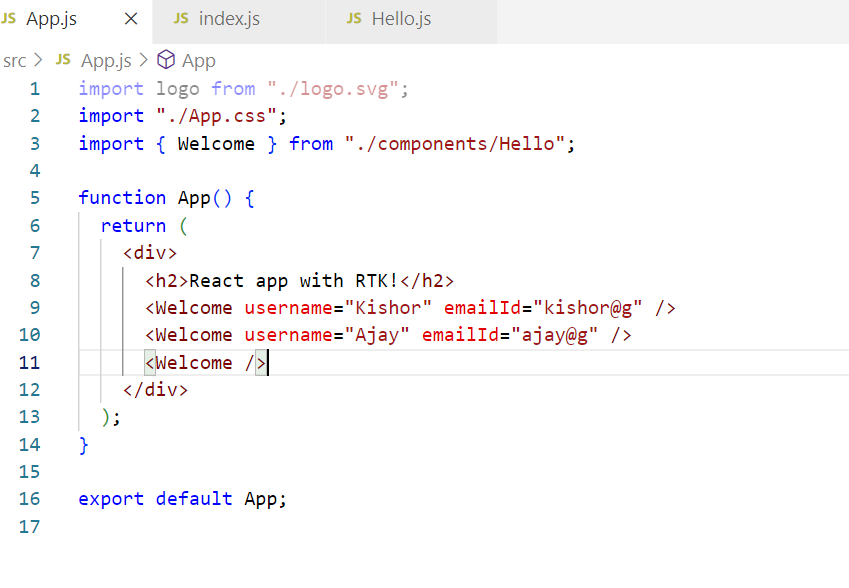
<Welcome username=”Kishor” emailId=”kishor@gmail.com”/>  
<Welcome username=”Ajay” emailId=”ajay@gmail.com” />

function Welcome(props) {   
 let username = props.username;  
 let email = props.emailId;  
 return Welcome {username}, your email is {email};  
}

Hello.js



App.js



Conditional rendering

Based on the conditions you can render the content

if(username==undefined) { return <div>Welcome Guest </div>}  
else { <div>return {username} your id {email} </div> }

OR: You can use ternary operator( ?: )  
return (username == undefined)   
? <div>Welcome Guest</div>   
:   
<div>Welcome { username } </div>



Activity:

Create a component Avatar that displays the image, reuse this component inside Comment component

Create a component Comment that reuses Avatar component by passing image url and width & height 100 \* 100 to the Avatar component, comment component must display Avatar & below that username, comment text and date.

App component passes comment to the Comment component as below

<Comment comment = { username: “Raj”, imageUrl : “googleImageURL” , text: “some text” date = “2024-06-14”} />

Output:

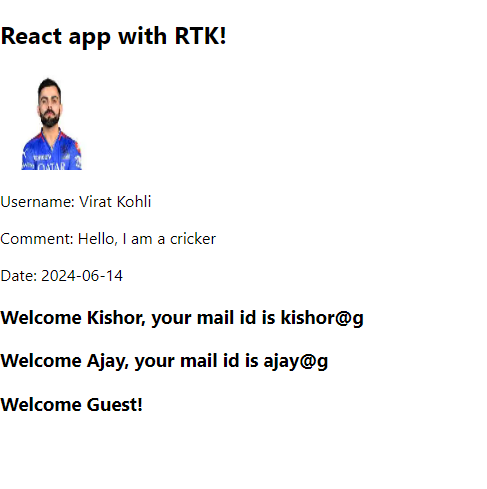
Image  
Username: “Raj”  
Your comment : Some text  
Date : 2024-06-14



App.js



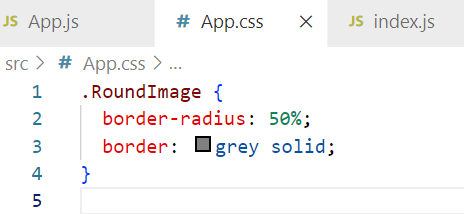
Output:



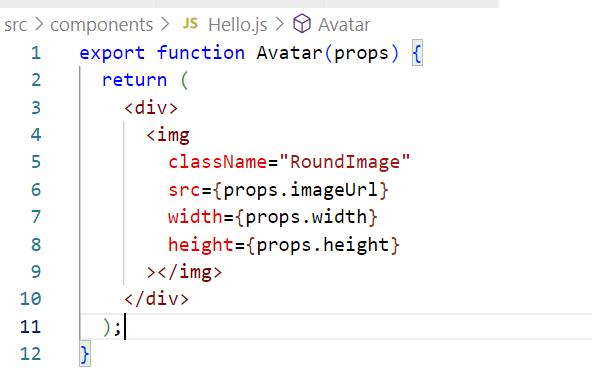
How to apply CSS in React

1. You can write classes in .css files & use their names
2. You can download third party css and use their classes

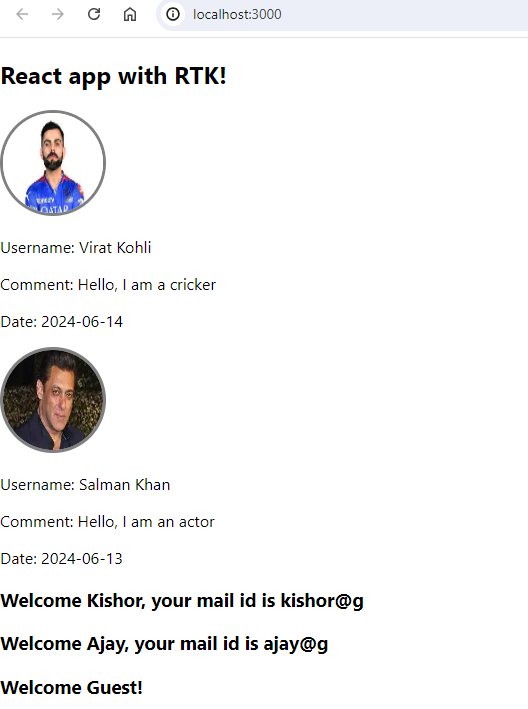
App.css



Hello.js



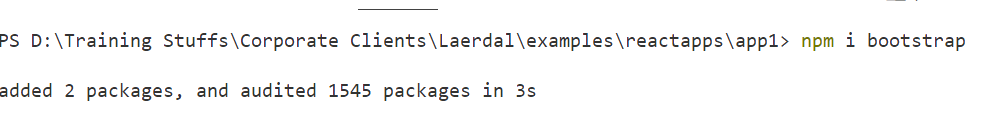
Output:



Adding 3rd party CSS

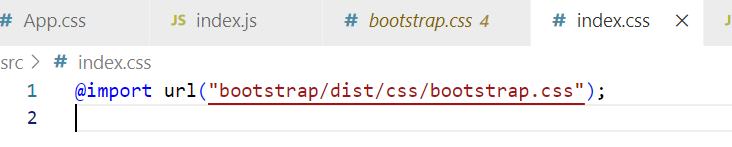
You must download the css in your project, so that it will be part of the build also, you must import either in index.css or index.js

npm install bootstrap or npm i bootstrap



Note: You can observe the bootstrap folder inside node\_modules

index.css



How npm start works

npm start >> package.json >> runs scripts present in node\_modules >> they launch index.js & index.html

How to change port number

React.js internally uses .env file that has server configurations, you can create .env file inside the project & mention the port

Types of components

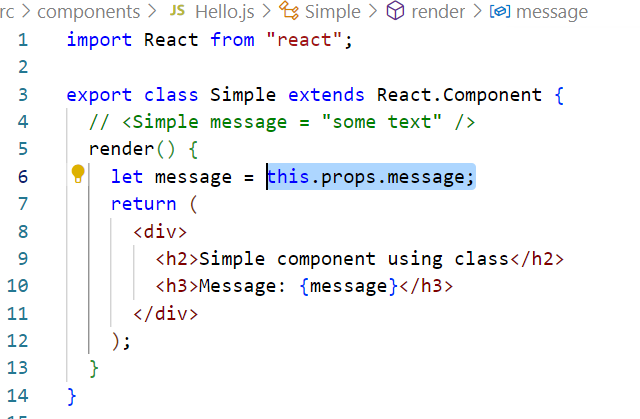
1. Function based component - components which you create from functions
2. Class based component - components which you create from classes

Difference between class based components & function based component

|  |  |  |
| --- | --- | --- |
|  | Function | Class |
| Usage | function App() { return content } | class App extends React.Component {   render() { return content } } |
| Features before 16 | Components are read-only, supports only props, but no states | Components are read and write both, supports props(read-only) and states(read & write) |
| Feature before 16 | No life cycle methods | Has life cycle methods |
| Features above 16 | Function supports all the features class supports because of react hook functions |  |

How to create components using classes

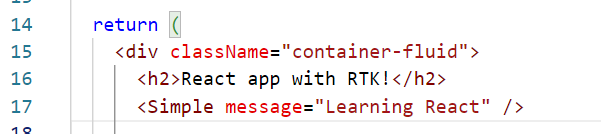
class Demo extends React.Component {   
 render() {   
 // automatically called to render the content  
 }  
 componentDidMount() { } // life cycle method called when component is added to the DOM  
 componentWillUnmount() { } // life cycle method called before component is removed from the DOM  
}



Render the component in the App component

<Simple message = “some text message” />

src/App.js



Stateful components

Components that can have states which are modifiable, earlier functions were not supporting the states only classes were supporting the states, but from react v16 onwards functions also support the states.

How to create states in functions

React has provided an inbuilt hook function to support states in the function based components, this hook function is available from react v16 onwards.

Note: hooks can be used only in function based components, it can’t be used in classes

useState(initValue): It is a hook function, that provides a variable & a function to modify the variable value

let [username, setUsername] = useState(“Guest”)  
let [email, setEmail] = useState(“Not Provided”);

To modify you must use the 2nd property of the useState i.e., setUsername & setEmail

How to modify the state

1. You must handle an event & modify the state
2. You must get the data from the backend & modify the state

How to handle the event & modify the state

React handles the events using the event names which is similar to Javascript events, but it uses camel-case

onclick = onClick  
onsubmit = onSubmit  
onchange = onChange  
onblur = onBlur

We must attach an event handler to these events, which are executed when the event occurs

<button onClick = { ()=> { statements } >MyButton</button>  
[or]  
let handleClick = () => { …. }   
<button onClick = { handleClick } >MyButton</button>