SQL

REST

WEB

UI

(React JS App)

Web

Server

Application

(Spring Boot Application)

DB

(DB2/Maria-DB/Oracle/MySQL/MS-SQL/H2)

Development Environment Check:

Check for node js:

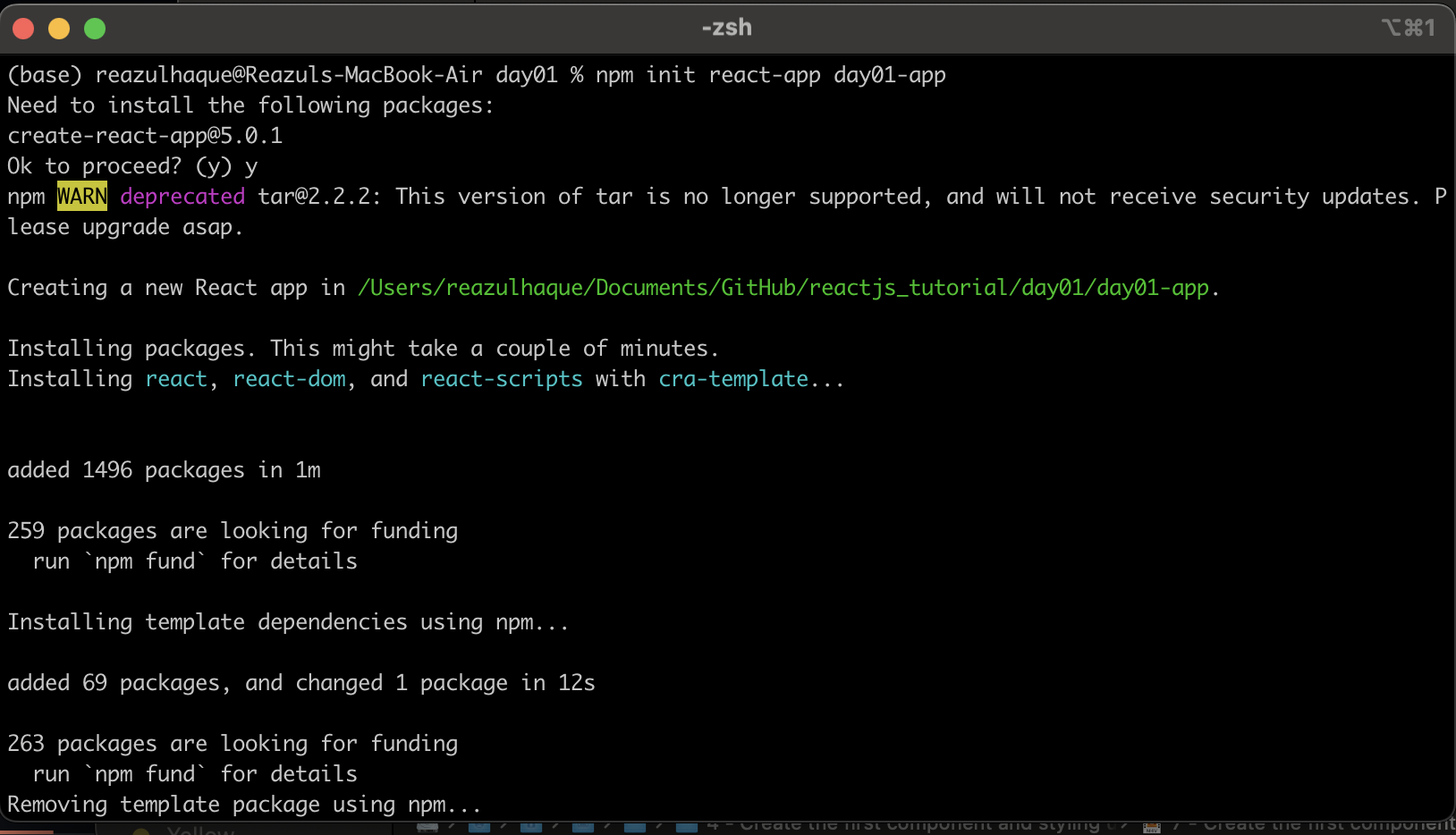
node --version

Check for npm version:

npm --version

Create a Ractjs application

npm init react-app day01-app



Go to the application folder

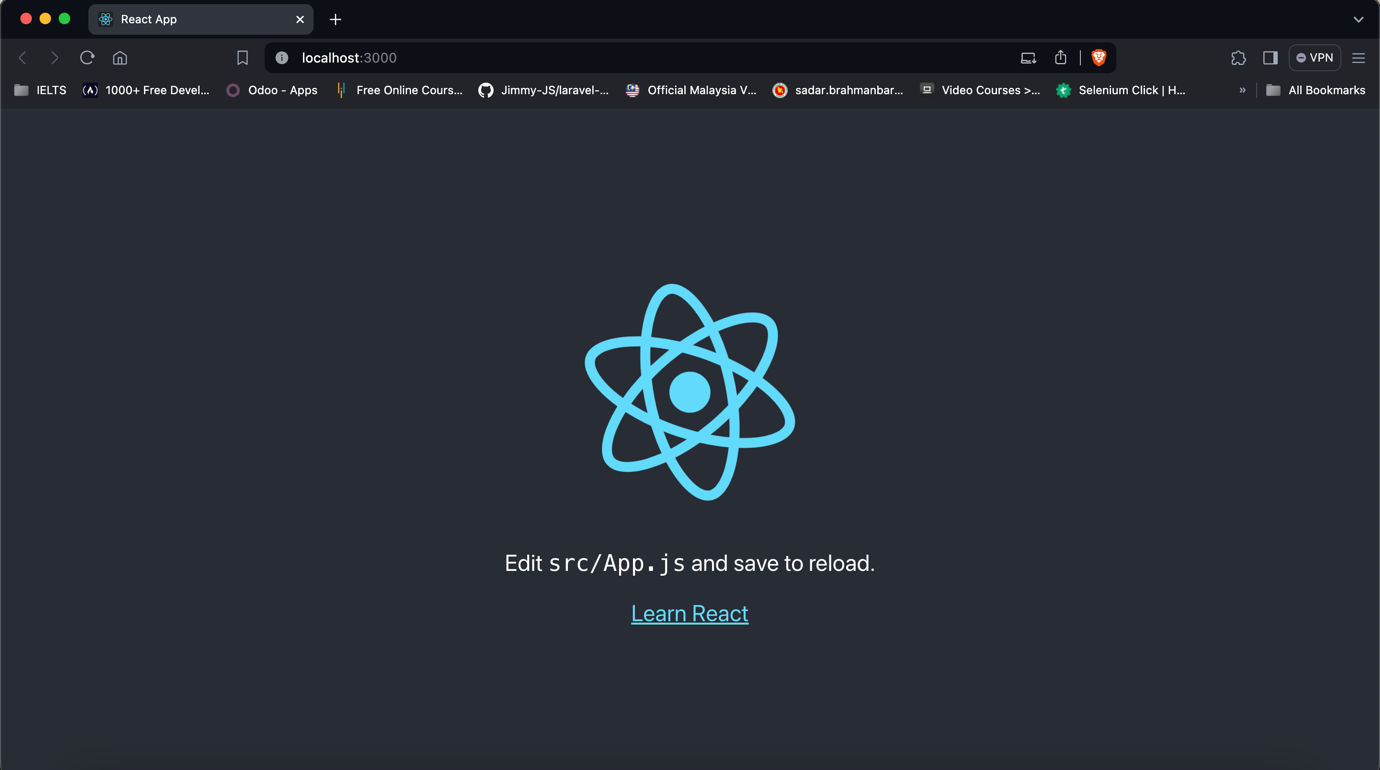
cd day01-app

Then run

npm start

This command will run the reactjs application on default port(3000). If the default port used by other application, reactjs will try to use next available port for 3001. It’s a incremental way of searching free port, means next port to serach is 3002, 3003 and so on….

Fig:Reactjs\_defaultPage



Let’s look to the folder structure:

We’ll use Visual Studio Code (VS Code) for reactjs development, you can also use your favorite IDE for reactjs. Let’s add the application folder to VS code workspace.



**node\_modules**: this folder contains all dependent libraries required by reactjs

**public**: all public files are in public folder

**src**: this is our main folder, where all codes belong.

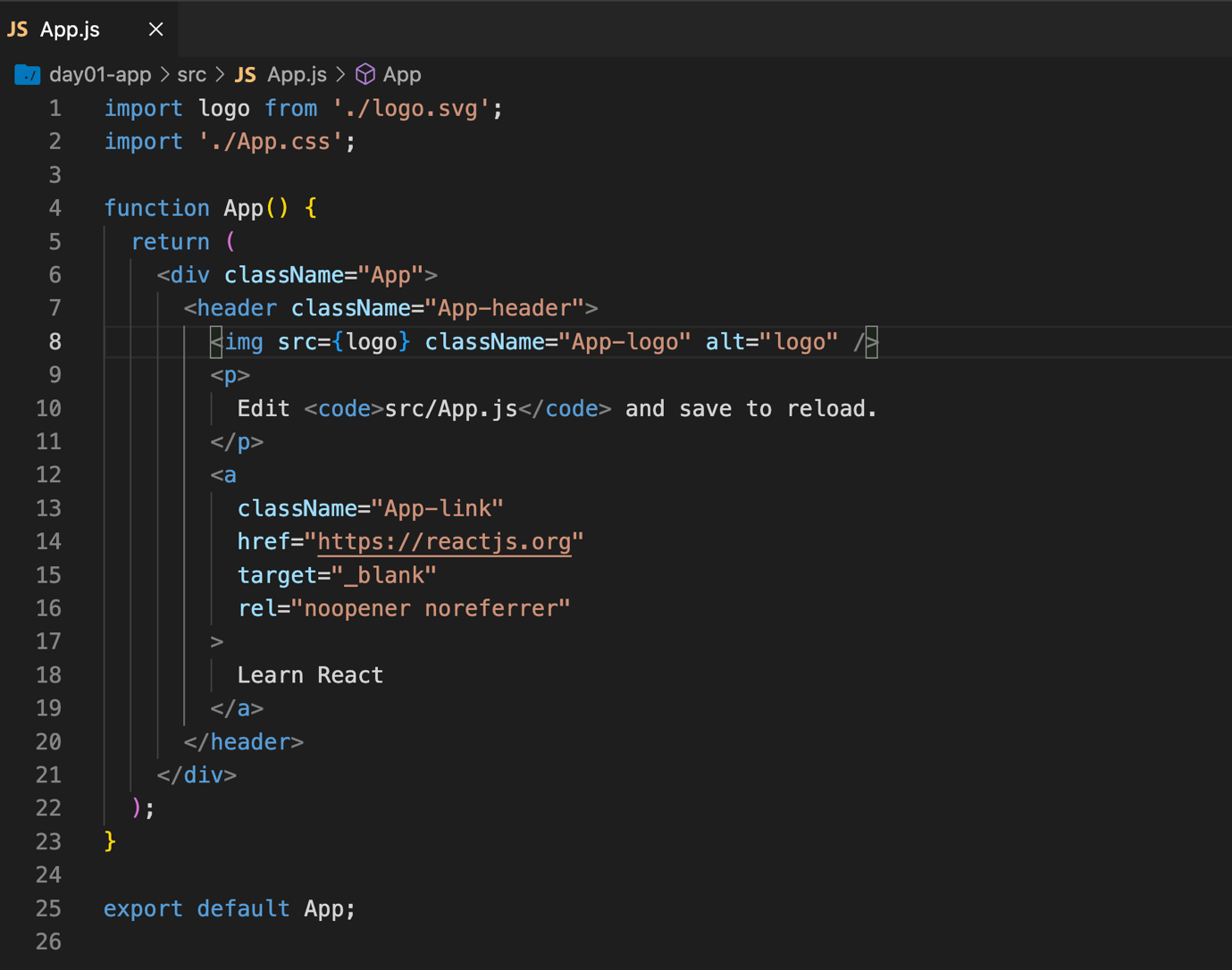
**.gitignore**: it’s git file for ignoring which file or folder we don’t add to git e.g node\_modules folder. We don’t need to add this folder to git.

**package.json**: it’s a dependency manager file like pom.xml. We can add our required library in this file.

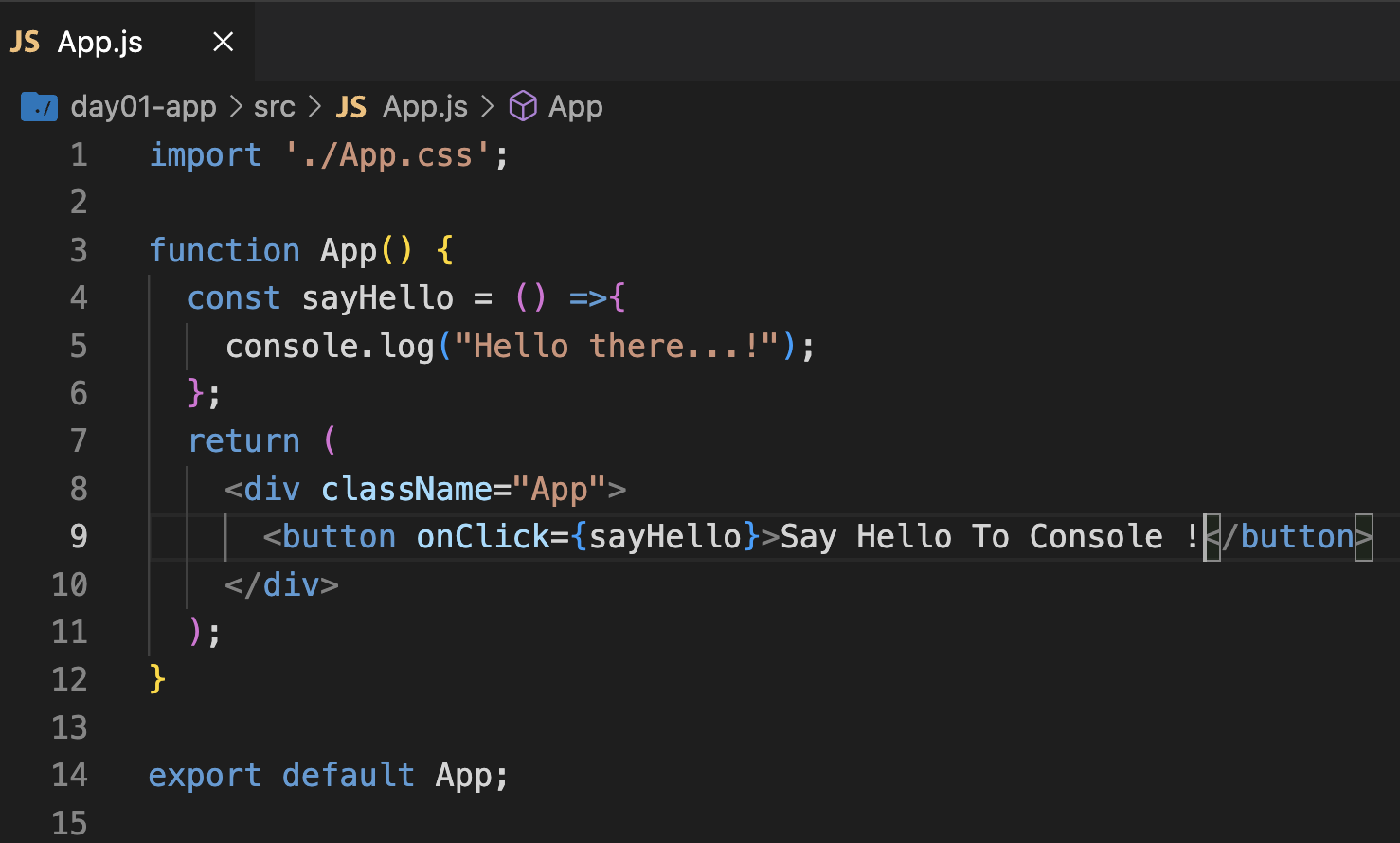
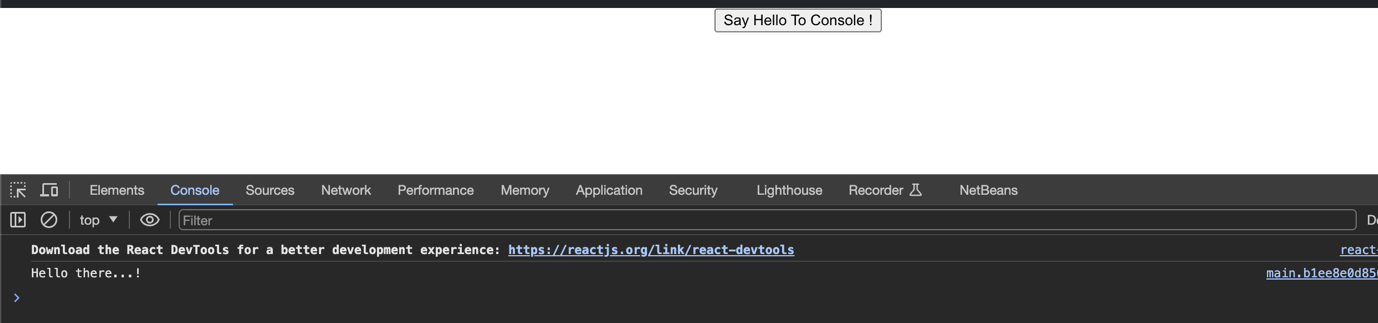
**README.md**: default file for git, to describe about the application or whatever comments we want to add for others.

Edit Code:

src/App.js is the default file from reactjs



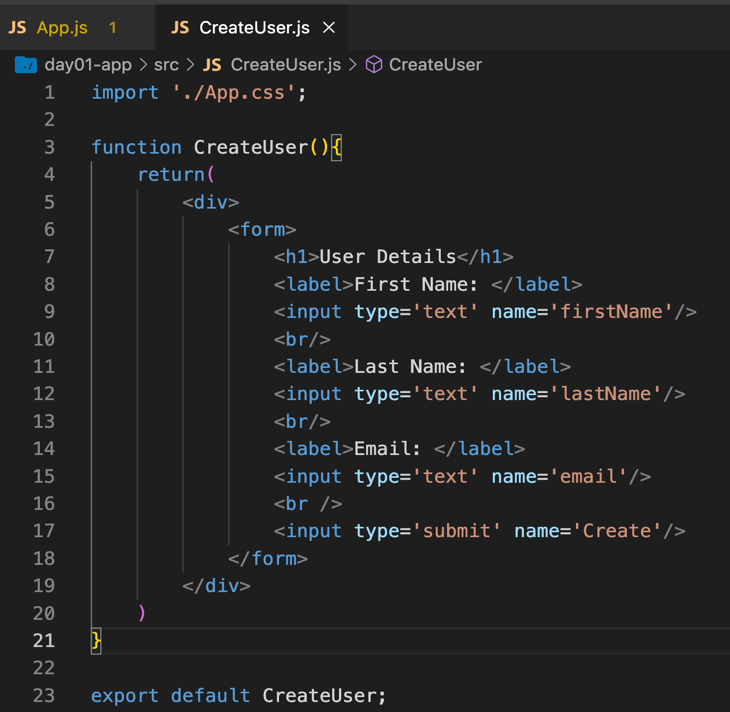
This is the default code by reactjs while we create our application. This code rander in browser as in Fig:Reactjs\_defaultPage. Now we’ll add our own code to src/App.js file:



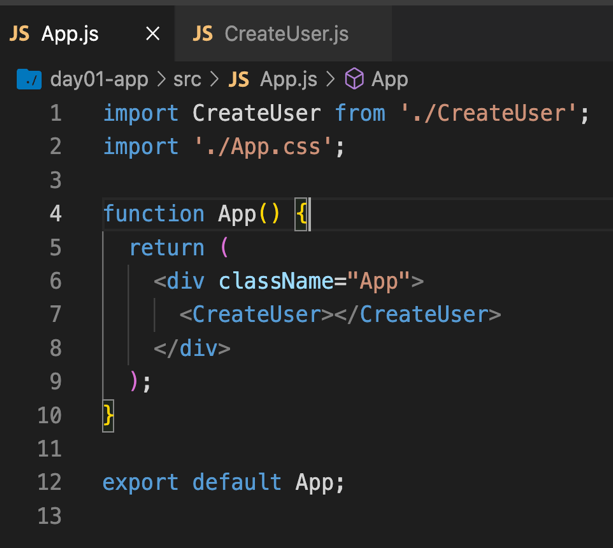
Let’s create our first component:

There are two different ways to implement component one is functional component (which is most popular) other one is class component. We’ll use functional component for learning and development.

Create a js file in src folder name CreateUser.js and the below code:



Now, import our newly created component to App.js as in below



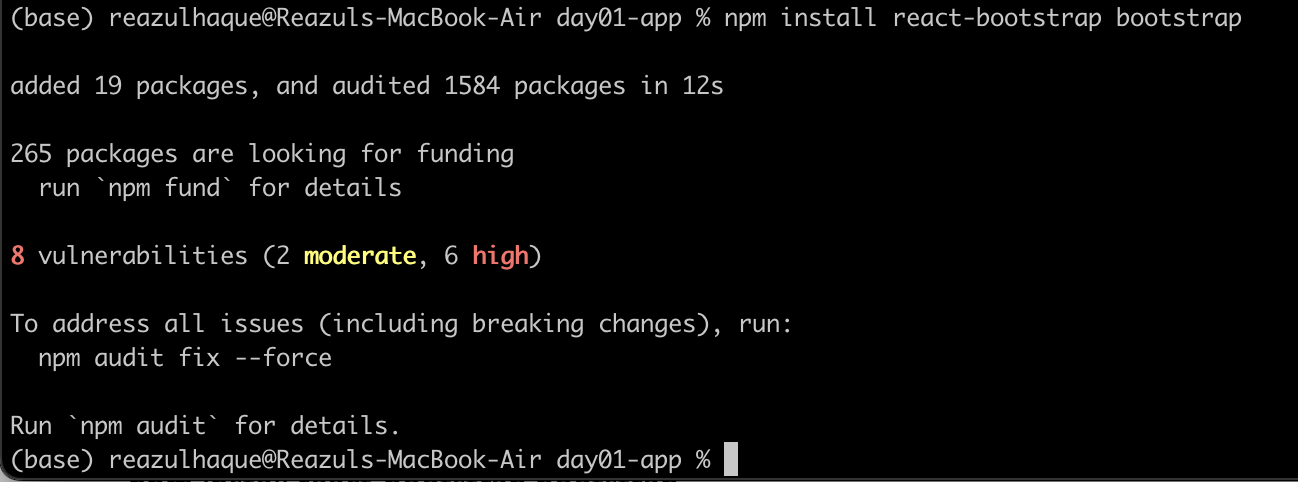
After update the App.js, we’ll get the view as below



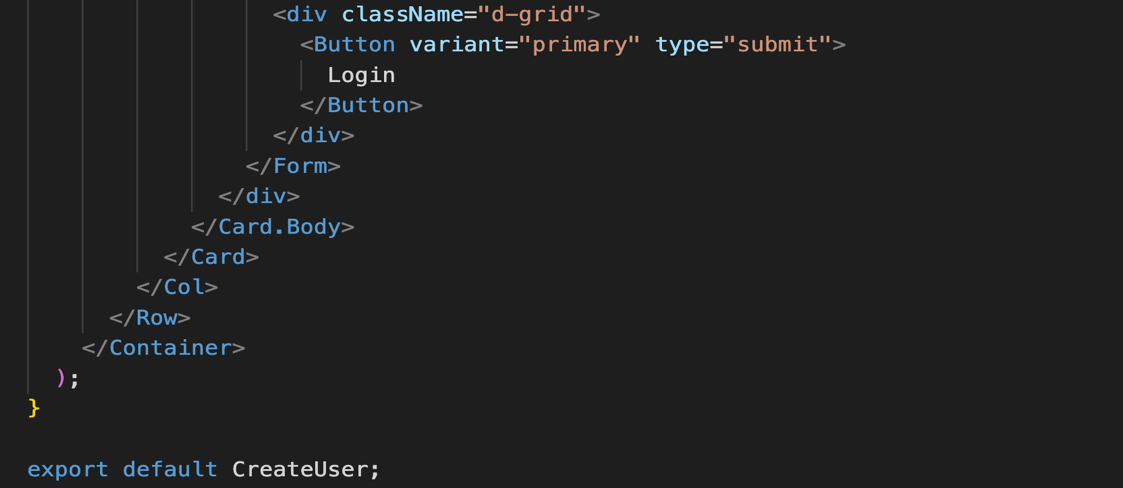
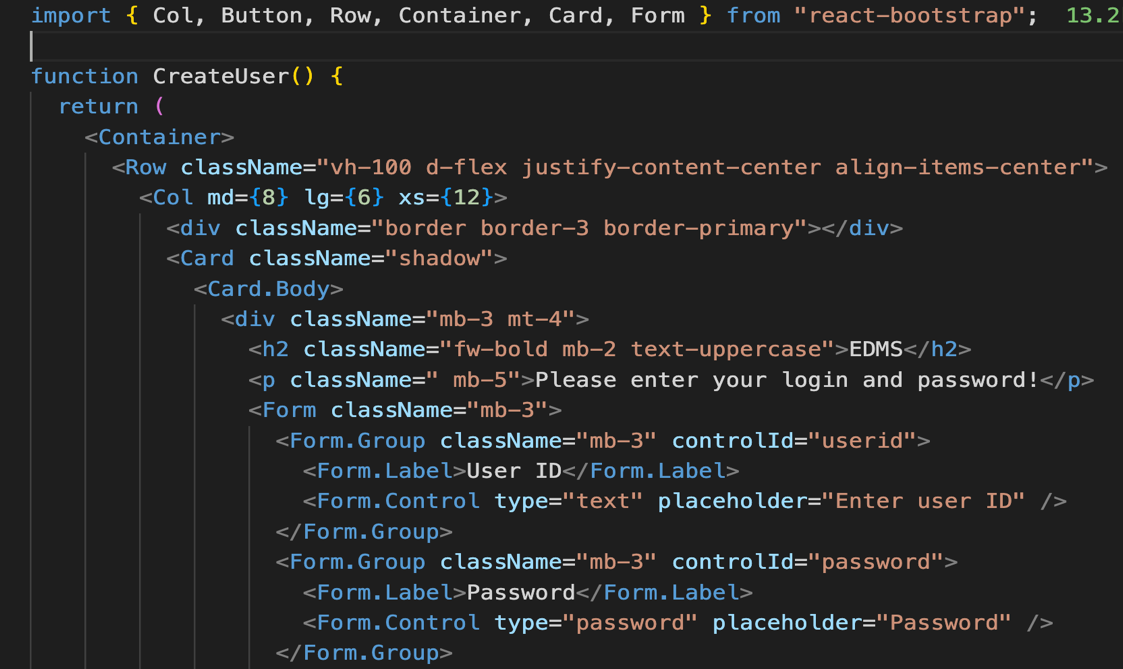
Oops… it’s not a professional look at all, so it’s time to add mighty bootstrap to our project using below command

npm install react-bootstrap bootstrap

We’ll get a view as in below after bootstrap installation



Now, we’ll add bootstrap to our CreateUser component. After adding bootstrap, our CreateUser component will be look like in below

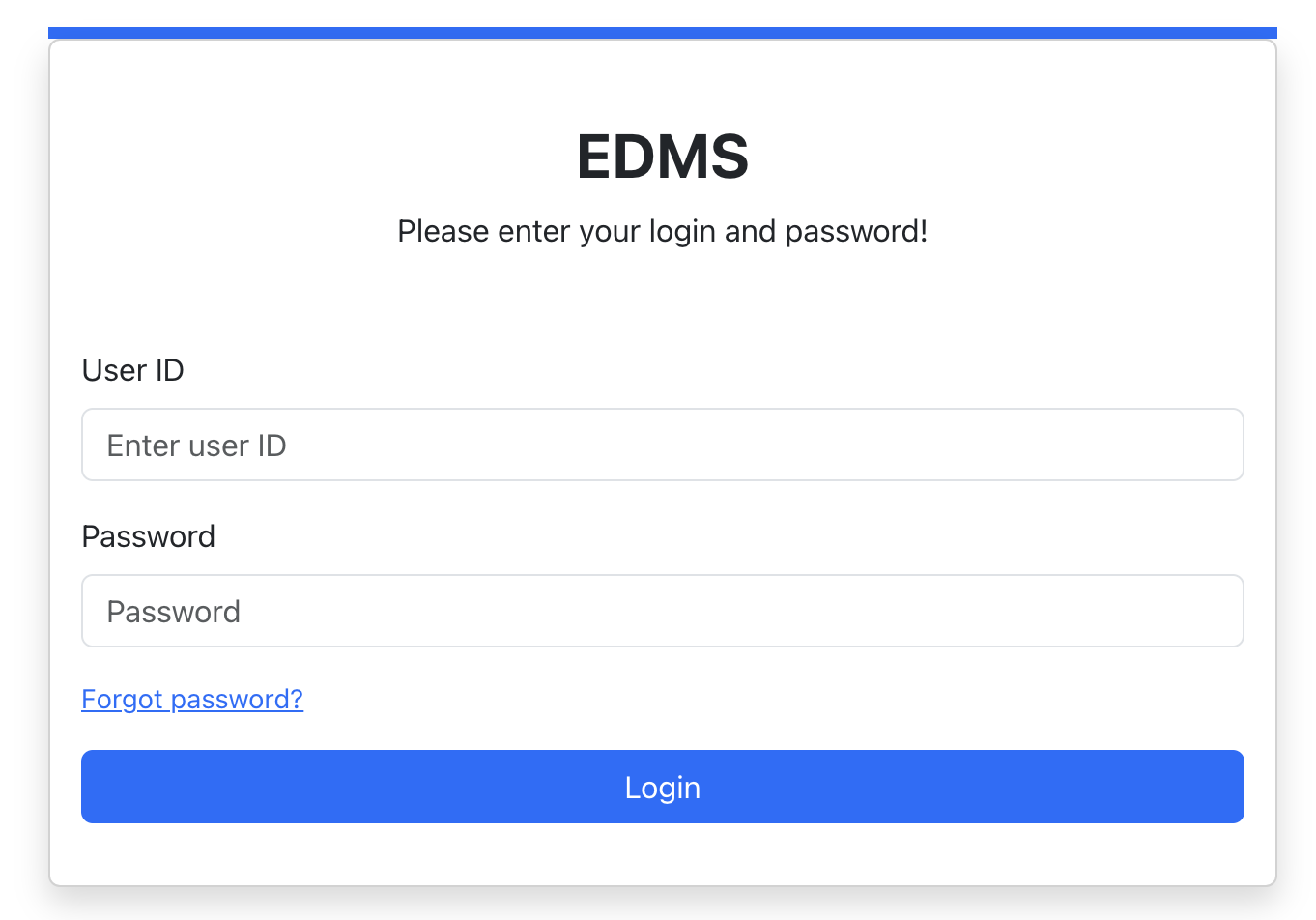


Also need to add bootstrap css to index.js file



Adding bootstrap css

By adding bootstrap our application looks like below



Cool…. You can checkout react-bootstrap official documentation for some awesome design

<https://react-bootstrap.netlify.app/>

Now, we’ll try to add react hook useState. useState is react hook that allows you to add state to a functional component. It returns an array with two values, one is the current state and other is a function to update it. The hook takes an initial state as an argument and returns an updated state value whenever the setter function is called.

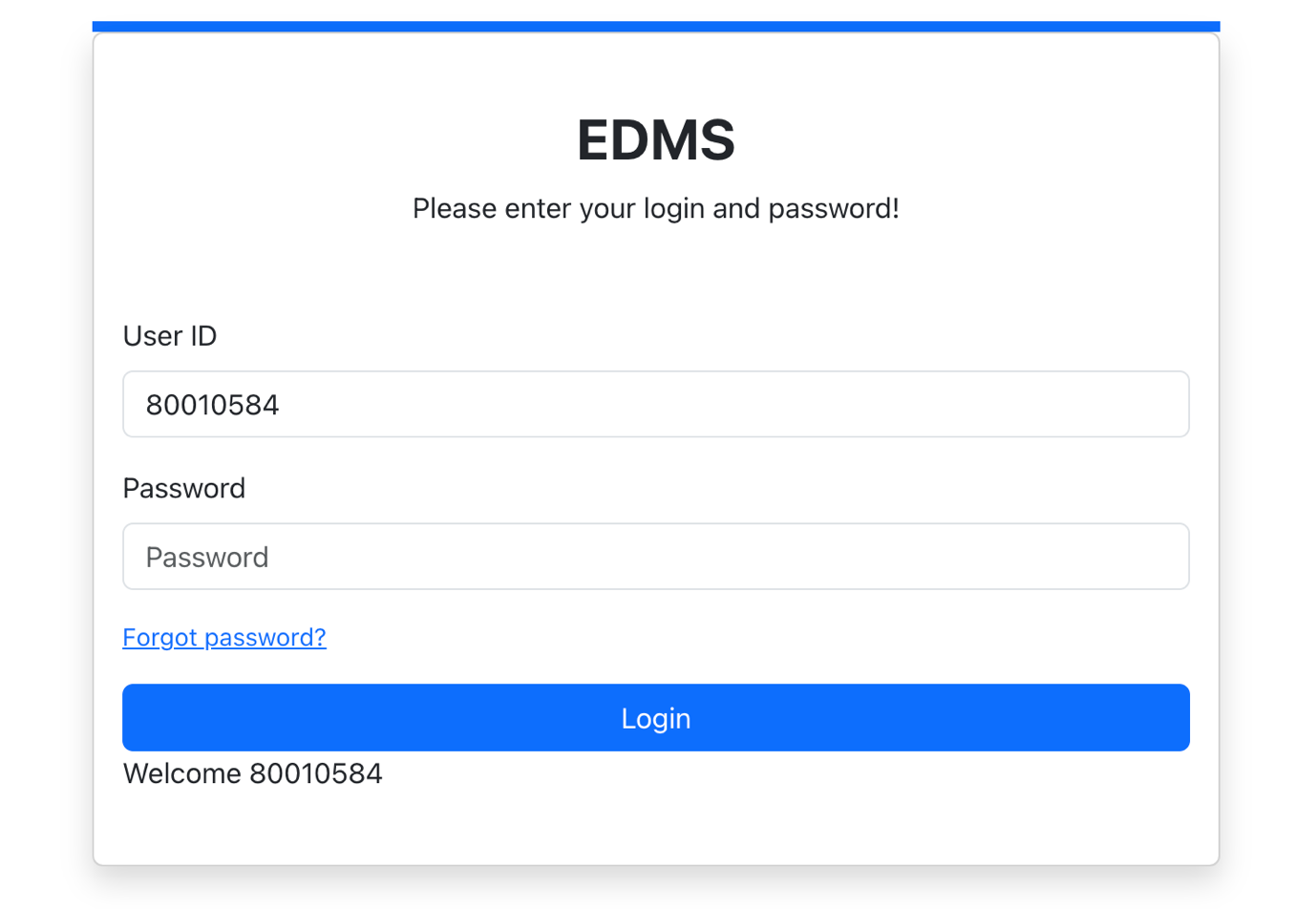
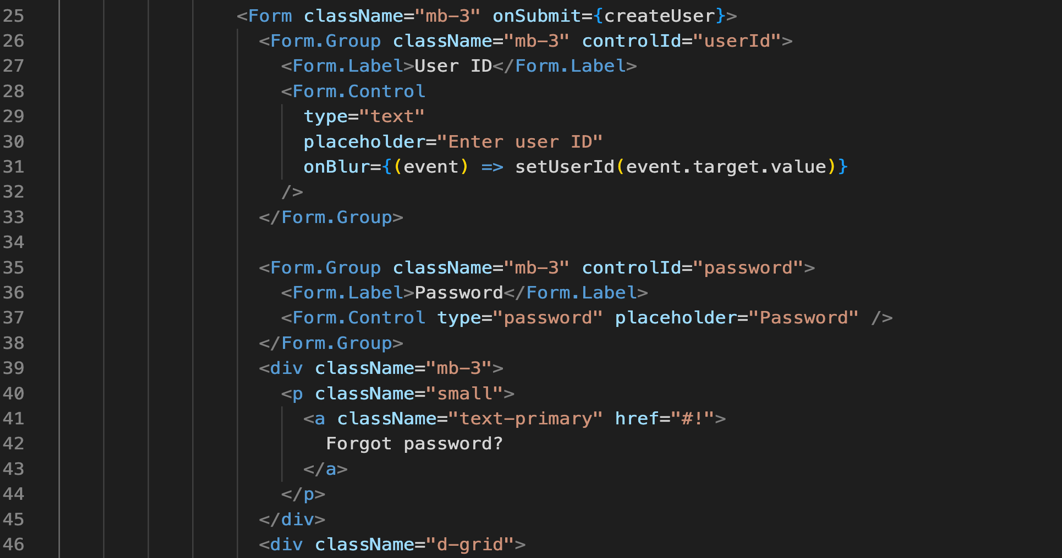
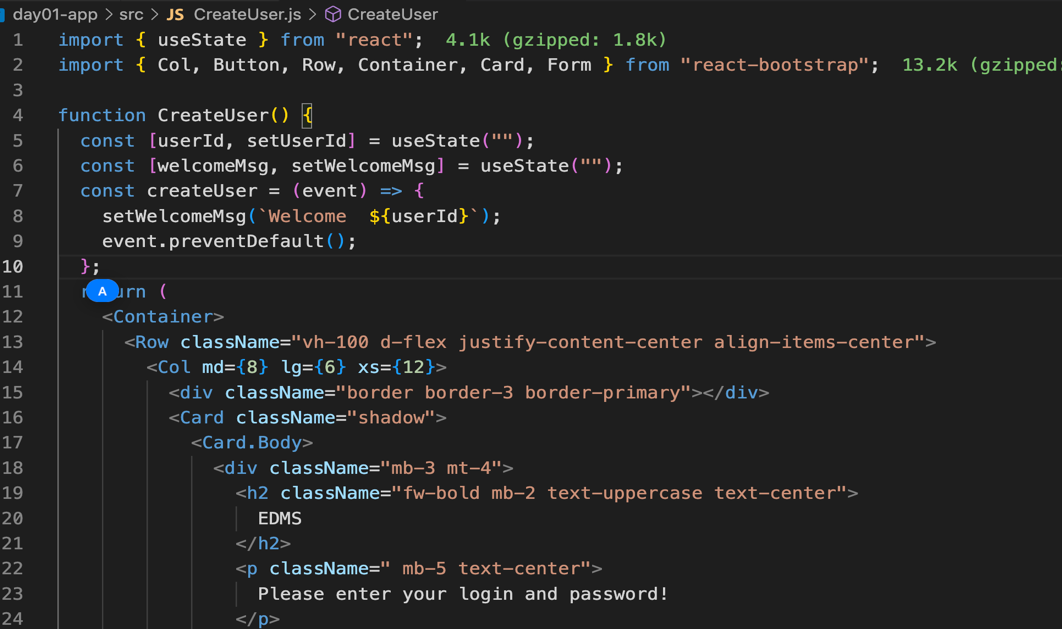
Example:

const [userId, setUserId] = useState("");

const [welcomeMsg, setWelcomeMsg] = useState("");

In react, useState can store any type of value, whereas the state in a class component is limited to being an object. This includes primitive data types like string, number and Boolean as well as complex data types such as array, object and function. It can even cover custom data types like class instances. Basically, anything that can be stored in a JavaScript variable can be stored in a state managed by useState.

Updated CreateUser.js File:



================ Day 02 End ================

Sharing Data Between Two Components using props (Properties)

Let’s create a new component name Message.js

function Message(props){

return(

<label>{props.msg}</label>

);

}

export default Message;

Now it’s time to update our previous CreateUser component

import { useState } from "react";

import { Col, Button, Row, Container, Card, Form } from "react-bootstrap";

import Message from "./Message";

function CreateUser() {

const [userId, setUserId] = useState("");

const [welcomeMsg, setWelcomeMsg] = useState("");

const [civility, setCivility] = useState("Mr");

const createUser = (event) => {

setWelcomeMsg(`Welcome ${civility} ${userId}`);

event.preventDefault();

};

const chooseCivility = (event) => {

setCivility(event.target.value);

}

return (

<Container>

<Row className="vh-100 d-flex justify-content-center align-items-center">

<Col md={8} lg={6} xs={12}>

<div className="border border-3 border-primary"></div>

<Card className="shadow">

<Card.Body>

<div className="mb-3 mt-4">

<h2 className="fw-bold mb-2 text-uppercase text-center">

EDMS

</h2>

<p className=" mb-5 text-center">

Please enter your login and password!

</p>

<Form className="mb-3" onSubmit={createUser}>

<Form.Group className="mb-3" controlId="userId">

<Form.Label>User ID</Form.Label>

<Form.Control

type="text"

placeholder="Enter user ID"

onBlur={(event) => setUserId(event.target.value)}

/>

</Form.Group>

<Form.Group className="mb-3" controlId="password">

<Form.Label>Password</Form.Label>

<Form.Control type="password" placeholder="Password" />

</Form.Group>

<Form.Group className="mb-3" controlId="civility">

<Form.Label>Civility</Form.Label>

<Form.Select aria-label="Default select" onChange={chooseCivility} value={civility}>

<option>Select Civility</option>

<option value="Mr">Mr</option>

<option value="M.">M.</option>

<option value="Mrs">Mrs.</option>

</Form.Select>

</Form.Group>

<div className="mb-3">

<p className="small">

<a className="text-primary" href="#!">

Forgot password?

</a>

</p>

</div>

<div className="d-grid">

<Button variant="primary" type="submit">

Login

</Button>

</div>

{/\* <Form.Label>{welcomeMsg}</Form.Label> \*/}

<Message msg={welcomeMsg}></Message>

</Form>

</div>

</Card.Body>

</Card>

</Col>

</Row>

</Container>

);

}

export default CreateUser;

Destructuring props:

For this one create an new component called Title.js

function Title(props){

console.log(props);

return(

<span>{props.teamName} {props.appName}</span>

);

}

export default Title;

and the CreateUser

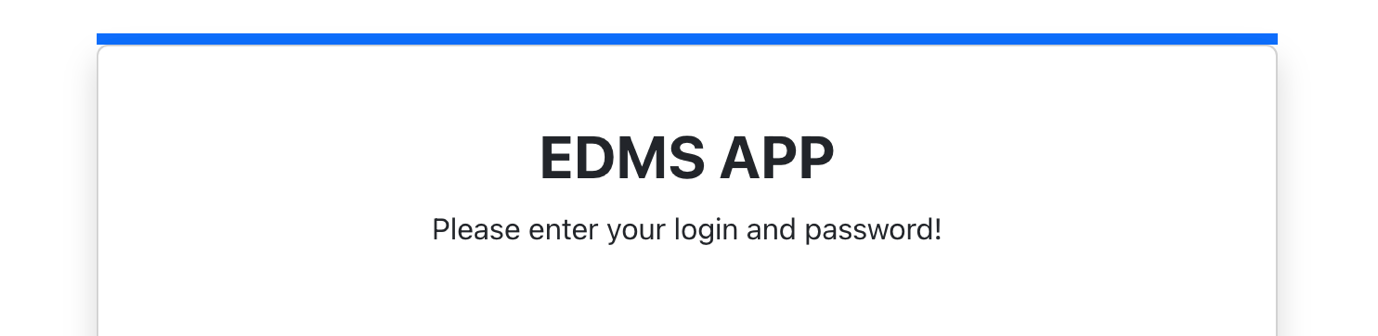
……

<h2 className="fw-bold mb-2 text-uppercase text-center">

<Title teamName="EDMS" appName="App"/>

</h2>

……



Title component destructured code

function Title(props){

console.log(props);

const { teamName, appName } = props;

return(

<span>{teamName} {appName}</span>

);

}

export default Title;

Test the Immutability of props:

Today we’ll look at multiple useState. Let’s checkout the below code

Form.js component

import { useState } from "react";

export default function Form(){

const [name, setName] = useState({firstName: "", lastName: ""});

return(

<div>

{name.firstName} - {name.lastName}

<form>

<input onChange={(e) => setName({...name, firstName: e.target.value})} type="text" value={name.firstName}/>

<input onChange={(e) => setName({...name, lastName: e.target.value})} type="text" value={name.lastName}/>

</form>

</div>

);

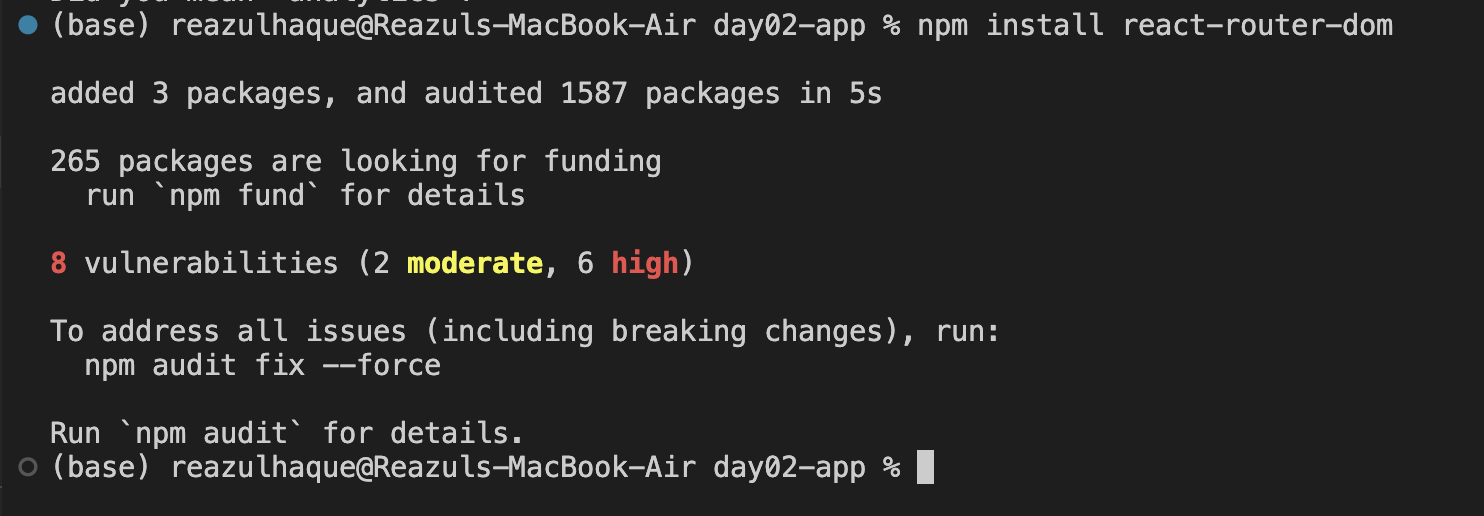
}

Next, we’ll look at most important par of Reactjs, which is react-router. By using react router we’ll be able to navigate in different pages. There are three different types of router

1. Memory router (mostly use for SPA’s)
2. Browser router (for web applications which have multiple pages)
3. Hash router (use for static web page and also for SPA’s)

To use react router we nned to install router by executing the below command

npm install react-router-dom



Let’s update index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { BrowserRouter, HashRouter, MemoryRouter } from 'react-router-dom';

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

root.render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

{/\* <MemoryRouter>

<App />

</MemoryRouter> \*/}

{/\* <HashRouter>

<App />

</HashRouter> \*/}

</React.StrictMode>

);

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

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

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

reportWebVitals();

Also update the App.js

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

import CreateUser from './CreateUser';

import Form from './Form';

import { Link, Route, Routes } from 'react-router-dom';

import NotFound from './NotFound';

function App() {

return (

<div className="App">

<nav>

<Link to ="/"> Home </Link> ||

<Link to ="/cart"> Cart </Link>

</nav>

<Routes>

{/\* <Route path ={["/"]} element= {<Form />}/> \*/}

{['/', '/home'].map(path => <Route path={path} element={<Form />} />)}

<Route path ="/cart" element= {<CreateUser />}/>

<Route path ="\*" element= {<NotFound />}/>

</Routes>

</div>

);

}

export default App;

====== Day 05 ======

Let’s try another reach hook called useEffect

By default, this function runs after every render of the component.

We can optionaly provide a dependency array as the second argument.

The useEffect will only run again if any of the values in the dependency array change.

Syntax useEffect(<FUNCTION>, <DEPENDENCY>)

FUNCTION: contains the code to be executed when useEffect triggers.

DEPENDENCY: it an optional parameter, useEffect triggers when the given dependency value changed.

useEffect on every render, without dependency value

useEffect(() =>{

//Code

});

useEffect only once on the first render with dependency array

useEffect(()=>{

// Code

},[]);

Next, now we’ll try to consume REST API’s using axios. In order to use axios we nned to install axios in our project. Run the below command to install axios

npm install axios

Now, create a client.js file and the below code

import axios from 'axios';

const Client = axios.create({

baseURL: 'http://dummy.restapiexample.com/api/v1', // base URL of the API

headers: {

'Content-Type': 'application/json',

},

});

export default Client;

Use the create() method of the axios object to create an Axios instance. The create() method takes in an object as an argument. The object contains the configuration for the Axios instance. The configuration object can contain the following properties. The baseURL property is the base URL of the API. This is the URL that will be used for all requests. The headers property is an object that contains the headers for all requests. The headers object can contain any number of headers. The headers object is optional. You can also set the headers for each request.

Now create a new component (GetData.js) to load and show all data also add that component to App.js Route

import { useEffect, useState } from "react";

import Client from "./client";

export default function GetData() {

const [products, setProducts] = useState([]);

useEffect(() => {

Client.get("/products").then((response) => {

setProducts(response.data.products);

});

});

return (

<div className="app">

<h2>All Products</h2>

{products.map((product) => {

return (

<div className="post-card" key={product.id}>

<h2 className="post-title">{product.title}</h2>

<p className="post-body">{product.description}</p>

</div>

);

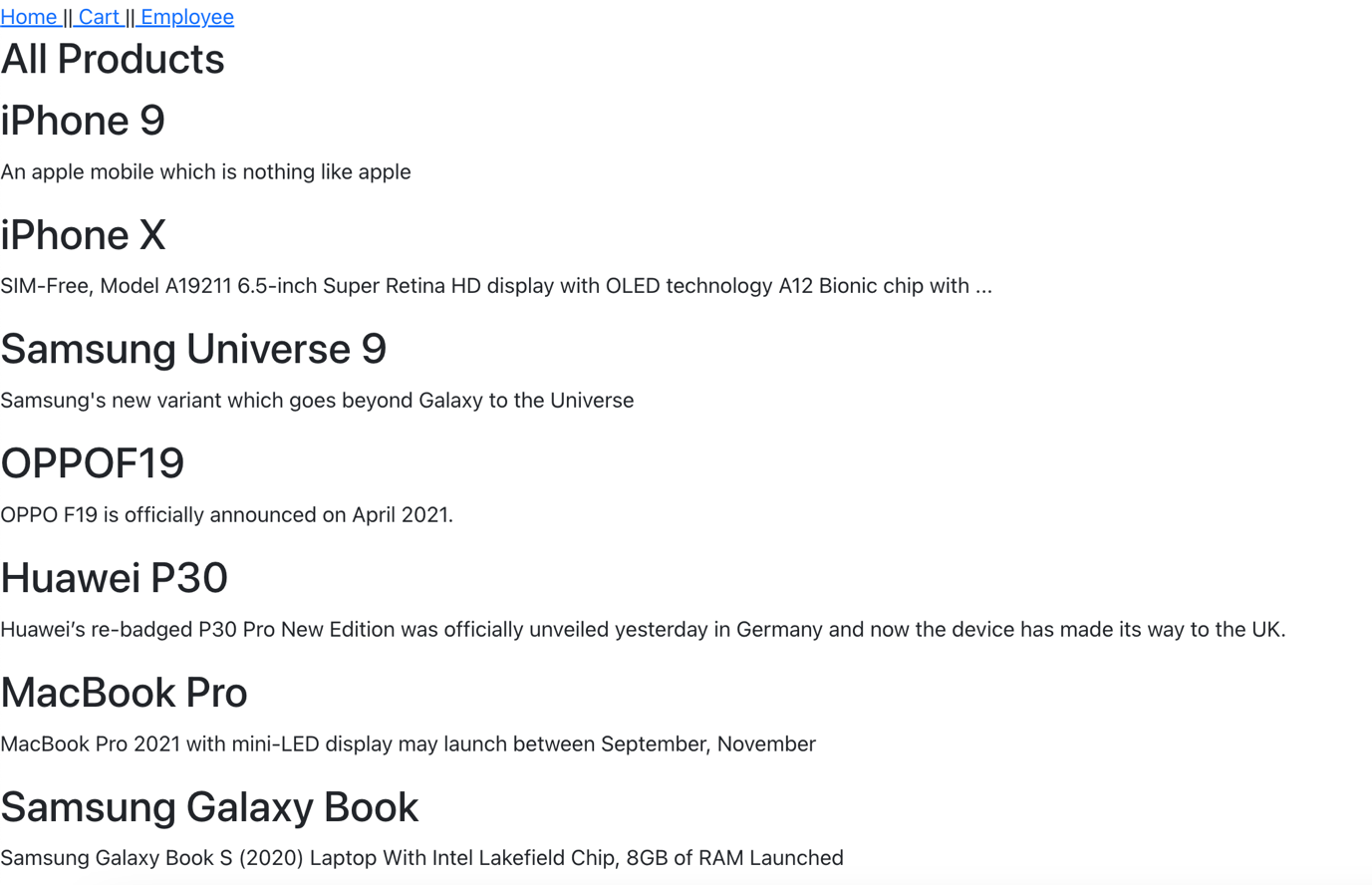
})}

</div>

);

}

In Browser, output will be look like below image



====== Day 06 ======

Now, we’ll try to implement authentication in our application. In order to add authentication, we need AuthProvider and AuthContext. Create a AuthProvider.js and below code

import { createContext, useState } from "react";

const AuthContext = createContext({});

export const AuthProvider = ({children}) => {

const [auth, setAuth] = useState({});

return(

<AuthContext.Provider value={{auth, setAuth}}>

{children}

</AuthContext.Provider>

);

}

export default AuthContext;

Let’s try to understand the code.

The AuthContext is generally used for managing states that will be needed across an application. For example, we need our user data or token return by response in the dashboard component as well as some other parts of our application.

The AuthProvider component is designed to wrap the application and provide the authentication context to its child components using the AUthCntext.Provider.

Now, wrap our application using AuthProvider

Update the index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

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

import App from './App';

import reportWebVitals from './reportWebVitals';

import { AuthProvider } from './context/AuthProvider';

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

root.render(

<React.StrictMode>

<AuthProvider>

<App />

</AuthProvider>

</React.StrictMode>

);

reportWebVitals();

Let’s check out interesting hook call useRef

1. Does not cause re-render
2. Dom element access
3. Track state changes

const [inputValue, setInputValue] = useState("");

const previousInputValue = useRef("");

useEffect(() => {

previousInputValue.current = inputValue;

}, [inputValue]);

return (

<>

<input

type="text"

value={inputValue}

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

/>

<h2>Current Value: {inputValue}</h2>

<h2>Previous Value: {previousInputValue.current}</h2>

</>

);

Let’s run this code

Our final Login.js

import { useContext, useEffect, useRef, useState } from "react";

import AuthContext from './context/AuthProvider';

import axios from "./api/axios";

const LOGIN\_URL = '/auth/authenticate';

export default function Login(){

const {setAuth} = useContext(AuthContext);

const userRef = useRef();

const errRef = useRef();

const[email, setEmail] = useState('');

const[password, setPassword] = useState('');

const[errMsg, setErrMsg] = useState('');

const[success, setSuccess] = useState(false);

useEffect(() => {

userRef.current.focus();

}, []);

useEffect(() => {

setErrMsg('');

},[email, password]);

const handleSubmit = async (e) => {

e.preventDefault();

try{

const response = await axios.post(

LOGIN\_URL,

JSON.stringify({email: email, password: password}),

{

headers: {'Content-Type': 'application/json'},

withCredentials: true

}

);

console.log(JSON.stringify(response.data));

const accessToken = response.data.access\_token;

const roles = response.data.role;

setAuth(email, password, roles, accessToken);

setEmail('');

setPassword('');

setSuccess(true);

}catch(error){

if(!error.response){

setErrMsg('No Server Response');

}else if(error.response.status === 400){

setErrMsg('Missing Email or Password');

}else if(error.response.status === 401){

setErrMsg('Unauthorized');

}else{

setErrMsg('Login Failed');

}

errRef.current.focus();

}

}

return(

<>

{success ? (

<section>

<h1>You are Logedin</h1>

<br />

<p>

<a href="#"> Go to Home</a>

</p>

</section>

) : (

<section>

<p ref={errRef} className={errMsg ? "errMsg" : "offscreen"} aria-live="assertive">{errMsg}</p>

<h1>Login</h1>

<form onSubmit={handleSubmit}>

<label htmlFor="email">Email:</label>

<input type="text" id="email"

ref={userRef}

autoComplete="off"

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

value={email}

required

/>

<label htmlFor="password">Password:</label>

<input type="password" id="password"

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

value={password}

required

/>

<button>Login</button>

</form>

</section>

)}

</>

);

}

Axios instance

import axios from "axios";

export default axios.create({

baseURL: 'http://localhost:8080/api/v1'

});