**B1)** **What is JSX?**

* JSX stands for JavaScript XML.
* JSX allows us to write HTML in React.
* JSX makes it easier to write and add HTML in React.

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement()  and/or appendChild() methods.

**B2) What is React.createClass?**

* React. createClass **allows you to generate component "classes."** Under the hood, your component class is using a bespoke class system implemented by React. With ES6, React allows you to implement component classes that use ES6 JavaScript classes.

**B3) How React works? How Virtual-DOM works in React?**

* Virtual DOM: **React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM)**. So for every object that exists in the original DOM, there is an object for that in React Virtual DOM.

**B4) What is ReactDOM and what is the difference between ReactDOM and React?**

* **The react package holds the react source for components, state, props and all the code that is react.** **The react-dom package as the name implies is the glue between React and the DOM**. Often, you will only use it for one single thing: mounting your application to the index.

**B5) What are the differences between a class component and functional component?**

* A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX). A class component requires you to extend from React. Component and create a render function which returns a React element. There is no render method used in functional components.

**B6) What is the difference between state and props?**

* Props are used to pass data from one component to another. The state is a local data storage that is local to the component only and cannot be passed to other components.

**B7) What are controlled components?**

* In React, Controlled Components are **those in which form's data is handled by the component's state**. It takes its current value through props and makes changes through callbacks like onClick, onChange, etc.

**B8) What is a higher order component?**

* A higher-order component (HOC) is **an advanced technique in React for reusing component logic**. HOCs are not part of the React API, per se. They are a pattern that emerges from React's compositional nature. Concretely, a higher-order component is a function that takes a component and returns a new component.

**B9) What is create-react-app?**

Create React App is **a comfortable environment for learning React, and is the best way to start building a new single-page application in React**. It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production.

**I1) Compare and contrast creating React Components in ES5 and ES2015 (also known as ES6). What are the advantages and disadvantages of using one or the other? Include notes about default. props, initial state, PropTypes, and DisplayName.**

* The differences between ES5 and ES6 syntax when writing a React application.

1. require vs import.
2. Exports vs export.
3. React components.
4. React props.
5. React state.
6. React methods.
7. React context.

**I2) Compare and contrast the various React Component lifecycle methods. How might understanding these help build certain interfaces/features?**

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases. The three phases are: **Mounting, Updating, and Unmounting**.

**Mounting**

Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

The render() method is required and will always be called, the others are optional and will be called if you define them.

## **Updating**

The next phase in the lifecycle is when a component is *updated*.

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

The render() method is required and will always be called, the others are optional and will be called if you define them.

## **Unmounting**

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

* componentWillUnmount()

**I3) Compare and contrast incorporating mixins and enforcing modularity in React Components. ( extend , createClass and mixins, HOCs) Why would you use these techniques, and what are the drawbacks of each?**

At Facebook, React usage has grown from a few components to thousands of them. This gives us a window into how people use React. Thanks to declarative rendering and top-down data flow, many teams were able to fix a bunch of bugs while shipping new features as they adopted React.

However it’s inevitable that some of our code using React gradually became incomprehensible. Occasionally, the React team would see groups of components in different projects that people were afraid to touch. These components were too easy to break accidentally, were confusing to new developers, and eventually became just as confusing to the people who wrote them in the first place. Much of this confusion was caused by mixins. At the time, I wasn’t working at Facebook but I came to the [same conclusions](https://medium.com/@dan_abramov/mixins-are-dead-long-live-higher-order-components-94a0d2f9e750) after writing my fair share of terrible mixins.

This doesn’t mean that mixins themselves are bad. People successfully employ them in different languages and paradigms, including some functional languages. At Facebook, we extensively use traits in Hack which are fairly similar to mixins. Nevertheless, we think that mixins are unnecessary and problematic in React codebases. Here’s why:

**Mixins introduce implicit dependencies**

Sometimes a component relies on a certain method defined in the mixin, such as getClassName(). Sometimes it’s the other way around, and mixin calls a method like renderHeader() on the component. JavaScript is a dynamic language so it’s hard to enforce or document these dependencies.

### **Mixins cause name clashes**

There is no guarantee that two particular mixins can be used together. For example, if FluxListenerMixin defines handleChange() and WindowSizeMixin defines handleChange(), you can’t use them together. You also can’t define a method with this name on your own component.

### **Mixins cause snowballing complexity**

Even when mixins start out simple, they tend to become complex over time. The example below is based on a real scenario I’ve seen play out in a codebase.

A component needs some state to track mouse hover. To keep this logic reusable, you might extract handleMouseEnter(), handleMouseLeave() and isHovering() into a HoverMixin. Next, somebody needs to implement a tooltip. They don’t want to duplicate the logic in HoverMixin so they create a TooltipMixin that uses HoverMixin. TooltipMixin reads isHovering() provided by HoverMixin in its componentDidUpdate() and either shows or hides the tooltip.

## **Migrating from Mixins**

Let’s make it clear that mixins are not technically deprecated. If you use React.createClass(), you may keep using them. We only say that they didn’t work well for us, and so we won’t recommend using them in the future.

Every section below corresponds to a mixin usage pattern that we found in the Facebook codebase. For each of them, we describe the problem and a solution that we think works better than mixins. The examples are written in ES5 but once you don’t need mixins, you can switch to ES6 classes if you’d like.

**I4) What is the significance of keys in React?**

* Keys **help React identify which items have changed, are added, or are removed**. Keys should be given to the elements inside the array to give the elements a stable identity: const numbers = [1, 2, 3, 4, 5]; const listItems = numbers.

**I5) What is the significance of refs in React?**

Refs **provide a way to access DOM nodes or React elements created in the render method**. In the typical React dataflow, props are the only way that parent components interact with their children. To modify a child, you re-render it with new props.

**I6) Compare and contrast significance of each, and when would you use one or the other? In addition, how might you use any data layer with React (like Backbone, Ember, or Redux)? What are the systemic requirements of doing so?**

* Redux, Ember, Backbone- all of these JavaScript frameworks are used for implementing client side scripting.
* all use the [Model–view–controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) pattern. This is something that JQuery lacks.
* Also, all of these are [Single-page application](https://en.wikipedia.org/wiki/Single-page_application) serving single most important purpose of making HTML dynamic.
* **EmberJS** is highly opinionated. Which means, you have to do everything 'the Ember way'. As a result, its learning curve is longer.
* **BackboneJS,** unlike Ember, is least opinionated(very versatile). You can get your web-apps up and running really quickly, as its learning curve is really small. Another benefit predominantly associated with BackboneJS is its small size-its very lightweight- about 6Kbs when minified and compressed for production.
* **Redux** is an **Open Source Library** which provides a central store, and actions to modify the store. It can be used with any project using **JavaScript** or **TypeScript**, but since we are comparing it to **Context API**, so we will stick to **React-based Applications**.

**I7) What are pure functional Components?**

A function is said to be pure **if the return value is determined by its input values only and the return value is always the same for the same input values**. A React component is said to be pure if it renders the same output for the same state and props.

**I8) How might React handle or restrict Props to certain types, or require certain Props to exist?**

**React allows us to pass data from parent component to child components with the help of props**. If we want to check the type of props we can do so with the help of PropTypes. Proptypes helps us to perform runtime type checks on the props and if the check fails we get a warning on the browser's console.

*PropTypes* define the type of a *prop*. So each time, a value is passed through a prop, it gets validated for it’s type. If you pass a value through a *prop* with a different data type than it is specified in the *PropTypes*, an error message will be printed in the console of your browser:

[caption id=”attachment\_233" align=”alignnone” width=”919"]



The error message you get, when you pass a value with a wrong data type[/caption].

**I9)** **Create menu using array ?**

const values = ['A', 'B', 'C'];

// ...

return (

<select>

{values.map(val => <option value={val}>{val}</option>)}

</select>

);

**I10) Using Events is React js ?**

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.

Handling events with react have some syntactic differences from handling events on DOM. These are:

1. React events are named as **camelCase** instead of **lowercase**.
2. With JSX, a function is passed as the **event handler** instead of a **string**. For example:

**Event declaration in React:**

1. <button onClick={showMessage}>
2. Hello JavaTpoint
3. </button>

**A1) The TextInput component renders an input element in the DOM and accepts a ref that is forwarded to that input element. Finish the FocusableInput component:**

**• The component should accept a focused prop.**

**• When the focused prop is changed from false to true, and the input is not focused, it should receive the focus.**

**• If on mounting the focused prop is true, the input should receive the focus.**

import React from "react";

import ReactDOM from "react-dom";

*/\*The TextInput component renders an input element in the DOM*

*and accepts a ref that is forwarded to that input element.*

*Finish the FocusableInput component:*

*The component should accept a focused prop.*

*When the focused prop is changed from false to true,*

*and the input is not focused, it should receive the focus.*

*If on mounting the focused prop is true,*

*the input should receive the focus.\*/*

class Input extends React.PureComponent {

render() {

let { forwardedRef, ...otherProps } = this.props;

return <input {...otherProps} ref={forwardedRef} />;

}

}

const TextInput = React.forwardRef((props, ref) => {

return <Input {...props} forwardedRef={ref} />;

});

class FocusableInput extends React.Component {

ref = React.createRef();

render() {

return <TextInput ref={this.ref} />;

}

*// When the focused prop is changed from false to true,*

*// and the input is not focused, it should receive focus.*

*// If focused prop is true, the input should receive the focus.*

*// Implement your solution below in cDU and cDM:*

componentDidUpdate(prevProps) {

if (!prevProps.focused && this.props.focused) this.ref.current.focus();

}

componentDidMount() {

this.props.focused && this.ref.current.focus();

}

}

FocusableInput.defaultProps = {

focused: false

};

const App = props => <FocusableInput focused={props.focused} />;

document.body.innerHTML = "<div id='root'></div>";

const rootElement = document.getElementById("root");

ReactDOM.render(<App />, rootElement);

**A2) Write a TodoList component that expects an items prop, and a list of objects, each with text and done properties.**

* **TodoList also accepts an onItemClick function prop, which should be called when a user clicks an item in the list, if the item is not marked as done. Otherwise, the onItemClick should not be called and the click event itself should not be propagated further. The function should be called the item object from the items list as the first parameter and the event as the second parameter.**

import React, { useState } from 'react';

*//mock data*

import data from "./data.json";

*//components*

import Header from "./Header";

import ToDoList from "./ToDoList";

import ToDoForm from './ToDoForm';

function App() {

const [ toDoList, setToDoList ] = useState(data);

const handleToggle = (id) => {

let mapped = toDoList.map(task => {

return task.id === Number(id) ? { ...task, complete: !task.complete } : { ...task};

});

setToDoList(mapped);

}

const handleFilter = () => {

let filtered = toDoList.filter(task => {

return !task.complete;

});

setToDoList(filtered);

}

const addTask = (userInput ) => {

let copy = [...toDoList];

copy = [...copy, { id: toDoList.length + 1, task: userInput, complete: false }];

setToDoList(copy);

}

return (

<div className="App">

<Header />

<ToDoList toDoList={toDoList} handleToggle={handleToggle} handleFilter={handleFilter}/>

<ToDoForm addTask={addTask}/>

</div>

);

}

export default App;

**A3)** **Create a Web page and set Routing on Each web pages in React Js.**

we are going to assume that you have created a React application and have multiple components that you want to navigate between.

1. **Add React Router**

npm install react-router-dom

This will install the react-router-dom package to your application which is necessary to implement a dynamic routing for the individual pages, synchronize the URLs, and access the history API.

**2. Build Your Pages**

We're going to make a collection of pages, in the /src/pages directory. In this example, we have a Homepage, 404 page, blog and contact page to navigate between.

In this example, the home, contact and blog page follow the same general structure:

import React from 'react';

function Home() {

return (

<h1>this is the homepage</h1>

);

}

export default Home;

NoPage is also the same, but is designed as a 404 error page.

We'll come back to "layout.jsx" in step 4.

**3. Make The Navbar**

Now that we understand the structure of the pages, let's make a navbar to jump between each. We suggest making an individual navbar component as it's easy to edit in the future. Some other tutorials lump the Nav into the layout.jsx file, which is also fine.

Here's what navbar.jsx looks like:

import React from "react";

import { BrowserRouter, Route, Link } from "react-router-dom";

function Navbar() {

return (

<nav>

<ul>

<li>

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

</li>

<li>

<Link to="/blogs">Blogs</Link>

</li>

<li>

<Link to="/contact">Contact</Link>

</li>

</ul>

</nav>

);

}

export default Navbar;

The <link/> element is the same as an <a href=> in HTML.

**4. Bring Everything Together With Layout.jsx**

Now, we'll tie the page content (think of it as the main) and the navigation between each page together.

Here's what that file looks like:

import React from "react";

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

import Navbar from "../Navbar";

const Layout = () => {

return (

<>

<Navbar />

<Outlet />

</>

);

};

export default Layout;

The <Outlet/> element spits out the component routed to in step 5.

**5. Create Routes**

In our index.js, we'll create the actual routing.

import ReactDOM from "react-dom";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Layout from "./pages/Layout";

import Home from "./pages/Home";

import Blogs from "./pages/Blogs";

import Contact from "./pages/Contact";

import NoPage from "./pages/NoPage";

export default function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Layout />}>

<Route index element={<Home />} />

<Route path="blogs" element={<Blogs />} />

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

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

</Route>

</Routes>

</BrowserRouter>

);

}

ReactDOM.render(<App />, document.getElementById("root"));

This is the actual way we route between each component, or page. The structure is important here: by default, we'll render the Layout template, which in turn outputs the pages within. <Route index element={<Home />} />, By setting the index route, the naked domain will load the Home template.

These routes will show the blog page when the url is /blog and contact page when the url is /contact. Add as many pages and routes as needed.

<Route path="blogs" element={<Blogs />} />

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

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

This final route is a catchall, if no other page exists. It's good for a 404.

**A4) Create a signup Form in react Js and Set Form Validation.**

import React, { useState } from 'react'

import { useNavigate } from 'react-router-dom'

function Signup() {

    const[formvalue,setformvalue] = useState({

        id:new Date().getTime().toString(),

        name:"",

        email:"",

        password:"",

        mobile:""

    });

    const redirect=useNavigate();

    function ChangeHandle(e){

        setformvalue({...formvalue,[e.target.name]:e.target.value,id:new Date().getTime().toString()});

        console.log(formvalue);

    }

    function submitHandle(e){

        e.preventDefault();

        fetch('https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyBcOY9FqPYhmJgibwpIWXKSHwAGPdL91x0', {

            method: 'POST',

            body: JSON.stringify(formvalue),

            headers: {

                'Content-type': 'application/json; charset=UTF-8'

            },

            })

            .then((response) => response.json())

            .then((data) => {

                if(data.localId){

                    fetch('https://react-admin-f0817-default-rtdb.firebaseio.com/customer.json', {

                    method: 'POST',

                    body: JSON.stringify(formvalue),

                    headers: {

                        'Content-type': 'application/json; charset=UTF-8',

                    },

                    })

                    .then((response) => response.json())

                    .then((data) => {

                       console.log(data);

                        console.log("Data added successfully");

                        redirect("/");

                        setformvalue({name:"",email:"",password:"",mobile:""})

                    });

                }

             });

     }

  return (

    <div>

        <div id="">

            <div className="main-page signup-page">

                <h2 className="title1">Signup</h2>

                <div className="sign-up-row widget-shadow">

                    <form method="post">

                            <div className="sign-u">

                                <label>Name</label>

                                <input type="text" className="form-control"  id="name" name="name" onChange={ChangeHandle} value={formvalue.name} />

                                <div className="clearfix"> </div>

                            </div>

                            <div className="sign-u">

                                <label>Username</label>

                                <input type="text" className="form-control" id="email" name="email" onChange={ChangeHandle} value={formvalue.email} />

                                <div className="clearfix"> </div>

                            </div>

                            <div className="sign-u">

                                <label>Password</label>

                                <input type="password" className="form-control" id="password" name="password" onChange={ChangeHandle} value={formvalue.password} />

                                <div className="clearfix"> </div>

                            </div>

                            <div className="sign-u">

                                <label>Mobile</label>

                                <input type="tel" className="form-control" id="mobile" name="mobile" onChange={ChangeHandle} value={formvalue.mobile} />

                                <div className="clearfix"> </div>

                            </div>

                            <br/    >

                            <button type="submit" className="btn btn-default" onClick={submitHandle}>Submit</button>

                            <div className="registration">

                                Already Registered.

                                <a href='/'>Login</a>

                            </div>

                    </form>

                </div>

            </div>

        </div>

    </div>

  )

}

export default Signup

**A5) Create a Contact Form in React Js and Set Form validation.**

import React from 'react';

import { Form, Button } from 'semantic-ui-react';

import { useForm } from "react-hook-form";

export default function FormValidation() {

const { register, handleSubmit, formState: { errors } } = useForm();

const onSubmit = (data) => {

console.log(data);

}

return (

<div>

<Form onSubmit={handleSubmit(onSubmit)}>

<Form.Field>

<label>First Name</label>

<input

placeholder='First Name'

type="text"

{...register("firstName", { required: true, maxLength: 10 })}

/>

</Form.Field>

{errors.firstName && <p>Please check the First Name</p>}

<Form.Field>

<label>Last Name</label>

<input

placeholder='Last Name'

type="text"

{...register("lastName", { required: true, maxLength: 10 })}

/>

</Form.Field>

{errors.lastName && <p>Please check the Last Name</p>}

<Form.Field>

<label>Email</label>

<input

placeholder='Email'

type="email"

{...register("email",

{

required: true,

pattern: /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/

})}

/>

</Form.Field>

{errors.email && <p>Please check the Email</p>}

<Form.Field>

<label>Password</label>

<input

placeholder='Password'

type="password"

{...register("password", {

required: true,

pattern: /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{6,15}$/

})}

/>

</Form.Field>

{errors.password && <p>Please check the Password</p>}

<Button type='submit'>Submit</Button>

</Form>

</div>

)

}

**A6) Create a Navigation and Apply the Routing In web page in React Js.**

* + *Refer to answer A3)*

**A7) Create a Register Page in React Js and Provide Form validation in all Field.**

import React, {useState,setState} from 'react';

function RegistrationForm() {

const [firstName, setFirstName] = useState(null);

const [lastName, setLastName] = useState(null);

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

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

const [confirmPassword,setConfirmPassword] = useState(null);

const handleInputChange = (e) => {

const {id , value} = e.target;

if(id === "firstName"){

setFirstName(value);

}

if(id === "lastName"){

setLastName(value);

}

if(id === "email"){

setEmail(value);

}

if(id === "password"){

setPassword(value);

}

if(id === "confirmPassword"){

setConfirmPassword(value);

}

}

const handleSubmit = () => {

console.log(firstName,lastName,email,password,confirmPassword);

}

return(

<div className="form">

<div className="form-body">

<div className="username">

<label className="form\_\_label" for="firstName">First Name </label>

<input className="form\_\_input" type="text" value={firstName} onChange = {(e) => handleInputChange(e)} id="firstName" placeholder="First Name"/>

</div>

<div className="lastname">

<label className="form\_\_label" for="lastName">Last Name </label>

<input type="text" name="" id="lastName" value={lastName} className="form\_\_input" onChange = {(e) => handleInputChange(e)} placeholder="LastName"/>

</div>

<div className="email">

<label className="form\_\_label" for="email">Email </label>

<input type="email" id="email" className="form\_\_input" value={email} onChange = {(e) => handleInputChange(e)} placeholder="Email"/>

</div>

<div className="password">

<label className="form\_\_label" for="password">Password </label>

<input className="form\_\_input" type="password" id="password" value={password} onChange = {(e) => handleInputChange(e)} placeholder="Password"/>

</div>

<div className="confirm-password">

<label className="form\_\_label" for="confirmPassword">Confirm Password </label>

<input className="form\_\_input" type="password" id="confirmPassword" value={confirmPassword} onChange = {(e) => handleInputChange(e)} placeholder="Confirm Password"/>

</div>

</div>

<div class="footer">

<button onClick={()=>handleSubmit()} type="submit" class="btn">Register</button>

</div>

</div>

)

}

export default RegistrationForm

**A8) Create a Login Form in React Js and Provide Fom Validation.**

import React, { useState }  from 'react'

import { Link, useNavigate } from 'react-router-dom'

function Login() {

    const redirect=useNavigate();

    const[formvalue,setformvalue]=useState({

        email:"",

        password:"",

        returnSecureToken:""

    })

    function ChangeHandle(e){

        setformvalue({...formvalue,[e.target.name]:e.target.value,returnSecureToken:true});

        console.log(formvalue);

    }

    function submitHandle(e){

        e.preventDefault();

        fetch('https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyBcOY9FqPYhmJgibwpIWXKSHwAGPdL91x0', {

            method: 'POST',

            body: JSON.stringify(formvalue),

            headers: {

                'Content-type': 'application/json; charset=UTF-8',

            },

            })

            .then((response) => response.json())

            .then((data) => {

                //console.log(data);

                if(data.registered)

                {

                  //  alert('Login Successfull');

                    localStorage.setItem('localId', data.localId);

                    localStorage.setItem('email', data.email);

                    localStorage.setItem('idToken', data.idToken);

                    redirect('/dashboard');

                }

                else{

                    alert('Login Failed due to Wrong Creadential');

                }

            });

    }

    return (

        <div>

            <div id="">

                <div className="main-page login-page">

                    <h2 className="title1">Login</h2>

                    <div className="widget-shadow">

                        <div className="login-body">

                            <form action="" method="post">

                                <input type="email" className="user" name="email" value={formvalue.email} onChange={ChangeHandle}  placeholder="Enter Your Email" required />

                                <input type="password" name="password"  value={formvalue.password} onChange={ChangeHandle} className="lock" placeholder="Password" required />

                                <div className="forgot-grid">

                                    <label className="checkbox"><input type="checkbox" name="checkbox" defaultChecked /><i />Remember me</label>

                                    <div className="forgot">

                                        <a href="#">forgot password?</a>

                                    </div>

                                    <div className="clearfix"> </div>

                                </div>

                                <input type="submit" onClick={submitHandle}  name="Sign In" defaultValue="Sign In" />

                                <div className="registration">

                                    Don't have an account ?

                                    <a href="/signup">

                                        Create an account

                                    </a>

                                </div>

                            </form>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    )

}

export default Login

**A9) Create a Table In React Js and Insert data in Table.**

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

import Footer from '../Component/Footer'

import NavBar from '../Component/NavBar'

function Add\_Product() {

    const[formvalue,setformvalue]=useState({

        id:new Date().getTime().toString(),

        productName:"",

        productDesc:"",

        productImage:""

    })

    const[alldata,setalldata]=useState([]);

    function ChangeHandle(e){

        setformvalue({...formvalue,[e.target.name]:e.target.value, id:new Date().getTime().toString()});

        console.log(formvalue);

    }

    function submitHandle(e){

        e.preventDefault();

        fetch('https://react-admin-f0817-default-rtdb.firebaseio.com/product.json', {

            method: 'POST',

            body: JSON.stringify(formvalue),

            headers: {

                'Content-type': 'application/json; charset=UTF-8',

            },

            })

            .then((response) => response.json())

            .then((data) => {

                setformvalue({productName:"",productDesc:"",productImage:""})

                alert("Data added successfully");

            });

    }

    function deletedata(deleteid){

        const afterdelete = alldata.filter((item)=> item.id!= deleteid);

        setalldata(afterdelete);

    }

    useEffect(()=>{

        fetchData();

    },[]);

    function fetchData(){

    fetch('https://react-admin-f0817-default-rtdb.firebaseio.com/product.json')

        .then((response) => response.json())

        .then((data) =>{

            setalldata(data);

        });

    }

  return (

    <>

            <NavBar />

            <div id="page-wrapper">

                <div className="main-page">

                    <div className="panel-body widget-shadow">

                        <form method="post">

                            <div className="form-group">

                                <label>Product Name</label>

                                <input type="text" className="form-control"  id="name" name="productName" onChange={ChangeHandle} value={formvalue.productName} />

                            </div>

                            <div className="form-group">

                                <label>Description</label>

                                <textarea className="form-control" id="description" name="productDesc" onChange={ChangeHandle} value={formvalue.productDesc} />

                            </div>

                            <div className="form-group">

                                <label>Product Image</label>

                                <input type="url" className="form-control" id="productimage" name="productImage" onChange={ChangeHandle} value={formvalue.productImage} />

                            </div>

                            <button type="submit" className="btn btn-default" onClick={submitHandle}>Submit</button>

                        </form>

                    </div>

                </div>

                <div className="table-responsive bs-example widget-shadow">

                    <h4 style={{"padding":"20px"}}>Database Table:</h4>

                        <table className="table table-bordered">

                            <thead>

                                <tr>

                                    <th>Product Id</th>

                                    <th>Product Name</th>

                                    <th>Description</th>

                                    <th>Image</th>

                                    <th>Action</th>

                                </tr>

                            </thead>

                            <tbody>

                                {

                                Object.keys(alldata).map((item,index)=>{

                                    const{productName,productDesc,productImage,id} =alldata[item];

                                    return(

                                        <tr key={index}>

                                            <id>{id}</id>

                                            <td>{productName}</td>

                                            <td style={{"width":"30%"}}>{productDesc}</td>

                                            <td><img src={productImage} width="70px" alt={productName}/></td>

                                            <td>

                                                <button className="btn btn-danger" onClick={()=>deletedata(item)}>Delete</button>

                                            </td>

                                        </tr>

                                        )

                                    })

                                }

                            </tbody>

                        </table>

                    </div>

            </div>

            <Footer />

        </>

  )

}

export default Add\_Product