https://shorturl.at/3lCHe 14 Jun 2024

Imports and Exports
Lifecycle Methods
Higher Order Components
Pure Components
Unit Testing
React With Redux
Multiple Reducers
Use of Thunk Middleware
Deploying React Application on Firebase

==========

Imports and exports in ReactJS

==========

- As react developer we can export variables, objects, functions, etc
- As react developer we import any exported variables, objects, functions, etc

Note:-

1. While exporting there must be ONE member as default.
2. While exporting
   export {member1, member2, member3,...}
   where member1 is default.
3. If we write the default keyword, then it can be imported with any name.

Directory structure

<>

src

ios

- variables.js
- functions.js
- myComponent.js

variables.js

```
let url = `http://localhost:8080`
let flag = true
let score = 88
let db_config = {
    host: "localhost",
    user: "root",
    password: "root",
    database: "nodedb",
    table: "products"
}

export default url;
export { flag, score, db_config }
```

functions.js

```
function fun_one() {
    return `Welcome to My function`
}

function auth(arg1, arg2) {
    if (arg1 === 'admin' && arg2 === 'admin')
        return true
    else
        return false
}

export default fun_one
export { auth }
```

myComponent.js

```
import React from 'react'
//import url, { flag, score,db_config } from './variables'
import * as obj from './variables'
import my_fun, { auth } from './functions'
export default class MyComponent extends React.Component {
    render() {
        return (
            <div>
                {/*<p style={{ color: 'rgb(255,0,0)' }}>Url:-  {url} </p>
                <p style={{ color: 'rgb(200,0,0)' }}>Flag:-
{JSON.stringify(flag)} </p>
                <p style={{ color: 'rgb(180,0,0)' }}>Score :- {score}</p>
                <p style={{ color: 'rgb(160,0,0)' }}>Database Configuration :-
{JSON.stringify(db_config)}</p>*/}
                <p style={{ color: 'rgb(0,255,0)' }}>Url:-  {obj.default} </p>
                <p style={{ color: 'rgb(0,200,0)' }}>Flag:-
{JSON.stringify(obj.flag)} </p>
                <p style={{ color: 'rgb(0,180,0)' }}>Score :- {obj.score}</p>
                <p style={{ color: 'rgb(0,160,0)' }}>Database Configuration :-
{JSON.stringify(obj.db_config)}</p>
                <p style={{ color: 'rgb(0,0,255)' }}>{my_fun()} </p>
                <form onSubmit={this.login}>
                    <input type='text' placeholder='Enter Username'
name='uname'></input>
                    <br /><br />
                    <input type='password' placeholder='Enter Password'
name='upwd'></input>
                    <br /><br />
                    <input type='submit' value='Login'></input>
                </form>
            </div>
        )
    }
    login = (e) => {
        let uname = e.target.uname.value
        let upwd = e.target.upwd.value
        let login = auth(uname, upwd)
        if (login)
```

```
            alert('Authentication Success')
        else
            alert('Authentication Failed')
    }
}
```

==========

Lifecycle Methods

==========

Parent.js

```
import React from 'react'
import Child from './Child'
export default class Parent extends React.Component {
    constructor() {
        super()
        console.log('Parent Constructor')
        this.state = {
            technology: 'ReactJS'
        }
        /*
           - constructor gets called at loading of component
           - constructor will execute only once.
           - it is recommanded to define state in constructor.
       */
    }
    componentWillMount() {
        console.log('Parent componentWillMount')
        /*
          - This method executes after constructor
          - This method executes only once.
          - It is recommanded to change initial state in this method.
          - It is recommanded to set global parameters here
      */
        if (window.innerWidth < 600) {
            this.setState({
                width: 'Small Page'
            })
        }
    }
    render() {
        console.log('Parent render')
        /*
           - render is mandatory lifecycle method
           - it will execute after componentWillMount method.
           - we place presentation logic here
           - it will always gets called when state change
       */
        return (
            <div>
                <h1>Parent Component</h1>
                <p style={{ color: 'blue' }}>{this.state.technology} </p>
                <p style={{ color: 'red' }}>{this.state.width} </p>
                <button onClick={() => this.setState({ technology: 'MERN'
})}>Change</button>
                <Child key1={this.state.technology}></Child>
```

```
            </div>
        )
    }
    componentDidMount() {
        console.log('Parent componentDidMount')
    }
    /*
    --- Execution Flow ---
    - Parent Constructor
    - Parent componentWillMount
    - Parent render
    - Child Constructor
    - Child componentWillMount
    - Child render
    - if state / props change detected
    - Parent render
    - Child reder
    - Child componentDidMount
    - Parent componentDidMount
    */
    componentWillReceiveProps() {
        /*
            - this method will execute if component receives props
        */
        console.log("Parent componentWillReceiveProps called")
    }
    shouldComponentUpdate() //control state change
    {
        /*
            - this method controls the state change
            - return true   -> change the state
            - return false  -> dont change the state
        */
        console.log("Parent shouldComponentUpdate")
        return true
    }
    componentWillUpdate() {
        console.log("Parent componentWillUpdate")
    }
    componentDidUpdate() {
        console.log("Parent componentDidUpdate")
    }
    /*
       - Before unmounting the components react library will execute the
following method
       - this method is used to perform cleanup operations
       - eg    nullifying instances, empty states, empty props, cancel
subscriptions, etc
    */
    componentWillUnmount() {
        console.log("Parent componentWillUnmount")
    }
}
```

Child.js

```
import React from "react"
export default class Child extends React.Component {
    constructor() {
        super()
        console.log('Child constructor')
    }
    componentWillMount() {
        console.log('Child componentWillMount called')
    }
    render() {
        console.log('Child render')
        return (
            <div>
                <h1>Child Component</h1>
                <h4>{this.props.key1} </h4>
            </div>
        )
    }
    componentDidMount() {
        console.log('Child componentDidMount')
    }
    componentWillReceiveProps() {
        console.log("Child componentWillReceiveProps called")
    }
    shouldComponentUpdate() {
        console.log("Child shouldComponentUpdate")
        return true
    }
    componentWillUpdate() {
        console.log("Child componentWillUpdate")
    }
    componentDidUpdate() {
        console.log("Child componentDidUpdate")
    }
    componentWillUnmount() {
        console.log("Child componentWillUnmount")
    }
}
```

==========

Higher Order Components

==========

- Create Cloths component with functionality to calculate total + 18% GST
- Create same component Food with 10% GST

Directory Structure

<>

src

HOCeg

- Cloths.js
- Food.js
- myComponent.js

Cloths.js

```
import React from 'react'
class Cloths extends React.Component {
    constructor() {
        super()
        this.state = {
            final: 0
        }
    }
    render() {
        return (
            <div>
                <h1 style={{ color: 'blue' }}>Welcome to Cloths
Department</h1>
                <form className="w-25 mx-auto" onSubmit={this.calculate}>
                    <div className="form-group">
                        <lable>Quantity</lable>
                        <input className="form-control" type='number'
placeholder="Quantity" name="qty"></input>
                    </div>
                    <div className="form-group">
                        <lable>Rate</lable>
                        <input className="form-control" type='number'
placeholder="Rate" name="rate"></input>
                    </div>
                    <input type='submit' value='Calculate' className="btn
btn-primary"></input>
                </form>
                <h3 style={{ color: 'navy' }}>Total amount:-
{this.state.final}</h3>
            </div>
        )
    }
    calculate = (e) => {
        let gst = 18
        let total = e.target.qty.value * e.target.rate.value
        let final = total + total * gst / 100
        this.setState({ final: final })
    }
}

export default Cloths
```

Similarly create Food.js

myComponent.js

```
import React from 'react'
import Cloths from './Cloths'
import Food from './Food'
class MyComponent extends React.Component {
    render() {
        return (
            <div>
```

```
                <Cloths />
                <Food />
            </div>
        )
    }
}
export default MyComponent
```

Here the problem is that the code is duplicated.

Def. Higher Order Components is an advanced technique in ReactJS for reusing component functionality.

In this technique we pass component as argument and it returns a component.

Eg
const EnhancedComponent = HigherOrderComponent(WrappedComponent)

Create component GSTCalc this will be HOC

GSTCalc.js

```
import React from "react";
const GSTCalc = (WrappedComponent, dept, gstRate) => {
    class GSTCalc extends React.Component {
        constructor() {
            super()
            this.state = {
                final: 0
            }
        }
        render() {
            return (
                <div>
                    <h1 style={{ color: 'blue' }}>Welcome to {dept}
Department</h1>
                    <form className="w-25 mx-auto" onSubmit={this.calculate}>
                        <div className="form-group">
                            <lable>Quantity</lable>
                            <input className="form-control" type='number'
placeholder="Quantity" name="qty"></input>
                        </div>
                        <div className="form-group">
                            <lable>Rate</lable>
                            <input className="form-control" type='number'
placeholder="Rate" name="rate"></input>
                        </div>
                        <input type='submit' value='Calculate' className="btn
btn-primary"></input>
                    </form>
                    <WrappedComponent final={this.state.final} />
                </div>
            )
```

```
        }
        calculate = (e) => {
            e.preventDefault()
            let gst = gstRate
            let total = e.target.qty.value * e.target.rate.value
            let final = total + total * gst / 100
            this.setState({ final: final })
        }
    }
    return GSTCalc
}
export default GSTCalc
```

update Cloths and Food components as

Cloths.js

```
import React from 'react'
import GSTCalc from './GSTCalc'
class Cloths extends React.Component {
    render() {
        return (
            <div>
                <h3>Total amount with GST :- {this.props.final}</h3>
            </div>
        )
    }
}
export default GSTCalc(Cloths, 'Cloths', 18)
```

==========

Pure components

==========

<>

src

PureComponents

- Parent.js
- Child.js
- Gchild.js

Note : - Pure component prevents re-rendering of components when state value remains same.

Parent.js

```
import React from 'react'
import Child from './Child'
//export default class Parent extends React.Component {
export default class Parent extends React.PureComponent {
    constructor() {
        super()
        this.state = {
            num: 100
```

```
        }
    }
    componentDidMount() {
        setInterval(() => {
            //this.setState({ num: Math.random() * 100 })
            this.setState({ num: 100 })
        }, 1000)
    }
    render() {
        console.log('Parent render')
        return (
            <div className='container mt-5'>
                <p>num : {this.state.num} </p>
                <button onClick={() => {
                    this.setState({ num: 102 })
                }}>Change</button>
                <Child key1={this.state.num} />
            </div>
        )
    }
}
```

Child.js

```
import React from "react";
import Gchild from "./Gchild";
//export default class Child extends React.Component {
    export default class Child extends React.PureComponent {
    render() {
        console.log('Child render')
        return (
            <div>
                <p>Child:- {this.props.key1} </p>
                <Gchild key2={this.props.key1} />
            </div>
        )
    }
}
```

Gchild.js

```
import React from "react";
//export default class Gchild extends React.Component {
    export default class Gchild extends React.PureComponent {
    render() {
        console.log('GChild render')
        return (
            <div>
                <p>GChild:- {this.props.key2} </p>
            </div>
        )
    }
}
```

==========

Unit Testing

==========
Testing:-
- Testing is categorised into
    i) Manual Testing
    ii)Automation Testing
- Now a days manual testing is deprecating
- Automation testing is categorised into
    i) Unit testing
    ii)Integration testing
    iii)End to End testing

Unit Testing:-
- Testing particular functionalities with dummy scenarios is called as unit testing.
- 'jest' is the javascript testing framework used to write unit tests.
- unit test cases files must have extension as '.test.js'
- it(-,-) / test(-,-) are predefined functions in jest framework.
- these functions are used to write test suits.
- describe(-,-) function subdivids test suits.
- expect() function is used to access results.
- unit test cases are executed using following command
    >yarn test

nomanip.js
```
let num = 0
const increment = () => {
    return num += 1
}
const decrement = () => {
    return num -= 1
}
export default increment
export { decrement }
```

Nomanip.test.js
```
import increment, { decrement } from "./nomanip"

describe("Increment function testing", () => {
    test("Increment function increments number by 1", () => {
        expect(increment()).toBe(1)
    })
})

describe("Decrement function testing", () => {
    test("Decrement function decrements number by 1", () => {
        expect(decrement()).toBe(-1)
    })
})
```

App.test.js
```
import { render, screen } from '@testing-library/react';
```

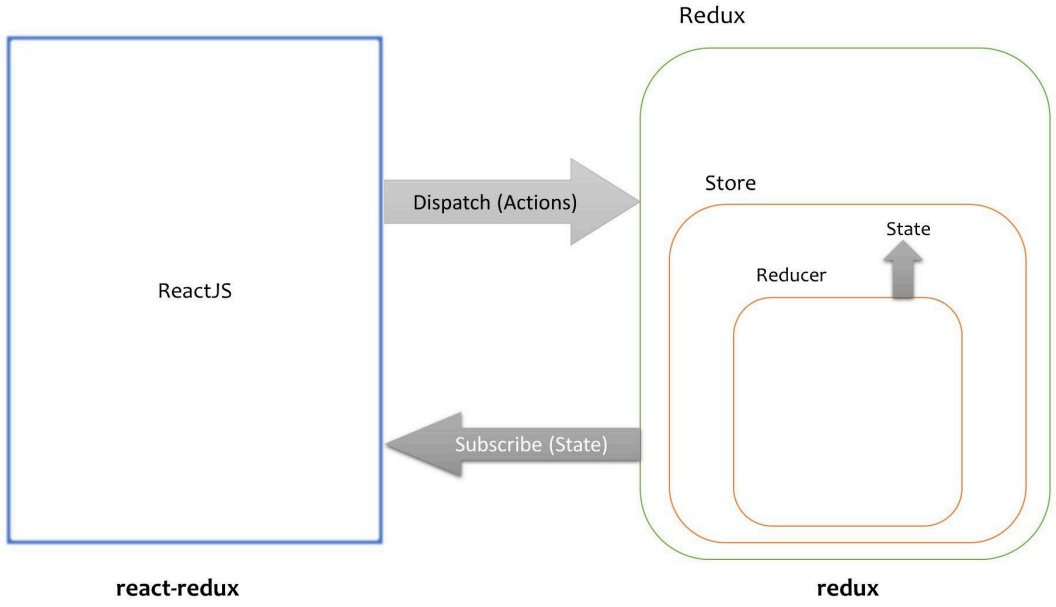
```
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});

test('Check "Edit" on the screen', () => {
  render(<App />);
  const linkElement = screen.getByText(/Edit/);
  expect(linkElement).toBeInTheDocument();
});
```

==========

Redux

==========

- Redux is used for global state management.



Redux Architecture

- Create a store using 'redux' library.
- global business logic written in reducer.
- Output of the reducer is state.
- integrate this architecture with any front-end technology, eg ReactJS.
- 'react-redux' library is used to integrate react with redux.
- a request sent by reactjs is called as dispatch.
- dispatch contain various actions.
- Eg FETCH,

- WITHDRAW,
- UPDATE,
- DELETE,
- DEPOSIT,
- ...
- response received by reactjs is as subscribe
- subscribe contains state, implies received response is state.

Download libraries
redux
react-redux
>yarn add redux react-redux --save

Directory structure
<>
src
reduxeg
reducer
- reducer.js
- myComponent.js
- index.js

create reducer

reducer.js
```
const initialState = {
    products: []
}

const reducer = (state = initialState, actions) => {
    switch (actions.type) {
        case 'PRODUCTS':
            return {
                ...state,
                products: [
                    { "p_id": 111, "p_name": "P_one", "p_cost": 10000 },
                    { "p_id": 222, "p_name": "P_two", "p_cost": 20000 },
                    { "p_id": 333, "p_name": "P_three", "p_cost": 30000 },
                    { "p_id": 444, "p_name": "P_four", "p_cost": 40000 },
                    { "p_id": 555, "p_name": "P_five", "p_cost": 50000 }
                ]
            }
    }
    return state
}
export default reducer
```

myComponent.js
```
import React from 'react'
import { connect } from "react-redux";
class MyComponent extends React.Component {
```

```
    render() {
        return (
            <div>
                <button onClick={this.props.getProducts}>Products</button>
                <br /><br />
                <h4>{JSON.stringify(this.props.products)} </h4>
            </div>
        )
    }
}

const receive = (state) => {
    return {
        products: state.products
    }
}

const send = (dispatch) => {
    return {
        getProducts: () => {
            dispatch({ type: 'PRODUCTS' })
        }
    }
}

export default connect(receive, send)(MyComponent)
```

index.js

```
//import reducer
import reducer from './06 reduxeg/reducer/reducer'
//import createStore
import { legacy_createStore as createStore } from 'redux';
//import Provider
import { Provider } from 'react-redux';
//create the store
const store = createStore(reducer)
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <MyComponent />
  </Provider>
);
```

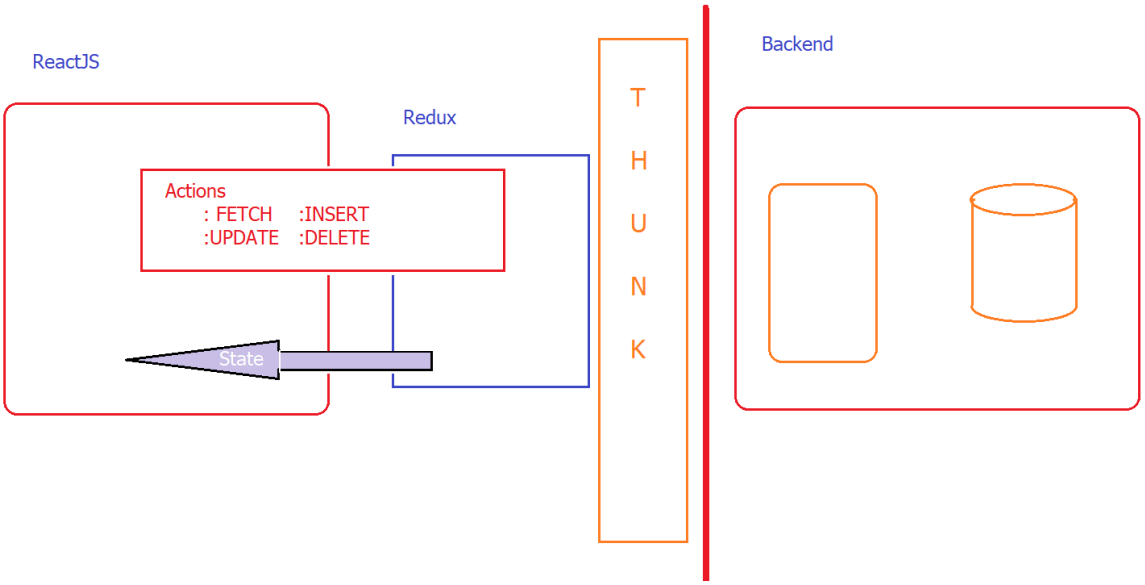
==========

Thunk Middleware
CRUD Application

==========

1. Create react application
   >create-react-app thunkapp

2. Switch to application
   >cd thunkapp



Terminologies
- Actions
  : Used to monitor following actions
    : FETCH
    : INSERT
    : UPDATE
    : DELETE
- reducer (Redux)
  : used to maintain global states.

3. Download following libraries
- For api calls -> axios
- For redux -> redux
- For react with redux -> react-redux
- Redux with thunk -> redux-thunk
- Bootstrap styling -> bootstrap, react-bootstrap

>yarn add axios redux react-redux bootstrap react-bootstrap redux-thunk --save

4. Create actions
<>
src
actions
- actions.js
- url.js

url.js
```
let url = `-- your url --`
```

```
export default url
```

actions.js

```
import axios from "axios"
import url from "../url"
const readAction = (records) => {
    return {
        type: 'FETCH', value: records
    }
}

export const getProducts = () => {
    return (dispatch) => {
        return axios.get(url + '/fetch')
            .then((posRes) => {
                dispatch(readAction(posRes.data))
            }, (errRes) => {
                console.log(errRes)
            })
    }
}
```

5. Create reducer

<>

src

reducer

- reducer.js

reducer.js

```
const intialState = {
    data: []
}

const reducer = (state = intialState, actions) => {
    switch (actions.type) {
        case 'FETCH':
            state.data = []
            return {
                ...state,
                data: state.data.concat(actions.value)
            }
    }
    return state
}
export default reducer
```

App.js

```
import React from 'react'
import * as actions from './actions/actions'
import { connect } from 'react-redux'
class App extends React.Component {
  componentDidMount() {
    this.props.getProducts()
  }
```

```
  render() {
    return (
      <div>
        data : {JSON.stringify(this.props.data)}
      </div>
    )
  }
}
const receive = (state) => {
  return {
    data: state.data
  }
}

const send = (dispatch) => {
  return {
    getProducts: () => {
      dispatch(actions.getProducts())
    }
  }
}

export default connect(receive, send)(App)
```

index.js

```
import { legacy_createStore as createStore } from 'redux';
import reducer from './reducer/reducer';
import { applyMiddleware } from 'redux';
import {thunk} from 'redux-thunk'
import { Provider } from 'react-redux';
const store = createStore(reducer, applyMiddleware(thunk))
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

//////////////////
TEST APPLICATION AT THIS STAGE
AFTER THAT PROCEED
//////////////////

actions.js

```
import axios from "axios"
import url from "../url"
const readAction = (records) => {
    return {
        type: 'FETCH', value: records
    }
}
```

```
export const getProducts = () => {
    return (dispatch) => {
        return axios.get(url + '/fetch')
            .then((posRes) => {
                dispatch(readAction(posRes.data))
            }, (errRes) => {
                console.log(errRes)
            })
    }
}

const insertAction = (result) => {
    return {
        type: 'INSERT', value: result
    }
}

export const insertProduct = (record) => {
    return (dispatch) => {
        return axios.post(url + '/insert', record)
            .then((posRes) => {
                dispatch(insertAction(posRes.data))
            }, (errRes) => {
                console.log(errRes)
            })
    }
}

const updateAction = (result) => {
    return {
        type: 'UPDATE', value: result
    }
}

export const updateProduct = (record) => {
    return (dispatch) => {
        return axios.post(url + '/update', record)
            .then((posRes) => {
                dispatch(updateAction(posRes.data))
            }, (errRes) => {
                console.log(errRes)
            })
    }
}

const deleteAction = (result) => {
    return {
        type: 'DELETE', value: result
    }
}

export const deleteProduct = (record) => {
    return (dispatch) => {
        return axios.post(url + '/delete', record)
            .then((posRes) => {
```

```
                dispatch(deleteAction(posRes.data))
            }, (errRes) => {
                console.log(errRes)
            })
    }
}
```

reducer.js

```
const intialState = {
    data: []
}

const reducer = (state = intialState, actions) => {
    switch (actions.type) {
        case 'FETCH':
            state.data = []
            return {
                ...state,
                data: state.data.concat(actions.value)
            }
        case 'INSERT':
        case 'UPDATE':
        case 'DELETE':
            return {
                ...state,
                state: actions.value
            }
    }
    return state
}
export default reducer
```

App.js

```
import React from 'react'
import * as actions from './actions/actions'
import { connect } from 'react-redux'
import { Modal, Table } from 'react-bootstrap'
import 'bootstrap/dist/css/bootstrap.css'
let arr = []
class App extends React.Component {
    constructor() {
        super()
        this.state = {
            loading: false,
            status: false,
            insertPopup: false,
            updatePopup: false
        }
    }
    showPopup = (msg) => {
        if (msg === `addRec`) {
            this.setState({
                status: true,
```

```
            insertPopup: true,
            updatePopup: false
        })
    }
    else {
        this.setState({
            status: true,
            insertPopup: false,
            updatePopup: true
        })
    }
}
closePopup = () => {
    this.setState({
        status: false
    })
}
componentDidMount() {
    if (arr != [])
        this.setState({
            loading: true
        })
    else
        this.setState({
            loading: false
        })
    this.props.getProducts()
}
save = (e) => {
    e.preventDefault()
    if (this.state.insertPopup)
        this.insert(e)
    else if (this.state.updatePopup)
        this.update(e)
    this.closePopup()
}
insert = (e) => {
    let obj = {
        "p_id": e.target.p_id.value,
        "p_name": e.target.p_name.value,
        "p_cost": e.target.p_cost.value
    }
    this.props.insertProduct(obj)
    this.setState({
        result: "Insert Success"
    })
    arr.push(obj)
}
update = (e) => {
    let obj = {
        "p_id": e.target.p_id.value,
        "p_name": e.target.p_name.value,
        "p_cost": e.target.p_cost.value
    }
    this.props.updateProduct(obj)
    this.setState({
```

```
            result: "Update Success"
        })
        arr.forEach((e) => {
            if (e.p_id == obj.p_id) {
                e.p_name = obj.p_name
                e.p_cost = obj.p_cost
            }
        })
    }
    delette = (_id) => {
        this.props.deleteProduct(_id)
        this.setState({
            result: "Delete Success"
        })
        arr.splice(arr.findIndex((e, i) => {
            return e.p_id == _id
        }), 1)
    }
    render() {
        arr = this.props.data
        return (
            <div className='container mt-5'>
                <button className='btn btn-outline-primary mb-2 mr-auto'
                    onClick={() => { this.showPopup('addRec') }}>
                    Add +
                </button>
                {/* -----  modal code start------- */}
                <Modal show={this.state.status}
                    onHide={this.closePopup}
                    size='sm'
                    centered>
                    <div className='modal-header'>
                        <div className='modal-title'>Add / Update</div>
                    </div>
                    <div className='modal-body'>
                        <form onSubmit={this.save}>
                            <div className='form-group'>
                                <label>P_ID</label>
                                <input type='number'
                                    className='form-control my-2'
                                    placeholder='Enter P_ID'
                                    name='p_id'></input>
                            </div>

                            <div className='form-group'>
                                <label>P_NAME</label>
                                <input type='text'
                                    className='form-control my-2'
                                    placeholder='Enter P_NAME'
                                    name='p_name'></input>
                            </div>

                            <div className='form-group'>
                                <label>P_COST</label>
                                <input type='number'
                                    className='form-control my-2'
```

```
                                    placeholder='Enter P_COST'
                                    name='p_cost'></input>
                            </div>
                            <input type='submit' value='Add / Update'
className='btn btn-success m-3'></input>
                            <button className='btn btn-danger m-3'
onClick={this.closePopup}>Close</button>
                        </form>
                    </div>
                </Modal>
                {/* -----  table code start------- */}
                <Table bordered
                    variant='primary'
                    size='sm'
                    hover
                    striped
                    className='text-center'>
                    <thead>
                        <tr>
                            <th>SNO</th>
                            <th>ID</th>
                            <th>NAME</th>
                            <th>COST</th>
                            <th>EDIT</th>
                            <th>DELETE</th>
                        </tr>
                    </thead>
                    <tbody>
                        {arr.map((element, index) => (
                            <tr key={index}>
                                <td>{index + 1}</td>
                                <td>{element.p_id}</td>
                                <td>{element.p_name}</td>
                                <td>{element.p_cost} </td>
                                <td><button className='btn btn-warning'
onClick={() => { this.showPopup("update") }}> E </button> </td>
                                <td><button className='btn btn-outline-danger'
onClick={() => { this.delette(element.p_id) }}> D </button> </td>
                            </tr>
                        ))}
                    </tbody>
                </Table>
                <h1 className='text-info'>{this.state.result} </h1>
            </div>
        )
    }
}
const receive = (state) => {
    return {
        data: state.data
    }
}
const send = (dispatch) => {
    return {
        getProducts: () => { dispatch(actions.getProducts()) },
```

```
        insertProduct: (record) => { dispatch(actions.insertProduct(record))
},
        updateProduct: (record) => { dispatch(actions.updateProduct(record))
},
        deleteProduct: (id) => { dispatch(actions.deleteProduct({ "p_id": id
})) }
    }
}
export default connect(receive, send)(App)
```

Deploying react application to (Firebase)

* build ReactJS application
  >npm run build

1. https://console.firebase.google.com/
2. Create new project
3. Continue 2 times
4. Configure Google Analytics -> default account
5. Click on create project wait till finish setup
   click on continue
6. Click on web (</>)
   register app
   add firebase sdk
   left side panel under build select hosting
7. click on get started
8. install firebase tools
   >npm install -g firebase-tools
9. after installing click on next
10. Initialyse your project
    Sign in to google
    >firebase login
11. initialyse project
    >firebase init
    - y
    - select hosting Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
    - hit spacebar to select and hit enter
    - use existing project -> select projectname
    - public directory 'build'
    - configure single page application -> y
    - setup auto deploy -> no
    - **DO NOT OVERWRITE index.html**
12. Click on next
13. Firebase deploy
    >firebase deploy
14. Click on continue to console