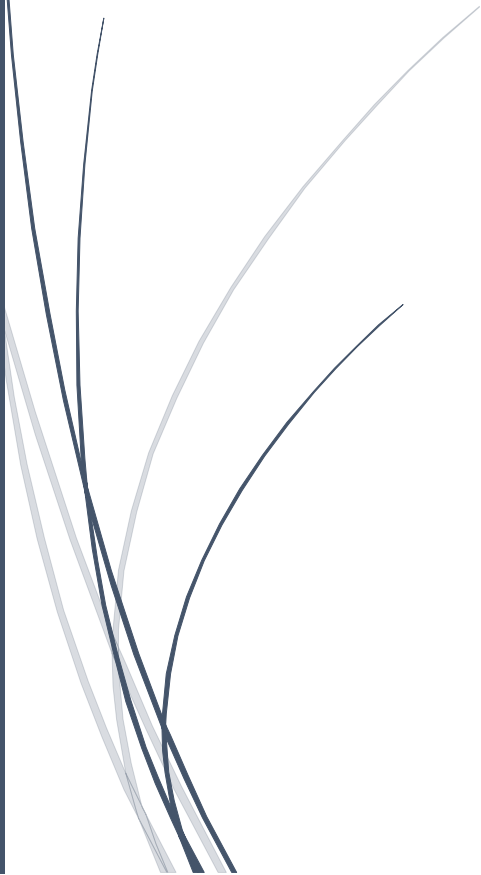




React Js



## Syllabus:

- Components
- Multiple Components
- State&Props
- Events&refs
- Pure Components
- Life cycle Hooks
- Unit Test Cases
- Introduction to Redux
- Multiple Reducers
- Integration of react and redux
- Saga Middleware
- Thunk Middleware
- Single Page Applications
- Lazy Loading
- Routing Parameters in SPA
- Crud Operations (MYSQL)
- Mini Project (MYSQL-MSSQL-MONGODB)
- Crud Operations (Thunk-Middleware)
- Mini project (Saga-Middleware)
- Server Files (MYSQL-MSSQL-MONGODB)



## Commands

- Create the React application

```
Create-react-app <application-name>
```

- Switch to project

```
Cd <project-name>
```

- Run the Project

```
yarn start
```

- Run the server file

```
Node <server filename.js>
```

- Download node modules

```
yarn add <module name1> <module name2>... --save
```

- Download bootstrap ,react bootstrap,material-ui

```
yarn add bootstrap react-bootstrap @material-ui --save
```



## Introduction

- ✓ React is the UI library.
- ✓ React library given by facebook.
- ✓ React acting as UI Layer in web applications.
- ✓ ReactJS can interact with the Angular,VueJS,.....
- ✓ we will develop ReactJS Applications by using JSX.
- ✓ "JSX" stands for JavaScript + XML.
- ✓ in general, browsers won't understand XML.
- ✓ so, we must convert XML to Equivalent JavaScript.
- ✓ React simplifies the Complex UI with the help of Components.
- ✓ React Applications are Component Based Applications.
- ✓ React Components are Reusable.
- ✓ "babel" is the tool used to convert the XML to Equivalent JavaScript.
- ✓ "React" Applications are faster applications, because of virtual DOM.

## ReactJS Environmental Setup

### 1) download and install NodeJS

- React Installation Depending on Node Server.
- "npm" is the tool available in NodeJS, helps to install the React.
- "npm" stands for node packaging manager.

website : <https://nodejs.org/en/>

file : node-v14.4.0-x64.msi



## 2) install yarn tool

- "yarn" tool provided by facebook
- "yarn" tool used to download the libraries.
- we will install yarn tool by using command.

```
> npm install -g yarn@latest
```

- "npm" stands for node packaging manager
- "-g" stands for global installation

## 3) install "create-react-app" tool

- "create-react-app" tool used to create the "react applications"
- "create-react-app" tool provided by facebook.
- we will install "create-react-app" tool by using npm.

```
> npm install -g create-react-app
```

## steps to create the react applications

### 1) create the directory

Ex. Demo

### 2) create the react application

```
> create-react-app first-app
```

- where "first-app" is the react application

### 3) switch to react application

```
> cd first-app
```

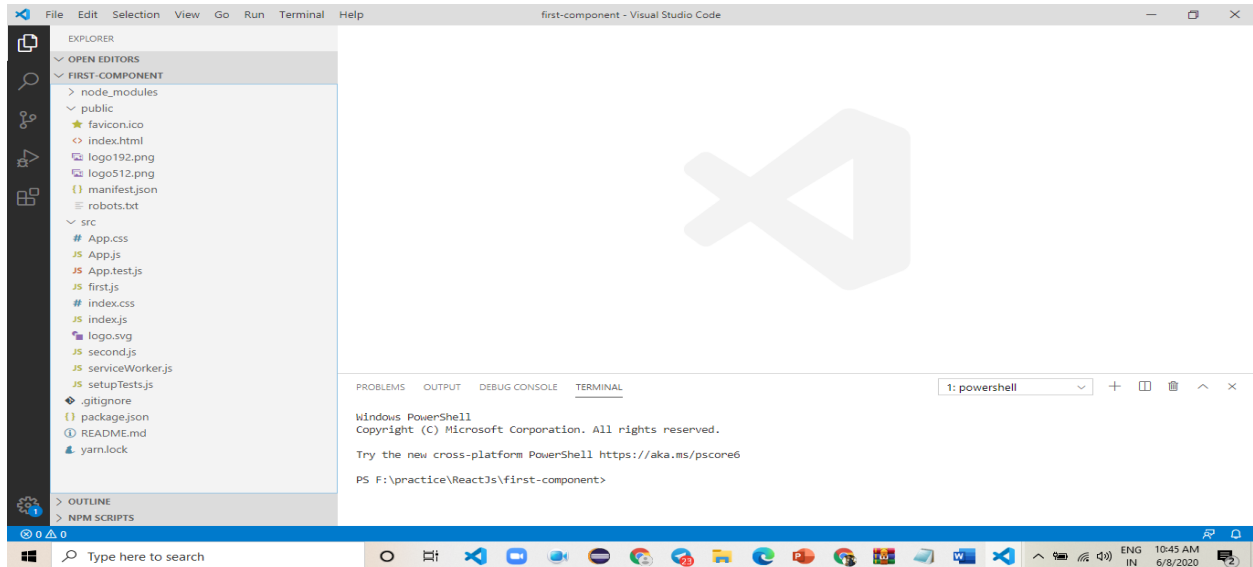
### 4) execute the react application

```
> yarn start
```



- by default react application running on port no.3000

## Directory Structure



```
> create-react-app first-app
```

### node\_modules

- node\_modules contains libraries.
- these libraries help to execute the react applications.

### public/index.html

- this template is the main template.
- main template contains DOM Element(div) whose id is "root".
- we will manipulate above DOM Element with the help of react library.

### public/favicon.ico

- /logo192.png
- /logo512.png



- these logos helps to execute the react applications in all devices with clear clarity images.

public/manifest.json

- To create shortcut icons on devices screens, we must configure the app details in manifest.json file.
- we will create shortcut icons for eazily accessing.

public/robots.txt

- it is used to define secuurity rules to applications.

src

- it is used to deploy the applications resorces

Ex. Components,style sheets,images,audio files,video files

src/App.js

App.css

App.test.js

- App compoennt is the default component.
- "App.css" file is the default style sheet for App Component.
- "App.test.js" file is the Testing file for the App Component.

src/index.js

- this file is the entry file of react application.
- ReactDOM is the predefined class in react library,helps to render the particular component to DOM Element.



### `src/serviceWorker.js`

- this file used to develop the mobile applications.

### `src/setupTests.js`

- this file helps to write the unit test cases.

### `package.json`

- `package.json` file used to download the 3rd party libraries.

Ex.

`redux`

`redux-thunk`

`redux-saga`

`axios`

### `yarn.lock`

- this file is the configuration file of yarn tool





## Chapter-1 (Components)

- Simple "JSX Class" behaves like Component.
- React Applications are Component Based Applications.
- we can create more than one Component.
- These Components are Reusable.
- Because of Components, React Simplifies the Complex UI.

React Offers Two Types of Components.

- 1) Stateless Components (Functional Components)
- 2) Stateful Components (Class Components)

### Stateful Components

- Simple JSX Class behaves like Stateful Component.
- Stateful Components have so many advantages Compared to Stateless Components.
- Stateful Components offers "life cycle hooks", whereas Stateless Components wont offers.
- Stateful Components offers "state" object to store Component data.

#### 1) create the react project

```
> create-react-app first-app
```

#### 2) switch to react project

```
> cd first-app
```

#### 3) add the react bootstrap

- "react-bootstrap" is the library provided by facebook.



- "react-bootstrap" is used to create the Rich UI (Components) .

> yarn add react-bootstrap bootstrap --save

4) add the related CDN'S provided by facebook in "index.html"

```
<script
src="https://unpkg.com/react/umd/react.production.min.js"
crossorigin></script>

<script src="https://unpkg.com/react-dom/umd/react-
dom.production.min.js" crossorigin></script>

<script src="https://unpkg.com/react-bootstrap@next/dist/react-
bootstrap.min.js" crossorigin></script>

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/css/bootst
rap.min.css" integrity="sha384-
9aIt2nRpC12Uk9gS9baD1411NQApFmC26EwAOH8WgZl5MYYYxFfc+NcPb1dKGj7Sk
"crossorigin="anonymous"/>
```

5) create the Stateful Component

Directory Structure

```
*****

first-app

    src

        First.js

        index.js

*****
```



First.js

```
import React, { Component } from "react";

import Alert from "react-bootstrap/Alert";

import Container from "react-bootstrap/Container";

import Button from "react-bootstrap/Button";

export default class First extends React.Component{

  render() {

    return(

      <div>

        <Container fluid>

          /* <Alert variant={'primary'}>Primary Alert</Alert>

            <Alert variant={'info'}>Info Alert</Alert>

            <Alert variant={'danger'}>Danger Alert</Alert>

            <Alert variant={'dark'}>Dark Alert</Alert>

            <Alert variant={'secondary'}>secondary Alert</Alert>

            <Alert variant={'warning'}>Warning Alert</Alert>

            <Alert variant={'light'}>light Alert</Alert>

            <Alert variant={'success'}>success Alert</Alert>

          */

        {/*

          [['primary','info','danger','success','dark']].map((element,index) => (
```



```
        <Alert variant={element}>Alert</Alert>

    ))) */}

    <Alert variant={'danger'}>

    <Alert.Heading>Heading</Alert.Heading>

    <p>Ea dolores voluptua dolor eos sadipscing et ipsum lorem
    gubergren, no et diam nonumy nonumy dolores, takimata rebum sed
    duo stet magna, clita tempor amet clita sea amet, ut voluptua
    consetetur at et sed kasd, at eirmod eirmod sit at diam. Est sed
    ut no justo nonumy. Invidunt elitr amet.</p>

    <Button>Learn More</Button>

    </Alert>                </Container>

        </div>

    )   } };
```

### Index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import First from './First';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

    <First />,

    document.getElementById('root'));
```



// unregister() to register() below. Note this comes with some pitfalls.

```
serviceWorker.unregister();
```

\*\*\*\*\*

## Stateless Components

- in functional level components, we can't define state.
- in functional level components, everything we must define.
- in functional level components, no life cycle hooks to handle the applications

### 1) create the react project

```
> create-react-app first-app
```

### 2) switch to react project

```
> cd first-app
```

### 3) create the Stateless Component

## Directory Structure

\*\*\*\*\*

first-app

src

App.js

User.js

index.js

\*\*\*\*\*



App.js

```
import React,{ Component } from "react";

import User from "../User";

export default class App extends React.Component{

  render(){

    return(

      <div>

        <User key1="welcome to">NareshIT</User>

        <User key1="18.x version of">ReactJS</User>

        <User key1="popular database is ">MongoDB</User>

      </div>

    )

  };

};
```

User.js

```
import React from "react";

let User = (props)=>{

  return (<div>{props.key1} {props.children}</div>);

};

export default User;
```



Index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>,

  document.getElementById('root')

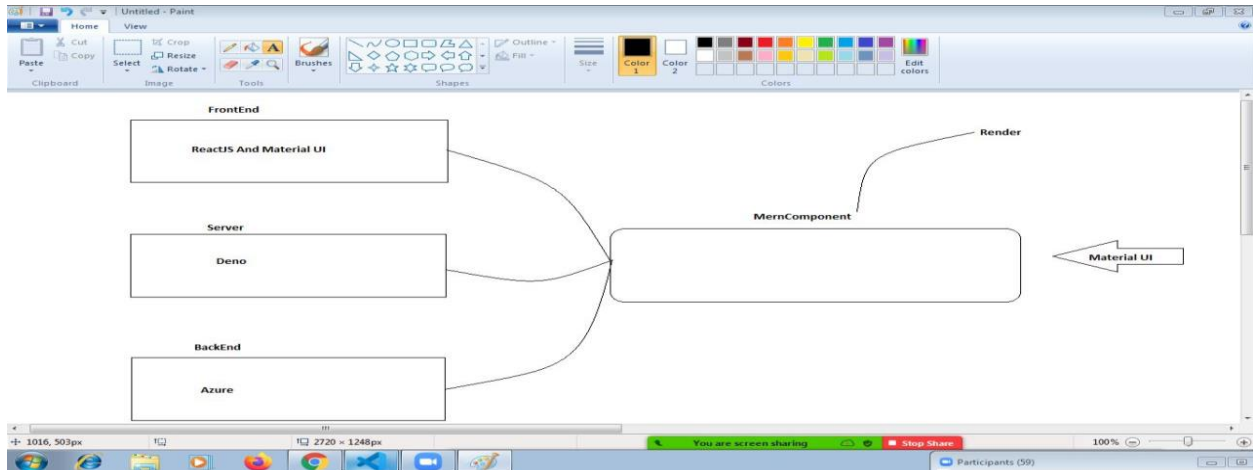
);

// If you want your app to work offline and load faster, you can
change

// unregister() to register() below. Note this comes with some
pitfalls.
```



## Chapter-2 (Multiple-Components)



1) create the react application

```
> create-react-app demo-app
```

2) switch to react application

```
> cd demo-app
```

3) add the "material ui" library

➤ "material ui" library used to design the rich templates

```
"https://material-ui.com/"
```

```
yarn add @material-ui/core
```

```
yarn add @material-ui/icons
```

index.html

```
<link rel="stylesheet"
```

```
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" />
```





```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons"
/>
```

```
<script src="https://unpkg.com/@material-
ui/core@latest/umd/material-ui.development.js"></script>
```

```
<script src="https://unpkg.com/@material-
ui/core@latest/umd/material-ui.production.min.js"></script>
```

#### 4) create the Components

\*\*\*\*\*

demo-app

src

FrontEnd.js

Server.js

BackEnd.js

MernComponent.js

index.js

\*\*\*\*\*

#### FrontEnd.js

```
import React,{ Component } from "react";
```

```
import Typography from '@material-ui/core/Typography';
```

```
import Avatar from "@material-ui/core/Avatar";
```

```
export default class FrontEnd extends React.Component{
```

---



```
render() {  
  return(  
    <div>  
      <Typography variant="h5">ReactJS With React  
Material UI</Typography>  
      <Avatar alt="Remy Sharp"  
src="https://restcountries.eu/data/afg.svg" />  
    </div>  
  )  
}  
};
```

### Server.js

```
import React,{ Component } from "react";  
import Typography from '@material-ui/core/Typography';  
export default class Server extends React.Component{  
  render() {  
    return(  
      <div>  
        <Typography variant="h5">Deno</Typography>  
      </div>  
    )  
  } };
```



### BackEnd.js

```
import React,{ Component } from "react";

import Typography from '@material-ui/core/Typography';

export default class BackEnd extends React.Component{

  render(){

    return(

      <div>

        <Typography variant="h5">Azure</Typography>

      </div>

    )

  }

};
```

### MernComponent.js

```
import React,{ Component } from "react";

import FrontEnd from "./FroneEnd";

import Server from "./Server";

import BackEnd from "./BackEnd";

export default class MernComponent extends React.Component{

  render(){

    return(

      <div>
```



```
        <FrontEnd />

        <Server />

        <BackEnd />

    </div>

)

}

};
```

### index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import MernComponent from "./MernComponent";

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

    <MernComponent />,

    document.getElementById('root')

);


// If you want your app to work offline and load faster, you can
change
```



// unregister() to register() below. Note this comes with some pitfalls.

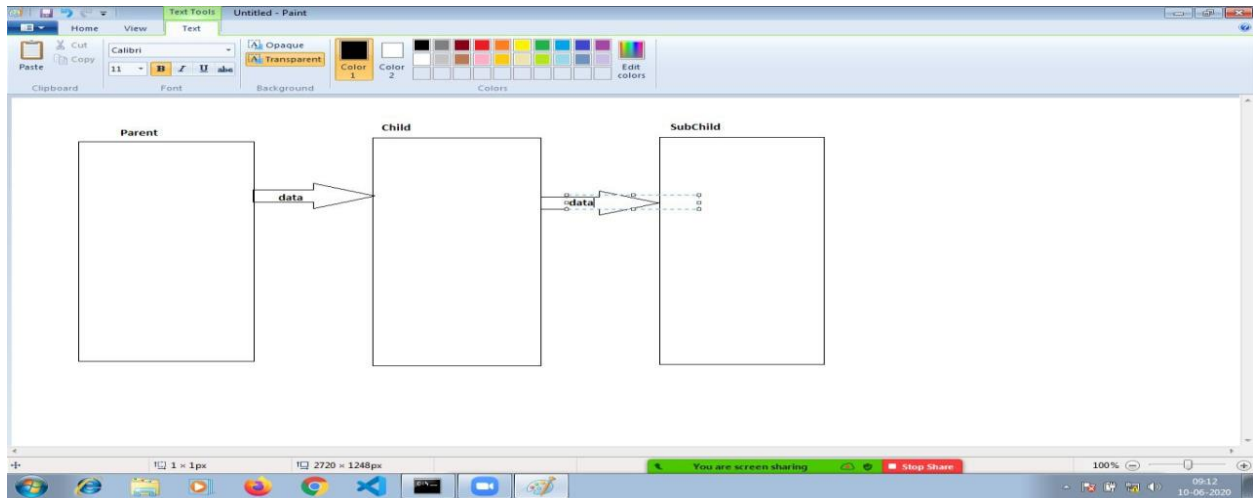
// Learn more about service workers: <https://bit.ly/CRA-PWA>

serviceWorker.unregister();

\*\*\*\*\*



## Passing the Data from Parent to SubChilds:



1) create the react project

```
> create-react-app data-app
```

2) switch to react project

```
> cd data-app
```

3) create the Components

### Directory Structure

\*\*\*\*\*

data-app

src

Parent.js

Child.js

Subchild.js

index.js

\*\*\*\*\*



### Parent.js

```
import React,{ Component } from "react";

import Child from "../Child";

export default class Parent extends React.Component{

  constructor(){

    super();

    this.state = {

      data : "welcome to reactjs"

    }

  }

  render(){

    return(

      <div>

        <Child key1={this.state.data}/>

      </div>

    )

  }

};
```

### Child.js

```
import React,{ Component } from "react";

import Subchild from "../Subchild";
```

---



```
export default class Child extends React.Component{

  constructor() {

    super();

    this.state = {

      data : "Material UI"

    }

  }

  render() {

    return(

      <div>

        <Subchild {...this.props}
key2={this.state.data}/>

      </div>

    )

  }

};
```

### Subchild.js

```
import React,{ Component } from "react";

export default class Subchild extends React.Component{

  render() {

    return(

      <div>
```





```
<h1>{this.props.key1}.....{this.props.key2}</h1>

      </div>

    )

  }}
}
```

### index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import Parent from './Parent';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

  <Parent />,

  document.getElementById('root')

);

// If you want your app to work offline and load faster, you can
change

// unregister() to register() below. Note this comes with some
pitfalls.
```



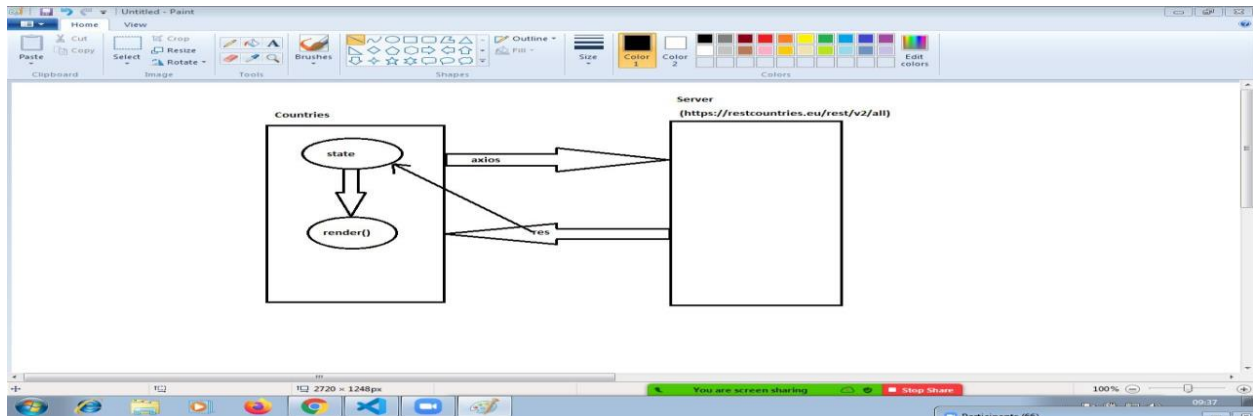
// Learn more about service workers: <https://bit.ly/CRA-PWA>

`serviceWorker.unregister();`

\*\*\*\*\*



## Get The Data from URL/SERVER:



1) create the react application

```
> create-react-app get-ex
```

2) switch to react application

```
> cd get-ex
```

3) add the axios module

```
yarn add axios --save
```

4) create the Components

### Directory Structure

\*\*\*\*\*

Get-ex

src

Countries.js

index.js

\*\*\*\*\*



### Countries.js

```
import React,{ Component } from "react";

import axios from "axios";

export default class Countries extends React.Component{

  constructor() {

    super();

    this.state = {

      arr : []

    };

  }

  componentDidMount() {

    axios.get("https://restcountries.eu/rest/v2/all")

      .then((posRes)=>{

        this.setState({

          arr : posRes.data

        })

      }, (errRes)=>{

        console.log(errRes);

      });

  }

};
```



```
render() {  
  return(  
    <div>  
      <table border="1"  
        cellPadding="10px"  
        cellSpacing="10px"  
        align="center">  
        <thead style={{backgroundColor: "grey"}}>  
          <tr>  
            <th>SNO</th>  
            <th>Name</th>  
            <th>Capital</th>  
            <th>NativeName</th>  
            <th>Population</th>  
            <th>Flag</th>  
          </tr>  
        </thead>  
        <tbody>  
          {this.state.arr.map((element, index) =>(  
            <tr key={index}>  
              <td>{index+1}</td>  
              <td>{element.name}</td>
```



```
        <td>{element.capital}</td>

        <td>{element.nativeName}</td>

        <td>{element.population}</td>

        <td><img width="100px"
height="50px" src={element.flag}></img></td>

    </tr>

    )))

</tbody>

</table>

</div>

)

}

};
```

### index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import Countries from './Countries';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(
```

---



```
<Countries />,

document.getElementById('root')

);

// If you want your app to work offline and load faster, you can
change

// unregister() to register() below. Note this comes with some
pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();
```



## Chapter-3 (State&Props)

- State is used to store the Component data.
- "state" is the predefined property in reactjs.
- Recommended place to define "state" is constructor.
- constructor will execute at booting time.
- React components has a built-in `state` object.
- The `state` object is where you store property values that belongs to the component.
- When the `state` object changes, the component re-renders.

### 1) create the react project

```
> create-react-app first-app
```

### 2) switch to react project

```
> cd first-app
```

### 3) create the Stateless Component

#### Directory Structure

```
*****
```

```
first-app
  src
    App.js
    index.js
```

```
*****
```

#### App.js

```
import React from 'react';
```





```
export default class App extends React.Component
{
  constructor()
  {
    super();
    this.state={
      boolean:"true",
      error:11001,
      data:"data from server soon .....!",
      "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}]

    }
  }
  render() {
    return (
      <div>
        <table border="1"
          cellPadding="10px"
          cellSpacing="10px"
          align="center">
          <thead style={{backgroundColor:"grey"}}>
            <tr>
              <th>SNO</th>
              <th>PID</th>
              <th>PNAME</th>
              <th>PCOST</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>1</td>
              <td>111</td>
              <td>p_one</td>
              <td>10000</td>
            </tr>
            <tr>
              <td>2</td>
              <td>222</td>
              <td>p_two</td>
              <td>20000</td>
            </tr>
            <tr>
              <td>3</td>
              <td>333</td>
              <td>p_three</td>
              <td>30000</td>
            </tr>
            <tr>
              <td>4</td>
              <td>444</td>
              <td>p_four</td>
              <td>40000</td>
            </tr>
            <tr>
              <td>5</td>
              <td>555</td>
              <td>p_five</td>
              <td>50000</td>
            </tr>
          </tbody>
        </table>
      </div>
    );
  }
}
```



```
        </tr>
      </thead>
      <tbody>
        {this.state.products.map((element,index)=>
(
          <tr>
            <td>{index+1}</td>
            <td>{element.p_id}</td>
            <td>{element.p_name}</td>
            <td>{element.p_cost}</td>
          </tr>
        ) ) }
      </tbody>
    </table>
    <h1>{JSON.stringify(this.state.boolean)}</h1>
    <h1>{this.state.error}</h1>
    <h1>{this.state.data}</h1>
  </div>
)
}
}
```

### index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import * as serviceWorker from './serviceWorker';
```



```
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
);  
  
// If you want your app to work offline and load faster, you can  
change  
  
// unregister() to register() below. Note this comes with some  
pitfalls.  
  
// Learn more about service workers: https://bit.ly/CRA-PWA  
serviceWorker.unregister();
```

\*\*\*\*\*

## setState()

- To change a value in the state object, use the `this.setState()` method.
- When a value in the `state` object changes, the component will re-render, meaning that the output will change according to the new value(s).

### 1) create the react project

```
> create-react-app first-app
```

### 2) switch to react project

```
> cd first-app
```



### 3) create the Stateless Component

#### Directory Structure

\*\*\*\*\*

first-app

src

App.js

index.js

\*\*\*\*\*

#### App.js

```
import React,{ Component } from "react";
export default class App extends React.Component{
  constructor() {
    super();
    this.state = {
      "sub" : "Angular9"
    };
  };
  render() {
    return(
      <div>
<h1>Sub:<span style={{color:'red'}}>{this.state.sub}</span></h1>
      <br></br>
      <button onClick={this.myFun}>Change</button>
      </div>
    )
  }
}
```



```
};  
myFun = ()=>{  
  this.setState({  
    sub : "Angular10"  
  });  
};  
};
```

### index.js

```
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import './index.css';  
  
import App from './App';  
  
import * as serviceWorker from './serviceWorker';  
  
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
)  
  
  
// If you want your app to work offline and load faster, you can  
change  
  
// unregister() to register() below. Note this comes with some  
pitfalls.  
  
// Learn more about service workers: https://bit.ly/CRA-PWA
```



```
serviceWorker.unregister();
```

```
*****
```

## Props:

- "props" is the predefined property in react library
- "props" used to receive the data from 3rd party resources (Components, Redux, Saga, Thunk, ....)
- "Props" is a special keyword in React, which stands for properties and is being used for passing data from one component to another.
- "props" are basically kind of global variable or object.
- The data in states and props are used to render the Component with dynamic data.
- Props are passed to components via HTML attributes.
- We can say, props are immutable (unable to be changed).

### 1) create the react project

```
> create-react-app first-app
```

### 2) switch to react project

```
> cd first-app
```

### 3) create the Stateless Component

## Directory Structure

---



\*\*\*\*\*

first-app

src

First.js

second.js

index.js

\*\*\*\*\*

### First.js-:

- In this project, we will create two components.
- First.js is the first component.
- We will share the data from First.js(first component) to Second.js(second component) by using state() method.
- Here we will call the second component.

Code:

```
import React,{ Component } from "react";

import Second from "../Second";

export default class First extends React.Component{

  constructor(){

    super();

    this.state = {

      "data" : "React",

      "version" : 16.8,
```



```
        "flag" : true,

        "obj" : {"cost":100000},

        "arr" : [{"eno":111,"ename":"e_one","esal":10000},

                   {"eno":222,"ename":"e_two","esal":20000},

{"eno":333,"ename":"e_three","esal":30000},

                   {"eno":444,"ename":"e_four","esal":40000},

                   {"eno":555,"ename":"e_five","esal":50000}]

    };

}

render() {

    return(

        <div>

            <Second key1={this.state.data}

                        key2={this.state.version}

                        key3={this.state.flag}

                        key4={this.state.obj}

                        key5={this.state.arr}/>

        </div>

    )

};

};
```

---





### Second.js-:

- This is our second component in this project.
- We will receive the data from First.js by using a predefined keyword "props".

Code:

```
import React,{ Component } from "react";

export default class Second extends React.Component{

  render(){

    return(

      <div>

        <table border="1"

          cellPadding="10px"

          cellSpacing="10px"

          align="center">

          <thead style={{backgroundColor:"grey"}}>

            <tr>

              <th>SNO</th>

              <th>ENO</th>

              <th>ENAME</th>

              <th>ESAL</th>

            </tr>
```



```
        </thead>

        <tbody>

            {this.props.key5.map((element,index)=>(

                <tr>

                    <td>{index+1}</td>

                    <td>{element.eno}</td>

                    <td>{element.ename}</td>

                    <td>{element.esal}</td>

                </tr>

            ) ) }

        </tbody>

    </table>

    <h1>{JSON.stringify(this.props.key4)}</h1>

    <h1>{JSON.stringify(this.props.key3)}</h1>

    <h1>{this.props.key2}</h1>

    <h1>{this.props.key1}</h1>

</div>

)

}

};
```

Index.js-:



- It is the registration file of react project.
- Here we will register our first component.

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import First from './First';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

  <React.StrictMode>

    <First />

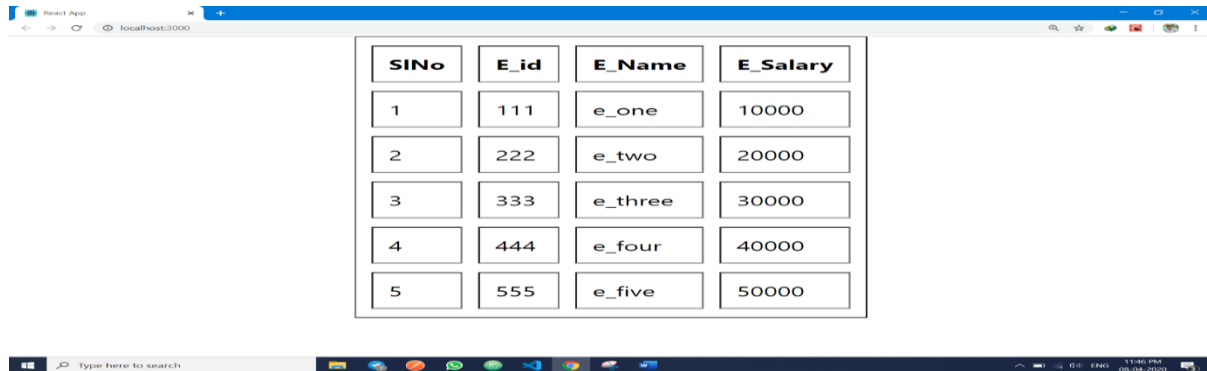
  </React.StrictMode>,

  document.getElementById('root')

);

serviceWorker.unregister();
```



**OutPut:**

The screenshot shows a web browser window with a table containing employee data. The table has four columns: SNo, E\_id, E\_Name, and E\_Salary. The data is as follows:

SNo	E_id	E_Name	E_Salary
1	111	e_one	10000
2	222	e_two	20000
3	333	e_three	30000
4	444	e_four	40000
5	555	e_five	50000

**Difference Between States and Props in react JS-:**

state	props
<ol style="list-style-type: none"><li>1. State is variables, but directly initialized and managed by the component.</li><li>2. A state is a variable which exists inside a component, that cannot be accessed and modified outside the component and can only be used inside the component. Works very similarly to a variable that is declared inside a function that cannot be accessed outside the scope of the function in normal Javascript.</li><li>3. The component itself will update the state using the setState() function.</li><li>4. States are declared inside the constructor via super() method.</li></ol>	<ol style="list-style-type: none"><li>1. Props are owned by a parent component</li><li>2. props are read-only in the child component that receives from parent component. However, call back functions can also be passed, which can be executed inside the child to initiate an update.</li><li>3. props are immutable (unable to be changed).</li><li>4. Props" is a special keyword in React, which stands for properties and is being used for passing data from one component to another.</li></ol>



## Chapter-4 (Events&Refs)

### Events:

1) create the react project

```
> create-react-app events-app
```

2) switch to react project

```
> cd events-app
```

3) create the Event Component

### Directory Structure

```
*****
```

```
events-app
```

```
  src
```

```
    Event.js
```

```
    index.js
```

```
*****
```

### Event.js

```
// import React,{ Component } from "react";

// export default class Events extends React.Component{

//     fun_one = ()=>{

//         console.log("event without parameters");
```



```
//    };

//    render() {

//        return(

//            <div>

//                <button
onClick={this.fun_one}>ClickMe</button>

//            </div>

//        )

//    }

// };


// import React,{ Component } from "react";

// export default class Events extends React.Component{

//     fun_one = (arg1,arg2)=>{

//         if(arg1 == "admin" && arg2 == "admi"){

//             alert("Login Success");

//         }else{

//             alert("Login Fail");

//         }

//     };

//     render() {
```

---



```
//      return(  
//          <div>  
//              <button onClick={ ()=>{  
this.fun_one("admin","admin") } }>ClickMe</button>  
//          </div>  
//      )  
//  };  
//  };
```

### index.js

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import './index.css';  
import App from './App';  
import Events from './Events';  
import * as serviceWorker from './serviceWorker';  
  
ReactDOM.render(  
  <Events />,  
  document.getElementById('root')  
) ;  
  
// If you want your app to work offline and load faster, you can  
change  
// unregister() to register() below. Note this comes with some  
pitfalls.  
// Learn more about service workers: https://bit.ly/CRA-PWA  
serviceWorker.unregister();
```



\*\*\*\*\*

## Refs:

"ref" is the attribute used to assign logical name to DOM Elements.

### 1) create the react project

```
> create-react-app events-app
```

### 2) switch to react project

```
> cd events-app
```

### 3) add axios module

```
> yarn add axios --save
```

### 4) create the Event Component

## Directory Structure

\*\*\*\*\*

events-app

src

Event.js

index.js

\*\*\*\*\*

## Event.js

```
import React,{ Component } from "react";
```





```
export default class Events extends React.Component{

  constructor() {

    super();

    this.state = {

      login : ""

    };

  };

  render() {

    return(

      <div>

        <fieldset>

          <legend>Login Form</legend>

          <input type="text"

            ref="uname"

            placeholder="User Name"></input>

          <br></br><br></br>

          <input type="password"

            ref="upwd"

            placeholder="User Password"></input>
```



```
        <br></br><br></br>

        <button onClick={this.login}>Login</button>

        <br></br><br></br>

        <h1>{this.state.login}</h1>

    </fieldset>

</div>

)

}

login = ()=>{

    if( this.refs.uname.value == "admin" &&
this.refs.upwd.value == "admin"){

        this.setState({

            login : "success"

        })

    }else{

        this.setState({

            login : "fail"

        })

    }

}
```



```
};
```

### index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import Events from './Events';
import * as serviceWorker from './serviceWorker';
```

```
ReactDOM.render(
  <Events />,
  document.getElementById('root')
);
```

```
// If you want your app to work offline and load faster, you can
change
```

```
// unregister() to register() below. Note this comes with some
pitfalls.
```

```
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

```
*****
```

### Http Post Request:

1) create the react project

```
> create-react-app events-app
```

2) switch to react project



```
> cd events-app
```

### 3) create the Event Component

#### Directory Structure

```
*****
```

```
events-app
```

```
  src
```

```
    Event.js
```

```
    index.js
```

```
*****
```

#### Event.js

```
import React,{ Component } from "react";
```

```
import axios from "axios";
```

```
export default class Events extends React.Component{
```

```
  constructor(){
```

```
    super();
```

```
    this.state = {
```

```
      res : {}
```

```
    };
```

```
  }
```

```
  render(){
```

```
    return(
```



```
    <div>

      <input type="text"

        ref="msg"

        placeholder="Enter Message"></input>

      <button onClick={this.fun_one}>Change</button>

      <h1>{JSON.stringify(this.state.res)}</h1>

    </div>

  )

};

fun_one = ()=>{

  axios.post("http://test-
routes.herokuapp.com/test/uppercase", {"message":this.refs.msg.va
lue})

    .then((posRes)=>{

      this.setState({

        res : posRes.data

      })

    }, (errRes)=>{

      console.log(errRes);

    });

}

};
```



## index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import Events from './Events';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
  <Events />,
  document.getElementById('root')
);

// If you want your app to work offline and load faster, you can
// change
// unregister() to register() below. Note this comes with some
// pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

\*\*\*\*\*

## Sharing the data from class level component to functional level component

1) create the react project

```
> create-react-app demo-app
```

2) switch to react project



```
> cd demo-app
```

### 3) create the Components

#### Directory Structure

```
*****
```

```
demo-app
```

```
  src
```

```
    Compone.js    (class level)        (state)
```

```
    Comptwo.js    (functional level)    (props)
```

```
    styles.css
```

```
    index.js
```

```
*****
```

#### Compone.js

```
import React,{ Component } from "react";
```

```
import Comptwo from "../Comptwo";
```

```
export default class Compone extends React.Component{
```

```
  constructor(){
```

```
    super();
```

```
    this.state = {
```

```
      data1 : "ReactJS",
```

```
      data2 : "Material UI",
```

```
      data3 : "Deno",
```



```
        data4 : "MongoDB"

    };

};

render() {

    return(

        <div>

            <Comptwo s_o={this.state.data1}

                s_t={this.state.data2}

                s_three={this.state.data3}

                s_f={this.state.data4}/>

        </div>

    )

}

};
```

### Comptwo.js

```
import React from "react";

import "./styles.css"

const Comptwo = (props)=>{

    return(

        <div>
```





```
        <h2 className="sub_one">{props.s_o}</h2>

        <h2>{props.s_t}</h2>

        <h2>{props.s_three}</h2>

        <h2>{props.s_f}</h2>

      </div>

    )
  };

export default Comptwo;
```

### styles.css

```
.sub_one{

  color: white;

  background : linear-gradient(180deg,red,black);

  text-align: center;

  padding: 10px;

  margin: 10px;

  border-radius: 20px;

  box-shadow: 0px 0px 10px palevioletred;

}
```

### index.js

```
import React from 'react';

import ReactDOM from 'react-dom';
```



```
import './index.css';

import App from './App';

import Functional from "./Functional";

import Compone from "./Compone";

import Demol from "./Demol";

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

  <compone />,

  document.getElementById('root')

);

// If you want your app to work offline and load faster, you can
change

// unregister() to register() below. Note this comes with some
pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();
```

```
*****
```



## Chapter-5 (Pure-components)

Just like Pure Functions in JavaScript, a React component is considered a Pure Component if it renders the same output for the same state and props value. React provides the Pure Component base class for these class components. Class components that extend the `React.PureComponent` class are treated as pure components.

It is the same as `Component` except that Pure Components take care of `shouldComponentUpdate` by itself, it does the *shallow comparison* on the state and props data. If the previous state and props data is the same as the next props or state, the component is not Re-rendered.

React Components re-renders in the following scenarios:

1. `"setState"` is called in Component
2. `"props"` values are updated
3. `this.forceUpdate()` is called

In the case of Pure Components, the React components do not rerender blindly without considering the updated values of React `"props"` and `"state"`. If updated values are the same as previous values, render is not triggered.

### Pure Components restricts Re-Rendering

Take the scenario below, in which we're updating the component's state variable at a continuous interval of one second. With every call to `"setState"`, we update the counter value to the same value.



Here, `setState` is called and the value of "counter" is set to the same value. When `setState` is called, the component is re-rendered. In this scenario, the updated component view remains the same. There is effectively no difference in the UI – the new values are unchanged. Re-rendering, in this case, is an overhead.

In order to cater to this problem, React introduced Pure Components. They compare the initial and final values for the state and props variables. If there is no difference between the two, they won't trigger the re-rendering logic for the component.

```
class ImpureComponent extends React.PureComponent {
  Constructor ()
  {
    Super ();
    this.State = {
      counter: 0
    }

    // The value of Counter is updated to same value during
    continues interval
    setInterval(() => {
      this.setState({
        counter: 0
      });
    }, 1000);
  }
  render() {

    // This function wont be re-rendered in case when the new
    state is same as previous
  }
}
```

---



```
return <b>Counter Value: {this.state.counter}</b>
  }
}
```

Here, `setState` is called and the value of "counter" is set to the same value. When `setState` is called, the component is re-rendered. In this scenario, the updated component view remains the same. There is effectively no difference in the UI – the new values are unchanged. Re-rendering, in this case, is an overhead.

In order to cater to this problem, React introduced Pure Components. They compare the initial and final values for the state and props variables. If there is no difference between the two, they won't trigger the re-rendering logic for the component.

```
class      ImpureComponent      extends
React.PureComponent {      constructor() {
super();      this.state = {      counter:
0
      }
}
```

```
      // The value of Counter is updated to same value during
continues interval
      setInterval(() => {
this.setState({
counter: 0
      });
      }, 1000);
}
render() {
```

---



```
// This function wont be re-rendered in case when the new
state is same as previous
return <b>Counter Value: {this.state.counter}</b>
}
}
```

In the code above the component is inherited from `React.PureComponent`, each time the state or props are updated it compares the previous and next value – if the values are the same the Render function is not triggered. Performance is improved by not calling "render" recursively.

### Conclusion

Pure Components are introduced for performance enhancement. You can use this optimization to improve the performance of your components

### Example:

#### 1) create the react project

```
> create-react-app demo-app
```

#### 2) switch to react project

```
> cd demo-app
```

#### 3) create the Components

### Directory Structure



\*\*\*\*\*

demo-app

src

Demo1.js (state) (refresh 1 sec)

Demo2.js (props)

Index.js

\*\*\*\*\*

### Demo1.js

```
import React,{ Component,PureComponent } from "react";
import Demo2 from "../Demo2";

export default class Demo1 extends React.PureComponent{

  constructor(){

    super();

    this.state = {

      num : 1

    };

  };

  render(){

    console.log( "Parent Render !!!");

    return(
```



```
        <div>

            <Demo2 key1={this.state.num}/>

        </div>

    );

};

componentDidMount() {

    setInterval(() => {

        this.setState({

            num : Math.random()

        })

    }, 1000);

};

};
```

### Demo2.js

```
import React from "react";

const Demo2 = (props) => {

    console.log("child render");

    return (

        <div>{props.key1}</div>
```





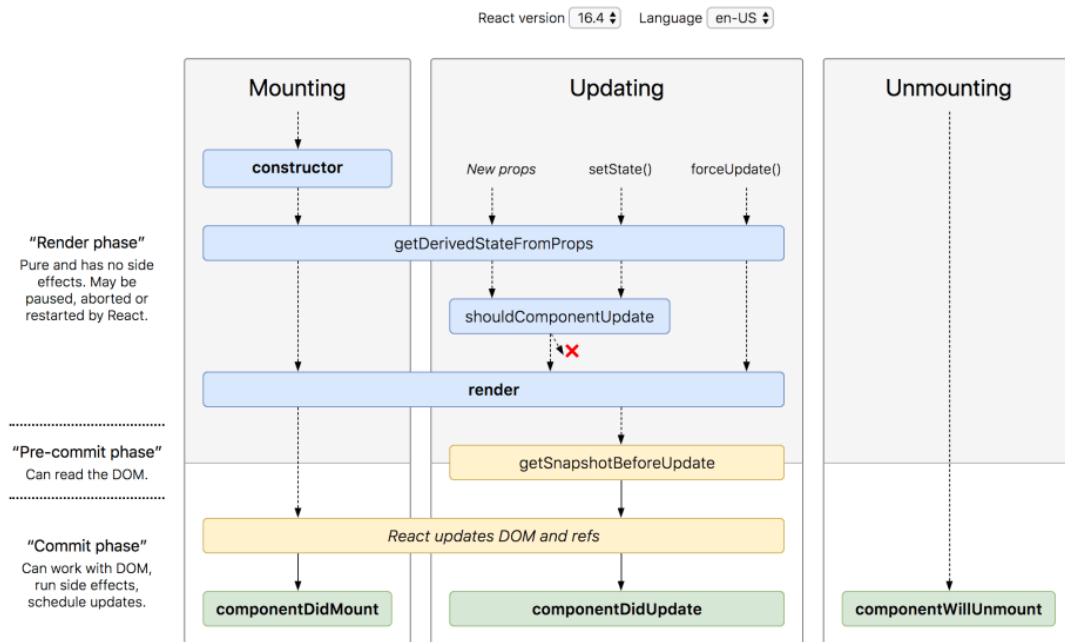
```
    )  
};  
  
export default Demo2;  
  
index.js  
  
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import './index.css';  
  
import App from './App';  
  
import Functional from "./Functional";  
  
import Compone from "./Compone";  
  
import Demo1 from "./Demo1";  
  
import * as serviceWorker from './serviceWorker';  
  
ReactDOM.render(  
  <Demo1 />,  
  document.getElementById('root')  
)  
;  
  
// If you want your app to work offline and load faster, you can  
change  
  
// unregister() to register() below. Note this comes with some  
pitfalls.  
  
serviceWorker.unregister();
```



## Chapter-6 (Life-Cycle-Hooks)

- `componentWillMount` is executed before rendering, on both the server and the client side.
- `componentDidMount` is executed after the first render only on the client side. This is where AJAX requests and DOM or state updates should occur. This method is also used for integration with other JavaScript frameworks and any functions with delayed execution such as `setTimeout` or `setInterval`. We are using it to update the state so we can trigger the other lifecycle methods.
- `componentWillReceiveProps` is invoked as soon as the props are updated before another render is called. We triggered it from `setNewNumber` when we updated the state.
- `shouldComponentUpdate` should return true or false value. This will determine if the component will be updated or not. This is set to true by default. If you are sure that the component doesn't need to render after state or props are updated, you can return false value.
- `componentWillUpdate` is called just before rendering.
- `componentDidUpdate` is called just after rendering.
- `componentWillUnmount` is called after the component is unmounted from the dom. We are unmounting our component in `main.js`.





### Example:

#### 1) create the react project

```
> create-react-app demo-app
```

#### 2) switch to react project

```
> cd demo-app
```

#### 3) create the Components

### Directory Structure

\*\*\*\*\*

demo-app

src

app.js



child.js

Index.js

\*\*\*\*\*

### App.js

```
import React,{ Component } from "react";
import Child from "./Child";
export default class App extends React.Component{
  constructor(){
    super();
    console.log("parent constructor");
    //constructor will execute at booting time
    //constructor will execute only once
    //it's recommended to define state
    this.state = {
      name : "react"
    };
  };
  componentWillMount(){
    //after constructor componentWillMount will execute
    //it will execute only once
    //it is recommended to change initial state
    //it is also recommended to set global parameters like "wi
ndow width", "window height",....
    console.log("parent componentWillMount");
    if(window.innerWidth<600){
      this.setState({
        width : window.innerWidth
      })
    }
  }
}
```



```
    }  
  };  
  render() {  
    //it will execute immediately after componentWillMount  
    //in general we will place presentation logic (JSX)  
    //render() is the mandatory life cycle hook  
    //when ever change detected in state/props automatically  
    this life cycle hook will execute  
    console.log("parent render");  
    return(  
      <div>  
        {this.state.name}  
        <br></br>  
        {this.state.width}  
        <br></br><br></br>  
        <Child key1={this.state.name}></Child>  
        <br></br><br></br>  
        <button onClick={this.changeSub}>Change</button>  
      </div>  
    )  
  };  
  changeSub = ()=>{  
    this.setState({  
      name : "reactjs"  
    });  
  };  
  /*execution flow  
    parent constructor  
    parent componentWillMount  
    parent render
```



```
Child constructor
Child componentWillMount
Child render

//if state/props change detected
parent render
child render
*/

componentDidMount(){
  //after render function componentDidMount will execute
  //first priority goes to child component componentDidMount()
  //life cycle hooks
  //in general, we will make rest api calls Ex. get request
  //it's recommended to change state
  console.log("parent componentDidMount");
};
/*
  execution flow
  parent constructor
  parent componentWillMount
  parent render

  Child constructor
  Child componentWillMount
  Child render

  child componentDidMount
```

---



```
        parent componentDidMount
    */
    componentWillReceiveProps () {
        //if component receives the props, then this life cycle hook will execute
        console.log("parent componentWillReceiveProps");
    };
    /*

        parent render
        child componentWillReceiveProps
        child render
    */

    shouldComponentUpdate () {
        //deciding life cycle hook on state change
        //true -- change the state
        //false - dont change state
        console.log("parent shouldComponentUpdate");
        return true;
    };

    //before unmounting the component react library will execute following life cycle hooks
    //these life cycle hooks used to perform the clean up operations
    //Ex. nullify the instance members
    //Ex. empty the state
    //Ex. empty the Props ,.....
    componentWillUpdate () {
        console.log("parent componentWillUpdate");
    }
}
```

---



```
    }  
    componentDidUpdate() {  
      console.log("parent componentDidUpdate");  
    }  
    componentWillUnmount() {  
      console.log("parent componentWillUnmount");  
    }  
  };  
};
```

### Child.js

```
import React, { Component } from "react";  
export default class Child extends React.Component {  
  constructor() {  
    super();  
    console.log("child constructor");  
  };  
  componentWillMount() {  
    console.log("child componentWillMount");  
  };  
  render() {  
    console.log("child render");  
    return (  
      <div>  
        {this.props.key1}  
      </div>  
    )  
  };  
  componentDidMount() {  
    console.log("child componentDidMount");  
  };  
};
```





```
componentWillReceiveProps() {  
    console.log("child componentWillReceiveProps");  
};  
shouldComponentUpdate() {  
    console.log("Child shouldComponentUpdate");  
    return false;  
};  
componentWillUpdate() {  
    console.log("child componentWillUpdate");  
}  
componentDidUpdate() {  
    console.log("child componentDidUpdate");  
}  
componentWillUnmount() {  
    console.log("child componentWillUnmount");  
}  
};
```

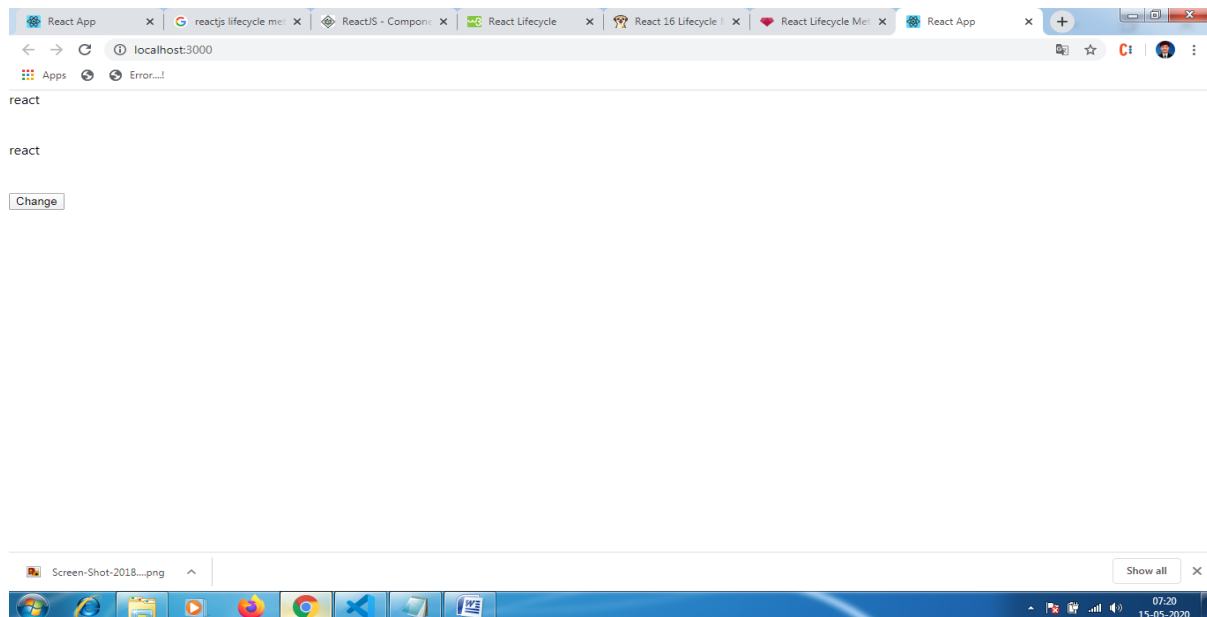
### index.js

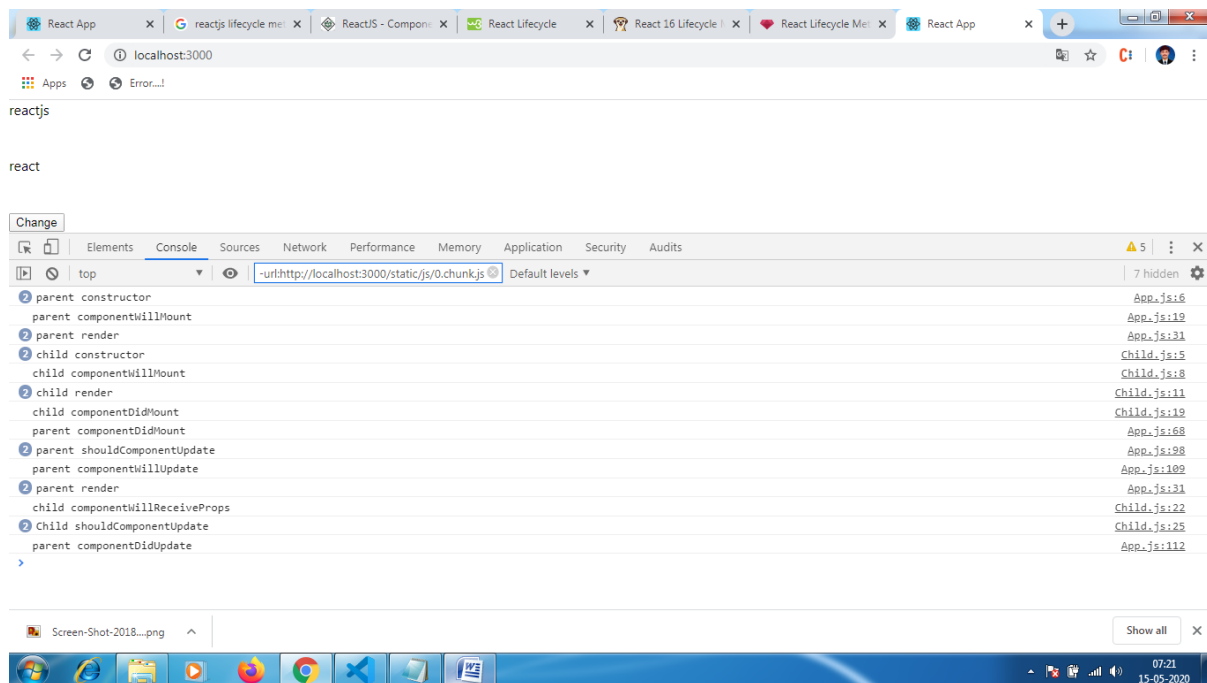
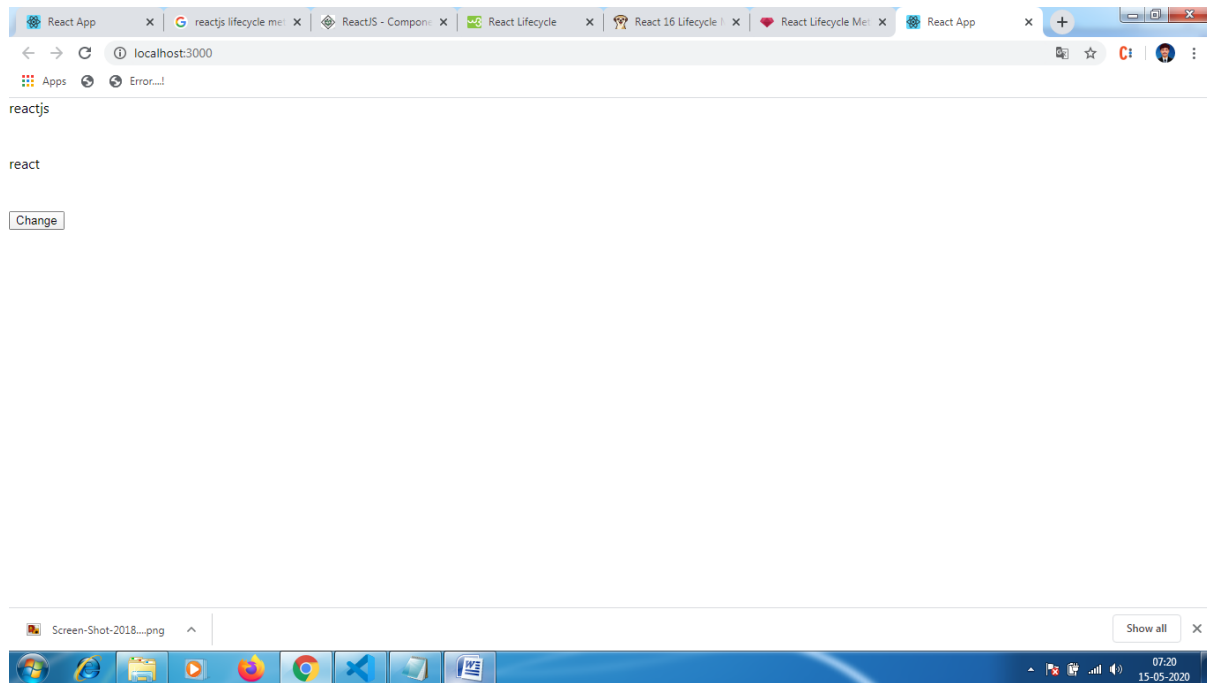
```
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import './index.css';  
  
import App from './App';  
  
import Functional from "././Functional";  
  
import Compone from "./Compone";  
  
import Demo1 from "./Demo1";  
  
import * as serviceWorker from './serviceWorker';
```



```
ReactDOM.render(  
  
  <App />,  
  
  document.getElementById('root')  
  
);  
  
// If you want your app to work offline and load faster, you can  
change  
  
// unregister() to register() below. Note this comes with some  
pitfalls.  
  
// Learn more about service workers: https://bit.ly/CRA-PWA  
  
serviceWorker.unregister();
```

### Output:





## Chapter-7 (Unit Test Cases)

- Jest is the tool, used to write the unit test cases to "react" and "vue" applications.
- we will install jest tool by using following command.
- `yarn add react-test-renderer --save`
- all the testing files have the ".test.js" extension.
- we will test unit test cases by using "yarn test" or "jest"

### Step 1:Create The React Application

```
>create-react-app test-app
```

### Step 2:Download the modules

- `yarn add react-test-renderer --save`

### Step 3:Create the target components

#### Directory Structure

```
*****
```

```
Test-app
```

```
  Src
```

```
    App.test.js
```

```
*****
```

#### App.test.js

```
import React from "react";
```

```
import App from "../App";
```



```
//render() is the predefined function

//render() function used to get the reference of Component to
testing Environment

import { render } from "@testing-library/react";

//jest starts the execution from test() function

test("check msg in App component", ()=>{

    const { getByText } = render(<App />);

    const status = getByText( /Learn React/i );

    expect(status).toBeInTheDocument();

});

test("another sample test case", ()=>{

    const { getByText } = render(<App />);

    const status = getByText(/Edit src App.js and save to
reload/i);

    expect(status).toBeInTheDocument();

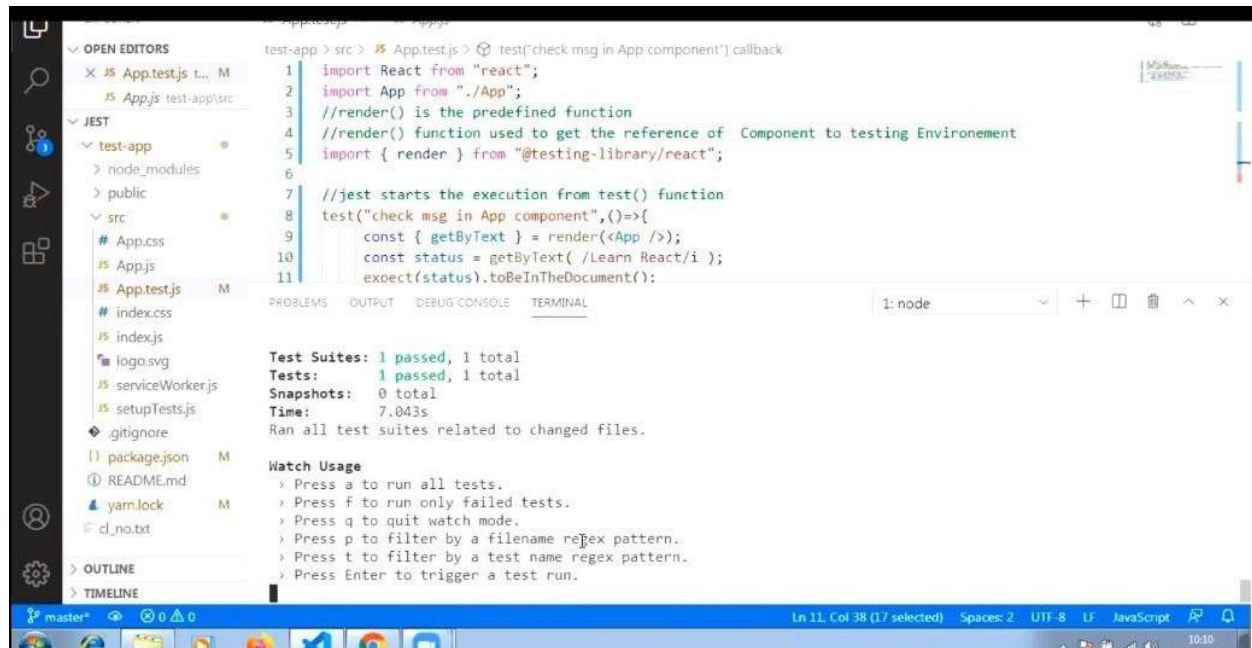
});
```

#### Step 4:Excute the applicaton

```
>yarn test
```



## Result:



The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project structure with files like `App.test.js`, `App.js`, `index.css`, `index.js`, `logo.svg`, `serviceWorker.js`, `setupTests.js`, `package.json`, `README.md`, `yarn.lock`, and `cd_no.txt`. The code editor shows the content of `App.test.js`, which includes imports for `React` and `App`, a comment about `render()`, and a `test` function that renders the `App` component and checks if the text `/Learn React/i` is present in the document. The terminal shows the output of running the tests, indicating that 1 test suite and 1 test passed. The status bar at the bottom shows the current file is `App.test.js` at line 11, column 38.

```
test-app > src > App.test.js > test("check msg in App component") callback
1  import React from "react";
2  import App from "../App";
3  //render() is the predefined function
4  //render() function used to get the reference of Component to testing Environment
5  import { render } from "@testing-library/react";
6
7  //jest starts the execution from test() function
8  test("check msg in App component", () => {
9    const { getByText } = render(<App />);
10    const status = getByText(/Learn React/i);
11    expect(status).toBeInTheDocument();
12  });
```

Test Suites: 1 passed, 1 total  
Tests: 1 passed, 1 total  
Snapshots: 0 total  
Time: 7.043s  
Ran all test suites related to changed files.

Watch Usage  
› Press a to run all tests.  
› Press f to run only failed tests.  
› Press q to quit watch mode.  
› Press p to filter by a filename regex pattern.  
› Press t to filter by a test name regex pattern.  
› Press Enter to trigger a test run.





elements exist.

## WHAT IS REDUX:

- Redux is a predictable state container for JavaScript applications. It helps to you write applications, run in different environments and easy to test.
- and simply we called as Redux is a state management tool
- It is lightweight, so we don't have to worry about it making your application's asset size bigger.
- Redux, the state of your application is kept in a store, and each component can access any state that it needs from this store.

## Redux works mainly three principles:

- Application state is stored in a single object. Redux stores state in a single JavaScript object to make it easier to map out and pass data throughout the entire application. Centralizing state in a single object also make the process of testing and debugging faster.
- Application state is immutable. In redux, states cannot be modified. The only way to change the state is to provide an action. Actions are immutable JavaScript objects that describes the state changes. Actions are executed in an order to prevent race conditions.
- Reducers specify how the action transform the state. Reducers are JavaScript functions that create a new state with the given current state and action. They centralized data mutations and can act on all or part of





the state. Reducers can also be combined and reused.

## TERMINOLOGY OF REDUX :

- working of redux is simple. There is a central store that holds the entire state of the application.
- Each component can access the stored state without having to send down props from one component to another
- Components in redux

Actions

Store

Reducers

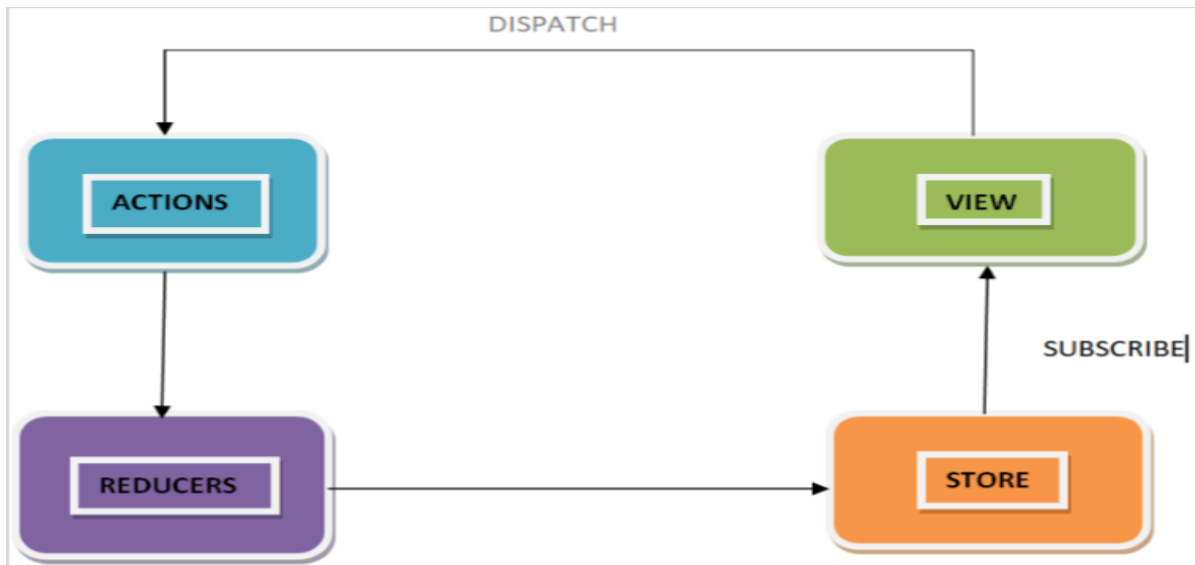
Dispatch

Subscribe

State

getState()





### ACTIONS IN REDUX:

- Simply we call that actions are events.
- They are the only way we can send data from application to Redux store. The data can be from user interactions, API calls, or even form submissions.
- Actions are sent using the `store.dispatch()` method.
- Actions are plain JavaScript objects, and they must have a `type` property to indicate the type of action to be carried out.
- They must also have a payload that contains the information that should be worked on the action.

### STORE IN REDUX:

- The store holds the application state. There is only one store in any Redux application.
- redux has one global store
- we can access the state stored, update the state, and



register or unregister listeners via helper methods.

- The central idea of a Redux app is to separate the state of the app from the app itself.
- The state of the app is stored in a store. Hence the state of the app which is in the store.
- The state stored in the store is a simple JS object with slices. As the state thus hierarchically contains other structures it also called a state tree
- Actions performed on the state always return a new state, hence state is very easy and predictable
- create redux store:

Redux comes with one predefined function that creates store, i.e. `createStore`

#### Syntax:

```
import { createStore } from 'redux';  
  
const store = createStore();
```

#### REDUCERS IN REDUX:

- Reducers are a pure function in Redux, these Pure functions are predictable.
- Reducers are the only way to change states in Redux.
- It is the only place where you can write logic and calculations.
- reducers main job is to take "current state" and an "action" and return the "new state"



- Reducer function will accept the previous state of app and action being dispatched, calculate the next state and returns the new object.
- Redux reducers work just like the function is pass to Array, the main thing is they reduce is actions.
- They reduce a set of actions (over time) into a single state.
- The difference is that with Array's reduce it happens all at once, and with Redux, it happens over the lifetime of our running app.
- The following few things should never be performed inside the reducer:
  - Mutation of functions arguments
  - API calls & routing logic
  - Calling non-pure function

The following is the syntax  
of a reducer:

```
(state,action) => newState
```

**getState in redux:**

- Returns the current state tree of your application. It is equal to the last value returned by the store's reducer.
- and `getState()` returns current state tree of your application.
- It helps you retrieve the current state of your Redux store.
- The syntax for `getState` is as follows:

```
store.getState()
```

**DISPATCH in REACT:**



- It allows to dispatch an action to change a state in your application.
- Dispatch() is the only way to trigger a state change.
- The store's reducing function will be called with the current getState() result and the given action synchronously.
- Its return value will be considered the next state.
- It will be returned from getState() from now on, and the change listeners will immediately be notified.
- Simply dispatch is the one request to the store to perform a specific action.
- dispatch() is the function that belongs to the store, which means the store not only holds the state of the app it also operates as a dispatcher that dispatches commands.
- The syntax for dispatch is as follows:

```
store.dispatch({key      :  
  'deposit'})
```

#### DISPATCH WORKING:

- we created store has a built-in function called dispatch
- then Call it with an action and Redux will call our reducer with that action and then replace the state with whatever your reducer returned.

#### SUBSCRIBE IN REACT:

- It helps to register a callback that Redux store will call when an action has been dispatched.
- As soon as the Redux state has been updated, the



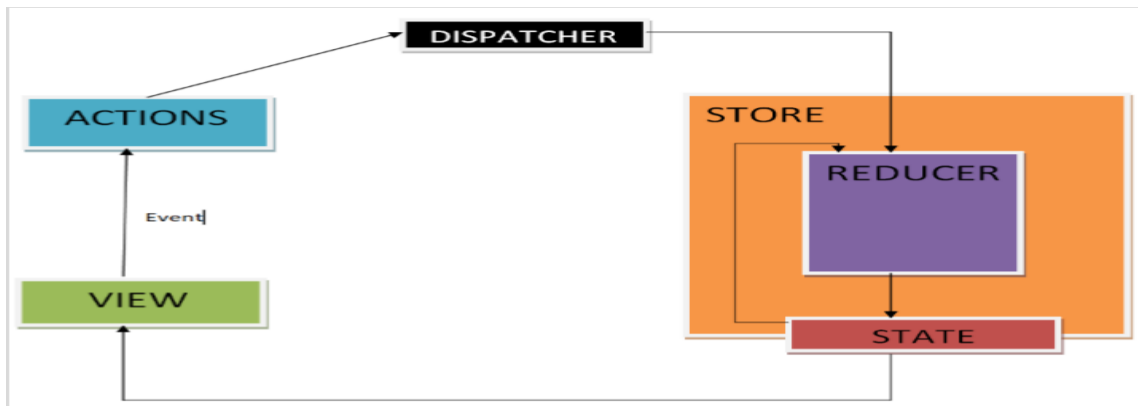
view will re-render automatically.

- Adds a change listener. It will be called any time an action is dispatched, and some part of the state tree may potentially have changed. You may then call `getState()` to read the current state tree inside the callback.
- The subscriptions are snapshotted just before every `dispatch()` call. If you subscribe or unsubscribe while the listeners are being invoked, this will not have any effect on the `dispatch()` that is currently in progress. However, the next `dispatch()` call, whether nested or not, will use a more recent snapshot of the subscription list.
- To unsubscribe the change listener, invoke the function returned by `subscribe`.

#### ARCHITECTURE/FLOW OF REDUX:

- Redux follows the unidirectional data flow. It means that application data will follow in one-way binding data flow.
- As the application grows & becomes complex, it is hard to reproduce issues and add new features if you have no control over the state of your application. -
- Redux reduces the complexity of the code, by enforcing the restriction on how and when state update can happen.





- An action is dispatched when a user interacts with the application.
- The root reducer function is called with the current state and the dispatched action. The root reducer may divide the task among smaller reducer functions, which ultimately returns a new state.
- The store notifies the view by executing their callback functions.
- The view can retrieve updated state and re-render again

## REDUX – INSTALLATION:

- Before installing Redux, we have to install Nodejs .
- Below are the instructions that will help you install it.  
You can skip these steps if you already have Nodejs installed in your device.
- Website: <https://nodejs.org/>
- You can check successful installation by opening the command prompt and type " node -v "
- This will show you the latest version of Node in your system.







## Directory Structure

\*\*\*\*\*

### ReduxEx

Demo.js

\*\*\*\*\*

### Demo.js

```
const { createStore }= require("redux");

const initialState = {

  bal : 5000

};

const reducer = (state=initialState,actions)=>{

  const newState = {...state};

  switch(actions.type){

    case "DEPOSIT":

      newState.bal+=actions.value;

      break;

    case "WITHDRAW":

      newState.bal-=actions.value;

      break;

  }

  return newState;
```



```
};

const store = createStore(reducer);

store.subscribe(()=>{

    console.log(store.getState());

});

store.dispatch({type:"DEPOSIT",value:5000});

store.dispatch({type:"DEPOSIT",value:5000});

store.dispatch({type:"DEPOSIT",value:5000});

store.dispatch({type:"WITHDRAW",value:1000});

store.dispatch({type:"DEPOSIT",value:5000});

store.dispatch({type:"WITHDRAW",value:10000});
```

Execute the above application by using following  
command :      `node demo.js`

## Chapter-9(Integration of React and Redux)

1) create the react application

```
> create-react-app redux-ex
```

2) switch to react application

```
> cd redux-ex
```

3) download the libraries



=> redux

=> react-redux

- "redux" library used to create the "redux application"
- "react-redux" used to integrate the "redux" to "react application".

> yarn add redux react-redux --save

#### 4) create the reducer

```
*****  
  
redux-ex  
  
  src  
  
    reducer  
  
      reducer.js
```

```
*****
```

5) create the "store" and make the availability to "App" component with the help of "index.js" file

#### 6) create the App component

```
*****  
  
redux-ex  
  
  src  
  
    reducer  
  
      reducer.js  
  
      App.js
```



## Index.js

\*\*\*\*\*

reducer.js

```
const initialState = {

  products : []

};

const reducer = (state=initialState,action)=>{

  switch(action.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;
```

---



App.js

```
import React,{ Component } from "react";

import { connect } from "react-redux";

class App extends React.Component{

  render() {

    return(

      <div>

        <button

onClick={this.props.getProducts}>Products</button>

        <br></br>

        <h6>{JSON.stringify(this.props.products)}</h6>

      </div>

    )  };

};

//subscribe

const receive = (state)=>{

  return{

    products : state.products

  }

};

//dispatch

const send = (dispatch)=>{
```

---



```
    return{  
      getProducts : ()=>{ dispatch({type:"PRODUCTS"}) }  
    }  
  };  
  
export default connect(receive,send) (App) ;
```

### Index.js

```
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import './index.css';  
  
import App from './App';  
  
import * as serviceWorker from './serviceWorker';  
  
//import reducer  
  
import reducer from "./reducer/reducer";  
  
//import Provider  
  
//Provider used to make the availability of "store" to App  
component  
  
import { Provider } from "react-redux";  
  
//import createStore function  
  
//createStore function used to create the store object  
  
import { createStore } from "redux";  
  
//create the store  
  
const store = createStore(reducer);
```



```
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>,  
  document.getElementById('root')  
);  
  
// If you want your app to work offline and load faster, you can  
change  
  
// unregister() to register() below. Note this comes with some  
pitfalls.  
  
serviceWorker.unregister();
```



## Chapter-10 (Multiple Reducers)

### 1) create the react application

```
> create-react-app multiple-reducers-app
```

### 2) switch to react application

```
> cd multiple-reducers-app
```

### 3) download the libraries

```
=> redux
```

```
=> react-redux
```

- "redux" library used to create the "redux application"
- "react-redux" used to integrate the "redux" to "react application".

```
> yarn add redux react-redux --save
```

### 4) create the reducers

```
*****
```

```
multiple-reducers-app
```

```
src
```

```
reducer
```

```
reducerA.js
```

```
reducerB.js
```

```
*****
```

### 5) create the "store" and make the availability to "App" component with the help of "index.js" file





## 6) create the App component

\*\*\*\*\*

multiple-reducers-app

src

reducer

reducerA.js

reducerB.js

app.js

index.js

\*\*\*\*\*

### reducerA.js

```
const initialState = {
```

```
  num1 : 1
```

```
};
```

```
const reducerA = (state=initialState,action)=>{
```

```
  switch(action.type){
```

```
    case "UPDATE_A":
```

```
      return{
```

```
        ...state,
```

```
        num1: state.num1 + action.value
```

```
      }
```



```
        break;

    }

    return state;

};

export default reducerA;
```

### reducerB.js

```
const initialState = {

    num2 : 1

};

const reducerB = (state=initialState,action)=>{

    switch(action.type){

        case "UPDATE_B":

            return{

                ...state,

                num2:state.num2 + action.value

            }

            break;

    }

    return state;

};

export default reducerB;
```



app.js

```
import React,{ Component } from "react";

import { connect } from "react-redux";

class App extends React.Component{

  render() {

    return(

      <div>

        Num1 : <span

style={{color:"red",marginRight:100}}>{this.props.num1}</span>

        Num2 : <span

style={{color:"red"}}>{this.props.num2}</span>


        <br></br><br></br><br></br>

        <button style={{marginRight:100}} onClick={ ()=>{

this.props.updateNum1(this.props.num2) } }>UPDATE_NUM1</button>

        <button onClick={ ()=>

{this.props.updateNum2(this.props.num1) } }>UPDATE_NUM2</button>

      </div>

    )

  }

};

const receive = (state)=>{

  return{
```

---



```
    num1 : state.rA.num1,

    num2 : state.rB.num2

  }

};

const send = (dispatch)=>{

  return{

    updateNum1 : (data)=>{
dispatch({type:"UPDATE_A",value:data}) },

    updateNum2 : (data)=>{
dispatch({type:"UPDATE_B",value:data}) }

  }

};

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

index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import * as serviceWorker from './serviceWorker';

import reducerA from "./reducers/reducerA";
```

---



```
import reducerB from "../reducers/reducerB";

import { createStore,combineReducers } from "redux";

import { Provider } from "react-redux";

const rootReducer = combineReducers({

  rA : reducerA,

  rB : reducerB

});

const store = createStore( rootReducer );

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

// If you want your app to work offline and load faster, you can
change

// unregister() to register() below. Note this comes with some
pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();
```



## Chapter-11 (Saga Middleware)

- Saga middleware watches the dispatches from Component.
- Saga middleware watches with the help of generators concept.
- We will download saga by using yarn tool

### 1) create the react application

```
> create-react-app redux-saga-example
```

### 2) switch to react application

```
> cd redux-saga-example
```

### 3) download the libraries

```
=> redux
```

```
=> react-redux
```

- "redux" library used to create the "redux application"
- "react-redux" used to integrate the "redux" to "react application".

```
> yarn add redux-saga redux react-redux -save
```

### 4) Create The saga(Action.js) file

```
*****
```

```
redux-saga-example
```

```
src
```

```
saga
```

```
saga.js
```

```
*****
```



### saga.js

```
import { delay } from "redux-saga";
import { takeLatest, put, takeEvery } from "redux-saga/effects";

function* ageUpAsync() {
  yield delay(4000);
  yield put({ type: "AGE_UP_ASYNC", value: 1 });
}

function* ageDownAsync() {
  yield delay(2000);
  yield put({ type: "AGE_DOWN_ASYNC", value: 1 });
};

export function* watchAgeUp() {
  yield takeEvery("AGE_UP", ageUpAsync);
  yield takeEvery("AGE_DOWN", ageDownAsync);
}
```

### 5) Create The reducer(reducer.js) file

\*\*\*\*\*

redux-saga-example

src

store

reducer.js

\*\*\*\*\*



### reducer.js

```
reducer.js
const initialState = {
  age: 20
};

const reducer = (state = initialState, action) => {
  const newState = { ...state };

  switch (action.type) {
    case "AGE_UP_ASYNC":
      newState.age += action.value;
      break;

    case "AGE_DOWN_ASYNC":
      newState.age -= action.value;
      break;
  }
  return newState;
};

export default reducer;
```

## 6) Create The Components file

\*\*\*\*\*

redux-saga-example

src





app.js

index.js

\*\*\*\*\*

### Index.js

```
import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

import App from "./App";

import reducer from "./store/reducer";


import { Provider } from "react-redux";

import { createStore, applyMiddleware } from "redux";

import createSagaMiddleware from "redux-saga";

import { watchAgeUp } from "./sagas/saga";

const sagaMiddleware = createSagaMiddleware();

const store = createStore(reducer,
  applyMiddleware(sagaMiddleware));

sagaMiddleware.run(watchAgeUp);

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,
```



```
document.getElementById("root")  
  
);
```

### app.js

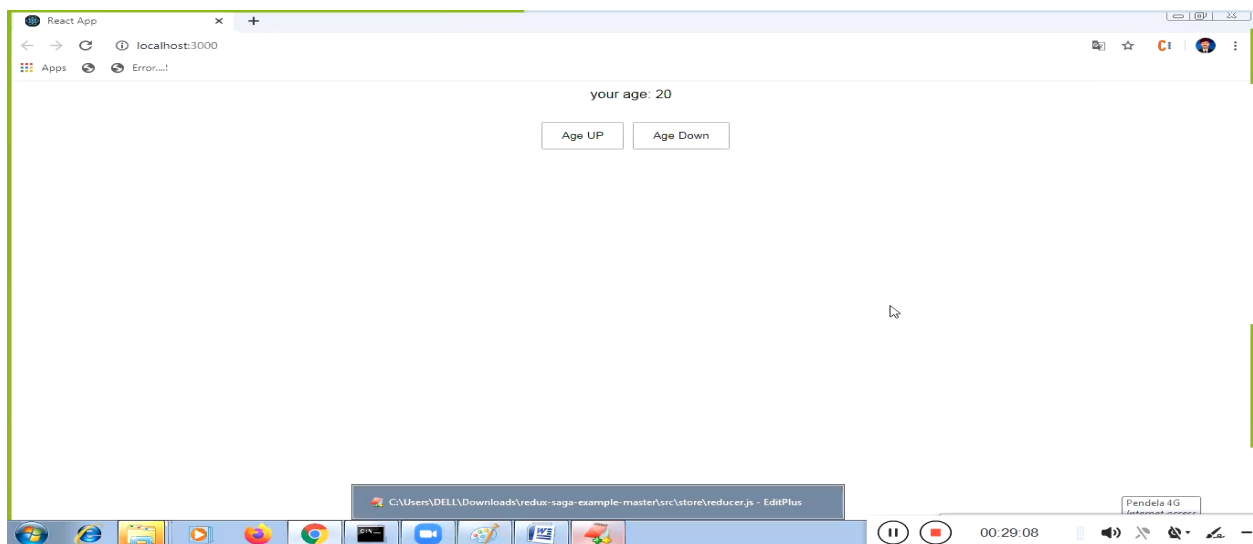
```
import React, { Component } from "react";  
import "./App.css";  
import { connect } from "react-redux";  
  
class App extends Component {  
  render() {  
    return (  
      <div className="App">  
        <div className="Age-label">  
          your age: <span>{this.props.age}</span>  
        </div>  
        <button onClick={this.props.onAgeUp}>Age UP</button>  
        <button onClick={this.props.onAgeDown}>Age Down</button>  
      </div>  
    );  
  }  
}  
  
const mapStateToProps = state => {  
  return {  
    age: state.age  
  };  
};  
  
const mapDispatchToProps = dispatch => {
```

---



```
return {  
  onAgeUp: () => dispatch({ type: "AGE_UP", value: 1 }),  
  onAgeDown: () => dispatch({ type: "AGE_DOWN", value: 1 })  
};  
};  
export default connect(  
  mapStateToProps,  
  mapDispatchToProps  
) (App);
```

OutPut:



## Chapter-12 (thunk Middleware)

### 1) create the react application

```
> create-react-app redux-thunk-example
```

### 2) switch to react application

```
> cd redux-thunk-example
```

### 3) download the libraries

- "redux" library used to create the "redux application"
- "react-redux" used to integrate the "redux" to "react application".

```
> yarn add redux-thunk redux react-redux -save
```

### Directory Structure

```
*****
```

```
redux-thunk-example
```

```
  src
```

```
    actions
```

```
      actions.js
```

```
    reducer
```

```
      reducer.js
```

```
    app.js
```

```
    index.js
```

```
*****
```



#### 4) Create The reducer(reducer.js) file

\*\*\*\*\*

redux-thunk-example

src

reducer

reducer.js

\*\*\*\*\*

reducer.js

```
    const initialState = {  
  
    bal : 5000  
  
};  
  
const reducer = (state=initialState,action)=>{  
  
    const newState = {...state};  
  
    switch(action.type){  
  
        case "WITHDRAW":  
  
            newState.bal-=action.value;  
  
            break;  
  
        case "DEPOSIT":  
  
            newState.bal+=action.value;  
  
            break;
```



```
    }  
  
    return newState;  
};  
  
export default reducer;
```

## 5) Create The thunk(Actions.js) file

\*\*\*\*\*

redux-thunk-example

src

actions

actions.js

\*\*\*\*\*

actions.js

```
export const onWithdraw = (val)=>{  
    return {type:"WITHDRAW",value:val};  
};
```

```
export const onDepositAsync = (val)=>{  
    return {type:"DEPOSIT",value:val};  
};
```

```
export const onDeposit = (val)=>{
```



```
    return (dispatch)=>{  
      setTimeout(()=>{  
        dispatch(onDepositAsync(val));  
      },5000);  
    }  
  };  
};
```

6)create the "store" and make the availability to "App" component

\*\*\*\*\*

redux-thunk-example

src

index.js

\*\*\*\*\*

index.js

```
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import './index.css';  
  
import App from './App';  
  
import * as serviceWorker from './serviceWorker';
```



```
import reducer from "../reducer/reducer";

import { Provider } from "react-redux";

import { createStore, applyMiddleware } from "redux";

import thunk from "redux-thunk";

const store = createStore(reducer, applyMiddleware(thunk));

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

// If you want your app to work offline and load faster, you can
// change

// unregister() to register() below. Note this comes with some p
itfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();
```

7) create the "store" and make the availability to "App" component

---





\*\*\*\*\*

redux-thunk-example

src

app.js

\*\*\*\*\*

App.js

```
import React, { Component } from "react";

import { connect } from "react-redux";

import * as actions from "../actions/actions";

import "../App.css";

class App extends React.Component{

  render() {

    return (

      <div className="App">

        <h1>Balance:<span style={{color:"green"}}>{this.props.bal}</span></h1>

        <button onClick={this.props.withdraw}><b>Withdraw</b></button>

        <button onClick={this.props.deposit}><b>Deposit</b></button>

      </div>

    )

  }

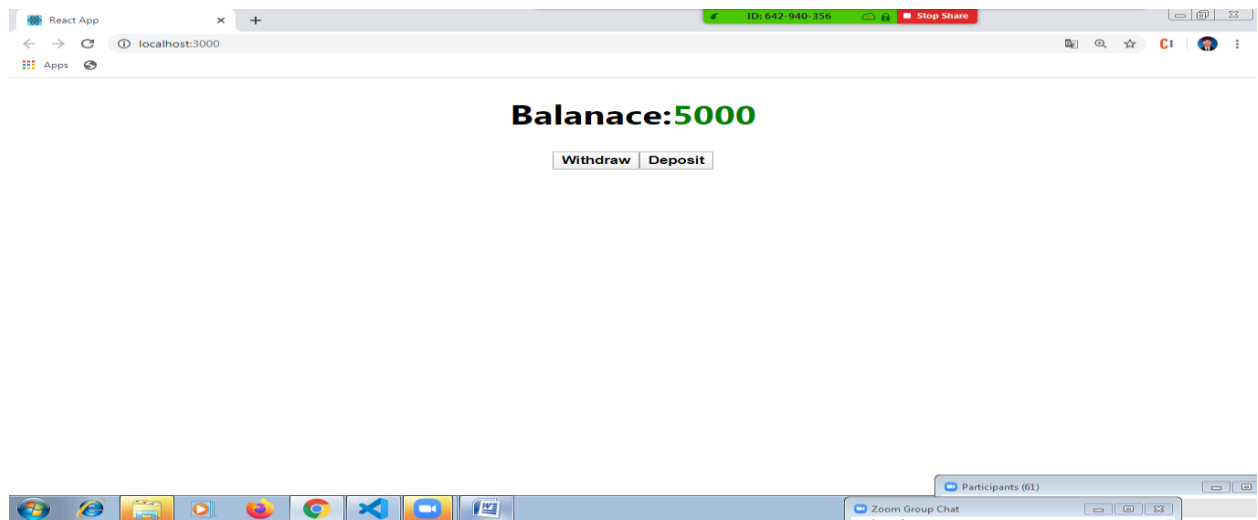
}
```



```
    }  
  }  
  
  const receive = (state)=>{  
    return{  
      bal : state.bal  
    }  
  };  
  
  const send = (dispatch)=>{  
    return{  
      withdraw : ()=>{ dispatch(actions.onWithdraw(1000)) },  
      deposit : ()=>{ dispatch(actions.onDeposit(5000)) }  
    }  
  };  
  
  export default connect(receive,send) (App) ;
```

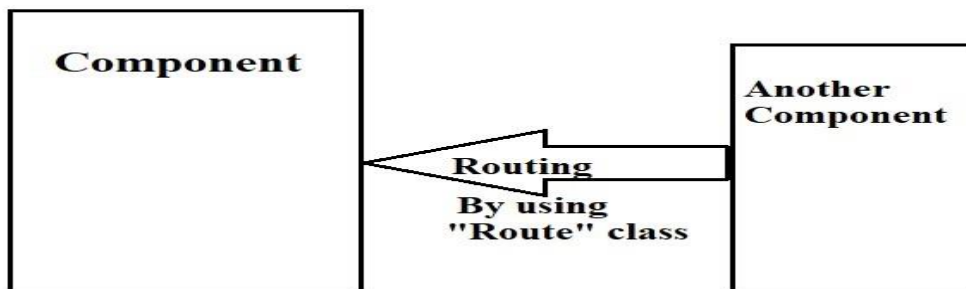
OutPut:





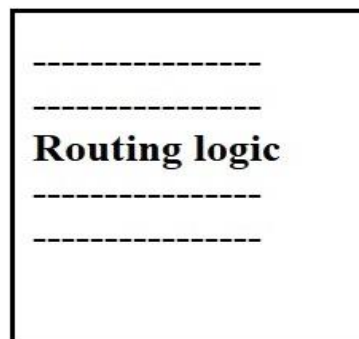
## Chapter-13(Single Page Applications)

- ✓ Loading one component to Another Component without Refreshing the whole webpage called as "Single Page Application".
- ✓ Navigating from one component to another component in Single Page Application called as "Routing".



- ✓ "Route" is the predefined class , used to implement the "Routing" in Single Page Application.
- ✓ "BrowserRouter" is the predefined class ,used to hold/Space the Routing Logic

### Browser Router



- ✓ "NavLink" is the predefined class, used to define the "Router Links".



- ✓ We will get the support of Single Page Application by using "react-router-dom" module.
- ✓ We will download above module by using "yarn" tool.

```
>yarn add react-router-dom --save
```

For example :

Step 1. Create the react application

```
> create-react-app spa-app
```

Step 2. Switch to react application

```
>cd spa-app
```

Step 3. Download "react-router-dom" library

```
>yarn add react-router-dom --save
```

Step 4. Create the target components

Directory Struture

\*\*\*\*\*

Spa-app

Src

About.js

Home.js

Contact.js

App.js

Index.js



\*\*\*\*\*

## About.js

```
JS About.js spa-app\src JS Home.js JS Contact.js JS App.js JS About.js D:\...\apps\... X
C:\> Telegram > React Js 7_30PM > 03rdApr@7.30PM(ReactJS) > apps > spa-app > src > JS About.js > ...
1  import React,{ Component } from "react";
2  export default class About extends React.Component{
3      render(){
4          return(
5              <div>
6                  <h1 style={{color:"red"}}>About Page !!!</h1>
7              </div>
8          )
9      }
10 };
```

## Home.js

```
JS About.js JS Home.js X JS Contact.js JS App.js
spa-app > src > JS Home.js > ...
1  import React,{ Component } from "react";
2  export default class Home extends React.Component{
3      render(){
4          return(
5              <div>
6                  <h1 style={{color:"green"}}>Home Page !!!</h1>
7              </div>
8          )
9      }
10 };
```

## Contact.js



```
JS About.js JS Home.js JS Contact.js X JS App.js
spa-app > src > JS Contact.js > ...
1  import React,{ Component } from "react";
2  export default class Contact extends React.Component{
3      render(){
4          return(
5              <div>
6                  <h1 style={{color:"blue"}}>Contact Page !!!</h1>
7              </div>
8          )
9      }
10 };
```

## App.js

```
import React,{ Component } from "react";

//import BrowserRouter

//BrowserRouter helps to write the Single Page Applications
logic.

//import NavLink

//NavLink helps to create the hyperlinks (router links) in
single page applications

import { BrowserRouter as Router, NavLink } from "react-router-
dom";

//import Route

//Route is the predefined class used to implement the Routing

import Route from "react-router-dom/Route";

import About from "../About";

import Home from "../Home";
```



```
import Contact from "../Contact";

export default class App extends React.Component{

  render() {

    return(

      <Router>

        <div>

          <NavLink to="/"

            exact strict

            activeStyle={{color:"green"}}

            style={{marginRight:100}}><b>About</b></NavLink>

          <NavLink to="/home"

            exact strict

            activeStyle={{color:"green"}}

            style={{marginRight:100}}><b>Home</b></NavLink>

          <NavLink to="/contact"

            exact strict

            activeStyle={{color:"green"}}

            style={{marginRight:100}}><b>Contact</b></NavLink>

          <br></br><br></br>

          <Route path="/" component={About} exact strict></Route>
```





```
<Route path="/home" component={Home} exact strict></Route>

<Route path="/contact" component={Contact} exact strict></Route>

</div>

</Router>

    )

  }

};
```

### Index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import * as serviceWorker from './serviceWorker';

ReactDOM.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>,

  document.getElementById('root')

);
```



// If you want your app to work offline and load faster, you can change

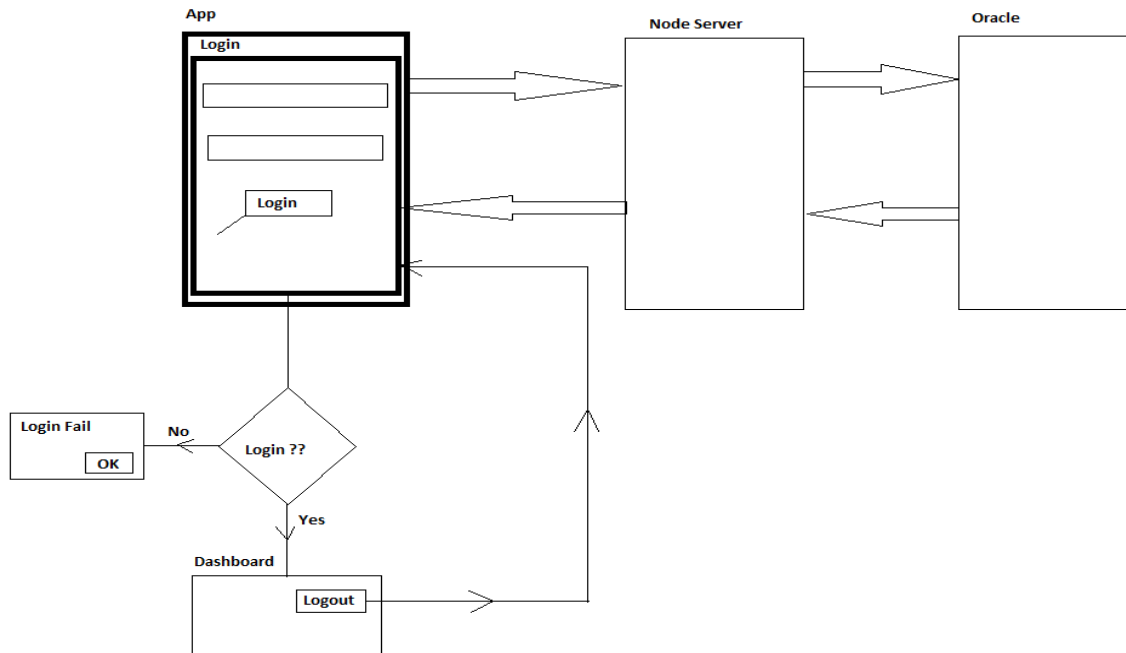
// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: <https://bit.ly/CRA-PWA>

serviceWorker.unregister();



## Navigation and Routing Parameters in Single Page Applications



### Step 1.

Create the table in oracle database.

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\DELL>sqlplus

SQL*Plus: Release 18.0.0.0.0 - Production on Wed May 13 08:18:32 2020
Version 18.4.0.0.0
Copyright (c) 1982, 2018, oracle. All rights reserved.
Enter user-name: system
Enter password:
Last successful login time: Wed May 13 2020 05:48:51 +05:30

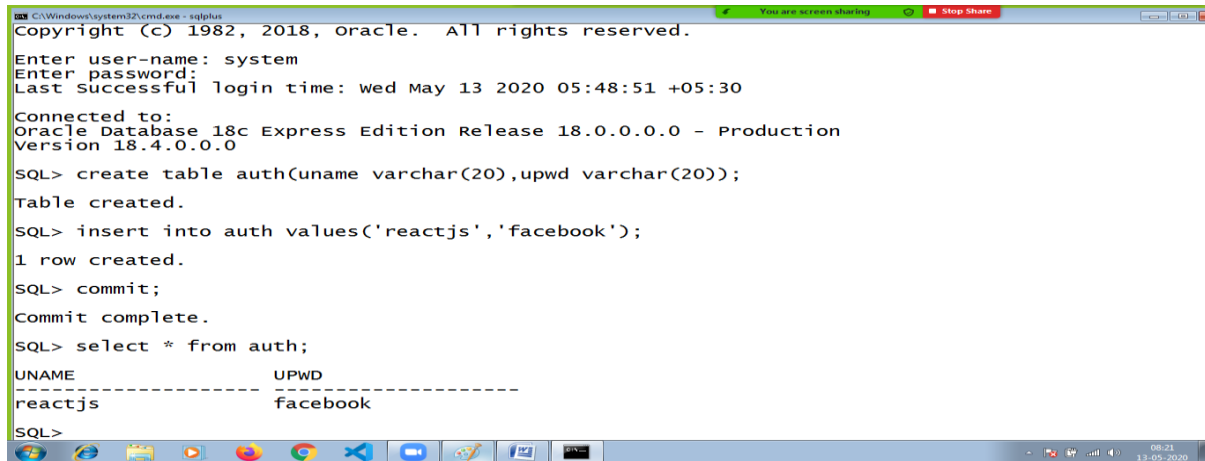
Connected to:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0

SQL> create table auth(uname varchar(20),upwd varchar(20));
Table created.

SQL> insert into auth values('reactjs','facebook');
1 row created.

SQL>
```





```
C:\Windows\system32\cmd.exe - sqlplus
Copyright (c) 1982, 2018, Oracle. All rights reserved.
Enter user-name: system
Enter password:
Last Successful login time: Wed May 13 2020 05:48:51 +05:30

Connected to:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0

SQL> create table auth(uname varchar(20),upwd varchar(20));
Table created.

SQL> insert into auth values('reactjs','facebook');
1 row created.

SQL> commit;
Commit complete.

SQL> select * from auth;

UNAME                UPWD
-----
reactjs              facebook

SQL>
```

\*\*\*\*\*

```
user           :    system

password       :    root

connectString  :    localhost/XE

table          :    auth

columns        :    uname , upwd
```

\*\*\*\*\*

step 2.

create the react application

➤ Create-react-app parameters-app

Step 3.

Switch to react application



```
➤ cd      parameters-app
```

## Step 4.

### Download node modules

- 1) express
- 2) oracledb
- 3) body-parser
- 4) cors

- express module used to develop the rest apis
- oracledb module used to connect to oracle database
- body-parser module used to read the client data
- cors module used to enable the "cors policy"

we will download above modules by using "yarn" tool

```
➤ yarn add express oracledb body-parser cors --save
```

## Step 5.

### Implement the node server

```
*****
```

```
parameters-app
```

```
server
```

```
server.js
```



\*\*\*\*\*

### server.js

```
//import the modules
```

```
//require() is the predefined function, used to import the modules
```

```
let express = require("express");
```

```
let oracledb = require("oracledb");
```

```
let cors = require("cors");
```

```
let bodyparser = require("body-parser");
```

```
//create the rest object
```

```
let app = express();
```

```
//enable cors policy
```

```
app.use(cors());
```

```
//set the json as MIME Type
```

```
app.use(bodyparser.json());
```

```
//read the client data
```

```
app.use(bodyparser.urlencoded({extended:false}));
```

```
//create the post request
```



```
app.post("/login", (req, res) => {  
    oracledb.getConnection({user: "system",  
        password: "root",  
        connectString: "localhost/XE"}, (err, connection) => {  
        if (err) throw err;  
        else {  
            connection.execute(`select * from auth where uname='`  
                `${req.body.uname}` and upwd='`${req.body.upwd}`'`, (err, records) => {  
                if (err) throw err;  
                else {  
                    if (records.rows.length > 0) {  
                        res.send({ "login": "success" });  
                    } else {  
                        res.send({ "login": "fail" });  
                    }  
                }  
            });  
        }  
    });  
});
```



```
//assign the port no
```

```
app.listen(8080);
```

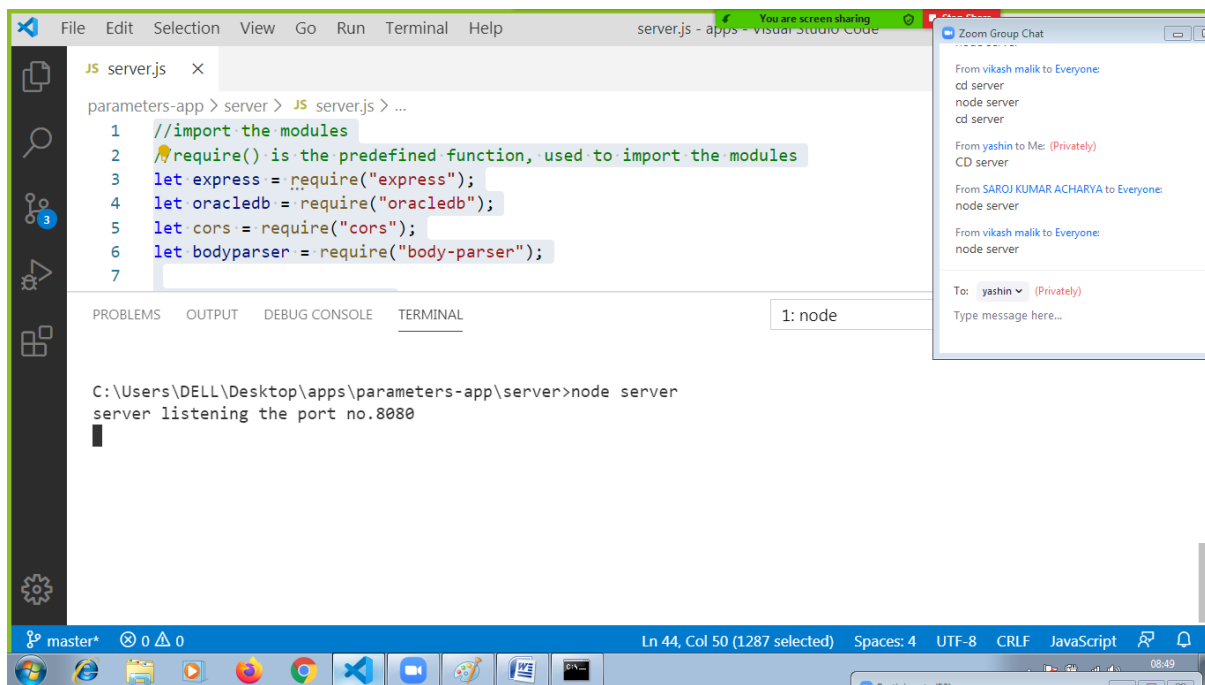
```
console.log("server listening the port no.8080");
```

## Step 6.

### Start the server

➤ `cd parameters-app/server`

➤ `node server`

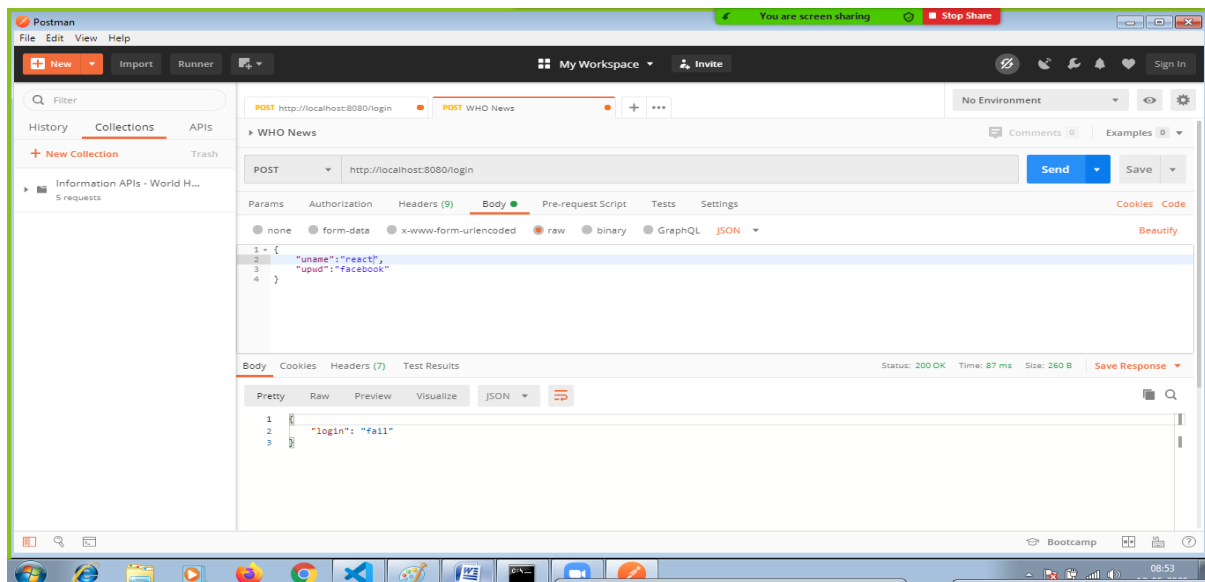
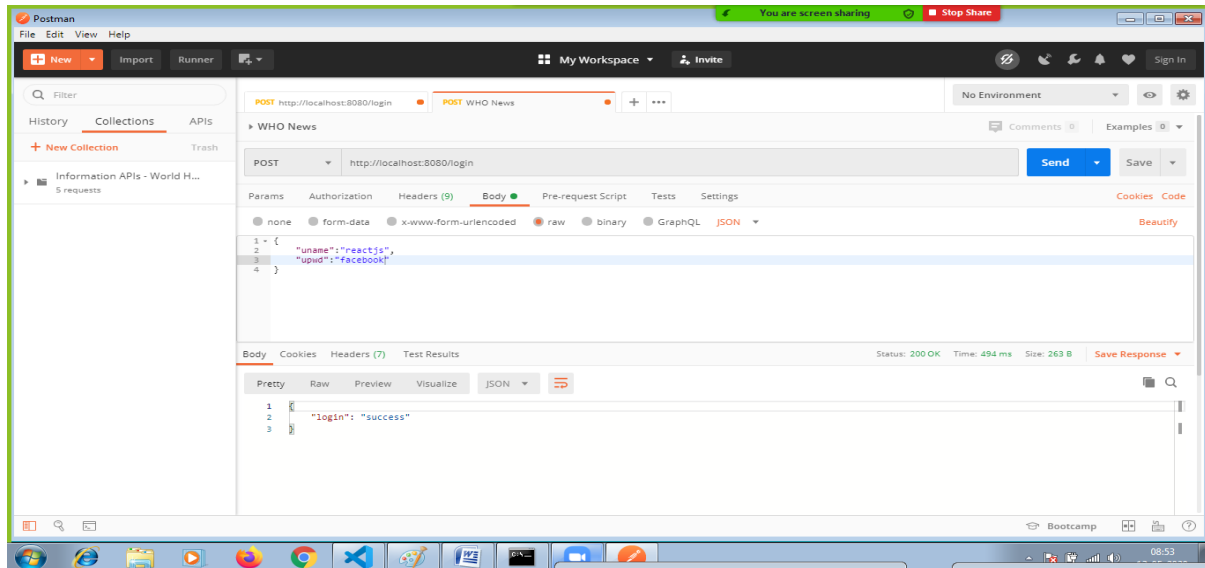




## Step 7.

Test the rest api by using Postman

⇒ <http://localhost:8080/login> (POST)



## Step 8.

download axios & react-router-dom module

➤ yarn add axios & react-router-dom --save

## Step 9.

Implement the single page application

\*\*\*\*\*

Parameters-app

Src

App.js

Login.js

Dashboard.js

\*\*\*\*\*

### Login.js

```
import React,{ Component } from "react";
```

```
import axios from "axios";
```

```
export default class Login extends React.Component{
```

```
  constructor(){
```

```
    super();
```

```
    this.state = {
```

```
      msg : "welcome to dashboard module"
```



```
    };  
  }  
  
  render() {  
    return(  
      <div>  
        <fieldset>  
          <legend>Login</legend>  
          <input type="text" ref="uname" placeholder="User Name"/>  
  
          <br></br><br></br>  
          <input type="password" ref="upwd" placeholder="User Password"/>  
  
          <br></br><br></br>  
          <button onClick={this.login}>Login</button>  
        </fieldset>  
      </div>  
    )  
  };  
  
  login = ()=>
```



```
    let var1 = "welcome to dashboard module";

    axios.post("http://localhost:8080/login", {"uname":this.refs.uname.value,

                                                                 "upwd":this.refs.upwd.value})

    .then((posRes)=>{

        if(posRes.data.login === "success"){

            this.props.history.push(`/dashboard/${this.state.msg}`);

        }else{

            alert("Login Fail");

        }

    }, (errRes)=>{

        console.log(errRes);

    });

};

};
```

### Dashboard.js

```
import React,{ Component } from "react";

export default class Dashboard extends React.Component{

    render(){

        return(
```



```
        <div>

            <h1 style={{color:"red"}}>{this.props.match.params.param1}</h1>

        </div>

    )

};

};
```

### App.js

```
import React,{ Component } from "react";

import { BrowserRouter as Router } from "react-router-dom";

import Route from "react-router-dom/Route";

import Login from "./Login";

import Dashboard from "./Dashboard";

export default class App extends React.Component{

    render() {

        return(

            <div>

                <Router>

                    <Route path="/" exact strict component={Login}></Route>

                    <Route path="/dashboard/:param1" exact strict component={Dashboard}></Route>

                </Router>

            </div>

        )

    }

}
```



```
        </div>
      )
    }
  };
```

## Step 10.

Execute the react project

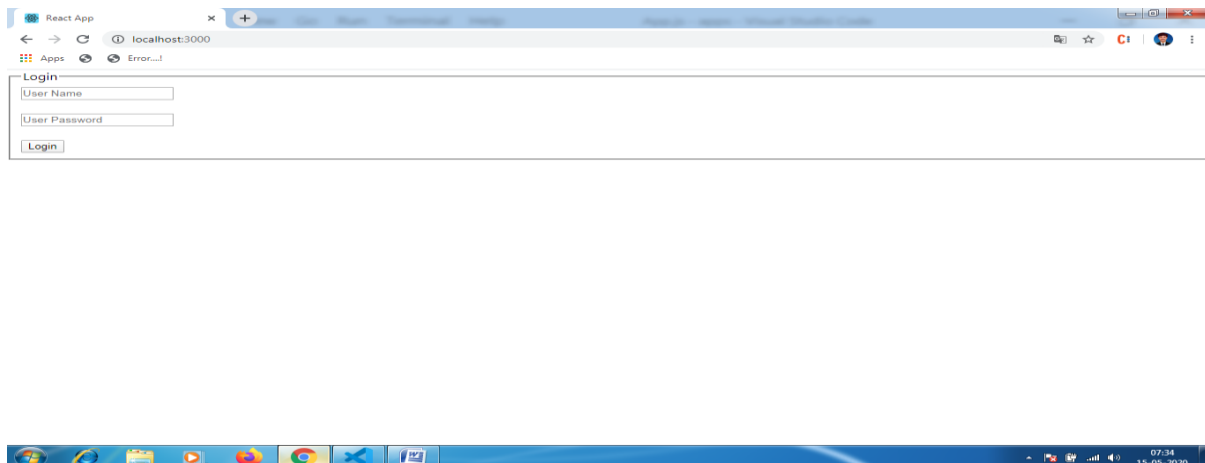
Terminal-1

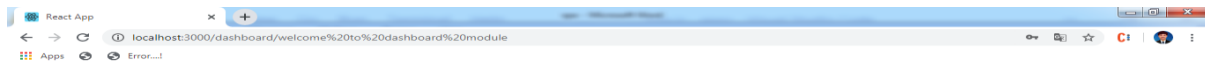
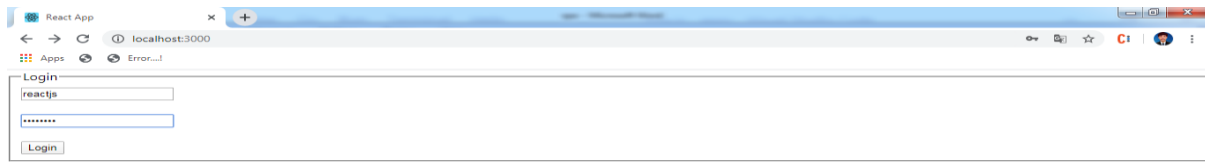
- `cd parameters-app/server`
- `node server`

Terminal-2

- `cd parameters-app`
- `yarn start`

## Output:





**welcome to dashboard module**



### Lazy loading:

Step 1. Create the react application

```
> create-react-app lazy-app
```

Step 2. Switch to react application

```
>cd Lazy-app
```

Step 3. Download "react-router-dom" library

```
>yarn add react-router-dom --save
```

Step 4. Create the target components

### Directory Struture

```
*****
```

```
Lazy-app
```

```
  Src
```

```
    Components
```

```
      Mycomp.js
```

```
      App.js
```

```
      Index.js
```

```
*****
```

### Mycomp.js

```
import React from "react";
```





```
export default () => {  
  return <div>Hi there I am now loaded!</div>;  
};
```

### App.js

```
import React, { Component, lazy, Suspense } from "react";  
  
import "./App.css";  
  
//import MyComp from './components/myComp';  
  
const MyComp = lazy(() => import("./components/MyComp"));  
  
class App extends Component {  
  render() {  
    return (  
      <div className="App">  
        <header className="App-header">  
          <div>another component</div>  
          <Suspense fallback={<div>Loading.....</div>}>  
            <MyComp />  
          </Suspense>  
        </header>  
      </div>
```



```
    );  
  }  
}
```

```
export default App;
```

### Index.js

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import './index.css';  
import App from './App';  
import * as serviceWorker from './serviceWorker';
```

```
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  document.getElementById('root')  
) ;
```

```
// If you want your app to work offline and load faster, you can  
change
```

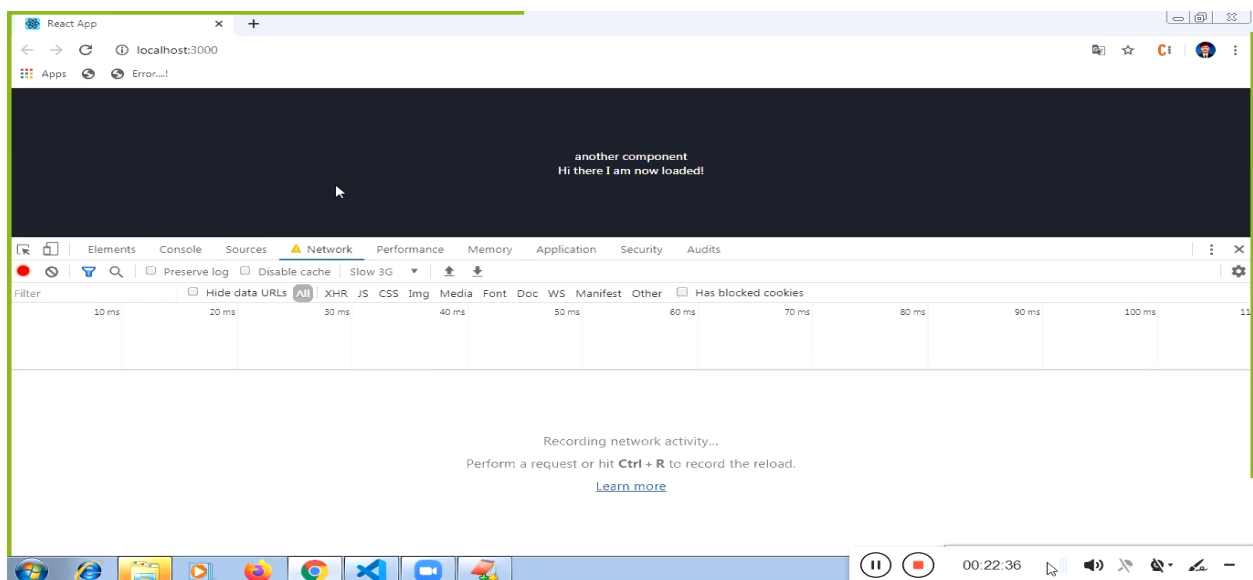
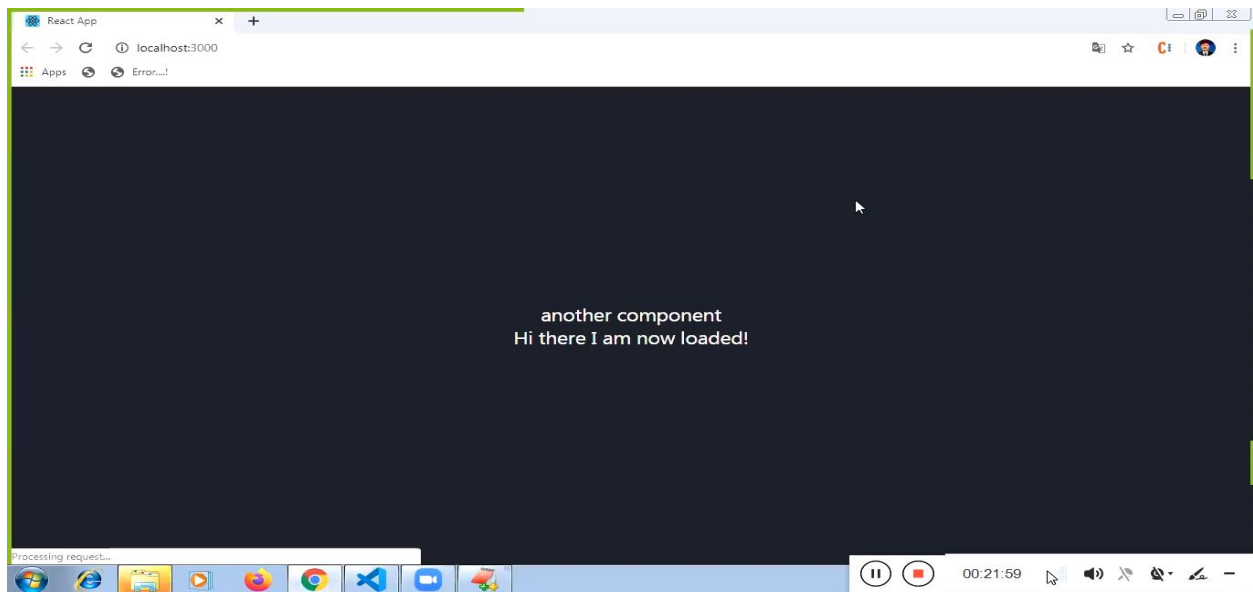


```
// unregister() to register() below. Note this comes with some  
pitfalls.
```

```
// Learn more about service workers: https://bit.ly/CRA-PWA
```

```
serviceWorker.unregister();
```

Output:



## Chapter-14 (Crud Operations)

### CRUD Operations on MySQL DataBase

-----

1) install MySQL DataBase.

Ex.

mysql-essential-5.2.3-falcon-alpha-win32.msi

2) create the table in MySQL DataBase.

### SQL Queries

-----

Default Password : root

> create schema reactjs;

- automatically "reactjs" DataBase will create.

> use reactjs;

- we can switch to reactjs DataBase.

> create table employee(eno integer,ename varchar(50),esal integer);

- automatically "employee" table will create.

> insert into employee values(111,'e\_one',10000);

- record will be inserted into employee table.

> select \* from employee;



- we can fetch the data from employee table.

\*\*\*\*\*

host : localhost

user : root

Password: root

database: reactjs

table : employee

\*\*\*\*\*

### 3) create the react application

```
> create-react-app crud-app
```

- where "crud-app" is the react application.

### 4) switch to react application

```
> cd crud-app
```

### 5) download the node modules, in order to implement server application

```
=> express
```



```
=> cors
=> mysql
=> body-parser
```

- "express" module used to develop the rest apis.
- "cors" module used to enable the ports communication.
- "mysql" module used to connect to mysql database.
- "body-parser" modules used to read the client data.

```
> yarn add express cors mysql body-parser --save
```

## 6) develop rest apis by using nodejs

```
*****
crud-app
  server
    server.js
*****
```

## 7) switch to server directory

```
> cd crud-app/server

> node server
```

## 8) Test the following rest apis by using Postman

```
=> http://localhost:8080/fetch    (GET)
```



=> http://localhost:8080/insert (POST)

=> http://localhost:8080/update (PUT)

=> http://localhost:8080/delete (DELETE)

## 9) download axios module

- "axios" module used to connect to server asynchronously.

Terminal-2

-----

> cd crud-app

> yarn add axios --save

## 10) create the Components

\*\*\*\*\*

crud-app

server

server.js

src

Fetch.js



Insert.js

Update.js

Delete.js

\*\*\*\*\*

Server.js

\*\*\*\*\*

```
//import modules
//require() function used to import the modules
let express = require("express");
let mysql = require("mysql");
let cors = require("cors");
let bodyparser = require("body-parser");

//create the master object
let app = express();
//where "app" object, used to develop the rest apis

//set the JSON as MIME Type
app.use(bodyparser.json());

//read the client data
app.use(bodyparser.urlencoded({extended:false}));

//enable ports communication
app.use(cors());

//create connection object
```





```
let connection = mysql.createConnection({
    host:"localhost",
    user:"root",
    password:"root",
    database:"reactjs"

});

//create the get request
app.get("/fetch",(req,res)=>{
    connection.query(`select * from employee`,
        (err,records,fields)=>{
            if(err) throw err;
            else{
                res.send(records);
            }
        });
});

//create the post request
app.post("/insert",(req,res)=>{
    connection.query(`insert into employee values(${req.body.eid},'${req.body.ename}',${req.body.esal})`,(err,result)=>{
        if(err) throw err;
        else{
            res.send({insert:"success"});
        }
    });
});
```



```
//create the put request
app.put("/update",(req,res)=>{
    connection.query(`update employee set ename='${req.body.ename}',es
al=${req.body.esal} where eno=${req.body.eid}`,(err,result)=>{
        if(err) throw err;
        else{
            res.send({"update":"success"});
        }
    });
});

//create the delete request
app.delete("/delete",(req,res)=>{
    connection.query(`delete from employee where eno=${req.body.eid}`,
(err,result)=>{
        if(err) throw err;
        else{
            res.send({"delet":"success"});
        }
    });
});

//assign the port no
app.listen(8080);
console.log("server listening the port no.8080");
```



**Fetch.js**

\*\*\*\*\*

```
import React, { Component } from "react";
import axios from "axios";
import Insert from "./Insert";
import Update from "./Update";
import Delete from "./Delete";
export default class Fetch extends React.Component{
  constructor(){
    super();
    this.state={
      records : []
    };
  };
  //componentDidMount() function used to make the rest api calls(GET
  Request)
  componentDidMount(){
    axios.get("http://localhost:8080/fetch")
      .then((posRes)=>{
        this.setState({
          records:posRes.data
        })
      },(errRes)=>{
        console.log(errRes);
      });
  };
  render(){
    return(
```



```
<div>
  <table border="1"
    cellPadding="10px"
    cellSpacing="10px"
    align="center">
    <thead
      style={{backgroundColor:"grey"}}>
      <tr>
        <th>SNO</th>
        <th>ENO</th>
        <th>ENAME</th>
        <th>ESAL</th>
      </tr>
    </thead>
    <tbody>
      {this.state.records.map((element,index)=>(
        <tr>
          <td>{index+1}</td>
          <td>{element.eno}</td>
          <td>{element.ename}</td>
          <td>{element.esal}</td>
        </tr>
      ))}
    </tbody>
  </table>

  <hr style={{height:5,backgroundColor:"grey"}}></hr>
  <br></br><br></br>
```



```
        <Insert />

        <hr style={{height:5,backgroundColor:"grey"}}></hr>
        <br></br><br></br>

        <Update />

        <hr style={{height:5,backgroundColor:"grey"}}></hr>
        <br></br><br></br>

        <Delete />

    </div>
  )
};
};
```

Insert.js

\*\*\*\*\*

```
import React,{ Component } from "react";
import axios from "axios";
export default class Insert extends React.Component{

  constructor(){

    super();

    this.state={
```



```
        status:""

    };

};

render(){

    return(

        <div>

            <fieldset>

                <legend>Insert</legend>

                <input type="number"

                    placeholder="enter employee number"

                    ref="eno"/>

                <br></br><br></br>

                <input type="text"

                    placeholder="enter employee name"

                    ref="ename"/>

                <br></br><br></br>

                <input type="number"

                    placeholder="enter employee sal"

                    ref="esal"/>

            </div>

        </div>

    );

}
```



```
        <br></br><br></br>

        <button onClick={this.insert}>Insert</button>

        <br></br>

        <h1>{this.state.status}</h1>

    </fieldset>

</div>

)

};

insert = ()=>{

    let record = {

        "eid":this.refs.eno.value,

        "ename":this.refs.ename.value,

        "esal":this.refs.esal.value

    };

    axios.post("http://localhost:8080/insert",

                                                record)

        .then((posRes)=>{

            this.setState({

                status:posRes.data.insert

            })

        })

    }

}
```



```
        })  
        },(errRes)=>{  
            console.log(errRes);  
        });  
    };  
};
```

Update.js

\*\*\*\*\*

```
import React,{ Component } from "react";  
import axios from "axios";  
export default class Update extends React.Component{  
    constructor(){  
        super();  
        this.state={  
            status : ""  
        };  
    };  
    render(){  
        return(  
            <div>
```





```
        <fieldset>

            <legend>Update</legend>

            <input type="number"

                ref="eid"

                placeholder="employee id"/>

            <br></br><br></br>

            <input type="text"

                ref="ename"

                placeholder="employee name"/>

            <br></br><br></br>

            <input type="number"

                ref="esal"

                placeholder="Employee Sal"/>

            <br></br><br></br>

            <button onClick={this.update}>Update</button>

            <h1>{this.state.status}</h1>

        </fieldset>

    </div>

    )

    };

    update = ()=>{
```



```
    let record = {
      "eid":this.refs.eid.value,
      "ename":this.refs.ename.value,
      "esal":this.refs.esal.value
    };

    axios.put("http://localhost:8080/update",record)

      .then((posRes)=>{
        this.setState({
          status :posRes.data.update
        })
      },(errRes)=>{
        console.log(errRes);
      });
  };
};
```

Delete.js

\*\*\*\*\*

```
import React,{ Component } from "react";
import axios from "axios";
export default class Delete extends React.Component{
  constructor(){
```



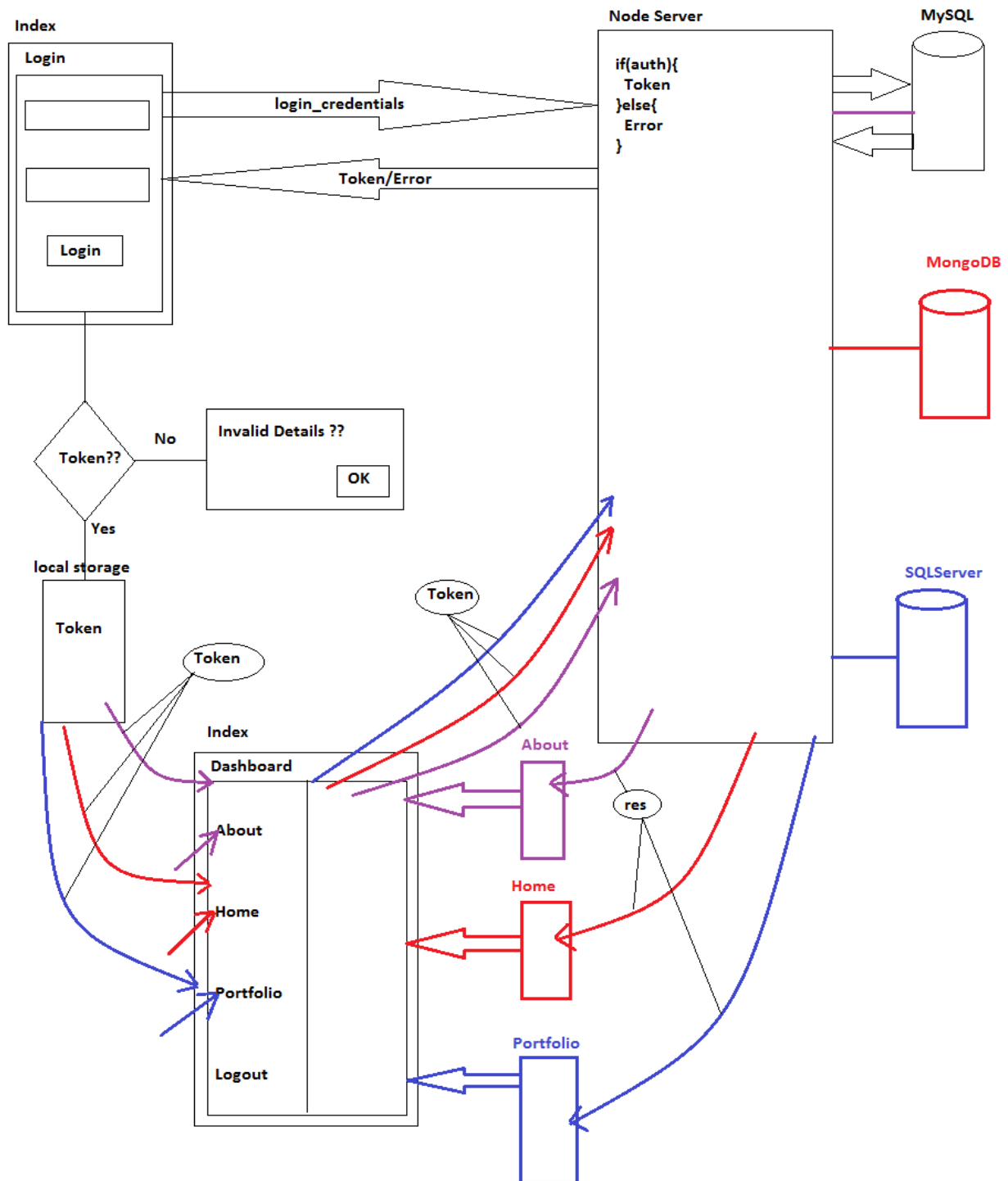
```
super();
this.state={
  status : ""
};
};
render(){
  return(
    <div>
      <fieldset>
        <legend>Delete</legend>
        <input type="number"
          ref="eid"
          placeholder="Employee ID"/>
        <br></br><br></br>
        <button onClick={this.remove}>Delete</button>
        <h1>{this.state.status}</h1>
      </fieldset>
    </div>
  )
};
remove = ()=>{
  let record = {
    "eid":this.refs.eid.value
  };
  axios.delete("http://localhost:8080/delete",{data:record})
    .then((posRes)=>{
      this.setState({
        status:posRes.data.delet
      })
    })
}
```



```
    }, (errRes) => {  
      console.log(errRes);  
    });  
  };  
};
```



## Chapter-15 (Mini Project)



=> in this miniproject(POC) we have following databases.

MySQL

MongoDB

SQLServer

=> MySQL database for both login authentication as well as about module in miniproject.

=> MongoDB database for "home" module present in MiniProject

=> SQLServer for "portfolio" module present in MiniProject.

=> we will develop rest apis by using node js.

=> in the above miniproject, we are maintaing token based authentication system.

=> Coming to the ReactJS, we are dividing ReactJS Application into Multiple Modules

- Index
- Login
- Dashboard

@About

@Home

@Portfolio

=> In ReactJS, mainly we are focusing on Single Page Applications.



=> Loading "One Component" to "Another Component" without refreshing whole web project called as Single Page Application.

## Implementation

-----

1) download and install MySQL database.

mysql-essential-5.2.3-falcon-alpha-win32.msi

2) create the tables in MySQL database.

Default Password : root

```
> create schema reactjs;
```

- automatically "reactjs" database will ready.

```
> use reactjs;
```

- we can switch to reactjs database.

```
> create table login_details(username varchar(20),  
                                upwd varchar(20));
```

- automatically login\_details table will be created.

```
> insert into login_details values("admin","admin");
```

- we can insert the record into login\_details table.

```
> create table about(sno integer,name varchar(20));
```

- we can create the "about" table.

```
> insert into about values(1,"MERN Stack");
```

- we can insert the record into about table.



```
> select * from login_details;
```

```
select * from about;
```

- we can fetch the data from tables.

```
*****
```

```
host :      localhost
```

```
user :      root
```

```
password: root
```

```
database: reactjs
```

```
tables      :      login_details
```

```
about
```

```
*****
```

3) make the MongoDB database ready for miniproject Implementation.

```
> mongod
```

- automatically mongo server will start.

```
> mongo
```

- we can connect to mongo server.

```
> use reactjs;
```

- automatically reactjs database will be created  
and switches also.

```
> db.createCollection("home");
```

- automatically "home" collection will be created(table)





```
> db.home.insert({sno:1,name:"reactjs"});

- we can insert the record into home collection.

> db.home.find();

- we can fetch the data from "home" collection.
```

\*\*\*\*\*

```
protocol   :    mongodb
port       :    27017
host       :    localhost
database   :    reactjs
collection :    home
```

\*\*\*\*\*

4) make the SQLServer ready for miniproject Implementation.

Name : SQL Server 2014 Management Studio

\*\*\*\*\*

```
User       :    sa
Password   :    123
server     :    localhost
database   :    reactjs
table      :    portfolio
```

\*\*\*\*\*



## 5) create the react application

```
> create-react-app miniproject-app
```

## 6) switch to react application

```
> cd miniproject-app
```

## 7) download the node modules

```
=> express
```

```
=> mysql
```

```
=> mongodb@2.2.32
```

```
=> mssql
```

```
=> body-parser
```

```
=> cors
```

```
=> jwt-simple
```

- "express" module used to develop the rest api.
- "mysql" module used to connect to mysql database.
- "mongodb@2.2.32" module used to connect to mongodb database.
- "mssql" module used to interact with the SQLServer
- "body-parser" module used to read the post parameters.
- "cors" module used to enable the ports communication.
- "jwt-simple" module used to generate the tokens.

```
> yarn add express mysql mongodb@2.2.32 mssql body-parser cors  
jwt-simple --save
```



8) develop the rest apis for miniproject by using nodejs

\*\*\*\*\*

miniproject-app =>server

config

mysql\_properties.js

mysql\_connection.js

mssql\_properties.js

token.js

generateToken.js

auth.js

login

login.js

about

about.js

home

home.js

portfolio

portfolio.js

logout

logout.js

server.js

\*\*\*\*\*



- "mysql\_properties.js" file used to maintain the mysql database details.
- "mysql\_connection.js" file used to create the mysql connection object.
- "mssql\_properties.js" file used to maintain the SQLServer details.
- "token.js" file used to save the server side token.
- "generateToken.js" file used to generate the token by using jwt-simple module.
- "auth.js" file used to compare the client side token with server side token.
- "login.js" file used to compare the react application login credentials with login\_details table present in mysql database.
- "about.js" file used to fetch the data from about table present in MySQL database.
- "home.js" file used to fetch the data from home collection present in MongoDB database.
- "portfolio.js" file used to fetch the data from portfolio table present in SQLServer.
- "logout.js" file used to delete the server side token.
- "server.js" file is the main server file.

### mysql\_properties.js

```
//this file used to maintain the MySQL Database properties.
```

```
//we will create and export JSON Object
```

```
let obj = {  
  
  host      :   "localhost",
```



```
    user      :    "root",

    password:    "root",

    database:    "reactjs"

};

module.exports = obj;

mysql connection.js

//import mysql module

let mysql = require("mysql");

//import db details

let obj = require("./mysql_properties");

//create and export JSON Object

let conn = {

    getConnection : ()=>{

        return mysql.createConnection(obj);

    }

};
```

```
//export conn
```

```
module.exports = conn;
```

### mssql properties.js

```
//this file used to maintain the SQLServer Properties.
```

```
let obj = {
```



```
    server : "localhost",  
    user    : "sa",  
    password: "123",  
    database: "reactjs"  
};
```

```
module.exports = obj;
```

### token.js

```
//this file used to maintain the server side token.
```

```
let obj = {  
    token : ""  
};
```

```
module.exports = obj;
```

### generateToken.js

```
//this file used to generate the token by using jwt-simple  
module
```

```
//import jwt-simple module
```

```
let jwt = require("jwt-simple");
```

```
//converting readable data to unreadable data with custom  
password called as token
```

```
let my_fun = (data,password)=>{  
    return jwt.encode(data,password);  
};
```



```
module.exports = my_fun;
```

### auth.js

```
//this file used to compare the "client side token" with  
"server side token"
```

```
//client sending the token through "headers".
```

```
//server side token present in "token.js" file.
```

```
//comparing tokens before processing requests called as  
"middleware"
```

```
let auth = (req,res,next)=>{  
  
  let allHeaders = req.headers;  
  
  let c_token = allHeaders.token;  
  
  console.log(c_token);  
  
  let s_token = require("./token").token;  
  
  console.log(s_token);  
  
  if(c_token === s_token){  
  
    next();  
  
  }else{  
  
    res.send("Unauthorized User....!");  
  
  }  
  
};  
  
module.exports = auth;
```



### login.js

```
//this file used to create the module.

//this module used to create the login rest api

//login rest api used to compare the reactjs application
"login credentials" with "login_details" table present in
mysql database.

//get the connection object

let conn = require("../config/mysql_connection");

let connection = conn.getConnection();

//import token.js file, used to store the token

let obj = require("../config/token");

//import generateToken.js file, used to generate token based
on successful authentication.

let my_fun = require("../config/generateToken");

//create and export the module

module.exports =
require("express").Router().post("/", (req, res) => {

    connection.query(`select * from login_details where
uname='${req.body.uname}' and upwd='${req.body.upwd}'`,

(err, records, fields) => {

    if (records.length > 0) {

        let token = my_fun({
```





```
        uname:req.body.uname,

        upwd:req.body.upwd

    }, "hr@tcs.in");

    obj.token = token;

    res.send({login:"success", token:token});

    }else{

        res.send({login:"fail"});

    }

    });

});
```

### about.js

```
let conn = require("../config/mysql_connection");

let connection = conn.getConnection();

let auth = require("../config/auth");

module.exports =

require("express").Router().get("/", [auth], (req, res) => {

    connection.query(`select * from

    about`, (err, records, fields) => {

        res.send(records);

    });

});
```

### home.js



```
//this file used to fetch the data from home collection
present in reactjs database

let mongodb = require("mongodb");

let auth = require("../config/auth");

module.exports =
require("express").Router().get("/", [auth], (req, res) =>{

    let nareshIT = mongodb.MongoClient;

    nareshIT.connect("mongodb://localhost:27017/reactjs", (err, db) =
    >{

        db.collection("home").find().toArray((err, array) =>{

            res.send(array);

        });

    });

});
```

### portfolio.js

```
//this file used to fetch the data from portfolio table
present in reactjs database in SQLServer.

let mssql = require("mssql");

let obj = require("../config/mssql_properties");

let auth = require("../config/auth");

module.exports =
require("express").Router().get("/", [auth], (req, res) =>{
```



```
mssql.connect(obj, (err) => {  
  if (err) {  
    console.log("Error...!");  
  } else {  
    let request = new mssql.Request();  
    request.query(`select * from  
portfolio`, (err, records) => {  
      if (err) {  
        console.log("Error...!");  
      } else {  
        res.send(records);  
      }  
      mssql.close();  
    });  
  }  
});  
});
```

### logout.js

```
//this file used to delete the server side token  
  
let obj = require("../config/token");  
  
let auth = require("../config/auth");
```

---



```
module.exports =  
require("express").Router().get("/", [auth], (req, res) => {  
    obj.token = "";  
    res.send({logout: "success"});  
});
```

### server.js

```
//node starts the execution from server.js file.  
  
let express = require("express");  
  
let bodyparser = require("body-parser");  
  
let cors = require("cors");  
  
let app = express();  
  
app.use(bodyparser.json());  
  
app.use(bodyparser.urlencoded({extended: false}));  
  
app.use(cors());  
  
app.use("/login", require("./login/login"));  
  
app.use("/about", require("./about/about"));  
  
app.use("/home", require("./home/home"));  
  
app.use("/portfolio", require("./portfolio/portfolio"));  
  
app.use("/logout", require("./logout/logout"));  
  
app.listen(8080);  
  
console.log("server listening the port no.8080");
```



**Step 9:**

**Start the node server**

```
> cd miniproject-app
```

```
> cd server
```

```
> node server
```

automatically node server will start on port no 8080

**Step 10.**

**Test the following rest apis by using Postman**

```
=> http://localhost:8080/login (POST)
```

```
=> http://localhost:8080/about (GET)
```

```
=> http://localhost:8080/home (GET)
```

```
=> http://localhost:8080/portfolio (GET)
```

```
=> http://localhost:8080/logout (GET)
```

**Step 11.**

**download the axios module**

- "axios" module used to make the rest api calls

```
> cd miniproject-app
```

```
> yarn add axios --save
```



## Step 12.

implement the Components with Rest API Calls.

\*\*\*\*\*

Miniproject-app

src

Login.js

About.js

Home.js

Portfolio.js

\*\*\*\*\*

### Login.js

```
import React,{ Component } from "react";
```

```
import "../App.css";
```

```
import axios from "axios";
```

```
export default class Login extends React.Component{
```

```
  render() {
```

```
    return (
```

```
      <div className="App">
```

```
        <fieldset>
```

```
          <legend>Login</legend>
```



```
        <label
style={{marginRight:40}}>Uname:</label>

        <input type="text"

            ref="uname"

            placeholder="User Name"></input>

        <br></br><br></br>

        <label
style={{marginRight:40}}>Upwd:</label>

        <input type="password"

            ref="upwd"

            placeholder="Password"></input>

        <br></br><br></br>

        <button onClick={this.login}>Login</button>

    </fieldset>

</div>

)

}

login = ()=>{

    axios.post("http://localhost:8080/login",

        {

            'uname':this.refs.uname.value,

            'upwd':this.refs.upwd.value
```



```
    }) .then( (posRes) => {  
        let str = JSON.stringify(posRes.data);  
  
window.localStorage.setItem("login_details", str);  
  
        //navigate to Dashboard  
        this.props.history.push('/dashboard')  
  
    }, (errRes) => {  
        console.log(errRes);  
    });  
};  
};
```

### About.js

```
import React, { Component } from "react";  
  
import axios from "axios";  
  
import "./App.css";  
  
export default class About extends React.Component {  
    constructor() {  
        super();  
  
        this.state = {  
            res : "Please Wait....!"  
        };  
    }  
};
```





```
}

render() {

  return(

    <div className="App">

      <h1 style={{color:"red"}}>{this.state.res}</h1>

    </div>

  )

};

componentDidMount() {

  let str = window.localStorage.getItem("login_details");

  let obj = JSON.parse(str);

  console.log(obj.token);

  axios.get("http://localhost:8080/about", {headers: {token:obj.token}})

    .then((posRes)=>{

      console.log(posRes);

      let str = JSON.stringify(posRes.data);

      this.setState({

        res:str

      })

    }, (errRes)=>{
```



```
        console.log(errRes) ;  
    });  
}  
};
```

### Home.js

```
import React,{ Component } from "react";  
  
import axios from "axios";  
  
import "./App.css";  
  
export default class Home extends React.Component{  
    constructor(){  
        super();  
        this.state = {  
            res : "please wait....!"  
        }  
    };  
  
    render(){  
        return(  
            <div className="App">  
                <h1  
style={{color:"green"}}>{this.state.res}</h1>  
                </div>  
        )  
    }
```



```
};

componentDidMount() {

    let token =
JSON.parse(window.localStorage.getItem("login_details"))

        .token;

    axios.get("http://localhost:8080/home",{headers:{token:token}}) .
    then((posRes)=>{

        this.setState({

            res:JSON.stringify(posRes.data)

        });

        },(errRes)=>{

            console.log(errRes);

        });

    });

};
```

### Portfolio.js

```
import React,{ Component } from "react";

import axios from "axios";

import "./App.css";

export default class Portfolio extends React.Component{

    constructor(){
```



```
    super();

    this.state = {

        res : "please wait....!"

    }

};

render() {

    return(

        <div className="App">

            <h1 style={{color:"blue"}}>{this.state.res}</h1>

        </div>

    )

};

componentDidMount() {

    let token =

JSON.parse(window.localStorage.getItem("login_details"))

        .token;

    axios.get("http://localhost:8080/portfolio",

{headers:{token:token}}).then((posRes)=>{

        this.setState({

            res:JSON.stringify(posRes.data)

        });

    });

}
```



```
    }, (errRes)=>{  
        console.log(errRes) ;  
    });  
};  
};
```

### Step 13.

download "react-router-dom" module

- "react-router-dom" module used to implement the Single Page Applications.

```
> cd miniproject-app  
  
> yarn add react-router-dom --save
```

### Step 14.

create the Dashboard Module

\*\*\*\*\*

Miniproject-app

Src

Dashboard.js

\*\*\*\*\*

Dashboard.js

```
import React,{ Component } from "react";  
  
import axios from "axios";
```



```
//BrowserRouter used to hold the target pages without refreshing

//NavLink used to define the HyperLinks

import { BrowserRouter as Router,NavLink } from "react-router-dom";

//import Route

import Route from "react-router-dom/Route";

//import Target Components

import About from "../About";

import Home from "../Home";

import Portfolio from "../Portfolio";

export default class Dashboard extends React.Component{

  render(){

    return(

      <Router>

        <div>

          <NavLink to="/about"

            activeStyle={{color:"green"}}

            exact strict

            style={{marginRight:100}}>About</NavLink>

          <NavLink to="/home"

            activeStyle={{color:"green"}}
```



```
        exact strict

style={{marginRight:100}}>Home</NavLink>

        <NavLink to="/portfolio"

            activeStyle={{color:"green"}}

            exact strict

style={{marginRight:100}}>Portfolio</NavLink>

        <button
onClick={this.logout}>Logout</button>

        <br></br><br></br>

        <Route path="/about" exact strict
component={About}></Route>

        <Route path="/home" exact strict
component={Home}></Route>

        <Route path="/portfolio" exact strict
component={Portfolio}></Route>

    </div>

</Router>

) };

logout = ()=>{

    let str = window.localStorage.getItem("login_details");
```



```
    let obj = JSON.parse(str);

    let token = obj.token;

    axios.get("http://localhost:8080/logout",{headers:{"token":token
    }})

        .then((posRes)=>{

            if(posRes.data.logout === "success"){

window.localStorage.removeItem("login_details");

                //navigate to login page

                this.props.history.push('/')

            }

        }, (errRes)=>{

            console.log(errRes);

        });

    });};
```

Step 15.

**create the Main Component**

\*\*\*\*\*

Miniproject-app

Src

Main.js





\*\*\*\*\*

### Main.js

```
import React,{ Component } from "react";

import { BrowserRouter as Router } from "react-router-dom";

import Route from "react-router-dom/Route";

import Login from "../Login";

import Dashboard from "../Dashboard";

export default class Main extends React.Component{

  render() {

    return(

      <Router>

        <div>

          <Route path="/" exact strict
component={Login}></Route>

          <Route path="/dashboard" exact strict
component={Dashboard}></Route>

        </div>

      </Router>

    )

  }

};
```

Step 16.



## Register the Main Component.

\*\*\*\*\*

Miniproject-app

Src

index.js

\*\*\*\*\*

index.js

```
import React from 'react';
```

```
import ReactDOM from 'react-dom';
```

```
import './index.css';
```

```
import Main from "./Main";
```

```
import * as serviceWorker from './serviceWorker';
```

```
ReactDOM.render(<Main />, document.getElementById('root'));
```

```
// If you want your app to work offline and load faster, you can  
change
```

```
// unregister() to register() below. Note this comes with some  
pitfalls.
```

```
// Learn more about service workers: https://bit.ly/CRA-PWA
```

```
serviceWorker.unregister();
```



### Step 17.

execute the application with the help of node server.

```
//Terminal-1
```

```
> cd miniproject-app
```

```
> cd server
```

```
> node server
```

```
//Terminal-2
```

```
> mongod
```

```
//Terminal-3
```

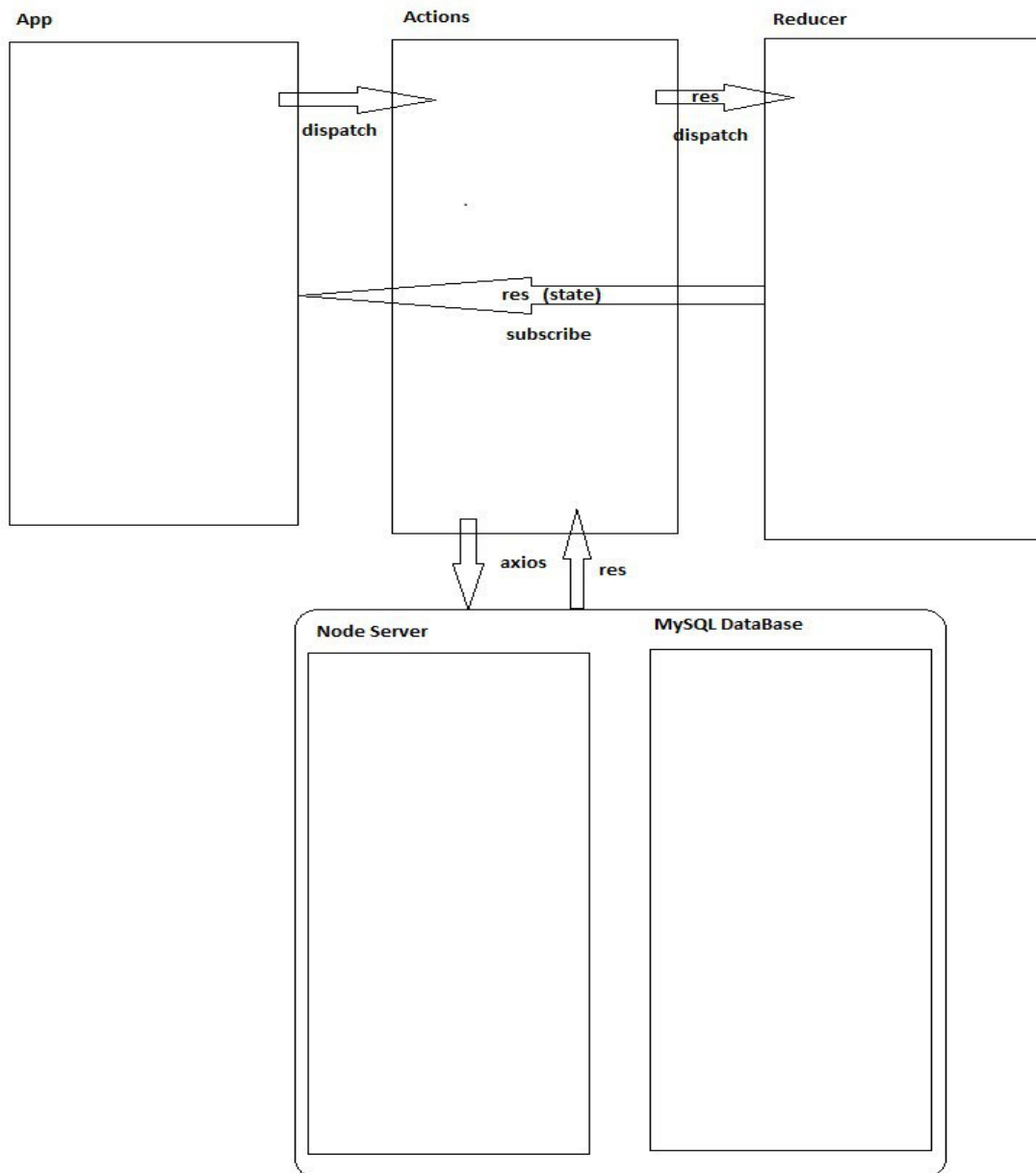
```
> yarn start
```

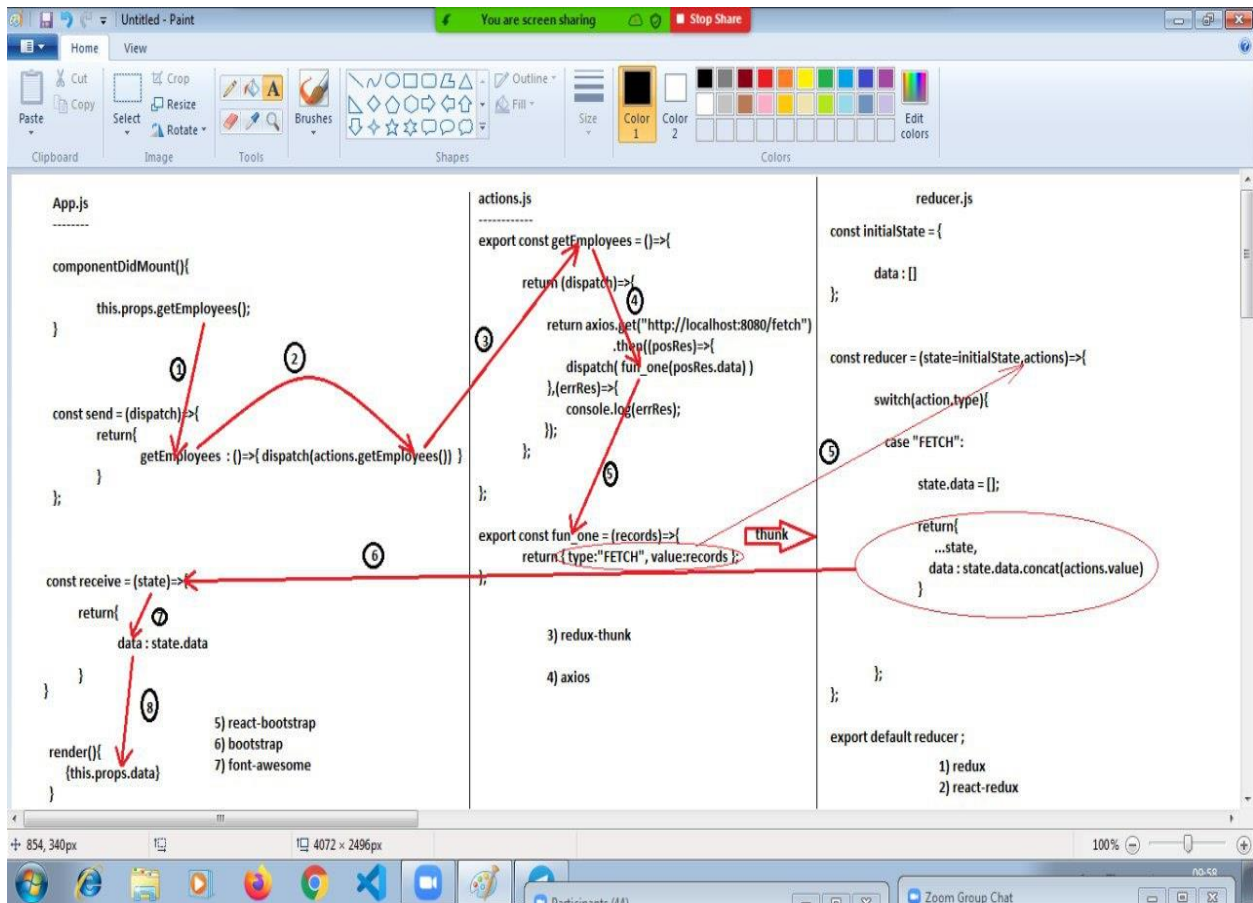


## Chapter-16 (Crud Operations-Thunk)

### CRUD Operations on MySQL DataBase

-----





## 1) create the directory

Ex.

MySQL

## 2) download node modules

- 1) express
- 2) mysql
- 3) body-parser
- 4) cors



- "express" module used to develop the rest apis.
- "mysql" module used to interact with the mysql DataBase.
- "body-parser" module used to read the data from client.
- "cors" module used to enable the cors policy

```
> cd MySQL
```

```
> yarn add express mysql body-parser cors --save
```

3) create the table in MySQL Database.

```
*****
```

```
host      :   localhost
```

```
user      :   root
```

```
password  : root
```

```
database  : fullstack
```

```
table     : products
```

```
*****
```

4) develop rest apis

```
*****
```

MySQL

config



```
    db_properties.js

    db_connection.js

fetch

    fetch.js

insert

    insert.js

update

    update.js

delete

    delete.js

server.js
```

\*\*\*\*\*

- "db\_properties.js" file used to maintain the mysql database details.
- "db\_connection.js" file used to create and return the mysql database connection object.
  - "fetch.js" file used to create the GET Request.
  - "insert.js" file used to create the POST Request.
  - "update.js" file used to create the PUT Request.
  - "delete.js" file used to create the DELETE Request.
  - "server.js" file is the main node server file.



### db\_properties.js

```
let obj = require("../db_properties");

let mysql = require("mysql");

let conn = {

  getConnection : ()=>{

    return mysql.createConnection(obj);

  }

};

module.exports = conn;
```

### db\_connection.js

```
let obj = {

  host : "localhost",

  user : "root",

  password : "root",

  database : "fullstack_9pm"

};

module.exports = obj;
```

### fetch.js

```
let conn = require("../config/db_connection");

let connection = conn.getConnection();

let fetch = require("express").Router().get("/", (req, res)=>{
```





```
        connection.query(`select * from
products`, (err, records, fields) => {

            if (err) throw err;

            else {

                res.send(records);

            }

        });

    });
```

```
module.exports = fetch;
```

### insert.js

```
let conn = require("../config/db_connection");

let connection = conn.getConnection();

let insert = require("express").Router().post("/", (req, res) => {

    connection.query(`insert into products
values (${req.body.p_id},

    '${req.body.p_name}',

    ${req.body.p_cost})`, (err, result) => {

        if (err) throw err;

        else {

            res.send({ "insert": "success", record: { "p_id": `${req.body.p_id}`,
```



## update.js



```
"p_cost":req.body.p_cost}}});  
  
    }  
  
    });  
  
});  
  
module.exports = update;
```

### delete.js

```
let conn = require("../config/db_connection");  
  
let connection = conn.getConnection();  
  
let remove = require("express").Router().delete("/", (req,res)=>{  
  
    connection.query(`delete from products where  
p_id=${req.body.p_id}`, (err,result)=>{  
  
        if(err) throw err;  
  
        else{  
  
            res.send({delete:"success","p_id":req.body.p_id});  
  
        }  
  
    });  
  
});  
  
module.exports = remove;
```

### server.js

```
let express = require("express");
```

---



```
let app = express();

let bodyparser = require("body-parser");

let cors = require("cors");

app.use(cors());

app.use(bodyparser.json());

app.use(bodyparser.urlencoded({extended:false}));

app.use("/fetch",require("./fetch/fetch"));

app.use("/insert",require("./insert/insert"));

app.use("/update",require("./update/update"));

app.use("/delete",require("./delete/delete"));

app.listen(8080);

console.log("server listening the port no.8080");
```

## 5) start the node server

```
> cd MySQL
```

```
> node server
```

## 6) test the rest apis by using Postman

```
=> http://localhost:8080/fetch      (GET)
```

```
=> http://localhost:8080/insert     (POST)
```

```
=> http://localhost:8080/update     (PUT)
```

```
=> http://localhost:8080/delete     (DELETE)
```



## 7) create the react application

```
> create-react-app crud-app
```

## 8) switch to crud-app

```
> cd crud-app
```

## Terminology

### Actions

-----

- this layer used to monitor the Actions

Ex.

Requesting Total Products (GET)

Add the New Product (POST)

### Reducer

-----

- maintain the global business logic
- To implement above architecture we need following node modules

=> redux-thunk

=> redux

=> react-redux

=> bootstrap

=> react-bootstrap

=> axios



- "redux-thunk" library used to maintain and monitor the separate actions.
- "redux" used to create the reducers
- To integrate reducer to component we will use react-redux
- "bootstrap" & "react-bootstrap" used to implement the bootstrap in react component
- "axios" module used to make the asynchronous calls

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

```
*****
```

```
public
```

```
    index.html
```

```
*****
```

```
Index.html
```

```
    <script  
src="https://unpkg.com/react/umd/react.production.min.js"  
crossorigin></script>
```

```
<script  
  
    src="https://unpkg.com/react-dom/umd/react-  
dom.production.min.js"  
  
    crossorigin></script>
```



```
<script
  src="https://unpkg.com/react-bootstrap@next/dist/react-
bootstrap.min.js"
  crossorigin></script>

<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootst
rap.min.css"

  integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh
"

  crossorigin="anonymous"

/>

<script
src="https://kit.fontawesome.com/a076d05399.js"></script>
```

## 9) create the Actions

\*\*\*\*\*

```
src
  actions
    actions.js
```



\*\*\*\*\*

### actions.js

```
import axios from "axios";

/*****/

export const getProducts = ()=>{

  return (dispatch)=>{

    return axios.get("http://localhost:8080/fetch")

      .then( (posRes)=>{

        dispatch( fun_one(posRes.data) );

      }, (errRes)=>{

        console.log(errRes);

      });

  }

};

export const fun_one = (records)=>{

  return {type:"FETCH",value:records};

};

/*****/

/*****/

export const deleteProduct = (record)=>{
```

---





```
    return (dispatch)=>{

        return
        axios.delete("http://localhost:8080/delete",{data:record})

            .then( (posRes)=>{

                dispatch(fun_two(posRes.data)) ;

            }, (errRes)=>{

                console.log(errRes) ;

            }) ;

    };

};

export const fun_two = (result)=>{

    return {type:"DELETE",value:result};

};

/*****/

/***** */

export const addProduct = (record)=>{

    return (dispatch)=>{

        return axios.post("http://localhost:8080/insert",record)

            .then( (posRes)=>{

                dispatch(fun_three(posRes.data)) ;

            }, (errRes)=>{
```



```
        console.log(errRes) ;

    });

};

};

export const fun_three = (result)=>{

    return { type:"INSERT", value:result };

}

/***** */

/***** */

export const updateProduct = (record)=>{

    return (dispatch)=>{

        return axios.put("http://localhost:8080/update",record)

            .then( (posRes)=>{

                dispatch(fun_four(posRes.data)) ;

            }, (errRes)=>{

                console.log(errRes) ;

            });

    };

};

};

export const fun_four = (result)=>{

    return { type:"UPDATE", value:result };

}
```

---



```
}
```

```
/****** */
```

## 10) create the reducer

```
*****
```

src

reducer

reducer.js

```
*****
```

### reducer.js

```
const initialState = {
```

```
  data : []
```

```
};
```

```
const reducer = (state=initialState,action)=>{
```

```
  switch(action.type){
```

```
    case "FETCH":
```

```
      state.data = [];
```

```
      return{
```

```
        ...state,
```

```
        data : state.data.concat(action.value)
```

```
      }
```



```
        break;

    case "INSERT":

    case "UPDATE":

    case "DELETE":

        return{

            ...state,

            status : action.value

        }

        break;

    }

    return state;

};

export default reducer;
```

## 11) configure the thunk middleware

```
*****

src

    index.js

*****
```



index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import * as serviceWorker from './serviceWorker';

import { createStore, applyMiddleware } from "redux";
import { Provider } from "react-redux";
import reducer from "./reducer/reducer";
import thunk from "redux-thunk";

const store = createStore(reducer, applyMiddleware(thunk));

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

// If you want your app to work offline and load faster, you can
change
```

---



```
// unregister() to register() below. Note this comes with some  
pitfalls.
```

```
// Learn more about service workers: https://bit.ly/CRA-PWA
```

```
serviceWorker.unregister();
```

## 12) create the Component

```
*****
```

```
src
```

```
App.js
```

```
*****
```

### App.js

```
import React,{ Component } from "react";  
  
import { connect } from "react-redux";  
  
import * as actions from "../actions/actions";  
  
import Container from "react-bootstrap/Container";  
  
import Table from "react-bootstrap/Table";  
  
import Button from "react-bootstrap/Button";  
  
import Modal from "react-bootstrap/Modal";  
  
import Form from "react-bootstrap/Form";  
  
class App extends React.Component{  
  
  constructor(){  
  
    super();  
  
    this.state = {
```



```
        status : false,

        insertPopup : false,

        updatePopup : false

    };

};

componentDidMount() {

    this.props.getProducts();

};

showPopup = (msg)=>{

    if(msg === "insert"){

        this.setState({

            updatePopup:false,

            insertPopup : true,

            status : true

        })

    }else{

        this.setState({

            insertPopup : false,

            updatePopup : true,

            status : true

        })

    }

}
```



```
};
```

```
closePopup = ()=>{  
  this.setState({  
    status : false  
  })  
};
```

```
deleteRecord = (id)=>{  
  this.props.deleteRecord(id);  
};
```

```
save = ()=>{  
  if(this.state.insertPopup){  
    let obj = { "p_id" : this.refs.p_id.value,  
               "p_name" : this.refs.p_name.value,  
               "p_cost" : this.refs.p_cost.value};  
    this.props.addProduct(obj);  
  }  
  else if(this.state.updatePopup){  
    let obj = { "p_id" : this.refs.p_id.value,  
               "p_name" : this.refs.p_name.value,
```





```
        "p_cost" : this.refs.p_cost.value};

        this.props.updateProduct(obj);

        this.closePopup();

    }

    this.closePopup();

};

render() {

    return(

        <Container fluid className="mt-5">

            <Button className="float-right"

                onClick={() => { this.showPopup("insert")

            }}>Add <i className="fas fa-plus"></i>

            </Button>

            { /* ----- modal code start----- */

                <Modal show={this.state.status}

                    onHide={this.closePopup}

                    size="sm"

                    centered>
```



```
      <Modal.Header closeButton>

        <Modal.Title>Add/Update</Modal.Title>

      </Modal.Header>

      <Modal.Body>

        <Form>

          <Form.Group>

            <Form.Label>P_ID</Form.Label>

            <Form.Control type="number"

placeholder="Enter P_ID"

ref="p_id"></Form.Control>

          </Form.Group>

          <Form.Group>

            <Form.Label>P_NAME</Form.Label>

            <Form.Control type="text"

placeholder="Enter P_NAME"

ref="p_name"></Form.Control>

          </Form.Group>

        </Form.Group>

      </Modal.Body>

    </Modal>

  </div>

</div>
```



```
<Form.Label>P_COST</Form.Label>

                                <Form.Control type="number"

placeholder="Enter P_COST"

ref="p_cost"></Form.Control>

                                </Form.Group>

                                </Form>

                                </Modal.Body>

                                <Modal.Footer>

                                    <Button variant="success" onClick={()=>{
this.save("insert",this.props.status) }}>Add/Update</Button>

                                    <Button variant="danger"
onClick={this.closePopup}>Close</Button>

                                </Modal.Footer>

                                </Modal>

/* ----- modal code end----- */

<Table bordered

                                variant="dark"

                                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>

{this.props.data.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><i className="fas fa-edit"
onClick={()=>{ this.showPopup("update") }}></i></td>

        <td><i className="fas fa-trash-alt"
onClick={()=> { this.deleteRecord(element.p_id) } }></i></td>

    </tr>

    )}}

</tbody>

</Table>

</Container>

)

}

};
```

```
const receive = (state)=>{

    if(state.hasOwnProperty("status")){

        if(state.status.delete === "success"){

            let id = state.data.findIndex((element,index)=>{

                return element.p_id === state.status.p_id;

            });
```



```
        state.data.splice(id,1);
    }

    if(state.status.insert === "success"){
        state.data.push(state.status.record);
    }

    if(state.status.update === "success"){

        state.data.forEach((element,index)=>{
            if(element.p_id === state.status.record.p_id){
                element.p_name = state.status.record.p_name;
                element.p_cost = state.status.record.p_cost;
            }
        });
    }
}

return{
    data : state.data,
    status : state.status
}
```

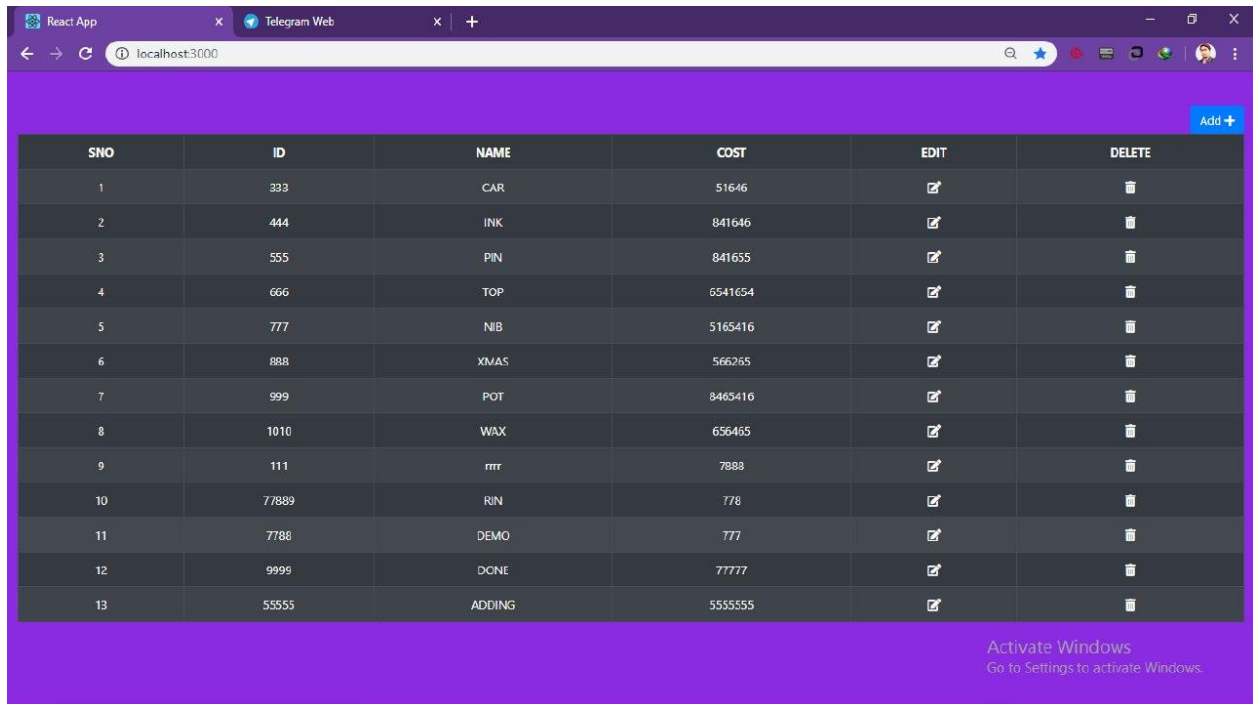




























```
};
```

```
const send = (dispatch)=>{  
  return{  
    getProducts : ()=>{ dispatch(actions.getProducts()) },  
    deleteRecord : (id)=>{  
dispatch(actions.deleteProduct({"p_id":id})) },  
    addProduct    : (record)=>{  
dispatch(actions.addProduct(record)) },  
    updateProduct : (record)=>{  
dispatch(actions.updateProduct(record)) }  
  }  
};  
  
export default connect(receive,send) (App);
```



## Output:



SNO	ID	NAME	COST	EDIT	DELETE
1	333	CAR	51646		
2	444	INK	841646		
3	555	PIN	841655		
4	666	TOP	6541654		
5	777	NIB	5165416		
6	888	XMAS	566265		
7	999	POT	8465416		
8	1010	WAX	656465		
9	111	rrrr	7888		
10	77889	RIN	778		
11	7788	DEMO	777		
12	9999	DONE	77777		
13	55555	ADDING	5555555		

Activate Windows  
Go to Settings to activate Windows.





## Chapter-17 (MiniProject-Saga)

### Step 1) install MongoDB

- MongoDB is the NoSQL database.
- MongoDB database is the "JSON" database.
- MongoDB database supports only "json".
- as a MongoDB developer, we can perform CRUD Operations on JSON.
- MongoDB follows the "client-server" architecture.
- MongoDB follows the "mongodb" protocol.
- MongoDB by default running on port no.27017

i) download and install MongoDB.

website :

[https://www.mongodb.com/dr/fastdl.mongodb.org/win32/mongodb-win32-x86\\_64-2012plus-4.2.8-signed.msi/download](https://www.mongodb.com/dr/fastdl.mongodb.org/win32/mongodb-win32-x86_64-2012plus-4.2.8-signed.msi/download)

file : mongodb-win32-x86\_64-2012plus-4.2.8-signed.msi

ii) create the directory structure

c:/data/db

above directory structure, used to maintain the data backup.

iii) set the path environmental variable

path = C:\Program Files\MongoDB\Server\4.2\bin



## Step 2) create the database

\*\*\*\*\*

### MongoDB Queries

\*\*\*\*\*

> mongod

- mongod is used to start the server

> mongo

- mongo is used to connect to server

> use miniproject@4pm;

- automatically miniproject@4pm database will be created and switches also.

> db.createCollection("login\_details");

> db.createCollection("angular");

> db.createCollection("react");

> db.createCollection("deno");

- automatically tables will be created.

> db.login\_details.insert({"uname":"admin","upwd":"admin"});

> db.angular.insert({"sno":1,"message":"angular discussions soon....!"});

> db.react.insert({"sno":1,"message":"react discussions soon....!"});



```
> db.deno.insert({"sno":1,"message":"deno discussions soon...!"});
```

- automatically we can insert the data into collections

```
> db.login_details.find().pretty();
```

```
> db.angular.find().pretty();
```

```
> db.react.find().pretty();
```

```
> db.deno.find().pretty();
```

- automatically we can fetch the data from collections.

```
*****
```

```
host      : localhost
```

```
protocol  : mongodb
```

```
port      : 27017
```

```
database  : miniproject@4pm
```

```
collections : login_details
```

```
angular
```

```
react
```

```
deno
```

```
*****
```

### Step 3) download node modules

```
=> mongodb@2.2.32
```

```
=> express
```

```
=> cors
```



=> body-parser

=> jwt-simple

- mongodb@2.2.32 module used to interact with the "mongodb" database.
- express module used to develop the rest apis
  - GET
  - POST
  - PUT
  - DELETE
  - HEAD
  - ---
  - ---
  - ---
- cors module used to enable the ports communication
- body-parser module used to read the client data.
- jwt-simple module used to generate the tokens.

```
> yarn add mongodb@2.2.32 express cors body-parser jwt-simple --save
```

Step 4) develop the rest apis by using NodeJS

\*\*\*\*\*

miniproject@4pm

config

generateToken.js

token.js

auth.js



```
login
    login.js
angular
    angular.js
react
    react.js
deno
    deno.js
logout
    logout.js
server.js
```

\*\*\*\*\*

- generateToken.js file used to generate the token.
- "token.js" file used to save the server side token.
- "auth.js" file used to compare the client side token with server side token.
- "login.js" file used to create the login rest api(post) .
- "angular.js" file used to create the angular rest api(GET) .
- "react.js" file used to create the react rest api(GET) .
- "deno.js" file used to create the deno rest api(GET) .
- "logout.js" file used to create the logout rest api (GET)
- "server.js" file used to collabrate the above modules.

**generateToken.js**



```
//this file used to generate the token by using jwt-simple
module

//require() function used to import the modules in nodejs

let jwt = require("jwt-simple");

let generateToken = (data,password)=>{

    return jwt.encode(data,password);

};

module.exports = generateToken;
```

### token.js

```
//create the json object

let obj = {

    "token":""

};

module.exports = obj;
```

### auth.js

```
//this file used to compare the client side token with the
server side token (middleware)

//import server side token

let obj = require("../token");

//create the function

let auth = (req,res,next)=>{

    let allHeaders = req.headers;
```



```
    let c_token = allHeaders.token;

    if(c_token == obj.token){

        next();

    }else{

        res.send({msg:"unauthorized user !!!"});

    }

};

module.exports = auth;

login.js

//this file used to develop the login rest api  (POST)

//import mongodb module

let mongodb = require("mongodb");

//create the client

//mongodb follows the "client server" architecture

let nareshIT = mongodb.MongoClient;

//where "nareshIT" is the client

//import token.js file

//token.js file used to save the server side token

let obj = require("../config/token");
```



```
//generate the token

let generateToken = require("../config/generateToken");


//create and export the "module"

let login = require("express").Router().post("/", (req, res) => {

  nareshIT.connect("mongodb://localhost:27017/authproject", (err, db)
    => {

      if (err) throw err;

      else {

        db.collection("login_details")

          .find({ "uname": req.body.uname, "upwd": req.body.upwd })

            .toArray((err, array) => {

              if (err) throw err;

              else {

                if (array.length > 0) {

                  let token =

generateToken({ "uname": req.body.uname,

"upwd": req.body.upwd }, "hr@nareshit.in");

                  obj.token = token;

                }

              }

            }

          )

        }

      }

    }

  )

}
```





```
        res.send({login:"success",token:token})

    }else{

        res.send({login:"fail"});

    }

}

});

}

});

});

module.exports = login;
```

### angular.js

```
let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let angular =
require("express").Router().get("/",[require("../config/auth")],
(req,res)=>{

nareshIT.connect("mongodb://localhost:27017/authproject", (err,db
)=>{

    if(err) throw err;

    else{

db.collection("angular").find().toArray((err,array)=>{
```



```
        if(err) throw err;

        else{

            res.send(array);

        }

    });

}

});

module.exports = angular;

react.js

let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let react =
require("express").Router().get("/", [require("../config/auth")],
(req, res)=>{

nareshIT.connect("mongodb://localhost:27017/authproject", (err, db
)=>{

    if(err) throw err;

    else{

        db.collection("react").find().toArray((err, array)=>{

            if(err) throw err;

            else{
```



```
        res.send(array) ;

    }

});

}

});

});

module.exports = react;

deno.js

let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let deno =
require("express").Router().get("/", [require("../config/auth")],
(req, res)=>{

nareshIT.connect("mongodb://localhost:27017/authproject", (err, db
)=>{

    if(err) throw err;

    else{

        db.collection("deno").find().toArray((err, array)=>{

            if(err) throw err;

            else{

                res.send(array) ;

            }

        }

    }

}
```

---



```
        });  
    }  
});  
  
});  
  
module.exports = deno;
```

### logout.js

```
let obj = require("../config/token");  
  
let logout =  
require("express").Router().get("/", [require("../config/auth")],  
(req, res) => {  
    obj.token = "";  
    res.send({ "logout": "success" });  
});  
  
module.exports = logout;
```

### server.js

```
let express = require("express");  
  
let app = express();  
  
let cors = require("cors");  
  
let bodyparser = require("body-parser");  
  
app.use(cors());  
  
app.use(bodyparser.json());  
  
app.use(bodyparser.urlencoded({ extended: false }));
```



```
app.use("/login",require("./login/login"));  
app.use("/angular",require("./angular/angular"));  
app.use("/react",require("./react/react"));  
app.use("/deno",require("./deno/deno"));  
app.use("/logout",require("./logout/logout"));  
app.listen(8085);  
console.log("server listening the port no.8085");
```

**Step 5) start the server**

```
> node server
```

**Step 6) test the following rest apis by using Postman.**

```
=> http://localhost:8080/login      (POST)  
=> http://localhost:8080/angular    (GET)  
=> http://localhost:8080/react      (GET)  
=> http://localhost:8080/deno       (GET)  
=> http://localhost:8080/logout     (GET)
```

**Step 7) create the react application**

```
> create-react-app auth-proj
```

**Step 8) switch to the react project**

```
> cd auth-proj
```

**Step 9) download the react modules**



=> axios

=> redux

=> react-redux

=> redux-saga

=> react-router-dom

=> @material-ui/core

=> @material-ui/icons

CDN : <link rel="stylesheet"  
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500  
,700&display=swap" />

CDN : <link rel="stylesheet"  
href="https://fonts.googleapis.com/icon?family=Material+Icons"  
>

- axios module used to make the rest api calls
- redux module used to create the redux architecture
- react-redux module used to integrate the redux architecture to react application.
- redux-saga module used to separate the actions.
- react-router-dom module used to create the single page applications
- @material-ui/core && @material-ui/icons module used to implement the rich ui on react application.



```
> yarn add axios redux react-redux redux-saga react-router-dom @material-ui/core @material-ui/icons -save
```

### Index.html

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <meta name="theme-color" content="#000000" />

    <meta

      name="description"

      content="Web site created using create-react-app"

    />

    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png"

  />

  <!--

    manifest.json provides metadata used when your web app is
    installed on a

    user's mobile device or desktop. See
    https://developers.google.com/web/fundamentals/web-app-manifest/

  -->
```



```
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
```

```
<!--
```

Notice the use of %PUBLIC\_URL% in the tags above.

It will be replaced with the URL of the `public` folder during the build.

Only files inside the `public` folder can be referenced from the HTML.

Unlike `"/favicon.ico"` or `"favicon.ico"`,  
`"%PUBLIC_URL%/favicon.ico"` will

work correctly both with client-side routing and a non-root public URL.

Learn how to configure a non-root public URL by running  
``npm run build``.

```
-->
```

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500
,700&display=swap" />
```

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons"
/>
```

```
<title>React App</title>
```

```
</head>
```

```
<body>
```





```
<noscript>You need to enable JavaScript to run this  
app.</noscript>
```

```
<div id="root"></div>
```

```
<!--
```

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the `<body>` tag.

To begin the development, run ``npm start`` or ``yarn start``.

To create a production bundle, use ``npm run build`` or ``yarn build``.

```
-->
```

```
</body>
```

```
</html>
```

Step 10) create the action

---



\*\*\*\*\*

auth-proj

src

action

action.js

\*\*\*\*\*

action.js

```
import { takeLatest, put, call, take } from "redux-saga/effects";
```

```
import axios from "axios";
```

```
import { push } from 'react-router-redux';
```

```
import createHistory from 'history/createBrowserHistory';
```

```
const history = createHistory()
```

```
function login_call(data){
```

```
    return axios.post("http://localhost:8085/login",data);
```

```
};
```

```
function* login(action) {
```

```
    const res = yield call(login_call,action.value);
```

```
    if(res.data.login == "success"){
```



```
window.localStorage.setItem("login_details",JSON.stringify(res.data));

    //yield call(forwardTo,"/dashboard");

    //this.props.history.push("/dashboard");

    yield put({ type: "LOGIN", value: res.data });

    window.location.href = `/dashboard`

}

}else{

    //navigate to Error

    console.log("error");

}

};

function angular_call(){

    let str = window.localStorage.getItem("login_details");

    let obj = JSON.parse(str);

    return axios.get("http://localhost:8085/angular",{ headers:{

token : obj.token} });

};
```



```
function *angular(action) {  
    const res = yield call(angular_call);  
    yield put({type: "ANGULAR", value: res.data});  
}
```

```
function react_call() {  
    let str = window.localStorage.getItem("login_details");  
    let obj = JSON.parse(str);  
    return  
    axios.get("http://localhost:8085/react", {headers: {token :  
obj.token}});  
};
```

```
function *react(action) {  
    let res = yield call(react_call);  
    yield put({type: "REACT", value: res.data});  
};
```

```
function deno_call() {
```

---



```
    let str = window.localStorage.getItem("login_details");

    let obj = JSON.parse(str);

    return
    axios.get("http://localhost:8085/deno",{headers:{token :
obj.token}});

};
```

```
function *deno(action){

    let res = yield call(deno_call);

    yield put({type : "DENO",value : res.data});

};
```

```
function logout_call(){

    let str = window.localStorage.getItem("login_details");

    let obj = JSON.parse(str);

    return
    axios.get("http://localhost:8085/logout",{headers:{token :
obj.token}});

};
```



```
function *logout(action) {

  let res = yield call(login_call);

  if(res.data.logout == "success"){

    window.localStorage.removeItem("login_details");

    yield put({type:"LOGOUT",value:res});

    window.location.href = `/`

  }else{

    console.logout("error");

  }

};

export function* main_fun() {

  yield takeLatest("LOGIN_ACTION", login);

  yield takeLatest("ANGULAR_ACTION", angular);

  yield takeLatest("REACT_ACTION" , react);

  yield takeLatest("DENO_ACTION", deno);

  yield takeLatest("LOGOUT_ACTION",logout );

};

function forwardTo(location) {

  history.push(location);

}
```

---



```
}
```

## Step 11) create the reducer

```
*****
```

```
auth-proj
```

```
    reducer
```

```
        reducer.js
```

```
*****
```

### reducer.js

```
const reducer = (state={},action)=>{
```

```
    switch(action.type){
```

```
        case "LOGIN":
```

```
            return{
```

```
                ...state,
```

```
                login : action.value
```

```
            }
```

```
            break;
```

```
        case "ANGULAR":
```

```
        case "REACT":
```

```
        case "DENO":
```

```
            return{
```

```
                ...state,
```



```
        msg : action.value
      }
      break;
    case "LOGOUT":
      return{
        ...state
      }
      break;
  };
  return state;
};

export default reducer;
```

Step 12) create the "store" and deploy the "reducer" and make the availability "Main" component.

### index.js

```
import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

import App from "./App";

import reducer from "./reducer/reducer";
```





```
import { Provider } from "react-redux";

import { createStore, applyMiddleware } from "redux";

import createSagaMiddleware from "redux-saga";

import { main_fun } from "../action/action";

const sagaMiddleware = createSagaMiddleware();

const store = createStore(reducer,
  applyMiddleware(sagaMiddleware));

sagaMiddleware.run(main_fun);

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById("root")

);
```

### Step 13) Create the Components

\*\*\*\*\*

auth-proj

src

login.js

dashboard.js

angular.js

reactjs.js



deno.js

App.s

\*\*\*\*\*

### login.js

```
import React,{ Component } from "react";
import { connect } from "react-redux";

class Login extends React.Component{

  render() {

    return(

      <div>

        <fieldset>

          <legend>Login</legend>

          <input type="text"

            placeholder="user name"

            ref="uname"/>

          <br></br><br></br>

          <input type="password"

            placeholder="user password"

            ref="upwd"/>

          <br></br><br></br>

        </fieldset>

      </div>

    )

  }

}
```



```
        <button
onClick={()=>{this.props.login({"uname":this.refs.uname.value,

"upwd":this.refs.upwd.value})}}>Login</button>

        </fieldset>

    </div>

    )

    };

};

//subscribe

const receive = (state)=>{

    console.log(state);

};

//dispatch

const send = (dispatch)=>{

    return{

        login : (obj)=>{
dispatch({type:"LOGIN_ACTION",value:obj}) }

        }

    };

};
```



```
export default connect(receive,send) (Login) ;

dashboard.js

import React,{ Component } from "react";

import Angular from "../Angular";

import ReactJS from "../ReactJS";

import Deno from "../Deno";

import { BrowserRouter as Router, NavLink } from "react-router-dom";

import Route from "react-router-dom/Route";

import { connect } from "react-redux";

class Dashboard extends React.Component{

  render() {

    return(

      <Router>

        <NavLink to="/angular"

          activeStyle={{color:"red"}}

          exact strict

          style={{marginRight:100}}>Angular</NavLink>

        <NavLink to="/reactjs">
```



```
        activeStyle={{color:"red"}}
        exact strict

style={{marginRight:100}}>React</NavLink>

        <NavLink to="/deno"
        activeStyle={{color:"red"}}
        exact strict

style={{marginRight:100}}>Deno</NavLink>

        <button
onClick={this.props.logout}>Logout</button>

        <br></br><br></br>

        <Route path="/angular" exact strict
component={Angular} />

        <Route path="/reactjs" exact strict
component={ReactJS} />

        <Route path="/deno" exact strict
component={Deno} />

    </Router>

)

}

};
```



```
const receive = (state)=>{  
  return{  
  
  }  
};
```

```
const send = (dispatch)=>{  
  return{  
    logout : ()=>{  
      console.log("hello");  
      dispatch({type:"LOGOUT_ACTION",value:{}}) }  
    }  
};
```

```
export default connect(receive,send) (Dashboard);
```

### angular.js

```
import React,{ Component } from "react";  
import { connect } from "react-redux";
```

```
class Angular extends React.Component{
```



```
componentDidMount() {  
    this.props.getAngularData();  
};  
  
render() {  
    return(  
        <div>  
            {JSON.stringify(this.props.angular)}  
        </div>  
    )  
}  
};  
  
const receive = (state)=>{  
    console.log(state);  
    return{  
        angular : state  
    }  
};  
  
const send = (dispatch)=>{  
    return{  
        getAngularData : ()=>{  
dispatch({type:"ANGULAR_ACTION",value:{}}) }  
    }  
}
```



```
}

export default connect(receive,send) (Angular) ;

reactjs.js

import React,{ Component } from "react";

import {connect} from "react-redux";

class ReactJS extends React.Component{

  componentDidMount(){

    this.props.getReactModuleData();

  };

  render(){

    return(

      <div>

        {JSON.stringify(this.props.msg)}

      </div>

    )

  }

};

const receive = (state)=>{

  return{

    msg : state

  }

}
```





```
};

const send = (dispatch)=>{

  return{

    getReactModuleData : ()=>{
dispatch({type:"REACT_ACTION",value:{}} )}

  }

};

export default connect(receive,send) (ReactJS);
```

### deno.js

```
import React,{ Component } from "react";

import { connect } from "react-redux";

class Deno extends React.Component{

  componentDidMount(){

    this.props.getDenoData();

  };

  render(){

    return(

      <div>

        {JSON.stringify(this.props.msg)}

      </div>

    )

  }

}
```



```
};

const receive = (state)=>{

  return{

    msg : state

  }

};

const send = (dispatch)=>{

  return{

    getDenoData : ()=>{
dispatch({type:"DENO_ACTION",value:{}}) }

  }

};

export default connect(receive,send) (Deno);
```

### App.js

```
import React,{ Component } from "react";

import Login from "./Login";

import Dashboard from "./Dashboard";

import { BrowserRouter as Router } from "react-router-dom";

import Route from "react-router-dom/Route";

class App extends React.Component{

  render(){

    return(
```



```
    <Router>

      <Route path="/" component={Login} exact strict />

      <Route path="/dashboard" component={Dashboard} exact
strict/>

    </Router>

  )

  });

export default App;
```

#### Step 14)

execute the application with the help of node server.

//Terminal-1

```
> cd miniproject-app
> cd server
> node server
```

//Terminal-2

```
> mongod
```

//Terminal-3

```
>cd auth-proj
> yarn start
```



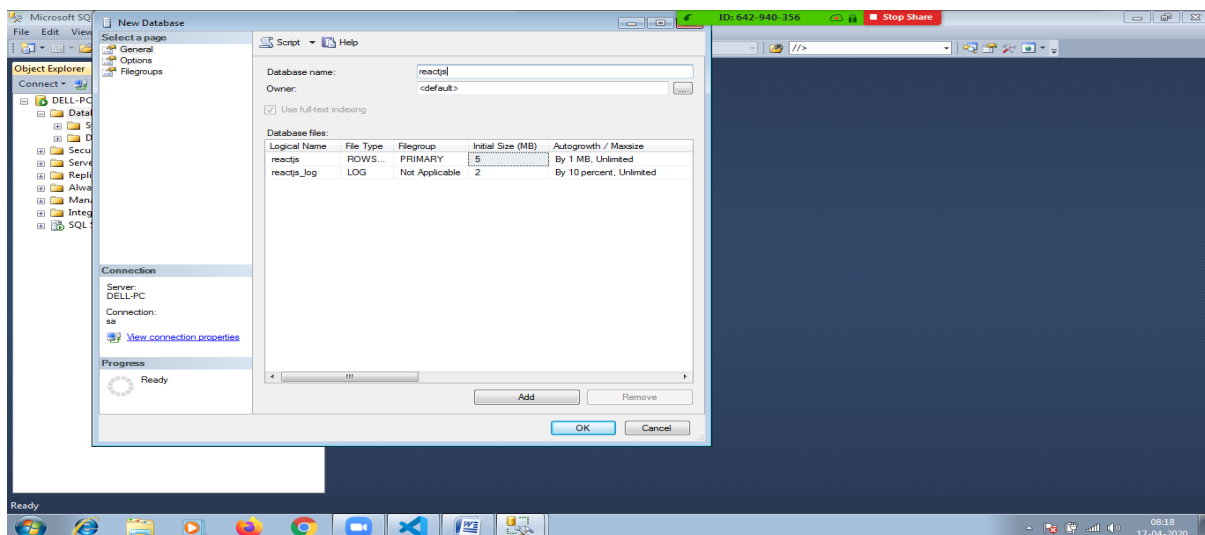
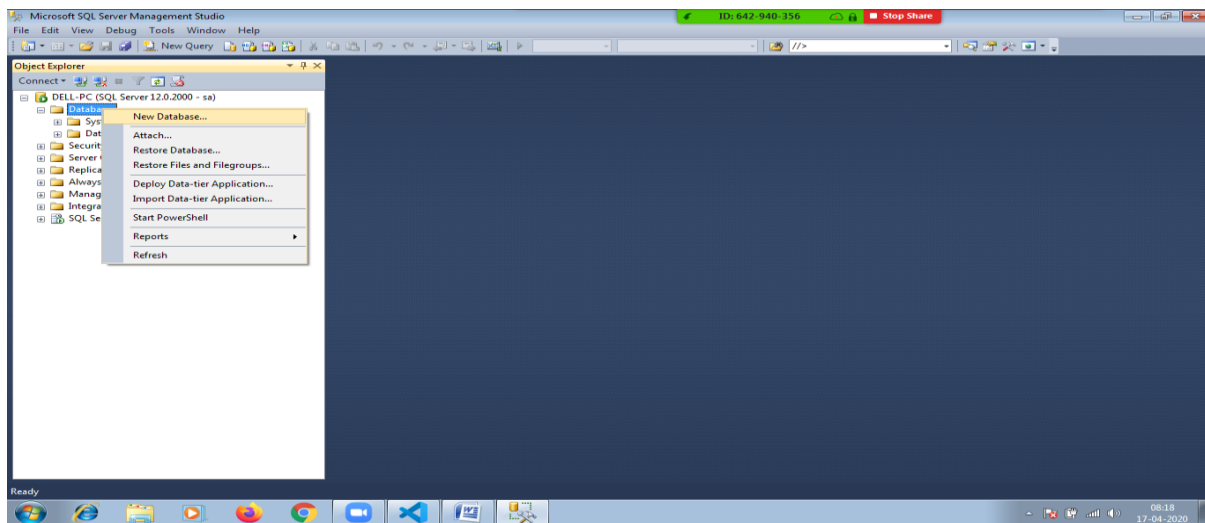
## Chapter-18 (Validations Example)

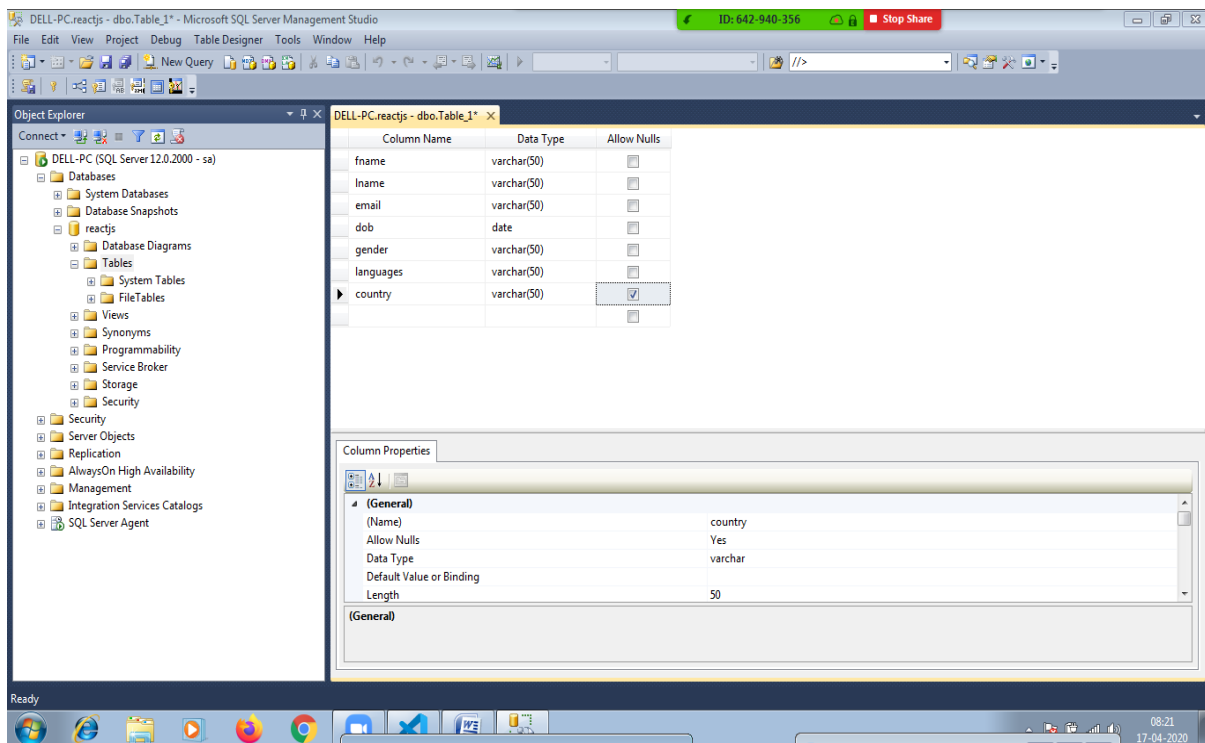
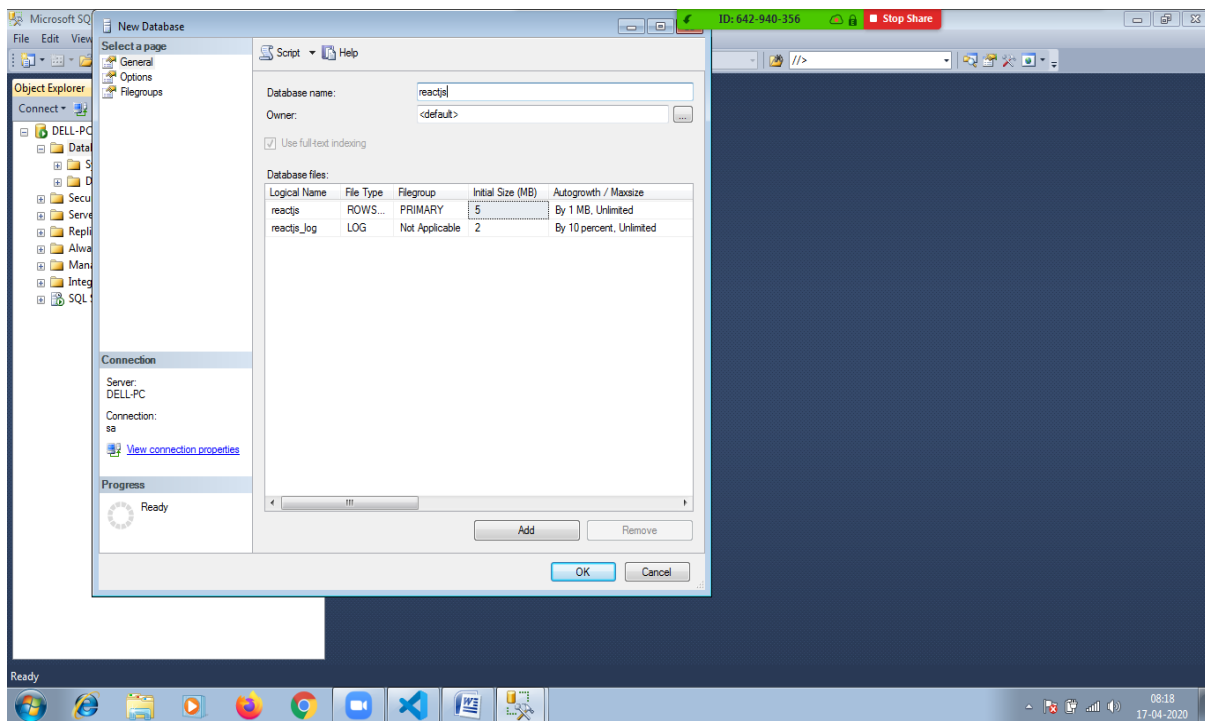
In this example we will store the data into database (SQLServer)

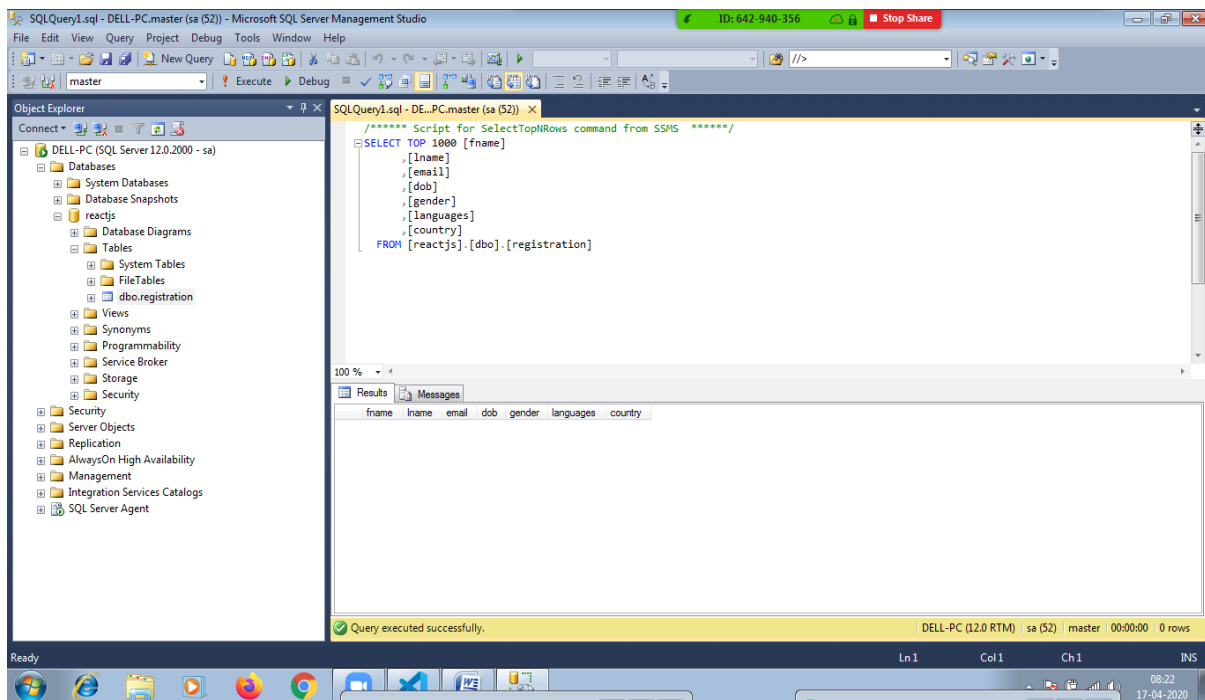
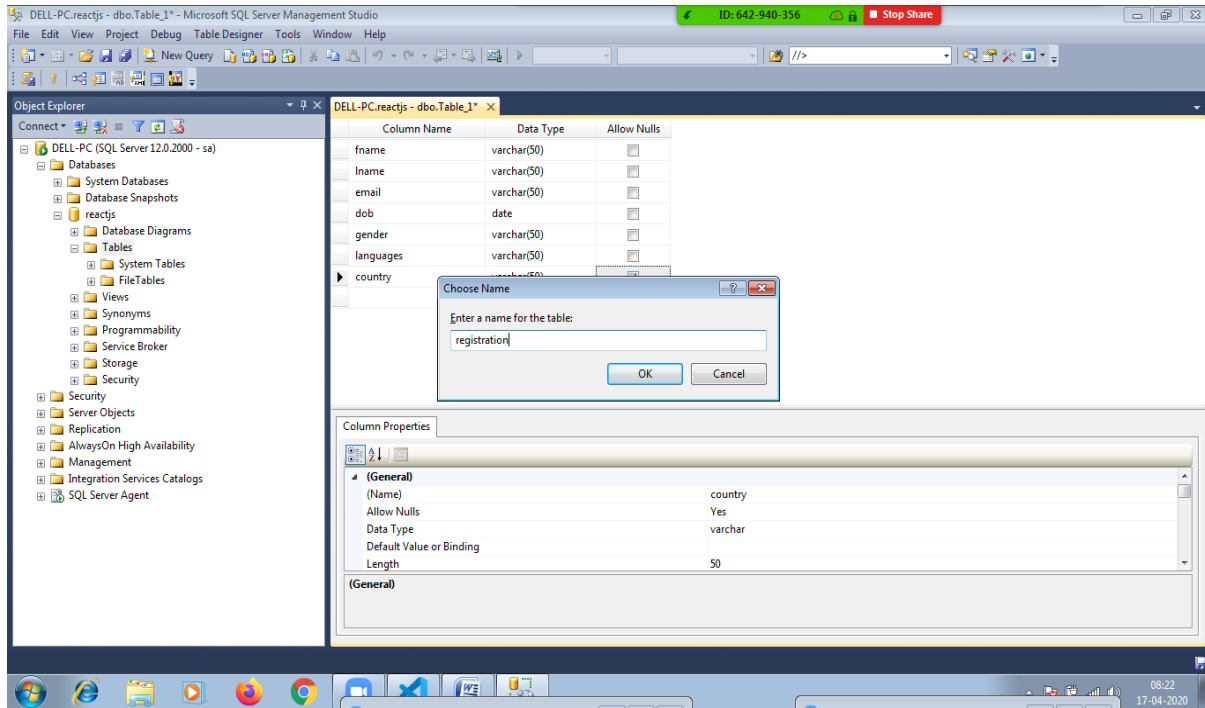
### Step 1) install SQLServer

SQL Server 2014 Management Studio

### Step 2) Make the database ready to store the data







\*\*\*\*\*

user : sa

password : 123

server : localhost

database : reactjs

table : registration

\*\*\*\*\*

Step 3) create the react project

> create-react-app registration-app

Step 4) switch to react application

> cd registration-app

Step 5) download the node modules

=> express

=> cors

=> mssql@6.0.1

=> body-parser

=> axios

- "express" is the node module, used to develop the rest apis
- "cors" module also node module, used to provide the communication between ports.
- "mssql@6.0.1" module used to connect to SQLServer.
- "body-parser" used to read the client data (post parameters)



- "axios" module used to make the asynchronous calls
- we will download above modules by using "yarn" tool

```
> yarn add express cors mssql@6.0.1 body-parser axios --save
```

Step 6) develop the rest api by using nodejs

```
*****
```

```
registration-app
```

```
    server
```

```
        server.js
```

```
*****
```

```
server.js
```

```
//import the modules
```

```
//require() function is used to import the modules
```

```
let express = require("express");
```

```
let mssql = require("mssql");
```

```
let cors = require("cors");
```

```
let bodyparser = require("body-parser");
```

```
//rest apis => GET,POST,PUT,DELETE,HEAD,OPTIONS,TRACE (urls)
```





```
//master obj    (rest obj)

//where "app" object is rest obj

//where "app" object used to develop the rest apis

let app = express();


//ports communication

app.use(cors());


//set the json as communication language (MIME Type)

app.use(bodyParser.json());


//read the client data  (form data) (extended parameters)

app.use(bodyParser.urlencoded({extended:false}));


//create the rest api

app.post("/registration", (req, res) => {

    //connect to SQLServer

    mssql.connect({

        user: "sa",

        password: "123",

        server: "localhost",
```



```
    database:"reactjs"

    }, (err)=>{

        if(err) throw err;

        else{

            //create the query object

            //query object used to execute the SQL Queries

            //where "request" is the query object

            let request = new mssql.Request();

            //execute SQL Query

            request.query(`insert into registration values('${req
q.body.fname}','${req.body.lname}','${req.body.email}','${req.bo
dy.dob}','${req.body.gender}','${req.body.languages}','${req.bod
y.country}')`),(err)=>{

                if(err) throw err;

                else{

                    res.send({registration:"success"});

                }

            })

        }

    });

});

//assign the port no
```

---



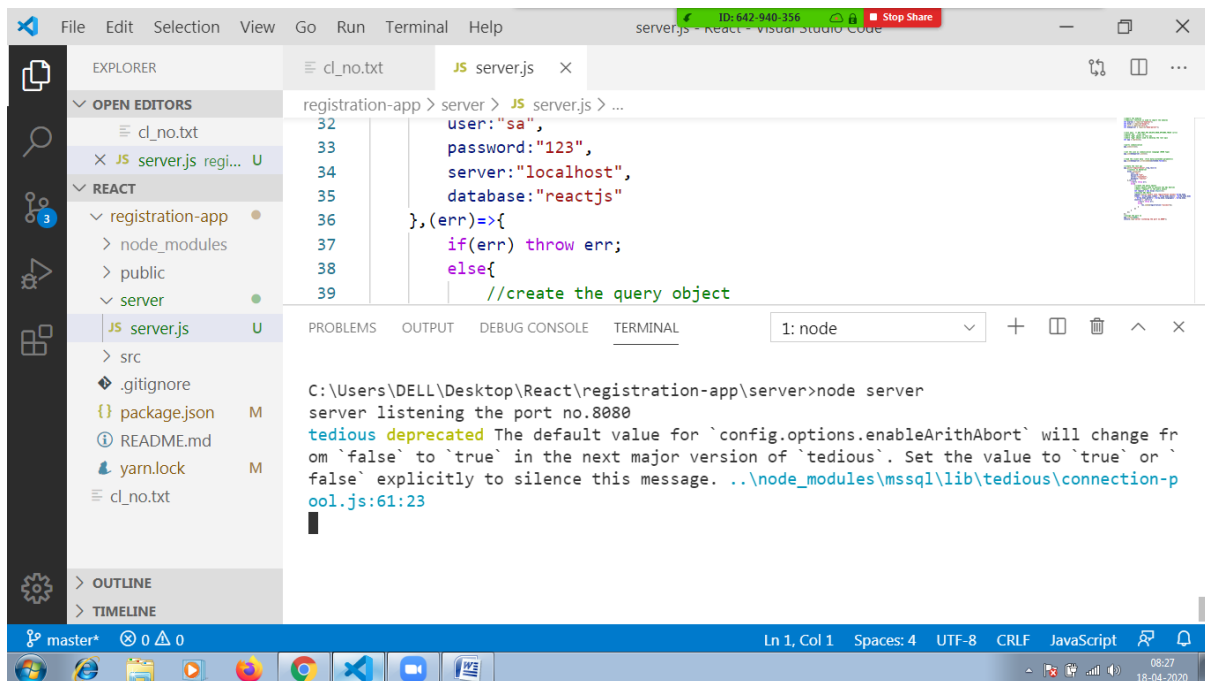
```
app.listen(8080);  
  
console.log("server listening the port no.8080");
```

Step 7) start the node server

```
> cd registration-app
```

```
> cd server
```

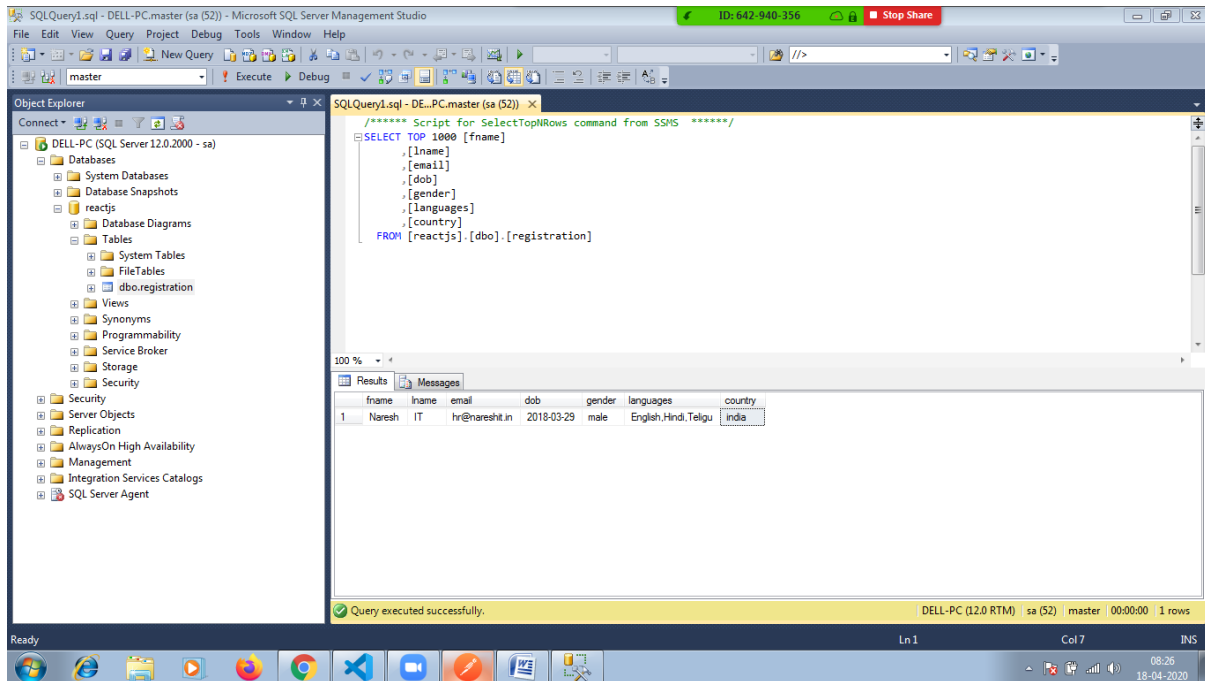
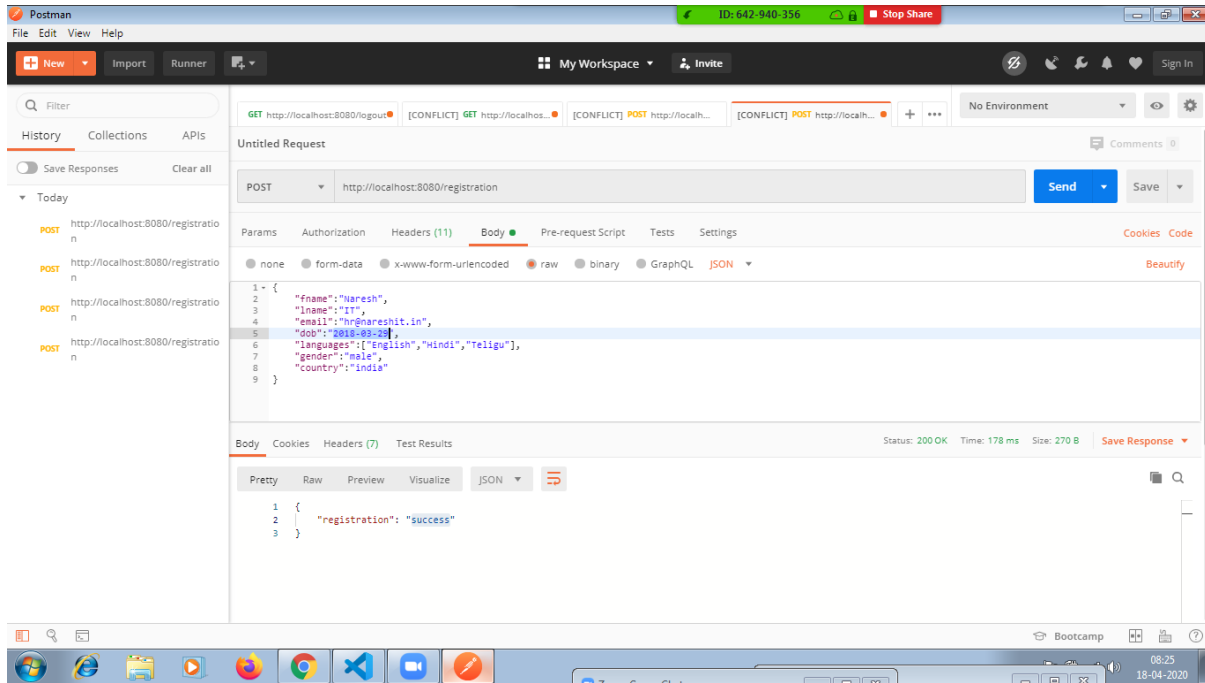
```
> node server
```



Step 8) test the following rest api by using Postman

```
=> http://localhost:8080/registration (POST)
```





## Step 9) download the bootstrap

Terminal-2

-----

```
> cd registration-app
```

```
> yarn add bootstrap -save
```

## Step 10) create the App Component

### App.js

```
import React,{ Component } from "react";

import "../node_modules/bootstrap/dist/css/bootstrap.min.css";

import axios from "axios";

export default class App extends React.Component{

  constructor(){

    super();

    this.state = {

      "country": "",

      "countryValid": false,
```



```
"Female":false,

"Male":false,

"genderValid":false,

"English":false,

"Hindi":false,

"Telugu":false,

"fname":"",

"lname":"",

"email":"",

"dob":"",

"dobValid":false,

"emailValid":false,

"fnameValid":false,

"lnameValid":false,

"formValid":false,

"languagesValid":false,

"formErrors":{"fname":"",

               "lname":"",

               "email":"",

               "dob":"",

               "languages":"",

               "gender":""},
```



```
        "country": ""}

    };

};

register = (event) => {

    let languages = [];

    let gender = "";

    for (let key in this.state) {

        if (key === "English" || key === "Telugu" || key ===
"Hindi") {

            if (this.state[key] === true) {

                languages.push(key);

            };

        } else if (key === "Male" || key === "Female") {

            if (this.state[key] === true) {

                gender = key

            };

        }

    }

}

let record = {

    "fname": this.state.fname,

    "lname": this.state.lname,
```

---



```
    "email":this.state.email,

    "dob":this.state.dob,

    "gender":gender,

    "languages":languages,

    "country":this.state.country

  };

  axios.post("http://localhost:8080/registration",record)

    .then((posRes)=>{

      console.log(posRes);

    },(errRes)=>{

      console.log(errRes);

    });

};

handleChange = (event)=>{

  let name = event.target.name;

  let value = event.target.value;

  switch(name) {

    case "English":
```





```
        this.setState(prevState=>({
            English : !prevState.English
        }),( )=>{this.validateForm(name,value)});

        break;

    case "Telugu":

        this.setState(prevState=>({
            Telugu : !prevState.Telugu
        }),( )=>{this.validateForm(name,value)});

        break;

    case "Hindi":

        this.setState(prevState=>({
            Hindi : !prevState.Hindi
        }),( )=>{this.validateForm(name,value)});

        break;

    case "Gender":

        if(value === "Male"){

            this.setState(prevState=>({
                Male : !prevState.Male
            }),( )=>{this.validateForm(name,value)});

        }else{

            this.setState(prevState=>({
                Female : !prevState.Female
```



```
        ) , () => { this.validateForm(name, value) } );  
    }  
    break;  
    default:  
        this.setState({  
            [name]: value  
        } , () => { this.validateForm(name, value) } );  
        break;  
    }  
};
```

```
validateForm(name, value) {  
    let l_countryValid = this.state.countryValid;  
    let l_genderValid = this.state.genderValid;  
    let l_fnameValid = this.state.fnameValid;  
    let l_lnameValid = this.state.lnameValid;  
    let l_emailValid = this.state.emailValid;  
    let l_dobValid = this.state.dobValid;  
    let l_languagesValid = this.state.languagesValid;  
    let l_formErrors = this.state.formErrors;  
    switch (name) {  
        case "country":
```



```
        l_countryValid = value.length<0;

        l_formErrors.country = l_countryValid ? "" : "select at least one country";

        break;

    case "Gender":

        l_genderValid = this.state.Male || this.state.Female;

        l_formErrors.gender = l_genderValid ? "" : "select gender";

        break;

    case "fname":

        l_fnameValid = value.length>6;

        l_formErrors.fname = l_fnameValid ? "" : "first name must be 6 characters";

        break;

    case "lname":

        l_lnameValid = value.length>3;

        l_formErrors.lname = l_lnameValid ? "" : "last name must be 3 characters";

        break;

    case "email":

        l_emailValid = value.match(/^([\w.%+-]+)@([\w-]+\.)([\w]{2,})$/i);;
```

---



```
        l_formErrors.email = l_emailValid ? "" : " email  
not matched the required pattern" ;  
  
        break;  
  
        case "dob":  
  
            l_dobValid= value.length>8;  
  
            l_formErrors.dob = l_dobValid ? "" : "please s  
elect dob";  
  
            break;  
  
        case "English":  
  
        case "Telugu":  
  
        case "Hindi":  
  
            l_languagesValid = this.state.English || this.s  
tate.Telugu || this.state.Hindi;  
  
            l_formErrors.languages = l_languagesValid?"":"s  
elect at least one language"  
  
            break;  
  
    };  
  
    this.setState({  
  
        fnameValid:l_fnameValid,  
  
        lnameValid:l_lnameValid,  
  
        emailValid:l_emailValid,  
  
        dobValid:l_dobValid,
```



```
        languagesValid:l_languagesValid,

        genderValid : l_genderValid,

        formErrors:l_formErrors

    },this.finalFun );

};


finalFun(){

    this.setState({

        formValid : this.state.fnameValid,

        formValid : this.state.lnameValid,

        formValid : this.state.dobValid,

        formValid : this.state.emailValid,

        formValid : this.state.languagesValid,

        formValid : this.state.genderValid

    })

};


render(){

    return(

        <div className="container mt-5">

            <div className="jumbotron" align-center >
```

---



```
<h3 className="text-center">Registration Form</h3>

<div className="row mt-4">

  <div className="col-md-4">

    <label>First Name</label>

  </div>

  <div className="col-md-8">

    <input type="text"

      name="fname"

      value={this.state.fname}

      placeholder="Enter First Name"

      required

      onChange={this.handleChange}></input>

  </div>

  <div>

    <h5 style={{color:"red"}}>{this.state.formErrors.fname}</h5>

  </div>

</div>

<div className="row">
```

---



```
<div className="col-md-4">

  <label>Last Name</label>

</div>

<div className="col-md-8">

  <input type="text"

    name="lname"

    value={this.state.lname}

    placeholder="Enter Last Name"

    required

    onChange={this.handleChange}></input>

  <div>

    <h5 style={{color:"red"}}>{this.state.formErrors.lname}</h5>

  </div>

</div>

<div className="row">

  <div className="col-md-4">

    <label>Email</label>

  </div>
```

---



```
<div className="col-md-8">

  <input type="email"

    name="email"

    value={this.state.email}

    placeholder="Enter Email"

    required

    onChange={this.handleChange}></input>

  <h5 style={{color:"red"}}>{this.state.formErrors.email}</h5>

</div>

</div>

<div className="row">

  <div className="col-md-4">

    <label>Date Of Birth</label>

  </div>

  <div className="col-md-8">

    <input type="date"

      name="dob"

      value={this.state.dob}
```





```
placeholder="Enter Date of Birth"

required

onChange={this.handleChange}></input>
put>

<h5 style={{color:"red"}}>{this.state.formErrors.dob}</h5>

</div>

</div>

<div className="row">

  <div className="col-md-4">

    <label>Languages</label>

  </div>

  <div className="col-md-8">

    <input type="checkbox"

      checked={this.state.English}

      name="English"

      value="English"

      onChange={this.handleChange}/> <b

>English</b>

    <input type="checkbox"

      className="ml-1"
```

---



```
        name="Telugu"

        value="Telugu"

        checked={this.state.Telugu}

        onChange={this.handleChange}/> <b
>Telugu</b>

        <input type="checkbox"

        className="ml-1"

        name="Hindi"

        value="Hindi"

        checked={this.state.Hindi}

        onChange={this.handleChange}/> <b
>Hindi</b>

        </div>

        <h5 style={{color:"red"}}>{this.state.formEr
rors.languages}</h5>

    </div>

    <br></br>

    <div className="row">

        <div className="col-md-4">

            <label>Gender</label>

        </div>

        <div className="col-md-8">
```



```
        <input type="radio"
              name="Gender"
              value="Male"
              checked={this.state.Male}
              onChange={this.handleChange}/> <b
>Male</b>

        <input type="radio"
              name="Gender"
              value="Female"
              checked={this.state.Female}
              onChange={this.handleChange}
              className="ml-2"/> <b>Female</b>

      </div>
    </div>

    <br></br><br></br>

    <div className="row">
      <div className="col-md-4">
        Country
      </div>
      <div className="col-md-8">
        <select name="country"
```

---



```
        value={this.state.country}

        onChange={this.handleChange}>

        <option value="india">India</option>

        <option value="usa">USA</option>

        <option value="uk">London</option>

    </select>

</div>

</div>

<br></br>

<div className="row">

    <div className="col-md-4"></div>

    <div className="col-md-8">

        <button className="btn btn-success btn-sm"

            disabled={!this.state.formValid}

            onClick={this.register}>Register</button>

    </div></div>

</div></div>

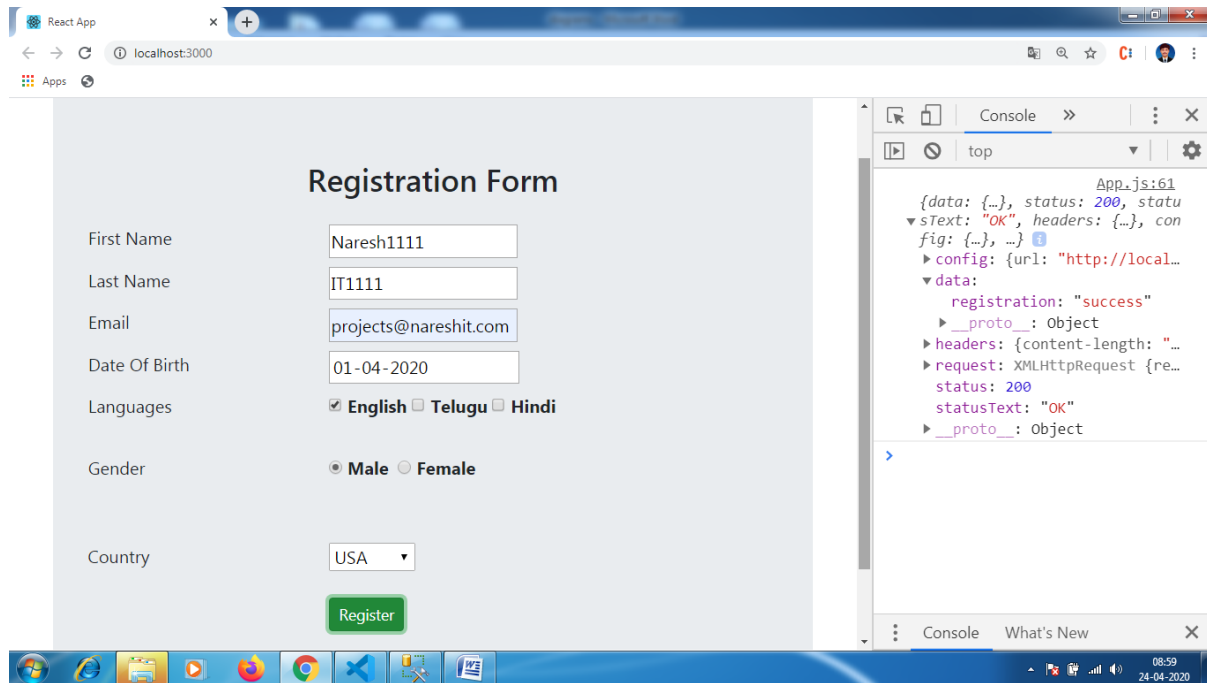
)

};

};
```

OutPut:





## (CRUD Operations-Server Files)

### (MYSQL-MSSQL-MONGODB)

1) install MySQL DataBase.

Ex.

```
mysql-essential-5.2.3-falcon-alpha-win32.msi
```

2) create the database in MySQL.

Queries:

Default Password : root

```
> show databases;
```

- it will show available databases.

```
> create schema crud;
```

- crud database will be created.

```
> use crud;
```

- we can switch to crud database.

```
> create table products(p_id integer,p_name  
varchar(20),p_cost integer);
```



- automatically products table will be created.

```
> insert into products values(111,'p_one',10000);
```

- automatically we can insert the record into database.

```
> select * from products;
```

- we can fetch the records from database.

\*\*\*\*\*

host : localhost

user : root

password : root

database : crud

table : products

\*\*\*\*\*

### 3) create the directory

Ex.

MySQL\_CRUD

### 4) switch to MySQL\_CRUD



```
> cd MySQL_CRUD
```

## 5) download the node modules

```
=> mysql
```

```
=> cors
```

```
=> body-parser
```

```
=> express
```

- express module used to develop the rest apis.

GET

POST

PUT

DELETE

HEAD

- cors module used to enable the ports communication.

- body-parser module used to read the client data.

- mysql module used to interact with the mysql database.

```
> yarn add mysql cors body-parser express --save
```

## 6) develop the rest apis by using NodeJS

```
=> GET      (Fetch the records from database)
```





=> POST      (Insert the record into database)

=> PUT        (Update the record in database)

=> DELETE    (Delete the record from database)

\*\*\*\*\*

## MySQL\_CRUD

config

db\_properties.js

db\_connection.js

fetch

fetch.js

insert

insert.js

update

update.js

delete

delete.js

server.js

\*\*\*\*\*

- "db\_properties.js" file used to maintain the database details.
- "db\_connection.js" file used to create and return the connection object.



- "fetch.js" file used to create the GET Request. (fetch the data from products table).
- "insert.js" file used to create the POST Request (insert the record into products table).
- "update.js" file used to create the PUT Request (update the record into products table).
- "delete.js" file used to create the DELETE Request (delete the record from products table).

### db\_properties.js

```
let obj = {  
  
  host : "localhost",  
  
  user : "root",  
  
  password : "root",  
  
  database : "crud"  
  
};
```

```
module.exports = obj;
```

### db\_connection.js

```
let obj = require("../db_properties");  
  
let mysql = require("mysql");  
  
let conn = {  
  
  getConnection : ()=>{  
  
    return mysql.createConnection(obj);  
  
  }  
  
}
```



```
};

module.exports = conn;

fetch.js

let conn = require("../config/db_connection");

let connection = conn.getConnection();

let fetch = require("express").Router().get("/", (req, res) => {

    connection.query(`select * from
products`, (err, records, fields) => {

        if (err) throw err;

        else {

            res.send(records);

        }

    });

});

module.exports = fetch;

insert.js

let conn = require("../config/db_connection");

let connection = conn.getConnection();

let insert = require("express").Router().post("/", (req, res) => {

    connection.query(`insert into products
values (${req.body.p_id}, '${req.body.p_name}', ${req.body.p_cost})
`,
```



```
(err,result)=>{

    if(err) throw err;

    else{

        res.send({insert:"success"});

    }

});

});

module.exports = insert;

update.js

let conn = require("../config/db_connection");

let connection = conn.getConnection();

let update = require("express").Router().put("/",(req,res)=>{

    connection.query(`update products set
p_name='${req.body.p_name}',p_cost=${req.body.p_cost} where
p_id=${req.body.p_id}` ,

        (err,result)=>{

            if(err) throw err;

            else{

                res.send({update : "success"});

            }

        });

});
```



```
});
```

```
module.exports = update;
```

### delete.js

```
let conn = require("../config/db_connection");
```

```
let connection = conn.getConnection();
```

```
let remove = require("express").Router().delete("/", (req, res) =>{
```

```
    connection.query(`delete from products where  
p_id=${req.body.p_id}`, (err, result) =>{
```

```
        if(err) throw err;
```

```
        else{
```

```
            res.send({delete:"success"});
```

```
        }
```

```
    });
```

```
});
```

```
module.exports = remove;
```

### server.js

```
let express = require("express");
```

```
let app = express();
```

```
let cors = require("cors");
```

```
let bodyparser = require("body-parser");
```

```
app.use(cors());
```

```
app.use(bodyparser.json());
```



```
app.use(bodyParser.urlencoded({extended:false}));  
app.use("/fetch",require("./fetch/fetch"));  
app.use("/insert",require("./insert/insert"));  
app.use("/update",require("./update/update"));  
app.use("/delete",require("./delete/delete"));  
app.listen(8080);  
console.log("server listening the port no.8080");
```

## 7) start the node server

```
> cd MySQL_CRUD  
 > node server
```

## 8) test the rest apis by using Postman.

```
=> http://localhost:8080/fetch          (GET)  
=> http://localhost:8080/insert        (POST)  
=> http://localhost:8080/update        (PUT)  
=> http://localhost:8080/delete        (DELETE)
```

\*\*\*\*\*



## MongoDB CRUD Operations

-----

### 1) create the directory

```
MongoDB_CRUD
```

### 2) switch to MongoDB\_CRUD

```
> cd MongoDB_CRUD
```

### 3) create the database

```
> mongod
```

```
- start the server
```

```
> mongo
```

```
- we will connect to server
```

```
> use crud;
```

```
- automatically crud database will be created.
```

```
> db.createCollection("products");
```

```
- products collection will be created
```

```
> db.products.insert({"p_id":111,  
                      "p_name":"p_one",  
                      "p_cost":10000});
```

```
- we can insert object into products collection.
```

```
> db.products.find();
```



- we can fetch the data from products collection.

\*\*\*\*\*

host : localhost

port : 27017

protocol : mongodb

database : crud

collection : products

\*\*\*\*\*

#### 4) download the node modules

=> express

=> mongodb@2.2.32

=> cors

=> body-parser

- "express" module used to develop the rest apis.
  - GET
  - POST
  - PUT
  - DELETE
- "mongodb@2.2.32" module used to interact with the mongodb database.
- "cors" module used to enable the ports communication.





- "body-parser" module used to read the client data.

```
> yarn add express mongodb@2.2.32 cors body-parser --save
```

## 5) develop the rest apis by using NodeJS

\*\*\*\*\*

MongoDB\_CRUD

fetch

fetch.js

insert

insert.js

update

update.js

delete

delete.js

server.js

\*\*\*\*\*

- fetch.js file used to create the GET Request (fetch the data from products collection)
- insert.js file used to create the POST Request (insert the record into products collection)
- update.js file used to create the PUT Request (update the record present in products collection)



- delete.js file used to create the DELETE Request (delete the record from products collection)
- server.js file used to collabrate the modules.

### fetch.js

```
let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let fetch = require("express").Router().get("/", (req, res) => {

  nareshIT.connect("mongodb://localhost:27017/crud", (err, db) => {

    if(err) throw err;

    else{

      db.collection("products").find().toArray((err, array) => {

        if(err) throw err;

        else{

          res.send(array);

        }

      })

    }

  });

});

module.exports = fetch;
```

---



insert.js

```
let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let insert = require("express").Router().post("/", (req, res) =>{

nareshIT.connect("mongodb://localhost:27017/crud", (err, db) =>{

    if(err) throw err;

    else{

db.collection("products").insertOne({"p_id": req.body.p_id,

"p_name": req.body.p_name,

"p_cost": req.body.p_cost}, (err, result) =>{

        if(err) throw err;

        else{

            res.send({insert : "success"});

        }

    });

}

});

});

module.exports = insert;
```

---



### update.js

```
let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let update = require("express").Router().put("/", (req, res) => {

  nareshIT.connect("mongodb://localhost:27017/crud", (err, db) => {

    if (err) throw err;

    else {

      db.collection("products").updateOne({ "p_id": req.body.p_id },

        { $set: { "p_name": req.body.p_name, "p_cost": req.body.p_cost } },

        (err, result) => {

          if (err) throw err;

          else {

            res.send({ update : "success" });

          }

        });

    }

  });

});
```

---



```
module.exports = update;
```

### delete.js

```
let mongodb = require("mongodb");

let nareshIT = mongodb.MongoClient;

let remove = require("express").Router().delete("/", (req, res) =>{

nareshIT.connect("mongodb://localhost:27017/crud", (err, db) =>{

    if(err) throw err;

    else{

db.collection("products").deleteOne({"p_id": req.body.p_id}, (err,
result) =>{

        if(err) throw err;

        else{

            res.send({delete : "success"});

        }

    });

}

});});

module.exports = remove;
```

### server.js

```
let express = require("express");
```

---



```
let app = express();

let cors = require("cors");

let bodyparser = require("body-parser");

app.use(cors());

app.use(bodyparser.json());

app.use(bodyparser.urlencoded({extended: false}));

app.use("/fetch", require("./fetch/fetch"));

app.use("/insert", require("./insert/insert"));

app.use("/update", require("./update/update"));

app.use("/delete", require("./delete/delete"));

app.listen(8080);

console.log("server listening the port no.8080");
```

## 6) start the server

```
> node server
```

## 7) test the rest apis by using Postman

```
=> http://localhost:8080/fetch    (GET)

=> http://localhost:8080/insert    (POST)

=> http://localhost:8080/update    (PUT)

=> http://localhost:8080/delete    (DELETE)
```

\*\*\*\*\*



## SQLServer CRUD Operations

-----

### 1) install SQLServer

Ex.

SQL Server 2014 Management Studio

### 2) create the database

\*\*\*\*\*

user : sa

password : 123

database : crud

table : products

\*\*\*\*\*

### 3) create the directory (MSSQL\_CRUD)

### 4) switch to MSSQL\_CRUD

> cd MSSQL\_CRUD

### 5) download node modules

=> express

=> cors

=> body-parser

=> mssql@6.0.1



```
> yarn add express cors body-parser mssql@6.0.1 --save
```

## 6) create the rest apis

```
*****
```

```
MSSQL_CRUD
```

```
    config
```

```
        db_properties.js
```

```
    fetch
```

```
        fetch.js
```

```
    insert
```

```
        insert.js
```

```
    update
```

```
        update.js
```

```
    delete
```

```
        delete.js
```

```
    server.js
```

```
*****
```

db properties.js

```
let obj = {
```

```
    server : "localhost",
```





```
    user    : "sa",

    password : "123",

    database : "crud"

};

module.exports = obj;

fetch.js

let obj = require("../config/db_properties");

let mssql = require("mssql");

let fetch = require("express").Router().get("/", (req, res) => {

    mssql.connect(obj, (err) => {

        if (err) throw err;

        else {

            let request = new mssql.Request();

            request.query(`select * from
products`, (err, records) => {

                if (err) throw err;

                else {

                    res.send(records.recordset);

                }

                mssql.close();

            });

        }

    });

}
```

---



```
    });  
  
  });  
  
module.exports = fetch;  
  
insert.js  
  
let obj = require("../config/db_properties");  
  
let mssql = require("mssql");  
  
let insert = require("express").Router().post("/", (req, res) => {  
  mssql.connect(obj, (err) => {  
    if (err) throw err;  
  
    else {  
      let request = new mssql.Request();  
  
      request.query(`insert into products  
values (${req.body.p_id},  
  
`${req.body.p_name}`,  
  
`${req.body.p_cost})`, (err, result) => {  
        if (err) throw err;  
  
        else {  
          res.send({ insert : "success" });  
        }  
  
        mssql.close();  
      }  
    }  
  }  
});
```



```
        });  
    }  
});  
  
});  
  
module.exports = insert;  
  
update.js  
  
let obj = require("../config/db_properties");  
  
let mssql = require("mssql");  
  
let update = require("express").Router().put("/", (req, res) => {  
    mssql.connect(obj, (err) => {  
        if (err) throw err;  
        else {  
            let request = new mssql.Request();  
            request.query(`update products set  
p_name='${req.body.p_name}', p_cost=${req.body.p_cost} where  
p_id=${req.body.p_id}` ,  
  
                (err, result) => {  
                    if (err) throw err;  
                    else {  
                        res.send({update : "success"});  
                    }  
                }  
            mssql.close();  
        }  
    }  
});
```



```
        });  
    }  
});  
  
module.exports = update;  
  
delete.js  
  
let obj = require("../config/db_properties");  
  
let mssql = require("mssql");  
  
let remove = require("express").Router().delete("/", (req, res) => {  
    mssql.connect(obj, (err) => {  
        if (err) throw err;  
        else {  
            let request = new mssql.Request();  
            request.query(`delete from products where  
p_id=${req.body.p_id}`, (err, result) => {  
                if (err) throw err;  
                else {  
                    res.send({delete: "success"});  
                }  
                mssql.close();  
            });  
        }  
    });  
}
```

---



```
    });  
  
});  
  
module.exports = remove;  
  
server.js  
  
let express = require("express");  
  
let app = express();  
  
let cors = require("cors");  
  
let bodyparser = require("body-parser");  
  
app.use(cors());  
  
app.use(bodyparser.json());  
  
app.use(bodyparser.urlencoded({extended: false}));  
  
app.use("/fetch", require("./fetch/fetch"));  
  
app.use("/insert", require("./insert/insert"));  
  
app.use("/update", require("./update/update"));  
  
app.use("/delete", require("./delete/delete"));  
  
app.listen(8080);  
  
console.log("server listening the port no.8080");
```

## 7) start the server

```
> cd MSSQL_CRUD  
  
> node server
```



8) test the following urls by using Postman

=> `http://localhost:8080/fetch` (GET)

=> `http://localhost:8080/insert` (POST)

=> `http://localhost:8080/update` (PUT)

=> `http://localhost:8080/delete` (DELETE)

