What is ReactJS?

ReactJS is an open-source JavaScript library used to create single page application

React is a JavaScript library created by Facebook.

Why we use ReactJS?

- It uses virtual DOM (JavaScript object), which improves the performance of the app.
- The JavaScript virtual DOM is faster than the regular DOM.

Framework - it is bootstrapped with nearly every single thing that you'll need to make a complete, large-scale app.

REACT PROJECT SET UP

1. Node js install (32bit/64 bit)
https://nodejs.org/en/download

2. Check node version
node-v

3. Create React Project

npx create-react-app <Enter Project Name> or
npm init react-app <Enter Project Name> or
a) npm install -g create-react-app

b) create-react-app -version

c) create-react-app <Enter Project Name>

4. Open the folder In vs code

5.Npm start

NPM – or "Node Package Manager" – is the default package manager, which will manage the packages which we will install

Package.json - gives package information and describe the project.

It keeps the track of all the version of packages

Package-lock.json - This file describes the exact versions of the dependencies used in an npm JavaScript project.

gitIgnore - /node-modules

- JSX stands for JavaScript XML. It is a JavaScript syntax extension. Its an XML or HTML like syntax used by ReactJS.

Allows us to write html and js in the component

React Fragments

A common pattern in React is for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.

We know that we make use of the render method inside a component whenever we want to render something to the screen. We may render a single element or multiple elements, though rendering multiple elements will require a **'div'** tag around the content as the render method will only render a single root node inside it at a time.

Ex:

```
Const App = () => {
        return (
            <div>
                <h2>Hello</h2>
                <p>How you doin'?</p>
            </div>
        );
}

export default App;
```

Components

A **Component** is one of the core building blocks of React. In other words, we can say that every application you will develop in React will be made up of pieces called components.

Components are independent and reusable bits of code.

2 types of component

1. Functional component
2. Class component

VIRTUAL DOM

When anything new is added to the application, a virtual DOM is created and it is represented as a tree. Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, a new Virtual DOM tree is created. This new Virtual DOM tree is then compared with the previous Virtual DOM tree and make a note of the changes. After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again.

PROPS

Props are arguments passed into React components. Props are passed to components via HTML attributes. props stands for properties. React Props are like function arguments in JavaScript and attributes in HTML. To send props into a component, use the same syntax as HTML attributes

Export

1. Default Export

2. Named Export

What are Hooks?

Hooks are used to give functional components an access to use the states and are used to manage side-effects in React. They were introduced React 16.8. They let developers use state and other React features

Types of Hooks

1. usestate

2. Useeffect

3. usememo
4. useref
5. usecontext
6. redux

React useState Hook

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

To use the `useState` Hook, we first need to `import` it into our component.

const [state, setState] = useState(initialState);
Ex:

1. Counter (number)
2. Checkbox (boolean)
3. Form (two variables)

USESTATE

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


function App() {

  const [fullName, setFullName] = useState("");

  const [selectBox, setSelectBox] = useState("");

  const [checkBox, setCheckBox] = useState(false);
```

```
  return (

    <div>

      <input  placeholder="Enter Full name" className="inputfeild"  value={fullName}

        onChange={(e) => {

          setFullName(e.target.value);

        }}/>
```

```
      <select  value={selectBox}  onChange={(e) => {

          setSelectBox(e.target.value);

        }}>

        <option value={""}>Choose option</option>

        <option value={"ec"}>Ec branch</option>

        <option value={"cs"}>cs branch</option>

        <option value={"mech"}>mech branch</option>

        <option value={"civil"}>civil branch</option>

      </select>
      <span>
      <input
        type="checkbox"
        checked={checkBox}
        onChange={(e) => {
          //setCheckBox(e.target.checked);
          setCheckBox(!checkBox)
        }}
      /> available</span>
```

```
      <div>
      <p>Entered name is {fullName}</p>
      <p>branch selected: {selectBox}</p>
      <p>{checkBox ? " Available" : " Not Available"}</p>

      </div>
     </div>
  );
}
export default App;
```

CSS in REACT

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

function App() {
  const applycss = {
    backgroundColor: "red",
    width: "200px",
    height: "200px",

    marginLeft: "10px",

  };

  return (

    <div>

      <div style={applycss}></div>

      <div

        style={{

          backgroundColor: "red",

          width: "200px",

          height: "200px",

          marginLeft: "10px",

        }}

      ></div>

    </div>

  );

}

export default App;
```

PROPS

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

import Card from "./card";

function App() {

  return (

    <div>

      <div style={{ display: "flex", flexWrap: "wrap" }}>

      <Card name="mobile" picture="https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180"/>

      <Card name="car" picture="https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180"/>

      <Card name="bike" picture="https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180"/>

      <Card name="laptop" picture="https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180"/>

    </div>
    </div>
  );
}
export default App;
```

```
import React from 'react'

const card = (props) => {
  return (
    <div>
        <div className="card">
        <img src={props.picture} alt="No image Found" />
        <h4>{props.name}</h4>
      </div>
    </div>
  )
}
export default card
```

```
import React, { useState } from "react";
import Card from "./card";

function App() {
  const jsonResponse = [
    {
      img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",
      name: "mobile",
    },
    {
      img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",
      name: "laptop",
    },
    {
      img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",
      name: "bike",
    },
    {
      img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",
      name: "car",
    },
    {
      img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",
      name: "AC",
    },
  ];
```

```
  return (
    <div>
      <div style={{ display: "flex", flexWrap: "wrap" }}>

```

```
        {jsonResponse.map((item, index) => {
          return <Card name={item.name} picture={item.img} />;
        })}
      </div>
    </div>
  );
}
export default App;
```

Export Named

```
export function phoneValidation(value){
    const regex = /^[7-9][0-9]{9}$/
    return regex.test(value)
}

export function passwordValidaion(value) {
    return value.length > 6
}
```

```
import { useState } from "react";

import {phoneValidation, passwordValidaion} from './utils'


const App = () => {

  const [number, setNumber] = useState("")
```

```
  function handleFunc(){

    let phoneValidations = phoneValidation(number)

    console.log(phoneValidations, 'phoneValidation')

```

```
  }

  return (

    <div>

      <input type="number" value={number} onChange={(e)=> setNumber(e.target.value)} />

   <button type="button" onClick={()=> handleFunc()}>Submit</button>

    </div>

  )

}
```

```
export default App;
```

Package Toastify

```
import { useState } from "react";

import { phoneValidation, passwordValidaion } from "./utils";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";


const App = () => {

  const [number, setNumber] = useState("");
  function handleFunc() {

    let phoneValid = phoneValidation(number);

    if(!phoneValid){

      toast.error("Please enter valid phone number")

      return;

    }

  }

  return (

    <div>

      <ToastContainer />

      <input

        type="number"

        value={number}

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

      />

      <button type="button" onClick={() => handleFunc()}>

        Submit

      </button>

    </div>

  );

};
```

```
export default App;
```

API Call & UseEffect

```
import { useEffect, useState } from "react";

import { phoneValidation, passwordValidaion } from "./utils";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";

const App = () => {

  const [number, setNumber] = useState("");

  const [userInfo, setUserInfo] = useState({});
  async function callingGetApi() {

    let response = await fetch("https://dummyjson.com/products/1");

    let user = await response.json();

    setUserInfo(user);

  }

 console.log("render ");

 useEffect(() => {

    callingGetApi();

    console.log("useeffect ");

  }, []);
```

```
  return (

    <div>

      <ToastContainer />

      <input   type="number"  value={number}    onChange={(e) => setNumber(e.target.value)}

 />
```

```
      {userInfo.brand}

      {userInfo.category}

    </div>

  );

};
```

```
export default App;
```

useEffect

The `useEffect` Hook allows you to perform side effects in your components.

1. No dependency passed:

```
useEffect(() => {
//Runs on every render
});
```

2. An empty array:

```
useEffect(() => {
```

```
//Runs only on the first render
}, []);
```

3. Props or state values:

```
useEffect(() => {
//Runs on the first render
//And any time any dependency value changes
}, [state]);
```

Ex:

```
useEffect(() => {

   console.log("useeffect ");

 }, [number]);
```

Routes

```
import React from "react";

import Section from "./section";

import AboutUs from "./about";

import Contact from "./contact";

import PageNotFound from './pageNotFound'

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


const App = () => {

  return (

    <div>

      <BrowserRouter>

        <Routes>

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

          <Route path="/aboutus" element={<AboutUs />}></Route>

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

          <Route path="*" element={<PageNotFound />}></Route>

        </Routes>

      </BrowserRouter>

    </div>

  );
};
export default App;
```

```
import React from 'react'

const section = () => {
  return (
    <div>section</div>
  )
}
export default section
```

```
import React from 'react'
const about = () => {
  return (
    <div>about</div>
  )
}
export default about
```

```
import React from 'react'
const contact = () => {
  return (
    <div>contact</div>
  )
}
export default contact
```

Page Not Found

```
import pageNotFoundImg from "./page-not-found.jpg";
const PageNotFound = () => {
  return (
    <div style={{ textAlign: "center" }}>
      <img src={pageNotFoundImg} style={{ width: "100px", height: "100px" }} />
      <h5 style={{ margin: 0 }}> page not found</h5>
    </div>
  );
};

export default PageNotFound;
```

LINK

```
import { Link } from "react-router-dom";
const header = () => {
  const applyCss = {display:'flex', justifyContent:'space-between', alignItems: 'center'}
  return (
    <div style={applyCss}>
      <div><h3>Header</h3></div>
      <div style={applyCss}>
          <p style={{margin:'0 20px'}}><Link to="/aboutus">About Us </Link></p>
          <p  style={{margin:'0 20px'}}><Link to="/contact">Contact </Link></p>
      </div>
    </div>
  )

}
export default header;
```

Navigate

```

import React from 'react';
import {useNavigate} from "react-router-dom"

const Section = () => {
  const navigate = useNavigate();

  return (
  <>
     <button onClick={()=>navigate('/')}>Go Back Home</button>
  </>
  )
};

export default Section;
```

Passing Props from child to parent component

```
import React from 'react';
import Menulist from './menulist';
import PageNotFound from './pageNotFound';
import MenuDescription from './menuDescription';
import { BrowserRouter, Routes, Route } from 'react-router-dom'

const App = () => {
  return (
    <div>
      <BrowserRouter>
      <Routes>
      <Route path='/' element={<Menulist />}></Route>
        <Route path='/menu-description/:id' element={<MenuDescription />}></Route>
        <Route path='*' element={<PageNotFound />}></Route>
      </Routes>
      </BrowserRouter>

    </div>
  )
}
```

```
export default App
```

Menulist.js

```
import React from 'react';

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

import {jsonResponse} from './utils'

import Card from './card'



const Menulist = () => {


  const navigate = useNavigate();
```

```
    const handleSelectedCard = (id) => {

    console.log(id, "selectedID")

    navigate(`./menu-description/${id}`)

    }


  return (

   <div>

     <div>menulist</div>

     <div style={{display:'flex', flexWrap:'wrap'}}>

     {jsonResponse.map((item, index)=> {

        return <Card key={item.id} id={item.id} img={item.img} name={item.name} selectedCard={handleSelectedCard}/>

    })}

     </div>


   </div>

  )

}
```

```
export default Menulist
```

Menu description

```
import React, { useState, useEffect } from "react";
import { useParams } from "react-router-dom";
import { jsonResponse } from "./utils";

const MenuDescription = () => {
  const params = useParams();
  const [description, setDescription] = useState([]);
  console.log(window.location, "location", params);
  let selectedId = window.location.href.split("/");
  selectedId = selectedId[4];
  useEffect(() => {
    const result = jsonResponse.filter((item) => {
      return item.id == selectedId;
    });
    setDescription(result);
  }, []);
```

```
  return (
    <div>
      <h2>menuDescription</h2>
      {description.map((item) => {
        return (
          <div style={{ textAlign: "center" }}>
            <img src={item.img} style={{ width: "80%" }} />
            <p>{item.description}</p>
          </div>
        );
      })}
    </div>
  );
};
```

```
export default MenuDescription;
```

Card component

```
const Card = (props) => {

  return (

    <div className="card" onClick={()=> {props.selectedCard(props.id)}}>

      <img src={props.img} />

      <p>{props.name}</p>

    </div>

  )



}
```

```
export default   Card;
```

JSON Response

```
export  const jsonResponse = [

    {

     img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",

     name: "mobile",

     id: 1,

     description: "A mobile phone is a wireless handheld device that allows users to make and receive calls. While the earliest generation of mobile phones could only make and receive calls, today’s mobile phones do a lot more, accommodating web browsers, games, cameras, video players and navigational systems. The first mobile phones, as mentioned, were only used to make and receive calls, and they were so bulky it was impossible to carry them in a pocket. These phones used primitive RFID and wireless systems to carry signals from a cabled PSTN endpoint. Later, mobile phones belonging to the Global System for Mobile Communications (GSM) network became capable of sending and receiving text messages. As these devices evolved, they became smaller and more features were added, such as multimedia messaging service (MMS), which allowed users to send and receive images"

    },

    {

     img: "https://images-na.ssl-images-amazon.com/images/I/812NShN3MpL._SL1500_.jpg",

     name: "laptop",

     id: 2,

     description: "A mobile phone is a wireless handheld device that allows users to make and receive calls. While the earliest generation of mobile phones could only make and receive calls, today’s mobile phones do a lot more, accommodating web browsers, games, cameras, video
```

```
players and navigational systems. The first mobile phones, as mentioned, were only used to
make and receive calls, and they were so bulky it was impossible to carry them in a pocket.
These phones used primitive RFID and wireless systems to carry signals from a cabled PSTN
endpoint. Later, mobile phones belonging to the Global System for Mobile Communications (GSM)
network became capable of sending and receiving text messages. As these devices evolved, they
became smaller and more features were added, such as multimedia messaging service (MMS), which
allowed users to send and receive images"
```

```
    },
    {
     img: "https://wallpapercave.com/wp/wp1925045.jpg",
     name: "bike",
     id: 3,
     description: "A mobile phone is a wireless handheld device that allows users to make and
receive calls. While the earliest generation of mobile phones could only make and receive
calls, today’s mobile phones do a lot more, accommodating web browsers, games, cameras, video
players and navigational systems. The first mobile phones, as mentioned, were only used to
make and receive calls, and they were so bulky it was impossible to carry them in a pocket.
These phones used primitive RFID and wireless systems to carry signals from a cabled PSTN
endpoint. Later, mobile phones belonging to the Global System for Mobile Communications (GSM)
network became capable of sending and receiving text messages. As these devices evolved, they
became smaller and more features were added, such as multimedia messaging service (MMS), which
allowed users to send and receive images"
    },
    {
     img: "http://wallup.net/wp-content/uploads/2016/07/21/397775-car-sports_car-Super_Car-
nature-landscape-road-clouds-Audi-Audi_R8-field-mountains-hills-sun_rays-plants-blue_cars-
vehicle.jpg",
     name: "car",
     id: 4,
     description: "A mobile phone is a wireless handheld device that allows users to make and
receive calls. While the earliest generation of mobile phones could only make and receive
calls, today’s mobile phones do a lot more, accommodating web browsers, games, cameras, video
players and navigational systems. The first mobile phones, as mentioned, were only used to
make and receive calls, and they were so bulky it was impossible to carry them in a pocket.
These phones used primitive RFID and wireless systems to carry signals from a cabled PSTN
endpoint. Later, mobile phones belonging to the Global System for Mobile Communications (GSM)
network became capable of sending and receiving text messages. As these devices evolved, they
became smaller and more features were added, such as multimedia messaging service (MMS), which
allowed users to send and receive images"
```

```
    },
    {
     img: "https://assets.hardwarezone.com/img/2014/02/LG_Inverter_AC_2.jpg",
     name: "AC",
     id: 5,
     description: "A mobile phone is a wireless handheld device that allows users to make and receive calls. While the earliest generation of mobile phones could only make and receive calls, today’s mobile phones do a lot more, accommodating web browsers, games, cameras, video players and navigational systems. The first mobile phones, as mentioned, were only used to make and receive calls, and they were so bulky it was impossible to carry them in a pocket. These phones used primitive RFID and wireless systems to carry signals from a cabled PSTN endpoint. Later, mobile phones belonging to the Global System for Mobile Communications (GSM) network became capable of sending and receiving text messages. As these devices evolved, they became smaller and more features were added, such as multimedia messaging service (MMS), which allowed users to send and receive images"
```

```
    },

   ];
```

useRef

useRef is a React Hook that lets you reference a value that’s not needed for rendering.

The useRef is a hook that allows to directly create a reference to the DOM element in the functional component

const ref = useRef(initialValue)

//Ex:

```
import React ,  {useRef} from 'react'


const App = () => {

    const inputText = useRef(null);

    function handleFunc() {

        inputText.current.value = "learning useref..."

        inputText.current.focus();

        inputText.current.style.color = "red"
```

```
    }

  return (

    <div>

        <input type='text' ref={inputText} />

        <button type='button' onClick={handleFunc}>Submit</button>

    </div>
  )
}
```

```
export default App
```

useMemo

useMemo is a React Hook that lets you cache the result of a calculation between re-renders.

const cachedValue = useMemo(calculateValue, dependencies)

Ex:

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


const App = () => {
  const [number, setNumber] = useState(0)
  const [counter, setCounter] = useState(0)
  const squareOfNumber = handleSquareNumber(number)
  const squareOfNumber = useMemo(()=> {
    return handleSquareNumber(number)
      }, [number])
```

```
  function handleSquareNumber(number){
    console.log("handleSquareNumber function is calling")
    return number * number;
  }
```

```
  return (
    <div>
      <input type='number' value={number} onChange={(e)=> setNumber(e.target.value)} />
      <p>Square Number: {squareOfNumber}</p>
      <button type='button' onClick={()=> setCounter(counter + 1)}> Counter</button>
```

```
     <p>Counter: {counter}</p>

    </div>

  )

}
```

```
export default App
```

useContext

useContext is a React Hook that lets you read and subscribe to context from your component.

const value = useContext(SomeContext)

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

import Header from './header'

//import Footer from "./footer"

export const themeContext = React.createContext()


const Demo = () => {

  const [toggleBtn, setToggleBtn] = useState(false)

  return (

    <div>

      <themeContext.Provider value={toggleBtn}>

      <button type='button' onClick={()=> setToggleBtn(!toggleBtn)}>

        {toggleBtn ? " Dark Mode" : " Light Mode"}</button>

      <Header />

     </themeContext.Provider>
```

```
    </div>

  )

}
```

```
export default Demo
```

Child component

```
import React, {useContext} from 'react'

import {themeContext} from "./demo"

const Header = () => {

const toggleValue = useContext(themeContext)

  return (

    <div style={toggleValue ?

      {width:'100px', height:'100px', border:'1px solid #000', margin:'5px', backgroundColor:'#fff', color:'#000'}

      : {width:'100px', height:'100px', border:'1px solid #000', margin:'5px', backgroundColor:'#000', color:'#fff'}}>Header</div>

  )

}
```

```
export default Header
```

Higher-order components

In React, a higher-order component is a function that takes a component as an argument and returns a new component that wraps the original component.

HOC are a powerful feature of the React library. They allow you to reuse component logic across multiple components.

Menu List

```
import React, { useMemo, useState } from "react";

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

import { jsonResponse } from "./utils";

import Card from "./card";

import { HigerOrderComponent } from "./components/HigerOrderComponent";
```

```
const Menulist = () => {
  const navigate = useNavigate();

  const handleSelectedCard = (id) => {
    console.log(id, "selectedID");
    navigate(`./menu-description/${id}`);
  };
  const [name, setName] = useState("")

  const BestSellerComponent = HigerOrderComponent(Card)

  return (
    <div>
      <div style={{ display: "flex", flexWrap: "wrap" }}>
        {jsonResponse.map((item, index) => {
          return (
            <div key={item.id}>
              {item.bestSeller ? <BestSellerComponent
               id={item.id}
               img={item.img}
               name={item.name}
               selectedCard={handleSelectedCard}
                /> :  <Card
                  id={item.id}
                  img={item.img}
                  name={item.name}
                  selectedCard={handleSelectedCard}
                /> }
            </div>
          );
        })}
      </div>
    </div>
  );
};
```

```
export default Menulist;
```

Higher order function

```
export const HigerOrderComponent = (BestSellerComponent) => {
  return(props) => {
    console.log(props, 'props...')
    return (
        <div style={{position:'relative'}}>
            <p className="best-seller-tag">Best seller</p>
            <BestSellerComponent
            {...props}/>
        </div>
      )
  }
}
```

Json Response

```
export  const jsonResponse = [
    {
     img: "https://tse3.mm.bing.net/th?id=OIP.vbHfsrK2IfWE1ZT9NMw5PgHaE7&pid=Api&P=0&h=180",
     name: "mobile",
     id: 1,
     bestSeller: true,
     description: "decription"
    },
    {
     img: "https://images-na.ssl-images-amazon.com/images/I/812NShN3MpL._SL1500_.jpg",
     name: "laptop",
     id: 2,
     bestSeller: false,
     description: "decription"
    },
    {
     img: "https://wallpapercave.com/wp/wp1925045.jpg",
     name: "bike",
     id: 3,
     bestSeller: false,
     description: "decription"
    },
    {
     img: "http://wallup.net/wp-content/uploads/2016/07/21/397775-car-sports_car-Super_Car-nature-landscape-road-clouds-Audi-Audi_R8-field-mountains-hills-sun_rays-plants-blue_cars-vehicle.jpg",
     name: "car",
     id: 4,
     bestSeller: true,
     description: "decription"
```

```
    },
    {
     img: "https://assets.hardwarezone.com/img/2014/02/LG_Inverter_AC_2.jpg",
     name: "AC",
     id: 5,
     description: "decription",
     bestSeller: false,
    },
   ];
```

Css

```
.best-seller-tag {
  position: absolute;
  top: 0;
  left: 10px;
  margin: 0;
  background-color: #000;
  color: #fff;

  padding: 5px;

  border-radius: 2px;

}
```

REDUX

its core building blocks work, such as store, actions, and reducers and how they all come together and make Redux the global state management library.

Why Redux ?

To pass data from one component to another in a large application and facilitate debugging, it can become challenging. That's why global state management was introduced.

store

```
import { configureStore } from "@reduxjs/toolkit";

import CartSlice from "./CartSlice"

const AppStore = configureStore({

    reducer: {

        cart: CartSlice,

        profile: [],

    }

})


export default AppStore;
```

Cartslice

```
import { createSlice } from "@reduxjs/toolkit";
```

```
const CartSlice = createSlice({
    name: "cart",
    initialState: {
        cartState: []
    },
    reducers: {
        addItem: (state, action) => {
            state.cartState.push(action.payload)
        }
    }
})
export const {addItem} = CartSlice.actions;
export default CartSlice.reducer;
```

Header

```
import React from "react";
import { FaShoppingCart } from "react-icons/fa";
import { useSelector } from "react-redux";

const Header = (props) => {
  const cartInfo = useSelector((store)=> store.cart.cartState)

  return (
    <div className="d-flex justify-content-between">
      <p>Header {props.id}</p>

      <div className="d-flex justify-content-between w-50">
        <div>
        <FaShoppingCart style={{margin:'5px'}} />
        <span className="cartvalue">{cartInfo.length}</span>
        </div>
      </div>
    </div>
  );
};
```

```
export default Header;
```

Card

```
import Button from 'react-bootstrap/Button';
import { useDispatch } from 'react-redux';
import { addItem } from './redux/CartSlice';
const Card = (props) => {
  const dispatch = useDispatch()
  const addItemToStore = (info) => {
    dispatch(addItem(info))
  }

  return (
    <div className="card">
      <div onClick={()=> props.selectedCard(props.id)}>
      <img src={props.imgs} alt='' style={{width:'100%', height: '170px', margin:'10px 0'}} />
      <p>{props.name}</p>
      </div>
      <Button variant="primary" onClick={()=> addItemToStore(props)}>Add To Cart</Button>
    </div>
  )

}
```

```
export default    Card;
```

Custom Hooks

```
import React from "react";
import UseApi from "./usehandleApiCall";
```

```
function Demo() {
  const { data, error } = UseApi("https://dummy.restapiexample.com/api/v1/employees","GET");

  if (error) {
    return <p>Error: {error.message}</p>;
  }

  return <div></div>;
}

export default Demo;
```

usehandleApiCall

```
import { useState, useEffect } from "react";

function UsehandleApiCall(url, type, body) {
  const [data, setData] = useState(null);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchData();
  }, [url]);

  const fetchData = async () => {
    try {
      const response = await fetch(url, {
        method: type,
        body,
      });
      const result = await response.json();
      setData(result);
    } catch (error) {
      setError(error);
    }
  };
```

```
  return { data, error };

}
```

```
export default UsehandleApiCall;
```

Object-Oriented Programming(OOP)

Object-Oriented Programming(OOP) is a programming paradigm(pattern) based on the concepts of Objects.

Object-oriented programming (OOP) is a programming paradigm used by developers to structure software applications into reusable pieces of code and blueprints using objects

1. Object

2. Class

3. Encapsulation

4. Inheritance

5. polymorphism

Object

Objects are like real-life entities. They have their properties and methods.

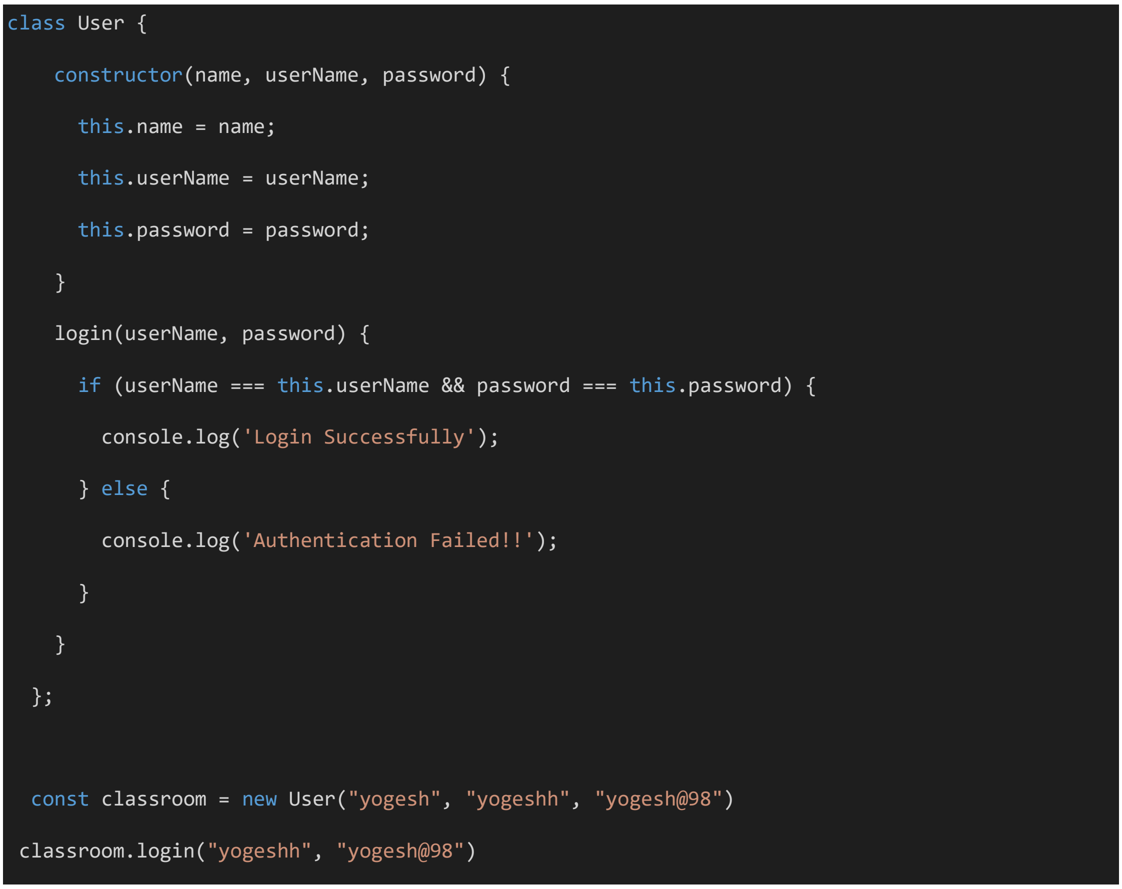
```
const person = {
    name: "ram",
    age: 22,
    greet: function(){
        return `Hello ${this.name}, you are ${this.age} years old`
    }
}
```

```
console.log(person.greet());
```

Class

Class is a blueprint of a real-life entity. It describes how the object will look alike, what characteristics it holds and what kind of actions we can perform on it.

Class is just a template. You can't perform any actions on it.



```
class User {
    constructor(name, userName, password) {
      this.name = name;
      this.userName = userName;
      this.password = password;
    }
    login(userName, password) {
      if (userName === this.userName && password === this.password) {
        console.log('Login Successfully');
      } else {
        console.log('Authentication Failed!!');
      }
    }
  };

  const classroom = new User("yogesh", "yogeshh", "yogesh@98")
 classroom.login("yogeshh", "yogesh@98")
```

Constructor - A constructor enables you to provide any custom initialization that must be done before any other methods can be called on an instantiated object.

Encapsulation

Encapsulation is the process of bundling related code into a single unit. Encapsulation makes it impossible for other parts of the code to manipulate or change how the bundled part of the application works unless you explicitly go into that unit and change them.

encapsulation is a concept that involves bundling data (attributes) and methods (functions) that operate on the data into a single unit, usually known as an object. This helps in organizing and protecting the internal state of an object from outside interference.

```
// Define a class using a constructor function

function Car(make, model) {

// Private variables

    var _make = make;

    var _model = model;


    this.getMake = function () {

      return _make;

    };


    this.getModel = function () {

      return _model;

    };


    this.setModel = function (newModel) {

      _model = newModel;

    };

  }


  var myCar = new Car('Toyota', 'Camry');


  console.log('Make:', myCar.getMake()); // Output: Make: Toyota

  console.log('Model:', myCar.getModel()); // Output: Model: Camry


  myCar.setModel('Corolla');

  console.log('Updated Model:', myCar.getModel()); // Output: Updated Model: Corolla
```

Attempting to access private variables directly from outside the object will result in an error, demonstrating encapsulation.

Encapsulation helps in hiding the internal implementation details of an object and provides a clean and controlled interface for interacting with the object.

Inheritance

When one class derived the properties and methods of another class it is called **inheritance**

The class that inherits the property is known as **subclass** or **child class** and the class whose properties are inherited is known as a **superclass** or **parent class**.

The main advantage of inheritance is **reusability**.

Inheritance in OOP reduces code duplication, enabling you to build a part of your application on another by inheriting properties and methods from that part of the application.

```
class User {
  #password;
  constructor(email, password) {
   this.email = email;
   this.#password = password;
  }

  login(email, password) {
   if (email === this.email && password === this.#password) {
    console.log('Login Successfully');
   } else {
    console.log('Authentication Failed!!');
   }
  }
 }
```

```
  class Author extends User {

    constructor(email, password) {

      super(email, password);

    }

  }
```

In the above example, the *Author* and *Admin* classes inherit the property of the *User* class using *extends* and *super* keywords.

The *extends* keyword is used to establish a parent-child relationship between two classes. In the first case, the *Author* becomes sub-class and the *User* becomes parent class.

Sub-class has access to all the public and protected members of a superclass. In addition, It can have its own properties and methods. This is how we can achieve **reusability** through inheritance.

The *super* keyword is a special keyword. Calling *super* in the child's constructor invokes the parent constructor. That's how we are initialising the properties in the *Author* and *Admin* classes.

The child class can also override the methods of a parent class. This introduces the concept of **polymorphism**.

Polymorphism

In programming, polymorphism is a term used to describe a code or program that can handle many types of data by returning a response or result based on the given data.

```
class User {

  constructor(email, password) {

    this.email = email;

    this.password = password;

  }


  login(email, password) {

    if (email === this.email && password === this.password) {
      console.log("Login Successfully");

    } else {

      console.log("Authentication Failed!!");

    }

  }

}
```

```
class Author extends User {

  constructor(email, password) {

    super(email, password);

  }

}
```

```
class Admin extends User {

  constructor(email, password) {

    super(email, password);

  }
```

```
  login(email, password) {

    if (email === this.email && password === this.password ) {

      console.log("Admin Login Successfully");

    } else {

      console.log("Authentication Failed!!");

    }

  }

}
```

```
const nehal = new Author("nm@gmail.com", "password:)");

nehal.login("nm@gmail.com", "password:)"); // Login Successfully
```

```
const json = new Admin("jason@gmail.com", "passwordinfo");

json.login("jason@gmail.com", "passwordinfo"); //admin login
```

Here, the *Author* and *Admin* both inherit the *User* class. Both classes have the *login* method of the User class. Now I need some extra level of verification for the admin account, so I have created a login method in the Admin class. It will override the parent's *login* method.

When an object of the *Admin* class calls the *login* method, it will invoke a function call to the *login* method of the *Admin* class.

This is how we have achieved polymorphism using method overriding.

Class component

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


export class ClassComponent extends Component {

  constructor(props) {

    super(props);

    this.state = {

      name: "",

    };

  }

  componentDidMount() {

    console.log("componentDidMount calling...");

  }
```

```
  componentDidUpdate(prevProps, prevState) {

    if (prevState.name !== this.state.name) {

      console.log("componentDidUpdate calling...");

    }
  }
  render() {
    return (
      <div>
        <input
          type="text"
          placeholder="Enter Your Name"
          value={this.state.name}
          onChange={(e) => this.setState({ name: e.target.value })}
        />
        <p>Entered Name: {this.state.name}</p>
      </div>
    );
  }
}
```

```
export default ClassComponent;
```

Constructor

Constructor is a method that is used to create and initialize an object created with a class and this must be unique within a particular class.

- It is used to initialize objects i.e, State in ReactJS.
- It is used to read or access a parent's property from a child's component in ReactJS

Super: Super is a keyword that is used to call the parent's class method and data. It is used for the inheritance model of class.

it is used to access parent properties and methods from the child components.

Lifecycle of Components

lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component's existence.

- **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
- **Mounting:** Mounting is the stage of rendering the JSX returned by the render method itself.
- **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.
- **Unmounting:** As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

Initialization

In this phase, the developer has to define the props and initial state of the component this is generally done in the constructor of the component. The following code snippet describes the initialization process.

```
class Clock extends React.Component {
    constructor(props)
    {
        // Calling the constructor of
        // Parent Class React.Component
        super(props);

        // Setting the initial state
        this.state = { date : new Date() };
    }
}
```

Mounting

Mounting is the phase of the component lifecycle when the initialization of the component is completed and the component is mounted on the DOM and rendered for the first time on the webpage.

- First initialize the data and the states in the constructor
- componentDidMount() Function: This function is invoked right after the component is mounted on the DOM i.e. this function gets invoked once after the render() function is executed for the first time

Update

Updation is the phase where the states and props of a component are updated followed by some user events such as clicking, pressing a key on the keyboard, etc.

- setState() Function: This function is used to update the state of a component.

- componentDidUpdate() Function: Similarly this function is invoked after the component is rerendered i.e. this function gets invoked once after the render() function is executed after the updation of State or Props.

Unmounting

This is the final phase of the lifecycle of the component which is the phase of unmounting the component from the DOM. The following function is the sole member of this phase.

- componentWillUnmount() Function: This function is invoked before the component is finally unmounted from the DOM i.e. this function gets invoked once before the component is removed from the page and this denotes the end of the lifecycle.

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


const FunctionalComponent = (props) => {

  const [name, setname] = useState("")
```

```
  useEffect(() => {

    console.log("componentDidUpdate is running...")

  }, [name])
```

```
  return (

    <div>

      <h2>Functional Component</h2>

      <p>{props.info}</p>
      <input type='text' placeholder='enter name'
      value={name}
       onChange={(e)=> setname(e.target.value)} />
 <p>{name}</p>
```

```

    </div>
  )
}
```

```
export default FunctionalComponent
```

```
import { Component } from "react";
class ClassComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "yogesh",
      phoneNumber: "",
    };
  }

  // componentDidMount - this will call only one time when the page reloads
  //(similar to useeffect with empty dependency)
  componentDidMount() {
    console.log("3")
    console.log("componentDidMount is running...");
  }
```

```
  componentDidUpdate(prevprops, prevState) {
    if(prevState.name !== this.state.name) {
      console.log("componentDidUpdate is running...")
    }
  }
  componentWillUnmount(){
    alert("componentWillUnmount")
    console.log("componentWillUnmount is running...")
  }
```

```
  render() {
    console.log("1")
    return (
      <div>
        <h2>Class Component</h2>
        <p>{this.props.info}</p>
        {console.log("2")}
        <input
          type="text"
          placeholder="enter name"
          value={this.state.name}
          onChange={(e) => {
            this.setState({ name: e.target.value, phoneNumber:""  });
          }}
        />
        <p>{this.state.name}</p>
      </div>
    );
  }
}
```

```
export default ClassComponent;
```

Lazy loading / code splitting

The goal is to improve the initial loading time and performance of your application by only loading the code that is necessary for the initial view.

This technique used to improve the performance of your application by splitting your JavaScript code into smaller, more manageable chunks. Instead of loading the entire JavaScript bundle at once when a user visits your application, code splitting allows you to load only the code that is necessary for the current view

Bundle- refers to a single, minified, and compressed JavaScript file that contains all the code needed for a particular part or the entirety of a web application. Bundling is a common practice in modern web development to optimize the loading and execution of JavaScript code.

```
import React, { Suspense, lazy } from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
const Demo = lazy(()=> import("./demo"))
const Menulist = lazy(()=> import("./menulist"))
const PageNotFound = lazy(()=> import("./pageNotFound"))

const App = () => {
 return (
  <div>
   <BrowserRouter>
    <Routes>
     <Route path="/" element={<Suspense fallback={"loading..."}><Demo /></Suspense>}></Route>
     <Route path="/list" element={<Suspense fallback={"loading..."}><Menulist /></Suspense>}></Route>
     <Route path="*" element={<Suspense fallback={"loading..."}><PageNotFound /></Suspense>}></Route>
    </Routes>
   </BrowserRouter>
  </div>
 );
};
export default App;
```

Refer Interveiw Questions:
https://www.interviewbit.com/react-interview-questions