Name : Patel Jenishkumar Kishorbhai

Class : M.Sc. ICT (sem - I)

Subject : Frontend Development using ReactJS

Roll No : 50

Assignment - 3

Q.1 ReactJS Application Architecture :

→ The React Library is just a UI Library.

→ ReactJs is a Javascript library for building user interfaces.

→ It focuses on building reusable components.

→ It used to build single-page application (SPA) with fast, responsive interfaces.

* Key Components of ReactJS :

1. Components :
→ The building blocks of React.

2. JSX :
→ Javascript XML syntax used for UI rendering.

3. Virtual DOM :
→ A lightweight representation of the real DOM.

4. Unidirectional Data Flow :
→ Data Flows in one direction, top-down.

5. State & Props:
→ Manage Dynamic data and pass it between components.

* ReactJS Architecture:

1. Component based architecture:
→ Applications are built using reusable components.

2. One-way data Flow:
→ data flows from parent component to child components through props.

3. Virtual DOM:
→ Updates are first reflected in the virtual DOM and then efficiently updated in the real DOM.

4. JSX:
→ Simplifies the process of writing and maintaining UI code.

Page No. 3.
Date

React App Architecture

Root Components

React Component	React UI Component
React UI Component	React UI Component

React Third party Components

Router Management	React Animation
React Advanced State Management	React REST API Management

Page No. 4
Date

Q.2 ReactJS class components :

→ A class component must include the extends React.Component statement.

→ This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

→ The component also requires a render() method, this method returns HTML.

Ex. Create a class component called Car :

```
class Car extends React.Component {
  render() {
    return <h2>Hi, This is a Car</h2>;
  }
}
```

Page No. 5
Date

Q.3 Functional components of ReactJS:

→ A function component also returns HTML, and behaves much the way as a Class component, but Function components can be written using much less code, are easier to understand.

Ex. Create a Function component called car:

```
function Car() {
  return <h2> Hi, This is a car </h2>;
}
```

Q.4 ReactJS Nested Components:

→ We can refer to components inside other components:

Ex:

```
function Car() {
  return <h2> Hi, This is a car </h2>;
}
function Garage() {
  return (
    <>
    <h1> Who lives in my Garage? </h1>
    <Car />
    </>
  );
}
```

```
const root = ReactDOM.createRoot
(document.getElementById('root'));
root.render(<Courage />);
```

Q.5 ReactJS conditional Components :

1) If Statement :

→ We can use the if statement Javascript operator to decide which component to render.

Ex.

```
function MissedGoal() {
  return <h1> Missed! </h1>;
}
function MadeGoal() {
  return <h1> Goal! </h1>;
}
```

→ Now, we'll create another component that chooses which component to render based on a condition :

```
function Goal(props) {
  const isGoal = props.isGoal;
  if (isGoal) {
    return <MadeGoal />;
  }
  return <MissedGoal />;
}
```

```
const root = ReactDOM.createRoot(document.
  getElement.ById('root'));
root.render(<Cool.isCool = {false}/>);
```

→ Try changing the isCool attribute to true:

```
const root = ReactDOM.createRoot(document.
  getElementById('root'));
root.render(<Cool.isCool = {true}/>);
```

2] Logical && Operater :-

→ Another way to conditionally render a React Component is by using the && operater.

Ex.

```
function Garage(props){
  const cars = props.cars;
  return(
    <>
      <h1>Garage</h1>
      {cars.length > 0 &&
        <h2>You have {cars.length}
          cars in your Garage.</h2>
      }
    </>
  );
}
const cars = ['Ford', 'BMW', 'Audi'];
const root = ReactDOM.createRoot(document
```

getElementById('root'));
root.render(<Garage cars={cars} />);

3) Ternary Operator :

→ condition ? true : false .

Ex. Return the MadeGoal component if isGoal is true, otherwise return the MissedGoal component :

```
function Goal(props) {
  const isGoal = props.isGoal;
  return (<>
    { isGoal ? <MadeGoal/> : <MissedGoal/>
    }
  </>
  );
}

const root = ReactDOM.createRoot
(document.getElementById('root'));
root.render(<Goal isGoal = {false} />);
```

4) map() method :

→ In react, you will render lists with some type of loop.

→ The JavaScript map() array method is generally the preferred method.

Ex.

```
function Car(props) {
  return <li> I am a {props.brand} </li>;
}

function Garage() {
  const cars = ['Ford', 'BMW', 'Audi'];
  return (
    <>
      <h1> Who lives in my garage? </h1>
      <ul> {cars.map((car) =>
        <Car brand={car} />)} </ul>
    </>
  );
}

const root = ReactDOM.createRoot
(document.getElementById('root'));
root.render(<Garage />);
```

Q.6 ReactJS State Management :

→ The state object is where you store property values that belong to the component.

→ When the state object changes, the component re-renders.

Ex.

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = { brand: "Ford" };
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

→ The state object can contain as many properties as you like.

→ State object anywhere in the component by using the this.state.property name.

Ex.

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = { brand: "Ford",
                   model: "Mustang",
                   color: "red",
                   year: 1964 };
  }
  render() {
    return (
      <div> <h1> my {this.state.brand} </h1>
```

```
<p> It is a {this.state.color}
                {this.state.model}
                from {this.state.year}.
    </p>
  </div>
 );
}}
```

→ To change the value in the state object, use the this.setState() method.

Add a button with an onClick event that will change the color property.

```
-Class car extends React.Component {
   constructor(props) {
     super(props);
     this.state = {
       brand: "Ford",
       model: "Mustang",
       color: "red",
       year: 1964 };
   }
changeColor = () => {
   this.setState({color: "blue"});
}
render() {
  return (<div>
     <h1> my {this.state.brand} </h1>
  <p> It is a {this.state.color}
     {this.state.model} from
```

{this.state.yes}.
</p>
<button type="button" onClick={this.changeColor}>Change Color</button>
 ); }
}

Q.7 ReactJS Props :
-> React props are like function arguments in Javascript and attributes in HTML.

e.g. Add a "brand" attribute to the Car element :

const myElement = <Car brand="Ford" />;

-> The component receives the argument as a props object :

function Car(props) {
return <h2> I am a {props.brand}!
</h2>;
}

* Pass Data :
-> Props are also how you pass data from one component to another, as parameters.

ex
```
function Car(props) {
    return <h2> I am a {props.brand} !</h2>;
}
function Garage() {
    return (
        <>
        <h1> Who lives in my garage ? </h1>
        <Car brand = "Ford" />
        </>
    );
}

const root = ReactDOM.createRoot
(document.getElementById('root'));
root.render(<Garage />);
```

Q.8 ReactJS Form components:

=> You add a Form with React like any other element:

eg.
```
function MyForm () {
    return ( <form>
        <label> Enter your name:
            <input type="text" />
        </label>
    </form> )
}

const root = ReactDOM.createRoot
(document.getElementById('root'));
root.render (<MyForm />);
```

Page No. 14
Date

→ You can control the values of more than one input field by adding a name attribute to each element.

1. State Management:
→ We use the useState hook to manage the form data.

→ The formData state object stores the values of the form fields.

2. Handling Input Changes:
→ The handleChange function updates the formData state whenever an input value changes.

→ It uses the spread operator (...) to create new object with the updated field.

3. Form Submission:
→ The handleSubmit function is called when the form is submitted.

→ It prevents the default form submission behaviour and logs the formData to the console.

→ You can replace the console.log with your desired actions, such as sending the data to a server.

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

function MyForm() {
  const [formData, setFormData] = useState
    ({ name: '',
       email: '',
       message: '' });

  const handleChange = (event) => {
    setFormData ({ ...formData,
      [event.target.name]:
        event.target.value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label> Name:
        <input type="text" name="name"
          value={formData.name}
          onChange={handleChange} />
      </label>
      <br />
      <label> Email:
        <input type="email" name="email"
          value={formData.email}
          onChange={handleChange} />
```

Page No. 16
Date

```
</label>
<br />
<label>Message:
  <textArea name="message"
            value={formData.message}
            onChange={handleChange} />
</label>
<br />
<button type="submit">Submit</button>
</form>
);
};

export default
MyForm;
```