Msc. ICT

Name :- Valvadiya Dhruvi m. roll no :- 73

Sub :- React js

Assignment - 3

1) Application architecture

→ ReactJS is a Javascript library for building user interfaces.

→ Architecture key Points :-

1. Component-Based Architecture : Applications are built using reusable components.
2. One-way Data Flow : Data flows from parent components 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.

→ React creates a virtual DOM in memory

→ Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

2) class components

→ Event handling methods are typically defined as methods of the class.

→ Binding of event handler methods in the constructor or using arrow function is required to access 'this'.

→ Binding 'this' is necessary for event handlers to access the component's instance state & Props.

→ Methods for binding :

1. Binding in the Constructor.
2. Using arrow functions in class fields.

→ Example of class components

```
class BtnClick extends React.Component {
    Constructor(Props) {
        Super(Props);
        this.state = {clicked: false};
        this.handleClick = this.handleClick.bind(this);
    }
    handleClick() {
        this.setState({clicked: true});
    }
    render() {
        return (
            <button onClick={this.handleClick}>
                {this.state.clicked ? 'Clicked!' : 'Click me'}
            </button>
        );
    }
}
```

3) Functional Components

→ Functional components use hooks for event handling, such as useState for state and useEffect for side effects.
→ No need for 'this' in functional components.
→ Event handlers are declared as functions with in the component.

→ Example of functional components

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

```
function AvengersFunctional () {
  const [avenger, setAvenger] = useState('None');
  const handleClick = (name) => {
    setAvenger(name);
  };
  return (
    <div>
      <h1> which New Avenger is your favorite? </h1>
      <button onClick={() => handleClick("Black Panther")}>
        > Black Panther </button>
      <button onClick={() => handleClick("Captain")}>
        Captain </button>
      <button onClick={() => handleClick('Spider')}>
        Spider </button>
      <h2> You selected : {avenger.} </h2>
    </div>
  );
}
export default AvengersFunctional;
```

4) Nested Components

→ In React, nested components are a powerful way to build complex UIs by breaking down the interface into smaller, reusable pieces. Here's quick overview of how to create and use nested components.

→ Example of Nested Components.

→ User.js

```
import React from 'react';
const User = ({name, email}) => {
```

```
    return (
        <div className="user">
            <h2>{name}</h2>
            <p>{email}</p>
        </div>
    );
}
export default User;
```

// UserList.js

```
import React from 'react';
import User from './User';

const UserList = ({ users }) => {
    return (
        <div className="user-list">
            {users.map((user, index) => (
                <User key={index} name={user.name}
                    email={user.email} />
            ))}
        </div>
    );
};
export default UserList;
```

// App.js

```
import React from 'react';
import UserList from './UserList';

const App = () => {
    const users = [
        {name: 'John', email: 'John@w.com'},
```

```
    {name: 'smith', email: 'smith@gmail.com'},
    {name: 'bob', email: 'bob@gmail.com'},
  ];
  return (
    <div className="app">
      <h1>User List</h1>
      <UserList users={users} />
    </div>
  );
};
export default App.
```

5) Conditional and looping constructs

→ Conditional rendering in React can be achieved using JavaScript expressions. Here are a few common Patterns:

1. You can use regular if-else statements inside the render method.
2. A more concise way is to use the ternary op.
3. You can also use the logical && operator for rendering components conditionally.

→ when you need to render lists lists of items, you can use the map() function to loop through arrays.

→ Example of conditional and looping constructs

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

```
const shoppingList = ({ items, isLoggedIn }) => {
  if (items.length === 0) {
    return <p>No items in your shopping list</p>;
  }
  return (
    <div>
      <h2>{isLoggedIn ? 'Your shoping list' :
        'Please log in to see your shopping list'}</h2>
      <ul>
        {isLoggedIn && items.map((item, index) =>
          (<li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
};
const APP = () => {
  const items = ['APPles', 'Banans', 'oran'];
  const isLogged In = true;
  return (
    <div>
      <shoppingList items={items}
        isLoggedIn={isLoggedIn} />
    </div>
  );};
export default APP;
```

6) State

→ React Components has a built-in state object.
→ The state object is where you store property values that belong to the component.

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

Create State object.

→ State object is initialized in the constructor

Example :-

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

Props

→ Props are arguments passed into React Components.

→ Props are passed to components via HTML attributes

→ Ex:- Const myEle = <Car brand="ford" />;

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

8) form (controlled component)

→ A controlled component is an input element whose value is controlled by React.

→ React handles the form's state and updates it with every input.

→ Example of a simple form

```
import React, {useState} from 'react';
function SimpleForm() {
    const [name, setName] = useState('');
    const handleChange = (e) => {
        setName(e.target.value);
    };
    const handleSubmit = (e) => {
        e.preventDefault();
        alert('form submitted Name: ${name}');
    };
return (
    <form onSubmit={handleSubmit}>
        <label>
            Name:
            <input type="text" value={name}
                onChange={handleChange} />
        </label>
        <button type="submit">Submit</button>
    </form>
); }
export default SimpleForm;
```