Marwadi University
Marwadi Chandarana Group

Unit -2

ReactJs

Advanced Web Technology

Dept. of Computer Engineering

Content

Introduction ReactJs

- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application.
- It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram.
- Facebook developed ReactJS in 2011 in its newsfeed section, but it was released to the public in the month of May 2013.

Templating using JSX

- In the context of JSX (JavaScript XML), templating refers to the process of dynamically generating or composing the structure and content of JSX elements based on data or variables. JSX is a syntax extension for JavaScript that allows you to write HTML-like code within your JavaScript code.

- In JSX, you can embed JavaScript expressions and statements by enclosing them in curly braces {}. This enables you to include dynamic content, perform computations, and make decisions based on the current state or props of your component.

Templating using JSX example:-

```
import React from 'react';

const User = ({ name, age }) => {
  return (
    <div>
      <h2>{name}</h2>
      <p>{age} years old</p>
    </div>
  );
};

const App = () => {
  const user = { name: 'John Doe', age: 25 };

  return (
    <div>
      <h1>User Details</h1>
      <User name={user.name} age={user.age} />
    </div>
  );
};

export default App;
```

Components

- Components are the building blocks of user interfaces in ReactJS. They are reusable and independent units of code that encapsulate specific functionality and UI elements. React components allow you to split your UI into smaller, manageable pieces, making it easier to develop, maintain, and test your application.

- There are two types of components in ReactJS:
    - Functional components.
    - Class components.

Functional components

- **Functional Components:** Functional components are JavaScript functions that receive **props** (properties) as arguments and **return JSX** elements.
- They are also known as stateless components because they don't have their own internal state.
- Functional components are simpler and easier to understand, making them a good choice for most UI elements that don't need to manage state or lifecycle methods.

Functional components

```
function App( ) {
  return (
            <p>This is the Example of Functional Component.</p>
  );
}
export default App;
```

Class components

- **Class Components:** Class components are JavaScript classes that extend the **React.Component** base class.
- They have their **own state** and can define **lifecycle methods**.
- Class components were the traditional way of creating components in React before the introduction of hooks.

Example class components

```
import { Component } from 'react';

class App extends Component {
  render( ) {
     return (
                 <p>Computer Department</p>
     );
  }
}
export default App;
```

Working with state and props

- In ReactJS, state and props are two fundamental concepts used to manage and pass data within components.
- **State** is an **object** that represents the internal data of a component. It allows you to store and manage **mutable data** that can change over time. State is used to keep track of information that affects the component's rendering and behavior.
- To work with state in a class component, you need to define the state object within the component's constructor using **this.state** = initialState;. You can then access and update the state using **this.setState(newState);**, which triggers a re-render of the component.

State example :-

```
import { useState } from "react";
export default function App(){
   const [count,setCount] = useState(0);

  const increment = ( )=>{
    setCount(count+1);
  }
  return(
    <>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </>
  )
}
```

Props

- **Props (short for properties)** are **read-only values** that are passed to a component from its parent component. They allow you to pass data and configuration from one component to another.

- Props are received as function arguments in functional components or accessed via 'this.props' in class components. They are **immutable** and cannot be modified within the receiving component.

Example props;-

```
import React from 'react';

const Greeting = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};

export default Greeting;
```

Props example:-

- Props can be passed from a parent component to a child component by specifying the prop values as attributes when rendering the child component.

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

const App = () => {
  return <Greeting name=“Samir" />;
};

export default App;
```

Rendering lists

- Rendering lists in ReactJS involves mapping over an array of data and generating a list of components based on that data.

- React provides a convenient way to render lists using the filter ( ) and map( ) methods of arrays.

Example rendering list:-

```
import React from 'react';

export default function App( ){
            const items = ['Item 1', 'Item 2', 'Item 3'];
            return (
               <ul>
                              {items.map( (item, index) => (
                                          <li key={index}>{item}</li>
                              ))}
             </ul>
            );
}
```

Rendering lists

- map ( ) creates a new array from calling a function for every array element
- map ( ) does not execute the function for empty elements
- map ( ) does not change the original array
- **Note:-** that the key prop should have a unique value for each item in the list. In this example, we're using the index as a key, but if your list items have unique IDs, it's recommended to use those instead for better performance.
- **Example:**

Ex-1)

```
const numbers = [4, 9, 16, 25];
const newArr = numbers.map(Math.sqrt)
```

Ex-2)

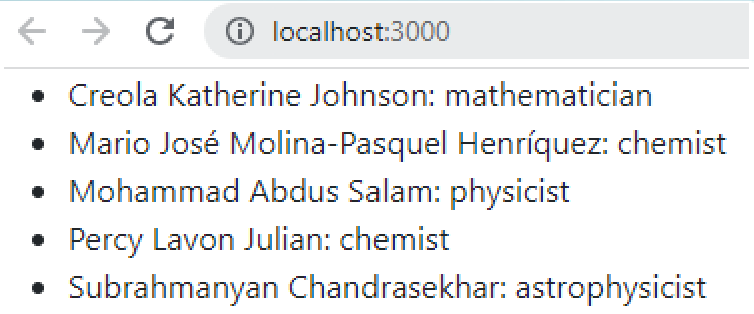
```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
        return num * 10;
}
```

Rendering lists

- **Example:**

```
Ex-3)
const people = [
  'Creola Katherine Johnson: mathematician',
  'Mario José Molina-Pasquel Henríquez: chemist',
  'Mohammad Abdus Salam: physicist',
  'Percy Lavon Julian: chemist',
  'Subrahmanyan Chandrasekhar: astrophysicist'
];
export default function List( ) {
            const listItems = people.map ( person =>  <li> {person} </li>  );
            return (
                        <ul> {listItems} </ul>
            );
}
```


localhost:3000

- Creola Katherine Johnson: mathematician
- Mario José Molina-Pasquel Henríquez: chemist
- Mohammad Abdus Salam: physicist
- Percy Lavon Julian: chemist
- Subrahmanyan Chandrasekhar: astrophysicist

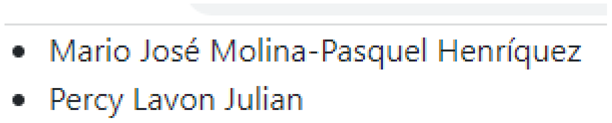
Rendering lists

Ex-4)

```
export default function App(){
 const people = [{
  id: 0,
  name: 'Creola Katherine Johnson',
  profession: 'mathematician',
}, {
  id: 1,
  name: 'Mario José Molina-Pasquel Henríquez',
  profession: 'chemist',
}, {
  id: 2,
  name: 'Mohammad Abdus Salam',
  profession: 'physicist',
}, {
  name: 'Percy Lavon Julian',
  profession: 'chemist',
}, {
  name: 'Subrahmanyan Chandrasekhar',
  profession: 'astrophysicist',
}];
```

```
const chemists = people.filter(person =>
  person.profession === 'chemist'
);
const listItems = chemists.map(person => <li> {person.name}
</li>);

  return (<ul> {listItems} </ul>);
}
```



- Mario José Molina-Pasquel Henríquez
- Percy Lavon Julian

Rendering lists

Ex-5) Write a JSX code in React to display following lists from given array ‘cars’:

(i) List of Maruti cars (ii) List of Honda cars (iii) List of SUV cars

```
const cars = [
 {name: 'Alto', brand: 'Maruti', type: 'Hatchback'},
 {name: 'i10', brand: 'Hyundai', type: 'Hatchback'},
 {name: 'Brio', brand: 'Honda', type: 'Hatchback'},
 {name: 'Ciaz', brand: 'Maruti', type: 'Sedan'},
 {name: 'Verna', brand: 'Hyundai', type: 'Sedan'},
 {name: 'Honda City', brand: 'Honda', type: 'Sedan'},
 {name: 'Brezza', brand: 'Maruti', type: 'SUV'},
 {name: 'Creta', brand: 'Hyundai', type: 'SUV'},
 {name: 'CRV', brand: 'Honda', type: 'SUV'}
 ];
```

Event handling in React

- In essence, event handling in React enables a user to interact with a webpage and take specified action whenever an event, like a click or a hover, takes place.
- Events in React are fired when a user interacts with an application, such as mouseovers, key presses, change events, and so on.

Example event handling :-

```
import React from 'react';

const ExampleComponent = () => {
  const handleClick = () => {
    console.log('Button clicked!');
  };

  const handleChange = (event) => {
    console.log('Input value:', event.target.value);
  };
  return (
    <div>
      <button onClick={handleClick}>Click Me</button>
      <input type="text" onChange={handleChange} />
    </div>
  );
};

export default ExampleComponent;
```

Some of the events in ReactJs

The following are some specific events in React that you could encounter when interacting with websites built with React:

- Clicking an element
- Submitting a form
- Scrolling page
- Hovering an element
- Loading a webpage
- Input field change
- User stroking a key
- Image loading

Difference Between Html And React Event Handling

- With a few syntax modifications, React event handling is identical to HTML. For example:
- React uses camelCase for event names, while HTML uses lowercase.
- Instead of passing a string as an event handler, we pass a function in React.

In HTML

```
<form onsubmit="console.log('clicked'); return false">
  <button type="submit">Submit</button>
</form>
```

In ReactJs

```
const App = () => {
      function handleClick(e) {
         e.preventDefault();
         console.log('Clicked');
       }

      return (
         <form onSubmit={handleClick}>
           <button type="submit">Submit</button>
         </form>
       );
};
export default App;
```

Understanding Component Life Cycle

- In React, functional components are the primary way of creating reusable UI components. Unlike class components, functional components don't have a built-in lifecycle. However, with the introduction of React Hooks, you can achieve similar functionality and manage component lifecycle events in functional components.

- Here's an overview of the lifecycle events in functional components using React Hooks:

- **Mounting Phase:**
- **useState**: Allows you to add state to your functional components.
- **useEffect** (with an empty dependency array [ ]): Equivalent to componentDidMount. It runs only once, immediately after the component is mounted. You can perform tasks such as data fetching, subscriptions, or manually managing event listeners here.

- **Updating Phase:**

- **useEffect (with a dependency array):** This hook allows you to perform side effects whenever one or more dependencies change. It's similar to **componentDidUpdate.** You can update the component state or interact with external resources based on the changes in the dependencies.

- **useMemo** and **useCallback**: These hooks can be used to memoize values and functions to optimize performance by preventing unnecessary recalculations.

- **Unmounting Phase:**

- **useEffect** (with a cleanup function): This hook allows you to clean up any resources or subscriptions created during the component's lifecycle. It's similar to **componentWillUnmount**. The cleanup function returned from **useEffect** is called when the component is unmounted.

- Additionally, there are hooks available for handling other scenarios, such as handling context, handling form inputs, and more.

useEffect Example:-

```
import { useState, useEffect } from "react";
export default function App(){
  const [count,setCount] = useState(0);
  const [cal,setCal] = useState(0);
  useEffect(()=>{
    setCal(count*2);
  },[count]);
  const inc = ()=>setCount(count+1);
   return(
    <>
    <h3>Count: {count}</h3>
    <button onClick={inc}>Increment</button>
    <h3>Cal: {cal}</h3>
    </>
   )
}
```

Forms in ReactJs

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

function MyForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    // Handle form submission logic here
    console.log('Name:', name);
    console.log('Email:', email);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <br />
      <label>
        Email:
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
      </label>
      <br />
      <button type="submit">Submit</button>
    </form>
  );
}

export default MyForm;
```

Forms

- Forms in ReactJS allow users to input and submit data. React provides a convenient way to manage form state and handle form submissions using controlled components.

Example for form :-

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

function MyForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [acceptedTerms, setAcceptedTerms] = useState(false);
  const [selectedOption, setSelectedOption] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    // Handle form submission logic here
    console.log('Name:', name);
    console.log('Email:', email);
    console.log('Accepted Terms:', acceptedTerms);
    console.log('Selected Option:', selectedOption);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <br />
      <label>
        Email:
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
      </label>
```

Example for form contd:-

```
<br />
    <label>
      <input
        type="checkbox"
        checked={acceptedTerms}
        onChange={(e) => setAcceptedTerms(e.target.checked)}
      />
      I accept the terms and conditions
    </label>
    <br />
    <label>
      Select an option:
      <select value={selectedOption} onChange={(e) => setSelectedOption(e.target.value)}>
        <option value="">Select an option</option>
        <option value="option1">Option 1</option>
        <option value="option2">Option 2</option>
        <option value="option3">Option 3</option>
      </select>
    </label>
    <br />
    <button type="submit">Submit</button>
  </form>
 );
}
```

Routing with React Router

- **Routing** is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application.

- **React Router** is a popular library in the React ecosystem that enables declarative routing in your React applications. It provides a set of components and utilities to handle navigation and rendering different components based on the current URL.

- React Router helps you build single-page applications with multiple views and enables a smooth user experience by updating the UI without full page reloads.

React Router

- Here are some key components provided by React Router:
- **BrowserRouter**: This component uses HTML5 history API to synchronize the UI with the URL. It should be placed around the top-level of your application to enable routing functionality.
- **Switch**: The Switch component is used to render the first matching Route or Redirect inside it. It allows exclusive rendering of components based on the current URL. When a Route inside the Switch matches, it renders the corresponding component, and the Switch stops evaluating the remaining Routes.

React Router

- **Route**: The Route component defines a mapping between a URL path and the component to render when the path matches the current URL. It takes a path prop and a component prop (or render prop) to specify which component to render. Optionally, you can use the exact prop to match the exact path instead of a partial match.

- **Link**: The Link component is used to navigate between different routes. It renders an <a> element with the specified destination URL. When clicked, it updates the URL and triggers the corresponding component rendering. This prevents full page reloads and provides a seamless user experience.

React Router

- **NavLink**: The NavLink component is similar to Link but provides additional styling and highlighting options. It allows you to add custom CSS classes to the active link based on the current URL, making it easy to apply different styles to active links.

- **Redirect**: The Redirect component is used to redirect the user to a different URL. It takes a to prop that specifies the destination URL. When rendered, it automatically updates the URL and redirects the user to the specified location.

```
import React from ‘react’

function Home( ) {
  return (
   <h4> Welcome to Home Page ! </h4>
)
}

export default Home
```

```
import React from ‘react’

function About( ) {
  return (
   <h4> Welcome to About ! </h4>
)
}

export default About
```

```
import React from ‘react’

function Contact( ) {
  return (
   <h4> Welcome to Contact ! </h4>
)
}

export default Contact
```

React Router example:-

```
import React from 'react';
import { BrowserRouter as Router, Routes,Route, Link } from 'react-router-dom';

// Components for each route
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';

function App() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
            <li>
              <Link to="/contact">Contact</Link>
            </li>
          </ul>
        </nav>

        <Routes>
          <Route path="/" exact element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

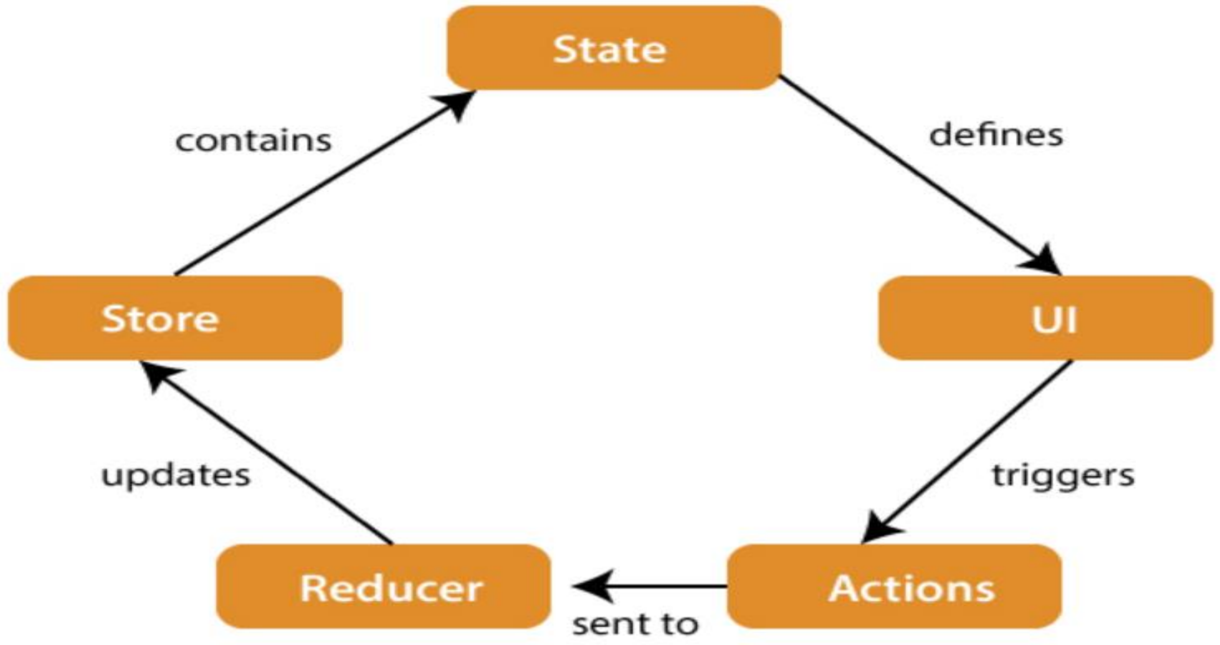
Redux

- **Redux** is an **open-source** JavaScript library used to manage **application state**. React uses Redux for building the **user interface**. It was first introduced by **Dan Abramov** and **Andrew Clark** in 2015.

- **Redux** is a state management **library** that helps manage the state of an application in a predictable and **centralized** manner. It is commonly used with React but can be used with any JavaScript framework or library. Redux follows a unidirectional data flow pattern and maintains the state of the entire application in a single store.

Redux

Redux Architecture



Redux

- **Action**: Actions are plain JavaScript objects that describe a specific event or intention to change the state of the application. Actions are dispatched to the Redux store and carry a type property that describes the action's purpose. They can also contain additional data or payload required to update the state.

- **Reducers**: Reducers are pure functions responsible for handling actions and updating the state of the application. They take the current state and an action as input and produce a new state based on the action's type and payload. Each reducer typically handles a specific portion of the overall application state.

- **Dispatch**: The dispatch function is used to send actions to the Redux store. It is provided by the store and is used to trigger state updates. When an action is dispatched, it is passed to the reducers, which update the state accordingly.

Redux

- **Store**: The store holds the application state and serves as the single source of truth. It is created using the createStore function from Redux. The store is responsible for dispatching actions, maintaining the application state, and notifying subscribers of state changes.

- **Subscribers**: Subscribers are functions or components that subscribe to the Redux store to receive updates whenever the state changes. They can be React components or other parts of the application interested in observing changes in the state. When the state changes, the subscribers are notified, allowing them to update the UI or take other actions.

- **View**: The view represents the user interface (UI) of the application. It can be a React component or any other UI rendering mechanism. The view interacts with the store by dispatching actions and receives updated state through subscriptions. It renders the updated state to the user interface, reflecting the changes in the application's state

Thank You