MERN: UNIT-2 Advanced React Concepts

Prepared By,

M.Gouthamm, Asst.Prof, CSE, MRUH

Advanced React Concepts

Topics Covered

1. **Styling in React**
2. **Conditional Rendering and Lists**
3. **Handling Events in React**
4. **React Router**

Styling in ReactJS

- React is used to style the React App or Component. The style attribute is the most used attribute for styling in React applications, which adds dynamically-computed styles at render time.
- It accepts a JavaScript object in camelCased properties rather than a CSS string. There are many ways available to add styling to your React App or Component with CSS.
- React allows component to be styled using CSS class through className attribute. Since, the React JSX supports JavaScript expression, a lot of common CSS methodology can be used.

Some of the top options are as follows −

- **CSS stylesheet** − Normal CSS styles along with className
- **Inline styling** − CSS styles as JavaScript objects along with camelCase properties.
- **CSS Modules** − Locally scoped CSS styles.

Styling in ReactJS

- **CSS stylesheet** − Normal CSS styles along with className
- CSS stylesheet is usual, common and time-tested methodology. Simply create a CSS stylesheet for a component and enter all your styles for that particular component.
- Then, in the component, use className to refer the styles.

Let us style our component.

- Create CSS file in src folder in React Project

Style.css

```
div.items{
  background-color: #008080;
  color: lightblue;
  padding: 40px;
  font-family: 'Courier New', Courier, monospace;
  text-align: center;
  }
```

Component.js: `<div className=“items”></div>`

Styling in ReactJS

Inline styling – CSS styles as JavaScript objects along with camelCase properties.

Inline Styling is one of the safest ways to style the React component. It declares all the styles as JavaScript objects using DOM based css properties and set it to the component through style attributes.

Component.js

```
const items={
        backgroundColor: '#008080',
        color: 'lightblue',
        padding: '40px',
        fontFamily: 'Courier New',
        textAlign: 'center',
      }
return(
<div> <h3 style={{color:’Green’}}> Hello, Inline Styling</h3></div>
<div style={items}></div>
```

Styling in ReactJS

CSS Modules − Locally scoped CSS styles.

- CSS Module is another way of adding styles to your application. It is a CSS file where all class names and animation names are scoped locally by default.
- It is available only for the component which imports it, means any styling you add can never be applied to other components without your permission, and you never need to worry about name conflicts.
- You can create CSS Module with the **.module.css** extension like a **Styles.module.css** name.

Styling in ReactJS

CSS Modules – Locally scoped CSS styles.

Styles.module.css

```
.items{
    background-color: #888080;
    color: lightcoral;
    padding: 40px;
    font-family: 'Courier New', Courier, monospace;
    text-align: center;
  }
```

Componet.js

```
Import styles from ‘./tyles.modules.css’
<div className={styles.items}></div>
```

Styling in ReactJS

Bootstrap Card

A card is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options. If you're familiar with Bootstrap 3, cards replace our old panels, wells, and thumbnails. Similar functionality to those components is available as modifier classes for cards.

Example

Cards are built with as little markup and styles as possible, but still manage to deliver a ton of control and customization.

Built with flexbox, they offer easy alignment and mix well with other Bootstrap components. They have no margin by default, so use spacing utilities as needed.

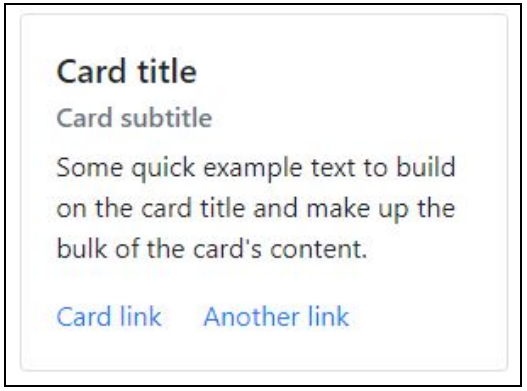
Styling in ReactJS

Bootstrap Card

Titles, text, and links

Card titles are used by adding **.card-title** to a **<h*> tag**. In the same way, links are added and placed next to each other by adding **.card-link to an <a> tag**.

Subtitles are used by adding a **.card-subtitle to a <h*> tag**. If the **.card-title** and the **.card-subtitle** items are placed in a **.card-body** item, the card title and subtitle are aligned nicely.



Card title

Card subtitle

Some quick example text to build on the card title and make up the bulk of the card's content.

Card link Another link

```
<div class="card" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <h6 class="card-subtitle mb-2 text-muted">Card subtitle</h6>
     <p class="card-text">Some quick example text to build on the card title and make up the bulk of the
card's content.</p>
    <a href="#" class="card-link">Card link</a>
    <a href="#" class="card-link">Another link</a>
  </div>
</div>
```

Styling in ReactJS

Example:

```
import React from 'react';
function BootstrapCardComponent(props) {
    const {employee1}=props;
 return(
    <div className='col-md-4'>
        <div className='card'>
            <div className='card-body'>
                <h5 className='card-title'>{employee1.firstName} {employee1.lastName}</h5>
                <p className='card-text'>Job: {employee1.job}</p>
                <p className='card-text'>salary: {employee1.salary}</p>
            </div>
        </div>
    </div>
);
}
```

Contd…

Styling in ReactJS

Example: contd…

```
const EmployeeListComponent=()=>{
    const employee = [
        { firstName: "George", lastName: "Smith", job: "writer", salary: 50000 },
        { firstName: "Michael", lastName: "Handler", job: "DJ", salary: 150000 },
        { firstName: "Larry", lastName: "David", job: "writer", salary: 250000 },
        { firstName: "Mindy", lastName: "Smith", job: "cook", salary: 120000 }
      ];
return(
    <div className='container-fluid'>
        <div className='row'>
            {employee.map((employee, index)=>(
                <BootstrapCardComponent key={index} employee1={employee}/>
            ))}
        </div>
    </div>
);
}
export default EmployeeListComponent;
```

Conditional Rendering

Conditional Rendering

- In React, we can create multiple components which encapsulate behavior that we need. After that, we can render them depending on some conditions or the state of our application.
- In other words, based on one or several conditions, a component decides which elements it will return. In React, conditional rendering works the same way as the conditions work in JavaScript.
- We use JavaScript operators to create elements representing the current state, and then React Component update the UI to match them.

There is more than one way to do conditional rendering in React. They are given below.

1. **Conditional (Ternary) Operator**
2. **Logical && Operator**
3. **if-else Statements**
4. **Switch Statements**
5. **Conditional Rendering Functions**

Conditional Rendering

1. **Conditional (Ternary) Operator:**

- The ternary operator is used in cases where two blocks alternate given a certain condition. This operator makes your if-else statement more concise. It takes three operands and used as a shortcut for the if statement.
- You can use the ternary operator **(condition ? trueCase : falseCase)** inside JSX to conditionally render components.

Example:

```
const MyComponent = ({ isLoggedIn }) => {
  return (
    <div>
      {isLoggedIn ? <WelcomeMessage /> : <LoginButton />}
    </div>
  );
};
```

Conditional Rendering

2. Logical && Operator

This operator is used for checking the condition. If the condition is true, it will return the element right after &&, and if it is false, React will ignore and skip it.

You can use the **logical AND (&&) operator** to conditionally render a component based on a condition.

Example:

```
const MyComponent = ({ isLoggedIn }) => {
  return (
    <div>
      {isLoggedIn && <WelcomeMessage />}
    </div>
  );
};
```

Conditional Rendering

3. if-else Statements

- It is the easiest way to have a conditional rendering in React in the render method. It is restricted to the total block of the component. IF the condition is true, it will return the element to be rendered.
- You can use traditional JavaScript **if-else** statements outside of JSX to conditionally render components.

```
Example:
const MyComponent = ({ isLoggedIn }) => {
  if (isLoggedIn) {
    return <WelcomeMessage />;
  } else {
    return <LoginButton />;
  }
};
```

Conditional Rendering

4. Switch Statements:

It is possible to have multiple conditional renderings. In the switch case, conditional rendering is applied based on a different state.

You can also use JavaScript switch statements to conditionally render components.

Example:

```
const MyComponent = ({ userType }) => {
  switch (userType) {
    case 'admin':
      return <AdminDashboard />;
    case 'user':
      return <UserDashboard />;
    default:
      return <DefaultDashboard />;
  }
};
```

Conditional Rendering

5. Conditional Rendering Functions

You can define separate functions to return JSX based on conditions and call them within your render method.

Example

```
const renderWelcome = (isLoggedIn) => {
  if (isLoggedIn) {
    return <WelcomeMessage />;
  } else {
    return <LoginButton />;
  }
};

const MyComponent = ({ isLoggedIn }) => {
  return (
    <div>
      {renderWelcome(isLoggedIn)}
    </div>
  );
};
```

ReactJS List

- Lists are useful to improve or create the UI of any website. Lists are mainly used for illustrating or showcasing menus on a website, for instance, the navbar menu.
- In regular JavaScript, we use arrays for creating lists, and in React we can create lists similarly as we do in standard JavaScript.
- Lists are crucial features of an app. A list of tasks like a calendar app, a list of pictures like Instagram, a list of items to shop in a shopping cart, etc., are the numerous use-cases where lists are used to improve the performance.
- Imagine an app with a huge list of videos or pictures appearing as you scroll.

Steps to Create and Traverse React JS Lists :

- We can create lists in React in a similar manner as we do in regular JavaScript i.e. by storing the list in an array. In order to traverse a list we will use the **map() function.**

Step 1: Create a list of elements in React in the form of an array and store it in a variable. We will render this list as an unordered list element in the browser.

Step 2: We will then traverse the list using the JavaScript **map() function** and update elements to be enclosed between <li> </li> elements.

Step 3: Finally we will wrap this new list within <ul> </ul> elements and render it to the DOM.

ReactJS List

Example-1: Display List of Items using map() method.

```
import React from 'react';

const MyListComponent = () => {
  const items = ['Item 1', 'Item 2', 'Item 3', 'Item 4', 'Item 5'];
  return (
    <div>
      <h1>List of Items</h1>
      <ul>
        {items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
};

export default MyListComponent;
```

ReactJS List

Example-2: Display List of Employee Details using map() method.

```
import React from 'react';

function EmpArrayComponent() {
  const data = [
    { firstName: "George", lastName: "Smith", job: "writer", salary: 50000 },
    { firstName: "Michael", lastName: "Handler", job: "DJ", salary: 150000 },
    { firstName: "Larry", lastName: "David", job: "writer", salary: 250000 },
    { firstName: "Mindy", lastName: "Smith", job: "cook", salary: 120000 }
  ];
```

ReactJS List

Example-2: Display List of Employee Details using map() method.

```
return (
        <div style={{height:500, width:500}}>
              <h5  className='text-danger'>Display  Student  Details  using  Array  map  method  in
React....</h5>
            <table class="table table-stripped">
              <thead class="thead-dark">
                <tr>
                  <th scope="col">S.No</th>
                  <th scope="col">First Name</th>
                  <th scope="col">LastName</th>
                  <th scope="col">Job</th>
                  <th scope="col">salary</th>
                </tr>
              </thead>
```

ReactJS List

Example-2: Display List of Employee Details using map() method.

```
    <tbody>
          {data.map((person, index) => (
     <tr>
                 <th>{index + 1}</th>
                 <td>{person.firstName}</td>
                 <td>{person.lastName}</td>
                 <td>{person.job}</td>
                 <td>{person.salary}</td>
           </tr>
                ))}
     </tbody>
     </table>
     </div>

  );
}
```

ReactJS List

Example-3: Display List of Directors info using map() method.

```
function DirectorsPage() {
const directors = [
  { name: 'Director 1', movies: ['Movie 1', 'Movie 2', 'Movie 3'] },
  { name: 'Director 2', movies: ['Movie 4', 'Movie 5', 'Movie 6'] },
  // Add more directors if needed
];
```

ReactJS List

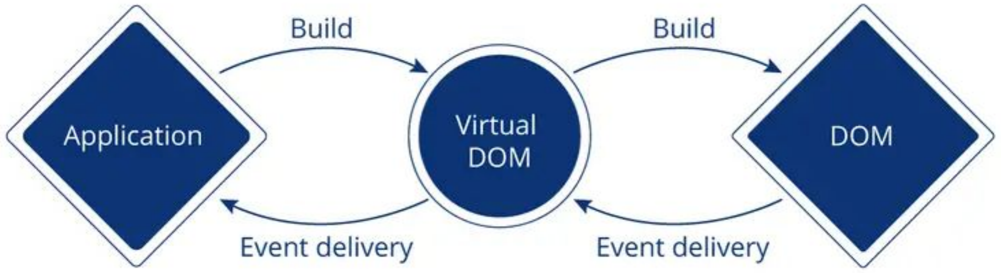
Example-3: Display List of Directors info using map() method.

```
return (
    <div className='container-fluid' style={{height: 500, width:500}}>
      <div className='card'>
      <h1>Directors Page</h1>
      {directors.map((director, index) => (
        <div key={index}>
          <h2>{director.name}</h2>
          <ol>
            {director.movies.map((movie, index) => (
              <li key={index}>{movie}</li>
            ))}
          </ol>
        </div>
      ))}
      </div>
</div>
  );
}export default DirectorsPage;
```

ReactJS Handling Events

- The system can start many different events, like clicking, page loading, key presses, scrolling, etc. React offers event management that is comparable to DOM element event handling.
- React events are synthetic events that cover the native event of the browser across all supported browsers. In every browser, it operates the same.

- Any time a user clicks on an object puts text into an input box, drags or drops an element, and so on, events often reflect some form of interaction between the user and the app.
- As a result, the app typically needs to respond to each event in some way. As programmers, we can construct a function that is used to react to a critical event whenever it occurs.
- Every programming language's heart of event management is a set of operations known as event handlers. A collection of methods are used to manage events in React.



ReactJS Handling Events

- Events are the things that javascript can react to. Event handling in React is quite similar to DOM element event handling.

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**

ReactJS Handling Events

React Events: Naming

Handling events with react is very similar to handling events in DOM elements, although there are some syntactic differences.

1. React events are written in camelCase.
2. A function is passed as the event handler rather than string.

The way to write events in html / DOM is below:

```
<button onclick=”handleClick()”>click me</button>
```

In React, events are named using camel case and you pass a function as event handler as shown below:

Like in a functional component, event is written like below:

```
<button onClick={handleClick}>click me</button>
```

In class based component ,event is written like below

```
<button onClick={this.handleClick}>click me</button>
```

ReactJS Handling Events

Let's see some of the event attributes:

1. **onmouseover : The mouse is moved over an element**
2. **onmouseup : The mouse button is released**
3. **onmouseout : The mouse is moved off an element**
4. **onmousemove: The mouse is moved**
5. **Onmousedown: mouse button is pressed**
6. **onload : A image is done loading**
7. **onunload: Exiting the page**
8. **blur : Losing Focus on the element**
9. **onchange : Content of a field changes**
10. **onclick: Clicking an object**
11. **ondblclick: double clicking an object**
12. **onfocus: element getting a focus**
13. **Onkeydown: pushing a keyboard key**
14. **Onkeyup: keyboard key is released**
15. **Onkeypress: keyboard key is pressed**
16. **Onselect: the text is selected**

ReactJS Handling Events

Example-1: Normal function triggering through onClick() event

```
// Normal Function
const EventExample1=()=>{
        function handleClick1() {
          console.log('Using functions Button clicked!');
        }
          return (
            <div>
                <button onClick={handleClick1}>Function Click </button>
            </div>
          );
}
```

ReactJS Handling Events

Example-2: **Arrow function triggering through onClick() event**

```
// Arrow Function
  const EventExample2=()=>{
    const handleClick2=()=> {
        console.log('Using Arrow functions Button clicked');
    }
      return (
        <div>
              <button onClick={handleClick2}>Arrow Function</button>
          </div>
      );
}
```

ReactJS Handling Events

Example-3: **Parameter passing to function triggering through onClick() event**

```
// Parameter to Function
    const EventExample3=()=>{
      const handleClick3=(param)=>{
        console.log('Parameter passing to function Button clicked:', param);
      }
      return (
        <div>
            <button onClick={()=>handleClick3('Hello, Parameter')}>Parameter Function</button>
        </div>
      );
  }
```

ReactJS Handling Events

Example-4: **function triggering through onClick(), onChange(), onSubmit() events**

```
const EventExample4=()=>{
    const handleClick=(e)=>{
      e.preventDefault();
      console.log('Button clicked!');
    }
    const handleChange=(e)=>{
      console.log('Input value:', e.target.value);
    }
   const handleSubmit=(e)=>{
      e.preventDefault();
      console.log('Form submitted!');
    }
```

contd…..

ReactJS Handling Events

Example-4: **function triggering through onClick(), onChange(), onSubmit() events**

```
  return(
     <div>
     {/* Handling click event */}
     <button onClick={handleClick}>Click me</button>

     {/* Handling change event */}
     <input type="text" onChange={handleChange} />

     {/* Handling form submission */}
     <form onSubmit={handleSubmit}>
       <input type="submit" value="Submit" />
     </form>
   </div>

   );
}
```

ReactJS Handling Events

Example-5: **Display Name based on input, through onChange() event**

```
const EventHandler1=()=>{

    const [input, setinput] = useState('');
    const ChangeText=(event)=>{       setinput(event.target.value);    }
    const ClickHandle=()=>{      console.log("Entered Value:", input);     }

    return(
      <div>
        <h5>Display Name Based on input through onChange Event...</h5>
        <label>Enter Name:</label>
                     <input   type="text"   id="name"   value={input}   placeholder="Enter   Name"
onChange={ChangeText} />
        <br/>
        <button onClick={ClickHandle}>Submit</button>
        <p>You Entered: <b className="text-danger">{input}</b></p>
      </div>
    );
  }
```

ReactJS Handling Events

Example-6: **Display DropDown List, through onChange() event handler**

```
const DropDownEvent=()=>{

  const subject=["subject1", "subject2", "subject3"]
  const country=["India", "US", "UK", "Canada"]
  const [selectoption, setselectoption] = useState('');
  const [subjects, setsubjects]=useState(subject);

  const SelectHandleChange=(event)=>{
    setselectoption(event.target.value);
  }
  const SubjectSelectHandle=(event)=>{
    setsubjects(event.target.value);
  }
```

Contd….

ReactJS Handling Events

Example-6: **Display DropDown List, through onChange() event handler**

```
return(
    <div className="container-fluid">
      <div>
      <lable>Select Country:</lable>
      <select value={selectoption} onChange={SelectHandleChange}>
      <option value="">select Option</option>
          {country.map((item)=>(
              <option value={item}>{item}</option>

        ))}
      </select>
      <br/>
      <p>You have Selected Option: <b className="text-danger">{selectoption}</b></p>
      </div>
```

Contd….

ReactJS Handling Events

Example-6: **Display DropDown List, through onChange() event handler**

```
   <hr/>
      <div>
      <lable>Select Subject:</lable>
      <select value={subjects} onChange={SubjectSelectHandle}>
      <option value="">select Option</option>
          {subject.map((item1)=>(
             <option value={item1}>{item1}</option>
        ))}
      </select>
      <br/>
      <p>You have Selected Option: <b className="text-danger">{subjects}</b></p>
      </div>
    </div>
  );
}
```

ReactJS Handling Events

Example-7: **Display Checkbox through onChange() event handler**

```
const CheckboxEventHandlerDemo=()=>{
  const [isChecked, setisChecked] = useState(false)
  const HandleCheckboxChange=(event)=>{
    setisChecked(event.target.checked)
  }
  return(
    <div className="container-fluid">
      <div>
      <label>
        <input type="checkbox" value={isChecked} onChange={HandleCheckboxChange}/>Checkbox
      </label>
      <p>checkbox is: {isChecked? 'checked': 'unchecked'}</p>
      </div>
    </div>
  );
}
```

ReactJS Handling Events

Example-8: **Display Radio through onChange() event handler**

```
const RadioButttonEventHandlerDemo=()=>{
  const [selectedoption, setselectedoption] = useState('');
  const GenderRadioHandle=(event)=>{
    setselectedoption(event.target.value)
  }
  return(
    <div className="container-fluid">
      <label>        Gender: </label>
                         <input   type="radio"   value="Male"   checked={setselectedoption==="male"}
onChange={GenderRadioHandle}/> Male
                      <input   type="radio"   value="Female"   checked={setselectedoption==="female"}
onChange={GenderRadioHandle}/> Female
      <div>
        Selected Gender: <b className="text-danger">{selectedoption}</b>      </div>
    </div>
  );
}
```

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. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.
- React Router is a **standard library system built on top of the React** and used to create routing in the React application using React Router Package.
- It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

Need of React Router

- React Router plays an important role to **display multiple views** in a single page application. Without React Router, it is not possible to display multiple views in React applications.
- Most of the social media websites like **Facebook, Instagram uses React Router** for rendering multiple views.

ReactJS

React Router

- Routing in reactJS is the mechanism by which we navigate through a website or web-application. Routing can be server-side routing or client-side routing. However, React Router is a part of client-side routing.

- React Routing without any knowledge of it can be manually implemented using useState and JSX for conditioning.

- But being inefficient for large-scale applications like e-commerce, it still can act as a boilerplate for understanding routing and act as a base for React JS router for example.

```
const[page, setpage] = useState("products")
const routeTo = (newpage) =>{
  setpage(newpage)
}
<button onClick={()= routeTo("cart")}> Cart />
{page === "cart" && (
<Cart cart={cart} setcart={setcart}/>)}
```

React Router

- React Router is a standard library for routing in React.
- It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.
- Let us create a simple application to React to understand how the React Router works.
- The application will contain three components the **home component, the About component, and the contact component.**

We will use React Router to navigate between these components.

Install React Router

Step 1: To install react-router in your application write the following command in your terminal

```
npm i react-router-dom
```

Step 2: Importing React Router

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

React Router

React Router Components:

The Main Components of React Router are:

1. **BrowserRouter:** BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState, and the popstate event) to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.

2. **Routes:** It’s a new component introduced in the v6 and an upgrade of the component.

   The main advantages of Routes over Switch are:

- Routes are chosen based on the best match instead of being traversed in order.

3. **Route:** Route is the conditionally shown component that renders some UI when its path matches the current URL.

4. **Link:** The link component is used to create links to different routes and implement navigation around the application. It works like an HTML anchor tag.

React Router

ReactRouterDemo

// App.js

```
import React from 'react’;
// npm install react-router-dom
import { BrowserRouter, Routes, Link, Route } from 'react-router-dom';

import Home from './Home';
import About from './About';
import Contact from './Contact’;
const App = () => {
  return (
    <BrowserRouter>
      <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>
```

React Router

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

export default App;
```

End Of UNIT-2