Department: Computer Science & Engineering
Semester: Summer-2025
Course Title: Web Programming
Course Code: CSE-479
Section: 02

Submitted To:

Md. Arman Hossain

Lecturer

Department of Computer Science and Engineering

Submitted By:

Asfar Hossain Sitab

Id: 2022-3-60-275

Department of Computer Science and Engineering

Date of Report Submission: 6th August 2025

React.js Fundamentals

Building Modern User Interfaces with React

Welcome to the World of React.js!

- What is React? A JavaScript library for building user interfaces.
- Developed and maintained by Facebook (now Meta).
- Used for building single-page applications (SPAs) and dynamic UIs.
- Focuses on the "View" layer of the application.

The "Why" Behind React's Popularity

- **Declarative:** You describe what the UI should look like, and React handles the rest.
- **Component-Based:** UIs are broken down into reusable, self-contained components.
- **Efficient:** Uses a Virtual DOM to minimize direct manipulation of the browser DOM.
- **Large Community:** Abundant resources, tutorials, and libraries.

Core Features of React

- **Virtual DOM:** A lightweight copy of the real DOM for faster updates.
- **JSX:** A syntax extension for JavaScript that looks like HTML.
- **Component-Based Architecture:** The foundation of all React apps.
- **Unidirectional Data Flow:** Data flows in one direction, making state management predictable.

React vs. The Competition

- **React (Library):** Flexible, "just the view," often paired with other libraries for a full stack.
- **Angular (Framework):** Opinionated, full-fledged framework with a steep learning curve.
- **Vue.js (Progressive Framework):** Easier to learn, very performant, and can be used progressively.
- **Summary:** React's flexibility and component model are its biggest differentiators.

Setting Up Your First React App

The easiest way to start a new React project is by using Create React App:

```
# Install Create React App (if not already installed)
npm install -g create-react-app

# Create a new React project named 'my-react-app'
npx create-react-app my-react-app

# Navigate into your new project directory
cd my-react-app

# Start the development server
npm start
```

See the output of these commands on the next slide.

Output: Setting Up React

Output: These commands set up a new React project with a ready-to-use development environment. After `npm start`, it will open your default browser to `http://localhost:3000` showing your new React app, typically with a spinning React logo.

Your React App is Running!

Open http://localhost:3000 to view it in the browser.

R

The Building Blocks: Components

- Everything in React is a component.
- Components are reusable and can be nested within each other.
- **Two types of components:**

  Functional Components: Simple, stateless (mostly). The modern standard.

  Class Components: State-driven, older syntax (still common in legacy code).

HTML in Your JavaScript? Meet JSX

JSX (JavaScript XML) allows you to write HTML-like code directly in your JavaScript files. It's not HTML, but it gets compiled into React elements.

```
// React Component using JSX
function App() {
  return (
    <div className="container">
      <h1>Hello, React World!</h1>
      <p>This is a paragraph inside JSX.</p>
    </div>
  );
}
```

See the visual output of this JSX on the next slide.

Output: JSX Syntax

Output: This JSX code would render a simple web page with a heading and a paragraph, styled by the `container` class.

Hello, React World!

This is a paragraph inside JSX.

Data In: Passing Information with Props

Props (short for properties) are how you pass data from a parent component to a child component. They are read-only.

```
// Child Component (Greeting.js)
function Greeting(props) {
  return <h2>Hello, {props.name}!</h2>;
}

// Parent Component (App.js)
function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
    </div>
  );
}
```

See how different props result in different outputs on the next slide.

Output: Props in React

Output: The `Greeting` component receives different `name` props and renders personalized greetings.

```
Hello, Alice!
Hello, Bob!
```

Data Out: Managing Component State (`useState`)

State allows components to manage data that changes over time, triggering re-renders.

```
// Counter Component (Counter.js)
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

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

See this interactive counter in action on the next slide.

Output: Counter Example

Output: A simple counter. Clicking the "Increment" button updates the `count` state, which then re-renders the component to show the new value.

Count: 0

Increment

Reacting to User Actions: Event Handling

Events in React are named using camelCase (e.g., `onClick`, `onChange`). You pass a function as the event handler.

```
function MyButton() {
  const handleClick = () => {
    console.log('Button was clicked!');
    // In a real app, you might update state, fetch data, etc.
  };

  return (
    <button onClick={handleClick} className="px-4 py-2 bg-blue-500 text-white rounded-md">
      Click Me
    </button>
  );
}
```

See the simulated output on the next slide.

Output: Event Handling

Output: Clicking the "Click Me" button would log "Button was clicked!" to your browser's console.

Click Me

(Check your browser's console for output when this code runs)

Showing and Hiding Components: Conditional Rendering

Control what is rendered based on a condition using `if`, ternary operator, or logical `&&`.

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

function LoginControl() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  let button;
  if (isLoggedIn) {
    button = <button onClick={() => setIsLoggedIn(false)}>Logout</button>;
  } else {
    button = <button onClick={() => setIsLoggedIn(true)}>Login</button>;
  }

  return (
    <div>
      {isLoggedIn ? <p>Welcome back!</p> : <p>Please log in.</p>}
      {button}
    </div>
  );
}
```

See how the UI changes based on the condition on the next slide.

Output: Conditional Rendering

Output: The text and button change based on the `isLoggedIn` state. This demonstrates how React can conditionally render elements.

Please log in.

Login

(Click "Login" to see change)

Displaying Collections: Lists and Keys

Use `map()` to render lists. Each list item needs a unique `key` prop for efficient updates.

```
// TodoList Component (TodoList.js)
function TodoList(props) {
  const todos = ['Learn React', 'Build an App', 'Deploy to Netlify'];

  return (
    <ul>
      {todos.map((todo, index) => (
        // Key should ideally be a stable ID from data, index is fallback
        <li key={index}>
          {todo}
        </li>
      ))}
    </ul>
  );
}
```

View the rendered Todo List on the next slide.

Output: Todo List Example

Output: A simple todo list. React efficiently updates the list items when data changes.

My Todos:

- Learn React
- Build an App
- Deploy to Netlify

Handling User Input: Forms

React uses "controlled components" where form data is handled by React state.

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

function NameForm() {
  const [name, setName] = useState('');

  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault(); // Prevent default form submission
    console.log('A name was submitted: ' + name);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange} className="border rounded-md px-2 py-1 ml-2"/>
      </label>
      <button type="submit" className="px-4 py-2 bg-indigo-500 text-white rounded-md hover:bg-indigo-600 mt-2">Submit</
```

See the interactive form output on the next slide.

Handling User Input: Forms

React uses "controlled components" where form data is handled by React state.

```
  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault(); // Prevent default form submission
    console.log('A name was submitted: ' + name);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange} className="border rounded-md px-2 py-1 ml-2"/>
      </label>
      <button type="submit" className="px-4 py-2 bg-indigo-500 text-white rounded-md hover:bg-indigo-600 mt-2">Submit</
button>
    </form>
  );
}
```

See the interactive form output on the next slide.

Output: Simple Form Example

Output: An input field where typing updates the state, and submitting shows a message with the entered name.

Name: Asfar Hossain Sitab

Submit

The Power of Hooks: `useState`

`useState` is the fundamental hook for adding state to functional components. It returns the current state value and a function to update it.

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

function TextInput() {
  const [message, setMessage] = useState('Hello, world!');

  const handleInputChange = (event) => {
    setMessage(event.target.value);
  };

  return (
    <div>
      <input
        type="text"
        value={message}
        onChange={handleInputChange}
        className="border rounded-md px-3 py-2"
      />
      <p className="mt-4 text-lg font-semibold">
        You typed: {message}
      </p>
    </div>
```

See this dynamic text input example on the next slide.

Output: `useState` Example (Text Input)

Output: As you type in the input box, the text below dynamically updates, demonstrating `useState` in action.

Hello, world!

You typed: Hello, world!

Side Effects in React: `useEffect`

`useEffect` lets you perform "side effects" like data fetching, subscriptions, or manual DOM manipulation after render.

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

function UserFetcher() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Simulate data fetching
    setTimeout(() => {
      setUser({ name: 'Jane Doe', email: 'jane.doe@example.com' });
      setLoading(false);
    }, 2000); // Simulate 2-second delay
  }, []); // Empty dependency array means run once on mount

  if (loading) {
    return <p>Loading user data...</p>;
  }

  return (
    <div>
      <h3>User Profile</h3>
```

See the simulated data fetching output on the next slide.

Output: `useEffect` Example (Data Fetch)

Output: Initially shows "Loading user data...", then after 2 seconds, displays the fetched user information. This simulates a common use case for `useEffect`.

Loading user data...

(Simulated delay of 2 seconds)

Navigating Your App: React Router

- React Router is the standard library for client-side routing.
- Enables single-page application navigation without full page reloads.
- **Key components:** `<BrowserRouter>`, `<Route>`, `<Link>`.
- Helps manage different pages/views in your application.

Passing Data Without Props: The Context API

- Avoids "prop drilling" (passing props through many intermediate components).
- Provides a way to share data (like user info or theme settings) that can be considered "global."
- **Three steps:** `createContext`, `Provider`, and `Consumer` (or the `useContext` hook).

Writing Better React Code

- Break down large components into smaller, reusable ones.
- Follow the **Single Responsibility Principle** for components.
- Use `PropTypes` or TypeScript for type checking.
- Keep state as local as possible.
- Write clean, readable code with consistent naming conventions.

Oops! Avoiding Common React Pitfalls

- **Forgetting keys** for list items.
- Mutating state directly (`` `state.count++` `` is bad; `` `setCount(count + 1)` `` is good).
- Unnecessary re-renders by not using the dependency array correctly in `useEffect`.
- Not understanding the difference between props and state.
- Putting a component inside another component's render method.

What's Next?

- **Recap:** React is a powerful library for building modern UIs.
- **Next Steps:**

  Build your own simple projects.
  Learn about more advanced hooks (`useReducer`, `useCallback`).
  Explore state management libraries like Redux or Zustand.
  Look into Next.js for server-side rendering and more.

- **Resources:**

  Official React Docs (reactjs.org)
  MDN Web Docs
  FreeCodeCamp, Scrimba, etc.