MODULE: 3 (JavaScript Essentials)

1. What is React Js?

- **React.js** is a popular open-source JavaScript library used for building user interfaces, particularly for single-page applications. It is maintained by Facebook and a community of developers. React allows developers to create reusable UI components and manage the state efficiently.
- **Key Features of React.js:**
- **Component-Based Architecture**:
- React divides the UI into small, reusable components, making the code modular and easier to manage.
- **Virtual DOM**:
- React uses a virtual DOM to efficiently update and render only the components that have changed, improving performance.
- **Declarative Syntax**:
- Developers can describe the UI with a declarative syntax, making the code predictable and easier to debug.
- **JSX (JavaScript XML)**:
- A syntax extension for JavaScript that allows mixing HTML with JavaScript, making the code more readable and expressive.
- **Unidirectional Data Flow**:
- Data flows in a single direction, ensuring better control and predictability in the application.
- **State and Props**:
- **State**: Manages the internal data of a component.
- **Props**: Passes data between components.
- **Extensive Ecosystem**:
- React has a rich ecosystem, including tools like React Router for navigation and libraries like Redux for state management.
- **Cross-Platform Development**:
- React Native, built on React.js, enables developers to create mobile applications using the same principles.
- **Benefits of Using React.js:**
- High performance due to Virtual DOM.
- Reusability of components reduces development time.
- Easier debugging with developer tools.
- Strong community support and extensive documentation.
- **Use Cases:**
- Dynamic web applications (e.g., Facebook, Instagram).
- Single-page applications (SPAs).
- Mobile applications (using React Native).
- React.js is widely adopted for its efficiency, flexibility, and ease of use, making it an excellent choice for modern web development.

2. What is NPM in React Js?

- **NPM (Node Package Manager)** is a tool that comes with Node.js and is used to manage packages (libraries and dependencies) for JavaScript applications, including React.js projects. It allows developers to install, update, and manage libraries that simplify development.
- In the context of React.js, NPM is crucial because it:
- Provides access to the React library and related tools.
- Helps manage other dependencies, such as state management libraries, routing tools, and build tools.
- ---
- **Key Roles of NPM in React.js:**
- **Installing React and Other Libraries**:
- You can use NPM to install React and ReactDOM:
- bash
- npm install react react-dom
- **Dependency Management**:
- NPM tracks all the dependencies of a project in a package.json file. This file lists all the required packages for the project and their versions.
- **Running Scripts**:
- The package.json file contains scripts (e.g., start, build, test) that can be run using NPM:
- bash
- npm start
- **Version Control of Packages**:
- NPM allows specifying exact or compatible versions of libraries to ensure the application behaves consistently across environments.
- **Sharing Custom Packages**:
- Developers can publish their own libraries or packages to NPM, making them reusable by others.
- **Global and Local Package Installation**:
- **Global**: Installs tools accessible system-wide (e.g., npm install -g create-react-app).
- **Local**: Installs packages for a specific project.
- ---
- **Benefits of Using NPM in React:**
- Access to a vast repository of packages.
- Easy to install, update, and remove dependencies.
- Ensures consistency across development environments.
- Simplifies project setup with tools like create-react-app.
- ---

- **Typical Workflow in a React Project Using NPM:**

- **Initialize a Project**:
- bash
- npm init -y
- **Install Dependencies**:
- bash
- npm install react react-dom
- **Run the Application**:
- bash
- npm start
- In short, NPM is an essential tool for managing the ecosystem around React.js, streamlining development and enabling the use of third-party libraries effectively.

3. What is Role of Node Js in react Js?

- **Node.js** plays a critical role in React.js development, especially during the development and build process. While React is used for building user interfaces in the browser, **Node.js** acts as a backend tool that supports development tasks such as building, bundling, and running React applications.
-

- **Key Roles of Node.js in React.js Development:**
- **Development Environment**:
- Node.js provides the environment to execute JavaScript code outside the browser. This is essential for tools like create-react-app, which is used to set up a React project.
- **Package Management (NPM)**:
- Node.js comes with **NPM (Node Package Manager)**, which is used to install and manage the packages and dependencies required for React projects (e.g., React, React Router, Webpack, Babel).
- **Build Tools**:
- React applications often use build tools like Webpack and Babel to:
- Transpile modern JavaScript (ES6/ESNext) into browser-compatible JavaScript.
- Bundle multiple JavaScript files into a single file for efficient delivery.
- These tools run in the Node.js environment.
- **Server-Side Rendering (SSR)**:
- React applications can be rendered on the server side using **Node.js**. This improves performance and SEO by delivering pre-rendered HTML to the browser.
- **Development Server**:
- During development, Node.js is used to run a local development server (e.g., with webpack-dev-server or tools like Vite) that provides hot reloading and live updates as you code.
- **API Proxying**:
- Node.js can proxy API calls during development, which is useful when your frontend React app communicates with a backend API hosted on a different server.

- **Code Compilation**:
- Tools like Babel (to transpile JSX and ES6+ JavaScript) and Webpack (to bundle files) are executed using Node.js.
- ---
- **Why Node.js is Important for React Development:**
- It simplifies the project setup and dependency management.
- Supports modern JavaScript tooling and workflows.
- Enables developers to run the same JavaScript language both on the client (React) and server (Node.js).
- Facilitates SSR for better performance and SEO.
- Enhances development efficiency with tools like hot reloading.
- ---
- **Does Node.js Run React Applications in Production?**
- **No**, Node.js is not required to run React applications in production. React applications are static files (HTML, CSS, and JavaScript) that can be served by any web server (e.g., Nginx, Apache).
- Node.js is essential for the development process, but it's optional in production unless you use server-side rendering or build a full-stack app with React and Node.js.
- In summary, Node.js is a vital part of the development process for React.js applications, providing the environment and tools needed to build, bundle, and run the app efficiently.

4. What is CLI command In React Js?

- **CLI (Command Line Interface)** commands in React.js refer to commands executed in the terminal or command prompt to interact with tools or frameworks used in React development. These commands help developers perform various tasks, such as creating new projects, running a development server, or building the application.
- ---
- **Common CLI Commands in React.js:**
- **1. Create a New React Application**
- The create-react-app tool is commonly used to set up a new React project:
- bash
- npx create-react-app my-app
- npx: Executes the package without globally installing it.
- create-react-app: A React project scaffolding tool.
- my-app: The name of the new React project directory.
- **2. Navigate to the Project Directory**
- bash
- cd my-app
- **3. Start the Development Server**
- Run the React application locally for development:
- bash
- npm start
- Opens the app in the default browser and listens for changes.

- **4. Build the Application**
- Generate an optimized production-ready build:
- bash
- npm run build
- Creates a build folder with the compiled static files for deployment.
- **5. Test the Application**
- Run unit tests:
- bash
- npm test
- Starts the test runner in interactive watch mode.
- **6. Eject Configuration**
- Expose the configuration files for customization (use with caution):
- bash
- npm run eject
- This action is irreversible and is not commonly needed.
- **7. Install Dependencies**
- Add new libraries or packages to the project:
- bash
- npm install <package-name>
- Example:
- bash
- npm install react-router-dom
- **8. Remove Dependencies**
- Uninstall packages:
- bash
- npm uninstall <package-name>
- **9. Upgrade Dependencies**
- Update all packages to their latest versions:
- bash
- npm update
- **10. Run Custom Scripts**
- Scripts defined in the package.json file can be run via NPM:
- bash
- npm run <script-name>
- Example:
- bash
- npm run lint
- ---
- **CLI Benefits in React.js:**
- Automates repetitive tasks (e.g., project setup, dependency management).
- Simplifies development workflows.
- Reduces the need for manual configuration.

- **Conclusion:**
- CLI commands streamline the React.js development process, enabling developers to create, test, and manage applications efficiently. Mastering these commands enhances productivity and simplifies project management.

5. What is Components in React Js?

- In React.js, **components** are the building blocks of the user interface. They allow developers to break the UI into smaller, reusable, and independent pieces, making the code modular and easier to manage.
- ---
- **Key Features of Components:**
- **Reusability**:
- A component can be reused multiple times, reducing redundancy in the codebase.
- **Modularity**:
- Each component encapsulates its logic, styling, and rendering, making it self-contained.
- **Composition**:
- Components can be nested within other components to create complex UIs.
- **Separation of Concerns**:
- Components focus on a single responsibility, making the code easier to debug and maintain.
- ---
- **Types of Components:**
- **Functional Components**:
- Written as JavaScript functions.
- Introduced with React Hooks, functional components can now handle state and lifecycle methods.
- Syntax is simpler and preferred in modern React development.
- Example:
- javascript
- function Greeting(props) {
- return <h1>Hello, {props.name}!</h1>;
- }
- **Class Components**:
- Written as ES6 classes.
- Used to include state and lifecycle methods (before Hooks were introduced).
- Example:
- javascript
- class Greeting extends React.Component {
- render() {
- return <h1>Hello, {this.props.name}!</h1>;
- }

- }
- ---
- **Core Concepts in Components:**
- **Props (Properties)**:
- Used to pass data from a parent component to a child component.
- Props are immutable and read-only.
- Example:
- javascript
- function Welcome(props) {
- return <h1>Welcome, {props.username}!</h1>;
- }
- **State**:
- Represents the component's internal data.
- Can be updated, triggering a re-render of the component.
- Example (using Hooks):
- javascript
- Copy code
- function Counter() {
- const [count, setCount] = React.useState(0);
-
- return (
-
- Count: {count}
- <button onClick={() => setCount(count + 1)}>Increment</button>
-
-);
- }
- **Lifecycle Methods**:
- Available in class components to manage component lifecycle events (e.g., mounting, updating, unmounting).
- In functional components, these are managed using Hooks like useEffect.
- ---
- **Benefits of Components:**
- **Code Reusability**: Reduces duplication and improves consistency.
- **Maintainability**: Smaller components are easier to test and maintain.
- **Better Collaboration**: Components can be developed independently by different team members.
- **Improved Readability**: Each component serves a specific purpose, making the code easier to understand.
- ---
- **Example of a Simple React Component:**
- JavaScript

- function App() {
- return (
-
- <Header />
- <Content />
- <Footer />
-
-);
- }
-
- function Header() {
- return <h1>Welcome to My Website</h1>;
- }
-
- function Content() {
- return This is the main content area. ;
- }
-
- function Footer() {
- return <footer>© 2024 My Website</footer>;
- }
- In this example, Header, Content, and Footer are all reusable components that together make up the App.
- **Conclusion:**
- Components are the essence of React.js, enabling developers to create scalable, maintainable, and reusable UIs. By organizing the UI into components, React simplifies the process of building complex applications
  .

6. What is Header and Content Components in React Js?

- In React.js, **Header** and **Content** are examples of components that represent distinct parts of a user interface. These components are typically used to divide the structure of a web application into logical sections, improving organization and reusability.
-

- **1. Header Component**
- The **Header** component represents the top section of a webpage or application. It usually contains:
- The website's logo.
- Navigation links (e.g., Home, About, Contact).
- A title or tagline.
- Sometimes, a search bar or user-related options (e.g., login, profile).
- **Example of a Header Component:**
- javascript

- function Header() {
- return (
- <header style={{ backgroundColor: "#333", color: "#fff", padding: "10px" }}>
- <h1>My Website</h1>
- <nav>
- Home
- About
- Contact
- </nav>
- </header>
-);
- }
- ---
- **2. Content Component**
- The **Content** component represents the main section of a webpage or application. It usually contains:
- The core information or functionality of the page.
- Text, images, forms, or any interactive elements.
- It dynamically changes based on the route or the user's actions.
- **Example of a Content Component:**
- javascript
- function Content() {
- return (
- <main style={{ padding: "20px" }}>
- <h2>Welcome to My Website</h2>
- This is where the main content goes.
-
- </main>
-);
- }
- ---
- **Usage in a React Application**
- You can combine the **Header** and **Content** components in a parent component (e.g., App) to create a complete webpage structure.
- **Example:**
- javascript
- function App() {
- return (
-
- <Header />
- <Content />
-

-);
- }
-
- export default App;
- ---
- **Styling and Props**
- Both Header and Content can accept **props** to dynamically update their content.
- Styling can be applied using CSS, inline styles, or styled-components.
- **Example with Props:**
- javascript
- function Header(props) {
- return (
- <header style={{ backgroundColor: "#333", color: "#fff", padding: "10px" }}>
- <h1>{props.title}</h1>
- <nav>
- Home
- About
- Contact
- </nav>
- </header>
-);
- }
-
- function Content(props) {
- return (
- <main style={{ padding: "20px" }}>
- <h2>{props.subtitle}</h2>
- {props.text}
- </main>
-);
- }
-
- // Usage
- function App() {
- return (
-
- <Header title="My Awesome Website" />
- <Content subtitle="Welcome!" text="This is the main content area." />
-
-);
- }
- ---

- **Conclusion**
- The **Header** component structures the top navigation and branding of your application.
- The **Content** component holds the main information or functionality of your app.
- Both components enhance reusability and modularity, making the code cleaner and easier to maintain.

7. How to install React Js on Windows, Linux Operating System? How to install NPM and How to check version of NPM?

- React.js requires **Node.js** and **npm** (Node Package Manager) to set up and manage dependencies. Here's a step-by-step guide:
- ---
- **1. Installing Node.js and npm**
- **a. On Windows:**
- Download Node.js:
- Visit Node.js official website.
- Download the **LTS** version (recommended for most users).
- Install Node.js:
- Run the installer and follow the setup instructions.
- Ensure the option "Add to PATH" is selected during installation.
- Verify installation:
- Open **Command Prompt** or **PowerShell**.
- Check Node.js version:
- bash
- node -v
- Check npm version:
- bash
- npm -v
- **b. On Linux:**
- Update your package manager:
- bash
- sudo apt update
- Install Node.js and npm:
- Using Ubuntu/Debian:
- bash
- sudo apt install nodejs npm
- Or install Node.js via NodeSource (recommended for the latest version):
- bash
- curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
- sudo apt-get install -y nodejs
- Verify installation:
- bash

- node -v
- npm -v
- ---
- **2. Installing React.js**
- Once Node.js and npm are installed, you can create a React project.
- Open a terminal or command prompt.
- Use **npx** to create a React app:
- bash
- npx create-react-app my-app
- my-app is the name of your project directory.
- npx ensures you use the latest version of create-react-app.
- Navigate to the project directory:
- bash
- cd my-app
- Start the development server:
- bash
- npm start
- This opens the React app in your default web browser.
- ---
- **3. Installing npm (If Not Installed with Node.js)**
- If npm is not installed with Node.js, you can manually install or update it:
- Install npm:
- bash
- sudo npm install -g npm
- Verify installation:
- bash
- npm -v
- ---
- **4. Checking the Version of npm**
- To check the installed version of npm:
- Open your terminal or command prompt.
- Run the following command:
- bash
- npm -v
- This displays the current version of npm.
- ---
- **Summary**
- **React.js** requires Node.js and npm for installation.
- Use the npx create-react-app command to set up a new React project.
- Always verify the versions of Node.js and npm after installation:
- Node.js: node -v
- npm: npm -v

8. How to check version of React Js?

- To check the version of React.js in your project, you can use the following methods:
- ---
- **1. Using npm Command**
- Run the following command in your project directory:
- bash
- npm list react
- This will display the React version installed in your project.
- ---
- **2. Check in package.json**
- Open the package.json file in the root of your React project.
- Look for the dependencies section. The React version will be listed as:
- json
- "dependencies": {
- "react": "^18.0.0",
- "react-dom": "^18.0.0"
- }
- ---
- **3. Using React.version in Code**
- You can check the React version directly in your code (useful for debugging):
- Open any component file.
- Add the following code:
- javascript
- import React from 'react';
-
- console.log(`React version: ${React.version}`);
- Run your application with npm start and check the browser's developer console to see the version.
- ---
- **Summary**
- Use npm list react in the terminal for a quick check.
- Check the package.json file for dependency details.
- Use React.version in your code to log the version dynamically.

9. How to change in components of React Js?

In React.js, you can make changes to components by modifying their code, updating their state, or passing different props. Here's a guide on how to change components:

---

1. Edit the Component Code

You can directly change the JSX or logic inside the component's function or class.

Example:

```
javascript
function Header() {
  return (
    <header style={{ backgroundColor: "#333", color: "#fff", padding:
"10px" }}>
      <h1>Updated Website Title</h1> {/* Changed Title */}
      <nav>
        <a href="#home" style={{ margin: "0 10px", color: "#fff"
}}>Home</a>
        <a href="#about" style={{ margin: "0 10px", color: "#fff"
}}>About</a>
        <a href="#services" style={{ margin: "0 10px", color: "#fff"
}}>Services</a> {/* Added a new link */}
      </nav>
    </header>
  );
}
```

---

2. Update State in a Component

If the component uses **state**, you can modify its behavior dynamically by updating the state.

Example:

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

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

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increase</button> {/*
Updated behavior */}
      <button onClick={() => setCount(0)}>Reset</button> {/* Added a reset
button */}
    </div>
  );
}
```

---

3. Pass Different Props

You can change the component's appearance or behavior by passing new **props** from the parent component.

Parent Component:

```
javascript
function App() {
  return (
    <div>
      <Header title="Dynamic Title" />
    </div>
  );
}
```

Header Component:

```
javascript
function Header({ title }) {
  return <h1>{title}</h1>; // Updated to accept a prop
}
```

---

4. Apply Conditional Rendering

You can modify the component to display different content based on conditions.

Example:

```
javascript
function Greeting({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Sign In</h1>} {/*
Conditional Rendering */}
    </div>
  );
}
```

---

5. Add Styles or Classes

You can update the styles of the component by adding CSS or modifying inline styles.

Example:

```
javascript
function StyledButton() {
  return <button className="primary-btn">Click Me</button>;
}

/* CSS */
.primary-btn {
  background-color: blue;
  color: white;
  padding: 10px 20px;
}
```

---

6. Use Lifecycle Methods (Class Components)

In class components, you can change the behavior by adding or modifying **lifecycle methods**.

Example:

```
javascript
class Timer extends React.Component {
  componentDidMount() {
    console.log("Timer started"); // Updated lifecycle behavior
  }

  render() {
    return <h1>Timer Component</h1>;
  }
}
```

---

7. Replace or Add New Elements

You can replace existing elements or add new ones to extend functionality.

Example:

Adding a footer to an existing app:

```
javascript
function App() {
  return (
    <div>
      <Header />
      <Content />
      <Footer /> {/* New Component Added */}
    </div>
  );
}

function Footer() {
  return <footer>© 2024 My Website</footer>;
}
```

Summary

- **Edit Component Code:** Directly modify JSX and logic.
- **Update State:** Dynamically change behavior with `useState` or `setState`.
- **Pass Props:** Use props to control component input dynamically.
- **Conditional Rendering:** Display content based on conditions.
- **Add Styles:** Update appearance with CSS or inline styles.
- **Use Lifecycle Methods:** Modify behavior in class components.
- **Extend Structure:** Add or replace elements for new functionality.

10. How to Create a List View in React Js?

Creating a **List View** in React.js involves iterating over an array of data and rendering each item in the list dynamically using the `map()` method. Here's a step-by-step guide:

---

1. Basic Example of a List View

Render a simple list of items:

Example:

```
javascript
import React from 'react';

function ListView() {
  const items = ["Apple", "Banana", "Cherry", "Date", "Elderberry"];
```

```
  return (
    <div>
      <h1>Fruit List</h1>
      <ul>
        {items.map((item, index) => (
          <li key={index}>{item}</li> // Use a unique key for each item
        ))}
      </ul>
    </div>
  );
}

export default ListView;
```

---

2. List View with Objects

If the data is an array of objects, you can render properties of each object.

Example:

```
javascript
import React from 'react';

function ListView() {
  const users = [
    { id: 1, name: "John Doe", age: 25 },
    { id: 2, name: "Jane Smith", age: 30 },
    { id: 3, name: "Sam Brown", age: 22 },
  ];

  return (
    <div>
      <h1>User List</h1>
      <ul>
        {users.map((user) => (
          <li key={user.id}>
            {user.name} - {user.age} years old
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ListView;
```

---

3. Styled List View

You can style the list using CSS or inline styles.

Example:

```
javascript
import React from 'react';

function ListView() {
  const items = ["Home", "About", "Services", "Contact"];
  const listStyle = { listStyleType: "none", padding: 0 };

  return (
    <div>
      <h1>Menu</h1>
      <ul style={listStyle}>
        {items.map((item, index) => (
          <li key={index} style={{ margin: "10px 0" }}>
            <a href={`#${item.toLowerCase()}`} style={{ textDecoration:
"none", color: "blue" }}>
              {item}
            </a>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ListView;
```

---

4. Interactive List View

Add interactivity by updating the state when an item is clicked.

Example:

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

function ListView() {
  const [clickedItem, setClickedItem] = useState(null);
  const items = ["React", "Angular", "Vue", "Svelte"];

  return (
    <div>
      <h1>Frontend Frameworks</h1>
      <ul>
        {items.map((item, index) => (
          <li
            key={index}
            onClick={() => setClickedItem(item)}
            style={{ cursor: "pointer", color: clickedItem === item ?
"green" : "black" }}
          >
            {item}
```

```
          </li>
        ))}
      </ul>
      {clickedItem && <p>You clicked on: {clickedItem}</p>}
    </div>
  );
}

export default ListView;
```

---

5. Dynamic List with API Data

Fetch data from an API and display it as a list.

Example:

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

function ListView() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then((response) => response.json())
      .then((data) => setPosts(data.slice(0, 5))); // Limit to first 5
items
  }, []);

  return (
    <div>
      <h1>Posts</h1>
      <ul>
        {posts.map((post) => (
          <li key={post.id}>
            <strong>{post.title}</strong>
            <p>{post.body}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ListView;
```

Summary

- Use the `map()` function to iterate over an array and render list items dynamically.
- Provide a **unique `key`** to each list item to help React identify and update the list efficiently.
- Customize the list using **CSS** or **inline styles**.
- Add interactivity using **state**.
- Fetch data dynamically to create a list from an API or backend

11. Create Increment decrement state change by button click?

- To create an increment and decrement state change by button click in React, you can use the useState hook to manage the state of a number and then update it when the buttons are clicked. Here's an example:
- **Example: Increment and Decrement Counter**
- javascript
- import React, { useState } from 'react';
-
- function Counter() {
- // Initialize state with a starting value of 0
- const [count, setCount] = useState(0);
-
- // Function to increment the count
- const increment = () => setCount(count + 1);
-
- // Function to decrement the count
- const decrement = () => setCount(count - 1);
-
- return (
- <div style={{ textAlign: "center", marginTop: "50px" }}>
- <h1>Counter: {count}</h1> {/* Display the current count */}
-
- {/* Buttons to increment and decrement */}
- <button onClick={increment} style={{ marginRight: "10px" }}>Increment</button>
- <button onClick={decrement}>Decrement</button>
-
-);
- }

- export default Counter;
- **Explanation:**
- **useState(0):** The useState hook initializes the count state to 0.
- **increment:** This function increases the count by 1 when the "Increment" button is clicked.
- **decrement:** This function decreases the count by 1 when the "Decrement" button is clicked.
- **Buttons:** The two buttons are used to trigger the increment and decrement functions when clicked.
- ---
- **How it works:**
- When you click the **Increment** button, the count increases by 1.
- When you click the **Decrement** button, the count decreases by 1.
- The updated value is displayed in the <h1> tag dynamically.
- ---
- **Example Output:**
- Initially, the counter shows 0.
- Clicking **Increment** will change the counter to 1, 2, 3, etc.
- Clicking **Decrement** will change it back to 0, -1, -2, etc.
- This is a simple way to manage and update state using buttons in React.