Class 06 – React JS

INTRODUCTION TO REACT JS

MERN STACK DEVELOPMENT - MODULE B

Introduction to React JS

React is a powerful JavaScript library, developed by **Facebook,** designed for building dynamic user interfaces. It stands out for its efficiency and flexibility, making it an ideal choice for creating interactive web applications.

What sets React apart is its component-based architecture, promoting code reusability and maintainability. In React, UIs are described declaratively, and the library efficiently handles updates to the Document Object Model (DOM) through its **virtual DOM mechanism.**

React finds widespread use in industry giants like **Facebook, Airbnb, and Netflix,** showcasing its capabilities in developing scalable and high-performance applications. Its popularity stems from the ability to **create single-page applications** and build engaging user interfaces.

React's Core Concepts

React revolves around fundamental concepts that empower the creation of dynamic user interfaces. At its core are **Components**, the building blocks that encapsulate various UI elements. These components come in two main types: functional components, which are stateless, and class components, which can manage state.

JSX (JavaScript XML) is a distinctive syntax extension employed in React for crafting component structures. It integrates HTML-like code into JavaScript, enhancing the readability of components.

Managing data within components is facilitated through the concepts of **State and Props**. The internal state of a component is handled by the **useState hook** or **this.state** in functional and class components, respectively. External inputs are managed through props, allowing for dynamic and interactive components.

React's efficiency is notably attributed to the implementation of a **Virtual DOM**, a lightweight in-memory representation of the actual DOM. This approach minimizes unnecessary re-renders, resulting in a more performant application.

Why React JS…?

Declarative Syntax: React allows developers to describe the desired UI state, and it takes care of the underlying complexities. This declarative syntax makes code more intuitive and easier to maintain.

Component-Based Architecture: The modular nature of React, built around components, fosters reusability. Components encapsulate functionality, making it simple to manage and scale complex applications.

Efficient Virtual DOM: React's Virtual DOM ensures optimal performance by updating only the necessary parts of the actual DOM. This leads to faster rendering and a smoother user experience.

Single Page Application (SPA) Development: React is ideal for building SPAs, where a single HTML page is dynamically updated as the user interacts with the app. This approach enhances speed and responsiveness.

React Native for Cross-Platform Development: React extends its reach to mobile app development through React Native. This allows developers to use React principles to build native mobile applications for iOS and Android.

JSX - JavaScript XML

JSX, or JavaScript XML, is a syntax extension commonly associated with React that enhances the way we write components in JavaScript. It brings a more readable and concise structure to defining UI elements within our code.

In essence, JSX allows us to write HTML-like code directly within JavaScript files. This integration of markup directly into our JavaScript makes the creation and understanding of React components more intuitive.

example

For example, a simple component written in JSX might look like this:

```
const MyComponent = () => {

    return <div>
                <h1>Hello, JSX!</h1>
                <p>This is a JSX-powered component.</p>
            </div>

};
```

Create a new React JS Project

Creating a new React JS project is a fundamental step in initiating your development journey. To set up a project, we commonly use the powerful tool called Create React App (CRA).

Open terminal or command prompt and run this command with the project name at the end, e.g.: my-app

```
npx create-react-app my-app
```

After creating React project, go to project directory:

```
cd my-app
```

Start project:

```
npm run start
```

Components in React

React's core architecture revolves around the concept of components. Components are the building blocks that encapsulate various UI elements, allowing for a modular and reusable approach to web development.

In React, a component can be either a Functional Component or a Class Component. Functional components are stateless and primarily responsible for presenting UI elements. Class components, on the other hand, can manage state, lifecycle methods, and are generally used for more complex logic.

example

```
// SimpleComponent.jsx

import React from 'react';

const SimpleComponent = () => {

return (
    <div>
                <h1>Hello, Functional Components!</h1>
                <p>This is a simple functional component.</p>
    </div>
) }

export default SimpleComponent;
```

NPM & NPX

NPM

NPM, or Node Package Manager, serves as a crucial tool in Node.js development. It functions as a package manager, enabling developers to effortlessly install, share, and manage third-party libraries and tools within their projects. NPM simplifies the process of incorporating dependencies, fostering collaboration and maintaining an organized project structure. With an extensive registry housing countless open-source packages, NPM has become an integral component of modern JavaScript and Node.js workflows.

NPX

NPX complements NPM by providing a convenient way to execute Node.js binaries. Unlike NPM, NPX allows developers to run these binaries **without the necessity of global installations.** This proves beneficial for managing dependencies on a per-project basis, eliminating concerns about global package conflicts or version discrepancies. NPX is particularly useful for executing one-off commands and seamlessly integrating packages into a project's build process, enhancing the efficiency and portability of Node.js applications.

Real DOM & Virtual DOM

Real DOM

The Real DOM, or Document Object Model, is the browser's live representation of the HTML document. It consists of a tree-like structure where each HTML element is a node. When changes occur in a web application, the Real DOM is updated, and this process triggers a reflow and repaint of affected elements. Direct manipulations of the Real DOM, while providing an accurate reflection of the current state, can be resource-intensive, especially in scenarios involving frequent updates or dynamic content. This traditional approach often leads to performance bottlenecks, impacting the overall responsiveness of web applications.

Virtual DOM

In contrast, the Virtual DOM is a lightweight, in-memory copy of the Real DOM maintained by React. When changes are made in a React application, they are first applied to the Virtual DOM rather than the Real DOM. This virtual representation allows React to analyze and calculate the most efficient way to update the Real DOM. By determining the minimal set of changes needed, React optimizes the update process. Once the calculations are complete, React updates the Real DOM only where necessary, minimizing the performance overhead associated with direct manipulations. This strategy significantly enhances the efficiency and responsiveness of web applications built with React.

CSR & SSR

CSR - Client-Side Rendering

Client-Side Rendering (CSR) is a web development approach where the rendering of web pages is primarily handled by the client's browser. In CSR, the initial HTML content is minimal, often consisting of a basic shell or a loading indicator. The majority of the rendering and content display occurs on the client side, driven by JavaScript. As a user navigates through the application, the client fetches data from the server and dynamically updates the DOM, resulting in a more interactive and dynamic user experience. CSR is commonly associated with JavaScript frameworks like React, Angular, and Vue, which excel at managing client-side rendering tasks.

SSR - Server-Side Rendering

Server-Side Rendering (SSR) is an alternative approach in web development where the server generates the complete HTML content for a web page and sends it to the client's browser. In SSR, the server processes the requests, fetches the necessary data, and renders the HTML on the server side before sending it to the client. This results in a fully formed HTML page delivered to the browser, providing quicker initial page loads and improved search engine optimization (SEO) as search engine crawlers can easily index the content. While SSR offers advantages in terms of initial load performance and SEO, it can require more server resources compared to CSR, where the client handles a significant portion of the rendering process.

SPA & MPA

Single Page Application (SPA)

A Single Page Application (SPA) is a web application design paradigm that aims to provide a seamless and fluid user experience by loading and updating content within a single HTML page dynamically. SPAs utilize JavaScript frameworks like React, Angular, or Vue to fetch and render data without the need for full page reloads. This results in faster and more responsive applications, as only the relevant portions of the page are updated, reducing server round trips. SPAs often rely on client-side routing to manage navigation and maintain a smooth user interface. Popular examples of SPAs include Gmail and Facebook, where interactions occur without navigating to new pages.

Multi-Page Application (MPA)

In contrast, a Multi-Page Application (MPA) follows the traditional web application model, where each interaction or request typically results in the loading of a new HTML page from the server. MPAs often involve full page reloads when navigating between different sections or features of the application. Unlike SPAs, MPAs may require more initial loading time, as entire pages need to be retrieved from the server. Common web applications, such as blogs, e-commerce platforms, and content-driven websites, often adopt the MPA architecture. While MPAs may have slightly longer load times, they can offer better search engine optimization (SEO) due to distinct URLs for each page.

Declarative & Imperative

Declarative

Declarative programming is a paradigm that emphasizes expressing the desired outcome or result without specifying the step-by-step process of achieving it. In the context of React and web development, declarative code describes what the UI should look like based on the current state or conditions. React, being a declarative framework, allows developers to articulate the structure and behavior of components in a straightforward manner. By focusing on the "what" rather than the "how," declarative code tends to be more concise and easier to understand. React's JSX syntax is an example of declarative programming, where UI components are expressed in a manner similar to HTML, making the code more readable and expressive.

Imperative

In contrast, imperative programming involves explicitly specifying the step-by-step instructions or procedures to achieve a desired outcome. Instead of declaring the end result, developers outline the exact sequence of actions that the computer must take. In the context of React, imperative code might involve manually manipulating the DOM or updating the UI through direct commands. While imperative code can be powerful and provides fine-grained control, it often leads to more verbose and complex implementations. React encourages a declarative approach, allowing developers to focus on defining the desired UI state rather than intricately detailing how to achieve it, resulting in cleaner and more maintainable code.

That's it for today!

Keep coding, stay curious, and enjoy your JavaScript journey!