CodeDojo Module : Introduction to ReactJS

Lesson 1: What is ReactJS?

1.1 Introduction to React

What is React

- React is a Javascript library created by Facebook, primary used for building user interfaces, especially for single-page-application (SPA)
- Initially developed to handle Facebook's complex UIs, React was released as an open-source project and gained popularity due to its efficient rendering and component-based architecture
- **Real-world uses** : React powers application by companies like Facebook, Instagram, Airbnb, Uber, and Netflix

Why Use React?

- **Component-Based** : Breaks down the UI into reusable components, making code more modular and maintainable
- **Declarative UI** : React updates the view when data changes, allowing developers to describe the intended state without directly manipulating the DOM
- **Virtual DOM** : React creates a virtual representation of the DOM, leading to faster UI updates by selectively rendering only changed parts of the page.
- **Single page application (SPA)** : SPAs are web applications that load a single HTML page and dynamically update it as the user interacts with the app. React simplifies building SPAs by enabling seamless updates to the UI without reloading the page

1.2 Core Principle of React

1. Declarative Programming

- In React, developers write declarative code that describes what the UI should look like for any given application state, and React takes care of the how behind updating it
- Contracts with imperative programming, where each UI change has to be handled manually

2. Component-Based Architecture

- Components are the building blocks of React. Each component is self-contained, managing its own content, structure and behavior
- Components can be nested, reused and composed to build complex Uis from small, reusable parts

3. Virtual DOM and React Reconciliation

- The Virtual DOM is a lightweight representation of the actual DOM in memory
- React updates only the part of the DOM that have changed, leading to faster performance that direct DOM manipulation

1.3 Setting up the Environment with Vite

Node.js and npm Installation

- Verify Node.js by running `node -v` in the terminal. If needed, download from nodejs.org.

Using Vite for a New React Project

- **What is Vite?**
    - Vite is a fast, modern build tool that offers faster development builds and hot module replacement (HMR). It's particularly effective with React applications, as it allows for near-instant page refreshes.

- **Setting up a New React Project with Vite**

In the terminal, run:

```
npm create vite@latest my-first-app
```

This command creates a new React project with Vite in a folder named `my-first-app`.

- **Running the React App**

Navigate to the project directory and start the development server

```
cd my-first-app
npm install
npm run dev
```

By default, Vite runs the app at `http://localhost:5173`.

1.4 Project Structure in a Vite React App

1. Key Files and Folders

- *Public* folder : Stores static assets (e.g images and icon)
- *Src* folder : Holds all Javascript files, components and styles
- *Main.jsx :* Entry point where the root component *(App)* is rendered to the DOM
- *App.jsx* : Main components of the app, usually containing high-level layout and routes

2. Understanding *main.jsx* and *App.jsx*

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

ReactDOM.createRoot(document.getElementById('root')).render(<App />);
```

App.jsx is the main component where other components will be nested and rendered

1.5 React Developer Tools (Optional)

1. Installing React DevTools

- The React Developer Tools extension helps inspect the React component hierarchy, state and props
- **Installation :** Find “React Developer Tools” in the Chrome Web Store or Firefox Add-ons

2. Using React DevTools

- **Components Tab :** Shows the components tree and allows inspecting props and state
- **Profiler Tab :** Profiles render times to help identify performance bottlenecks

Exercise : Creating Your First React App with Vite

1. **Setup** : Follow the steps to install NodeJS and create a new React project with Vite
2. **Edit App.jsx :** Replace the default content with:

```
function App() {
    return (
        <div>
            <h1>Hello, React with Vite!</h1>
            <p>Welcome to your first React application using Vite.</p>
        </div>
    );
}

export default App;
```

3. **Run the App** : Use *npm run dev* command and view the result at *http://localhost:5173*
4. **Explore DevTools :** Open React Developer Tools in the browser to inspect the *App* component

Lesson 2 : JSX and Rendering Elements

2.1 What is JSX

- **Definition :** JSX, or Javascript XML, is a syntax extension for Javascript that allow you to write HTML-like code within Javascript. Its not required to use React, but it simplifies UI development and makes the code more readable
- Why JSX?
    - **Easier to understand :** JSX combines HTML-like syntax with Javascript, making it easier to visualize UI structure directly in the code
    - **Helps detect errors :** JSX allow the use of Javascript syntax and validation, so you can catch common errors (like typos) in your editor
    - **Compilers to Javascript :** Under the hood, JSX is transformed into Javascript function calls that create React example, for example

```
const element = <h1>Hello, world!</h1>;
```

gets compiled to

```
const element = React.createElement('h1', null, 'Hello, world!');
```

2.2 Embedding Expression in JSX

1. Using Javascript expression

- JXS allows you to embed Javascript expression within { } (curly braces). For example:

```
const name = 'Student';
const element = <h1>Hello, {name}!</h1>;
```

- You can perform calculation or call function directly in JSX

```
const user = { firstName: 'Jane', lastName: 'Doe' };
const element = <h1>Hello, {user.firstName} {user.lastName}!</h1>;
```

2. Conditional expression

- Use inline expression for simple condition within JSX. For example:

```
const isLoggedIn = true;
const element = <h1>{isLoggedIn ? 'Welcome back!' : 'Please log in.'}</h1>;
```

2.3 Rendering Elements in React

1. Rendering a React Element:

- React elements are the smallest building blocks of React applications. They describe what you want to see on the screen
- React element are immutable; once created, you cant change their children or attributes. You need to create a new element to update the UI

2. Rendering an element to the DOM

- The *ReactDOM.render* method is used to render a React element to a specified DOM node:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
const element = <h1>Hello, React!</h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(element);
```

3. Single Root DOM Node:

- React requires a single root DOM node. Commonly, the *id="root"* element in the HTML file servers as its root.

2.4 The importance of the Virtual DOM in Rendering

1. Understanding the Virtual DOM

- The virtual DOM is a lightweight representation of the real DOM. When state or props change, React compares the updated VIrtual DOM with the previous version and calculates the minimal set of changes required to update the real DOM
- **Efficient rendering** : React reconciliation algorithm updates only the necessary parts of the DOM, improving performance for complex UIs.

2.5 JSX Syntax and Rules

1. **JSX attributes in JSX**

- JSX syntax resembles HTML, but uses camelCases for attributes. For example:

```
const element = <div className="container"></div>;
```

2. **Javascript in JSX**

- **Variables :** Embed variables and expression within { } (curly braces)
- **Functions :** Call function within { } if they return a valid JSX-compatible value

2.6 Javascript in JSX

1. Using inline Styles:

- Inline styles in JSX use camelCase and must be an object. For example:

```
const divStyle = { color: 'blue', backgroundColor: 'lightgray' };
const element = <div style={divStyle}>Styled with inline CSS!</div>;
```

2. CSS Classes:

- The *classname* attribute is used in place of *class* for applying CSS classes:

```
const element = <h1 className="header">This has a CSS class!</h1>;
```

2.7 JSX Gotchas

1. Javascript Reserved Words

- Avoid using reserved words directly in JSX. For example, instead of *class,* use *className*

2. Returning Multiple Elements:

- React components can only return one root element. To return multiple elements, use fragment (<> </>) or wrap them in a container *div.*
- Example

```
function App() {
    return (
        <>
            <h1>Welcome!</h1>
            <p>This is a paragraph.</p>
        </>
    );
}
```

2.8 Exercise : Building Basic JSX Components

1. Create a Simple Component: Inside `App.jsx`, add a basic greeting:

```
function Greeting() {
    const name = 'Student';
    return <h1>Hello, {name}!</h1>;
}
export default Greeting;
```

2. Embed Expression

- Update the component to include the current date and time:

```
function Greeting() {
    const name = 'Student';
    const currentDate = new Date().toLocaleDateString();
    return (
        <div>
            <h1>Hello, {name}!</h1>
            <p>Today's date is: {currentDate}</p>
        </div>
    );
}
```

3. Applying Inline Styling

- Add inline styles to the greeting

```
function Greeting() {
    const name = 'Student';
    const currentDate = new Date().toLocaleDateString();
    const greetingStyle = {
        color: 'blue',
        backgroundColor: 'lightgray',
        padding: '10px',
        borderRadius: '5px'
    };
    return (
        <div style={greetingStyle}>
            <h1>Hello, {name}!</h1>
            <p>Today's date is: {currentDate}</p>
        </div>
    );
}
```

Lesson 3 : Components and Props

3.1 What are components

- Components are the building blocks of React applications, used to encapsulate UI elements and logic. They allow us to divide the UI into independent, reusable pieces
- The are two main types of components: **Functional Components** and **Class Components**
- Why do we use Components?
    - Reusability
    - Isolation
    - Scalability

3.2 Functional Components

1. Defining a Functional Component:
    - Functional components are JavaScript functions that return JSX. They are the most common component type, especially with the addition of React Hooks.

Example

```
function Welcome(props) {
    return <h1>Hello, {props.name}!</h1>;
}
```

Note: The component's name must start with an uppercase letter to be recognized as a React component.

3.3 Rendering a Functional Component:

To use a component, place it in JSX like an HTML tag. For example

```
function App() {
    return (
        <div>
            <Welcome name="Alice" />
            <Welcome name="Bob" />
        </div>
    );
}
```

3.4 Props in React

1. What Are Props?

- Props (short for “properties”) are a way to pass data from a parent component to a child component. They are read-only and cannot be modified by the child component.

2. Using Props in Functional Components:

Props are passed as an argument to functional components

```
function Welcome(props) {
    return <h1>Hello, {props.name}!</h1>;
}
```

Using Props in Class Components:

In class components, props are accessed using `this.props`:

```
class Welcome extends React.Component {
    render() {
        return <h1>Hello, {this.props.name}!</h1>;
    }
}
```