# 

# 

# Section 3: Introduction to React

### **Section 3: Introduction to React**

* **What is React?**
  + **Declarative UI**
  + **Components**
  + **JSX**
* **Creating a React App:**
  + **Setting up a React project**
  + **Basic component structure**
  + **Rendering elements to the DOM**
* **Example:**
  + **Create a simple React component to display a list of users (data fetched from the Python API).**

## **Section 3: Introduction to React**

### **What is React?**

React is a JavaScript library for building user interfaces. It's used to create single-page applications (SPAs) and mobile apps. React follows a component-based architecture, making it easier to build complex UIs.

### **Declarative UI**

Instead of imperatively telling the UI how to change, React uses a declarative approach. You describe the desired state of the UI, and React efficiently updates the DOM to match.

### **Components**

Components are reusable building blocks of React applications. They encapsulate UI elements and their logic. Components can be nested to create complex structures.

### **JSX**

JSX is a syntax extension for JavaScript that resembles HTML. It makes writing React components more intuitive.

### **Creating a React App:**

**Software to download:**

* **Node.js:** Provides a runtime environment for JavaScript.
* **npm (or yarn):** Package manager for Node.js.

**Setting up a React project:** Use Create React App to quickly set up a new React project:

**npx create-react-app my-app**

**cd my-app**

**npm start**

This will create a new React project, install dependencies, and start the development server.

**Basic component structure:** A React component is typically defined as a JavaScript function or class. It returns JSX that describes the UI.

**Rendering elements to the DOM:** React uses a virtual DOM to efficiently update the real DOM. When the component's state changes, React re-renders the component and updates the DOM accordingly.

### **Example: Creating a simple React component to display a list of users (data fetched from the Python API)**

**import React, { useState, useEffect } from 'react';**

**function UserList() {**

**const [users, setUsers] = useState([]);**

**useEffect(() => {**

**const fetchUsers = async () => {**

**try {**

**const response = await fetch('http://localhost:5000/users'); // Replace with your API endpoint**

**const data = await response.json();**

**setUsers(data);**

**} catch (error) {**

**console.error('Error fetching users:', error);**

**}**

**};**

**fetchUsers();**

**}, []);**

**return (**

**<div>**

**<h2>Users</h2>**

**<ul>**

**{users.map(user => (**

**<li key={user.id}>{user.name}</li>**

**))}**

**</ul>**

**</div>**

**);**

**}**

**export default UserList;**

**Explanation:**

1. Import necessary modules: React, useState, useEffect for state management and lifecycle methods
2. **Fetch function:** The fetch function is used to make the API request.
3. **Async/await:** We use async/await to handle the asynchronous nature of the fetch request.
4. Create a functional component UserList.
5. Use useState to manage the users state.
6. Use useEffect to fetch user data from the API when the component mounts.
7. Render a list of users using JSX and map over the users array.

**Additional tips:**

* Use a code editor or IDE with React support for syntax highlighting, autocompletion, and debugging.
* Explore React's component lifecycle methods to understand how components behave.
* Learn about props and state to manage data flow in your components.
* Practice building different types of components to improve your skills.

By following these steps and exploring further, you'll gain a solid foundation in React development.

**Note:** This is a basic example. Real-world React applications often involve more complex components, state management, and routing.