##

React JS By Prasad

##

What is a Component?

In React, a component is a small, self-contained piece of user interface. It can be as simple as a single button or as complex as a full page. Components are like building blocks that you can combine to create larger, more sophisticated user interfaces.

Why Use Components?

Using components offers several benefits:

Reusability: You can create a component once and reuse it throughout your application. This reduces code duplication and makes your codebase more manageable.

Modularity: Components promote a modular approach to development. You can work on individual components without affecting others, which makes collaboration easier.

Readability: Components encapsulate specific functionality or UI, making your code easier to understand and maintain.

Abstraction: Components allow you to abstract complex functionality into simple, isolated pieces.

Creating a Component:

Creating a component involves writing a function or class that returns JSX, which describes how the UI should look. There are two main ways to create components:

1. Functional Components:

These are functions that return JSX. They are simple, easy to write, and suitable for many scenarios.

```
import React from 'react';

function ButtonComponent() {
  return <button>Click me</button>;
}

export default ButtonComponent;
```

2. Class Components:

These are ES6 classes that extend React.Component. They offer more features, like state and lifecycle methods, but tend to be more verbose.

```
import React, { Component } from 'react';

class ButtonComponentClass extends Component {
  render() {
    return <button>Click me</button>;
  }
}

export default ButtonComponentClass;
```

Using Components:

To use a component, you simply import it and use it within other components or the main application file

```
import React from 'react';
import ButtonComponent from './ButtonComponent';
function App() {
  return (
    <div>
      <h1>Hello, React!</h1>
      <ButtonComponent />
    </div>
  );
}
export default App;
```

Hierarchy of Components:

In React, components can be nested within each other to form a hierarchy. Parent components can contain child components, which can in turn contain other child components. This hierarchy is what makes React applications flexible and modular.

Summary:

At a beginner level, components in React are like building blocks that encapsulate UI functionality. You can create components using either functional components or class components. They offer reusability, modularity, and abstraction, making your codebase more organized and maintainable. As you progress, you'll learn how to pass data between components (using props) and manage component-specific state (using state and hooks).

MovieList.Js (Using Function Component)

```
import React from 'react';
const movies = [
  {
    title: 'Movie 1',
    genre: 'Action',
  },
  {
    title: 'Movie 2',
    genre: 'Comedy',
  },
  {
    title: 'Movie 3',
```

```
    genre: 'Drama',
  },
  // Add more movies here
];
function MoviesList() {
  return (
    <div className="movies-list">
      <h2>List of Movies</h2>
      <ul>
        {movies.map((movie, index) => (
          <li key={index}>
            <strong>{movie.title}</strong> - {movie.genre}
          </li>
        ))}
      </ul>
    </div>
  );
}
export default MoviesList;
```

MovieList.Js (Using Class Component):

```
import React, { Component } from 'react';
const movies = [
  {
    title: 'Movie 1',
    genre: 'Action',
  },
  {
    title: 'Movie 2',
    genre: 'Comedy',
  },
  {
    title: 'Movie 3',
    genre: 'Drama',
  },
```

```
  // Add more movies here
];
class MoviesListClass extends Component {
  render() {
    return (
      <div className="movies-list">
        <h2>List of Movies (Class Component)</h2>
        <ul>
          {movies.map((movie, index) => (
            <li key={index}>
              <strong>{movie.title}</strong> - {movie.genre}
            </li>
          ))}
        </ul>
      </div>
    );
  }
}
export default MoviesListClass;
```

Steps To Write Multiple Components:

Step 1: Set Up Your Environment

If you haven't already set up your environment, follow the steps mentioned earlier to create a new React app using Create React App.

Step 2: Create Components

In the src directory, create two files named Navbar.js and Footer.js to define the components:

Navbar.js

```
import React from 'react';
function Navbar() {
  return (
    <nav className="navbar">
      <div className="logo">My Website</div>
      <ul className="nav-links">
        <li>Home</li>
        <li>About</li>
```

```
        <li>Contact</li>
      </ul>
    </nav>
  );
}
export default Navbar;
```

Footer.js:

```
import React from 'react';
function Footer() {
  return (
    <footer className="footer">
      <p>&copy; 2023 My Website. All rights reserved.</p>
    </footer>
  );
}
export default Footer;
```

Step 3: Use the Components

Open the src/App.js file and replace its content with the following code:

```
import React from 'react';
import './App.css';
import Navbar from './Navbar';
import Footer from './Footer';
function App() {
  return (
    <div className="App">
      <Navbar />
      <main className="content">
        <h1>Welcome to My Website</h1>
        <p>This is a simple webpage using only components.</p>
      </main>
      <Footer />
    </div>
  );
}
```

export default App;

Step 4: Run the Application

In the terminal, make sure you're in the simple-components-app directory, and then run the following command to start your React app

npm start

Your app will display a basic webpage layout with a navbar, main content, and footer.

This example demonstrates how you can create a simple webpage structure using only functional components without using props, state, or hooks. The Navbar and Footer components are reusable building blocks that you can easily integrate into different pages of your website.