Day 2: Components & Props in React

Today, you'll learn about **React components**, **props**, and how to pass data between components.

---

1 What Are Components?

A **component** in React is a **reusable UI element**. Everything in React is made up of components.

Types of Components

1. **Functional Components** (Modern, Recommended)
   - A simple JavaScript function that returns JSX.
   - Example:

```
function Greeting() {
  return <h1>Hello, React!</h1>;
}
export default Greeting;
```

2. **Class Components** (Older, Less Used)
   - Uses `class` syntax and `render()` method.
   - Example:

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, React!</h1>;
  }
}
export default Greeting;
```

✅ **We will use Functional Components because they are simpler and faster!**

2️⃣ Creating Functional Components

📌 **Inside `src/components/`, create a new file `UserCard.jsx`**

```jsx
function UserCard() {
  return (
    <div>
      <h2>John Doe</h2>
      <p>Web Developer</p>
    </div>
  );
}

export default UserCard;
```

Now, import and use it in `App.jsx`:

```jsx
import UserCard from "./components/UserCard";

function App() {
  return (
    <div>
      <h1>My React App</h1>
      <UserCard />
    </div>
  );
}

export default App;
```

✅ **You just created and used your first component!** 🎉

3️⃣ What Are Props?

Props (short for "properties") allow passing data from a parent component to a child component.

Example Without Props

Currently, `UserCard.jsx` has hardcoded data. What if we want multiple users?

4️⃣ Passing Props to Components

Modify `UserCard.jsx` to accept **props**:

```jsx
function UserCard(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>{props.role}</p>
    </div>
  );
}

export default UserCard;
```

Now, pass **props** in `App.jsx`:

```jsx
import UserCard from "./components/UserCard";

function App() {
  return (
    <div>
      <h1>My React App</h1>
      <UserCard name="John Doe" role="Web Developer" />
      <UserCard name="Jane Smith" role="UI/UX Designer" />
    </div>
  );
}

export default App;
```

✅ **Now, you can reuse `UserCard` with different data!**

5️⃣ Using Object Destructuring in Props

Instead of `props.name` and `props.role`, we can **destructure** props like this:

```jsx
function UserCard({ name, role }) {
  return (
    <div>
      <h2>{name}</h2>
      <p>{role}</p>
    </div>
  );
}

export default UserCard;
```

✅ **Cleaner and more readable!**

6️⃣ Default Props

If no value is passed, we can set a **default value** using `defaultProps`:

```
function UserCard({ name = "Guest", role = "Unknown" }) {
  return (
    <div>
      <h2>{name}</h2>
      <p>{role}</p>
    </div>
  );
}

export default UserCard;
```

Now, if `name` or `role` is missing, it will show `"Guest"` and `"Unknown"`.

7️⃣ Props with Arrays & Objects

We can also pass **arrays or objects** as props.

📌 **Modify `App.jsx`:**

```
const userList = [
  { name: "Alice", role: "Frontend Developer" },
  { name: "Bob", role: "Backend Developer" },
];

function App() {
  return (
    <div>
      <h1>My React App</h1>
      {userList.map((user, index) => (
        <UserCard key={index} name={user.name} role={user.role} />
      ))}
    </div>
  );
}

export default App;
```

✅ **Now, `UserCard` dynamically renders multiple users!**

📌 Summary of Day 2

- ✔ Learned about Functional & Class Components
- ✔ Created reusable components
- ✔ Used **props** to pass data
- ✔ Implemented **default props**
- ✔ Rendered a **list of components dynamically**

💡 Next Step: Day 3 - State & Event Handling