DAY 2 – React Components, Props, Modular Structure & Mapping Arrays (FULL NOTES)

---

1. What Are Components? (Complete Explanation)

Simple Definition

A **component** is a reusable and independent piece of UI. Think of them like **LEGO blocks**.

Each block has a purpose and can be used multiple times.

Real-Life Analogy

- A button in a website appears in many pages → made once, reused everywhere.
- A card style (product card, student card, flashcard) → same component, but different data.

---

2. Why Components?

Components help with:

✔ Reusability ✔ Cleaner code ✔ Less duplication ✔ Better readability ✔ Faster development ✔ Easy maintenance ✔ Easy testing

---

3. Types of Components

1Functional Component (Most Common & Modern)

```
function Header() {
  return <h1>Hello!</h1>;
}
```

Arrow Function Component

```
const Header = () => {
  return <h1>Hello!</h1>;
};
```

Important Rule

Component names must start with **Capital Letter**.

✓ Correct → `Header` ❌ Wrong → `header`

React uses this to differentiate between:

- HTML tags → `<div>`
- React components → `<Header />`

---

4. How to Use a Component (Rendering)

```
function App() {
  return (
    <div>
      <Header />
      <Header />
      <Header />
    </div>
  );
}
```

Here `Header` is reused 3 times → React renders it 3 times.

---

5. Component File Structure (Best Practice)

❗ Instead of writing all components in App.jsx,

place each component in its own file.

Good Structure:

```
src/
  components/
    Header.jsx
    Card.jsx
    Flashcard.jsx
  App.jsx
  index.jsx
```

Why?

✓ Clean project ✓ Easy to find components ✓ Industry standard practice

---

6. What Are Props? (Very Important)

Full Definition

Props (short for “properties”) are **inputs** passed from a **parent component** to a **child component**.

Props allow components to be:

- Reusable
- Dynamic
- Configurable

Simple Example

Parent:

```
<Welcome name="Munees" />
```

Child:

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

More modern (destructuring):

```
function Welcome({ name }) {
  return <h1>Hello {name}</h1>;
}
```

---

7. Important Rules About Props

✓ Props are **read-only**

You CANNOT modify props inside a child component.

❌ Wrong:

```
props.name = "New Name";
```

✓ Correct: Use state if you want to modify values inside component.

---

8. Passing Multiple Props

Parent:

```
<Card
  title="React Basics"
  description="Learn components and props"
  author="Munees"
  rating={5}
/>
```

Child:

```
function Card({ title, description, author, rating }) {
  return (
    <div className="card">
      <h3>{title}</h3>
      <p>{description}</p>
      <p>By: {author}</p>
      <p>Rating: {rating}</p>
    </div>
  );
}
```

---

9. Default Props (Optional but Useful)

```
function Button({ label = "Click Me" }) {
  return <button>{label}</button>;
}
```

Even if user doesn't pass `label`, component uses `"Click Me"`.

---

10. Children Prop (Very Important Concept)

Children = Content inside component.

Parent:

```
<Card>
  <p>This is inside the card.</p>
</Card>
```

Child:

```
function Card({ children }) {
  return <div className="card">{children}</div>;
}
```

`children` makes components extremely flexible.

---

11. Mapping Arrays (EXTREMELY IMPORTANT)

Mapping is used for rendering lists:

- Flashcards
- Tasks
- Students
- Products
- Comments
- API results

Basic Example

```
const fruits = ["Apple", "Mango", "Orange"];

fruits.map(fruit => <p>{fruit}</p>);
```

Using index as key

```
fruits.map((fruit, index) => (
  <p key={index}>{fruit}</p>
));
```

---

12. Why Key Is Required?

React needs a unique `key` to:

- Track items
- Update only changed items
- Improve performance

Without a `key`, React cannot identify items properly.

---

13. Mapping Array of Objects

Example:

```
const users = [
  { id: 1, name: "Munees", age: 22 },
  { id: 2, name: "Arun", age: 21 }
];
```

Mapping:

```
users.map(user => (
  <div key={user.id}>
    <h3>{user.name}</h3>
    <p>Age: {user.age}</p>
  </div>
));
```

---

14. Real Project Example: Flashcard App

Step 1: Create sample data

```
const flashcards = [
  {
    id: 1,
    question: "What is React?",
    answer: "A JavaScript library for UI"
  },
  {
    id: 2,
    question: "What is JSX?",
    answer: "HTML-like syntax in JS"
  }
];
```

Step 2: Map the data and render Flashcard component

```
flashcards.map(card => (
  <Flashcard
    key={card.id}
    question={card.question}
    answer={card.answer}
  />
));
```

Step 3: Flashcard Component

```
function Flashcard({ question, answer }) {
  return (
    <div className="flashcard">
      <h3>{question}</h3>
      <p>{answer}</p>
    </div>
  );
}
```

---

15. Interactivity: Flip Card Logic (useState Intro)

```
function Flashcard({ question, answer }) {
  const [showAnswer, setShowAnswer] = useState(false);

  return (
    <div
      className="flashcard"
      onClick={() => setShowAnswer(!showAnswer)}
    >
      {showAnswer ? answer : question}
    </div>
  );
}
```

Students learn:

- Component state
- Click events
- Conditional rendering

---

16. Component Communication

Parent → Child

Uses **props**

Child → Parent

Uses **callback functions**

Example (simple):

```
function Parent() {
  function handleButtonClick() {
    alert("Child clicked!");
  }

  return <Child onClick={handleButtonClick} />;
}
```

Child:

```
function Child({ onClick }) {
  return <button onClick={onClick}>Click Me</button>;
}
```

---

17. Inline Styling in Components

```
<div style={{ backgroundColor: "yellow", padding: "10px" }}>
  Hello!
</div>
```

---

18. Reusable Component Patterns

Button example:

```
<Button text="Save" color="green" />
<Button text="Delete" color="red" />
<Button text="Login" color="blue" />
```

Child:

```
function Button({ text, color }) {
  return <button style={{ background: color }}>{text}</button>;
}
```

One component → 3 buttons with different purposes.

---

19. Best Practices for Components (Important)

✓ One component per file

✓ Use PascalCase for component names

✓ Use camelCase for props

✓ Keep components small

✓ Avoid deeply nested components

✓ Reuse components instead of rewriting

✓ Keep props meaningful

✓ Map only arrays

✓ Use keys correctly
