Module 2 — React + Vite: Creating Your First Component & Understanding Props (80 minutes)

> Audience: Kids 13–14 (beginners, continuing from Module 1). We keep building inside the same `react-app` project. Today we create code in `src/day2/`.

Learning Objectives

By the end of this module, students will: 1. Recall what components and JSX are from Module 1. 2. Create new **functional components** in their own files. 3. Understand **props**: what they are, how to pass them, and how they differ from state. 4. Reuse the same component with different props (like LEGO blocks with different stickers). 5. Build a **User component** hands-on and assign homework for a **Profile component**.

80-Minute Agenda

- **0–10 min** — Recap from Day 1 (quiz style)
- **10–25 min** — Creating a new functional component (User component)
- **25–40 min** — Deep dive into props: passing, receiving, destructuring
- **40–60 min** — Hands-on: Extend User component with props (name, age, hobby)
- **60–75 min** — Practice & mini-project
- **75–80 min** — Recap, Q&A, Homework assignment

Recap (0–10 min)

Quick questions to students: - What is JSX? (HTML in JS, compiled by React) - Why do we use `className` instead of `class`? - What's the difference between **props** and **state**? - What do fragments (`<>...</>`) allow us to do?

Creating a New Component (10–25 min)

We'll create a folder for today's code: `src/day2/`

Step 1 — User.jsx

```
// src/day2/User.jsx
// A simple functional component showing user info
```

```jsx
function User(props) {
  // props is an object holding the values passed in
  return (
    <div className="user-card">
      <h2>{props.name}</h2> {/* Display name from props */}
      <p>Age: {props.age}</p>
      <p>Hobby: {props.hobby}</p>
    </div>
  );
}

export default User;
```

Step 2 — Use it in App.jsx

Update `src/App.jsx`:

```jsx
import User from './day2/User.jsx';

function App() {
  return (
    <>
      <header className="app-header">
        <h1>React + Vite — Day 2: Props</h1>
      </header>

      <main className="container">
        {/* Passing props to User component */}
        <User name="Arta" age={14} hobby="painting" />
        <User name="Blerim" age={13} hobby="football" />
        <User name="Elira" age={14} hobby="chess" />
      </main>
    </>
  );
}

export default App;
```

> **Instructor tip:** Emphasize that each `<User />` is the same component definition, but the **props make it look different**.

Deep Dive: Props (25–40 min)

What are Props?

- **Props = inputs to components**.
- They're like function parameters. The parent component decides what values to pass.

Plain JS analogy:

```
function greet(name) {
  console.log('Hello, ' + name);
}

// Call function with different arguments
greet('Arta');
greet('Blerim');
```

React analogy:

```
<User name="Arta" />
<User name="Blerim" />
```

Accessing Props

- `props` is always an **object**.
- You can destructure for cleaner code:

```
function User({ name, age, hobby }) {
  return (
    <div className="user-card">
      <h2>{name}</h2>
      <p>Age: {age}</p>
      <p>Hobby: {hobby}</p>
    </div>
  );
}
```

Key Points

- Props are **read-only**. The child can't change them.
- If you want changing data, that's where **state** comes in (later).

Hands-On Practice (40–60 min)

Students extend the User component with more props.

Challenge: Add a favorite color

```
<User name="Driton" age={13} hobby="gaming" color="blue" />
```

Update component to:

```
function User({ name, age, hobby, color }) {
  return (
    <div className="user-card">
      <h2 style={{ color: color }}>{name}</h2>
      <p>Age: {age}</p>
      <p>Hobby: {hobby}</p>
      <p>Favorite color: {color}</p>
    </div>
  );
}
```

- Note inline style syntax: `style={{ color: color }}` (double braces). - Props make components flexible and reusable.

Practice & Mini-Project (60–75 min)

Create **UserList.jsx** in `src/day2/`:

```
// src/day2/UserList.jsx
import User from './User.jsx';

function UserList() {
  return (
    <div>
      <h2>Class Members</h2>
      <User name="Amina" age={13} hobby="drawing" color="purple" />
      <User name="Blerim" age={14} hobby="football" color="green" />
      <User name="Elira" age={14} hobby="piano" color="red" />
    </div>
  );
}
```

```
export default UserList;
```

Use inside `App.jsx`:

```
import UserList from './day2/UserList.jsx';

// Inside <main>
<UserList />
```

Recap & Homework (75–80 min)

Recap Talking Points

- Components = functions returning JSX.
- Props = inputs (like arguments).
- Props make components **reusable**.
- Props are **read-only**.

Homework

- Create a `Profile.jsx` component in `src/day2/`.
- Props: `name`, `age`, `city`, `bio`.
- Render in `App.jsx` at least 2 times with different data.
- Bonus: Add a self-closing `<img />` tag to show profile pictures.

Example usage:

```
<Profile
  name="Amina"
  age={14}
  city="Tetovo"
  bio="Loves painting and reading"
  imgUrl="/path/to/amina.jpg"
/>
```

Instructor Script (Suggested)

- "Props are like **arguments for functions**. Each call of `<User />` gets different inputs."
- "We can destructure props to make code cleaner."
- "Props are **immutable** in the child. Only the parent can decide what to send."

- “Reusable = same component definition, multiple looks.”

Mini-Quiz (for class discussion)

1. If we don’t pass `age` as a prop, what will happen?
2. Why can’t we change a prop value inside a component?
3. How are props similar to function parameters?