React + Vite Course (Kids 13–14)

Module 7 — Context API & Sharing State (80 minutes)

> **Audience:** Kids 13–14
> This module builds directly on Modules 1–6.
> All code stays in the same project.
> Today's work goes into `src/day7/`.

Why This Module Is VERY Important

Up until now, students have learned:

- Components
- Props
- State (`useState`)
- Side effects (`useEffect`)
- Styling

This module solves a **real React problem** that appears in **every real app**:

> ❌Passing data through too many components

This problem is called **prop drilling**.

Learning Objectives

By the end of this module, students will:

1. Understand what **prop drilling** is
2. Know **why prop drilling is bad**
3. Understand what **Context API** is
4. Create a **Context**
5. Provide and consume context data
6. Share state between many components
7. Understand **global state** (simple version)
8. See full homework solutions

80-Minute Agenda

- **0–10 min** — Recap (Modules 1–6)
- **10–25 min** — Prop drilling explained (very important)
- **25–40 min** — What is Context API?
- **40–60 min** — Hands-on: create & use context
- **60–70 min** — Updating context data
- **70–80 min** — Homework + recap

Recap from Module 6 (0–10 min)

Ask students:

- What is state?
- What are props?
- How do components communicate?
- What happens when state changes?

> **Reminder:** Props only go **from parent to child**.

The Big Problem: Prop Drilling (10–25 min)

What Is Prop Drilling?

Prop drilling means:

> Passing props through components that **do not need them**, just to reach a deeper component.

Visual Example

```
App
 └── Parent
      └── Child
           └── GrandChild
```

If `App` has data and `GrandChild` needs it:

```
<App user={user} />
```

Then:

```
<Parent user={user} />
<Child user={user} />
<GrandChild user={user} />
```

❌Parent and Child don't use `user`, but must pass it.

Real-Life Analogy (VERY IMPORTANT)

Imagine:

- Your teacher tells a message to one student
- That student tells another
- That student tells another

By the time it reaches the last student — it's slow and messy 💬

The Solution: Context API (25–40 min)

What Is Context API?

Context API allows us to:

- Store data in **one place**
- Let **any component** read it
- Avoid prop drilling

Teaching line: *"Context is like a global backpack everyone can open."*

Creating Our First Context (40–50 min)

Step 1: Create Context File

Create:

```
src/day7/UserContext.jsx
```

```jsx
import { createContext } from 'react';

const UserContext = createContext();

export default UserContext;
```

Explanation (Slow & Clear)

- `createContext()` creates a **container** for data
- At this point, it has **no value yet**

Providing Context (50–60 min)

Step 2: Wrap Components

Update `App.jsx`:

```
import UserContext from './day7/UserContext';

function App() {
  const user = { name: 'Alex', age: 14 };

  return (
    <UserContext.Provider value={user}>
      <Dashboard />
    </UserContext.Provider>
  );
}

export default App;
```

Explanation

- `Provider` makes data available
- `value` is the data we share
- All children can access it

Consuming Context (60–70 min)

Step 3: Use Context in Component

Create `src/day7/Dashboard.jsx`:

```
import { useContext } from 'react';
import UserContext from './UserContext';

function Dashboard() {
  const user = useContext(UserContext);
```

```
  return (
    <div>
      <h2>Dashboard</h2>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
    </div>
  );
}

export default Dashboard;
```

Explanation

- `useContext` reads shared data
- No props needed
- Works anywhere inside Provider

Updating Context Data (70–75 min)

Context can also share **functions**.

Example:

```
const userData = {
  name: 'Alex',
  age: 14,
  login: () => console.log('Logged in')
};
```

Components can call shared functions.

Homework (Module 7)

Task 1

Create a **ThemeContext** with:

- theme: `light` or `dark`

Task 2

Create a component that:

- Reads the theme
- Displays text depending on theme

Homework — FULL SOLUTION

ThemeContext.jsx

```
import { createContext } from 'react';

const ThemeContext = createContext('light');

export default ThemeContext;
```

App.jsx

```
import ThemeContext from './day7/ThemeContext';
import ThemeBox from './day7/ThemeBox';

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemeBox />
    </ThemeContext.Provider>
  );
}

export default App;
```

ThemeBox.jsx

```
import { useContext } from 'react';
import ThemeContext from './ThemeContext';

function ThemeBox() {
  const theme = useContext(ThemeContext);

  return (
    <div>
      <h2>Current Theme</h2>
```

```
      <p>{theme === 'dark' ? 'Dark Mode' : 'Light Mode'}</p>
    </div>
  );
}

export default ThemeBox;
```

Final Recap (VERY IMPORTANT)

- Prop drilling is bad
- Context shares data globally
- Provider gives data
- `useContext` reads data
- Used in every real app

Instructor Teaching Lines

- “Props are local, context is global.”
- “Context removes unnecessary props.”
- “Every big React app uses context.”

Next Module Preview

Module 8 — React Router & Navigation