Why Context API Exists

Normally in React:

```
Parent → Child → Grandchild → GreatGrandchild
```

If you want a value (like user info or theme) in a deep child, you'd have to **pass it through every component in between**. This is called **props drilling**—messy and repetitive.

Context API solves this. You can **store the data at a higher level** and let any child access it directly.

Let's do a **very simple React + Vite + Context API example** that **passes three values** using Context—easy to understand and beginner-friendly.

We'll pass **username, email, and age** from context to a component.

Step 1: Setup Vite React App

```
npm create vite@latest simple-context
cd simple-context
npm install
npm run dev
```

Let's extend your **simple UserContext example** so that **multiple children components** can access the same values. This shows the real power of Context—**no props drilling**.

Step 1: User Context (same as before)

UserContext.jsx:

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

// 1. Create Context
export const UserContext = createContext();

// 2. Create Provider
export const UserProvider = ({ children }) => {
  const user = {
    username: "Alice",
    email: "alice@example.com",
    age: 25,
  };

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

Step 2: Wrap App with Provider

main.jsx:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import { UserProvider } from "./UserContext.jsx";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <UserProvider>
      <App />
    </UserProvider>
  </React.StrictMode>
);
```

Step 3: Create Multiple Children Components

App.jsx

```
import UserInfo from "./UserInfo.jsx";
import UserContact from "./UserContact.jsx";

function App() {
```

```
  return (
    <div style={{ padding: "20px" }}>
      <h1>Context Example with Multiple Children</h1>
      <UserInfo />
      <UserContact />
    </div>
  );
}

export default App;
```

UserInfo.jsx (Child 1)

```
import { useContext } from "react";
import { UserContext } from "./UserContext.jsx";

function UserInfo() {
  const { username, age } = useContext(UserContext);

  return (
    <div>
      <h2>User Info</h2>
      <p>Username: {username}</p>
      <p>Age: {age}</p>
    </div>
  );
}

export default UserInfo;
```

UserContact.jsx (Child 2)

```
import { useContext } from "react";
import { UserContext } from "./UserContext.jsx";

function UserContact() {
  const { email } = useContext(UserContext);

  return (
    <div>
      <h2>Contact Info</h2>
      <p>Email: {email}</p>
    </div>
  );
}

export default UserContact;
```

Step 4: Run the App

```
npm run dev
```

You will see both children **independently accessing the same context values**:

```
Context Example with Multiple Children

User Info
Username: Alice
Age: 25

Contact Info
Email: alice@example.com
```

you can see how **Context API shares state and functions across multiple components**.

Step 1: User Context with user + function

`UserContext.jsx`:

```
import { createContext, useState } from "react";

// 1. Create Context
export const UserContext = createContext();

// 2. Create Provider
export const UserProvider = ({ children }) => {
  const [username, setUsername] = useState("Alice");

  // Function to change username
  const changeUsername = () => {
    setUsername(username === "Alice" ? "Bob" : "Alice");
  };

  return (
    <UserContext.Provider value={{ username, changeUsername }}>
      {children}
    </UserContext.Provider>
  );
};
```

 This is the **same as before**: `username` state + `changeUsername` function.

Step 2: Wrap App with Provider

`main.jsx`:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import { UserProvider } from "./UserContext.jsx";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <UserProvider>
      <App />
    </UserProvider>
  </React.StrictMode>
);
```

Step 3: Create Three Children Components

Child1.jsx – display username

```
import { useContext } from "react";
import { UserContext } from "./UserContext.jsx";

function Child1() {
  const { username } = useContext(UserContext);

  return (
    <div>
      <h2>Child 1</h2>
      <p>Username: {username}</p>
    </div>
  );
}

export default Child1;
```

Child2.jsx – button to change username

```
import { useContext } from "react";
import { UserContext } from "./UserContext.jsx";

function Child2() {
  const { changeUsername } = useContext(UserContext);

  return (
    <div>
      <h2>Child 2</h2>
      <button onClick={changeUsername}>Change Username</button>
    </div>
  );
}

export default Child2;
```

Child3.jsx – welcome message using username

```
import { useContext } from "react";
import { UserContext } from "./UserContext.jsx";

function Child3() {
  const { username } = useContext(UserContext);

  return (
    <div>
      <h2>Child 3</h2>
      <p>Welcome, {username}!</p>
    </div>
```

```
  );
}

export default Child3;
```

Step 4: Use All Children in App

App.jsx:

```
import Child1 from "./Child1.jsx";
import Child2 from "./Child2.jsx";
import Child3 from "./Child3.jsx";

function App() {
  return (
    <div style={{ padding: "20px" }}>
      <h1>React Context API Example with Three Children</h1>
      <Child1 />
      <Child2 />
      <Child3 />
    </div>
  );
}

export default App;
```

□ Result

- **Child 1** shows the username.
- **Child 2** has a button to toggle the username.
- **Child 3** displays a welcome message using the same username.

Clicking the button **updates the username in all children instantly**, showing how **Context API shares state and functions across multiple components** without passing props.