Context API

Question 1: What is the Context API in React? How is it used to manage global state across multiple components?

Ans.

The Context API is a built-in feature of React that allows you to share data globally across the component tree without passing props manually at every level (a problem known as *prop drilling*).

It is mainly used for global state management such as:

- Authentication (user info, login status)
- Theme (dark/light mode)
- Language / localization
- App settings
- Shared data across many components

Why do we need Context API?

Problem: Prop Drilling

Passing data from a parent component to deeply nested child components can become messy and hard to maintain.

```
<App>
 <Header>
  <Navbar>
   <Profile user={user} />
  </Navbar>
 </Header>
```

```
</App>
```

Here, user is passed through multiple components even if they don’t use it.

Context API solves this problem by providing a global store accessible anywhere.

Core Parts of Context API

1. **createContext()** – creates a Context object
2. **Provider** – provides the data (global state)
3. **Consumer / useContext()** – consumes the data

How Context API Works (Step-by-Step) :

1. Create a Context

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

export const UserContext = createContext();
```

2. Provide Global State using Provider

```
        import React, { useState } from "react";
import { UserContext } from "./UserContext";

function App() {
  const [user, setUser] = useState("Uday");

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

export default App;
```

3. Consume Context in Any Component

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

function Dashboard() {
  const { user, setUser } = useContext(UserContext);
  return (
    <div>
      <h2>Welcome, {user}</h2>
      <button onClick={() => setUser("Parmar")}>
        Change Name
      </button>
    </div>
  );
}

export default Dashboard;
```

Context API with Class Components (Optional) :

```
<UserContext.Consumer>
  {(value) => <h1>{value.user}</h1>}
</UserContext.Consumer>
```

Managing Global State with Context + useReducer :-

For complex state logic, Context is often combined with useReducer.

```
const initialState = { theme: "light" };

function reducer(state, action) {
  switch (action.type) {
    case "TOGGLE_THEME":
      return { theme: state.theme === "light" ? "dark" : "light" };
    default:
      return state;
  }
}

const ThemeContext = createContext();

function ThemeProvider({ children }) {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <ThemeContext.Provider value={{ state, dispatch }}>
      {children}
    </ThemeContext.Provider>
  );
}
```

When Should You Use Context API?

Use Context when:

- Data is needed by many components
- Avoiding prop drilling
- App-level state (theme, auth, settings)

Avoid Context when:

- State is local to a single component
- High-frequency updates (may cause re-renders)

Context API vs Redux (Quick Comparison)

Context API	**Redux**
Built-in React	External library
Simple setup	More boilerplate
Best for small–medium apps	Best for large apps
No middleware by default	Powerful middleware

Question 2: Explain how createContext() and useContext() are used in React for sharing state.

Ans.

In React, createContext() and useContext() work together to let you share state (data) across multiple components without passing props at every level.

1. createContext() – Creating a Shared State Container

createContext() creates a Context object.
Think of it as a global container where data can be stored and shared.

Syntax

```
const MyContext = React.createContext();
```

You can also provide a **default value**: const MyContext = React.createContext("default value");

The context object gives you:

- MyContext.Provider
- MyContext.Consumer

2. Provider – Supplying the State

The **Provider** is used to make data available to all child components.

Example: Creating and Providing State

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


export const UserContext = createContext();
```

```
function App() {
  const [user, setUser] = useState("Uday");

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

export default App;
```

value contains the **state and functions** you want to share
Any component inside the Provider can access this data

3. useContext() – Consuming the Shared State

useContext() is a **hook** that allows a component to read and use context data.

Syntax :

```
const value = useContext(MyContext);
```

Example: Accessing Shared State :

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

function Dashboard() {
  const { user, setUser } = useContext(UserContext);

  return (
    <div>
      <h2>Hello, {user}</h2>
      <button onClick={() => setUser("Parmar")}>
        Change User
      </button>
    </div>
  );
}

export default Dashboard;
```

How createContext() and useContext() Work Together

1. **createContext()** creates a shared data store
2. **Provider** supplies the data
3. **useContext()** reads the data in any child component

Simple Flow Diagram :

createContext()

↓

Provider (value = global state)

↓

Child Components

↓

useContext() → access shared state

Key Points to Remember

- createContext() → creates the context
- Provider → shares the state
- useContext() → consumes the state
- Helps avoid **prop drilling**
- Ideal for **global or app-wide state**

Real-World Use Cases

- User authentication
- Theme (dark/light mode)

- Language settings
- Cart data in e-commerce apps
