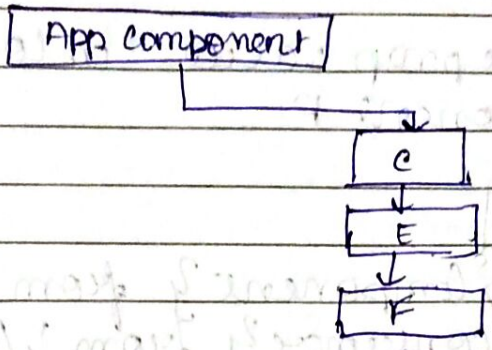
CONTEXT API

→ context provides a way to pass data through the component tree without having to pass props down manually at every level.



1. Create the context
2. Provide a context value
3. Consume the context value

App.js

```
import React, {Component} from 'react'
import './App.css'
import ComponentC from './components/ComponentC'
import {UserProvider} from './components/userContext'

class App extends Component {
  render() {
    return (
      <div className="App">
        <UserProvider value="Keshwar">
          <ComponentC />
        </UserProvider>
      </div>
    )
  }
}

export default App;
```

This value prop can be used in component C and also its nested components

we have component C
↓
component E
↓
component F

We can use the prop value in component C or component E or component F

ComponentF.js

```
import React, {Component} from 'react'
import {UserConsumer} from './usercontext'

class ComponentF extends Component {
  render() {
    return (
      <UserConsumer>
        {username => {
          return <div> Hello {username} </div>
        }}
      </UserConsumer>
    )
  }
}
export default ComponentF
```

UserContext.js

```
import React from 'react'

const UserContext = React.createContext()

const UserProvider = UserContext.Provider
const UserConsumer = UserContext.Consumer

export {UserProvider, UserConsumer}
```

Hello Vishuey

CONSUMING MULTIPLE CONTEXT

```
function context() {
  return (
    <ThemeContext.Consumer>
      {theme => (
        <UserContext.Consumer>
          {user => (
            <ProfilePage user={user} theme={theme} />
          )}
        </UserContext.Consumer>
      )}
    </ThemeContext.Consumer>
  );
}
```

* Context is primarily used when some data needs to be accessible by many components at different nesting levels. Apply it sparingly because it makes components reuse more difficult.

the class.contextType

```
class MyClass extends React.Component {

}
MyClass.contextType = UserContext
```

The contextType property on a class can be assigned a Context object created by React.createContext(). This lets you consume the nearest current value of that Context type using this.context. You can reference this in any of the lifecycle methods including the render function.

→ Context is designed to share data that can be considered "global" for a tree of React components, such as the current authenticated user, theme, or preferred language.

→ Context is primarily used when some data needs to be accessible by many components at different nesting levels.

→ Apply it sparingly because it makes component reuse more difficult.

→ If you only want to avoid passing some props through many levels, component composition is often a simpler solution than context.

API

~~create~~. React. createContext

const MyContext = React.createContext(defaultValue);

→ creates a Context object. When React renders a component that subscribes to this Context object it will read the current context value from the closest matching Provider above in the tree.

→ The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them.

Context.Provider

<MyContext.Provider value={/* some value */}>

→ Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.

Accepts a value prop to be passed to consuming components that are descendants of this Provider.

One Provider can be connected to many consumers.

Providers can be nested to override values deeper within the tree.

* All consumers that are descendants of a Provider will re-render whenever the Provider's value prop changes.

* Context.Consumer

```
<MyContext.Consumer>
    {value => /* render something based on context value */}
</MyContext.Consumer>
```

→ context.displayName

context object accepts a displayName string property. React DevTools uses this string to determine what to display for the context.

→ For example, the following component will appear as MyDisplayName in the DevTools.

eg:

```
const MyContext = React.createContext(/* ... */);
MyContext.displayName = 'MyDisplayName';

<MyContext.Provider> // "MyDisplayName.Provider" in DevTools
<MyContext.Consumer> // "MyDisplayName.Consumer" in DevTools
```

WITHOUT CONTEXT API.

App.js

```
import React, {Component} from "react";
import User from "./User";
export default class App extends Component{
    state = {
        name : "Rahul"
    };
    render(){
        return <User name = {this.state.name}/>;
    }
}
```

User.js

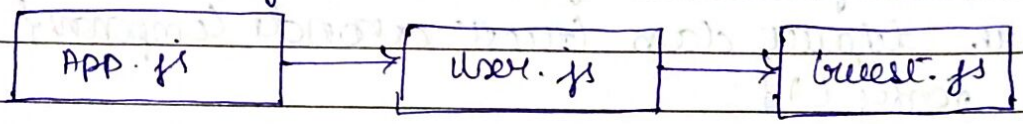
```
import React, {Component} from 'react';
import Guest from "./Guest";
export default class User extends Component{
    render(){
        return(
            <div>
                <h3> User </h3>
                <Guest name = {this.props.name}/>
            </div>
        )
    }
}
```

Guest.js

```
import React, { Component } from "react";

export default class Guest extends Component {
    render() {
        return (
            <div>
                <h3>Guest Component</h3>
                <h4>{this.props.name}</h4>
            </div>
        );
    }
}
```

So basically



intermediate component

~~X~~

we had to pass props even though it was not needed

USING CONTEXT API

```
import React, { Component } from "react";
import User from "./User";
export const MyContext = React.createContext();
export default class App extends Component {
    state = {
        name: "Rahul"
    };
    render() {
        return (
            <MyContext.Provider value={this.state.name}>
                <User />
            </MyContext.Provider>
        );
    }
}
```

this actually transfers all the data you need to these components wherever within this

User.js

```
import React, {Component} from "react";
import Guest from "./Guest";
export default class User extends Component{
    render(){
        return(
            <div>
                <h3> User Component </h3>
                <Guest />
            </div>
        );
    }
}
```

↳ no need to transfer props

Guest.js

```
import React, {Component} from "react";
import {MyContext} from "./App";
export default class Guest extends Component{
    render(){
        return(
            <div>
                <h3> Guest component </h3>
                <MyContext.Consumer>
                    {value => <h4> {value} </h4>}
                </MyContext.Consumer>
            </div>
        )
    }
}
```

You have to pass a function inside MyContext.Consumer instead of value you can use any name in consumer.

USING CONTEXT FOR MORE THAN ONE PROPS IN THE STATE

App.js

```
import React, {Component} from "react";
import User from "./User";
export const MyContext = React.createContext();
export default class App extends Component {
    state = {
        name: "Rahul",
        age: 10
        value: 10,
    };
    render() {
        return (
            <MyContext.Provider value={this.state}>
                <User />
            </MyContext.Provider>
        )
    }
}
```

User.js This will be the same as the previous example with a single property.

Guest.js

```
import React, {Component} from "react";
import {MyContext} from "./App";
export default class Guest extends Component {
    render() {
        return (
            <div> <h3> Guest Component </h3>
            <MyContext.Consumer>
                {data => (
                    <h4>
                        Name: {data.name}
                        Value: {data.value}
                    </h4>
                )}
            </MyContext.Consumer>
        </div>
        );
    }
}
```

IF WE WANT TO PASS ON A CONTEXT FUNCTION AS WELL

App.js

```
import React, {Component} from "react";
import User from "./User";
export const MyContext = React.createContext();
export default class App extends Component {
    state = {
        name: "Rahul",
        value: 10
    };
    handleClickContext = () => {
        this.setState({ value: this.state.value + 1});
    };
    render() {
        const contextValue = {
            data: this.state,
            handleClick: this.handleClickContext
        };
        return(
            <MyContext.Provider value={contextValue}>
                <User />
            </MyContext.Provider>
        );
    }
}
```

User.js will be the same as prev example.

Guest.js

```
import React, {Component} from "react";
import {MyContext} from "./App";
export default class Guest extends Component {

    render() {
        return(
```

```
<div>
  <h3>Cheese Component</h3>
  <MyContext.Consumer>
    {(data, handleClick) => (
      <div>
        <h4>Name: {data.name}
            value: {data.value}</h4>
        <button onclick={handleclick}>change
        value</button>
      </div>
    )}
  </MyContext.Consumer>
</div>
);
};
```