

Episode-11 | Data is The New Oil



Please make sure to follow along with the whole "**Namaste React**" series, starting from Episode-1 and continuing through each subsequent episode. The notes are designed to provide detailed explanations of each concept along with examples to ensure thorough understanding. Each episode builds upon the knowledge gained from the previous ones, so starting from the beginning will give you a comprehensive understanding of React development.



I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-11** first. Understanding what "**Akshay**" shares in the video will make these notes way easier to understand.

Q) What is prop drilling ?

In React, **prop drilling** refers to the process of **passing down props** (short for properties) through multiple layers of nested components. This happens when a piece of data needs to be transferred from a higher-level component to a deeply nested child component, and it must pass through several intermediary components in between.



Here's a simple example to illustrate prop drilling in React:

```

// Top-level component
function App() {
  const data = "Hello, prop drilling!";

  return (
    <div>
      <ParentComponent data={data} />
    </div>
  );
}

// Intermediate component
function ParentComponent({ data }) {
  return (
    <div>
      <ChildComponent data={data} />
    </div>
  );
}

// Deeply nested component that actually uses the data
function ChildComponent({ data }) {

  return <div>{data}</div>;
}

```

In this example, the `data` prop is passed from the `App` component through the `ParentComponent` down to the `ChildComponent`. The `ParentComponent` itself doesn't use the `data` prop; it merely passes it down. This process of passing data through intermediate components that don't use the data is what is referred to as `prop drilling`. Prop drilling can make the code harder to maintain, especially as the application grows and the number of components in the hierarchy increases. To mitigate this, developers often use other state management solutions, like the `React Context`

API , Redux , or other state management libraries , to avoid passing props through multiple layers of components. These alternatives provide a centralized way to manage and access state without the need for prop drilling.

Q) What is lifting the state up ?

Lifting state up in React refers to the practice of moving the state from a lower-level (child) component to a higher-level (parent or common ancestor) component in the component tree . This is done to share and manage state across multiple components.

When a child component needs access to certain data or needs to modify the data, instead of keeping that data and the corresponding state management solely within the child component, we move the state to a shared ancestor component. By doing so, the parent component becomes the source of truth for the state, and it can pass down the necessary data and functions as props to its child components.



Here's a simple example to illustrate lifting state up :

```
// Parent component
class ParentComponent extends React.Component {
```

```

constructor(props) {
  super(props);
  this.state = {
    count: 0,
  };
}


incrementCount = () => {
  this.setState((prevState) => ({
    count: prevState.count + 1,
  }));
};

render() {
  return (
    <div>
      <p>Count: {this.state.count}</p>
      <ChildComponent count={this.state.count} onIncrement=
{this.incrementCount} />
    </div>
  );
}
}

// Child component
function ChildComponent({ count, onIncrement }) {
  return (
    <div>
      <p>Child Count: {count}</p>
      <button onClick={onIncrement}>Increment</button>
    </div>
  );
}

```

In this example, the ParentComponent holds the state (count), and it passes both the state value (count) and a function (onIncrement) down to the ChildComponent as props. The child component can then display the count and trigger an increment when the button is clicked.

By lifting the state up to a common ancestor, you centralize the state management, making it easier to control and share state among components. This pattern is especially useful in larger React applications where multiple components need access to the same data or where the state needs to be synchronized across different parts of the application.  [Lifting State Up](#)

Q) What are Context Provider and Context Consumer ?

In React, the `Context` API provides a way to pass data through the component tree without having to pass props manually at every level .

The two main components associated with the `Context` API are the `Context Provider` and `Context Consumer` .

`Context Provider` ? The `Context Provider` is a component that allows its children to subscribe to a context's changes . It accepts a value prop, which is the data that will be shared with the components that are descendants of this provider. The Provider component is created using `React.createContext()` and then rendered as part of the component tree. It establishes the context and provides the data to its descendants.



Here's an example:

```
// Creating a context
const MyContext = React.createContext();
```

```
// Parent component serving as the provider
class MyProvider extends React.Component {
  state = {
    data: "Hello from Context!",
  };

  render() {
    return (
      <MyContext.Provider value={this.state.data}>
        {this.props.children}
      </MyContext.Provider>
    );
  }
}
```

Context Consumer 📖 The Context Consumer is a component that subscribes to the changes in the context provided by its nearest Context Provider ancestor. It allows components to access the context data without the need for prop drilling. The Consumer component is used within the JSX of a component to consume the context data. It takes a function as its child, and that function receives the current context value as an argument. Here's an example:

```
// Child component consuming the context
class MyConsumerComponent extends React.Component {
  render() {
    return (
      <MyContext.Consumer>
        {(contextData) => (
          <p>{contextData}</p>
        )}
      </MyContext.Consumer>
    );
  }
}
```

By using the Context Provider and Context Consumer, you can avoid prop drilling and make it easier to share global or shared state across different parts of your React application. This is particularly useful when passing data to deeply nested components without explicitly passing the data through each intermediate component.

Q) If we don't pass a value to the provider does it take the default value ?

Yes, If we don't pass a value to the Provider in React's Context API, it does use the default value specified when creating the context

using `React.createContext(defaultValue)` .



Here's the corrected explanation:

```
// Creating a context with a default value
const MyContext = React.createContext("Default Value");

// Parent component serving as the provider without providing
a value
class MyProvider extends React.Component {
  render() {
    return (
      <MyContext.Provider>
        {this.props.children}
      </MyContext.Provider>
    );
  }
}
```

In this example, if we don't provide a value to the `MyContext.Provider`, it will use the

default value ("Default Value" in this case) specified during the creation of the context. Any component that consumes this context using `MyContext.Consumer` will receive the default value if there is no Provider higher up the tree providing a different value.

Self-Notes-

Higher order component-

Higher order component is the function that takes a component inside and returns a new component. It just the normal JavaScript function.

Why do we use it, when do we use it and what components do it takes and what component returns-

Higher order component basically takes a component as an input and then enhances the component, it adds extra feature to that component and returns it back.

It adds like a enhancer kind of function, It takes the existing component and just enhances it , just modify the components just little and return it back.

Will develop the feature – In the swiggy.com will clearly see there are multiple restaurants but some of the restaurants are promoted. All the cards are similar only the difference for some cards have promoted label on the top. So we are going to create the feature promoted label on the restaurant card.

How we will know which restaurant are promoted or which one is not promoted?

First print the “`console.log(listOfRestaurants)`”

If I will go to each restaurant and see its data, there is a promoted variable which we are getting inside data, it gives us the information whether the restaurant are promoted or not.

If the promoted flag is true that means you have to show promoted label on the top of the card otherwise you don't have to show the promoted label on the top of the card.

Write the logic – if the restaurant is promoted than add a promoted label to it.

First of all create the higher order function that is give us the card with the promoted lable.

We want another restaurant card that is enhance version of the restaurant card but it has the promoted label top of it.

Create the Higher order component that take input as the RestaurantCard and the output will be RestaurantCard promoted.

withPromotedLabel is the higher order component which takes RestaurantCard as input and it will return another new component, but this time promoted label on the top of it, How do you return the component, so the component is just a function again so it will return another function inside it

Basically

Return() =>{ } -> is the new functional component and this function again return JSX and this JSX is basically the enhance component . It will have the promoted label inside of it and it have <Resturantcard/> with it.

```
// Higher Order Component

// input - RestaurantCard ==>> RestaurantCardPromoted

export const withPromotedLabel = (RestaurantCard) => {
  return (props) => {
    return (
      <div>
        <label className="absolute bg-black text-white m-2 p-2 rounded-lg">
          Promoted
        </label>
        <RestaurantCard {...props} />
      </div>
    );
  };
};
```

Here I am just adding label promoted over my restaurant card

Now just use the higher coder component and its generate the restaurant card for us.

So over here what we will do is we will now when we will render over here so according to promoted label we are going to render all the restaurant card but the restaurant promoted we will render the new promoted card.

How do we use it-

Import the withPromotedLabel

```
import RestaurantCard, {withPromotedLabel} from './RestaurantCard';
```

Now I will create the restaurantcard component which has the withPromotedLabel in it..

How will I have created-

```
const RestaurantCardPromoted = withPromotedLabel(RestaurantCard);
```

Now what will happen is this withPromotedLabel is the higher order function we have passed this RestaurantCard in it, It will return as an new component wich has the label inside of it. So now basically this RestaurantCardPromoted component has the PromtedLabel inside it.

Now what we are going to do is over here if the restaurant.data.promoted is true I will render my new RestaurantCardPromoted instead of this card- `<RestaurantCard resData={restaurant}`

Write here the logic-

```
{
  //If the resutrant is promoted then add a promoted label to it
  restaurant?.info.promoted ? (<RestaurantCardPromoted
resData={restaurant?.info} />): (<RestaurantCard resData={restaurant?.info} />
)
  In api promoted obj are not there so I am using restaurant?.info.isopen
}
```

`<RestaurantCardPromoted/>` also need resData so we are using here

`<RestaurantCardPromoted/>` comes from `const RestaurantCardPromoted = withPromotedLabel(RestaurantCard);`

But here we are using props to it-

`(<RestaurantCardPromoted resData={restaurant?.info}`

If we are pass props resData={restaurant?.info} here where will be received it
Will receive it return function –

```
export const withPromotedLabel = (RestaurantCard) => {  
  return (props) => {  
    return (  
      <div>  
        <label className="absolute bg-black text-white m-2 p-2 rounded-lg">  
          Promoted  
        </label>  
        <RestaurantCard {...props} />  
      </div>  
    );  
  };  
};
```

And when we called withPromotedLabel - > const RestaurantCardPromoted =
withPromotedLabel(RestaurantCard); it will return the this return() component .

Also we have to passing same resData in <RestaurantCard {...props} />

If I will write this ... spared operator it will pass all props that I will received , it will direct pass it
to the ResturantCard.

Why we are doing this when we write higher order component these HOC is pure function
Pure function means that it will not change or modify the code of existing ResturantCard, we are
not changing the behaviour of ResturantCard any way. Here I am just taking ResturantCard as
input and I am using it exactly how we are use ResturantCard , not changing anything in
ResturantCard ,not modify the feature of the ResturantCard, I am just adding top of it,
enhancing it. It will enhance the component may be its UI, may be passing extra props. But not
modify this ResturantCard directly, not change the code of the ResturantCard .

The important thing in react is to manage your data. In any UI is the different part and data is
the different part in any application and data layer is very important . In generic way all the UI
application have two layers one is the UI layer, and another one is the data layer and the Ui layer
is powered by the data layer. So basically this data layer consist of state , props your local

variables whatever data have in your application is the data layer and UI layer is powered by the data layer. This data layer is very important, If you know how to manage data in your react application your application will be super-fast, very performant and it very important part is to manage your data layer. Basically, when you say UI layer mostly consist of JSX, the code written inside JSX is the UI layer and the data layer is your state, props your local variable , curly braces inside the JSX the JavaScript code you write all modifying the data layer . This is UI layer and the data layer.

Here we will learn how we will manage the data layer, manager data properly in your app.

Will build the feature if I will go to the restaurant menu after click on it we have restaurant table on the page and we have separate section like recommended section, Newley added so on...

And the section have some items, we are going to build similar kind of things inside the application.

Fetch the data from API-

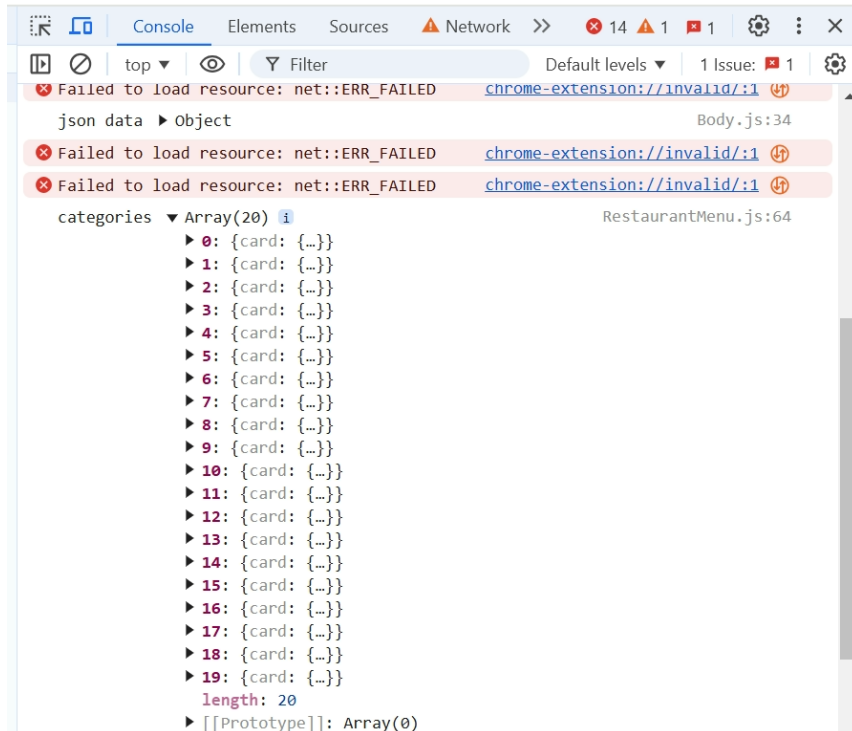
I want to find out all data of all the itemCategory , trying to filter all the cards which has "type.googleapis.com/swiggy.presentation.food.v2.ItemCategory"

```
const categories =
resInfo?.cards[4]?.groupedCard?.cardGroupMap?.REGULAR?.cards[4]?.card?.card.filter(c =>
c.card?.["card"]?.["@type"] ==
"type.googleapis.com/swiggy.presentation.food.v2.ItemCategory")

console.log(categories)
```

If the property which is not variable inside the JavaScript, You have to wrapped it inside [""]
Even we can do ["card"] this is also valid JavaScript. This is exactly same .card?.card
Because here is @ so we cant write directly like @type otherwise it will throw an error.

When we do console.log("categories", categories) will get all the categories



Now create the accordion –

It has two things – accordion header and accordion data

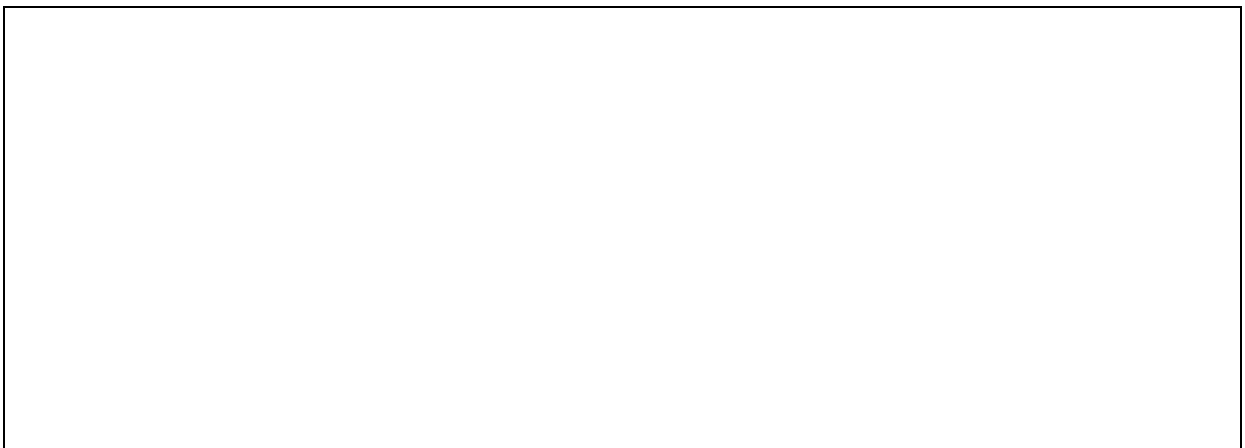
RestaurantCategory.js component – create for accordion header

ItemList.js – create for accordion body

create for accordion header-

1. We will loop all the categories and for each category we will build an accordion header/accordion item for it we will create the “RestaurantCategory.js” component

RestaurantCategory.js



```

import { useState } from "react";
import ItemList from "./ItemList";

const RestaurantCategory = ({ data, showItems, setShowIndex, dummy }) => {
  const handleClick = () => {
    setShowIndex();
  };
  return (
    <div>
      { /* Header */ }
      <div className="w-6/12 mx-auto my-4 bg-gray-50 shadow-lg p-4">
        <div
          className="flex justify-between cursor-pointer"
          onClick={handleClick}>

          <span className="font-bold text-lg">
            {data.title} ({data.itemCards.length})
          </span>

          <span>⌵</span>
        </div>

        {showItems && <ItemList items={data.itemCards} />}
      </div>
    </div>
  );
};

export default RestaurantCategory;

```

Loop the ResCategory –

RestaurantMenu.js

for each `category` returns `<RestaurantCategory/>` and import it on the top, now showing 20 resCategory, also Inside it passing key

```
{categories.map((category, index) => (  
  <RestaurantCategory  
    key={category?.card?.card.title}  
    data={category?.card?.card}  
  />))}
```

Let's build the header of each category, fist passing the props from **RestaurantMenu.js** to **RestaurantCategory.js**

Pass data= {category?.card?.card} as an props and
what data will be get -> let's call it data

```
const RestaurantCategory = ({ data }) => {
```

```
<span className="font-bold text-lg"> {data.title} ({data.itemCards.length}) </span>  
<span>⌵</span>
```

For Accordion body create one more component (**ItemList.js**)

I have to pass the itemCards as the props and get it on the ItemList.js

```
<ItemList items={data.itemCards} />
```

Inside **ItemList.js** print the name, Price and description and iterate it.

```
{items.map(item =>  
  <div  
    key={item.card.info.id} className="p-2 m-2 border-gray-200 border-b-2 text-  
left flex justify-between">
```

```

    <div className="w-9/12">
      <div className="py-2">
        <span>{item.card.info.name}</span>
        <span>
          - ₹
          {item.card.info.price
            ? item.card.info.price / 100
            : item.card.info.defaultPrice / 100}
        </span>
        <p className="text-xs">{item.card.info.description}</p>
      </div>
    </div>
  </div>)
}

```

Sometime we have price and sometime we have default price apply condition here -

```

    {item.card.info.price
      ? item.card.info.price / 100
      : item.card.info.defaultPrice / 100}
    </span>

```

Now apply image here and Add button -

```

<div className="w-3/12 p-4">
  <div className="absolute">
    <button
      className="p-2 mx-16 rounded-lg bg-black text-white shadow-lg"
      onClick={() => handleAddItem(item)}>
      Add +
    </button>
  </div>
  <img src={CDN_URL + item.card.info.imageId} className="w-full" />
</div>

```


Let's make the header clickable-

```
const handleClick = () => {  
  console.log("clickable")  
};  
  
<div className="flex justify-between cursor-pointer" onClick={handleClick}>  
  <span className="font-bold text-lg">{data.title}{data.itemCards.length} </span>  
  <span>⌵</span>  
</div>
```

Now we want on click of the header it should expand and collapse , how can I make it –
This <ItemList /> is on the UI layer , now I want to handle show and hide from data layer . I
have to work on my data layer. So this data layer has a state variable and the state variable
decide whether this data list will be show or not .

So now I will create the normal state variable , that variable will control show items or not
RestaurantCategory.js-

```
const [showItems, setShowItems] = useState(false);
```

Write the logic –

If the showItems will present then only show the <ItemList /> otherwise don't show it

```
{showItems && <ItemList items={data.itemCards} />}
```

Now all the accordion header collapse. Now I have written the logic of show it

```
const handleClick = () => {  
  setShowItems(true)  
};
```

It will only show the items. Now I have built it the toggle feature –

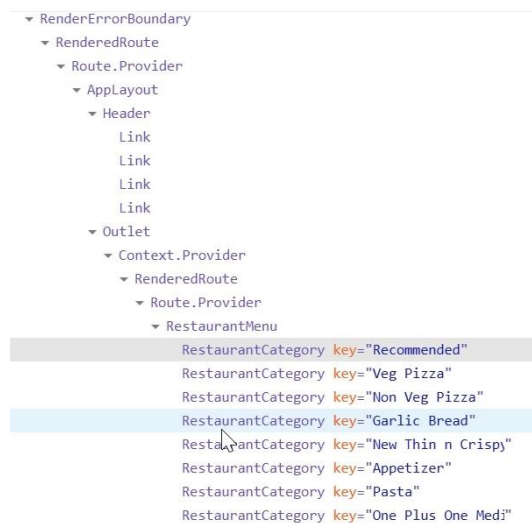
```
const handleClick = () => {  
  // toggle feature  
  setShowItems(!showItems)  
};
```

we have created the accordion it just shows and hide the itemList. Suppose I have to build a feature when I click on the accordion so it will close all the other accordion and open only clickable accordion.

It is very tough to build this type of feature because each of these restaurant categories have their own state , managing their own state , managing there show item

In other words this recommended accordion component controlling the item list and newly accordion component aware about what is happening inside recommended component, newly added component doesn't know whether the recommended component collapses or expended and whatever. So, what I have to do this suppose if I build this feature, if I open recommended it should collapse everything else. I want this state to be lifted up. So I don't give the power of show and collapse item list to this recommended. I will give the power of show and collapse to the parent of this child.

I don't want to give the power of **RestaurantCategory.js**. I want to give the power of show and hide to the **RestaurantMenu (parent)**. Basically, what I am trying to say is I want RestaurantCategory to take an input whether you have show items or not. So in RestaurantCategory whatever I am passing here should be follow. So basically I am lifting my state up. What I am trying to say RestaurantCategory has its show items here I don't want . I want **RestaurantMenu** to control all these RestaurantCategory that is how we can expend and collapse any RestaurantCategory here we want to



So whatever I pass in from the top it will do this . so now RestaurantCategory does not its own state , I don't want RestaurantCategory have its own state. I want RestaurantMenu to tell whether you should show items or not. RestaurantMenu is the parent and it has a lot of RestaurantCategory I want the parent to control its own children, If I say showItem it will show visa verse . I want to give this power to RestaurantMenu. So I don't want RestaurantCategory have there own state so will remove it

```
// const [showItems, setShowItems] = useState(false); (remove)
```

But it will take my showItems from props-

```
const RestaurantCategory = ({data, showItems, setShowIndex })
```

form where the props comes, It will comes from parents. So, if I will pass `showItems = "true"` it will show the accordion and if I will pass `showItems = "false"` it will not show the accordion. Here I will give you the jargon **controlled** and **uncontrolled** component and this is the controlled component (RestaurantCategory) because you are controlling by the parent and When RestaurantCategory has its own state, it was the uncontrolled component because RestaurantMenu (parent) doesn't its controlled if you want to show and hide it can do itself. It is controlling itself this was not controlled it was controlling itself so its un-controlled component. Parent doesn't have the control to their children.

This is **the difference between controlled and un-controlled component**

If the RestaurantCategory controlling itself the showItems, it would have been uncontrolled component and if I will take away this power then it is a controlled component because it is relying upon its parents to tell what it's do, now it's a controlled component because I am controlling it with RestaurantMenu.

There is no specific definition of controlled and un-controlled component , it is just the philosophy some people say RestaurantCategory is the controlled component that means I can controlled by a props , it doesn't have its own state , it can have its own state but the state doesn't have the main state , majorly I am controlling it by the parents , So it is the control controlled component .

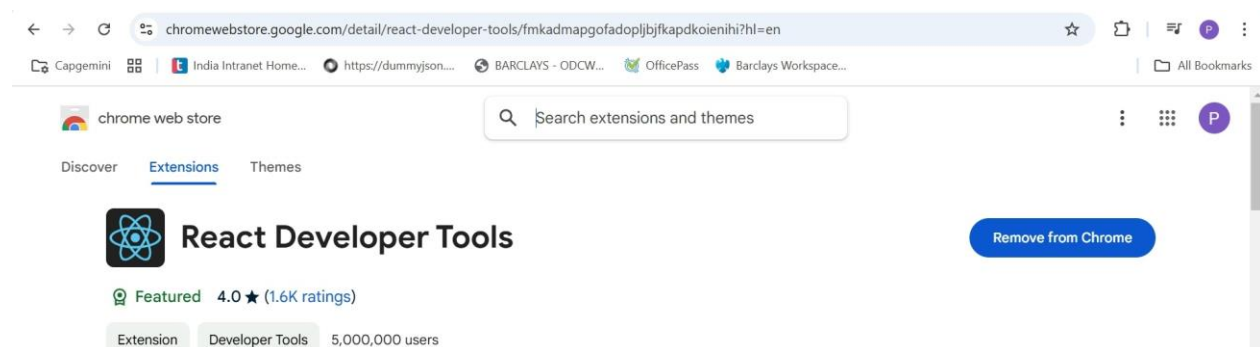
```
{categories.map((category, index) => (
  <RestaurantCategory
    key={category?.card?.card.title}
    data={category?.card?.card}
    showItems={false}
  />))}
```

If I am not sending `showItems={false}` and it will managing itself it was uncontrolled component.

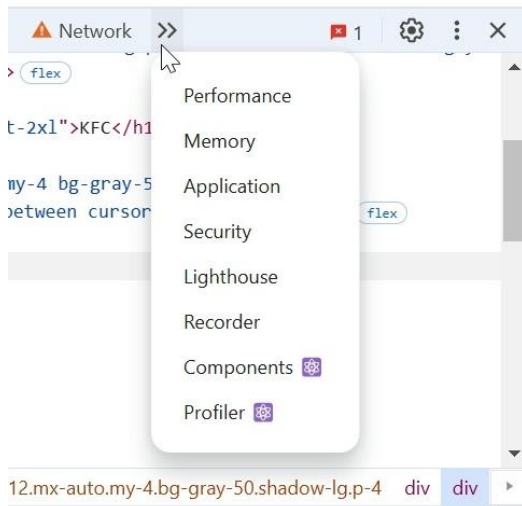
Now the parents has controlled to hide all the items /collapses all the accordion and parents controlled all the RestaurantCategory

If I will do `showItems={true}` it will expands all the accordians

Install react dev tools Chrome—



Once we have install this extension , our developer console will get some super powers
We will get – Components and profiler (It will be coming from the extension)



If we will click on **components**, It will show you all the list of components hierarchy over here. Another cool thing about this extension is if I click on any of the component it will show you the data layer.

So basically, left side is the UI layer and This is the JSX, you can also call as the virtual DOM/ Representation of the virtual Dom and the Right hand side is the data layer here it will show all the props that the component are receiving.

This extension is very useful for the debugging.

Profiler- When I click on the profiler there are three things – flamegraph, ranked, timeline

How do you use the profiler - It will basically record the react application

So, whatever you do inside the react application it will try to record it

So, when I move from one component to another it will record all are actions

And if I stop recording it will give you all the actions that we are doing. Also, it will be showing the time

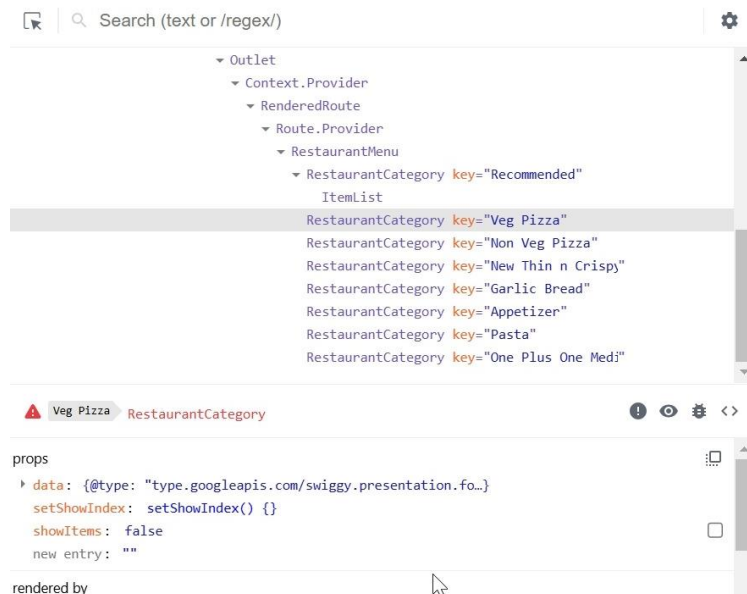
You might have doubt what is router provider, data provider, navigation
I did not add these component, these components added by the packages
Extra things are added by the libraries, external libraries that use we have import

Now we are trying to add the behaviour if I click on the accordion then the clickable accordion should open and other should close.

All the RestaurantCategory accordion items they have their own state, they have their own show items



The show items of recommendation are true right now. The show items of veg pizza is false right now.



What I am trying to say is all the RestaurantCategory are maintain their own state.

The power of collapse in the hand of RestaurantCategory

Suppose I have to build the feature if I open recommended should collapse everything else. I want this state to be lifted up. So I don't want to give the power of show and collapse to this recommended. I will give the power of show and expand and collapse to the parents of this child.

Now, the RestaurantCategory power to show and hide the item list

```
const handleClick = () => {  
  // toggle feature  
  // setShowItems(!showItems)  
};
```

But I don't want to give this power to RestaurantCategory, I want to give this power to RestaurantMenu to parent. What I am trying to say I want my RestaurantCategory to take an input showitem or not, whatever I am passing here should be follow.

```
components > JS RestaurantMenu.js > RestaurantMenu > categories.map() callback  
const RestaurantMenu = () => {  
  <li>Burgers</li>  
  <li>Diet Cake</li> */}  
  { /* </ul> */}  
  
  { /* categories accordions */}  
  {categories.map((category, index) => (  
    <RestaurantCategory  
      key={category?.card?.card.title}  
      data={category?.card?.card}  
      // showItems={false}  
  
      // Implement the lifting state up  
      showItems={index === showIndex ? true : false}  
      setShowIndex={() => setShowIndex(index)}  
    />))}  
  </div>  
)  
}
```

Basically, I am lifting my state up. I want RestaurantMenu to controlling all the list. That is how we can expand and collapse any RestaurantCategory here we want.

So now the RestaurantCategory will not have its own state, I don't want to RestaurantCategory have it show items. I want RestaurantMenu to tell whether you should

showItems or not. RestaurantMenu is the parents, and it has lots of RestaurantCategoryv I want to the parent control the children. If I say showItem = true, it will show and if I say hide it will hide that means showItem = false

So, **remove –**

```
// const [showItems, setShowItems] = useState(false);

const handleClick = () => {
  // console.log("clickable")
  // setShowItems(true)

  // toggle feature
  // setShowItems(!showItems)
};
```

But it should take my showItem from props

```
const RestaurantCategory = ({ data, showItems, setShowIndex }) => {

{showItems && <ItemList items={data.itemCards} />}

}
```

Basically, if I pass showItems is false then show my ItemList, if I pass is true doesn't show.

Controlled and Un-controlled components –

So now RestaurantCategory is the controlled component because you are controlling the parents the RestaurantMenu components controlling the RestaurantCategory now . So, it is the controlled component now

```

components > JS RestaurantMenu.js > RestaurantMenu > categories.map() callback
const RestaurantMenu = () => {
  <li>Burgers</li>
  <li>Diet Cake</li> */}
  /* </ul> */}

  /* categories accordions */}
  {categories.map((category, index) => (
    <RestaurantCategory
      key={category?.card?.card.title}
      data={category?.card?.card}
      // showItems={false}

      // Implement the lifting state up
      showItems={index === showIndex ? true : false}
      setShowIndex={() => setShowIndex(index)}
    />))}
  </div>
}

```

When it has own state, it was an uncontrolled component because RestaurantMenu does not have it controlled. If it want to show and hide something it can do itself

The parents don't have controlled over children so its uncontrolled component

RestaurantMenu doesn't have full control over RestaurantCategory

```

components > JS RestaurantCategory.js > RestaurantCategory
import { useState } from "react";
import ItemList from "../ItemList";

const RestaurantCategory = ({ data, showItems, setShowIndex }) => {
  // console.log("data", data)
  ⚡ const [showItems, setShowItems] = useState(false);

  const handleClick = () => {
    // console.log("clickable")
  }
}

```

If the RestaurantCategory controlling itself showItems it would have been a uncontrolled component and if I take away this power then it is an controlled component because it is relying on its parent to tell it what it do , Now this is an controlled component , I am controlling it from RestaurantMenu. There is no specific definition of controlled and uncontrolled component. It just the philosophy.

Suppose I want to only send true for my first accordion

```
{categories.map((category, index) => (
  <RestaurantCategory
    key={category?.card?.card.title}
    data={category?.card?.card}
    // showItems={false}

    // Implement the lifting state up
    showItems={index === 0 && true}

  />))}
```

Index are coming from my map function and first parameter “category” each item for this map and second is index

Result first accordion is expand and other accordion will be collapsed .

Let’s make the second one expanded-

```
{categories.map((category, index) => (
  <RestaurantCategory
    key={category?.card?.card.title}
    data={category?.card?.card}
    // showItems={false}

    // Implement the lifting state up
    showItems={index === 1 ? true : false}

  />))}
```

If my index 1 then make true otherwise all of them are false

RestaurantCategory are the controlled component

But I want to build the feature if I click on this recommended it will show itself and collapses all the other accordion. I am trying to build is at the time one accordion should be expand .

How will I controlled that I can take this index in my state

```
const [showIndex, setShowIndex] = useState(null);
```

If the showIndex is 0 first will be open, If the showIndex 1 second will be open and other will be collapse . Accordion to my showIndex I want to control showItems.

So If my index === showIndex, initially my show index is null or 0 so it send true. So if I change my show index somehow it will automatically work

Can I modify the state variable of my parent from my children can I do that

Ans is not possible directly but indirectly we can do that

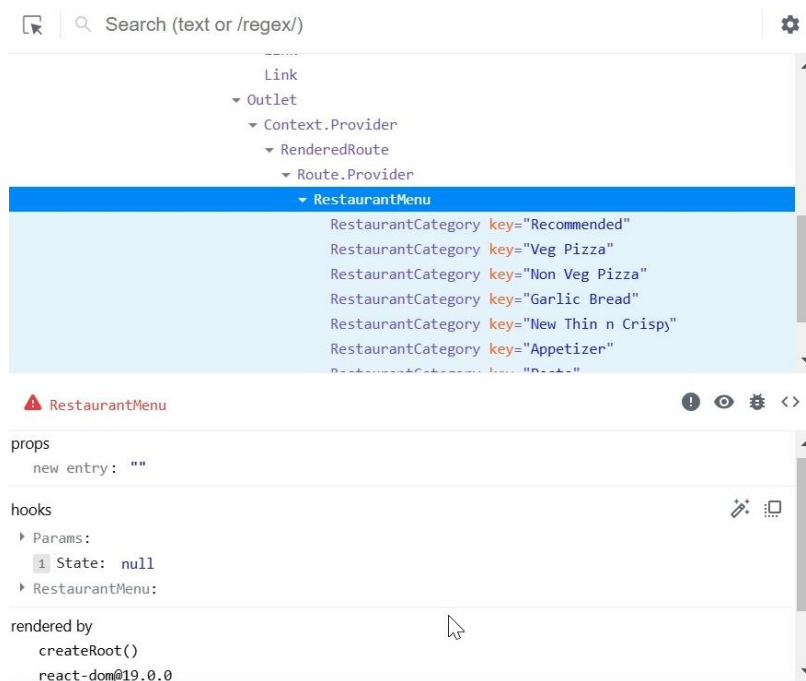
We will pass setShowIndex to my children. How will I pass I will pass my function and it will be called index

```
setShowIndex={() => setShowIndex(index)}
```

this function is basically set the ShowIndex of the particular index. So now basically I can set the setShowIndex I can pass the feature to my children

```
const RestaurantCategory = ({ data, showItems, setShowIndex }) => {  
  
  const handleClick = () => {  
    // lifting state up  
    setShowIndex()  
  };  
}
```

So now as soon as I click on accordion I will just set the showIndex from its parent, it will just change the index of it and all the other will be collapsed . So now it will show itself whenever we click on it.



Now the RestaurantMenu has the state variable and initially is 0 but when I click on it will change to 1

<https://react.dev/learn/sharing-state-between-components#lifting-state-up-by-example>

Props Drilling –

What happens in the real-world application there are lots of components and diff hierarchy. Basically a react application when it grows big so there are components and there are components within them. It's like a big tree structure, there are lots of levels nested inside it and passing data between the components is a very big challenge when the allocation is huge and react has one way data flow, data is passing from parents to children's and children to their children. Data flows from one direction \

Example –

Suppose I want to pass this dummy data to the leaf of the tree

In RestaurantMenu we have this data now I want to access this data inside my itemList. How I want to fetch that

```

  ▼ RestaurantMenu
    ▼ RestaurantCategory key="Recommended"
      ItemList
        RestaurantCategory key="Veg Pizza"
        RestaurantCategory key="Non Veg Pizza"
        RestaurantCategory key="New Thin n Crispy"
        RestaurantCategory key="Garlic Bread"
        RestaurantCategory key="Appetizer"
        RestaurantCategory key="Pasta"

```

Here is tree so can pass it directly , we have pass the data to the intermediate parents
First I want to pass the dummy data to RestaurantCategory and then pass to the itemlist

```
//Props Drilling
const dummy = "Dummy Data";
```

```
src > components > JS RestaurantMenu.js > RestaurantMenu > categories.map() callback
```

```
11  const RestaurantMenu = () => {
12      // <div>
13      // <ul>
14      // </ul>
15      // </div>
16      /* categories accordions */
17      /* controlled component */
18      {categories.map((category, index) => (
19          <RestaurantCategory
20              key={category?.card?.card.title}
21              data={category?.card?.card}
22              // showItems={false}
23              // Implement the lifting state up
24              showItems={index === showIndex ? true : false}
25              setShowIndex={() => setShowIndex(index)}
26              dummy={dummy}
27          />))}
28      </div>
29  )
30  }
```

Now the RestaurantCategory received the dummy data then its pass to the itemList children

```
const RestaurantCategory = ({ data, showItems, setShowIndex, dummy }) => {
```

```
{showItems && <ItemList items={data.itemCards} dummy={dummy}/>}
```

Now I can access dummy data inside my itemList, so now I am passing dummy data as a prop.

```
const ItemList = ({ items, dummy }) => {  
  // console.log("items", items)  
  console.log(dummy)
```

The screenshot shows a web browser displaying a Pizza Hut menu. The menu has a header with navigation links: Online Status, Home, About us, Contact us, Grocery, Cart, and Login. Below the header is a section titled "Recommended(20)" with four items, each with an "Add +" button. The items are: My Box - Veg- ₹249, My Box - Non Veg- ₹319, Hut Treat Meal for 2 - Non Veg- ₹589, and Hut Treat Meal for 2 - Veg- ₹529. To the right of the browser is the Chrome DevTools console. It shows several error messages: "web_accessible_resources manifest key in order to be loaded by pages outside the extension.", "Failed to load resource: net::ERR_FAILED", and "Returning a Promise is the preferred way to send a reply from an onMessage/onMessageExternal listener, as the sendResponse will be removed from the specs (See https://developer.mozilla.org/docs/Mozilla/Add-ons/WebExtensions/API/runtime/...)". There are also messages about "Dummy Data" and "ItemList.js:4".

It is printed in the item list

The problem with passing props

Passing props is a great way to explicitly pipe data through your UI tree to the components that use it.

But passing props can become verbose and inconvenient when you need to pass some prop deeply through the tree, or if many components need the same prop. The nearest common ancestor could be far removed from the components that need data, and lifting state up that high can lead to a situation called “prop drilling”.



Prop drilling



We have to avoid props drilling. suppose if we pass to data one or two levels this is still ok but what if the data present in somewhere and we have to access it somewhere else. how do I do that? In the large application this is the very common scenario where sometimes we need to have some kind of global data that can I access anywhere in my app how we can do that – we can do that by react context. We can avoid props drilling if we use context.

Lot of people think if I have to some global data I will keep it in the top level component and then pass it as an props but how much level. It will be fools if we pass 10 level deep, not a good way. So what we can do is we use context which is kind of global place where the data is kept and anybody can access that is known as react context.

Some scenarios-

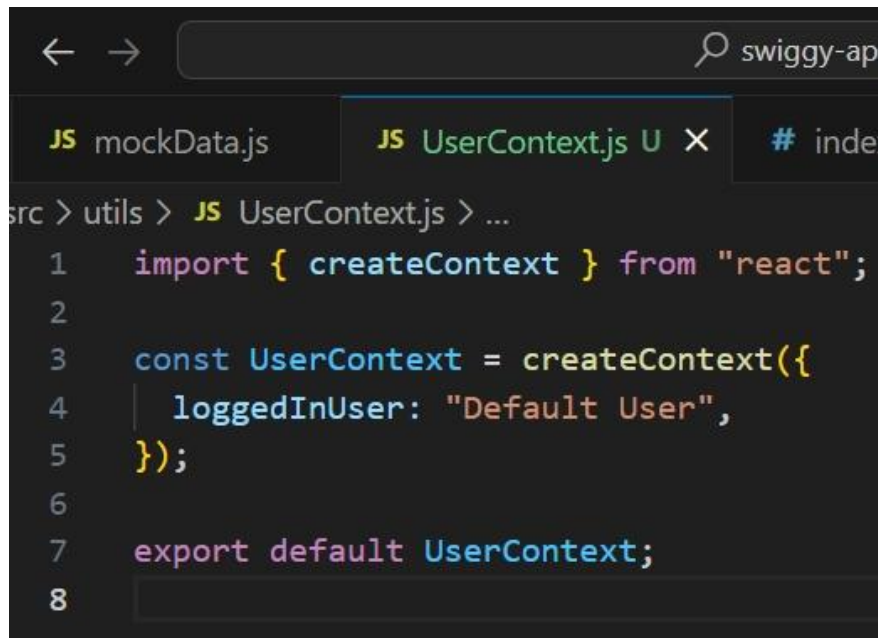
Suppose think of some data which can be needed anywhere in our application, so many components, so many pages we need to keep in the central place . What type of could it be One example of data is locked in user, once the user locks in sometimes we need the locking information inside it (header also) sometime you must seen if we go to Gmail on the top right hand corner you show your name pragati with locked you username and the locked in user name inside any card and it can we use in some other page. So basically, so piece of data we can access anywhere in our app one example is locked in info.

Another example is theme we need the theme (light and dark). Suppose if it is dark theme, I need it all across my app so there is some data that we need anywhere in app for that data we hold it inside in context and the context can we use anywhere in app. Let's try to create the context –

Context is avoid the props drilling if we have the central store we can keep the data and we can access anywhere in the app. There is many ways to avoid props drilling but the context is also the good way

Create the context – How do we create it we can create it using createContext and it comes from react library and when we create the context we will give it some piece of information that will hold . Its kind of central global object. We are passing the default value to initials here

```
loggedInUser: "Default User",
```

```
src > utils > JS UserContext.js > ...
1  import { createContext } from "react";
2
3  const UserContext = createContext({
4    |   loggedInUser: "Default User",
5  });
6
7  export default UserContext;
8
```

Now we can access it anywhere in our app

Suppose if I have to access it in my header component. We can access it by using react hook and pass in "UserContext"

```
import UserContext from "../utils/UserContext";

const { loggedInUser } = useContext(UserContext);
```

Display –

```
<li className="px-4 ">{loggedInUser}</li>
```

Why do we pass it UserContext?

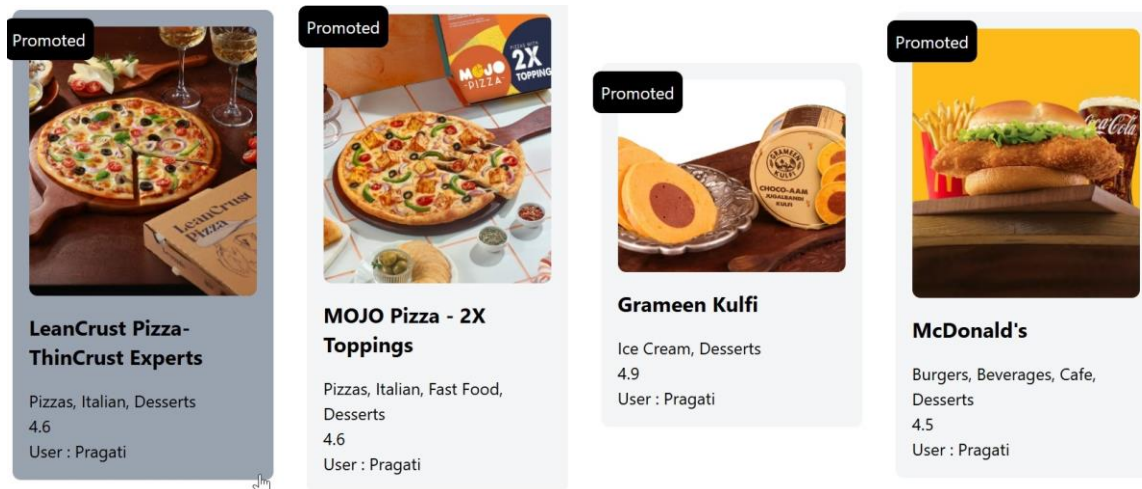
In our whole react application there is not just one context that we are creating. We can create as many contexts we want to, as many global context we want to

Now want to access data in RestaurantCard

```

RestaurantMenu.js (Working Tree) M    JS useContext.js (Untracked) U    JS Body.js (Working Tree)    JS RestaurantCard.js M X
src > components > JS RestaurantCard.js > RestaurantCard
1  // import the name import
2  import { useContext } from 'react';
3  import { CDN_URL } from '../utils/constants';
4  import useContext from '../utils/UserContext';
5
6  const RestaurantCard = (props) => {
7    const { resData } = props
8    const { cloudinaryImageId, name, cuisines, avgRating, costForTwo, deliveryTime } = resData?.info;
9    // const { deliveryTime } = resName.info.sla...
10
11    const { loggedInUser } = useContext(UserContext);
12
13
14    //console.log(loggedInUser);
15    return (
16      <div className="res-card m-4 p-4 w-[250px] rounded-lg bg-gray-100 hover:bg-gray-400"
17        // style={{ backgroundColor: "#f0f0f0" }}
18      >
19        <img className="res-logo" src={"https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/" + cloudinaryImageId} alt="" />
20
21        <img className="res-logo rounded-lg" src={CDN_URL + cloudinaryImageId} alt="" />
22
23        <h3 className="font-bold py-4 text-xl">{name}</h3>
24        <h4>{cuisines.join(", ")}</h4>
25        <h4>{avgRating}</h4>
26        <h4>{costForTwo / 100} FOR TWO</h4>
27        <h4>{deliveryTime}</h4>
28        <h4>User : {loggedInUser}</h4>
29      </div>
30    )
31  }

```



Now you can ask should we keep all the data inside context , If we keep all the data inside context then we don't have pass it props we don't have pass it data inside it components but that is not a case only the data that which you are use in the multiple places or you feel it can be use in multiple places that is where you can use you context . Don't put all data inside context does not make sense. But you can access it anywhere you want to

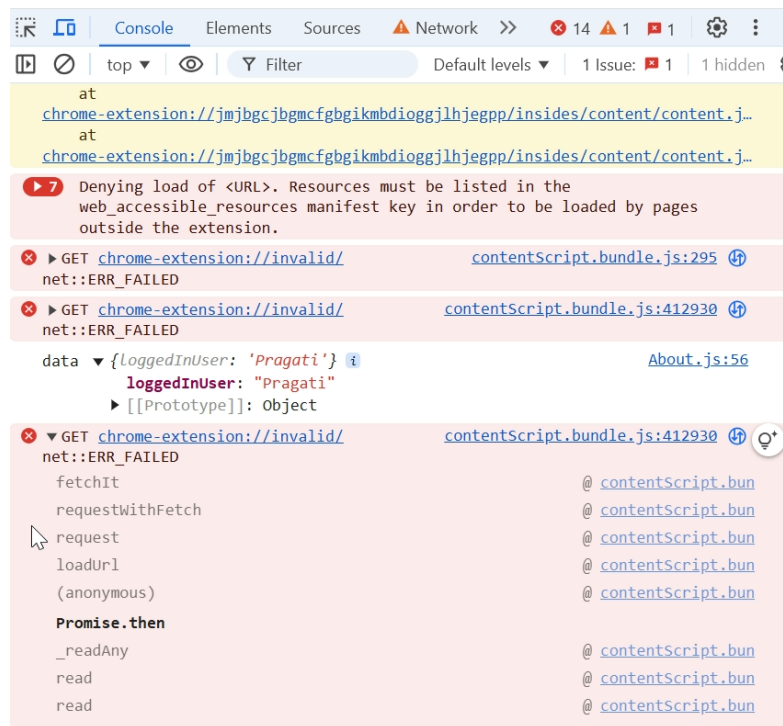
How can I access it context in class-based component (the way is different)–

```
import UserContext from "../utils/UserContext";

<div>
  LoggedIn User
  <UserContext.Consumer>
    ({ { loggedInUser } }) => (
      <h1 className="text-xl font-bold">{loggedInUser}</h1>
    )
  </UserContext.Consumer>
</div>
```

When you create the context react gives you power of .Consumer as well so you have two ways you can access the context

First one using the hooks in functional components and second way in class component if you don't have hooks use in this way - <UserContext.Consumer> and inside it you have JSX which has callback function and callback function get access to data and the data is the context data



Now print the loggedInUser

Both are the ways you consume context data

In the real world this is the default user / default value inside the contexts. What if we changed the value of user context. What will we not use the default value but somehow suppose we locked in as a user. I have written some of the authentication code in my app how would I modify the context

Let me go to the root level App.js component , suppose my authentication code written over here, make the API call to fetch some login see whether the password correct or not and we got some data so we have the authentication and we kept the data inside local state of our application

Suppose we have the Api gives us userInfo and we have kept our data inside useState

How we can implement the authentication –

I have created the useEffect and inside it we have the callback function and make an API call and send username and password and we got the result

Here I am using the dummy data

```
useEffect(() => {  
  // Make an API call and send username and password  
  const data = {  
    name: "Akshay Saini",  
  };  
}, []);
```

Now I want to keep this data inside the userInfo, How do I update the userInfo -> setUserInfo(userInfo) or give setUserInfo(data.name). So now Akshay Saini inside my userInfo Now we have to written the authentication logic here. Now everywhere value of my context is default user (Pragati) and suppose I want to update with Akshay Saini. How do I pass this context information to my app. To pass this new information to my app. I will use context provider. So just like we had the user.consumer

Import useContext

```
import useContext from "../utils/UserContext";
```

Now use the useContext.Provider that is provided to us by react and I will wrap my whole app inside this useContext.Provider and pass the whatever value I have to pass in . Suppose I have to pass in loggedInUser become my username. Value we are provided in whole app and whatever we wrap inside I have wrap my whole app inside the useContext.Provider that means anywhere inside my app we have the useContext.Provider the value will be this so now the app not using the default value , loggedInUser will be the new value - Akshay Saini

About

LoggedIn User - CallContext

Akshay Saini

This is the Namaste React Web Series



Online Status: ☒ Home About us Contact us Grocery Cart Login Akshay Saini

Like this way I am overriding the default value (Pragati) - like this way we are modifying the value . Here we are wrap the whole app , suppose I will just wrap the header so only in the header component content will modify and other place have the default value .

If I use my provider in the specific portion, I can do that too. This is the power of context Context is the global space I can provide to whole app or just the small portion of our app, I can create the multiple context and I can override if we want to , react will take care of it and it is performant

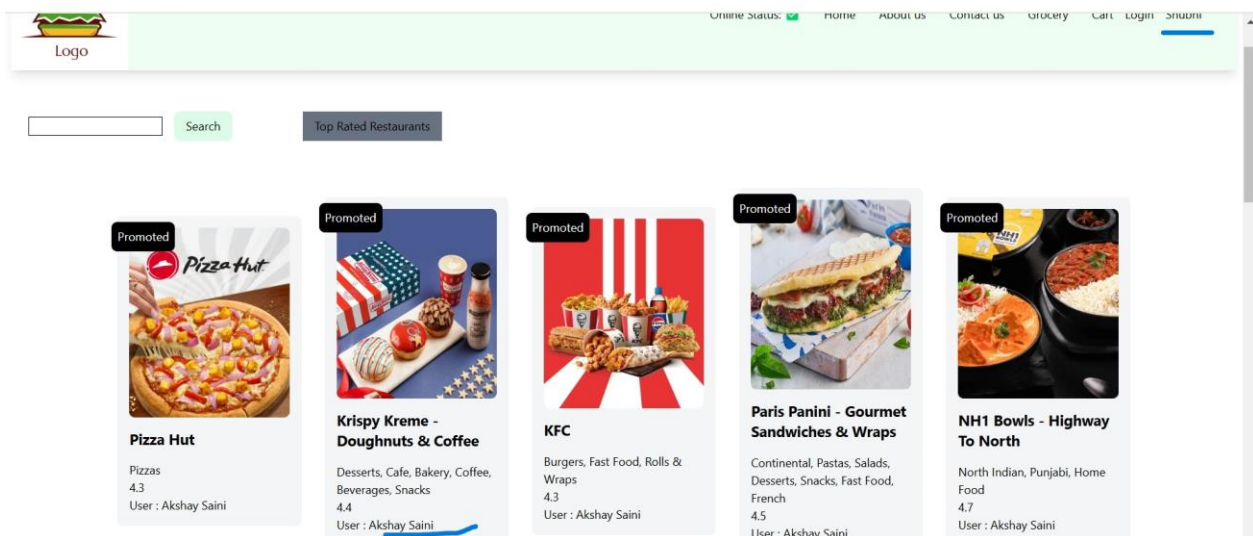
Que, can we do the nested provider

```

return (
  // Default value - Pragati
  <UserContext.Provider value={{ loggedInUser: userName }}>
    {/* Akshay Saini */}
    <div className="app">
      <UserContext.Provider value={{ loggedInUser: "Shubhi" }}>
        {/* Shubhi */}
        <Header />
      </UserContext.Provider>
      <Outlet />
    </div>
  </UserContext.Provider>
)
}

```

Header has Shubhi, all the place have **Akshay Saini**



Now just let us try to **modify the user context**

Create the input box and let's try to modify the username with inputbox

In my app.js the data is coming from

```
<UserContext.Provider value={{ loggedInUser: userName }}>
```

Whatever the data everybody is accessing this is coming from this value. How can I update this value , I have to update my username from that input box. How can I update it from inputbox?

First que how I will even update it by UserName so I will update it setUsername so how I will call the setUsername from that input box(body component)
I can pass the setUsername also here-

```
<UserContext.Provider value={{ loggedInUser: username, setUsername }} >
```

So I my body component if I want setUsername I can access it exactly the same way
First one import the useContext from react basically what I am done is in the context I have to put the function setUsername to modify itself so I have to access it everywhere

Import the context in body component –

Body.js-

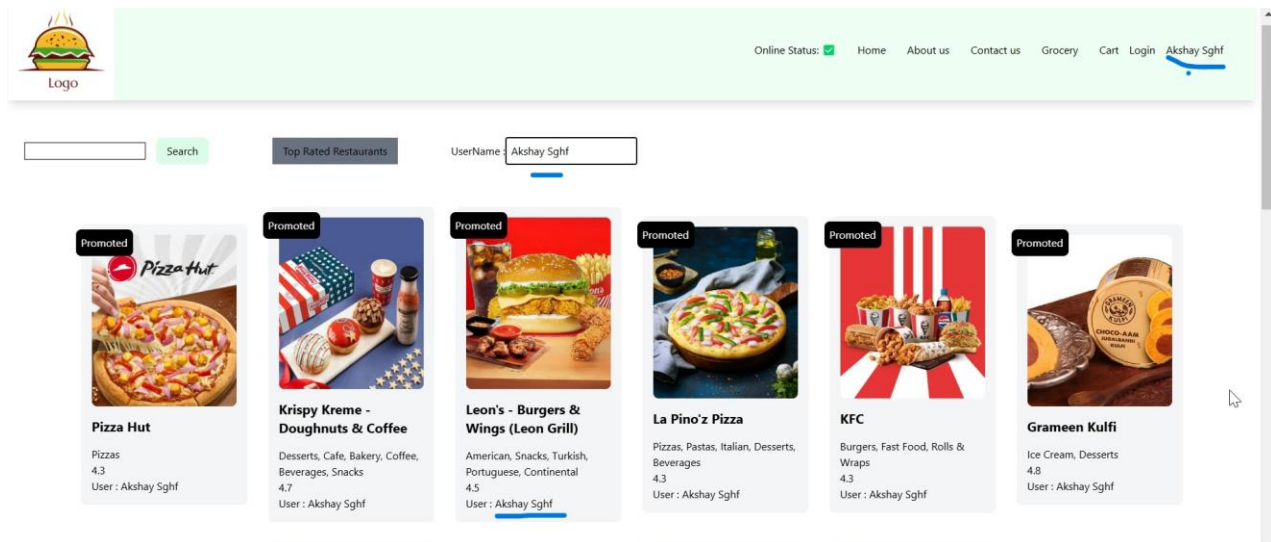
```
import UserContext from "../utils/UserContext"

const { loggedInUser, setUsername } = useContext(UserContext);

<div className="search m-4 p-4 flex items-center">
  <label>UserName : </label>
  <input
    className="border border-black p-2"
    value={loggedInUser}
    onChange={(e) => setUsername(e.target.value)}
  />
</div>
```

App.js-

```
<UserContext.Provider value={{ loggedInUser: userName, setUsername }}>
```

It will also change in about.js component –

