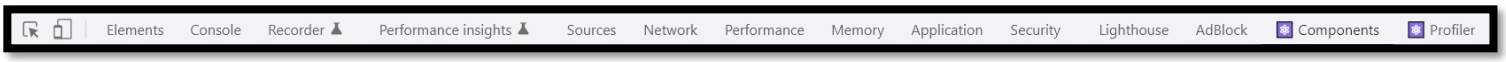
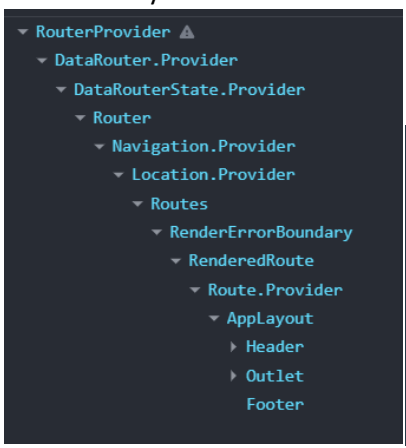
Some points :-

- Every react app is made up of 2 layers : Data layer and UI layer.
- Data layer is the one where the data is fetched and stored.
- UI layer is the one where the data fetched is actually rendered into the DOM.
- In data layer, the data is stored with the help of state and props. State can be imagined to be a local variable to a component that cannot be used outside that component. Props is a variable used to share some values from 1 component to another.
- **Props Drilling** :- Suppose we have a parent component with a state variable named user. Now we pass this user to a child component as props. Then we again pass user as props to a child of the child component and so on. This is called props drilling.
- **React Developer Tools** :- An extension that can be used for debugging react apps.
  Link: https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en

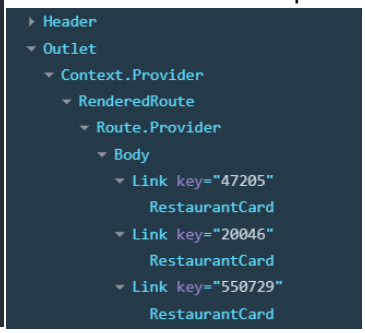
We can click on inspect for any react app and we can see 2 new sections after installing the extension :- Components and Profilers.



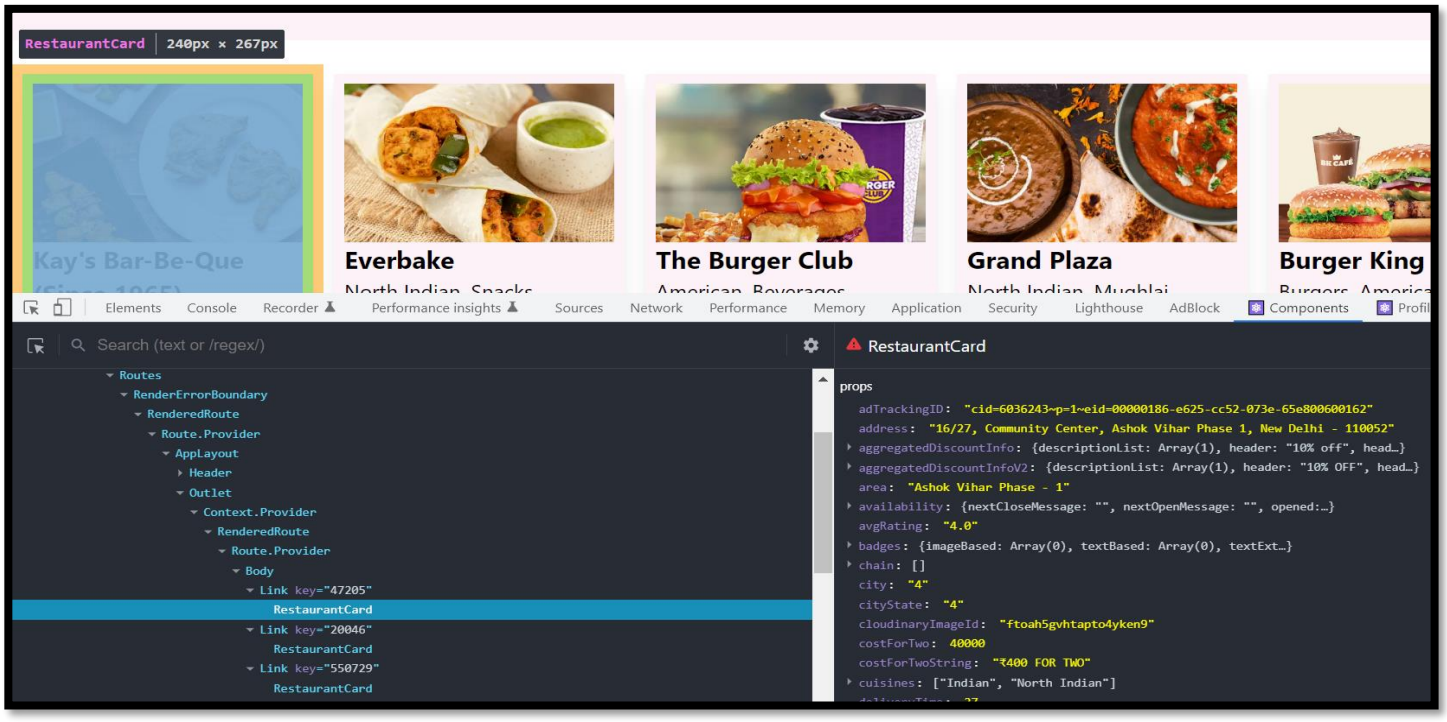
Now, we can click on components section. There we will find all the components that the react app is using. Some of them will be defined by React or other libraries directly, which is why we might not recognise them.
We know that our food app has the utmost parent component as AppLayout and it's child components are :- Header, Body and Footer.



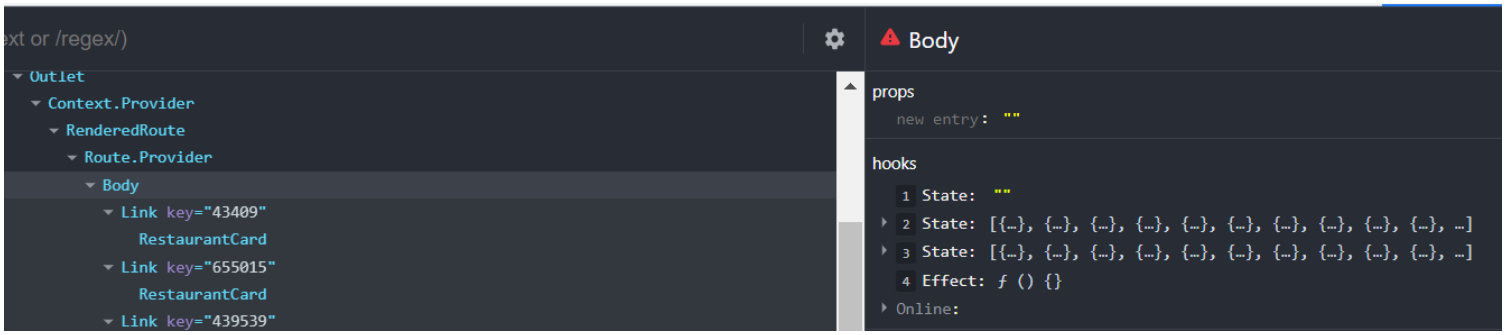
If we click on each of the child components, we will find further child components.
Example :- If we click on the Outlet component, we must find the Body component which must have a child component as Restaurant component.



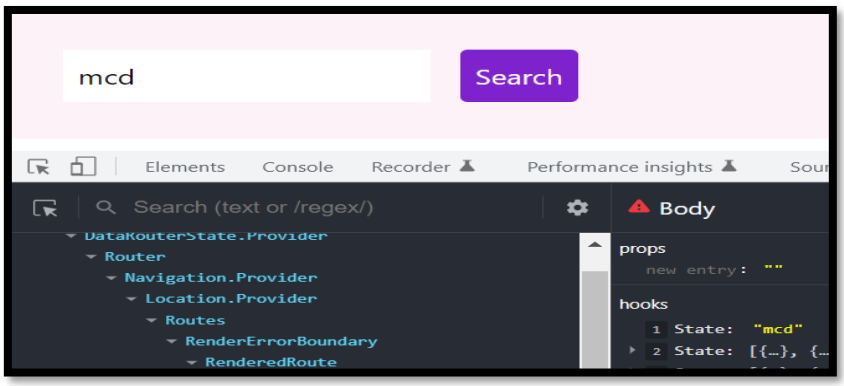
Now, if we click on the RestaurantCard components, we will find on the right side, all the props that is passed to that components, which in this case are the restaurant details.



Now, if we also see the Body component, we will find 3 different states being mentioned in the right side. The last 2 are for the allRestaurants and filteredRestaurants states and is an array. The first state is the search input which is an empty string by default.



Now, if we write something in the search input box, the state change will be reflected in the inspect section too.



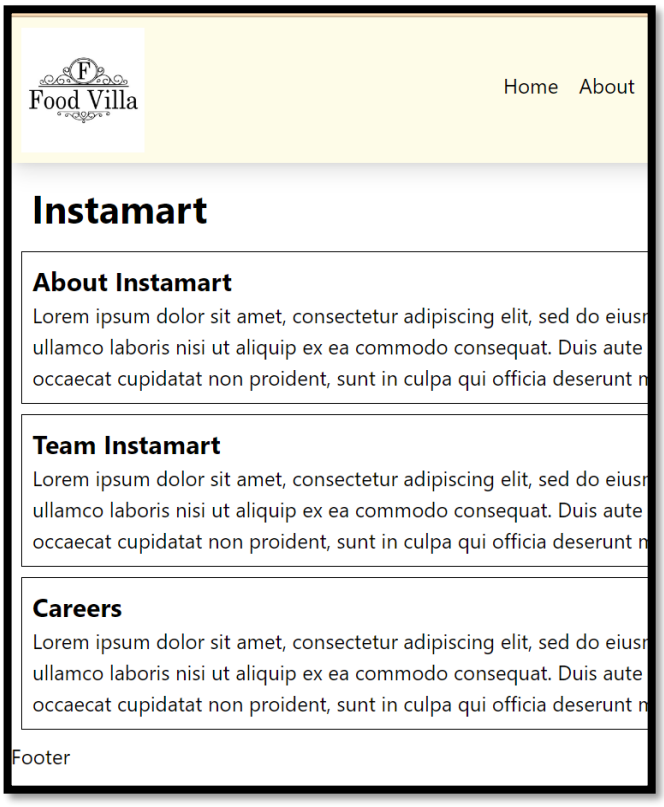
- **Problems of Props Drilling :-**
  Suppose we have passed the user as props from the AppLayout to Body and from Body to RestaurantCard. But the Body component is never using that user props and only RestaurantCard does, so we are unnecessarily passing props to the Body. This might also cause confusion during debugging the code.

Modifying the Instamart Page (Building an Collapsible Accordion) :-

Step 1 :-

We will make different sections in the Instamart page with each section having a heading and a description (filled with lorem ipsum here). The code and the output is :-

```jsx
const Section = ({ title, description }) => {
    return (
        <div className="border border-black p-2 m-2">
            <h3 className="font-bold text-xl">{title}</h3>
            <p>{description}</p>
        </div>
    )
}

const Instamart = () => {
    return (
        <div>
            <h1 className="text-3xl p-2 m-2 font-bold">Instamart</h1>
            <Section
                title={"About Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur a
            />
            <Section
                title={"Team Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur a
            />
            <Section
                title={"Careers"}
                description={"Lorem ipsum dolor sit amet, consectetur a
            />
        </div>
    )
}

export default Instamart;
```



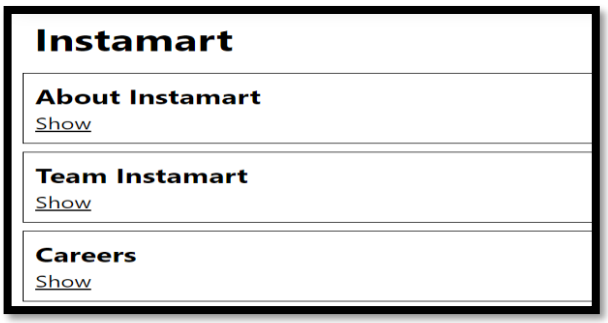
Step 2 :-

Now, I want to build a feature of Show&Hide such that if I click on a button the description will be hidden or shown. So, we want the visibility of a description of a section to change on clicking a button and we know that in React, any change in a component is brought about by a change in state variable. So, we have to create a state variable which denotes the visibility and this state must change on clicking a button that will be there for each section.

We will take a state variable named “isVisible” with a Boolean default value.

Therefore, we will render the description, only when isVisible is true.

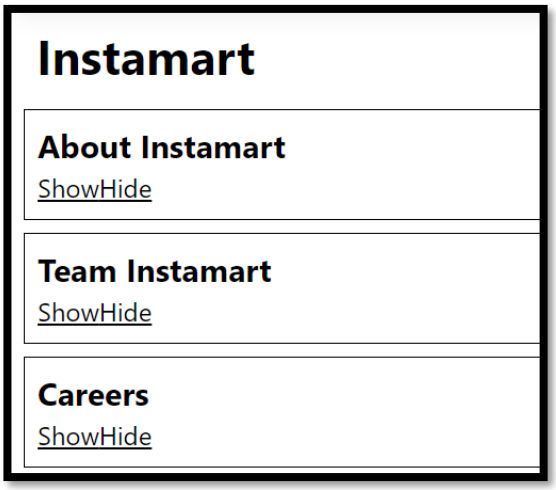
```
const Section = ({ title, description }) => {
    const [isVisible, useIsVisible] = useState(false);
    return (
        <div className="border border-black p-2 m-2">
            <h3 className="font-bold text-xl">{title}</h3>
            <button className="cursor-pointer underline">Show</button>
            {isVisible && <p>{description}</p>}
        </div>
    )
}
```



Now, we have to add the onClick function to the Show button, such that on clicking Show button, the isVisible is set to true and the description gets show.

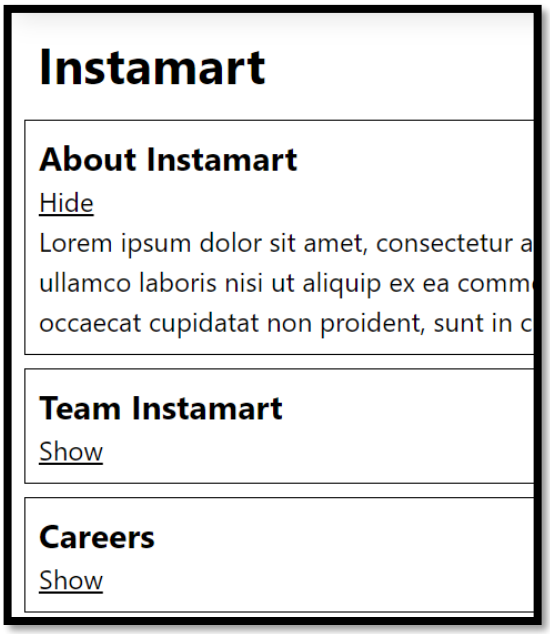
And we should also have a Hide button such that it’s onClick function does the opposite.

```
const Section = ({ title, description }) => {
    const [isVisible, setIsVisible] = useState(false);
    return (
        <div className="border border-black p-2 m-2">
            <h3 className="font-bold text-xl">{title}</h3>
            <button
                className="cursor-pointer underline"
                onClick={() => { setIsVisible(true) }}
            >Show</button>
            <button
                className="cursor-pointer underline"
                onClick={() => { setIsVisible(false) }}
            >Hide</button>
            {isVisible && <p>{description}</p>}
        </div>
    )
}
```



Now, we have both the Show and Hide buttons being shown which is undesirable. We want the Show button when description is hidden and Hide button when description is being shown. So, we conditionally render the buttons according to the isVisible state’s value.

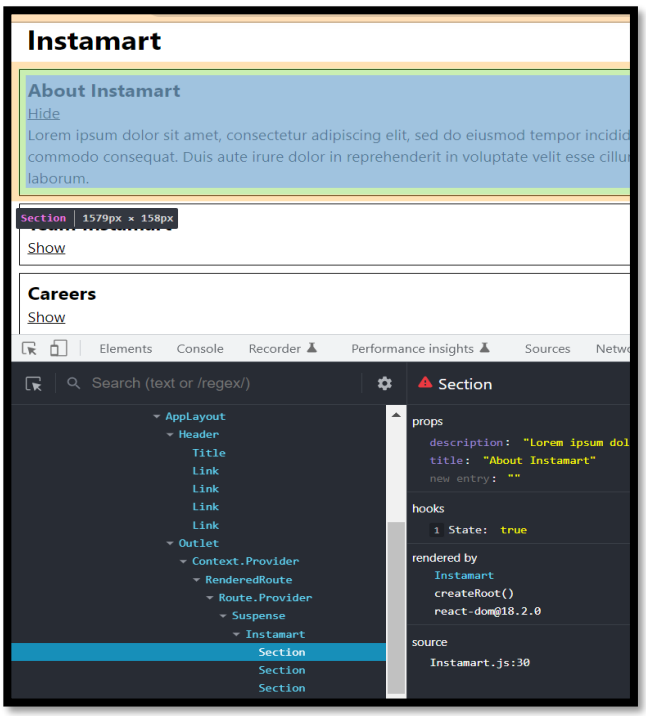
```
const Section = ({ title, description }) => {
    const [isVisible, setIsVisible] = useState(false);
    return (
        <div className="border border-black p-2 m-2">
            <h3 className="font-bold text-xl">{title}</h3>
            {isVisible ? (
                <button
                    className="cursor-pointer underline"
                    onClick={() => { setIsVisible(false) }}
                >Hide</button>
            ) : (
                <button
                    className="cursor-pointer underline"
                    onClick={() => { setIsVisible(true) }}
                >Show</button>
            )
            }

            {isVisible && <p>{description}</p>}
        </div>
    )
}
```



So, now we have built our own custom accordion component. To know about what is accordion, see these :- Link1.

Step 3 :-

Now, what we have build is a basic accordion. We want to build a collapsible accordion i.e. when one description is being shown, the others are automatically collapsed.

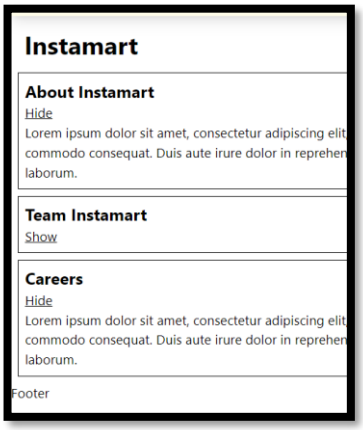


You can see in the picture beside that every section has it's own state and props. Also, the state of the first section (which actually is the isVisible useState hook) is true here because the description is being shown. As such, the states of other sections will be false. Now, we have to implement a functionality such that the state of one Section changes the state of its siblings i.e. the other Sections. This means, when we change the state of one Section from false to true, the state of the other sections should be converted to false. However, changing the state of a sibling component from any component is not possible.

What we can do is, instead of each Section component maintaining its own state, we can transfer the state to their parent component because parent has control over all its children. This concept of removing the control of a state from a child component and giving that to its parent component is called **Lifting The State Up.**

Now, the parent component (here Instamart component) will tell each of its child components (here Section component) which of their description should be visible. So, we will also remove the isVisible state variable from the Section component.

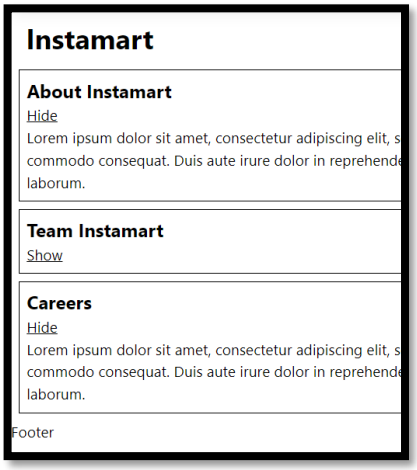
```
const Section = ({ title, description, isVisible }) => {
    return (
        <div className="border border-black p-2 m-2">
            <h3 className="font-bold text-xl">{title}</h3>
            {isVisible ? (
                <button
                    className="cursor-pointer underline"
                    onClick={() => { setIsVisible(false) }}
                >Hide</button>
            ) : (
                <button
                    className="cursor-pointer underline"
                    onClick={() => { setIsVisible(true) }}
                >Show</button>
            )
            }

            {isVisible && <p>{description}</p>}
        </div>
    )
}

const Instamart = () => {
    return (
        <div>
            <h1 className="text-3xl p-2 m-2 font-bold">Instamart</h1>
            <Section
                title={"About Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur a
                isVisible={true}
            />
            <Section
                title={"Team Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur a
                isVisible={false}
            />
            <Section
                title={"Careers"}
                description={"Lorem ipsum dolor sit amet, consectetur a
                isVisible={true}
            />
        </div>
    )
}
```



So, only the first and last section is visible. However, if you click on Show/ Hide button, it will throw an error because in the onClick function we have mentioned setIsVisible() function which is not there anymore.
(Error on clicking any Show/Hide button shown below)

Now, the isVisible props value is hardcoded. We want that to change on the click of a button. For this, we need to have a state variable. We will name it sectionConfig and it will have a default value of an object having 3 keys, one for each section to denote whether the description of that section should be visible or not.

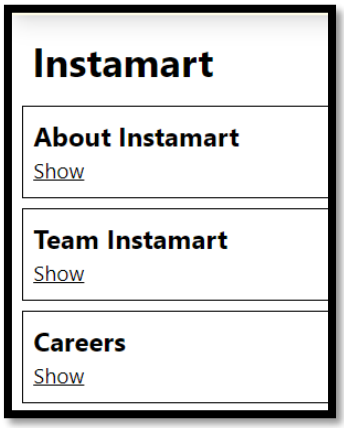
```
> const Section = ({ title, description, isVisible }) => {···
  }

  const Instamart = () => {
      const [sectionConfig, useSectionConfig] = useState({
          showAbout: true,
          showTeams: false,
          showCareers: true,
      })
      return (
          <div>
              <h1 className="text-3xl p-2 m-2 font-bold">Instamart</h1>
              <Section
                  title={"About Instamart"}
                  description={"Lorem ipsum dolor sit amet, consectetur a
                  isVisible={sectionConfig.showAbout}
              />
              <Section
                  title={"Team Instamart"}
                  description={"Lorem ipsum dolor sit amet, consectetur a
                  isVisible={sectionConfig.showTeams}
              />
              <Section
                  title={"Careers"}
                  description={"Lorem ipsum dolor sit amet, consectetur a
                  isVisible={sectionConfig.showCareers}
              />
          </div>
      )
  }
```



Now, we still have the problem of the error that comes up on clicking any Show/Hide button because of the setIsVisible() function not being there. However, we now made the visibility status of any section dependent on the value of the sectionConfig state variable. So, to change the visibility, we have to use the useSectionConfig too.

This useSectionConfig() should be somehow called on clicking the Show/Hide button. Therefore, we have to pass this function as props too to the Section components. When any Section component's Show button is clicked (for now the Hide button's functionality will not be shown), the state variable of the parent component should be changed such that only the Section whose Show button is clicked has its visibility as true and for the rest, it's false.

```
const Section = ({ title, description, isVisible, setIsVisible }) => {···
}

const Instamart = () => {
    const [sectionConfig, useSectionConfig] = useState({
        showAbout: false,
        showTeams: false,
        showCareers: false,
    })
    return (
        <div>
            <h1 className="text-3xl p-2 m-2 font-bold">Instamart</h1>
            <Section
                title={"About Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                isVisible={sectionConfig.showAbout}
                setIsVisible={() => {
                    useSectionConfig({
                        showAbout: true,
                        showTeams: false,
                        showCareers: false,
                    })
                }}
            />
            <Section
                title={"Team Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                isVisible={sectionConfig.showTeams}
                setIsVisible={() => {
                    useSectionConfig({
                        showAbout: false,
                        showTeams: true,
                        showCareers: false,
                    })
                }}
            />
            <Section
                title={"Careers"}
                description={"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                isVisible={sectionConfig.showCareers}
                setIsVisible={() => {
                    useSectionConfig({
                        showAbout: false,
                        showTeams: false,
                        showCareers: true,
                    })
                }}
            />
        </div>
    )
```



When we click the Show button of the About section, the description will be shown. When we click the button of Team section, the 1st one's description will collapse and only the 2nd one's description gets shown.

Btw, in the above code's Section component, when we are calling the setIsVisible() during the onClick function, we are still passing "true" or "false" values as arguments to the setIsVisible(), which does not have any effect on our result. But, we still have to built the functionality for Hide button too. This is where those arguments passing comes.

```jsx
const Instamart = () => {
    const [sectionConfig, useSectionConfig] = useState({
        showAbout: false,
        showTeams: false,
        showCareers: false,
    })
    return (
        <div>
            <h1 className="text-3xl p-2 m-2 font-bold">Instamart</h1>
            <Section
                title={"About Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur ad
                isVisible={sectionConfig.showAbout}
                setIsVisible={(visibility) => {
                    useSectionConfig({
                        showAbout: visibility,
                        showTeams: false,
                        showCareers: false,
                    })
                }}
            />
            <Section
                title={"Team Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur ad
                isVisible={sectionConfig.showTeams}
                setIsVisible={(visibility) => {
                    useSectionConfig({
                        showAbout: false,
                        showTeams: visibility,
                        showCareers: false,
                    })
                }}
            />
            <Section
                title={"Careers"}
                description={"Lorem ipsum dolor sit amet, consectetur ad
                isVisible={sectionConfig.showCareers}
                setIsVisible={(visibility) => {
                    useSectionConfig({
                        showAbout: false,
                        showTeams: false,
                        showCareers: visibility,
                    })
                }}
```

Here, we are mentioning a parameter named “visibility” for each Section inside the Instamart component’s setIsVisible prop. From the Section component, we will get an argument in the form of boolean value, that denotes whether to show/hide the description based on the type of button pressed. This argument’s value will be stored in the visibility variable and that variable will decide whether to show/hide the description of that particular section, keeping the key values being passed in the state object for other sections same.

Step 4 :-

Now, suppose we need to add more sections to the Instamart page. This means we have to add more keys to the object that we have taken as the default value of the sectionConfig variable. Not only that, we also have to add those keys on calling the useSectionConfig() too because we have to set the visibility of those sections to True or False too. This means we are writing redundant code.

To optimise the code, instead of having a state variable with visibility status for each section, we should keep a state variable containing the key of the section whose description should be visible (we have not specified any key for any section till now, we can also use any other identifier that can be used to identify a section). For this, I am using “visibleSection” state variable with it’s default value as “about” -> this means that by default, the About Section‘s description will be displayed.

```jsx
const Instamart = () => {
    const [visibleSection, setVisibleSection] = useState("about")
    return (
        <div>
            <h1 className="text-3xl p-2 m-2 font-bold">Instamart</h1>
            <Section
                title={"About Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusm
                isVisible={(visibleSection === "about")}
                setIsVisible={(visibility) => {
                    visibility ? setVisibleSection("about") : setVisibleSection("");
                }}
            />
            <Section
                title={"Team Instamart"}
                description={"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusm
                isVisible={(visibleSection === "teams")}
                setIsVisible={(visibility) => {
                    visibility ? setVisibleSection("teams") : setVisibleSection("");
                }}
            />
            <Section
                title={"Careers"}
                description={"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusm
                isVisible={(visibleSection === "careers")}
                setIsVisible={(visibility) => {
                    visibility ? setVisibleSection("careers") : setVisibleSection("");
                }}
            />
        </div>
    )
}
```

Using Profiler devtools :-

See the video from 1:57:05 to 2:08:09.

createContext() and useContext() Hook :-

Suppose in the App section we have a state variable named “loggedInUser” which stores the name and email addresses of the users that have logged in. Normally, when a user logs in our app, we should make an API call inside the useEffect() hook to authenticate the user and then update the details of the loggedInUser state variable with the current user details using the set function.

Now, suppose we need this user details inside the Header and the RestaurantCard component section. Normally we can only pass these details through props drilling but that is inefficient.

So, we should rather store these kind of details (which we need throughout our code) in a central storage space. Storing the details in local storage is not a good idea because updating from local storage is a time consuming operation. As such, React gives us a central storage to store these values, called **Context**.

Suppose we create an user in the App component as a state variable and give it some default value.

```
const AppLayout = () => {
    const [loggedInUser, setLoogedInUser] = useState({
        name: "Arpan Kesh",
        email: "arp@gmail.com"
    })
    return (
        <>
        <Header />
        <Outlet />
        <Footer />
    </>
)
}
```

Here the data is hardcoded, although in actual apps, we have to use useEffect() to authenticate the user, bla bla etc.

Now, we need this user throughout the code. So, instead of using this user data here, we have to create a separate context for a dummy user data. It is best to create an UserContext file in the utils folder for this purpose.

React provides a hook called **createContext**. It takes in the default value of our context and helps to make our data available throughout our app like :-

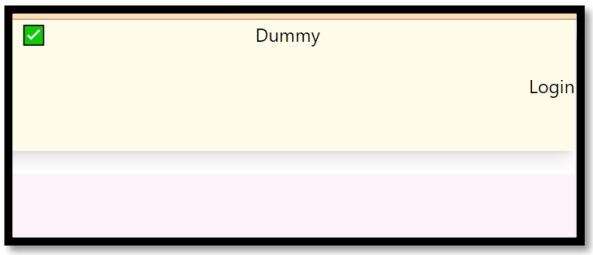
src > utils > UserContext.js > default

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

const UserContext = useContext({
    user: {
        name: "Dummy",
        email: "dummy@gmail.com"
    }
});

export default UserContext;
```

Now, we have successfully created an user data which we can globally use and exported it. We have to now access this data from the necessary components. To use the context, React provides us another hook called **useContext.**

We want to build a functionality which shows the logged in user details before the Login/Logout button of the Header Section.

```
import { useState, useContext } from "react";
import UserContext from "../utils/UserContext.js";

> const Title = () => ( ···
);

const Header = () => {
    const [isLoggedIn, setIsLoggedIn] = useState(false);

    const { user } = useContext(UserContext);

    let isOnline = useOnline();
    return (
        <div className="flex justify-between bg-pink-50 shadow-l
            <Title />
>           <div className="nav-items">···
            </div>
            <h1>{isOnline ? "✅" : "❌"}</h1>
            {user.name}
            {
                (!isLoggedIn) ?
                    (<button onClick={() => { setIsLoggedIn(true)
                    : (<button onClick={() => { setIsLoggedIn(fals
                }
        </div>
    )
};
```



We can also use these user details in our Footer section. (Check the code in Footer component for the changes)

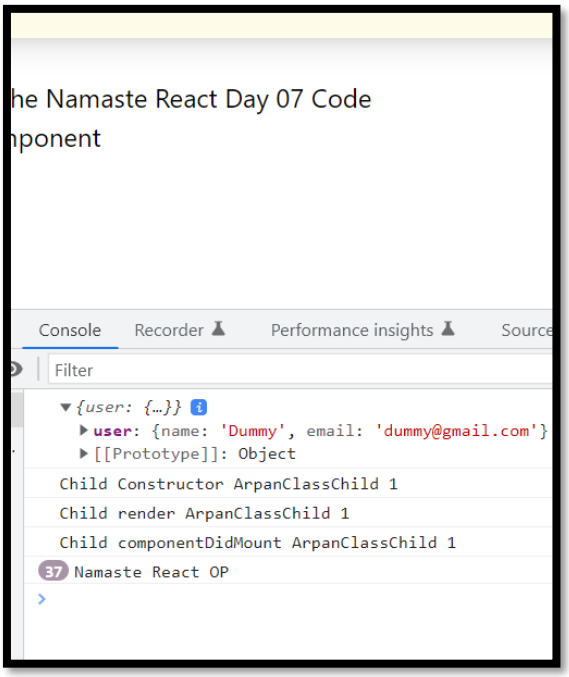
Using useContext() in a Class Based Component :-

We know that in CBC, we do not have any concept of hook. So, in CBC we first import the context that was created (here UserContext) and use it as a component. This component can accept a JSX piece of code, within which we can write a function where the function parameter is actually the context data.

In our app, we have used CBC in the About component, so will check this functionality there and we will try to print the user object in our console.

src > components > About.js > About > render

```
import Profile from "./ProfileClass";
import { Component } from "react";
import UserContext from "../utils/UserContext";

class About extends Component {
    constructor(props) {
    }

    componentDidMount() {
    }

    render() {
        // console.log("Parent render");

        return (
            <div>
                <h1>About Us Page</h1>
                <UserContext.Consumer>
                    {(value) => {
                        console.log(value)
                    }}
                </UserContext.Consumer>
                <p>This is a part of the Namaste React Day 07 Code</p>
                <Profile name="ArpanClassChild 1" />
            </div>
        )
    }
}
```

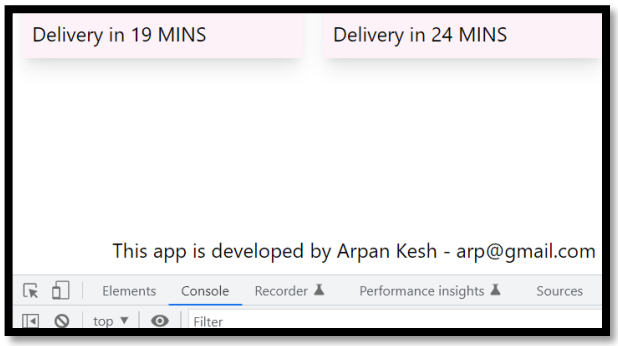


Using a .Provider to override default context value :-

Till now we are always using the dummy user data . But in an actual app, we first need to authenticate the user and then override the dummy data with the logged in user details. Remember that we have already hardcorded a loggedInUser data inside the App component (because we will making an API call to authenticate the user from that component's useEffect() hook).

Like we called the UserContext as a component in the CBC i.e. the About Component and used UserContext.Consumer to get the context data, here also we will a very similar **UserContext.Provider** to provide values to the context data.

```
const AppLayout = () => {
    const [loggedInUser, setLoogedInUser] = useState({
        name: "Arpan Kesh",
        email: "arp@gmail.com"
    })
    return (
        <UserContext.Provider value={{ user: loggedInUser }}>
            <Header />
            <Outlet />
            <Footer />
        </UserContext.Provider>
    )
}
```

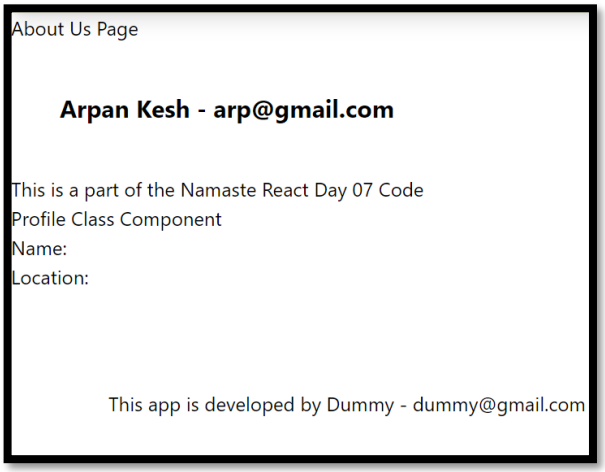


Now, here we have mentioned all the components i.e. Header, Outlet and Footer inside the Provider tag. This helps to update the context data for all the components. Had any of the component be used outside the provider, that component would have been served the default value of the context.

To demonstrate this, we will take the Footer component out of the Provider and the expected results is that it will show the Dummy user data.

In the below images, you can also see that the context data of the About component (which was inside the Outlet component) was updated, whereas the Footer's context data is still the default data.

```jsx
const AppLayout = () => {
    const [loggedInUser, setLoogedInUser] = useState({
        name: "Arpan Kesh",
        email: "arp@gmail.com"
    })
    return (
        <>
            <UserContext.Provider value={{ user: loggedInUser }}>
                <Header />
                <Outlet />
            </UserContext.Provider>
            <Footer />
        </>
    )
}
```



Changing the context using the set function of a state variable:-

Suppose we will have an input box beside the Search button with a value as the user name. Now, when we change that value, the user details should also be changed. (This is a useless functionality, but good for conceptualisation).
We know that the user detail (not the dummy one) is a state variable and we can only change a state variable with a set function. Also, we have to make this set function accessible throughout our app. So to do this, we need to pass this set function in our value props too (of the UserContext.Provider component of the App component).
Then in the Body component, we will make an input box with the value as the context data's username and we will also attach a onChange function to change the user details as per the input box's value.

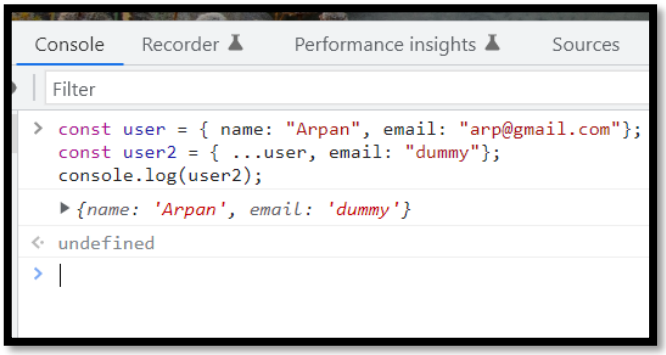
```jsx
const AppLayout = () => {
    const [loggedInUser, setLoogedInUser] = useState({
        name: "Arpan Kesh",
        email: "arp@gmail.com"
    })
    return (
        <UserContext.Provider value={
            {
                user: loggedInUser,
                setUser: setLoogedInUser
            }}>
            <Header />
            <Outlet />
            <Footer />
        </UserContext.Provider>
    )
}
```

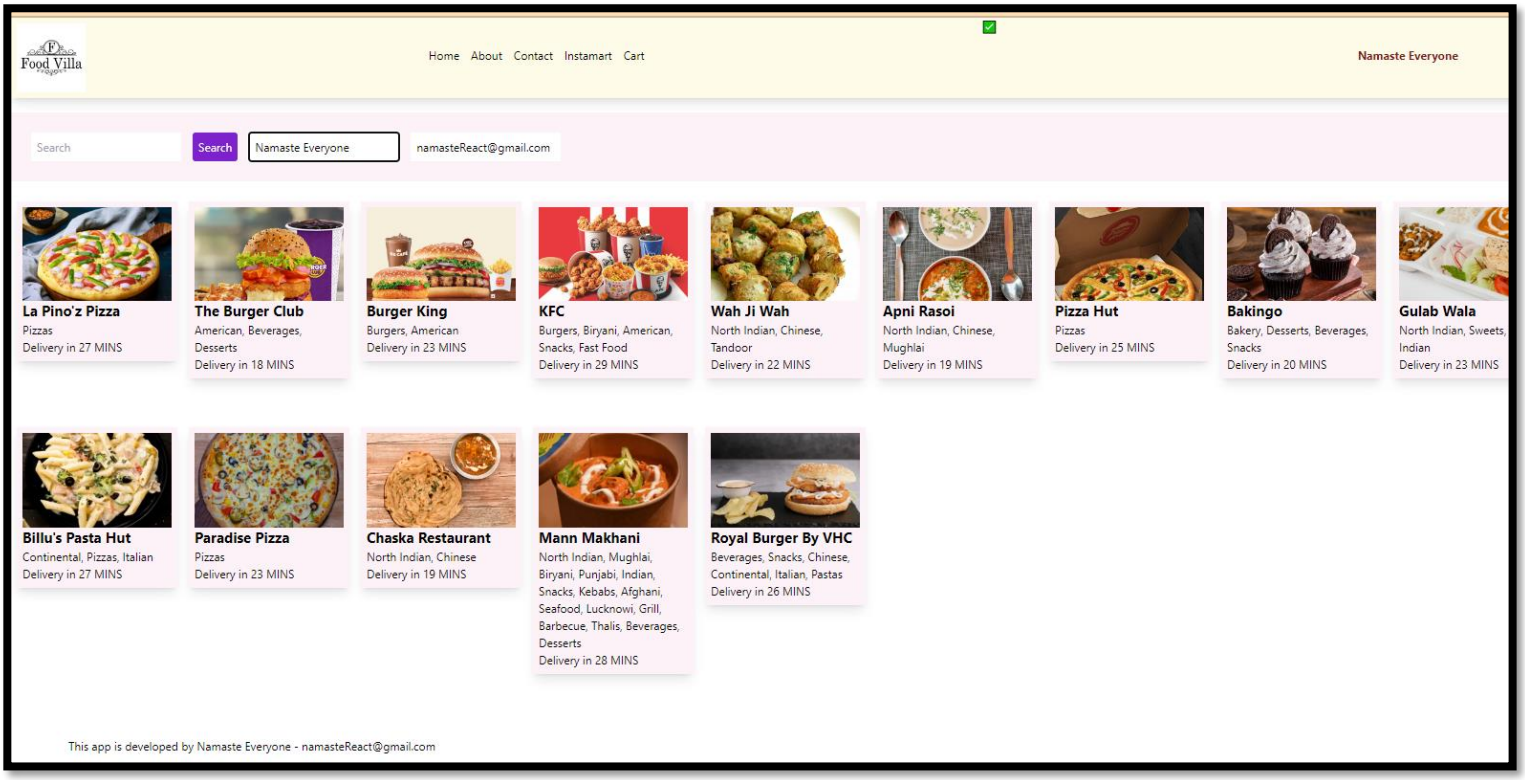
```jsx
    return (allRestaurants.length === 0) ? (
        <Shimmer />
    ) :
        (
            <>
                <div className="p-5 bg-pink-50 my-5">
                    <input ...
                    </input>
                    <button ...
                    >Search</button>
                    <input value={user.name} onChange={
                        (e) => {
                            setUser({
                                name: e.target.value,
                                email: `${e.target.value}@gmail.com`
                            })
                        }
                    }></input>
                </div>
                <div className="flex flex-wrap">...
                </div>
            </>
        )
};
```

We can also make another input box for email too instead of doing the above one like :-

```jsx
return (allRestaurants.length === 0) ? (
    <Shimmer />
) :
    (
        <>
            <div className="p-5 bg-pink-50 my-5">
                <input ...
                </input>
                <button ...
                >Search</button>
                <input value={user.name} className="p-2 m-2" onChange={
                    (e) => {
                    setUser({
                        ... user,
                        name: e.target.value,
                    })
                    }
                }></input>
                <input value={user.email} className="p-2 m-2" onChange={
                    (e) => {
                    setUser({
                        ... user,
                        email: e.target.value,
                    })
                    }
                }></input>
            </div>
            <div className="flex flex-wrap"> ...
            </div>
        </>
```

If you do not understand how the spread operator is working in the code, see the below example :-





See the video from 3:00:17 to 3:12:00 know how ReactRouterDom is actually using Context behind the scenes.