Chapter 11 (Data is the new Oil)

Topics to be discussed

1. Props Drilling

2. Lifting the state up

3. Build your own accordion

4. Data is the new oil

5. Create context and update context

6. React dev tools

7. Component and profiler

In react the whole UI layer (Look and feel on screen) is powered by Data Layer (props and states). We display the data to different UI components. We manage the UI with states and props.

In our application UI, we used restaurant card but the data associated with restaurant card is stored somewhere. This data is being handled by states and props.

Difference between state and props?

State: Suppose if we have a container or a component, and we want a variable which has a scope just with in this container. In that case we can use state variable. It is the local variable for the component.

Prop: if we have to pass data from one component to another component we use Prop.

Props Drilling:

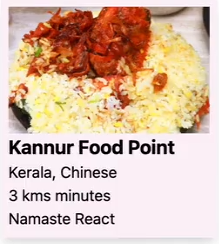
* When we pass data from parent component to child component and then to its child and then to its child and so on to the target child, it’s called props drilling.
* In props drilling we pass data/props through a chain of components. We are drilling all the components to get access to the props in the target component.
* If we have a hierarchy of 2 or 3 level its ok to use props drilling.

Cons:

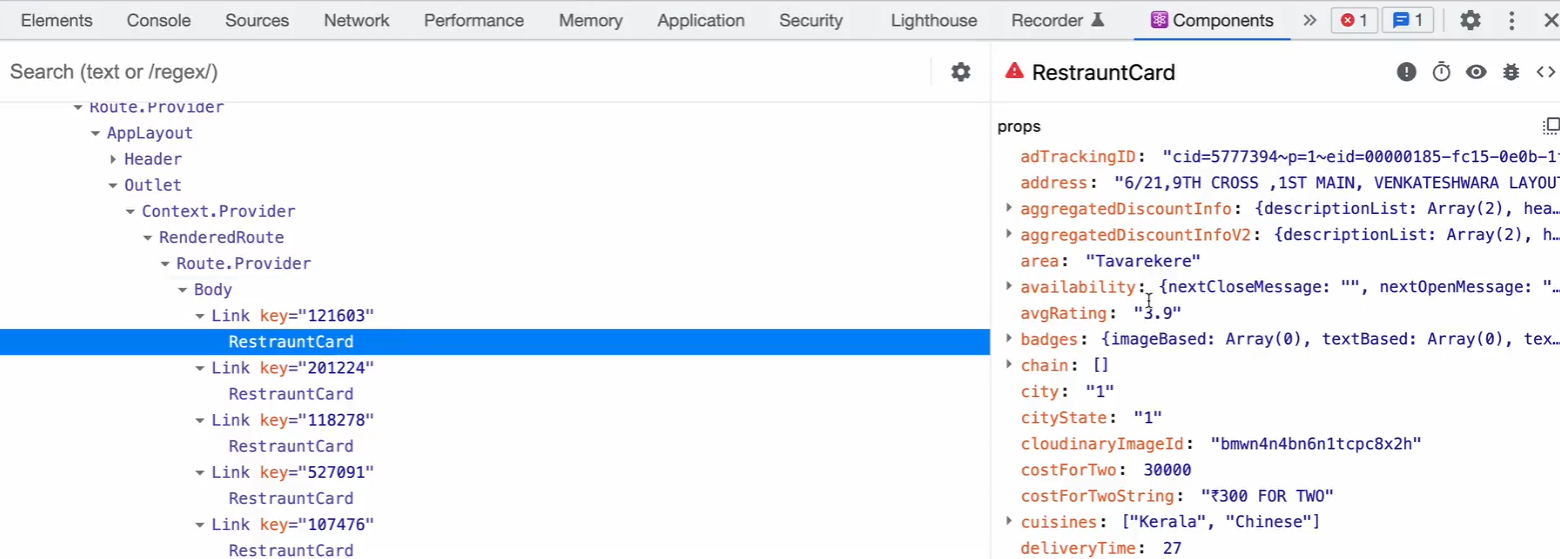
* It does not make sense to use >2 or >3 level props drilling. Because we have to pass props in some many components resulting so many unnecessary changes across the files. Redux helps to solve this problem. (Redux – we will discuss this in chapter 12)
* Props drilling rerenders all the components using the props in case there is a prop change.
* Props will not be used in other components in the components chain except the target component. React guideline says never use a prop in a component if the component does not use it.

React Developer Tool:

This tool provides component and profiler which helps us debugging the data layer faster.



The above card component is being powered by data layer. This card is receiving props which is represented below by component React developer Tool.



Assume Left hand side (V DOM) is a UI layer and right-hand side is a data layer which powers the UI.

Use cases:

* In a large-scale application, it’s very difficult to remember which component has what props what states and what hooks. By the help of React developer tool we get to know about the states props and hooks being used specific to specific component.
* This tells where exactly the data is coming from.
* This enables debugging process faster.

usually data flows from parent to child in react. This is one way but how do we pass the data from child to parent?

* Local storage (Not a good way)
* We can build our custom hooks
* Lifting the state up approach.

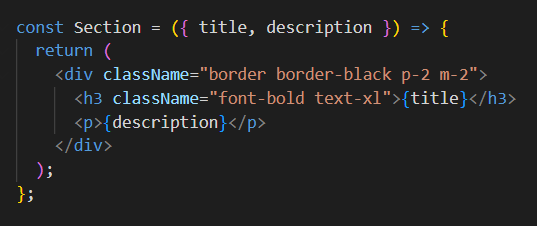
Let’s build our own component Instarmart



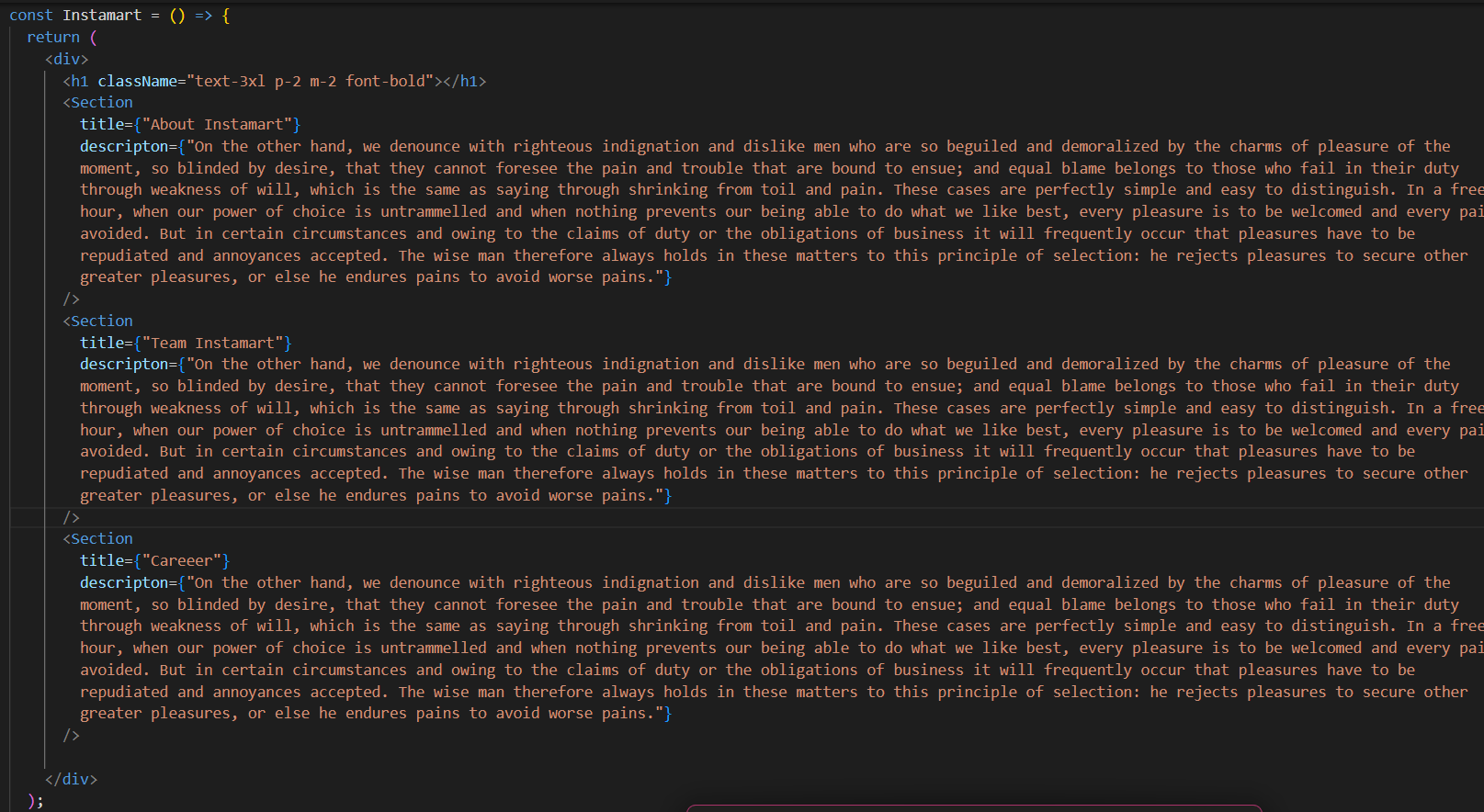
The instarmart component might have some other components inside of it. At the End of the day these components are sections for instarmart page. In order to use these sections, we have to create these components explicitly.

But let’s create a reusable component section and use it as sections inside instarmart component.

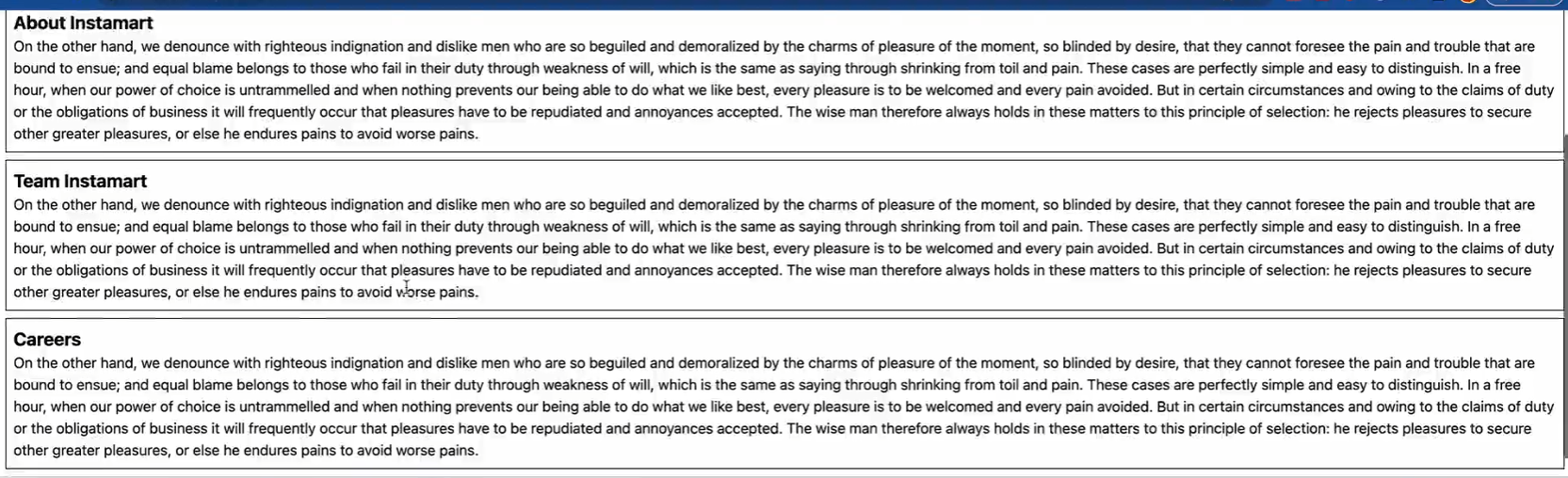
Section component:



Re-using Section component in instamart page:



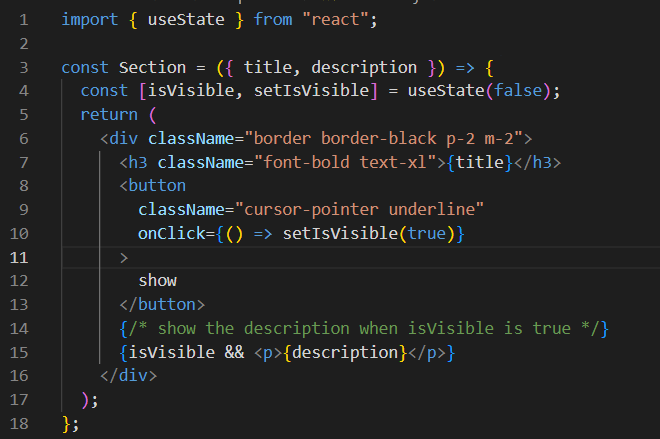
UI:



Let’s implement show & hide description functionality to each section.

To set description visibility to on and off, we have to create a state variable let’s say isVisible.

Implement Show functionality:



In above code snippet,

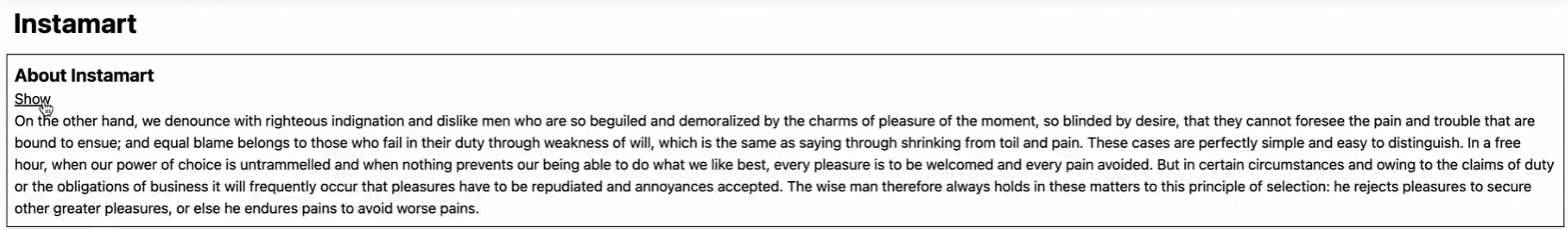
* In line #4, We have created a state variable isVisible and initialised this variable to false.
* In line #15, We put a condition that if isVisible is true than display the description.
* In line #8, we have set isVisible variable to true on a button click.

User interaction:

* When instamart page loads user can’t see the description below show button, because at this moment isVisible state is initially set to false and we don’t render description when isVisible is false. We render description when isVisible is true.

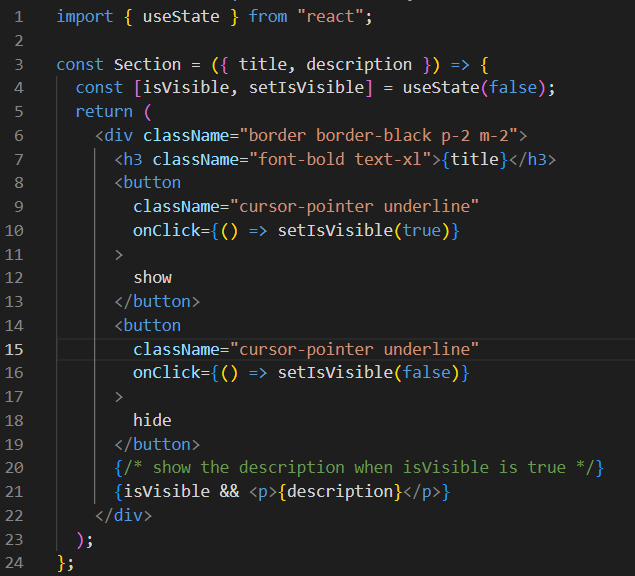


* Now when user clicks on show button we are changing isVisible state to true. That’s why user can see the description down below.



Note: This functionality works in all the sections inside instamart page.

Implement Hide functionality:

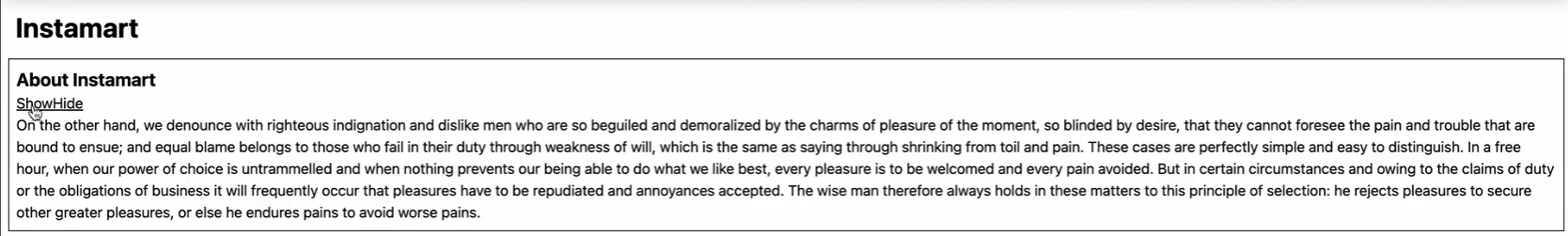


In above code snippet

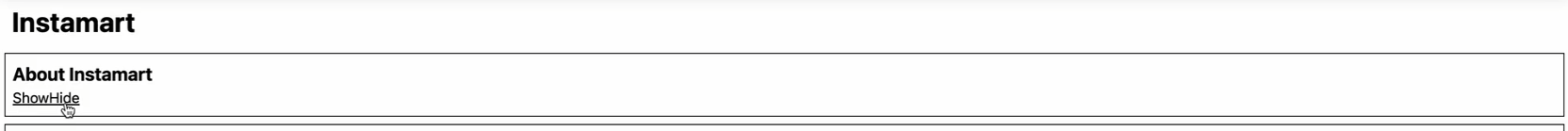
* In line #14 we have added a hide button and changed the state isVisible to false on button click.
* In line #21 we are only displaying the description when isVisible is true. In our case description won’t be shown to the UI when user clicks on hide button. In fact, this will hide the already shown description if there is.

User Interaction

* When user clicks on show it displays the description



* When user clicks on hide button it hides the description



Note: This functionality works in all the sections inside instarmart page.

Implement Toggle functionality:

Requirement:

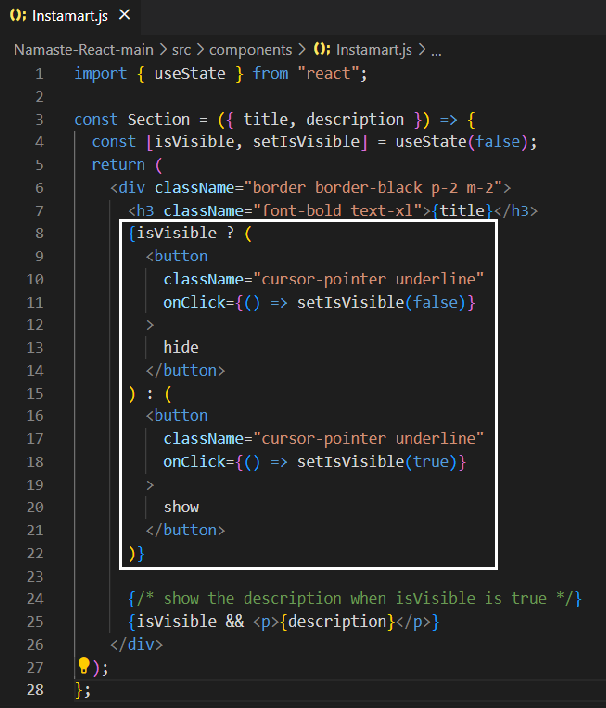
At this moment

* If user clicks on show button then the description is shown. When description is being shown we don’t want our show button to be displayed in the UI. We want our Hide button to be displayed in place of show.
* If user clicks on hide button then the description is hidden. When description is being hidden we don’t want our hide button to be displayed in the UI. We want our show button to be displayed in place of hide.
* This is toggle functionality.

Toggle Approach:

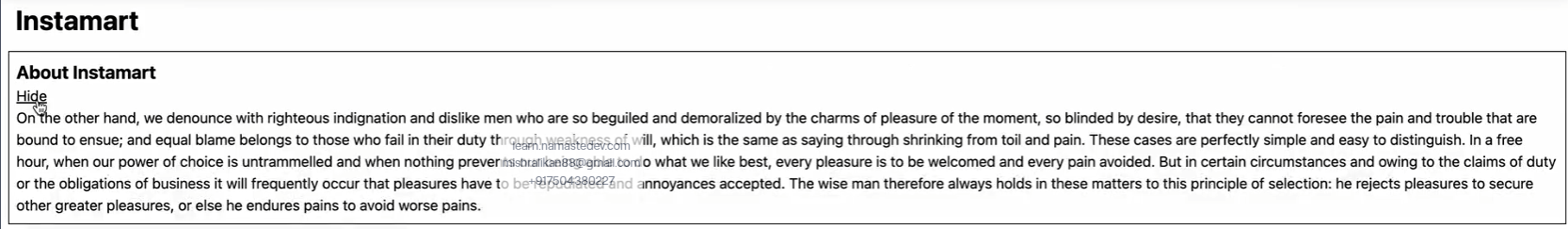
Render the button conditionally on the UI.

* If isVisible is true display the hide button in the UI. Meaning when the description is shown then display the Hide button in place of show.
* If isVisible is false display the show button in the UI. Meaning when the description is hidden then display the show button in place of hide.
* We can achieve this by conditional rendering.



If isVisible is true display hide unless display show button.

* When user clicked on show button hide button is displayed in place of show and the description is shown below.



* When user clicked on hide button show button is displayed in place of hide and the description is hidden below.



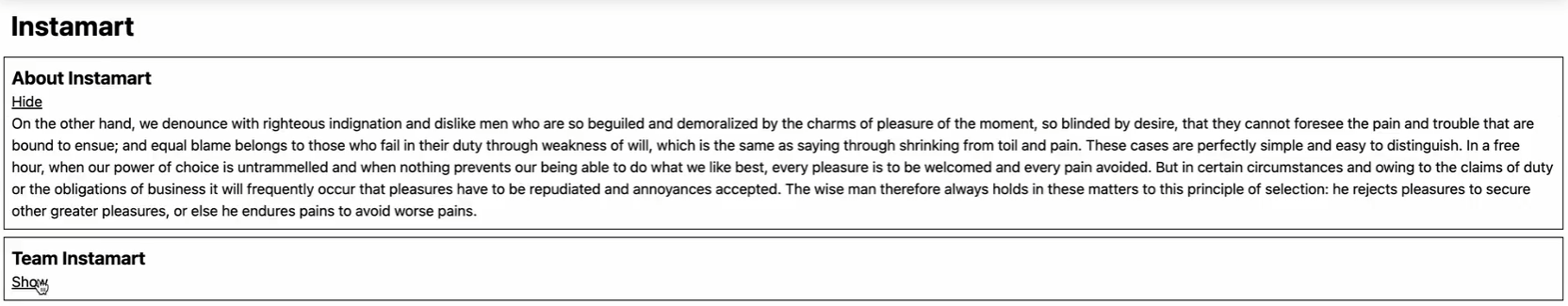
Note: This functionality works in all the sections inside instarmart page.

Problem with our Accordion:

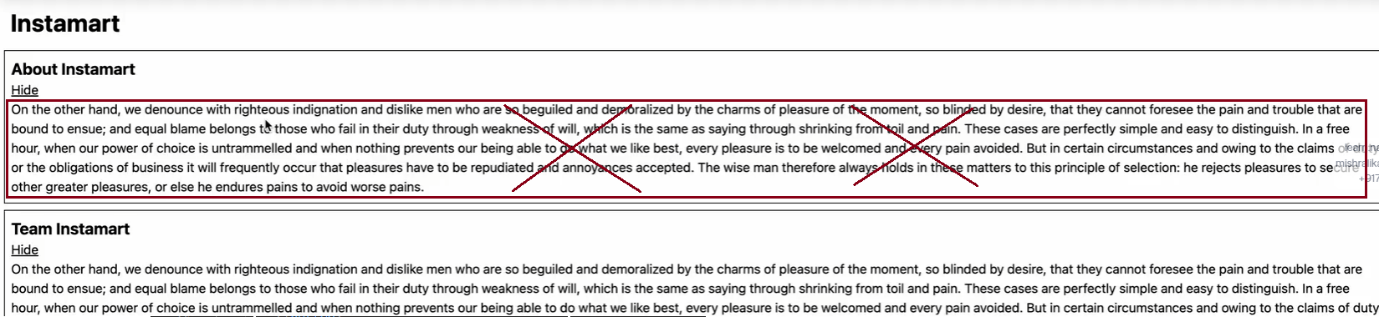
We have built our own accordion. But we have a problem. When I click on show button in one section, it should show the description of that specific section which in fact is working. But it should also hide the description of other sections as well irrespective of whether those sections descriptions are shown or not. This feature is not working.

Scenario creation:

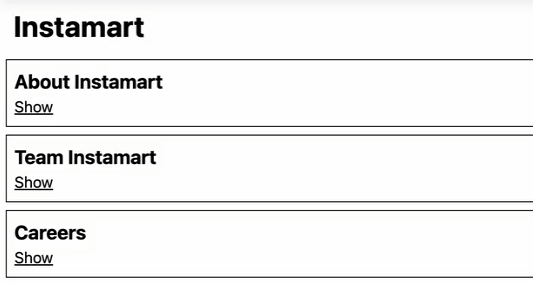
In below figure we have our section 1 description displayed.

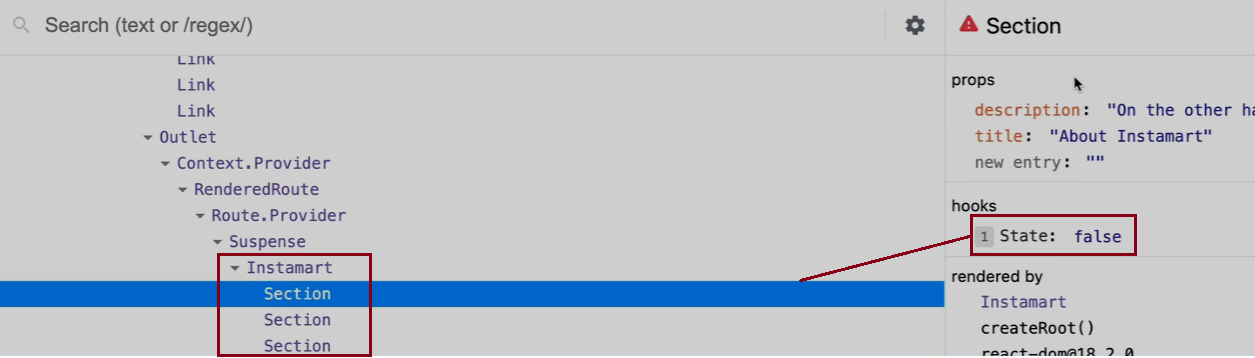


The moment if I click on show button of section 2, it should hide the description of section 1 and display the description of section 2. But it’s not hiding the description of section 1.



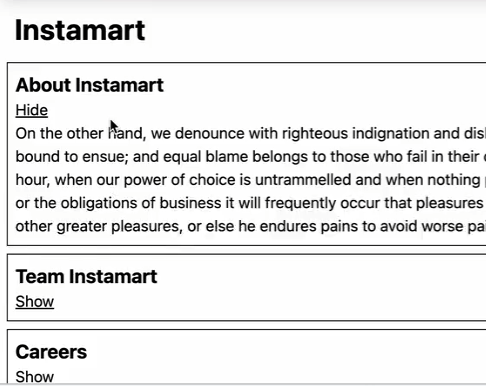
If we go back to the data layer we can see all section components have their own states and props.



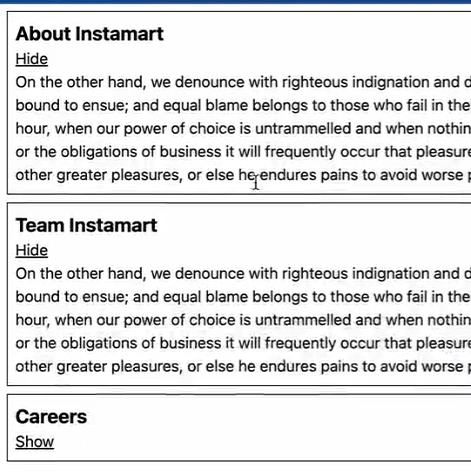


Current behaviour:

* If user clicks on section 1 show button, the state for this section gets changed to true and description is displayed. Meanwhile state for second and third section component remain unchanged which is false and description for them will not be displayed.



* If user clicks on section 2 show button, the state for this section gets changed to true and description gets displayed. Meanwhile state for third section component is false and description for it will not be displayed. At this moment state for section 1 is retained from previous step which is and description for section 1 is displayed.



* If user clicks on section 3 show button, the state for this section gets changed to true and description for it is displayed. Along the same, state for section 1 and section 2 retain their old values which is true (we have set this in previous two steps) and description for them also get displayed.

All sections description is displayed

Conclusion of Current behaviour -

* All these sections don’t share state among themselves. They have their own states and props.
* In our case three copies of state variable isVisible exist for three different sections. changing/updating one state variable will not impact another components state variable.

Expected behaviour-

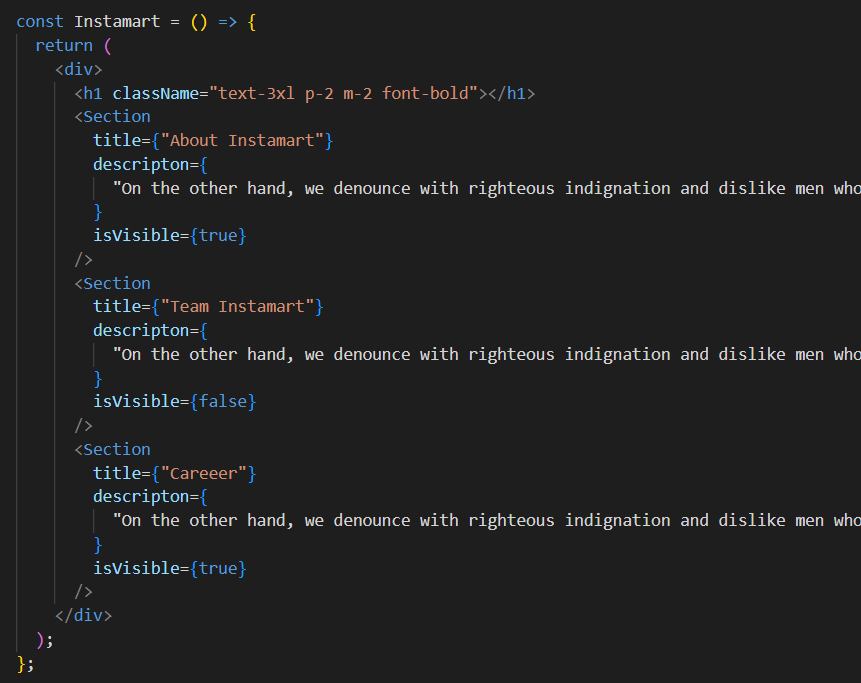
* If I click show button on any of the section component, only description associated with that section component should be displayed while other section components description should be hidden.
* We can achieve this behaviour, once we get access to other section components state and modify them while we are in source section component.
* I want to share my state across all my children/siblings/sections components, so as to achieve a proper accordion functionality.

Conclusion of Expected behaviour -

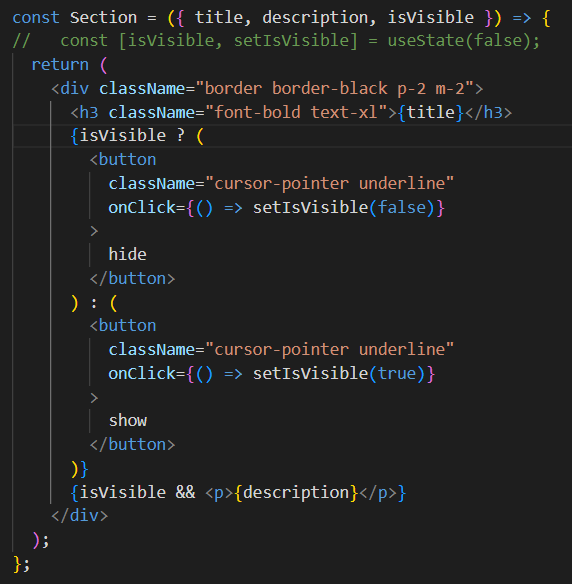
* We have to have one state variable isVisible which should be shared across all these section components.
* We cannot directly modify the state of a sibling from another sibling component.
* Instead of children maintaining their state, we offload their responsibility to their parent so that parent will have full control over all of its children. Now parent will decide what has to be displayed when. This concept is known as lifting the state Up.
* Here we are lifting state of section components (child) to instamart component(parent).

How do we do it?

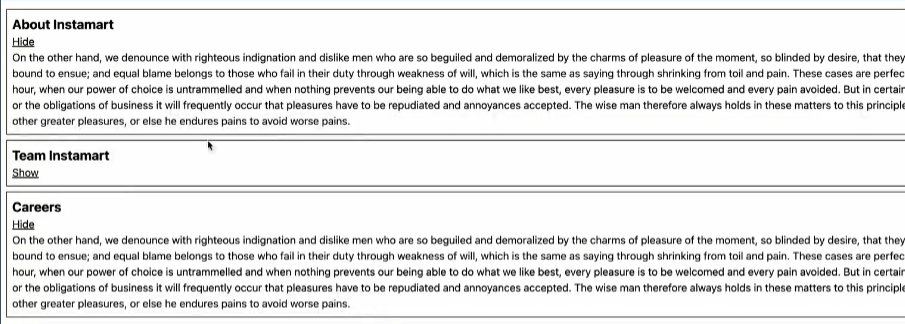
I want my instamart/parent component to control its children like which section to show or not on UI. I have passed a prop isVisible to all the section components wrapped inside parent Instamart component.



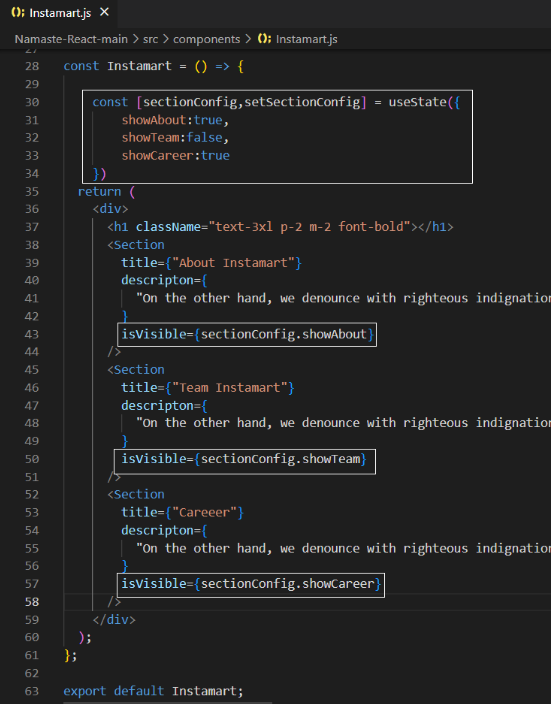
I have taken the power from the children I.e. to display their description on UI. I have deleted the state variable maintained by children components.

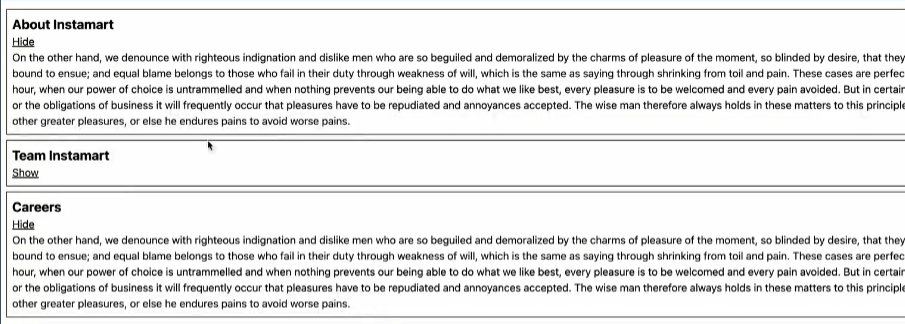


Now my parent has control over what needs to show what needs to hide. In our case parent component is controlling the description to be shown for section 1 and section 3 components.



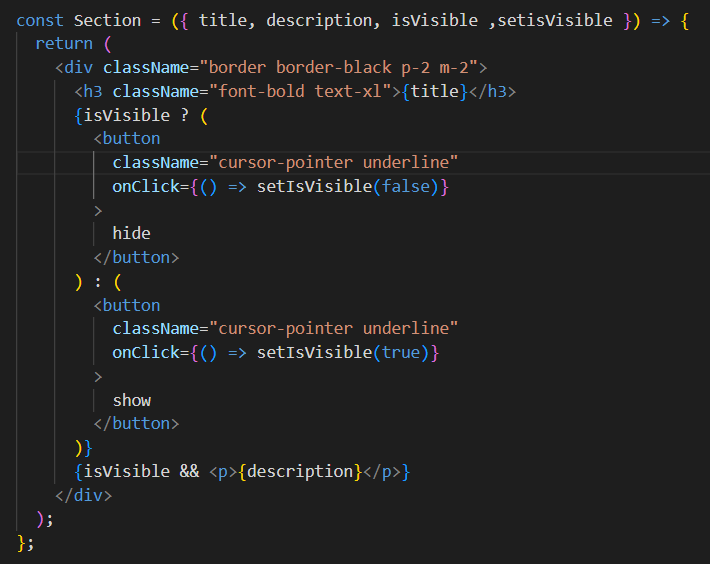
We are hardcoding everything so far. I want my parent to maintain the state of their children now.I will have to create a state for section in parent instamart component

1

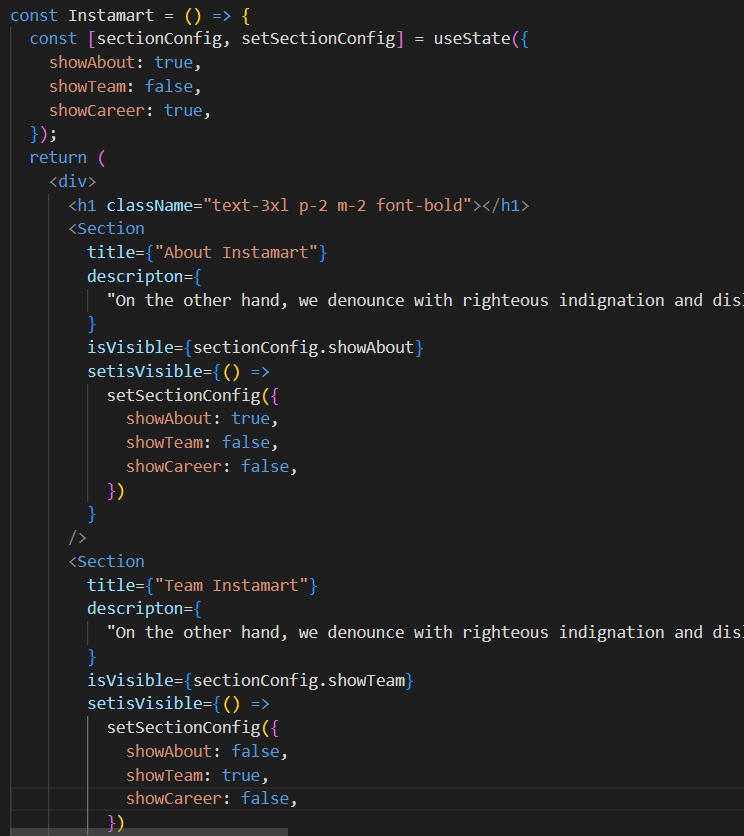


Now the parent component is controlling all of the section components.

Proper accordion functionality:



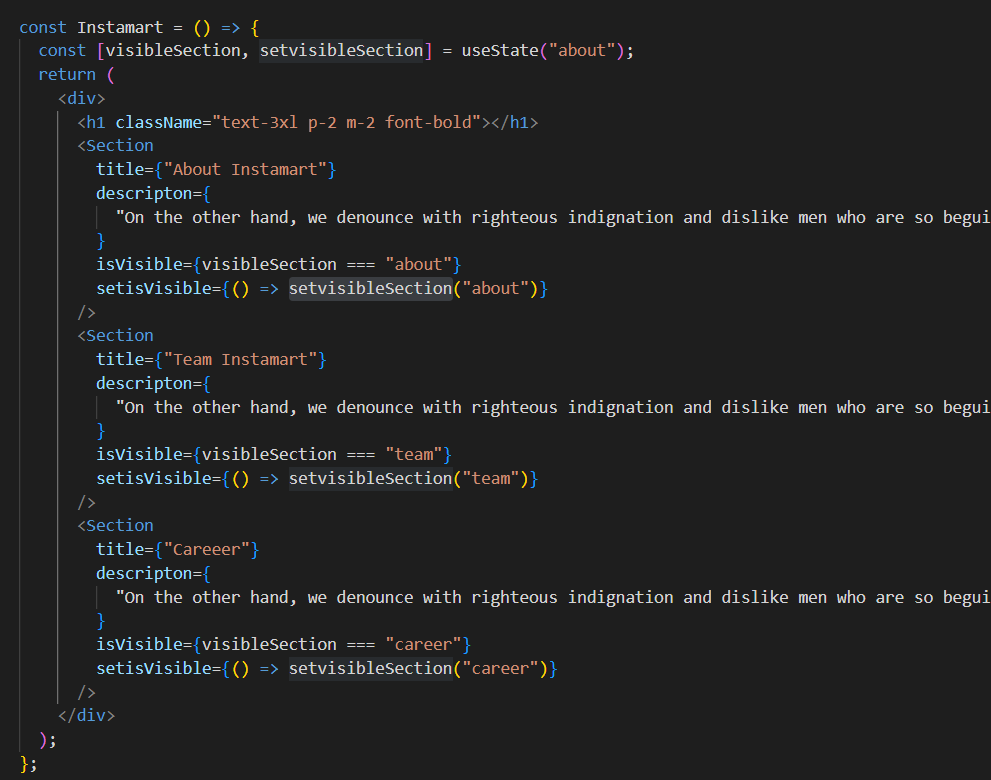
If user clicks on show button of the section component, we want that section description to be displayed and other section description should be collapsed which we are doing inside parent component’s sections call.



We are passing setisVisible from parent component Instamart to child component section. In any of the section component when we click on show button its calling setisVisible () which in turn sets the setSectionConfig state of parent Instamart component which is display the current section when show button of that section is clicked and hide the other two sections. This is a typical accordion behaviour.

However, the code we wrote so far is not optimal. What if we have more sections. we need to hardcode data for those sections as well, we need to change the description display configuration for each section. This will make the code look messy.

Better approach: (Use keys)



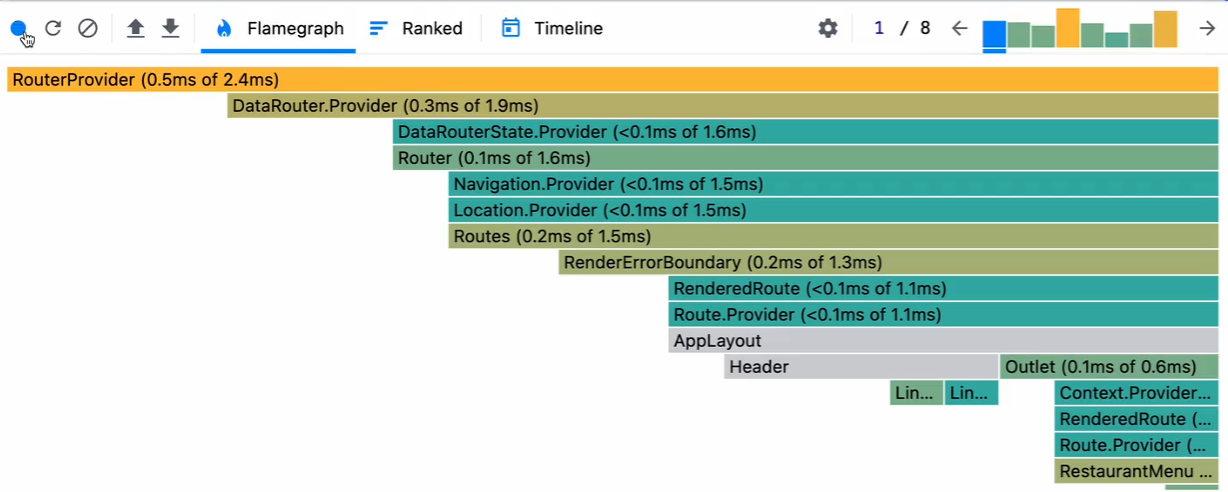
The functionality will work as it is but hide functionality won’t work **(HW)**

Profiler

When we build large scale application with so many components in it, it becomes very tough to debug. Profiling records the whole user journey. whatever operation user has performed on the UI, Profiler shows you how the page was rendered.

The react dev tools Profiler tracks user actions on UI and displays the entire user journey through a Flame graph.

Profiler provides a on off switch to turn on and off the recording. To see the graph, we have to start the recording, play with the UI and then stop the recording manually.

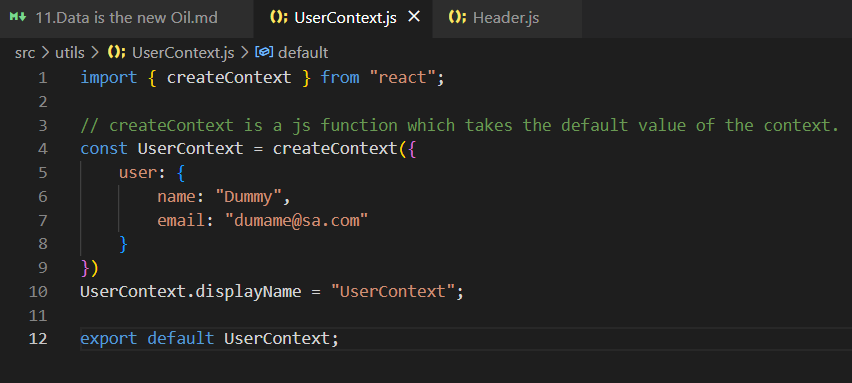


Flame graph represents how long does it take for a component to load on UI. Flame graph shows the time taken by Layout effect and passive effects to render on UI.

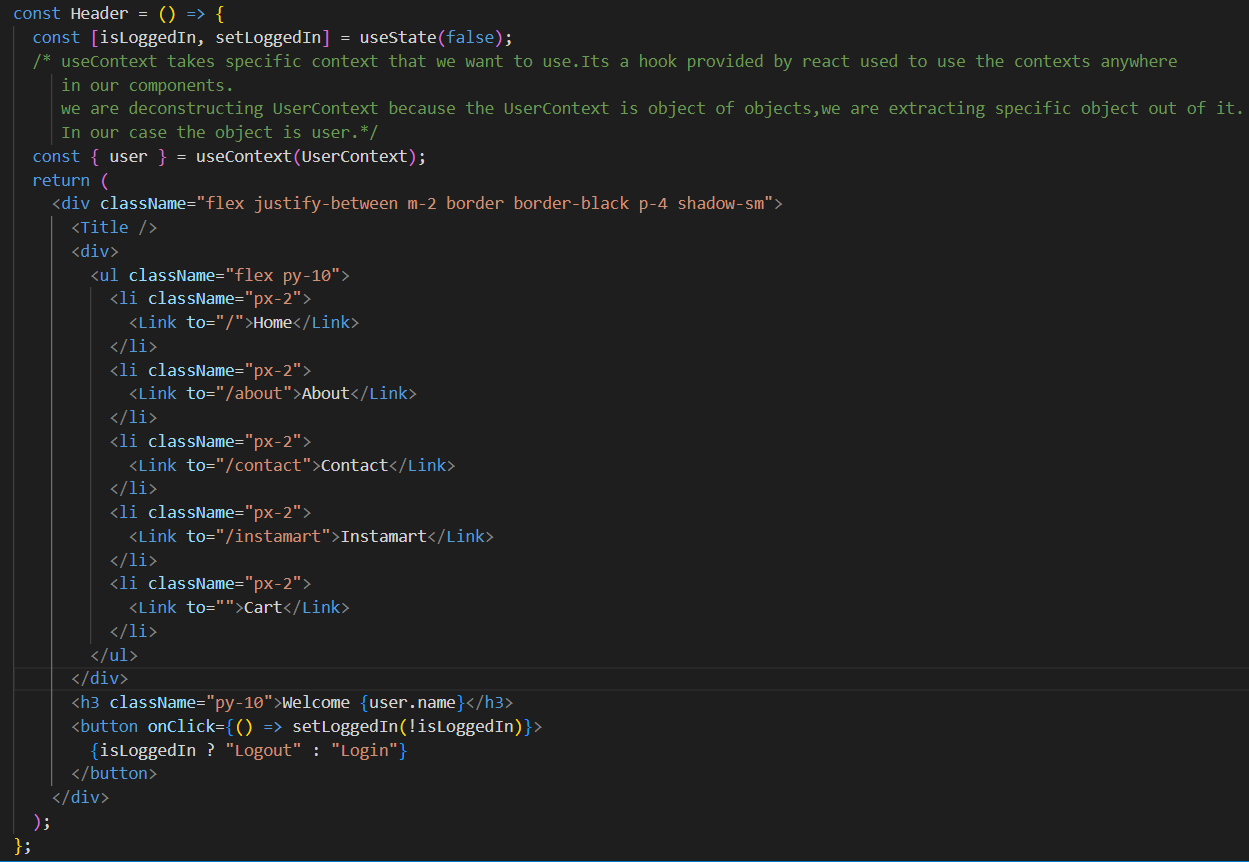
React Context:

* Whenever we need same data to be accessible anywhere in our application components, we have to store this in a central space, react gives us access to this central place which is known as react context.
* Context can’t be a local storage as it can’t be reliable and updating the local storage is heavy operation.
* Any Component can use this Context data.
* No need of props drilling.

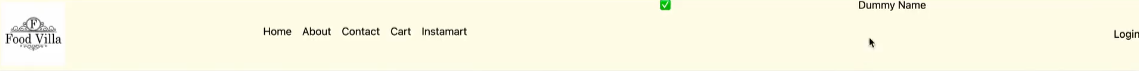
Creating User Context:



Using User Context:



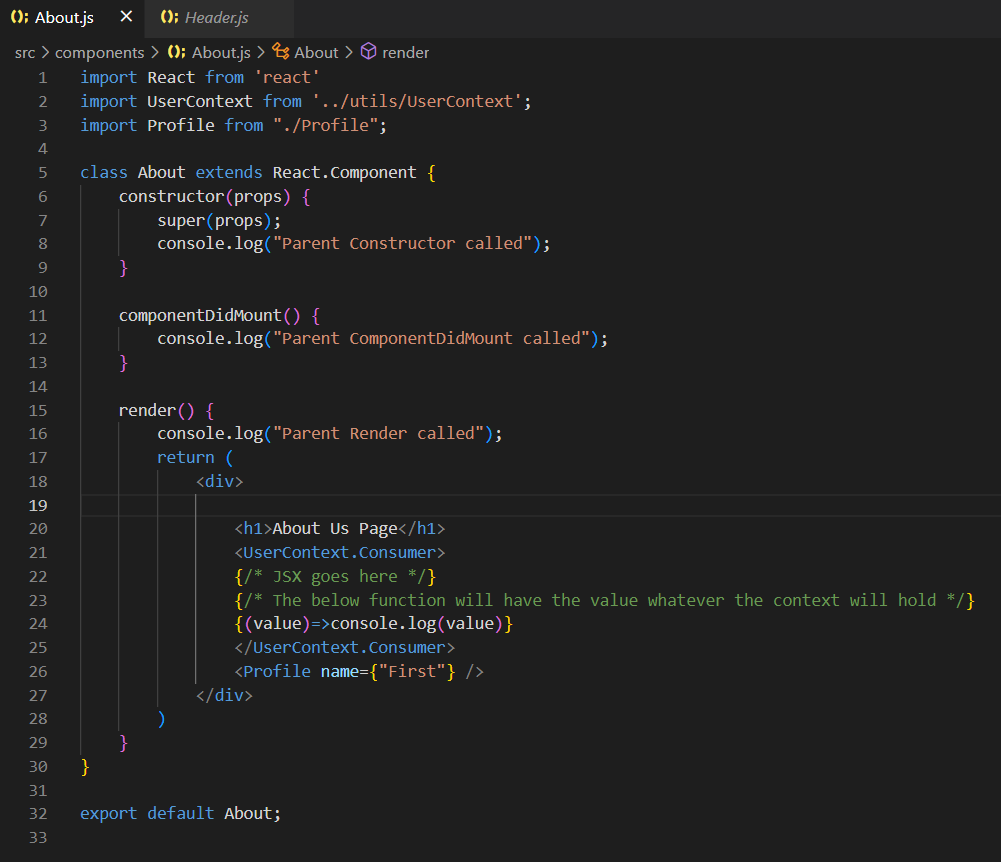
UI Screen:

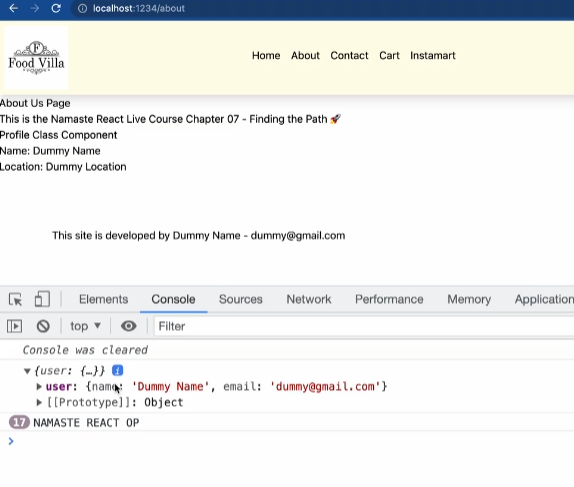


Note:

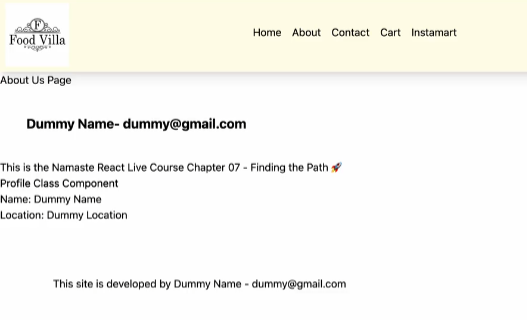
Props and states are tight to a component. context is not specific to any component. We can have context in a central space. Context is a central data for all of our components.

Consuming user Context in class-based Component:



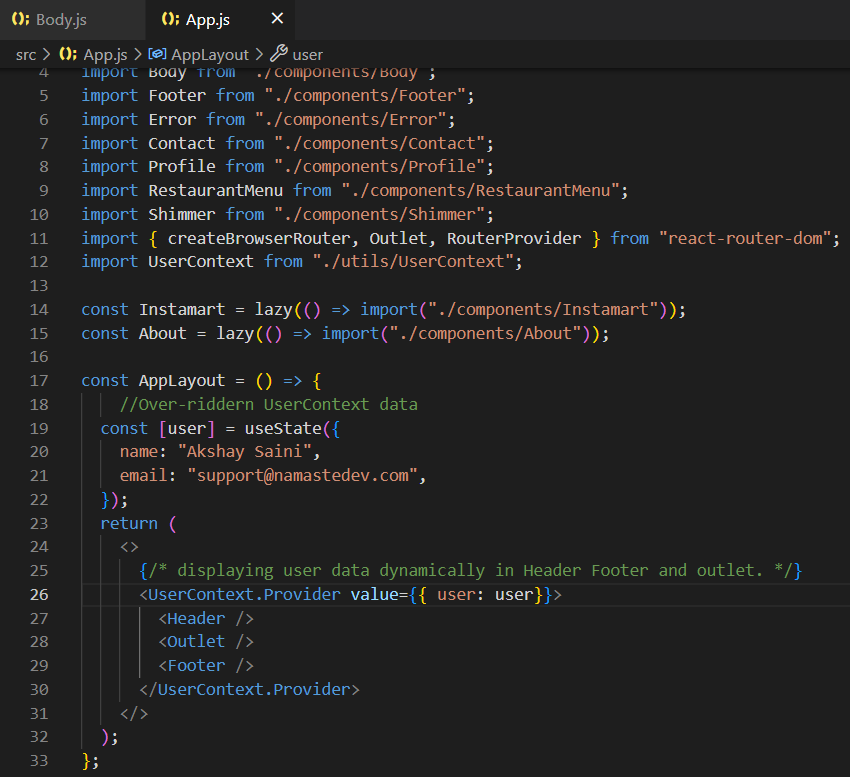


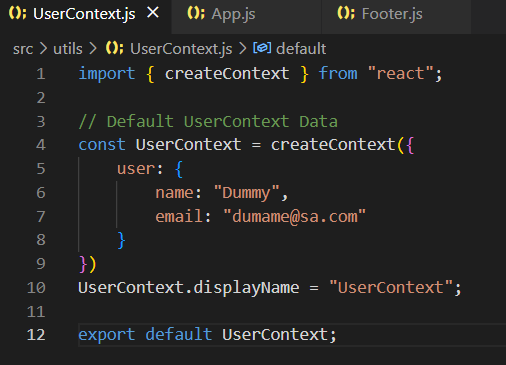




Rendering context data dynamically:

To render the context data dynamically react provides UserContext.Provider which overrides the default UserContext data.



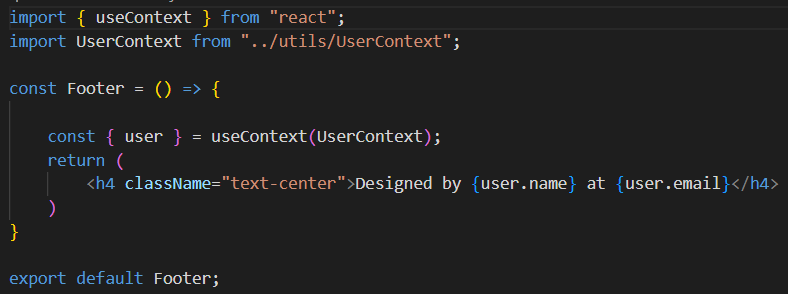


The components wrapped inside UserContext.Provider can use the context data dynamically.

The components outside UserContext.Provider cannot use the overridden context data. It can use the default context data.

Earlier these components were using default context data. But now they are using UserContext Provider data instead of default context data. For example:

Footer.js

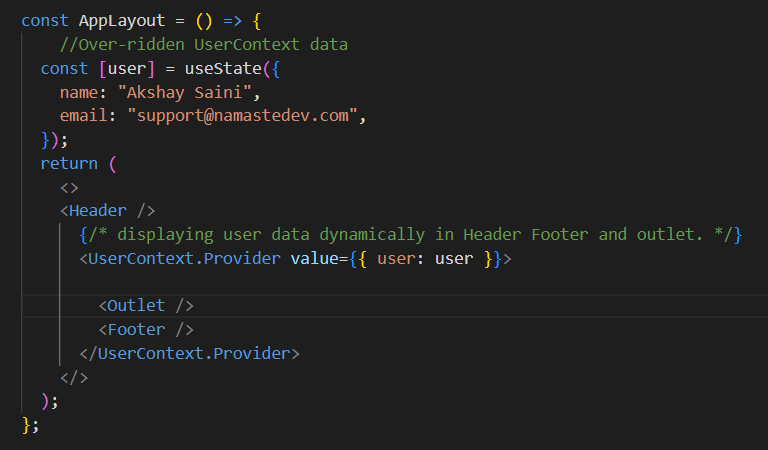


Footer UI



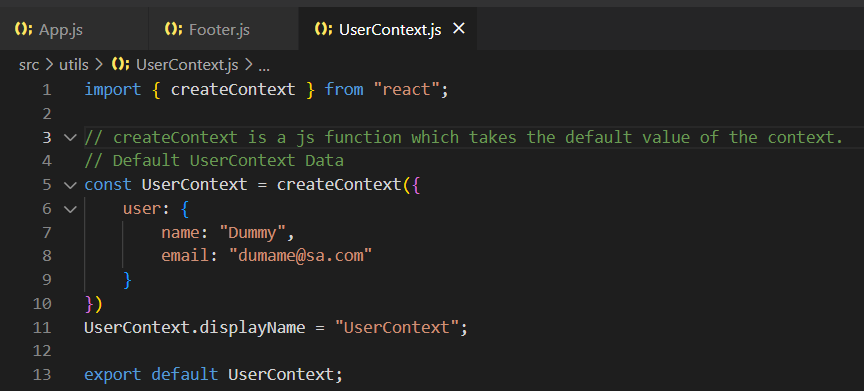
Here footer component is using dynamic context data used in appLayout component in app.js which over rides the default context data in Usercontext.js

Moving the header component outside of the User Context tag of app.js.





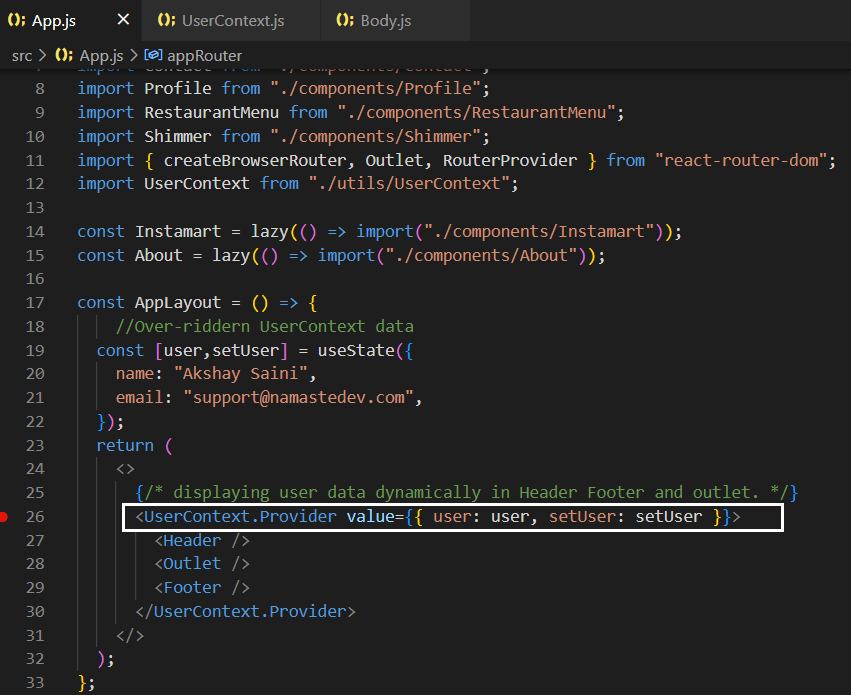
The header component is not wrapped outside context provider. In header component context data (Dummy Name/default context data) will be coming from UserContext.js.

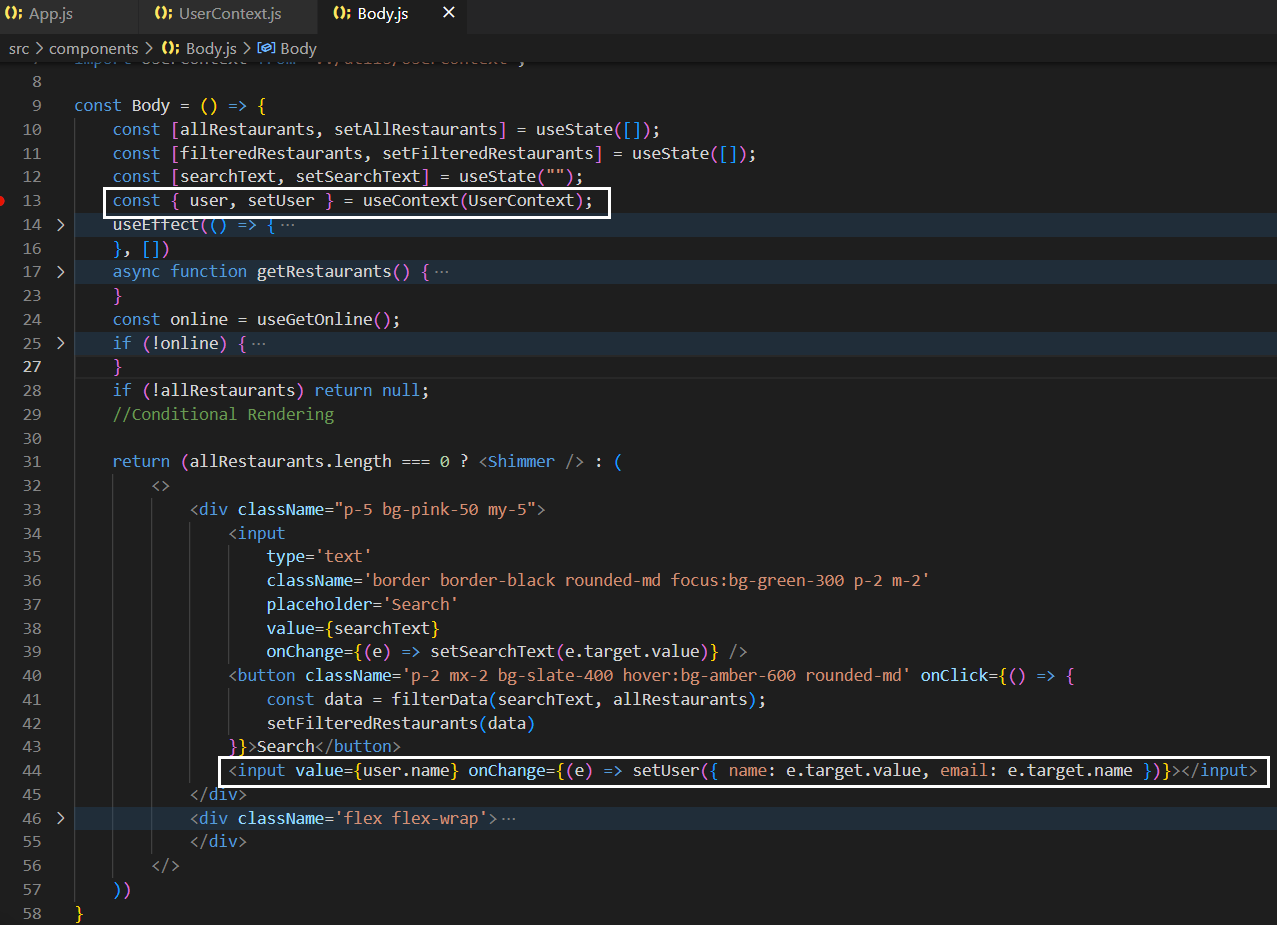


I can modify my context for smaller portion of my app. I can pass different data into different portion of my app.

Two Way Context Data binding:

Now let’s modify the Context by creating a text Input box near the search button inside body component. If I type anything in this textbox I want my user context data to be modified.

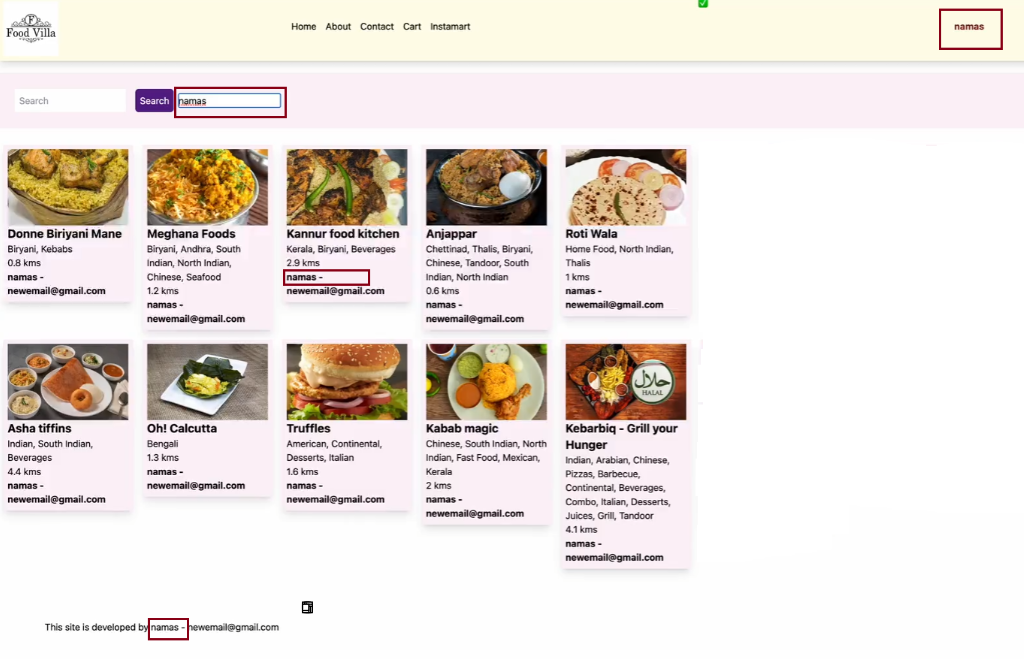




UI Screen:

When I type some texts inside the textbox, wherever we are using context, data will be modified.

We are using context data in header, restaurant card and footer section. The data is getting modified in these components. Any change in textbox triggers reconciliation which in turn change the context data very quickly.



Additional Notes:

* React router DOM uses react context behind the scenes.
* Can we create global variable to share the data across all of our component?

We can use global variable but react wont track this variable just like it can’t track local variables. React can’t trigger reconciliation if it cannot track/watch the variables. It’s like a data store.

**HW:**

1.Build your own accordion in react?

2.How to make react dev tool track context data?

3.Can a context come inside another context? can we have multiple nested contexts?

**Source:**

YouTube, Stack Overflow, ChatGPT