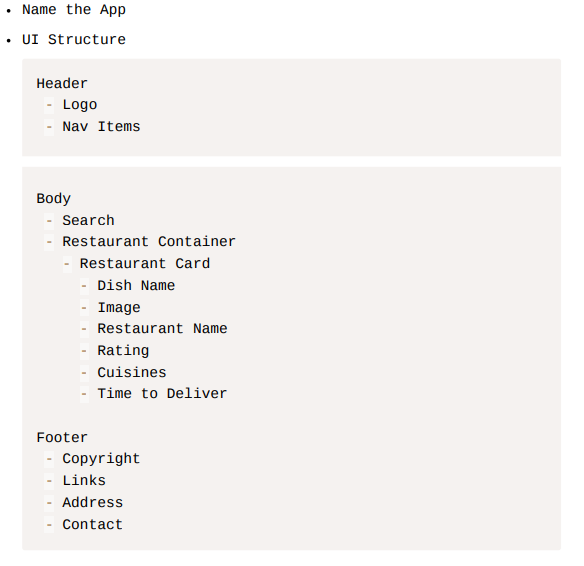
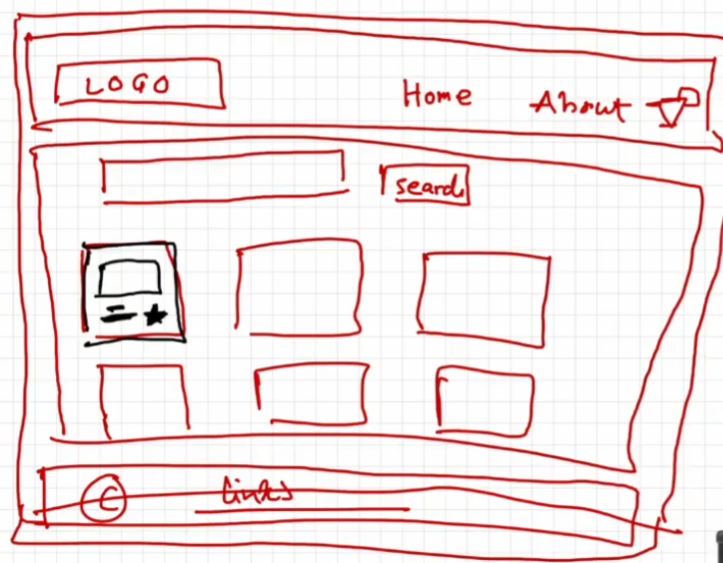
Chapter - 4: Talk is Cheap Show me the code

In this episode, we will start actual coding by starting a new project. Our app is going to a Food Ordering App. Planning for the UI Before we start coding, plan things out. Planning will make things easier to understand. We should know exactly what to build.

Low level design -



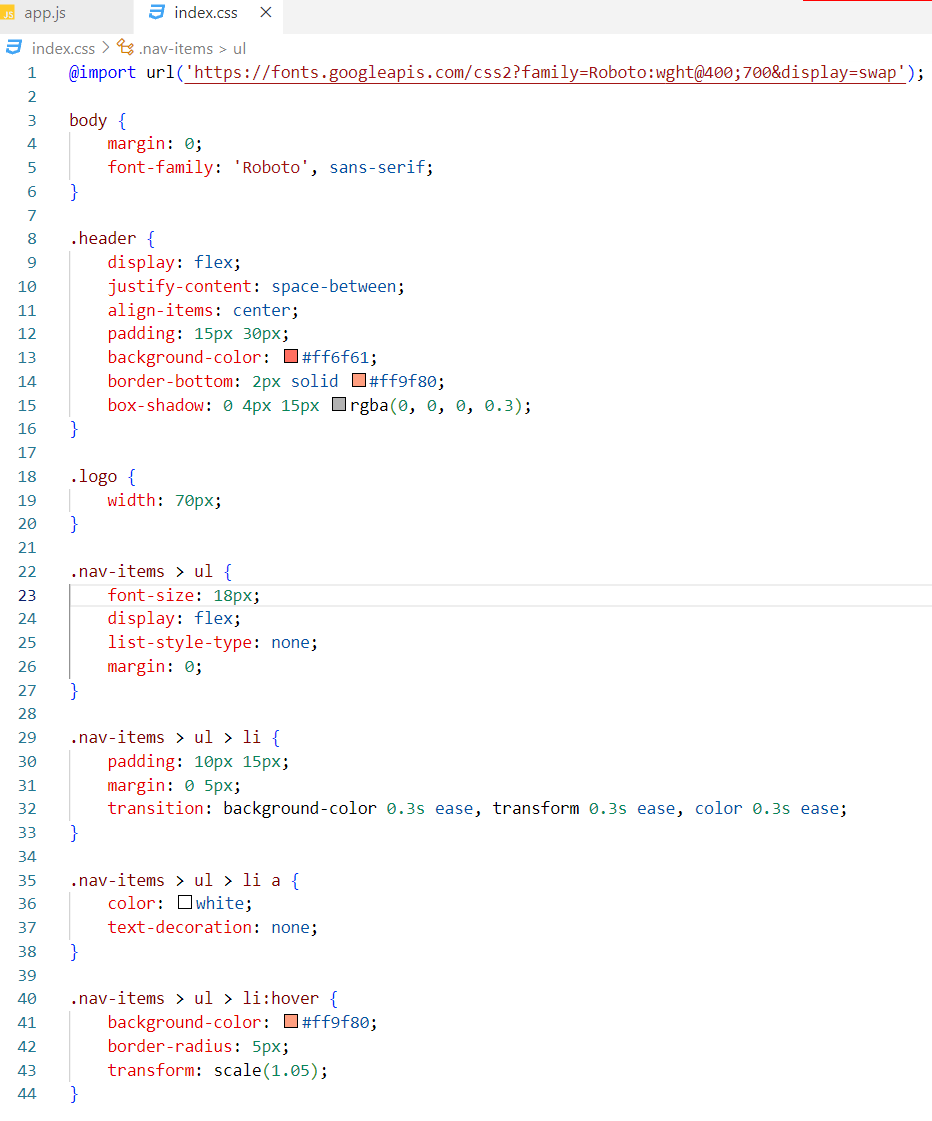


Let’s start coding!!!

Creating header component –

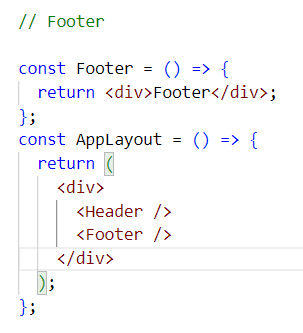


Designing header component –





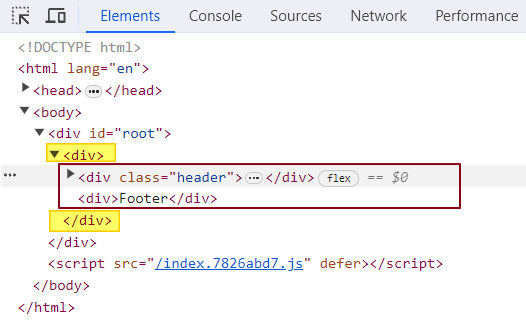
Creating Footer component –



Note - JSX expressions must have one parent element

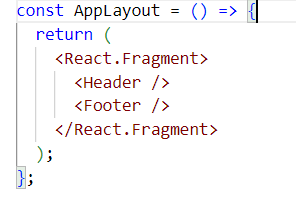
In the AppLayout component, there are two JSX expressions. One for the Header component and another for the Footer component. These expressions are enclosed within a <div> element, which adds an extra node to the DOM tree when rendered.

To eliminate this unnecessary <div> node, React Fragments can be used. Fragments allow you to group multiple elements without introducing an additional DOM element.



What is React Fragment?

React.Fragment is a component exported by the React library. It allows you to group a list of children without adding an extra <div> node to the DOM. In our case, React.Fragment groups the Header and Footer components, rendering them in the DOM tree without any additional nodes.



Additionally, you can use a shorthand syntax, <></>, instead of React.Fragment. However, note that you cannot apply styling to these empty angle brackets.

How can we achieve styling in React?

Inline Styling:

Use the style attribute with a JavaScript object.

Example: <div style= {{color: 'blue', fontSize: '20px'}}>Hello World</div>

External CSS:

Import a CSS file and apply class names to elements.

Example: <div className="myClass">Styled with CSS</div>

External Libraries (Tailwind, Bootstrap):

Use libraries like Tailwind for utility-first styling.

Example with Tailwind: <div className="text-blue-500 text-xl">Styled with Tailwind</div>

Styled Components:

CSS-in-JS where styles are written within JavaScript.

Example:

const StyledDiv = styled.div`

color: green;

font-size: 18px;

`;

<StyledDiv>Hello Styled Components</StyledDiv>

What is Inline Styling in react?

**Inline styling in React** refers to the practice of applying styles directly to elements using a JavaScript object. You create a JavaScript object that contains the style properties you want to apply, and then you use this object within the style attribute of your JSX, wrapped in curly braces.



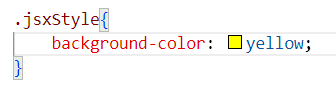


Alternate code :



What is External Styling in react?

Assign a className to the JSX element, then define the corresponding CSS rules in the CSS file.

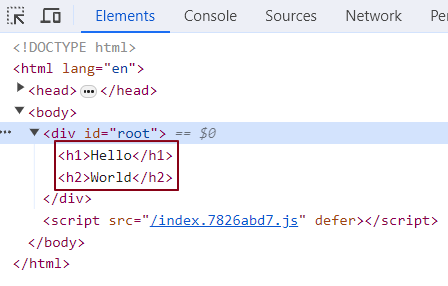


Can I use React.Fragment inside another React.Fragment ?

Yes, we can.

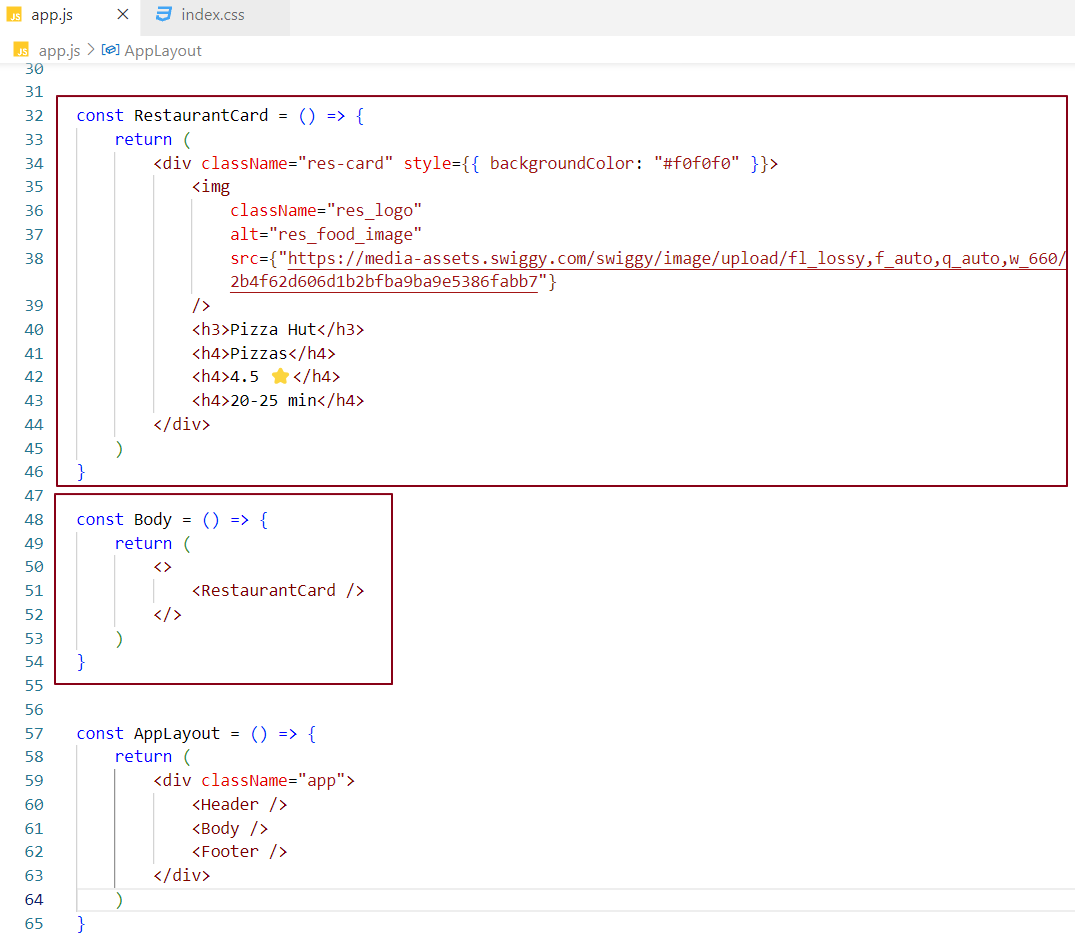


Empty Angular tags are not rendered in the DOM; they are used solely for grouping child elements.

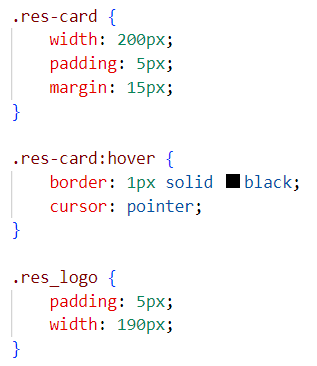
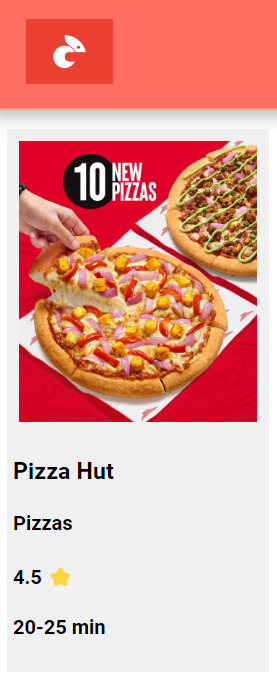


Creating body component –

We need to display a list of restaurant card components within the body component. Let's start by creating a single card component with hardcoded data.



Designing Body Component –

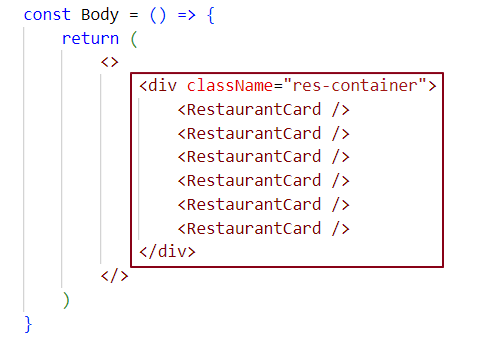
 

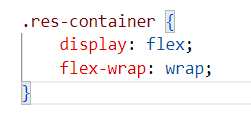
As we know, JavaScript expressions can be used inside JSX. Let's create a JavaScript object called pizzaHut and use it within the RestaurantCard component. We can access the object's properties using dot notation inside JSX by placing them within curly braces.

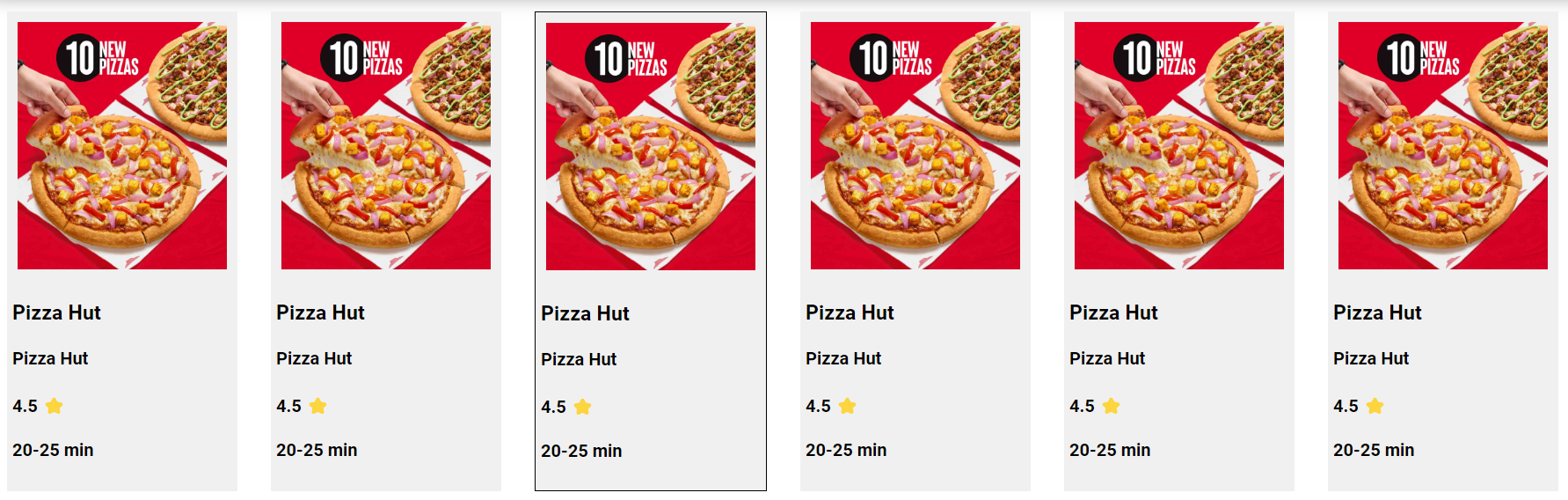


The app will continue to function as it did before, and the body component's UI will remain unchanged.

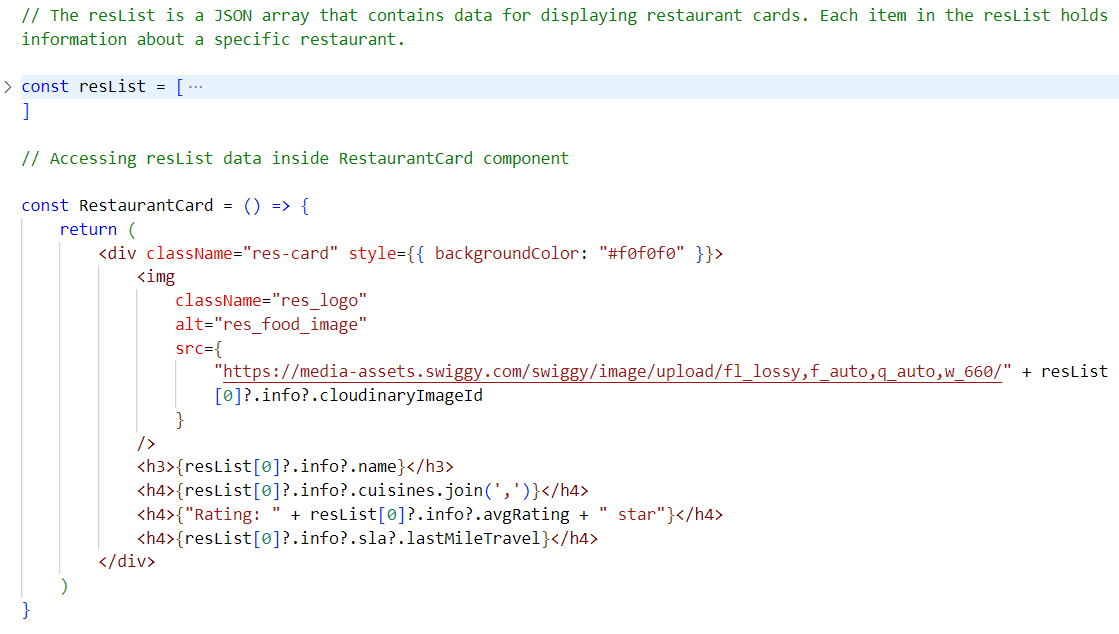
In a real-world application, we often see a list of card components with varying data. We can reuse the RestaurantCard component to display as many cards as needed in the UI. Let's simulate this behaviour.

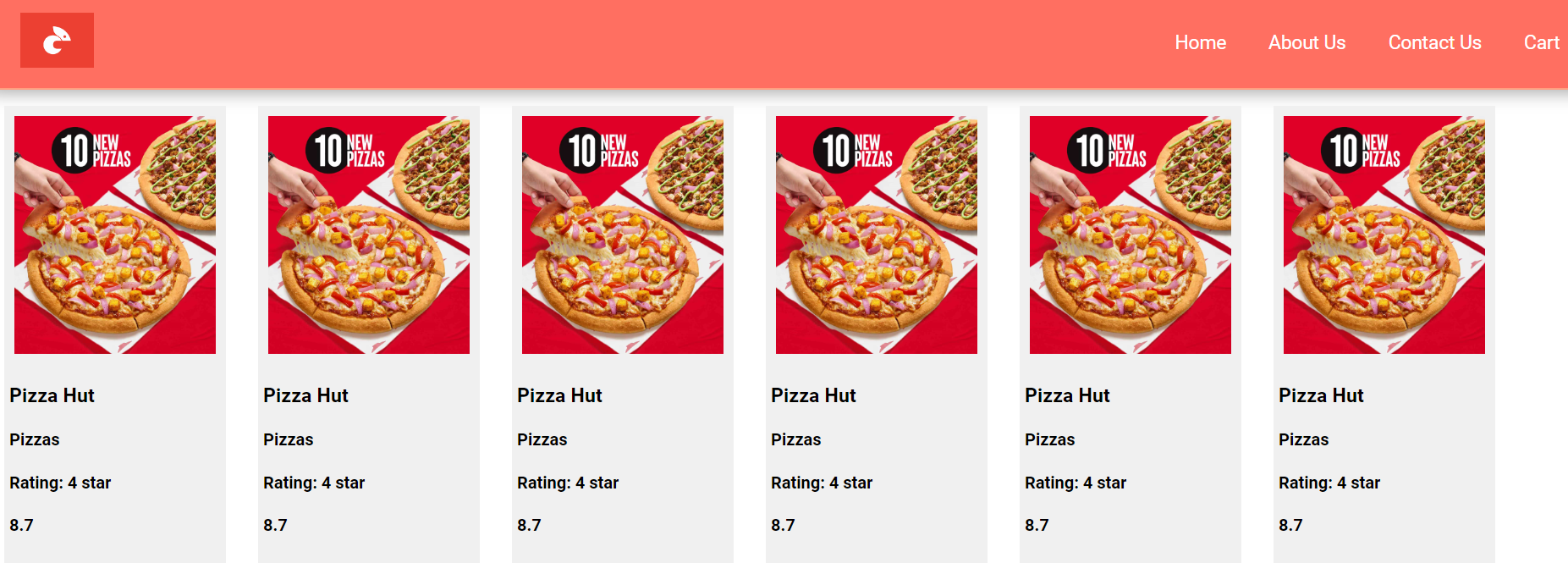






Let's fetch real-time restaurant data from the Swiggy API and display it using the RestaurantCard component in the UI.





Optional chaining (?.)

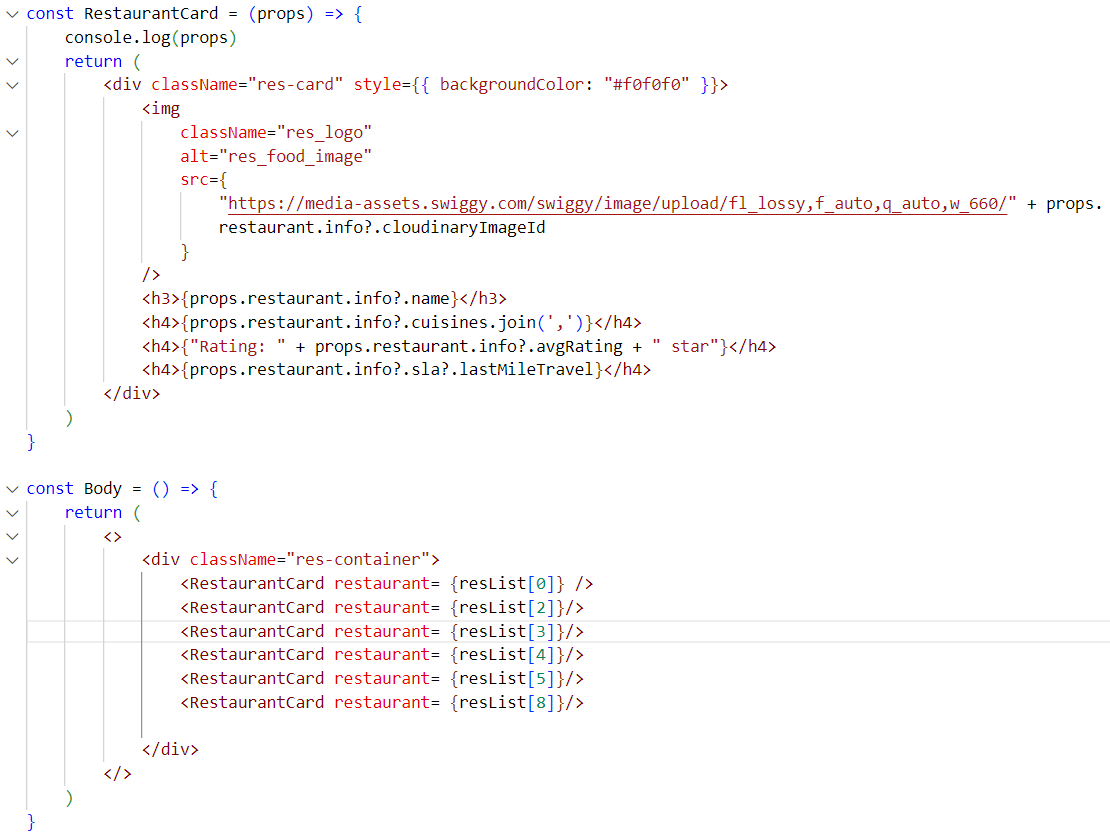
The optional chaining (?.) operator allows access to an object's properties or calls a function safely. If the object or function being accessed is undefined or null, the expression short-circuits and evaluates to undefined, preventing any errors.

So far, we could duplicate card components inside the body component. However, we need to display different data in each card, and this data should be dynamic.

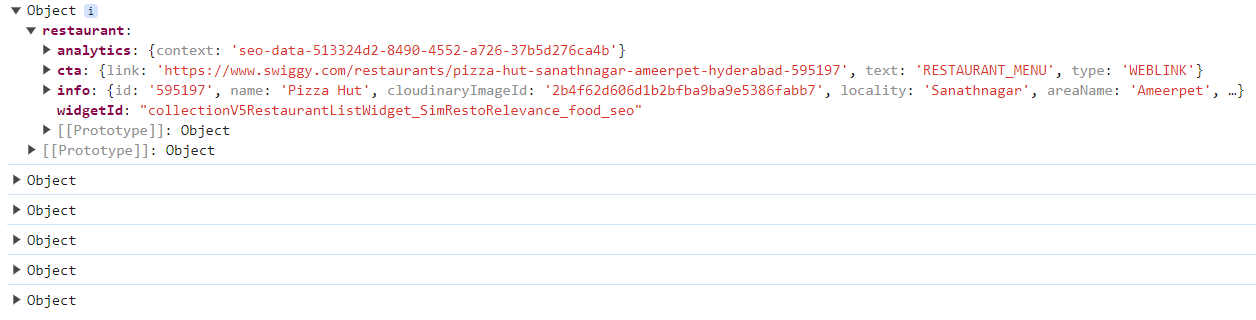
To achieve this, we must pass the data as props from the parent component to the child components. In our case, all restaurant components are rendered within the parent Body component. We need to pass dynamic data as props to each RestaurantCard component so that the child component can receive and display the relevant information.

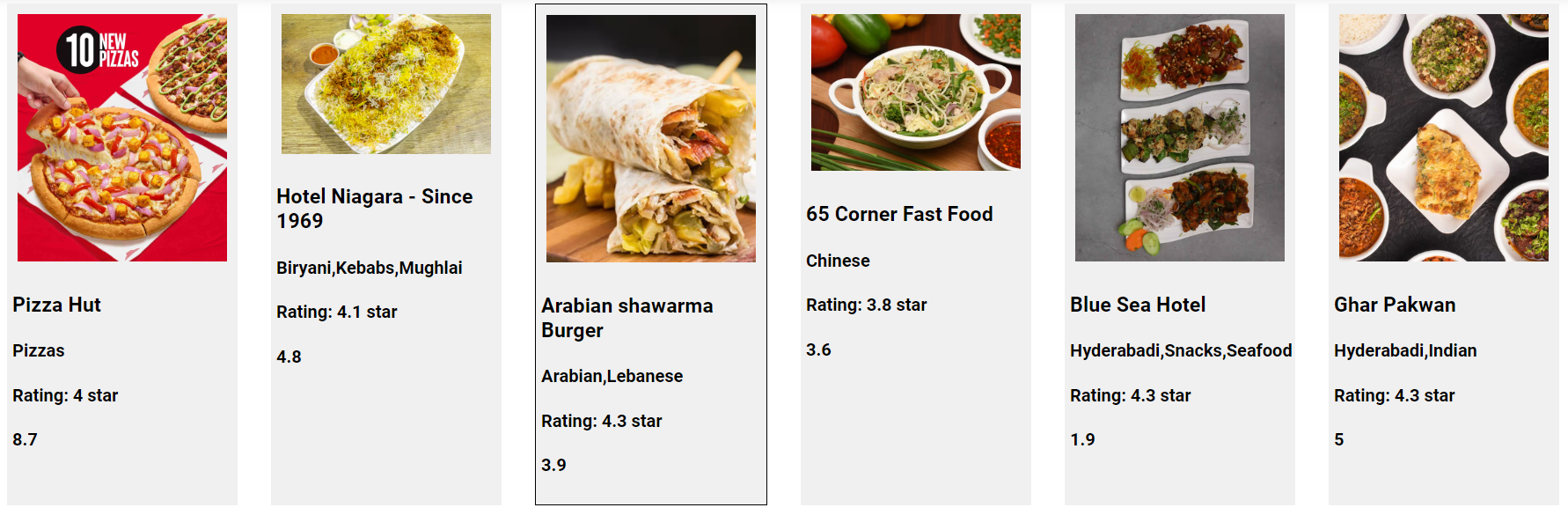
What are props?

Props, short for properties, are similar to passing arguments to a JavaScript function. When we pass props to a component, react wraps them into a JavaScript object. This allows us to access the properties within the component easily.



Here, we are passing a property restaurant from the parent component (Body) to the child component (RestaurantCard). As we can see in the console, this property is wrapped inside an object.





How can we avoid wrapping the restaurant property inside an object?

We can achieve this by using function call syntax.





This is a restaurant object, not a restaurant object wrapped inside another object. Now, the child component receives the props value directly instead of having it wrapped inside an object.

Destructuring props in the child component

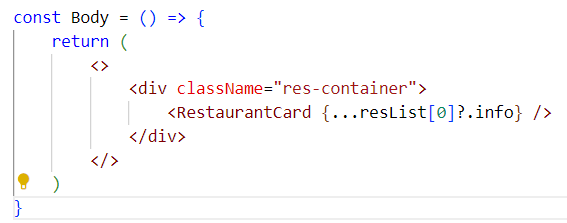




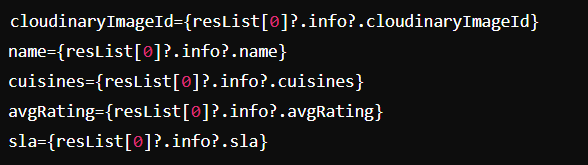
Destructuring props directly in the function parameter without using the spread operator -



Destructuring props directly in the function parameter using the spread operator -



The expression ...resList[0]?.info is using the spread operator for object destructuring. This spreads the properties of resList[0]?.info into individual key-value pairs, equivalent to writing:



What is Spread Operator (…)?

The spread operator expands or unpacks an Iterable, such as arrays or objects, into individual elements where multiple values are expected.

If we have 50 cards, calling the RestaurantCard component 50 times is not efficient. To improve the code, we can use a loop. Let's use the map () function, which is a higher-order JavaScript function. We'll use map () inside JSX within {}.

In the code below, we loop through each restaurant object in the restaurant list and destructure each restaurant's data when rendering the RestaurantCard component.



We have created the basic template for the header, footer, and body components. We will add a search bar to the body component in a later section.

Now, let's talk about what makes React fast.

Note: Use the rafce shortcut to quickly create a component.

Why is React fast?

React is fast because it uses a virtual DOM to handle DOM manipulation, which is typically a heavy and time-consuming task. By reducing direct interaction with the real DOM, the virtual DOM allows React to update the UI more efficiently, improving overall performance.

What is the virtual DOM, and why do we need it?

**Virtual DOM (VDOM)**:

* The virtual DOM is a lightweight representation of the actual DOM, implemented as a JavaScript object.
* Each node in the virtual DOM corresponds to a DOM element and contains properties like type, props, and children.

**Reconciliation Process**:

* When a component's state changes, react creates a new virtual DOM tree.
* React compares the new virtual DOM tree with the previous one using a process called **diffing**.
* This comparison occurs at the object level and involves:

**Node Type Comparison**: If the type of a node changes (e.g., from <div> to <span>), React replaces the entire node.

**Props Comparison**: If the node type remains the same but its properties change, React updates only the modified properties.

**Children Comparison**: For nodes with children, react recursively compares the child nodes to determine which need to be updated, added, or removed.

When you create a JSX element or a functional component in React and log it, you'll see that it outputs an object representing the virtual DOM.

Explanation:

* **JSX**: When you write JSX (e.g., <div>Hello, World!</div>), it gets transformed into a JavaScript object by React. This object contains information about the type of element (in this case, a div), its properties (props), and its children.
* **Functional Component**: Similarly, when you define a functional component and render it, react creates an object representation of that component in the virtual DOM.

Example:

function MyComponent () {

return <div>Hello, World!</div>;

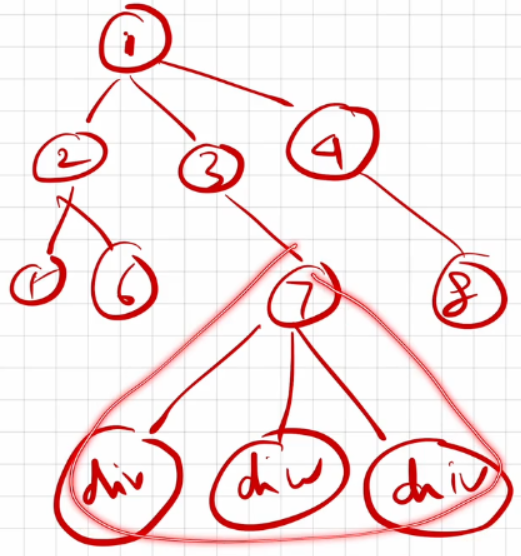
}

console.log(<MyComponent />);

In this case, logging <MyComponent /> will output an object that resembles the virtual DOM structure, which includes the component's type, props, and any nested children.

What is Reconciliation in react?

Reconciliation is the process React uses to update the browser's DOM. It involves a diffing algorithm that compares the actual DOM tree with the virtual DOM tree to identify differences. React then updates only the changed parts of the UI, rather than re-rendering the entire page. This approach makes React highly efficient and fast.

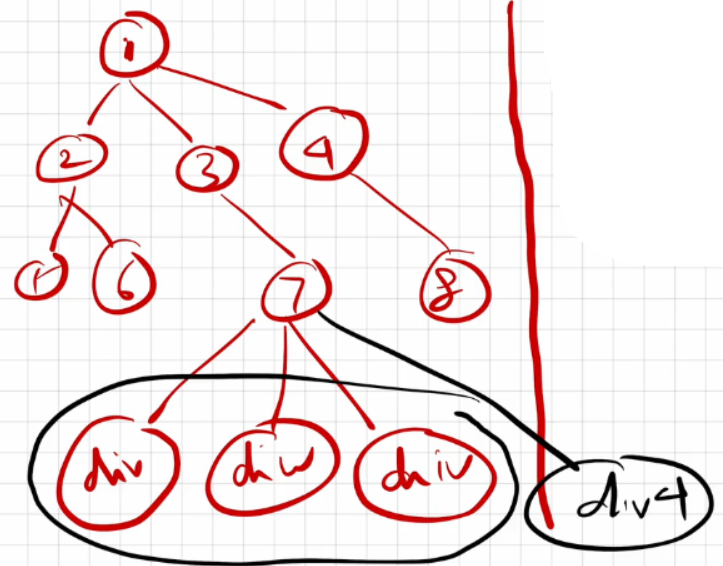


In the DOM tree, if a change happens at the 7th node, react doesn’t re-render the entire DOM. Instead, it re-renders only the updated part of the UI through reconciliation. The diffing algorithm detects the specific change and ensures only the modified section is updated, making the process more efficient.

What are Keys in React, and why do we need them?

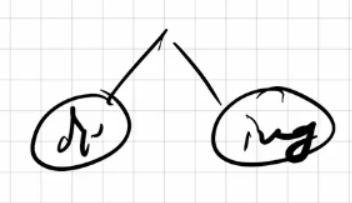
Keys are used in React to uniquely identify elements in a list, allowing React to determine which items have changed, been updated, or deleted. Keys provide a stable identity to elements, which helps React efficiently track and manage the list during re-renders. This improves performance by enabling React to update only the necessary elements instead of re-rendering the entire list.

case 1:



In this scenario, if a new div (let's call it div4) is added at the 7th node, react may get confused because all the elements are div tags. Without unique keys, react won't know which div was added, and it won't understand the order of the div elements in the DOM. This lack of identification can lead to inefficient updates and unexpected behaviour during re-renders.

case 2:



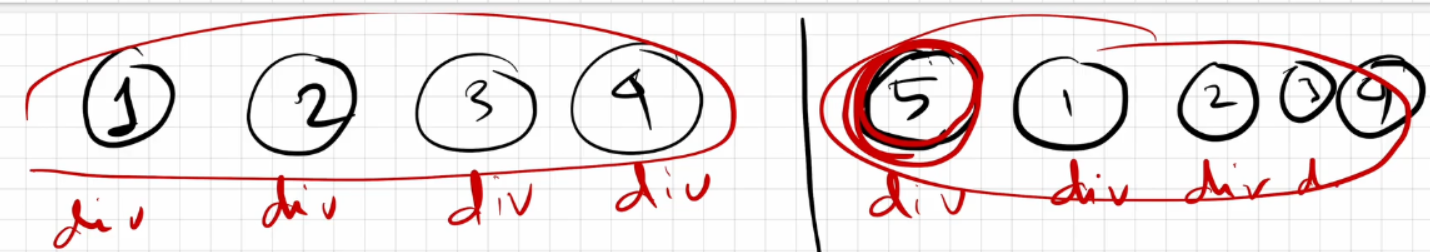
In this case, if a div is changed, an img tag is modified, or if both are swapped, react can accurately track these changes.

case 3:

Whenever we have multiple children with the same tag names, react cannot effectively track them.



Why keys? A detailed explanation.

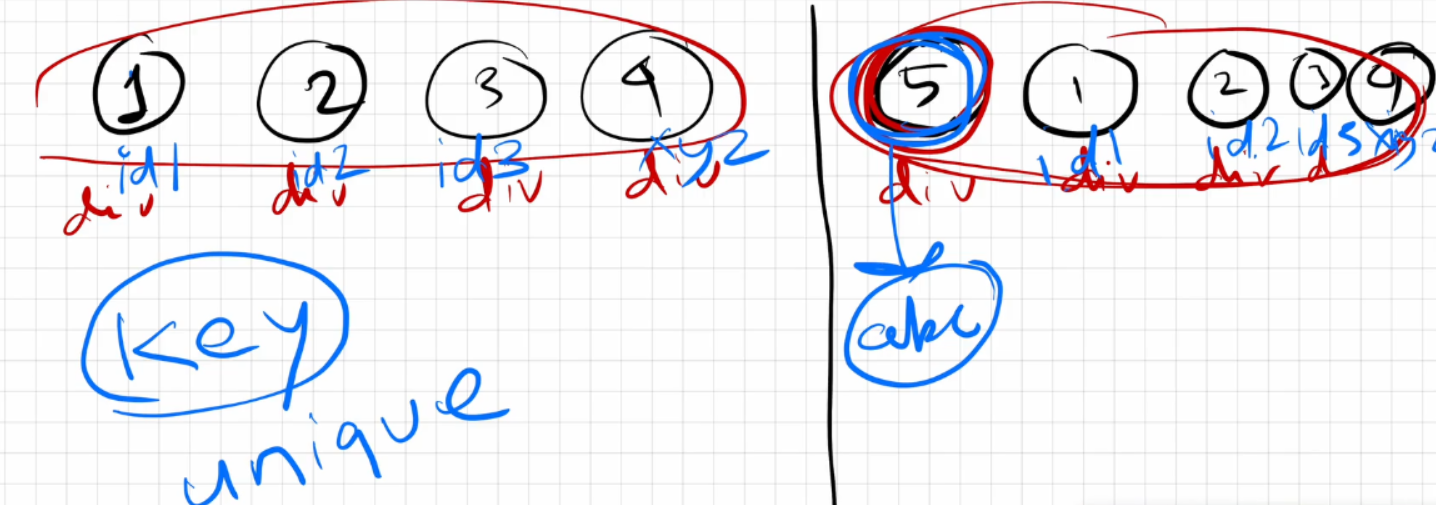


In this example, the left-hand side shows the Actual DOM. When we update something in the Actual DOM, the change doesn't happen right away. Instead, react makes a copy of the Actual DOM called the Virtual DOM, where the changes are applied first. In this case, the right-hand side represents the Virtual DOM.

React uses a diffing algorithm to figure out what has changed. For instance, if we add a 5th div at the beginning of the existing 4 div elements, react sees the new div as being the same as the other four. Because of this, react doesn't know where to place the new div in the order of div elements in the Actual DOM. It can't determine their placement order.

As a result, react re-renders all five div elements in the UI, even though we only wanted it to render the new 5th element.

To update efficiently, react needs to keep track of all the div elements. This is where keys come in handy. Keys help React uniquely identify each element, which allows it to handle updates properly. By using keys, react can figure out which elements have changed and only re-render the parts that need to be updated in the Actual DOM.



Now that we have given keys to all the div elements, react won't get confused this time. When we add the 5th div at the beginning of the 4 existing div elements, react uses the reconciliation technique along with the diffing algorithm to compare the actual and virtual DOM trees.

It checks the keys: four keys match, but the key for the 5th div in the virtual DOM does not match any of the existing keys. As a result, react understands that it needs to render only this new 5th element in the actual DOM. This process is known as the re-render phase.

How did React know where to place the 5th div?

In the virtual DOM, react checked where the 5th div was located. It found that the 5th div is right before the first div. React was able to track this position using the key names. Therefore, when React re-rendered the actual DOM, it correctly placed the 5th div before the first div.

Is JSX mandatory for React?

No, JSX is not mandatory for React. JSX and React are different; JSX is a syntax extension that makes it easier for developers to write HTML-like code in JavaScript. It serves as syntactic sugar over pure React code, allowing for more readable and convenient component creation.

Is ES6 mandatory for React?

No, you can write React code using pure JavaScript without ES6. However, ES6 introduces useful features like the spread operator and arrow functions that can help you build your React application more easily. While ES6 is recommended for modern React development, it is not required.

How to write comments in JSX?

JSX comments are written inside {/\* \*/} and can be used for both single-line and multi-line comments.

**Example:**

{/\* This is a single-line JSX comment \*/}

{/\*

This is a

multi-line

JSX comment

\*/}

What is React Fibre? Use cases of React Fibre.

In older versions of React (prior to version 16), React used the Stack Reconciler Engine (SRE) for reconciliation. However, it had some drawbacks, such as being synchronous.

SRE operates like a stack, working synchronously until the stack is empty.

Let’s consider an application with a text box in the UI that makes API calls to fetch data before any input can be entered. Until the data is fetched, if you type anything, nothing will be displayed. You will only be able to type in the text box once the API data is retrieved. This means our app behaves synchronously, and to perform both tasks in parallel, an asynchronous operation needs to be implemented.

To overcome this drawback, React Fiber was introduced after version 16. It uses an updated reconciliation engine that has been completely rewritten from the ground up and is capable of performing asynchronous tasks.

Perks/Benefits

\* React Fiber focuses on animations and responsiveness.

\* Fiber makes React faster and more efficient.

\* Fiber splits work into chunks and prioritizes tasks based on their importance.

\* Fiber can resume, pause, and restart rendering work on components as new updates come in.

\* Fiber reuses previously completed work and can even abort tasks that are no longer needed.

Can we use index as keys in React?

Yes, you can use the index of an item in a list as a key in React. However, it’s not a good practice for a few reasons:

1. Unexpected Behaviour: If the list changes, using the index as a key can cause React to mix up the elements. This can lead to items being rendered incorrectly or in the wrong order.
2. State Loss: If you use the index as a key and reorder or change the list, react might not remember the state of each item. This can cause issues, especially if you have form elements or animations.
3. Performance Issues: Using indexes can lead to performance problems when rendering large lists because React has to re-render more components than necessary.

Note: You can ask GPT for practical examples to enhance your understanding.

It's better to follow this order for keys:

1. Use unique keys if you have them (like IDs).
2. Use indexes only if you have to, but know it can cause issues.
3. Don't use no keys at all, because that can create problems.

What is Config Driven UI?

**Config Driven UI** is based on the configurations of the data that an application receives. Config Driven UI is managed from the backend.

A great example would be the Swiggy app's image carousel. Swiggy shows food offers based on geographical locations. Let’s say my home address is in Odisha and my friend’s home address is in Mumbai. We both access the Swiggy app from different places, but I might see different food offers in my region compared to what my friend sees in his region. Usually, we see these offers in the image carousel section of the Swiggy application. So, this image carousel section in the Swiggy app is config driven, meaning we are viewing configured data in the UI based on our locations.

These configurations are handled from the backend. In some regions where there aren’t great offers, no data is sent from the backend to the UI, so people don’t see the image carousel section at all.