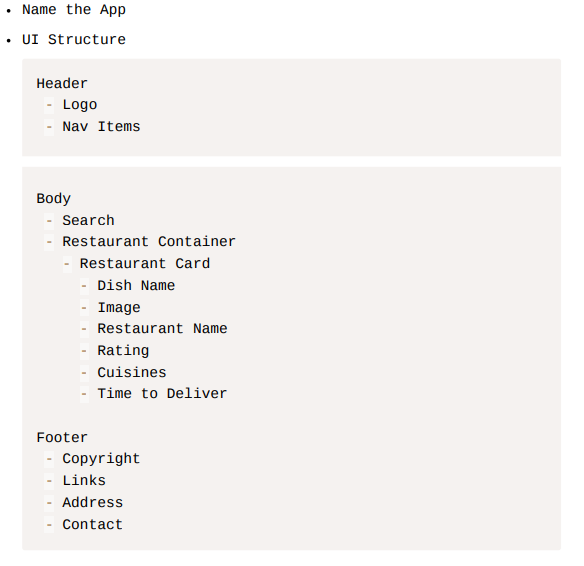
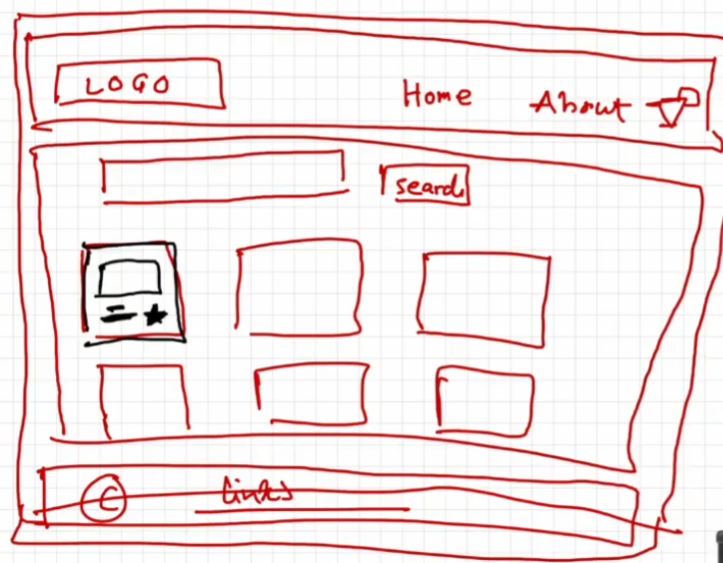
Chapter - 4: Talk is Cheap Show me the code

In this episode, we’ll begin actual coding by starting a new project — a Food Ordering App.

### Planning the UI

Before we dive into coding, it’s important to plan everything out. Proper planning will make the development process smoother and easier to understand. We should have a clear idea of what we’re building before we write any code.

Low level design -

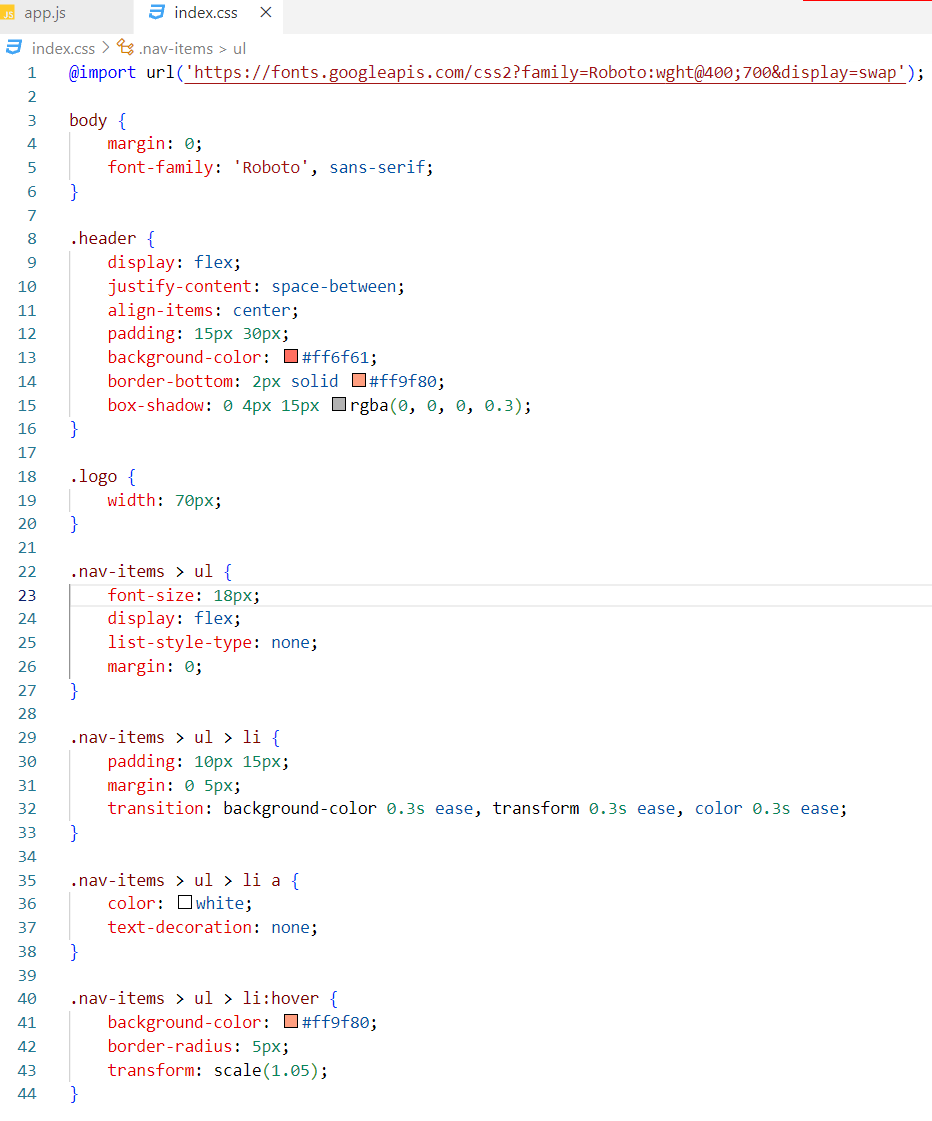


Awesome! Let’s dive in and start coding! 🚀

Implementing the Header Component in React –

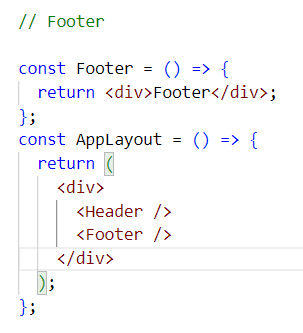


Designing the Header Component –





Implementing the Footer Component in React –



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 currently wrapped inside a <div> element, which introduces an extra node into the DOM tree when rendered.

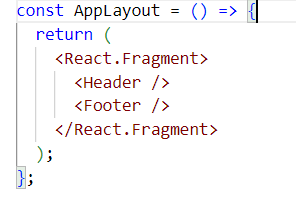
To avoid adding this unnecessary <div>, we can use React Fragments. Fragments allow us to group multiple elements without adding an extra DOM node, keeping the DOM cleaner and more efficient.



What is React Fragment?

React.Fragment is a special component provided by the React library. It lets you group multiple child elements without inserting an extra <div> or wrapper element into the DOM.

In our case, we use React.Fragment to wrap the Header and Footer components. This allows them to be rendered side by side without introducing any additional nodes into the DOM tree — keeping the structure clean and lightweight.



Additionally, you can use the shorthand syntax <>...</> instead of writing React.Fragment. However, it's important to note that **you cannot apply any props or styling** to this shorthand version — it’s purely for grouping elements.

Styling in React: Different Approaches

1. Inline Styling:

Inline styling in React can be done by passing a JavaScript object to the style attribute. The keys in the object represent CSS properties, and the values represent the corresponding styles. Note that the property names in the object are written in camelCase rather than hyphenated style (e.g., font-size becomes fontSize).



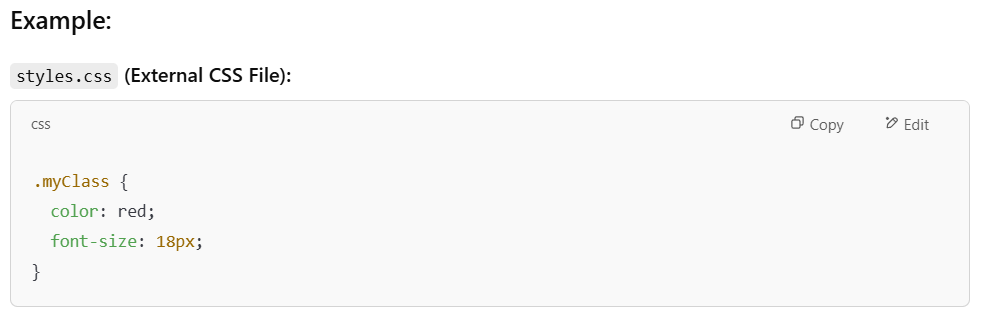
In this example:

* The color property is set to blue.
* The fontSize property is set to 20px.

This approach is useful when you want to apply styles directly within the JSX code without using external CSS files or stylesheets.

2. External CSS:

External CSS is a common way to apply styles to your React components. You can define your styles in a separate .css file and import that file into your component. Then, you apply the styles using the className attribute in JSX.





In this example:

* The styles.css file contains the .myClass class definition.
* The className="myClass" in the div element applies those styles.

Using external CSS is more maintainable for larger projects because it allows you to separate the structure (HTML/JSX) from the styling (CSS).

3.External Libraries (Tailwind, Bootstrap):

External libraries like **Tailwind CSS** and **Bootstrap** provide predefined classes that make styling faster and easier by offering utility-first or component-based approaches.

### Example with **Tailwind CSS**:

Tailwind CSS uses utility classes for styling, meaning you can apply multiple small utility classes directly to the element to achieve the desired look.



In this example:

* text-blue-500 sets the text color to blue (a specific shade from Tailwind’s color palette).
* text-xl applies a larger font size.

4.Styled Components:

Styled Components is a **CSS-in-JS** library that lets you write actual CSS code inside your JavaScript files to style your components.



* styled.div → It's a way to create a styled <div> using backticks (`). You write your CSS inside the backticks.
* StyledDiv → This is now your own custom component. It's like a normal <div>, but it already has styles built in.
* You can reuse StyledDiv anywhere in your app, and it will always look the same unless you change the styles.

This approach is ideal for **component-based architecture** and keeps logic and styles close together, which is perfect for modern React development.

What is Inline Styling in react?

Inline styling in React means applying styles directly to JSX elements using a JavaScript object.

You write the CSS properties inside an object using camelCase (instead of kebab-case), and then pass that object to the style attribute using double curly braces - one pair for JSX, and one for the object.



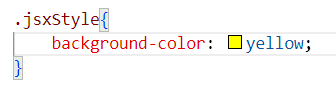
Alternative version of the code -





What is External Styling in react?

Assign a className to the JSX element and define the corresponding CSS rules in an external CSS file.



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

Yes, absolutely!

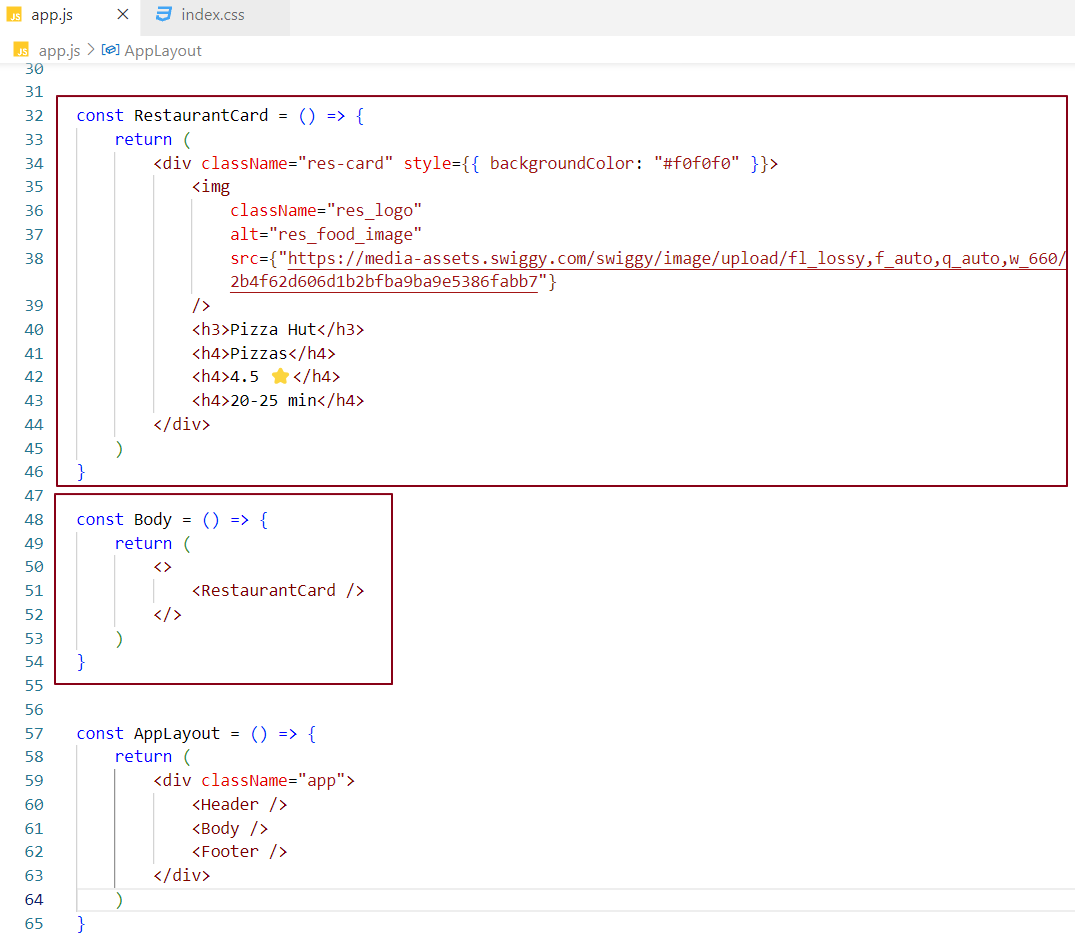


Empty Angular tags (like <ng-container>) are not rendered in the DOM; they are used purely to group child elements without adding extra nodes.

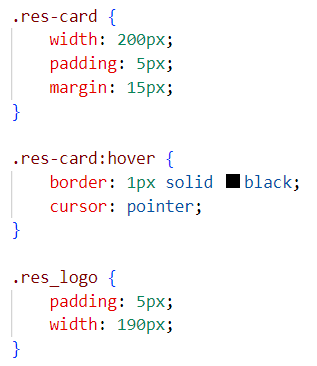
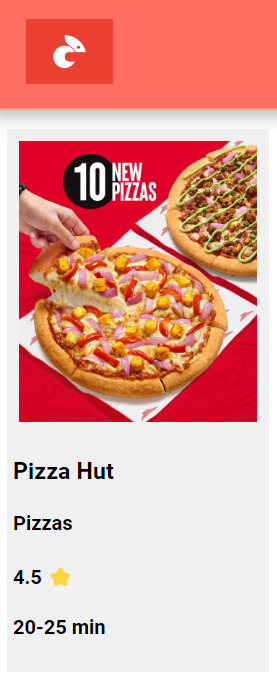


Implementing the Body Component in React –

To display a list of restaurant cards in the Body component, let's begin by creating a single Card component with hardcoded data.



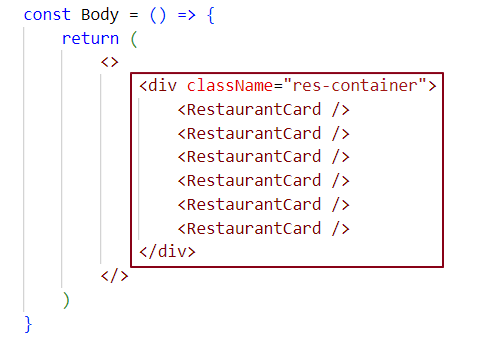
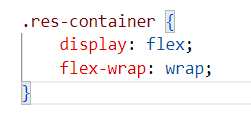
Designing the Body Component –

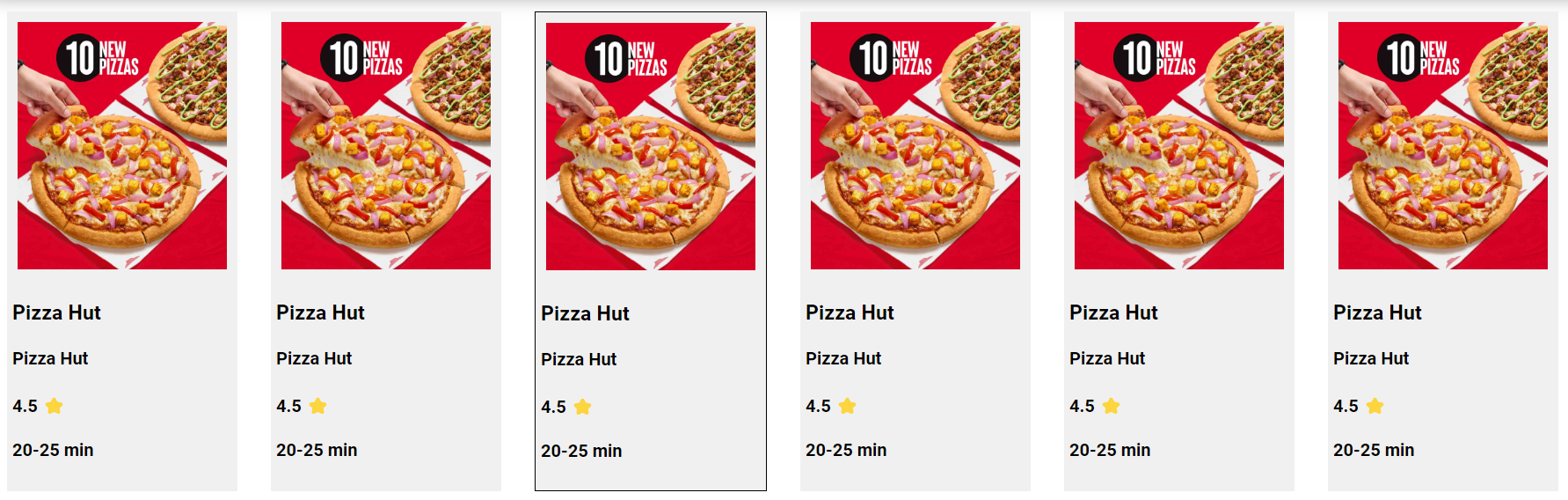
 

As you know, JavaScript expressions can be used within JSX. Let’s create a JavaScript object named pizzaHut and use it inside the RestaurantCard component. We can access the object’s properties using dot notation by wrapping them in curly braces within the JSX.



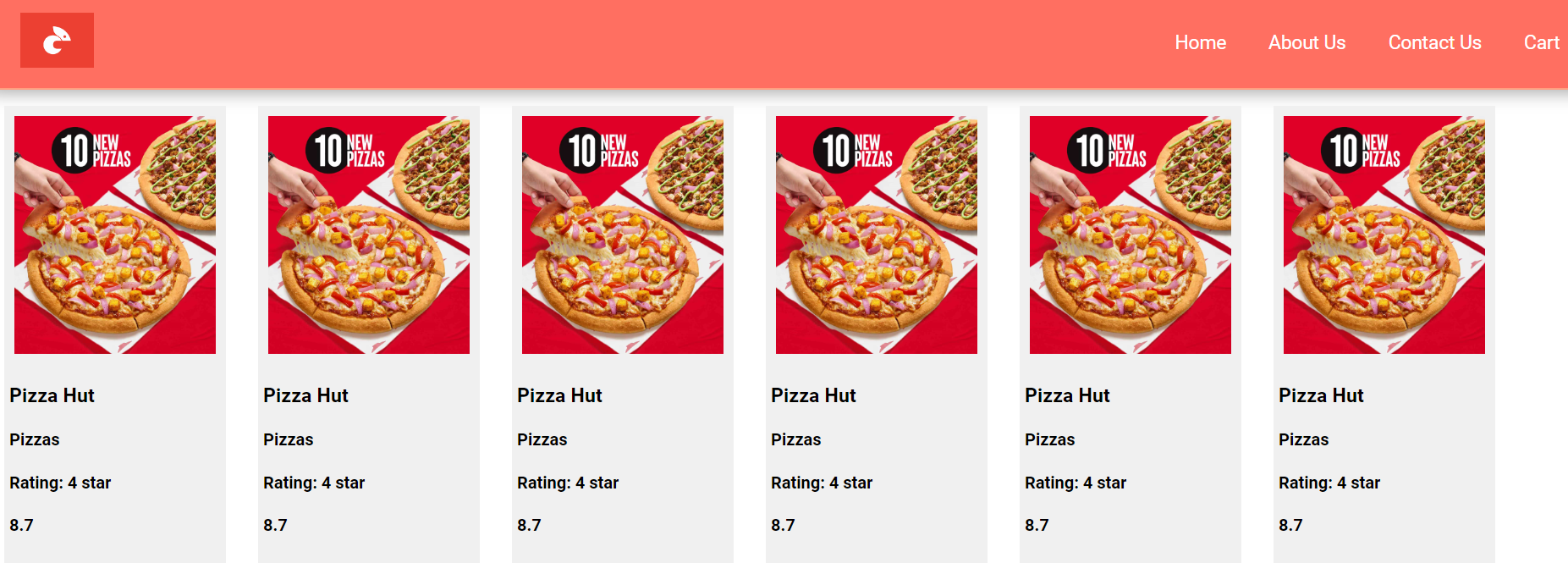
The app will continue to function as it did before, and the UI of the Body component will remain unchanged. In a real-world application, we typically display a list of card components with different data. We can reuse the RestaurantCard component to render as many cards as needed. Let’s simulate this behavior.





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





Optional chaining (?.)

The optional chaining (?.) operator allows safe access to an object's properties or functions. If the object or function is undefined or null, the expression short-circuits and returns undefined, preventing errors.

So far, we’ve been able to duplicate card components inside the Body component. However, to display different data in each card, the data should be dynamic. To achieve this, we need to pass the data as props from the parent component (Body) to the child components (RestaurantCard). Each RestaurantCard will receive its specific data through props and display the relevant information accordingly.

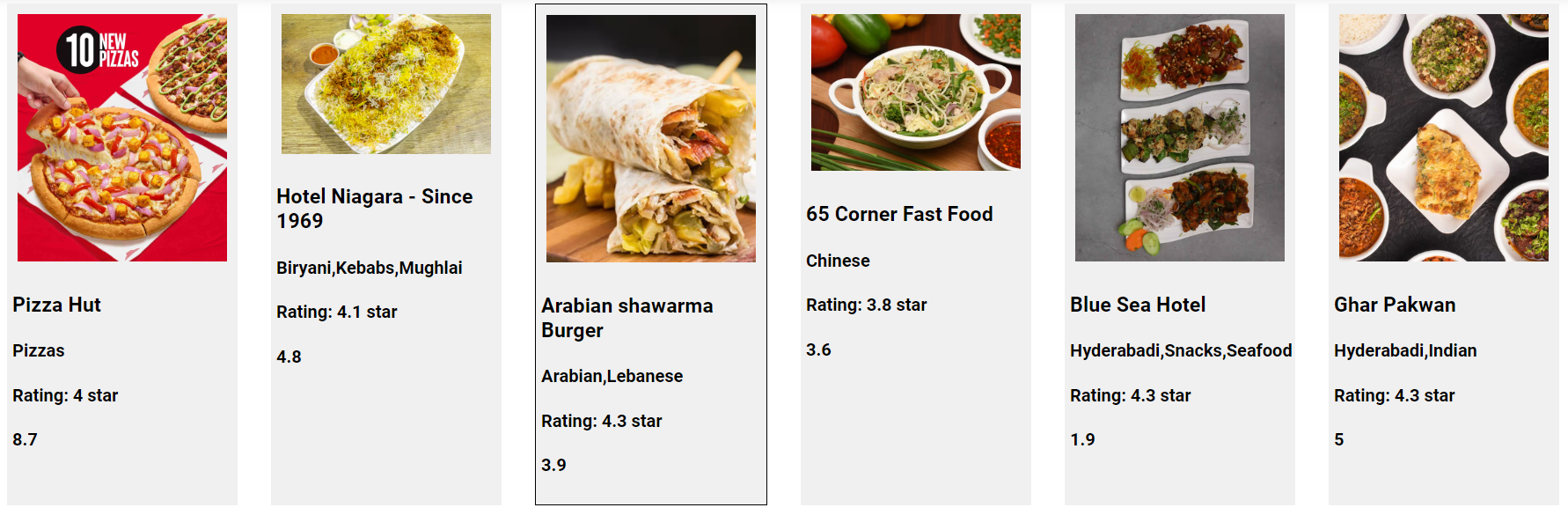
What are props?

Props (short for properties) are similar to passing arguments to a JavaScript function. When we pass props to a component, react automatically wraps them into a JavaScript object. This makes it easy to access and use the properties within the component.



In this example, we are passing the restaurant property from the parent component (Body) to the child component (RestaurantCard). As seen in the console, this property is automatically wrapped inside an object when passed as props.





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

We can achieve this by using **function call syntax** when passing props.





This is the restaurant object itself, not the restaurant object wrapped inside another object. Now, the child component receives the restaurant prop value directly, rather than accessing it from within an object.

Destructuring Props in the Child Component –

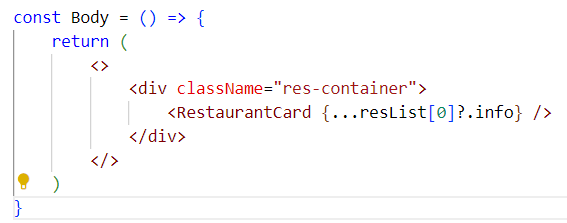




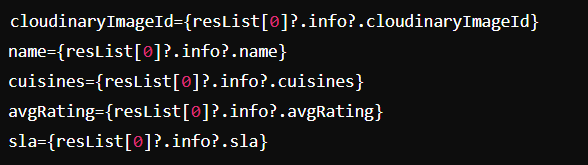
Destructuring Props in Function Parameters Without the Spread Operator-



Destructuring Props in Function Parameters with the Spread Operator-



The expression ...resList[0]?.info uses the spread operator to expand the properties of resList[0]?.info into individual key-value pairs. This is not object destructuring, but rather spreading the object’s properties into a new object. It is equivalent to manually writing out each property from resList[0]?.info.



What is the Spread Operator (...) ?

The spread operator is used to expand or unpack elements from an Iterable - such as an array or object—into individual elements. It is commonly used where multiple values or key-value pairs are expected.

* If we have 50 cards, manually calling the RestaurantCard component 50 times would be inefficient and repetitive.  
  Instead, we can use a loop to generate these components dynamically.
* In React, we typically use the map() function — a higher-order JavaScript function — to iterate over a list of data and return a component for each item.  
  The map() function can be used inside JSX by wrapping it with curly braces ({}).

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

We've created the basic structure for the Header, Footer, and Body components. A search bar will be added to the Body component later.

**Tip:** Use the rafce snippet in VS Code (with ES7+ React/Redux snippets extension) to quickly scaffold a functional component with an export.

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, thereby 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.

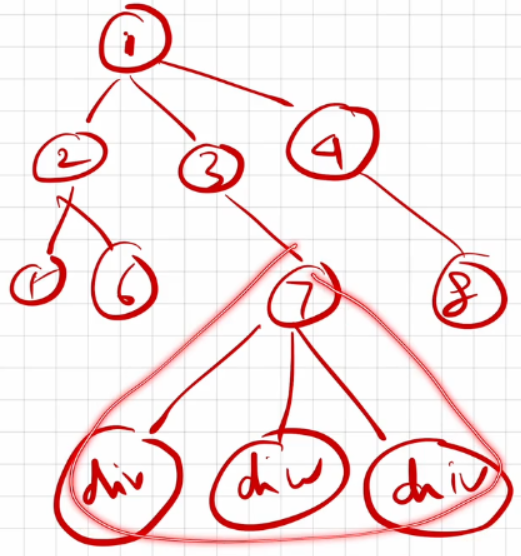
Example:



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 parts of the UI that have changed, rather than re-rendering the entire page. This approach makes React highly efficient and fast.

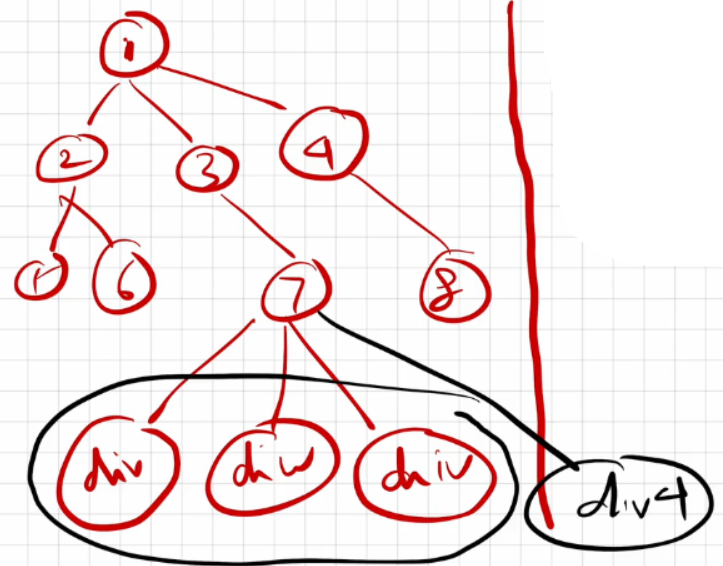


In the DOM tree, if a change occurs at the 7th node, react doesn’t re-render the entire DOM. Instead, it only re-renders the updated part of the UI through reconciliation. The diffing algorithm detects the specific change and ensures that only the modified section is updated, making the process more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

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

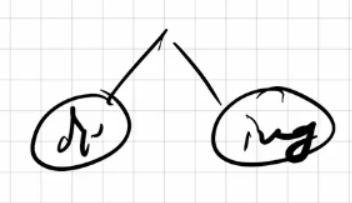
Keys are used in React to uniquely identify elements in a list, allowing React to determine which items have changed, been updated, or been deleted. Keys provide a stable identity to elements, which helps React efficiently track and manage the list during re-renders. This improves performance by enabling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

case 2:



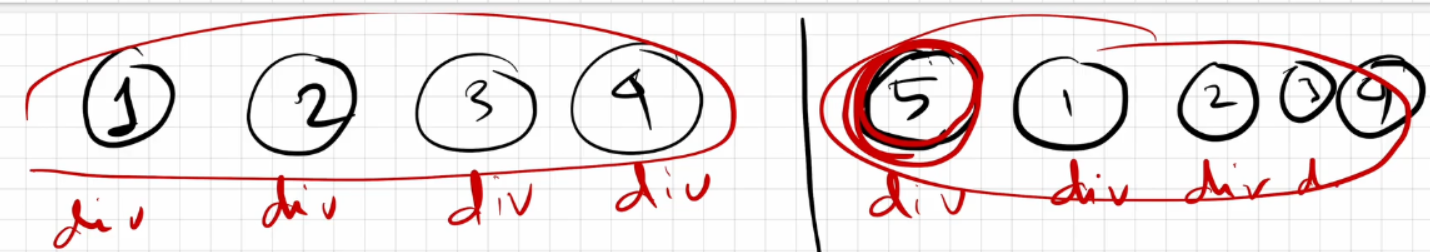
In this case, if a div is changed, an img tag is modified, or if both are swapped, react can accurately track these changes.

case 3:

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



Why Keys? A Detailed Explanation

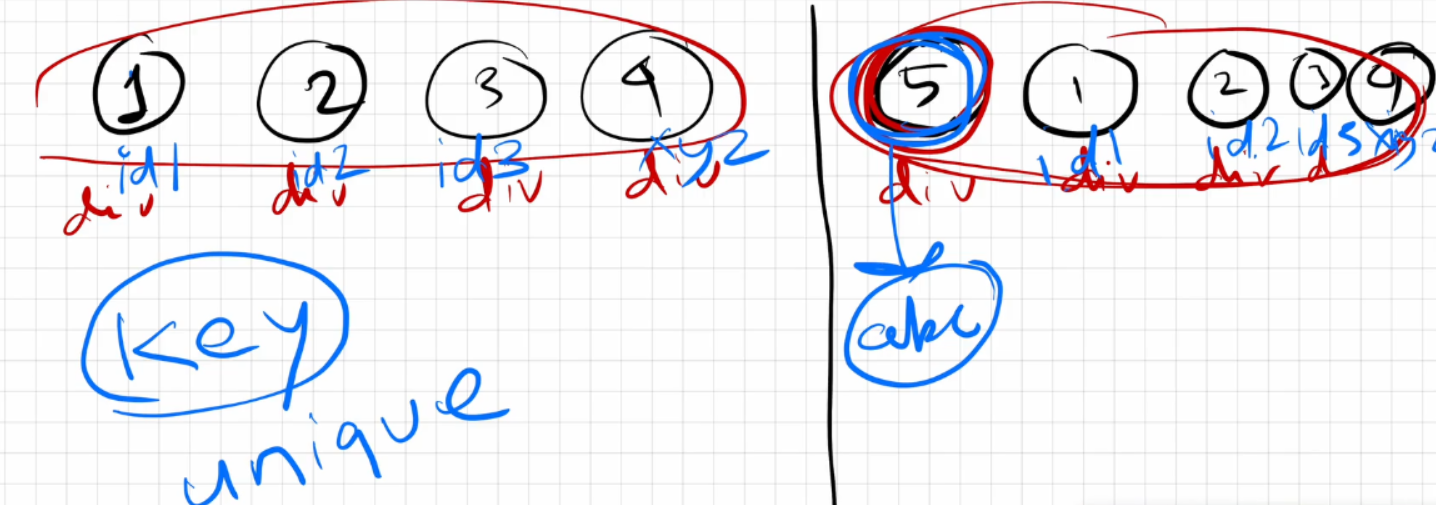


In this example, the left-hand side shows the **Actual DOM**. When we update something in the Actual DOM, the change doesn't happen immediately. Instead, react creates a copy of the Actual DOM called the **Virtual DOM**, where the changes are first applied. The right-hand side represents the **Virtual 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 similar to the others. Since React doesn’t have a way to uniquely identify each element, it doesn’t know where to place the new div in the order of the existing div elements in the Actual DOM. Without proper identification, react can't determine their correct placement order.

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

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



Now that we have assigned keys to all the div elements, react won't get confused. 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 the position of the 5th div and found that it was located 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 distinct concepts; JSX is a syntax extension that simplifies writing HTML-like code within JavaScript. It acts as syntactic sugar over pure React code, making it easier to create components in a more readable and convenient way.

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, which can make building React applications easier and more concise. While ES6 is recommended for modern React development, it is not mandatory.

How to write comments in JSX?

JSX comments are written inside {/\* \*/}, and they 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 Fiber? Use cases of React Fiber.

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 the 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 Behavior**: 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.

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 be aware 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. It is managed from the backend.

A great example would be the Swiggy app's image carousel. Swiggy shows food offers based on geographical locations. For instance, if 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. These offers are typically displayed 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 managed 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.