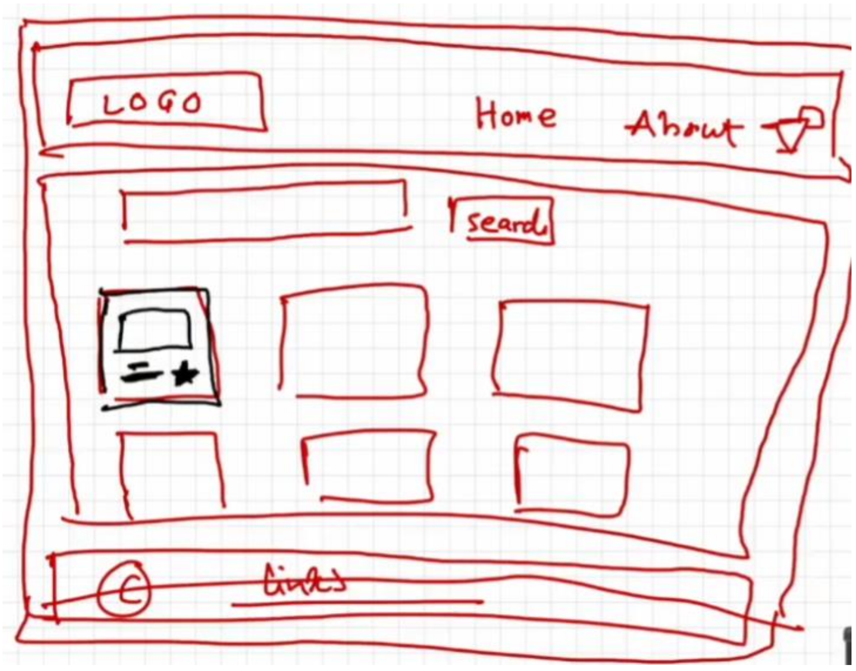
Chapter - 4: Talk is Cheap Show me the code

Let’s build a food ordering app but before we develop our application we need to plan and prepare a wireframe.



Low level design –

Header -

-Logo

- Nav Items

Body -

- search

- Restaurant container

    - Restaurant cards

        -Restaurant card with Image, Name of restaurant, star rating, cuisine, delivery time

Footer -

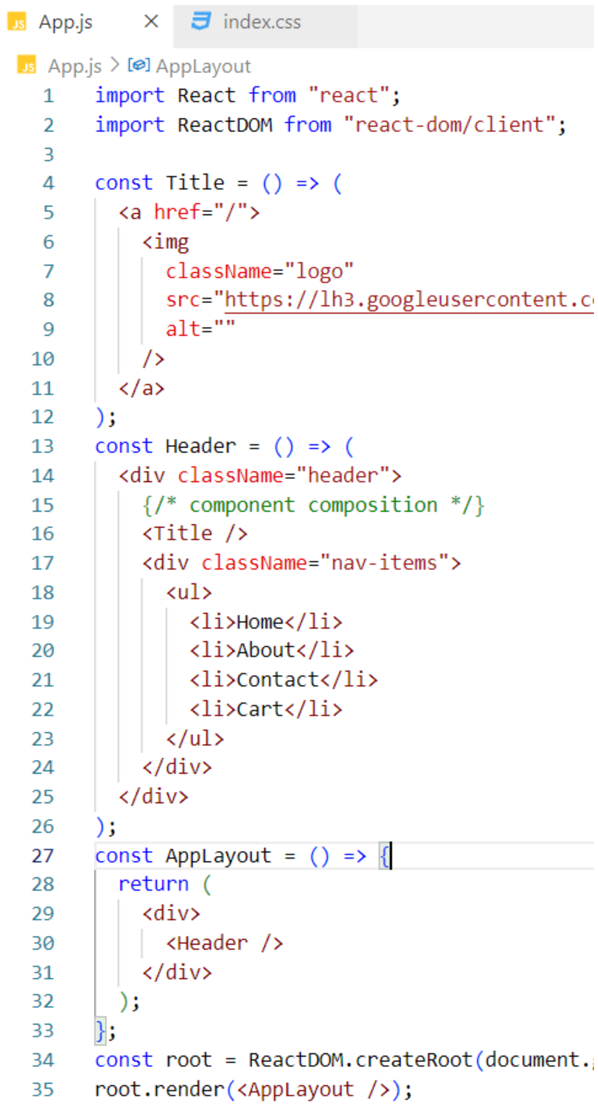
- Copyright

- Links

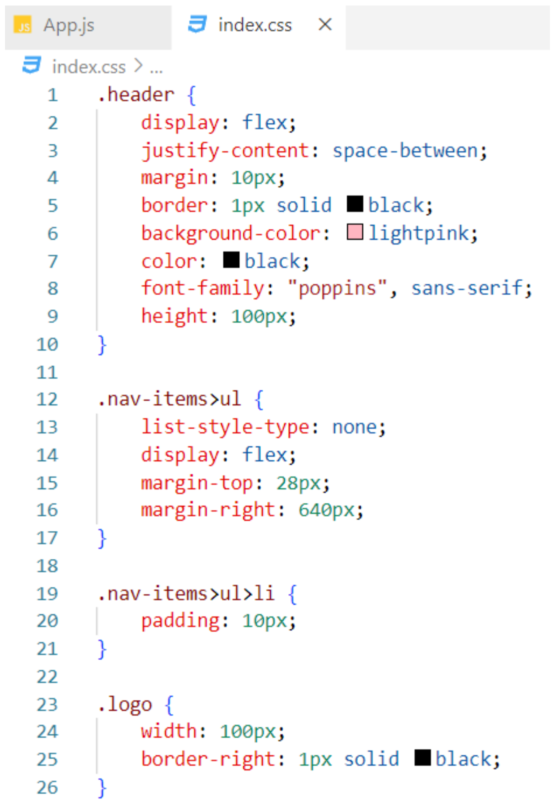
- Address

- contact

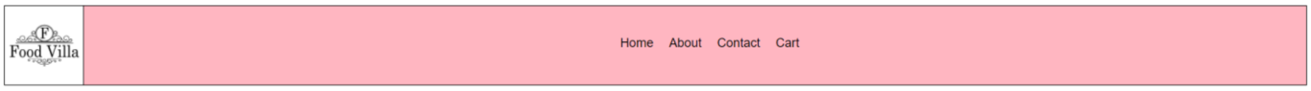
Creating header component –



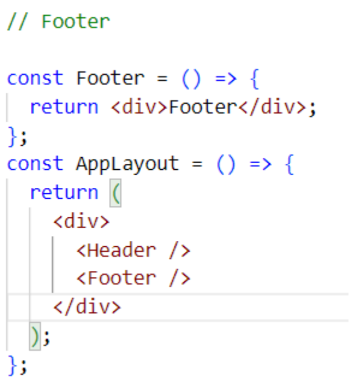
```jsx
import React from "react";
import ReactDOM from "react-dom/client";

const Title = () => (
  <a href="/">
    <img
      className="logo"
      src="https://lh3.googleusercontent.c
      alt=""
    />
  </a>
);
const Header = () => (
  <div className="header">
    {/* component composition */}
    <Title />
    <div className="nav-items">
      <ul>
        <li>Home</li>
        <li>About</li>
        <li>Contact</li>
        <li>Cart</li>
      </ul>
    </div>
  </div>
);
const AppLayout = () => {
  return (
    <div>
      <Header />
    </div>
  );
};
const root = ReactDOM.createRoot(document.
root.render(<AppLayout />);
```



```css
.header {
    display: flex;
    justify-content: space-between;
    margin: 10px;
    border: 1px solid black;
    background-color: lightpink;
    color: black;
    font-family: "poppins", sans-serif;
    height: 100px;
}

.nav-items>ul {
    list-style-type: none;
    display: flex;
    margin-top: 28px;
    margin-right: 640px;
}

.nav-items>ul>li {
    padding: 10px;
}

.logo {
    width: 100px;
    border-right: 1px solid black;
}
```



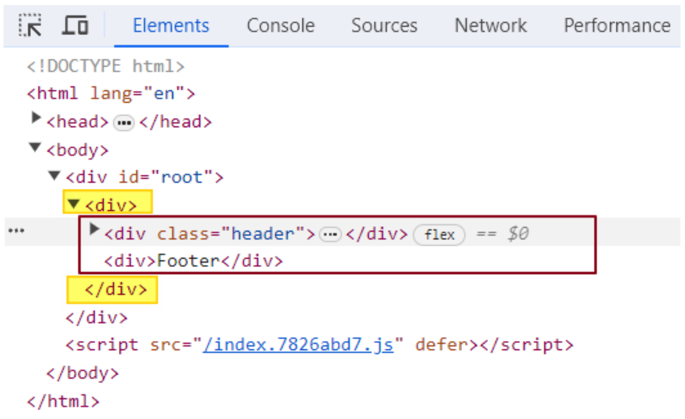
Creating Footer component –



```jsx
// Footer

const Footer = () => {
  return <div>Footer</div>;
};
const AppLayout = () => {
  return (
    <div>
      <Header />
      <Footer />
    </div>
  );
};
```

JSX expressions must have one parent element

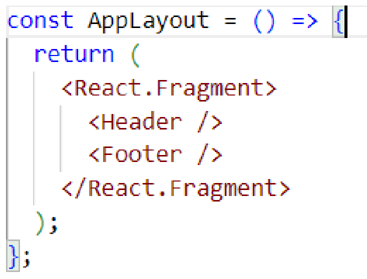
In AppLayout component we have two JSX expressions. One expression is Header component JSX and another one is Footer component JSX. These two expressions are enclosed with in a div element. Therefore, when these expressions are rendered in the DOM tree, the div element node also get added. To eliminate the extra div node react fragments are introduced.



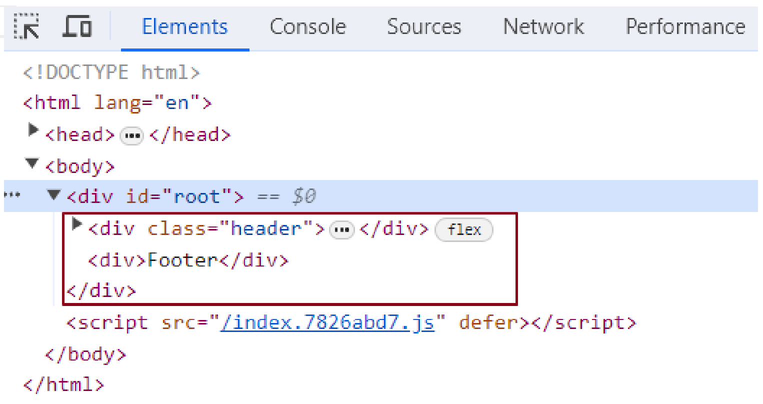
```
<!DOCTYPE html>
<html lang="en">
 ▶<head> … </head>
 ▼<body>
   ▼<div id="root">
     ▼<div>
       ▶<div class="header"> … </div> flex == $0
        <div>Footer</div>
      </div>
    </div>
    <script src="/index.7826abd7.js" defer></script>
  </body>
</html>
```

What is React Fragment?

React.Fragment is a component exported by react library. React fragment groups a list of children without adding an extra div node to the DOM.IN our case react fragment is grouping header and footer elements and rendering them in DOM tree without any extra node.



```
const AppLayout = () => {
  return (
    <React.Fragment>
      <Header />
      <Footer />
    </React.Fragment>
  );
};
```



```
<!DOCTYPE html>
<html lang="en">
 ▶<head> … </head>
 ▼<body>
   ▼<div id="root"> == $0
     ▶<div class="header"> … </div> flex
      <div>Footer</div>
    </div>
    <script src="/index.7826abd7.js" defer></script>
  </body>
</html>
```

Instead of React.Fragment we can also use a short hand syntax <></>

We can't add styling to these empty angular braces.

How can we achieve styling in React?

There are 3 ways

* Using inline styling

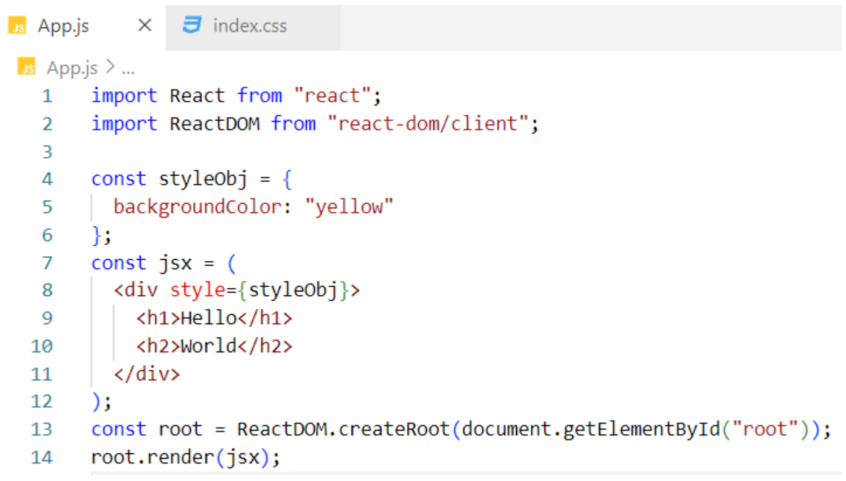
* Using external css

* Using external libraries like Tailwind, Bootstrap etc. (We will discuss this later)

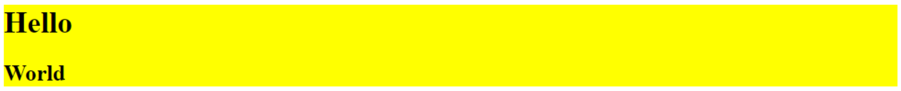
* Using Styled components. (We will discuss later)

What is Inline Styling in react?

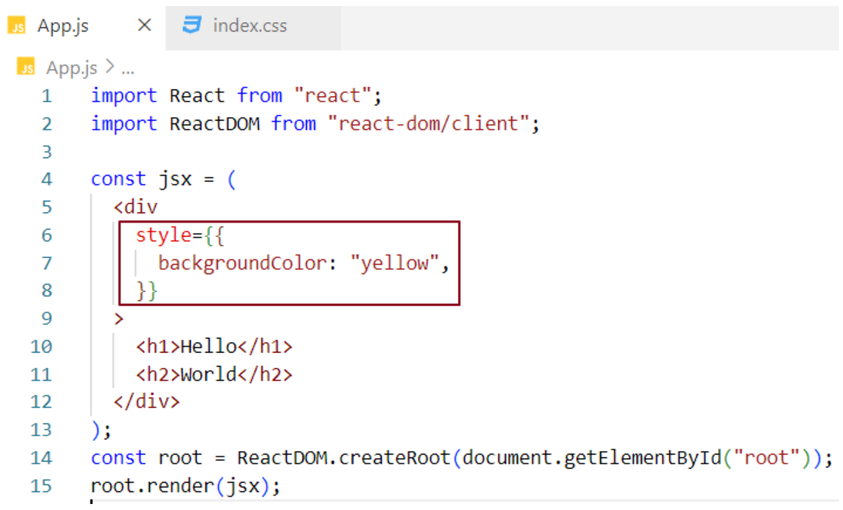
We create a JavaScript object where we add style properties. Then we use this JavaScript object inside style attribute of JSX with in curly braces.



```
import React from "react";
import ReactDOM from "react-dom/client";

const styleObj = {
  backgroundColor: "yellow"
};
const jsx = (
  <div style={styleObj}>
    <h1>Hello</h1>
    <h2>World</h2>
  </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(jsx);
```



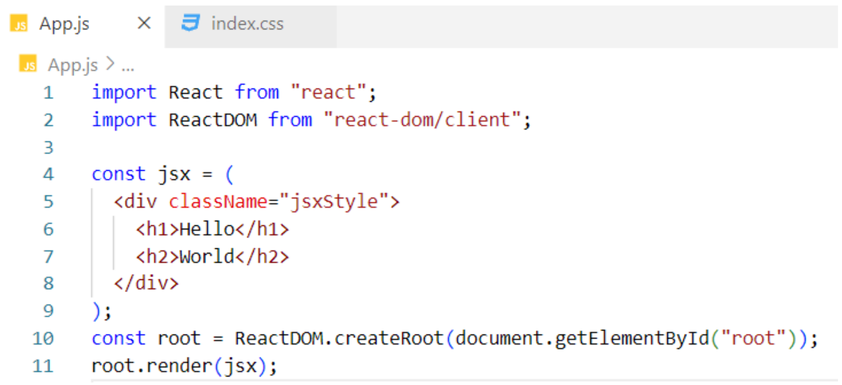
Alternate code –



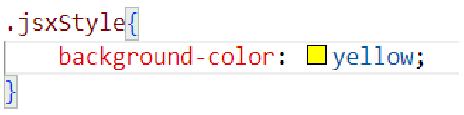
```
import React from "react";
import ReactDOM from "react-dom/client";

const jsx = (
  <div
    style={{
      backgroundColor: "yellow",
    }}
  >
    <h1>Hello</h1>
    <h2>World</h2>
  </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(jsx);
```

What is External Styling in react?

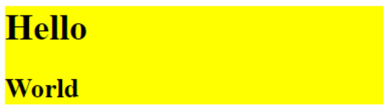
Give a className to the JSX element. add css for that class in CSS file.



```
import React from "react";
import ReactDOM from "react-dom/client";

const jsx = (
  <div className="jsxStyle">
    <h1>Hello</h1>
    <h2>World</h2>
  </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(jsx);
```

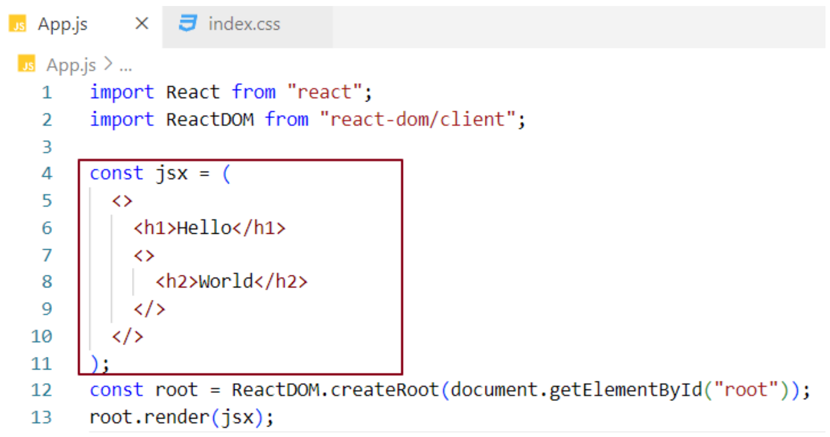


```
.jsxStyle{
    background-color: yellow;
}
```



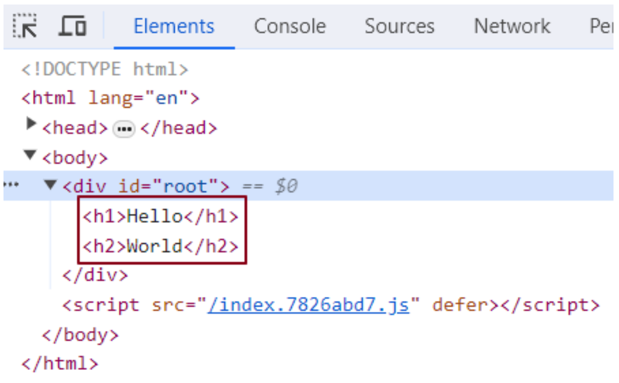
Can I use React.Fragment inside another React.Fragment ?

Yes, we can.



```
import React from "react";
import ReactDOM from "react-dom/client";

const jsx = (
  <>
    <h1>Hello</h1>
    <>
      <h2>World</h2>
    </>
  </>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(jsx);
```

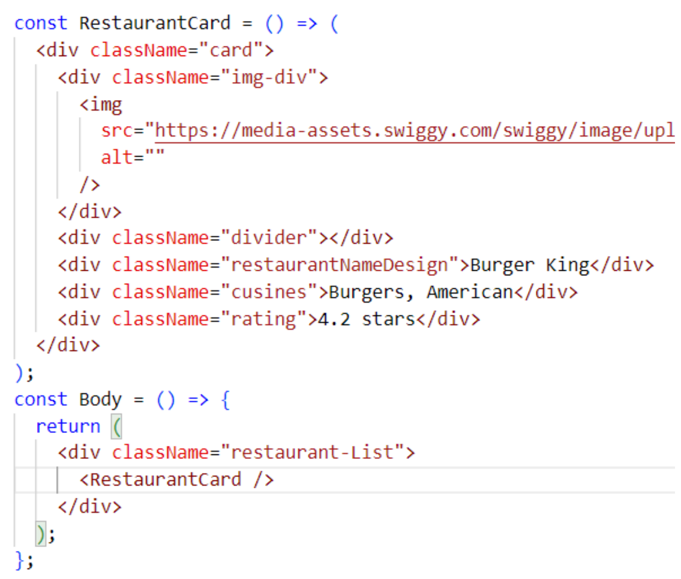
Empty angular tags are not rendered in DOM. These are only meant for grouping children.



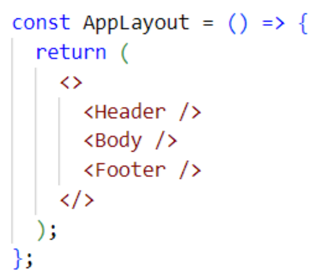
```
<!DOCTYPE html>
<html lang="en">
  <head> … </head>
  <body>
    <div id="root"> == $0
      <h1>Hello</h1>
      <h2>World</h2>
    </div>
    <script src="/index.7826abd7.js" defer></script>
  </body>
</html>
```

Creating body component –

We need to show list of restaurant cards component in body component. Let’s create a single card component with hardcoded data.

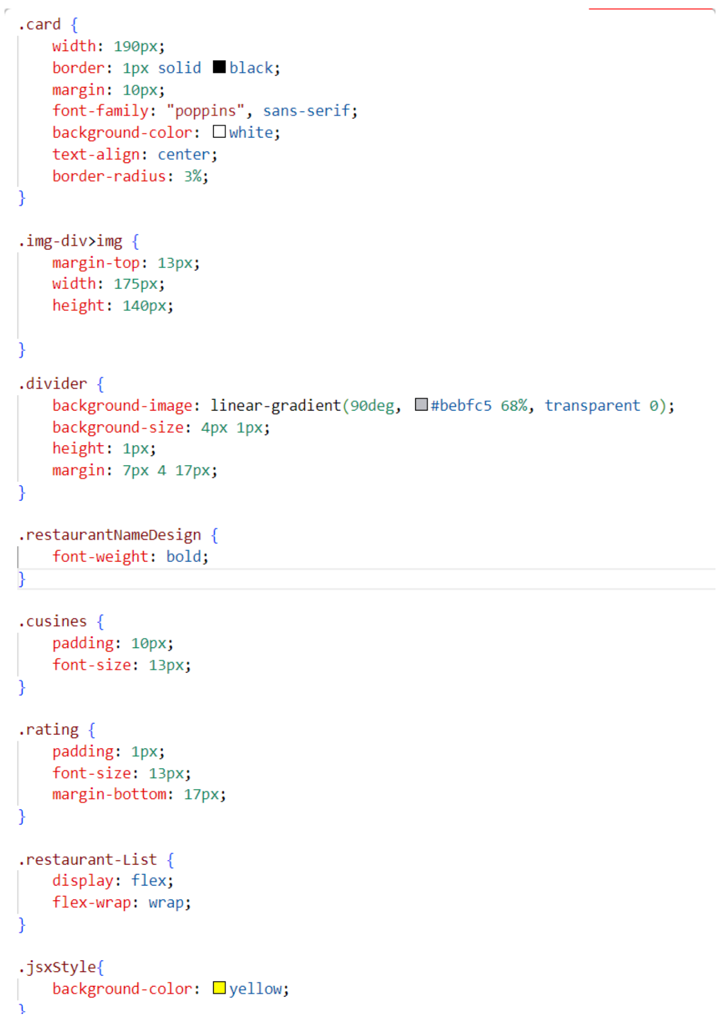


```
const RestaurantCard = () => (
  <div className="card">
    <div className="img-div">
      <img
        src="https://media-assets.swiggy.com/swiggy/image/upl
        alt=""
      />
    </div>
    <div className="divider"></div>
    <div className="restaurantNameDesign">Burger King</div>
    <div className="cusines">Burgers, American</div>
    <div className="rating">4.2 stars</div>
  </div>
);
const Body = () => {
  return (
    <div className="restaurant-List">
      <RestaurantCard />
    </div>
  );
};
```

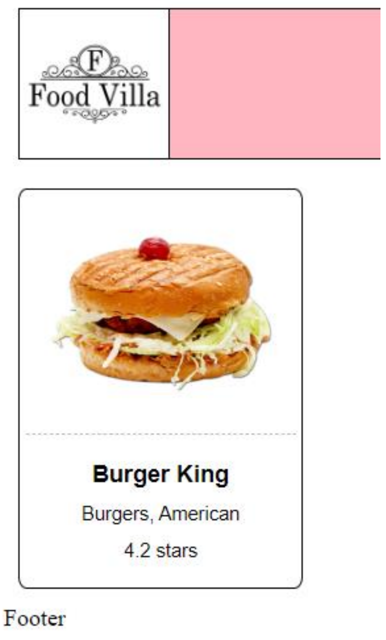


```
const AppLayout = () => {
  return (
    <>
      <Header />
      <Body />
      <Footer />
    </>
  );
};
```

Body Component css -



```
.card {
    width: 190px;
    border: 1px solid ■black;
    margin: 10px;
    font-family: "poppins", sans-serif;
    background-color: □white;
    text-align: center;
    border-radius: 3%;
}

.img-div>img {
    margin-top: 13px;
    width: 175px;
    height: 140px;

}
.divider {
    background-image: linear-gradient(90deg, ■#bebfc5 68%, transparent 0);
    background-size: 4px 1px;
    height: 1px;
    margin: 7px 4 17px;
}

.restaurantNameDesign {
    font-weight: bold;
}

.cusines {
    padding: 10px;
    font-size: 13px;
}

.rating {
    padding: 1px;
    font-size: 13px;
    margin-bottom: 17px;
}

.restaurant-List {
    display: flex;
    flex-wrap: wrap;
}

.jsxStyle{
    background-color: □yellow;
}
```



Burger King

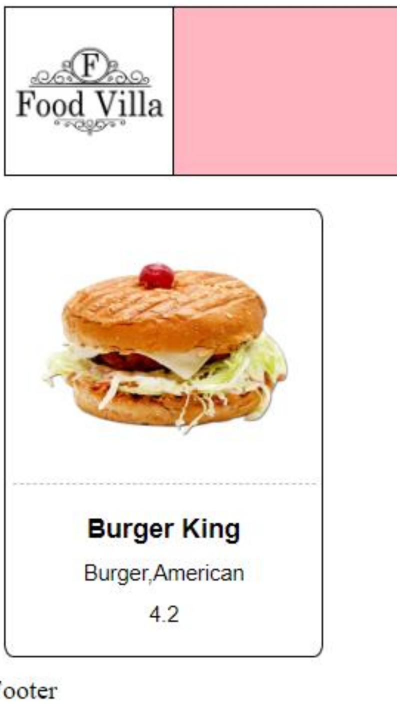
Burgers, American

4.2 stars

Footer

As we already know inside JSX we can use JS expression. Let's create a JavaScript object burgerKing and use it inside RestaurantCard component. We can use dot notation to access JavaScript object property inside JSX within a pair of curly braces.

```
const burgerKing = {
  name: "Burger King",
  image:
    "https://media-assets.swiggy.com/swiggy/image/up
  cusines: ["Burger", "American"],
  rating: "4.2",
};

const RestaurantCard = () => (
  <div className="card">
    <div className="img-div">
      <img src={burgerKing.image} alt="" />
    </div>
    <div className="divider"></div>
    <div>
      <div className="restaurantNameDesign">
        {burgerKing.name}
      </div>
      <div className="cusines">
        <span>{burgerKing.cusines.join(",")}</span>
      </div>
      <div className="rating">
        {burgerKing.rating}
      </div>
    </div>
  </div>
);
```

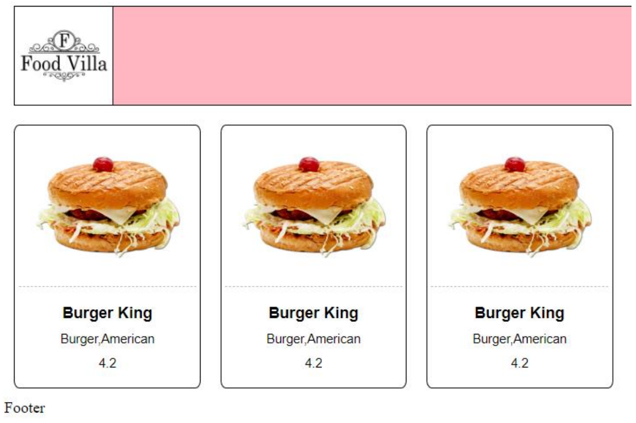


In real world application, we see list of different card components with different data.

We can reuse RestaurantCard component to display as many as the cards we expect to see in the UI.

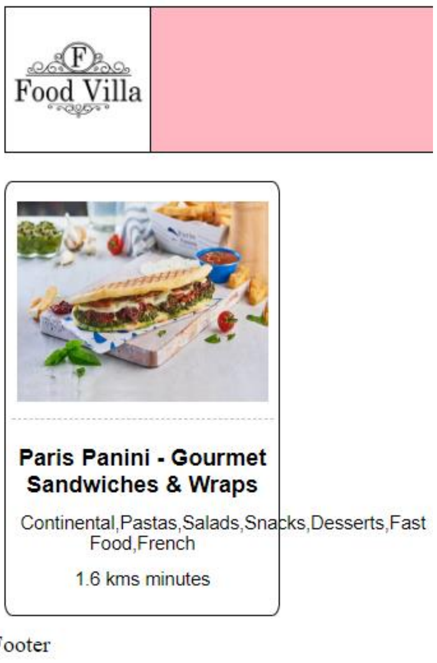
Let's simulate this behaviour.

```
> const burgerKing = {···
  };

> const RestaurantCard = () => (···
  );

  const Body = () => {
    return (
      <div className="restaurant-List">
        <RestaurantCard />
        <RestaurantCard />
        <RestaurantCard />
      </div>
    );
  };
```



Let’s get real time restaurant list data from the swiggy API and display a restaurant card component in UI.

```
// Fetching real data from swiggy API and storing the restaurant list in a const js variable restaurantList

const restaurantList = [ ···
];

// Accessing restaurantList data inside RestaurantCard component using {}

const RestaurantCard = () => (
  <div className="card">
    <div className="img-div">
      <img
        src={
          "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/" +
          restaurantList[0]?.data?.data?.cloudinaryImageId
        }
        alt=""
      />
    </div>
    <div className="divider"></div>
    <div>
      <div className="restaurantNameDesign">
        {restaurantList[0]?.data?.data?.name}
      </div>
      <div className="cusines">
        <span>{restaurantList[0]?.data?.data?.cuisines.join(",")}</span>
      </div>
      <div className="rating">
        {restaurantList[0]?.data?.data?.lastMileTravelString} minutes{" "}
      </div>
    </div>
  </div>
);
```


Food Villa

Paris Panini - Gourmet Sandwiches & Wraps

Continental,Pastas,Salads,Snacks,Desserts,Fast Food,French

1.6 kms minutes

Footer

Optional chaining (?.)

The optional chaining (?.) operator accesses an object's property or calls a function.

If the object accessed or function called using this operator is undefined or null, the expression short circuits and evaluates to undefined instead of throwing an error.

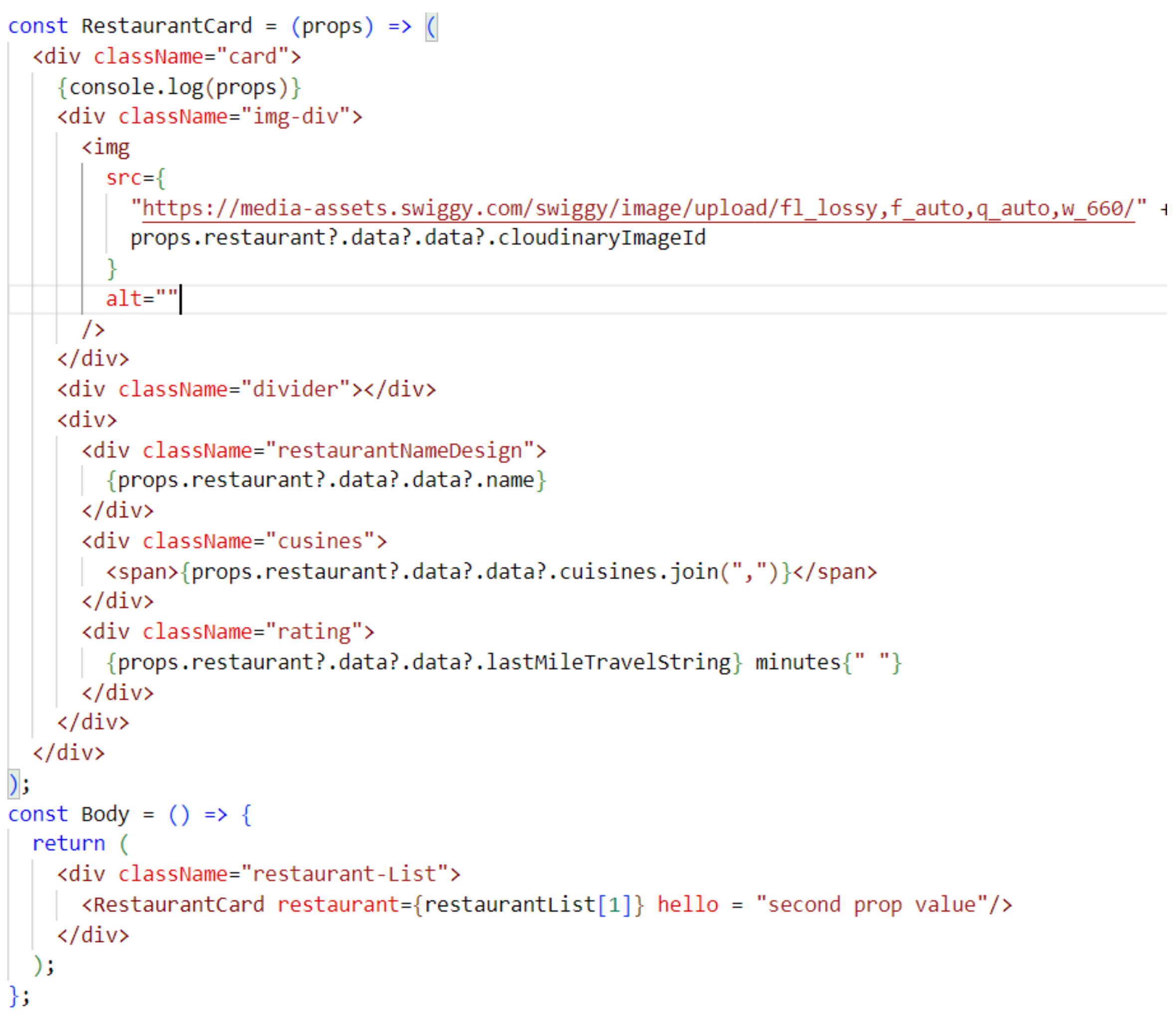
We could duplicate card components inside body component. but we need different data to be displayed in different card components. The data should be dynamic.

To show dynamic data to all restaurant card components we need to pass them as a PROP from parent component to Child components.

In our case all restaurant components are called inside parent Body component. To each restaurant card component, we need to pass dynamic data as a prop so that the prop will be received by the child restaurant card component.

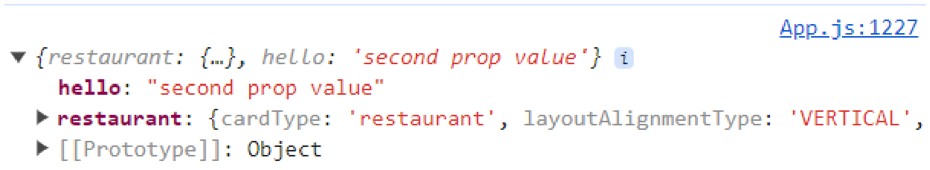
What are props?

Props stands for properties. passing props to a component is same as passing arguments to a JavaScript function. whatever properties we pass into the component react wraps them up into a JavaScript object.

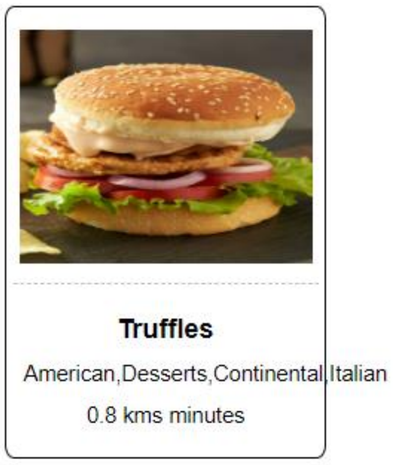


```
const RestaurantCard = (props) => (
  <div className="card">
    {console.log(props)}
    <div className="img-div">
      <img
        src={
          "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/" +
          props.restaurant?.data?.data?.cloudinaryImageId
        }
        alt=""
      />
    </div>
    <div className="divider"></div>
    <div>
      <div className="restaurantNameDesign">
        {props.restaurant?.data?.data?.name}
      </div>
      <div className="cusines">
        <span>{props.restaurant?.data?.data?.cuisines.join(",")}</span>
      </div>
      <div className="rating">
        {props.restaurant?.data?.data?.lastMileTravelString} minutes{" "}
      </div>
    </div>
  </div>
);
const Body = () => {
  return (
    <div className="restaurant-List">
      <RestaurantCard restaurant={restaurantList[1]} hello = "second prop value"/>
    </div>
  );
};
```

Here we are passing two properties hello and restaurant from parent Body component to child Restaurant card component. As we see in the console they are wrapped inside an object.



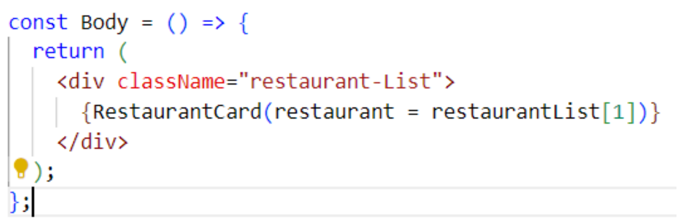
```
                                                              App.js:1227
▼ {restaurant: {…}, hello: 'second prop value'} i
    hello: "second prop value"
  ▶ restaurant: {cardType: 'restaurant', layoutAlignmentType: 'VERTICAL',
  ▶ [[Prototype]]: Object
```



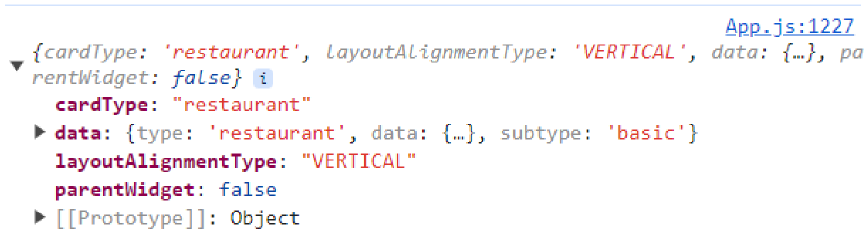
Footer

How can we not put these properties inside an object?

We will use function call syntax.



```
const Body = () => {
  return (
    <div className="restaurant-List">
      {RestaurantCard(restaurant = restaurantList[1])}
    </div>
  );
};
```

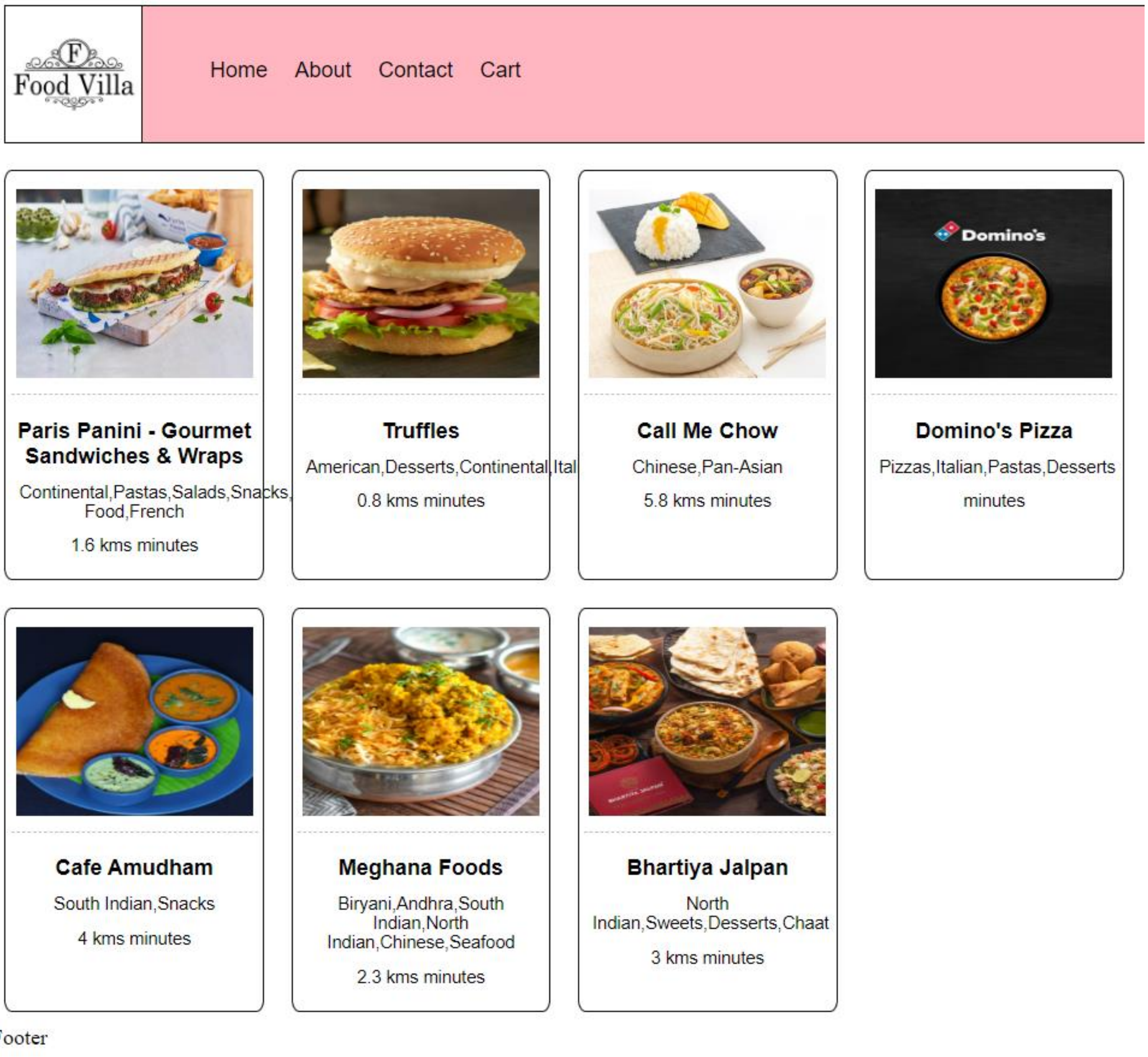


```
                                                    App.js:1227
▼ {cardType: 'restaurant', layoutAlignmentType: 'VERTICAL', data: {…}, pa
  rentWidget: false} i
    cardType: "restaurant"
  ▶ data: {type: 'restaurant', data: {…}, subtype: 'basic'}
    layoutAlignmentType: "VERTICAL"
    parentWidget: false
  ▶ [[Prototype]]: Object
```

This is a restaurant object. Not restaurant object wrapped inside another object Now child component receives props value directly instead of value wrapped up inside an object.

Passing dynamic data to multiple card components from parent Body component -

```
const Body = () => {
  return (
    <div className="restaurant-List">
      <RestaurantCard restaurant={restaurantList[0]} />
      <RestaurantCard restaurant={restaurantList[1]} />
      <RestaurantCard restaurant={restaurantList[2]} />
      <RestaurantCard restaurant={restaurantList[3]} />
      <RestaurantCard restaurant={restaurantList[4]} />
      <RestaurantCard restaurant={restaurantList[5]} />
      <RestaurantCard restaurant={restaurantList[6]} />
    </div>
  );
};
```



Paris Panini - Gourmet Sandwiches & Wraps
Continental,Pastas,Salads,Snacks,Food,French
1.6 kms minutes

Truffles
American,Desserts,Continental,Ital
0.8 kms minutes

Call Me Chow
Chinese,Pan-Asian
5.8 kms minutes

Domino's Pizza
Pizzas,Italian,Pastas,Desserts
minutes

Cafe Amudham
South Indian,Snacks
4 kms minutes

Meghana Foods
Biryani,Andhra,South Indian,North Indian,Chinese,Seafood
2.3 kms minutes

Bhartiya Jalpan
North Indian,Sweets,Desserts,Chaat
3 kms minutes

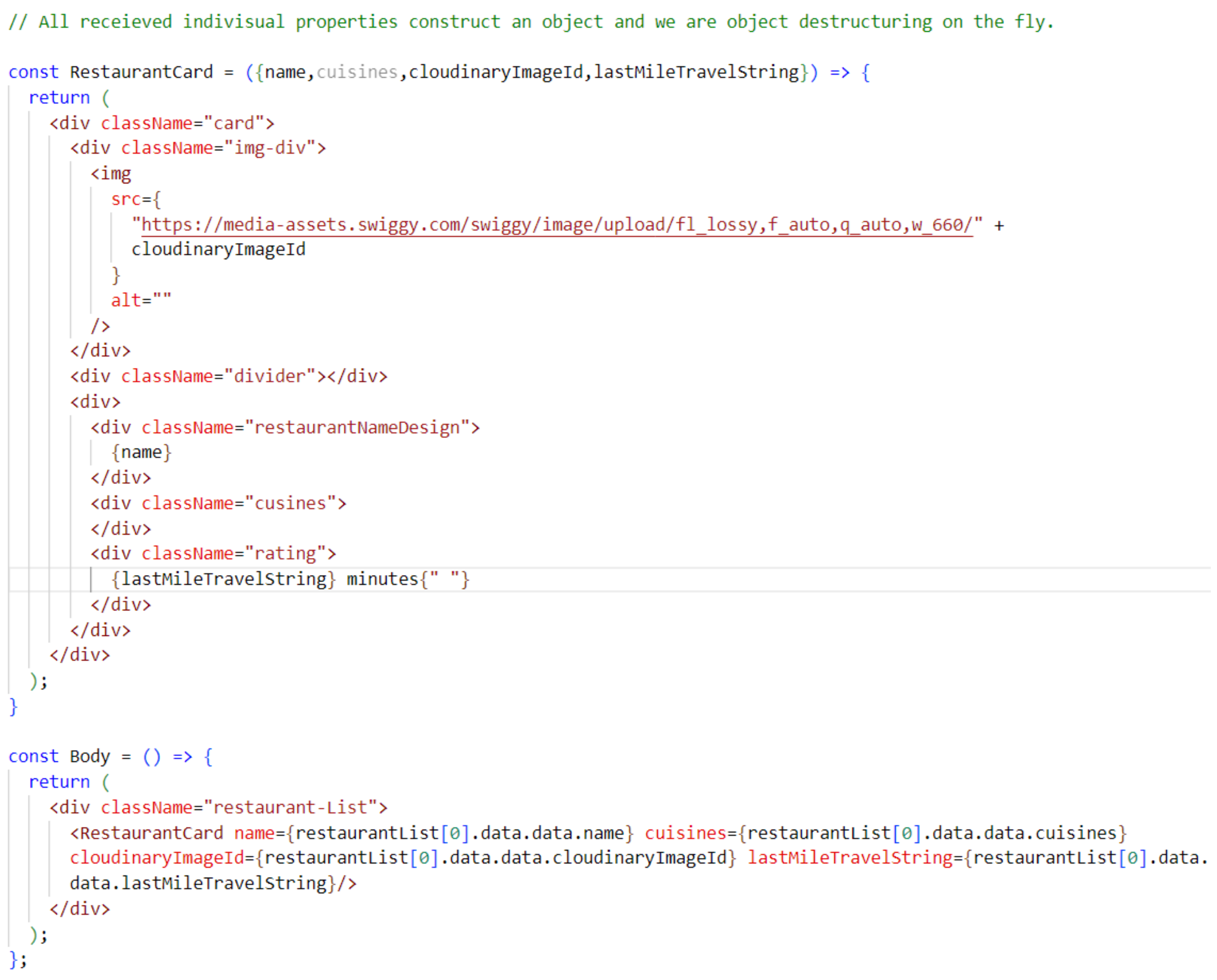
Footer

Props destructuring in child component –

```jsx
const RestaurantCard = (props) => {
  const { restaurant } = props;
  return (
    <div className="card">
      <div className="img-div">
        <img
          src={
            "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/" +
            restaurant?.data?.data?.cloudinaryImageId
          }
          alt=""
        />
      </div>
      <div className="divider"></div>
      <div>
        <div className="restaurantNameDesign">
          {restaurant?.data?.data?.name}
        </div>
        <div className="cusines">
          <span>{restaurant?.data?.data?.cuisines.join(",")}</span>
        </div>
        <div className="rating">
          {restaurant?.data?.data?.lastMileTravelString} minutes{" "}
        </div>
      </div>
    </div>
  );
};
```

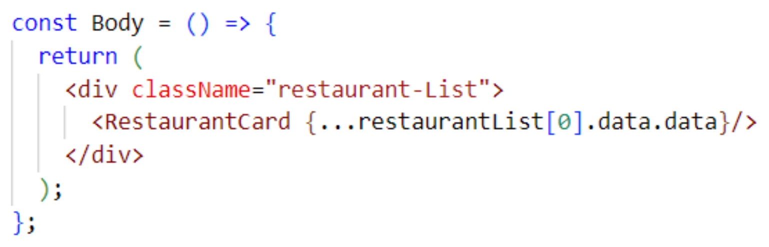
```jsx
const RestaurantCard = ({restaurant}) => {
  // destructuring restaurant
  const {name,cuisines,cloudinaryImageId,lastMileTravelString} =  restaurant.data.data
  return (
    <div className="card">
      <div className="img-div">
        <img
          src={
            "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/" +
            cloudinaryImageId
          }
          alt=""
        />
      </div>
      <div className="divider"></div>
      <div>
        <div className="restaurantNameDesign">
          {name}
        </div>
        <div className="cusines">
        {/* all restaurant does not have cuisines . we will end up doing undefined.join for those restaurants.Thus
        commenting this line */}
          {/* <span>{cuisines.join(",")}</span> */}
        </div>
        <div className="rating">
          {lastMileTravelString} minutes{" "}
        </div>
      </div>
    </div>
  );
}
```

props destructuring on the fly without using Spread operator –



```
// All receieved indivisual properties construct an object and we are object destructuring on the fly.

const RestaurantCard = ({name,cuisines,cloudinaryImageId,lastMileTravelString}) => {
  return (
    <div className="card">
      <div className="img-div">
        <img
          src={
            "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/" +
            cloudinaryImageId
          }
          alt=""
        />
      </div>
      <div className="divider"></div>
      <div>
        <div className="restaurantNameDesign">
          {name}
        </div>
        <div className="cusines">
        </div>
        <div className="rating">
          {lastMileTravelString} minutes{" "}
        </div>
      </div>
    </div>
  );
}

const Body = () => {
  return (
    <div className="restaurant-List">
      <RestaurantCard name={restaurantList[0].data.data.name} cuisines={restaurantList[0].data.data.cuisines}
      cloudinaryImageId={restaurantList[0].data.data.cloudinaryImageId} lastMileTravelString={restaurantList[0].data.
      data.lastMileTravelString}/>
    </div>
  );
};
```

props destructuring on the fly using Spread operator –



```
const Body = () => {
  return (
    <div className="restaurant-List">
      <RestaurantCard {...restaurantList[0].data.data}/>
    </div>
  );
};
```

...restaurantList[0].data.data is object destructuring using spread operator which generates name={restaurantList[0].data.data.name} cuisines={restaurantList[0].data.data.cuisines} cloudinaryImageId={restaurantList[0].data.data.cloudinaryImageId} lastMileTravelString={restaurantList[0].data.data.lastMileTravelString} and other key-value pairs

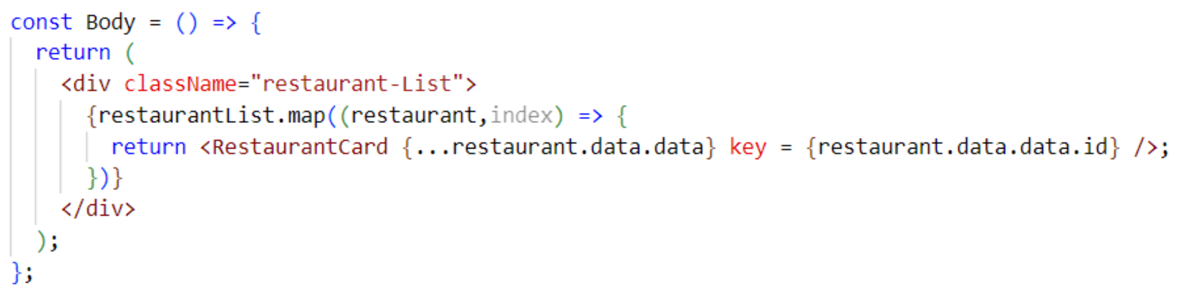
What is Spread Operator (…)?

Spread Operator spreads or expands an Iterable such as arrays and objects where multiple elements were expected.

HW- Explore more on spread and rest operator from JavaScript

What if we have 50 cards. calling 50 times RestaurantCard component is not ideal. We need to refactor the code. we need to use loop. let's use map. map is higher order JavaScript function. we will use map in JSX within {}.

In below code we are looping through each restaurant object in restaurant list and destructuring each restaurant data when calling the restaurant Card component



```
const Body = () => {
  return (
    <div className="restaurant-List">
      {restaurantList.map((restaurant,index) => {
        return <RestaurantCard {...restaurant.data.data} key = {restaurant.data.data.id} />;
      })}
    </div>
  );
};
```

We have built the basic template of header footer and body component, we will add search bar in body component in later chapter. Now let's discuss about What makes react fast.

Note – use rafce to create component instantly.

Why react is fast?

React is fast because react uses virtual DOM for DOM manipulation which is a very heavy task in other programming languages.

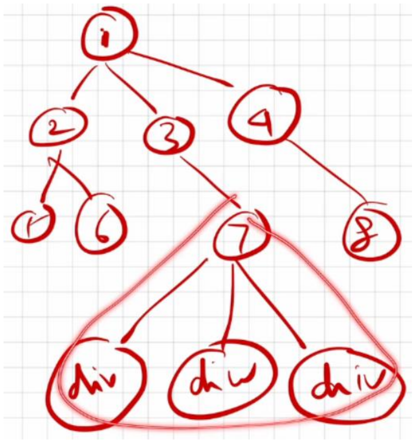
What is virtual DOM? Why do we need virtual DOM?

Virtual DOM is a concept where we keep a representation of actual DOM with us. We need virtual DOM for reconciliation in REACT.

What is Reconciliation in react?

Reconciliation is a process through which react updates browser DOM

Reconciliation internally uses diffing algorithm to find out the difference between Actual DOM Tree and Virtual DOM Tree and determines what needs to be updated in UI and it just updates the changed portion in the UI rather than updating the entire page. This makes react super-fast.

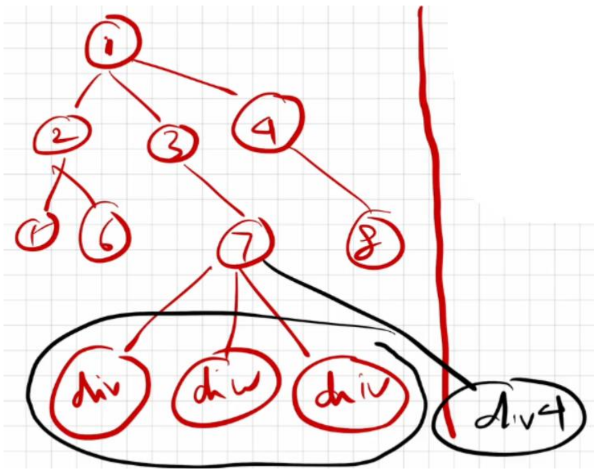


In this DOM tree, if something gets changed at 7th node, react does not re-render the entire DOM on the screen. React just rerenders updated portion in the UI using reconciliation. Diffing algorithm finds out this updated change.

What are Keys in React? Why do we need Keys in react?

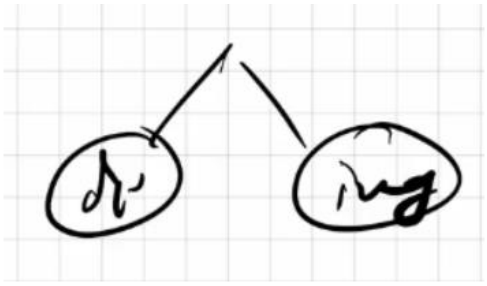
Keys are used in react to identify which items in the list are changed, updated or deleted. Keys are used to give an identity to the elements in the lists. react tracks elements by identifying their Key. Let's understand use cases of keys.

case 1 –



In this case, suppose div4 is added at 7th node. React will get confused because all of them are div. React does not know which div was added. React does not even know in which order div tags are there in the DOM.

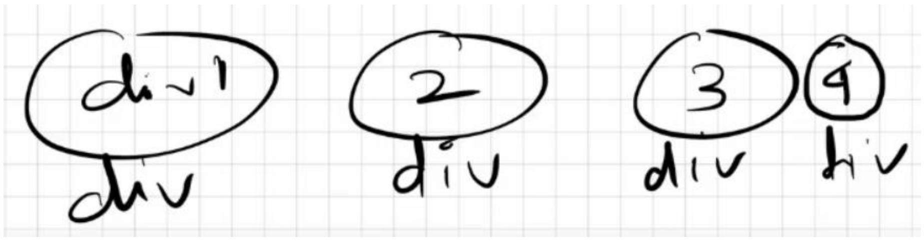
case 2 -



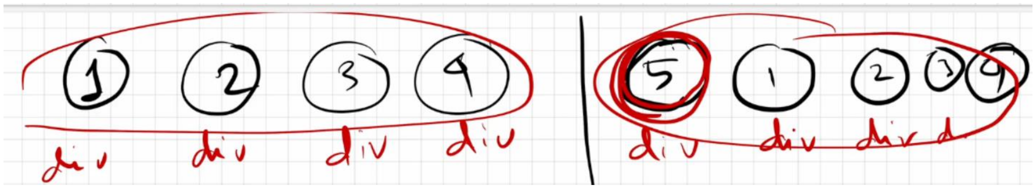
In this case If div is changed or Img tag is changed or if both of them were swapped, react can trace it.

case 3 -

whenever we have multiple children name with the same tags react cannot track them.



Why keys? A detailed explanation.

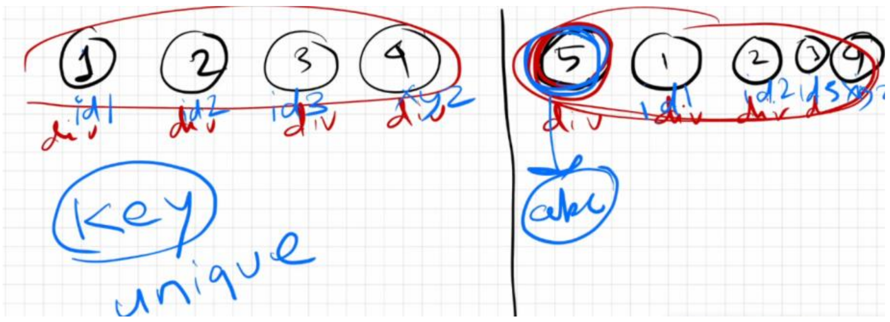


In this case, left hand side represents the Actual DOM. When we update anything in Actual DOM the change does not directly happen in the Actual DOM.

React creates a copy of Actual DOM which is called virtual DOM and then change takes place in the Virtual DOM. In our case Right hand side represents Virtual DOM.

Using Diffing algorithm react determined what was the change. We added an 5th div at the beginning of all 4 div children. But to react 5th div element is identical to rest of the 4 div elements. React does not know where to place the 5th div in the div sequence in the DOM. React is clueless about div elements placement order. In the end React rerenders all 5 div elements in DOM UI.

What we expected is react should render only the 5th element in the actual DOM. In order to do that react has to identify / track all the div elements. That's why keys were introduced so that react can track all these elements by their Keys.



Now we have provided keys to all div. So, react won't get confused this time.

When we added 5th div at the beginning of 4 children div, react used reconciliation Technique that in turn used diffing algorithm which differentiated actual and virtual DOM tree. keys were compared .4 keys were matched, 5th div key in VDOM could not get matched. And Finally react decided to render only this element in actual DOM.This render is a rerender phase.

How did react know where to put the 5th div?

In virtual DOM react checked where the 5th div is present. 5th div is present right before the first div. React tracked this by key name. Hence, react put 5th div before first div in actual DOM while re-rendering.

Is JSX mandatory for React?

No, JSX and React are two different things, they are not same. JSX is introduced in react because, this made developer life easy which enabled writing html like syntax in JavaScript. It's a syntactic sugar over pure react code.

Is ES6 mandatory for React?

No, without ES6 we can write react code using pure javascript.ES6 introduced some special features like spread operator, arrow function using which we can build our react application too.ES6 is recommended but not mandatory.

How to write comments in JSX?

JSX comments are written inside {/* */}. This is for both single or multiline comments.

```
Example -

{/* A JSX comment */}
{/*
  Multi
  line
  JSX
  comments
*/}
```

What is React Fibre? Use cases of React Fibre.

In older react versions (< 16), react was using Stack Reconciler Engine (SRE) for reconciliation. However, it had some drawbacks such as SRE is synchronous.

SRE works like a stack. SRE works synchronously until the stack is empty.

Let’s say we have an application, where we have a textbox in UI and the application is making API calls to fetch some data before we type anything in the textbox. Until we fetch the data, if we type anything, nothing will be printed. Once we get the data from the API then only we will be able to type something in the textbox. Meaning our app is behaving synchronously. To do both task in parallel asynchronous operation has to be performed.

To overcome such drawback after react version 16, React Fibre was introduced which uses updated react reconciler engine which is completely rewritten from the ground up and this engine performs asynchronous tasks.

Perks/Benefits -

* React Fibre focuses on Animations and responsiveness.
* Fibre makes react faster and smarter.
* Fibre splits work into chunks and prioritize tasks based on importance.
* Fibre resumes, pause and restarts rendering work on components as new updates come in.
* Fibre reuses previously completed work and even abort it if not needed.

Can we use index as keys in React?

Yes, we can use the index as keys, but it is not considered because adding indexes as keys can result in incorrect rendering or unexpected behaviour, and can even lead to performance issues.

Keys > index > no keys(unacceptable)

What is Config Driven UI?

Config Driven UI is based on the configurations of the data which an application receives. Config driven UI is handled from backend.

A great example would be swiggy app Image carousal. swiggy shows food offers based on geographical locations. Let’s say my home address location is Odisha and my friend’s home address is Mumbai. We both are accessing swiggy app from different places. I might get different food offers in my region, he may get different food offers in his region. Usually we see these offers in image carousal section of swiggy application. So, this image carousal section in swiggy app is config driven. Meaning we are seeing configured data in UI based on our locations.

These configurations are handled from backend. In some regions we do not get great offers, in that case no data was sent from backend to the UI, so people don’t see image carousal section at all