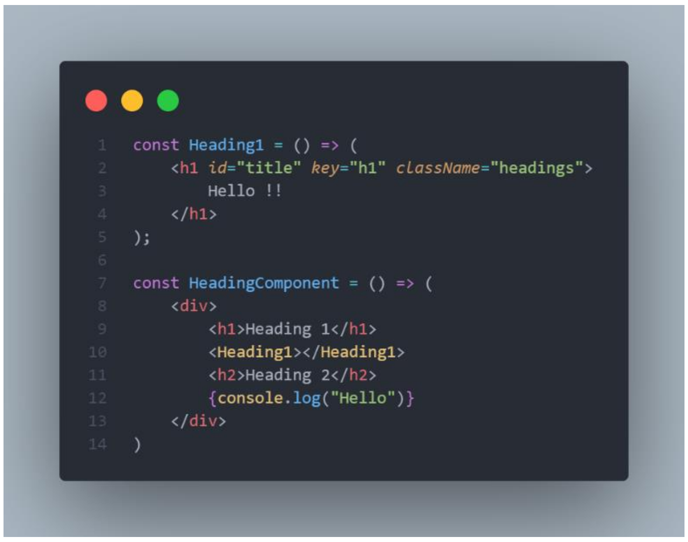
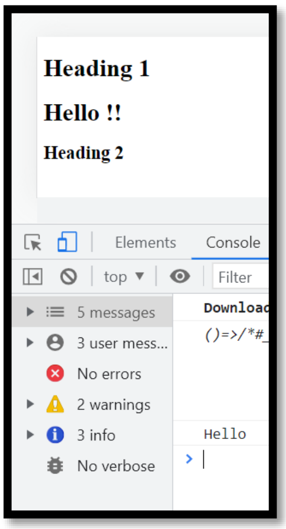
#NOTE :-

- There's another way of calling a functional component, other than <func_comp_name /> and {func_comp_name()}, it is by using it as a HTML tag i.e. **<funcCompName></funcCompName>.**
- If we do {console.log(100)} inside a JSX, that will also get rendered, and you will be able to see the output in the browser console.



```
const Heading1 = () => (
    <h1 id="title" key="h1" className="headings">
        Hello !!
    </h1>
);

const HeadingComponent = () => (
    <div>
        <h1>Heading 1</h1>
        <Heading1></Heading1>
        <h2>Heading 2</h2>
        {console.log("Hello")}
    </div>
)
```



Heading 1

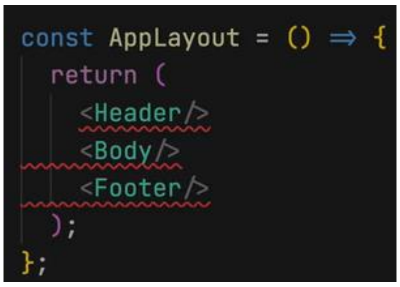
Hello !!

Heading 2

```
Download
()=>/*#
Hello
```

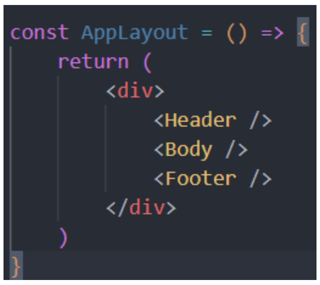
- Any piece of JSX component, should be wrapped in a parent element :-

Wrong



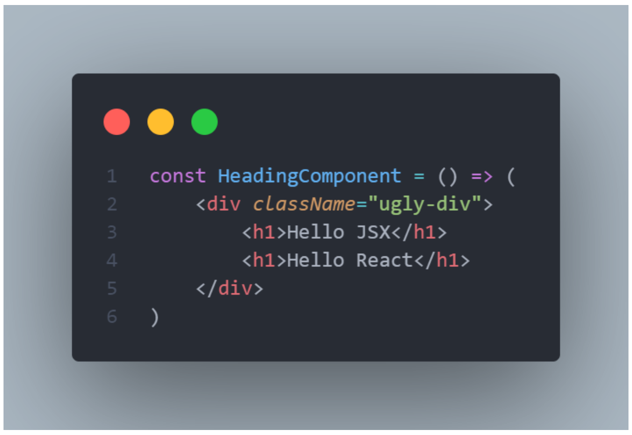
```
const AppLayout = () ⇒ {
  return (
    <Header/>
    <Body/>
    <Footer/>
  );
};
```

Right

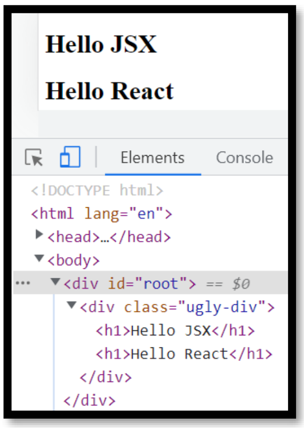


```
const AppLayout = () => {
    return (
        <div>
            <Header />
            <Body />
            <Footer />
        </div>
    )
}
```

- **React Fragment** :- Now, even though doing something like wrapping all the JSX elements inside a single parent works, it still looks ugly, because in root div, another unnecessary div (here named "ugly-div" is rendered) like :-



```
const HeadingComponent = () => (
    <div className="ugly-div">
        <h1>Hello JSX</h1>
        <h1>Hello React</h1>
    </div>
)
```

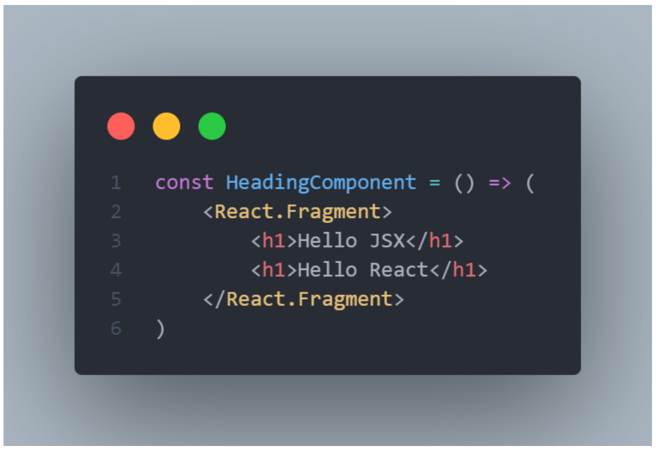


Hello JSX

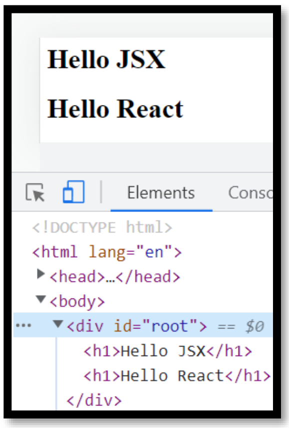
Hello React

```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
 ▼<body>
... ▼<div id="root"> == $0
     ▼<div class="ugly-div">
        <h1>Hello JSX</h1>
        <h1>Hello React</h1>
      </div>
    </div>
```

React library exports a component named React.Fragment and we can use this as a parent element to wrap like :-



```
const HeadingComponent = () => (
    <React.Fragment>
        <h1>Hello JSX</h1>
        <h1>Hello React</h1>
    </React.Fragment>
)
```

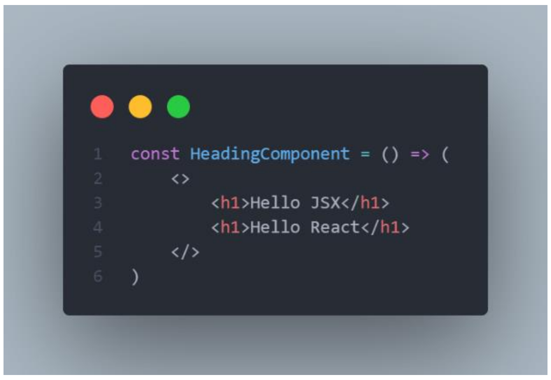


Hello JSX

Hello React

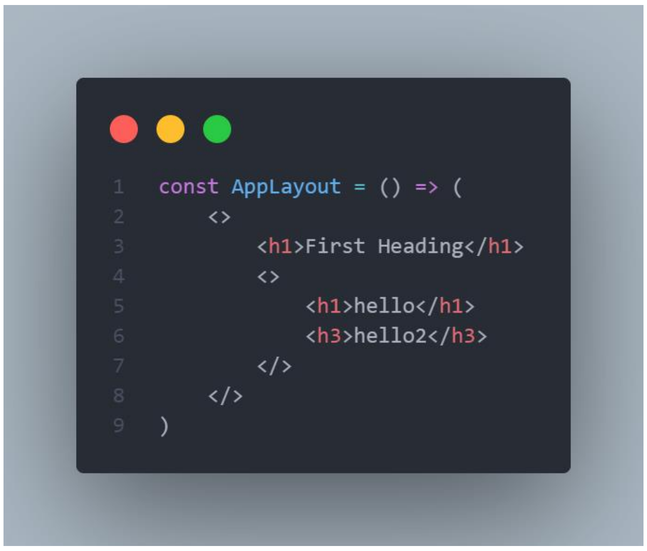
```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
 ▼<body>
... ▼<div id="root"> == $0
      <h1>Hello JSX</h1>
      <h1>Hello React</h1>
    </div>
```

- React.Fragments are treates as an empty tag. So, fb developers just said that instead of writing that long-ass name, write :- <>….</>, like :-

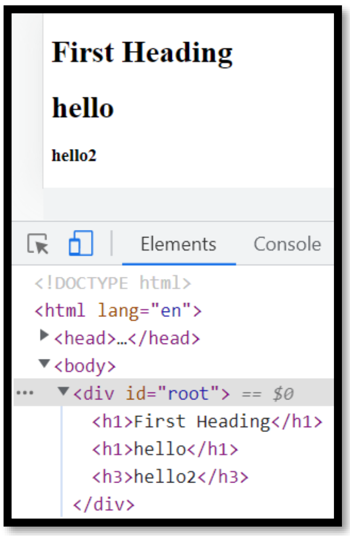


```
const HeadingComponent = () => (
    <>
        <h1>Hello JSX</h1>
        <h1>Hello React</h1>
    </>
)
```

- React Fragments can also be nested like :-



```
const AppLayout = () => (
    <>
        <h1>First Heading</h1>
        <>
            <h1>hello</h1>
            <h3>hello2</h3>
        </>
    </>
)
```



First Heading

hello

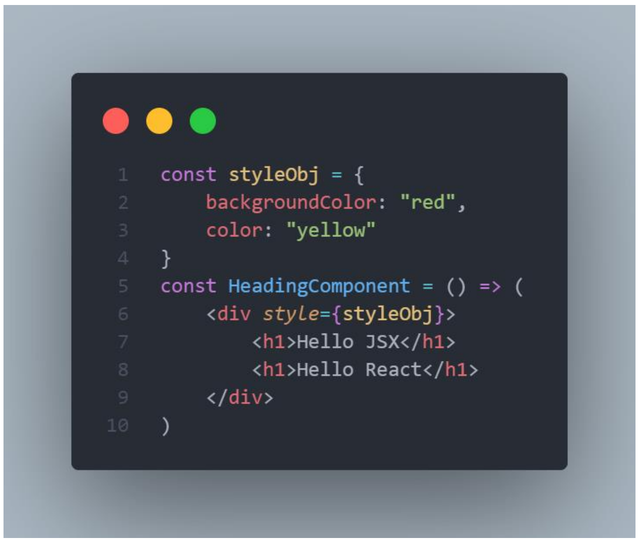
hello2

```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
 ▼<body>
…  ▼<div id="root"> == $0
      <h1>First Heading</h1>
      <h1>hello</h1>
      <h3>hello2</h3>
    </div>
```

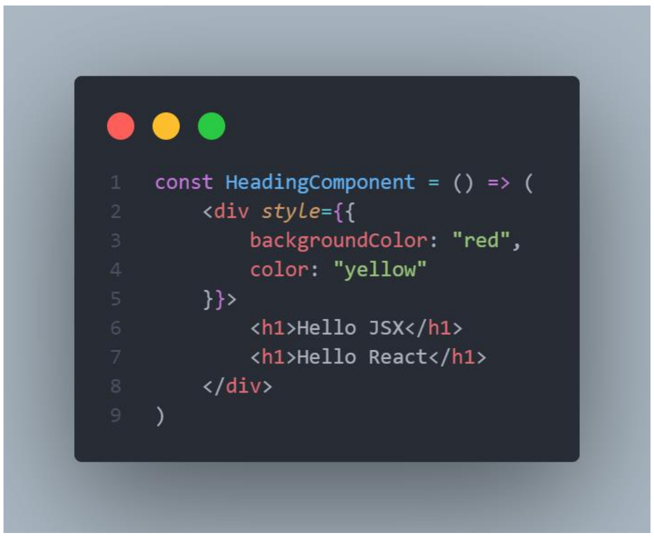
<u>CSS in JSX :-</u>

There are 2 ways of including CSS in JSX.

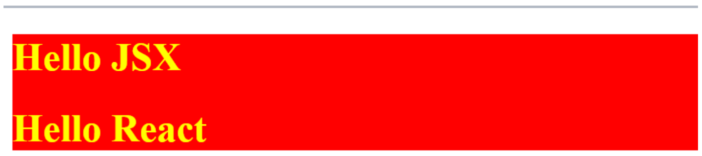
i. **<u>Inline styling</u>** is use the style attribute for that particular tag, but here instead of doing like <div style = "background: red; color: yellow"></div>, we have to pass an object to the style attribute. We know that object is a JS feature and any piece of JS should be wrapped inside " { } " in JSX and also since it's an object instead of using " ; " we should use ", " like ANY ONE of the below two :-



```
const styleObj = {
    backgroundColor: "red",
    color: "yellow"
}
const HeadingComponent = () => (
    <div style={styleObj}>
        <h1>Hello JSX</h1>
        <h1>Hello React</h1>
    </div>
)
```



```
const HeadingComponent = () => (
    <div style={{
        backgroundColor: "red",
        color: "yellow"
    }}>
        <h1>Hello JSX</h1>
        <h1>Hello React</h1>
    </div>
)
```



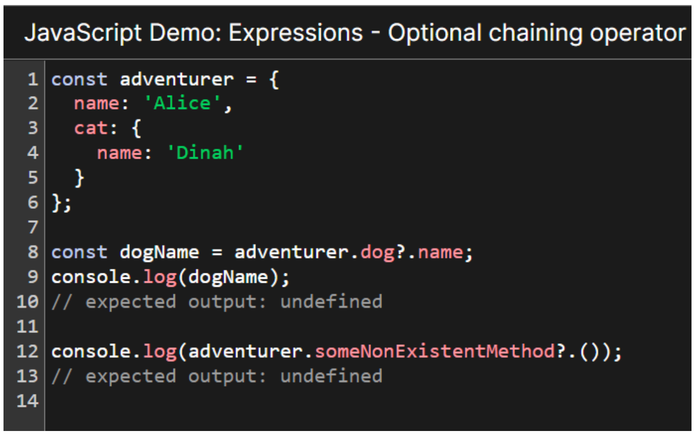
Hello JSX

Hello React

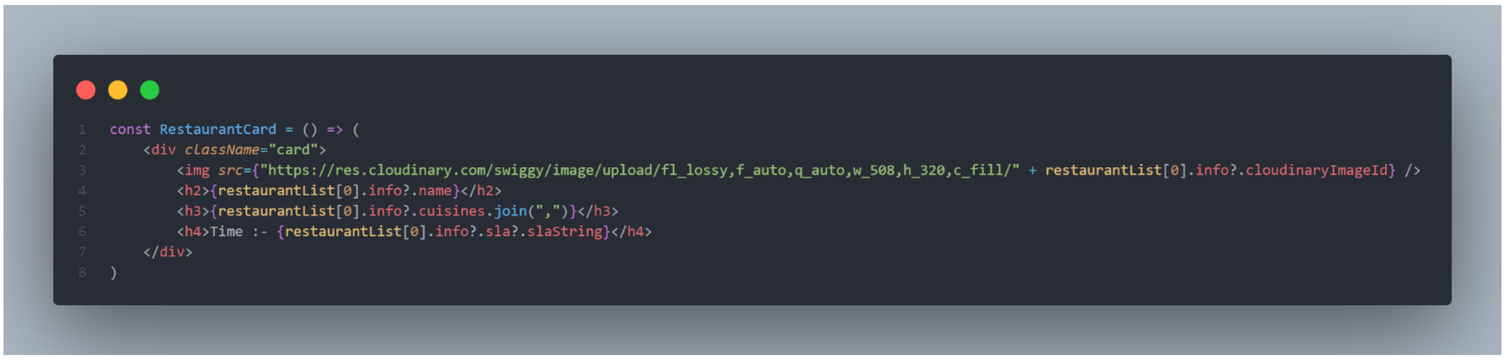
ii. **External styling** is the way of injecting CSS into an element by giving the element a className or id, and write its CSS in the CSS file

Optional Chaining :-

The optional chaining operator(?.) accesses an object's property or calls a function. If the object accessed r function called is undefined or null, it returns undefined without throwing an error. 1st example is in normal JS, 2nd example is using that operator in JS in JSX too.


JavaScript Demo: Expressions - Optional chaining operator

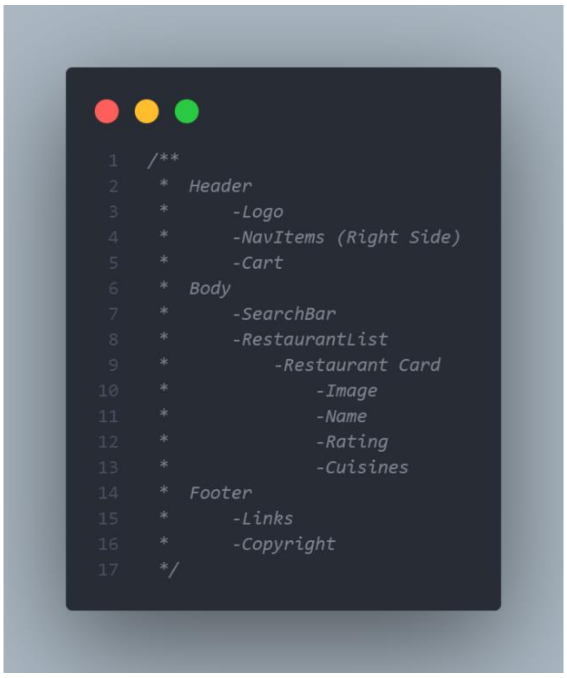
```
const adventurer = {
  name: 'Alice',
  cat: {
    name: 'Dinah'
  }
};

const dogName = adventurer.dog?.name;
console.log(dogName);
// expected output: undefined

console.log(adventurer.someNonExistentMethod?.());
// expected output: undefined

```



```
const RestaurantCard = () => (
    <div className="card">
        <img src={"https://res.cloudinary.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_508,h_320,c_fill/" + restaurantList[0].info?.cloudinaryImageId} />
        <h2>{restaurantList[0].info?.name}</h2>
        <h3>{restaurantList[0].info?.cuisines.join(",")}</h3>
        <h4>Time :- {restaurantList[0].info?.sla?.slaString}</h4>
    </div>
)
```

Building the Food Villa App

Layout :-

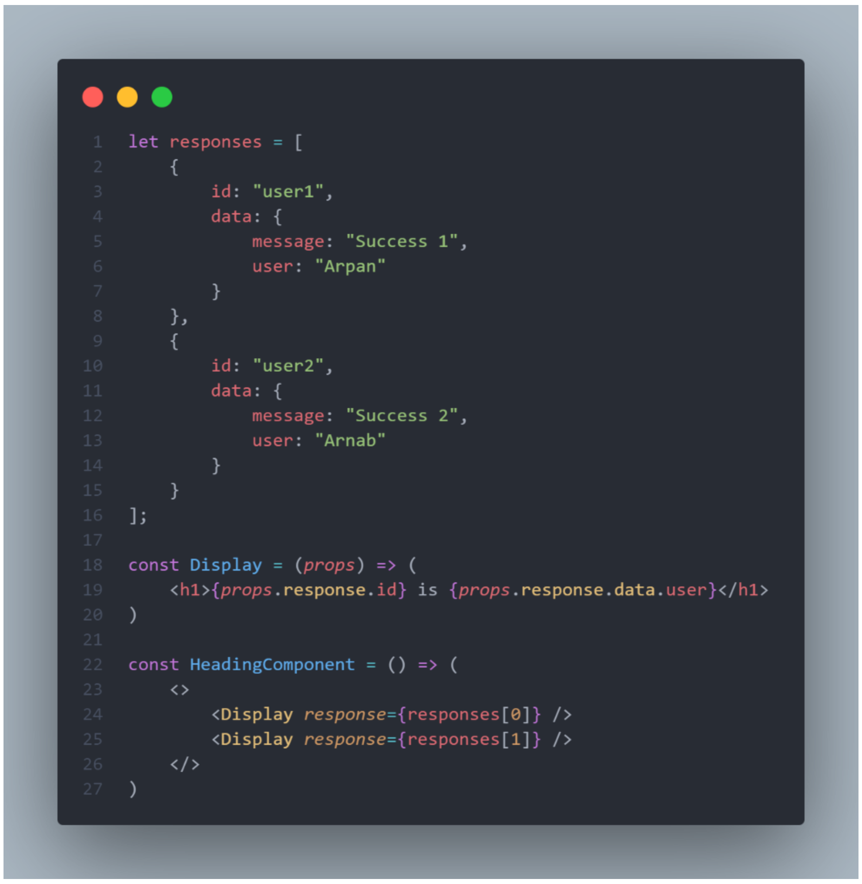


```
/**
 *  Header
 *      -Logo
 *      -NavItems (Right Side)
 *      -Cart
 *  Body
 *      -SearchBar
 *      -RestaurantList
 *          -Restaurant Card
 *              -Image
 *              -Name
 *              -Rating
 *              -Cuisines
 *  Footer
 *      -Links
 *      -Copyright
 */
```

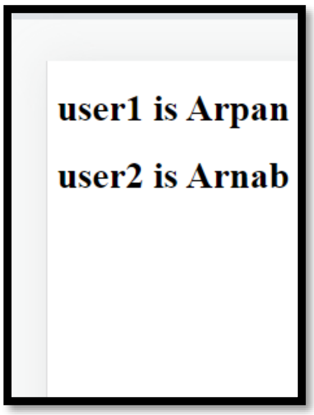
Config Driven UI and the fetching the data from Swiggy :- See the video from 1:31:30 to 1:56:56

Props :-   (Code is in Part_2)

Props (short for "Properties") in React are attributes passed to the components in JSX used to pass data. They are nothing but keys that hold some value. Also, React wraps the props along with the values associated, into an object named props, which is passed as an arguement to that specific functional component (The variable name "props" is just a convention, we can name it anything else too).

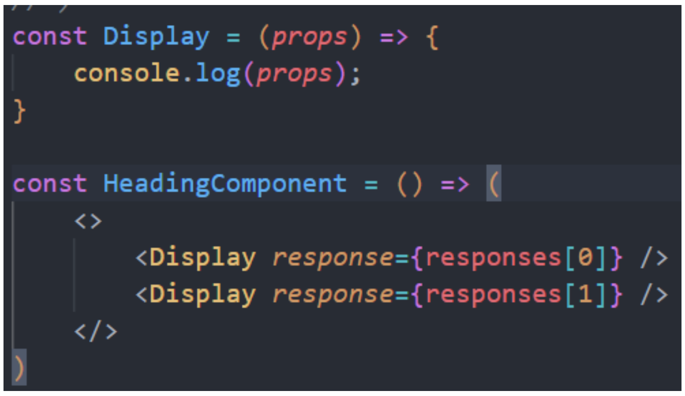


```jsx
let responses = [
    {
        id: "user1",
        data: {
            message: "Success 1",
            user: "Arpan"
        }
    },
    {
        id: "user2",
        data: {
            message: "Success 2",
            user: "Arnab"
        }
    }
];

const Display = (props) => (
    <h1>{props.response.id} is {props.response.data.user}</h1>
)

const HeadingComponent = () => (
    <>
        <Display response={responses[0]} />
        <Display response={responses[1]} />
    </>
)
```

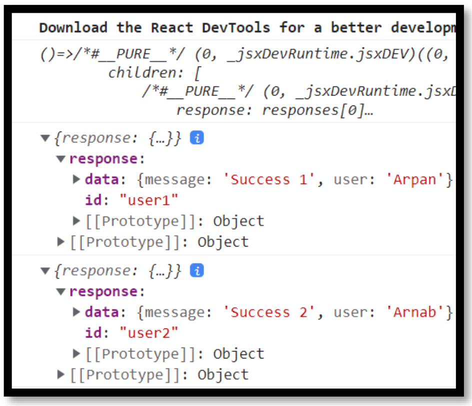


```
user1 is Arpan

user2 is Arnab
```

If you console.log the props object in the React component, you will see that it's just a JS object :-



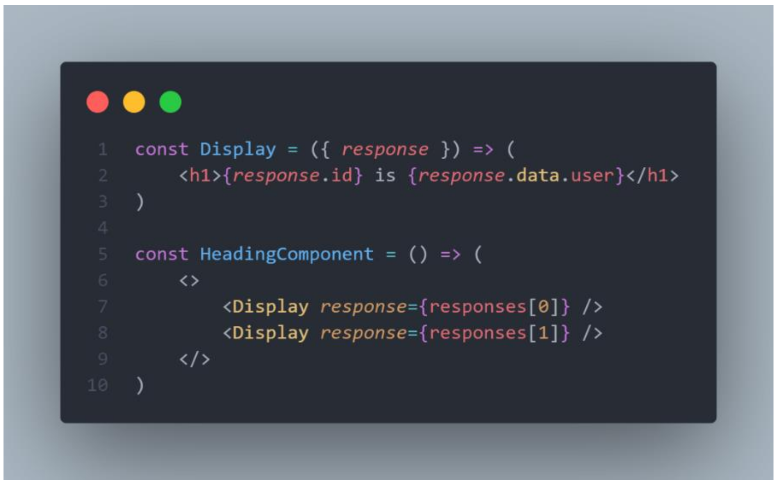
```jsx
const Display = (props) => {
    console.log(props);
}

const HeadingComponent = () => (
    <>
        <Display response={responses[0]} />
        <Display response={responses[1]} />
    </>
)
```



```
Download the React DevTools for a better developm
()=>/*#__PURE__*/ (0, _jsxDevRuntime.jsxDEV)((0,
        children: [
            /*#__PURE__*/ (0, _jsxDevRuntime.jsxD
                response: responses[0]…
▼{response: {…}} i
  ▼response:
    ▶data: {message: 'Success 1', user: 'Arpan'}
     id: "user1"
    ▶[[Prototype]]: Object
  ▶[[Prototype]]: Object
▼{response: {…}} i
  ▼response:
    ▶data: {message: 'Success 2', user: 'Arnab'}
     id: "user2"
    ▶[[Prototype]]: Object
  ▶[[Prototype]]: Object
```

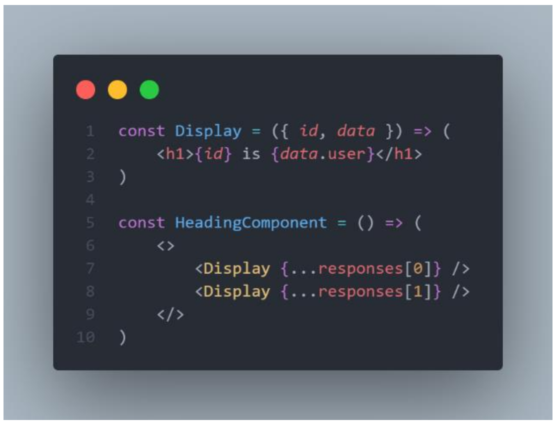
We can also pass props as parameters of React components, by destructuring it completely in the below ways :-

i. By destructuring it only in in the definition of component



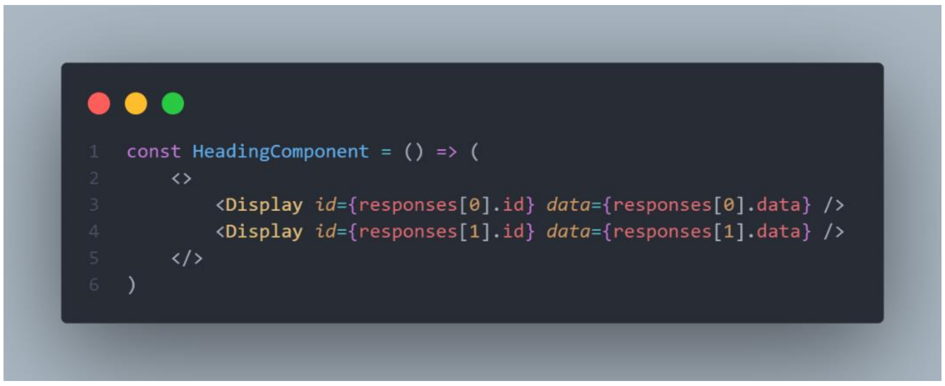
```jsx
const Display = ({ response }) => (
    <h1>{response.id} is {response.data.user}</h1>
)

const HeadingComponent = () => (
    <>
        <Display response={responses[0]} />
        <Display response={responses[1]} />
    </>
)
```

ii. By destructuring the props while calling the React component itself :-



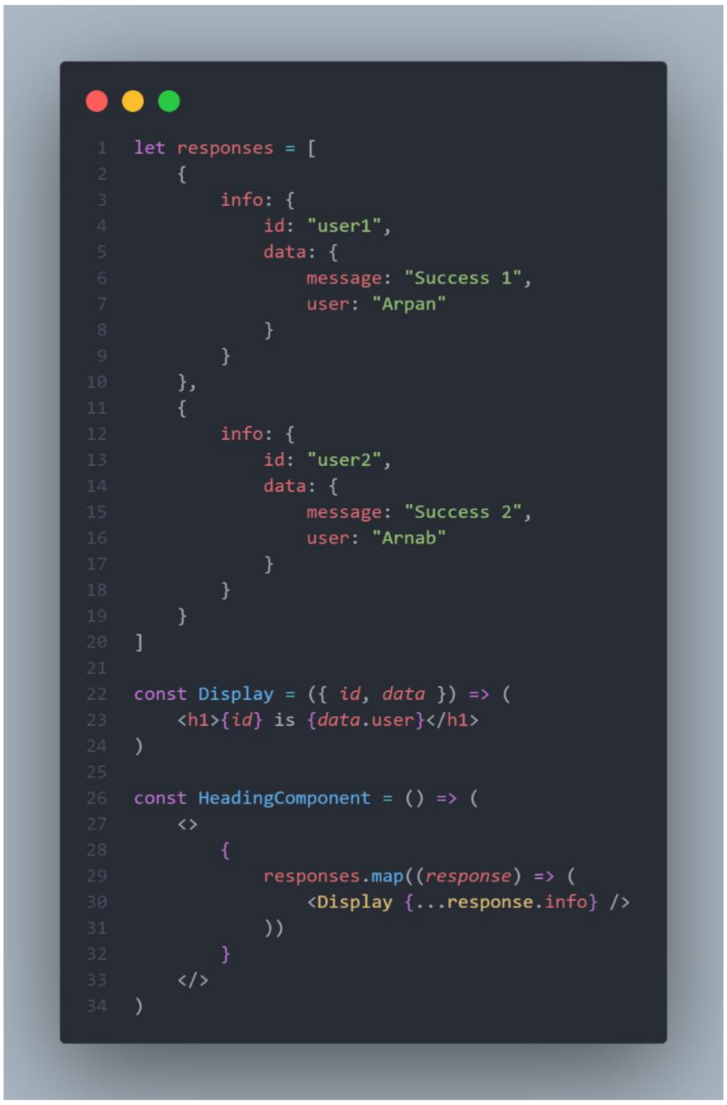
```
const Display = ({ id, data }) => (
    <h1>{id} is {data.user}</h1>
)

const HeadingComponent = () => (
    <>
        <Display {...responses[0]} />
        <Display {...responses[1]} />
    </>
)
```

From the above example, it is also evident that we can also pass multiple props and all the props will be passed as an object to the React component. So, instead of the above thing, we could have also passed the above props like :-



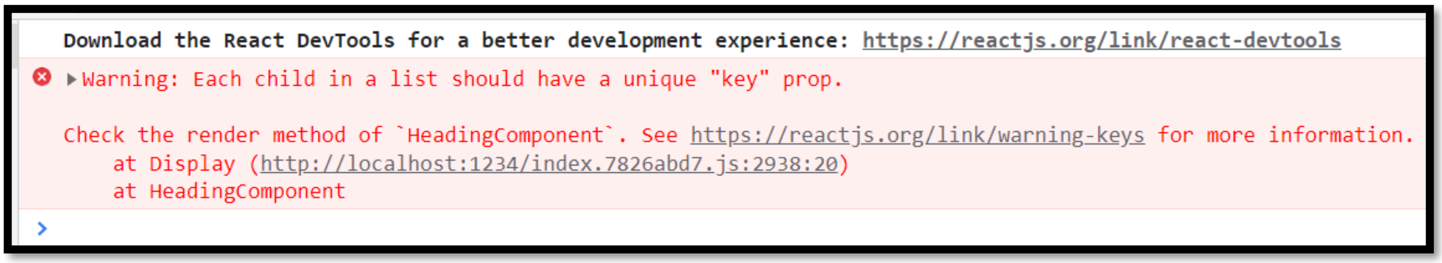
```
const HeadingComponent = () => (
    <>
        <Display id={responses[0].id} data={responses[0].data} />
        <Display id={responses[1].id} data={responses[1].data} />
    </>
)
```

Now, what we are doing is specifically mentioning eact of restaurant list objects one by one. Instead we can iterate over them by using the JS higher order functions like map()



```
let responses = [
    {
        info: {
            id: "user1",
            data: {
                message: "Success 1",
                user: "Arpan"
            }
        }
    },
    {
        info: {
            id: "user2",
            data: {
                message: "Success 2",
                user: "Arnab"
            }
        }
    }
]

const Display = ({ id, data }) => (
    <h1>{id} is {data.user}</h1>
)

const HeadingComponent = () => (
    <>
        {
            responses.map((response) => (
                <Display {...response.info} />
            ))
        }
    </>
)
```

But this displays a warning because we need to give unique keys to them so that React can differentiate between each React element. It doesn't display this warning when we were passing the array items one by one because somehow React's Diff or Fibre algo (discussed later) understood that those were different elements, but while using map, it does not.



```
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
▸Warning: Each child in a list should have a unique "key" prop.

Check the render method of `HeadingComponent`. See https://reactjs.org/link/warning-keys for more information.
    at Display (http://localhost:1234/index.7826abd7.js:2938:20)
    at HeadingComponent
>
```

Virtual DOM :-

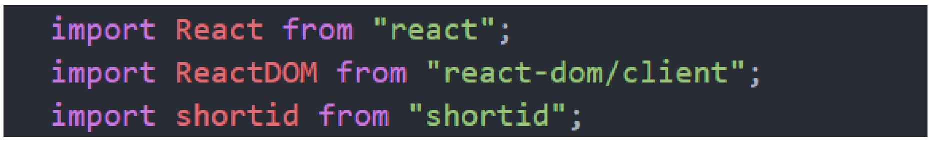
React keeps a representation of the actual DOM with it, which is called the Virtual DOM. This virtual DOM isn't just a concept of React, but other softwares too. We need Virtual DOM for Reconciliation in React (discussed earlier). In short, Reconciliation is an algorithm (called the Reconciliation/Diffing Algorithm) to diff one Virtual DOM tree from the other to determine what to change in out UI and what not to. This helps React to rerender only the changed portions in the children of our root element, instead of rerending all of them. This is why unique keys are used.

React Fibre :-

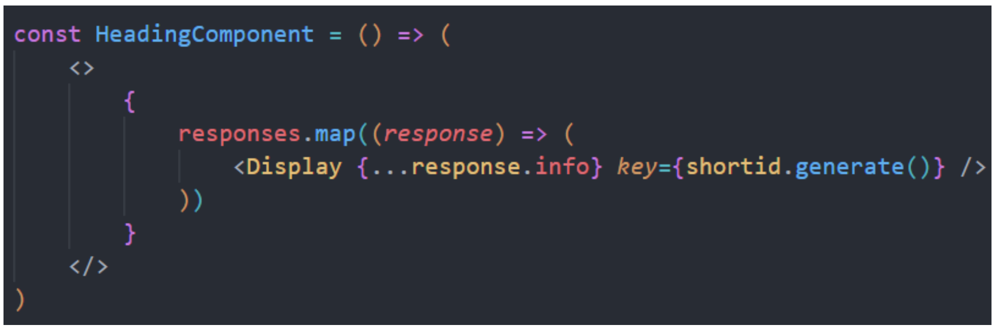
It is the new reconcilitation algo from React 16.
Read more about it from :-
Link1
Link2
(Just try to get the basic idea, no need to fuss about all the details)

Removing unique Key warning in map()

You can learn more about using map() in React from this Link. (If you could not find, then try search on google about "react index not used as key in map").
Now, to remove the warning we can use the unique IDs for each object in the array (if present) or else generate a new key by ourselves using an npm package named shorted (Read it from here link). Include the package in your app, then use it like :-



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



```
const HeadingComponent = () => (
    <>
        {
            responses.map((response) => (
                <Display {...response.info} key={shortid.generate()} />
            ))
        }
    </>
)
```

Now, you know that the map() function gives us another parameter "index" for each element of the array. But we should not use that index as our key. To know why, read :- Link