Chap 04 - Talk is Cheap, Show me the code

Date - 07/01/2023

"Building a Food-ordering App".

* Is JSX mandatory ? ⇒ NO
* Is Typescript mandatory ? ⇒ NO
* Is ES6 mandatory ? ⇒ NO

3 ways of component composition :-

1. {Title()}
2. <Title/> ⟶ used generally.
3. <Title></Title>

BUILDING OUR APP

Name : Food Villa

"Whenever you are writing code, do planning."

Our app look like this 

Food Villa

Nav-bar / Header

LOGO

Home About Support Cart

Body

Search

Image

Name of Restaurant
Cuisines
Star Rating

Image

Name of Restaurant
Cuisines
Star Rating

Image

Name of Restaurant
Cuisines
Star Rating

Image

Name of Restaurant
Cuisines
Star Rating

Footer

So, the app layout should have :-

* Header
* Body
* Footer.

```
Const AppLayout = () => {
  return (
    - Header
        - Logo
        - Nav Items (on right side)
        - Cart
    - Body
        - Searchbar
        - RestaurantList
            - RestaurantCard
                - Image
                - Name
                - Rating
                - Cusines
    - Footer
        - links
        - Copyrights
  )}
```

App.js

HEADER COMPONENT

For building header component :-

```
✡ Const Title = () => {
    <a href="/">
      <img
        alt="food_villa_logo"
        src="https://someurl"
      /></a>
  };

✡ Const HeaderComponent = () => {
    return (
      <div className="header">
        <Title/>
        <div className="nav-items">
          <ul>
            <li>Home</li>
            <li>About</li>
            <li>Contact</li>
            <li>Cart</li>
          </ul>
        </div>
      </div>
    );
  };
```

Note :-

※ Jsx expressions must have one parent element.

React.Fragment

→ is a component which is exported by 'React'.
[import React from "react";]

→ grouped a list of children without adding extra nodes to the DOM.

Eg :- import React from 'react';

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

→ Shorthand syntax <> </> is used instead of <React.Fragment> </React.Fragment>

* But, you can't pass styles to empty brackets.

Giving style inside React

To give inline styles in react, do :-

FIRST METHOD ↓

Eg :-

```
* const styleObj = {
      backgroundColor : "red",
  };
* const jsx = (
    <div style = {styleObj}>
      <h1> Hello </h1>
      <h2> World </h2>
    </div>
  );
```

styleObj is Normal JS object

inside this parantheis, you can write any piece of JS code.

→ In React, style is given using javascript objects

—▷ Alternative way :-

* const jsx = (

```
<div style = {{
    backgroundColor : "yellow";
}}>
<h1> Hello </h1>
<h2> World </h2>
</div>
);
```

SECOND METHOD ↴

—▷ give class name to the div (or whatever tag) and write CSS inside css file.

* const jsx = (

```
<div classname = "jsx" >
<h1> Hello </h1>
<h2> World </h2>
</div> );
```

* css file ↴

```
.jsx {
    backgroundColor : "blue";
}
```

);

THIRD METHOD ⬇

→ using external library like 'Tailwind css', 'Bootstrap', 'MaterialUI', etc

---

H.W → Can I use a 'React.Fragment' inside my 'React.Fragment' ?

A):- Yes, you can nest 'React.Fragment' components inside other 'React.Fragment' components.

Eg:-

```
import React from 'react';
const jsx = (
  <>
    <ChildA />
    <>
      <ChildB />
      <ChildC />
    </>
    <ChildD />
  </>
);
}
```

ChildB & C are siblings & will be grouped together without adding an extra node to the DOM.

BODY COMPONENT

While building restaurant cards, we need some data for this card. Ways :-

- Using hard coded data,
- integrate with api

Hard coded data - code will be like ↓

```
* const RestaurantCard = () => {
    return (
      <div classname = "card" >
        <img src = " https://some url" >
        <h2> Burger King </h2>
        <h3> Burgers, American </h3>
        <h4> 4.2 stars </h4>
      </div>
    ); };

* const Body = () => {
    return (
      <div>
        <RestaurantCard />
      </div>
    );
  };
```

The name, image and all other datas shown in the above card won't be same always. So, it should be dynamic.

As we are using JSX, we can do javascript inside HTML.

Making data dynamic

* const burgerKing = {
    name: "Burger King",
    image: "https://some url",
    cusines: ["Burger", "American"],
    rating: "4.2",
  };

In real world data does not comes like this

* const RestaurantCard = () => {
    return (

        <h2>{burgerking.name}</h2>
        <h3>{burgerking.cusines.join(",")}</h3>
        <h4>{burgerking.rating}</h4>

    );
  };

In real world, data is not like this. There are a number of restaurants.
To render many restaurants :-

(i) have many restaurant cards by doing :

```
Const Body = () => {
  return (
    <div>
      <RestaurantCard />
      <Restaurant Card />
      <Restaurant Card />
      <Restaurant Card />
    </div>
  );
};
```

(ii) Make all these cards dynamic.

In real world, data doesn't comes like above one [burgerking] having a single object.
It comes as "array of objects".
One of the obj would be that 'burgerking'.
Similarly, there would be other objects as well.

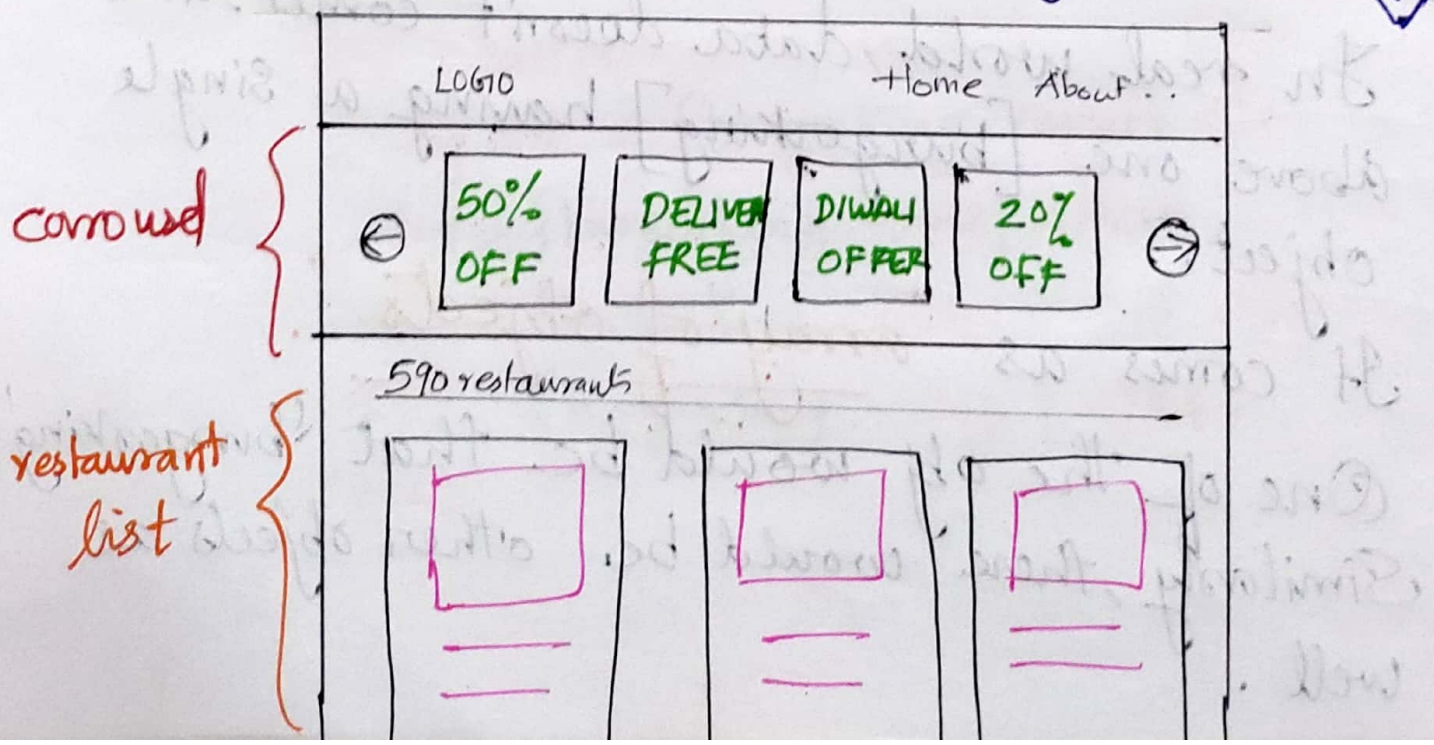
CONFIG DRIVEN UI

All the UI (let say, Swiggy homepage) is driven by a config which is send by backend (api)

Eg:- Swiggy should work in Kolkata, Dehradun, Delhi and whatever location it is, we should have different website info on each location.
For that, we control our front-end using 'config'. That is why it is known as Config driven UI.
All these are controlled by backend.

* How we design Config driven UI?

Suppose, our UI is something like this ↓



If our 'config' is coming from backend, and my data will come like:

```
const config = [
  {
    type : "carousel",
    cards : [
      {
        offerName : "50% OFF"
      },
      {
        offerName : "No Delivery Charge"
      }
    ]
  },
  {
    type : "restaurants",
    cards : [
      {
        name : "Burgerking",
        image : "https://some url",
        cusines : ["Burger", "American"],
        rating : "4.2",
      },
```

These offers are different for different locations

If there are no offers for a particular location, our backend will not send us this object or send empty list of cards

Only backend will change offers & website will render accordingly

```
    {
      name : "KFC",
      image = "https://some url",
      cusines : ["Burger", "American"],
      rating : "4.2",
    }
]]]
```

* Take data from swiggy website and build your own — watch from 01:45:37

Optional chaining

→ allows us to access an object's properties without having to check if the object or its properties exist.

→ represented by '?'.

→ new feature introduced in javascript ES2020.

So, after taking real data from swiggy, our 'Restaurant Card' function looks like this:

```
const RestaurantCard = () => {
  return (
    <div className="card">
      <img src={"https://cloudinaryurl" +
        restaurantList[1].data?.cloudinaryImgId
      } />
      <h2>{restaurantList[0].data?.name}</h2>
      <h3>{restaurantList[0].data?.cuisines.join(", ")}</h3>
      <h4>{restList[0].data?.time} minutes</h4>
    </div>
  );
};
```

Now, all my cards will show same data. I need to transfer dynamic data. We are having hard coded data right now. So, I'll make like :-

my first <RestaurantCard/> card should come from first object & second card from second object

So, the BODY will look like below :

```
* const Body = () => {
    return (
      <div className="restaurant-list">
        <RestaurantCard restaurant={restaurantList[0]} />
        <Restaurant card restaurant={restaurantList[1]} />
        <Restaurant card restaurant={restaurantList[2]} />
        <Restaurant card restaurant={restaurantList[3]} />
        <RestaurantCard restaurant={restaurantList[4]} />
      </div>
    );};
```

Whatever you pass in here is known as PROPS
(here, restaurant is the PROPS)

PROPS

- Shorthand for properties
- "passed props" means I'm passing some data or class. or properties into my functional component.
- restaurant = {restaurantList[3]}

This means react wraps up all these properties into a variable known as 'props'. I can call it anything.

→ So, our RestaurantCard function will now look like ↓

```
* const RestaurantCard = (props) => {
    return (
      <div className="card">
      <img src={"https://some url" +
          props.restaurant.data?.cloudinaryImgId}
      />
      <h2>{props.restaurant.data?.name}</h2>
      <h3>{props.restaurant.data?.cuisines.join(", ")}</h3>
      <h4>{props.restaurant.data?.time}minutes</h4>
      </div>
    );
};
```

→ OBJECT DESTRUCTURING

```
(props) => ({restaurant})
(restaurant.data?.name)
```

We can destructure our ({restaurant}) also ↓

* const {name, cuisines, cloudinaryImg, time} = restaurant.data;

then,
{restaurant.data?.name} ⟹ {name}

If I don't want to destructure like above. but want to destructure everything on the flyer Then it will look like ↓

* const RestaurantCard = ({
    name, cuisines, cloudinaryImgId, time
  }) => {

```
return (
  <div classname ="card">
  <img src = { "https://some url" +
     cloudinaryImgId }
  />
<h2> {name} </h2>
<h3> {cuisines.join(", ")}</h3>
<h4> {time} minutes </h4>
</div> ); };
```

→ Then, I have to pass my individual props in Body :-

```
* const Body = () => {
    return (
    <div classname = "restaurant-list">
    <RestaurantCard
      name = {restaurantList[0].data.name}
      Cuisines = {restaurantList[0].data.cuisines}
    />
    <RestaurantCard
      name = {restaurantList[1].data.name}
      cuisines = {restaurantList[1].data.cuisines}
    />
</div> ); };
```

This has all the props like name, cusines, time etc & I may want to pass all of these props. So, instead of writing each prop individually. I will do :-

```
{...restaurantList[1].data}
```

Spread Operator

→ denoted by three dots (...)

→ It allows an iterable (an object that can be looped over, such as an array or a string) to be expanded in places where multiple elements are expected.

Eg :- For above code for Body(), multiple elements are to be written for every <RestaurantCard/>

So, instead of that we can write

{...restaurantList[0].data}

So, my Body() fn will now look like ↓

```
* const Body = () => {
    return (
      <div className="restaurant-List">
        <RestaurantCard {...restaurantList[0].data} />
        <RestaurantCard {...restaurantList[1].data} />
        <RestaurantCard {...restaurantList[2].data} />
      </div>
    );};
```

→ But if I have a 100 restaurant, writing code like above is not a good way.
So, we can run a loop over this. But in functional programming, we don't use for loop.
We use `.map`.

`.map` is the best way to do it.

[* Watch map, filter & redux video on YT.]

→ HW :- difference between map & foreach.

→ So, using map, my code looks like ↓

```
* const Body = () => {
    return (
      <div className="restaurant-list">
        {restaurantList.map((restaurant) => {
          return (
            <RestaurantCard {...restaurant.data}/>
          );
        })}
      </div>
    ); };
```

→ restaurantList — is my array of objects.

restaurantList.map( ) —— means :-

This will map my array and I will pass a function (callback function). This callback fn takes each object. ie, (restaurant).

So, for each object in that array, I want ~~to~~ my function to return some piece of JSX. (That JSX is my <RestaurantCard/>

So, I will spread my (restaurant) object in card..

→ Everything that we build is a CONFIG DRIVEN UI.

→ After running, React will give a warning that "Each child in a list should have a unique "key" prop.

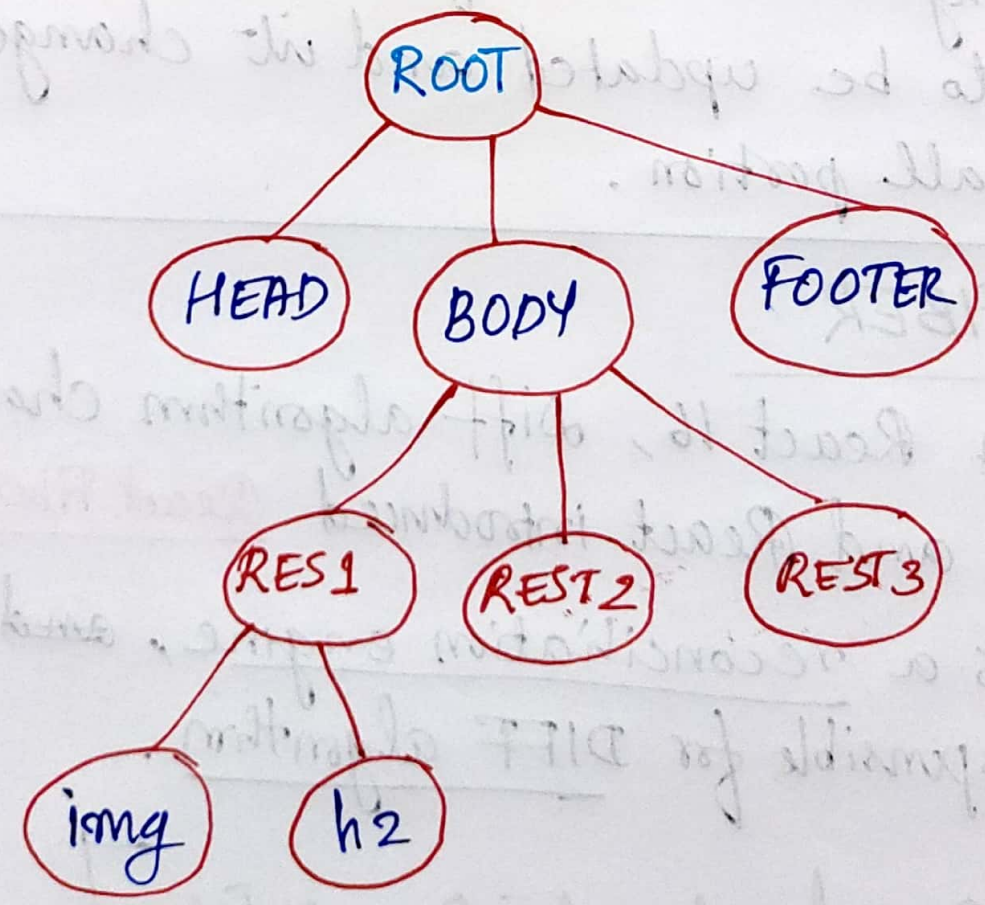
```
<RestaurantCard {...restaurant.data}
    key={restaurant.data.id}
/>
```

VIRTUAL DOM

→ Let the structure of our DOM look like ↓

```
<>
    <head>
    <body>
        <Rest 1>
        <Rest 2> <img .. />
        <Rest 3>
    </body>
    </head> <footer/>
</>
```

→ We keep a representation DOM with us, which is known as virtual DOM.



→ We need Virtual DOM for Reconciliation

* → Reconciliation is an algorithm that React uses to diff one tree from other. It uses Diff Algorithm and in determining what needs to change and what does not in UI.

[To find out DIFFERENCE between one tree (Actual DOM) and other (VIRTUAL DOM)]

→ Diff Algorithm then finds out what needs to be updated and it change only that small portion.

REACT FIBER

→ In React 16, Diff algorithm changed a little and React introduced React Fiber

→ It's a reconciliation engine, ~~and~~ which is responsible for DIFF algorithm.

[Read about React fiber]