Date .....

"Building a food-ordering App"

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

3 ways of component composition :

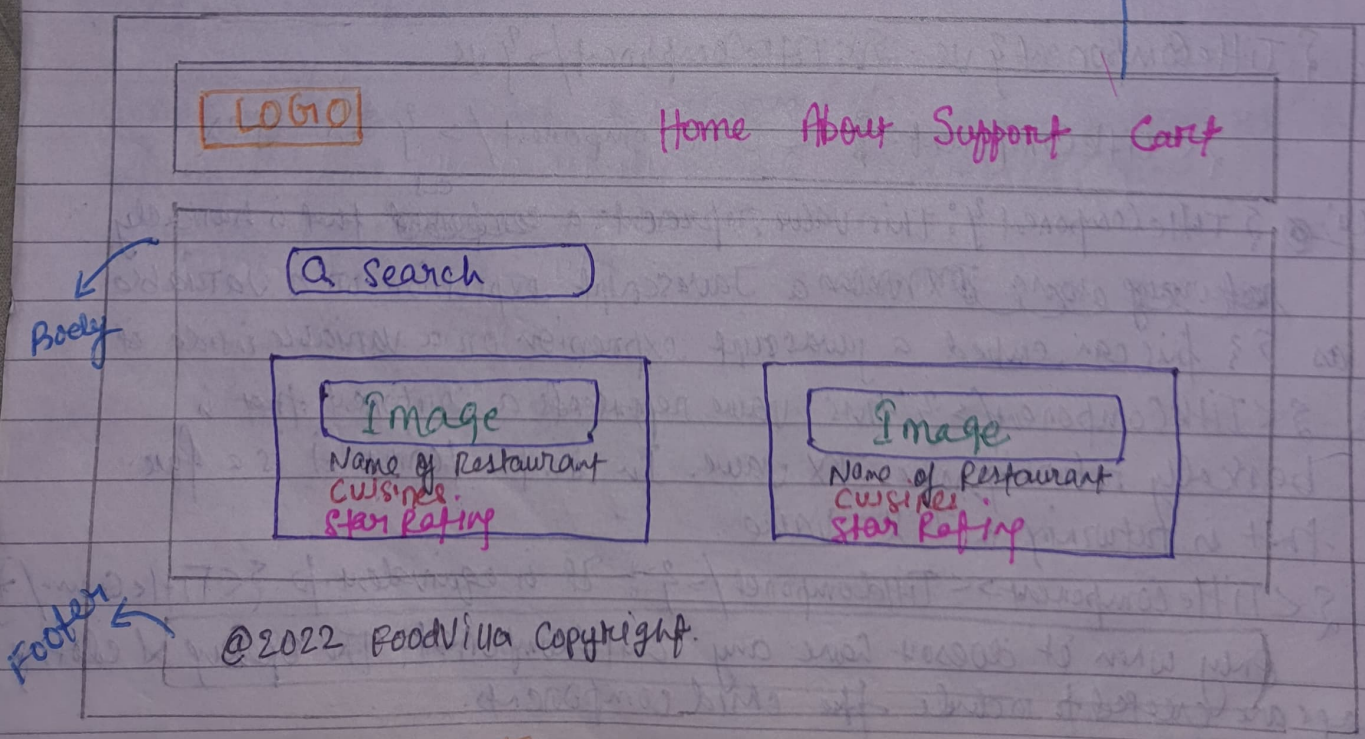
1. { Title() }
2. <Title/> ⟶ used generally
3. <Title> </Title>

BUILDING OUR APP

Name : Food Villa

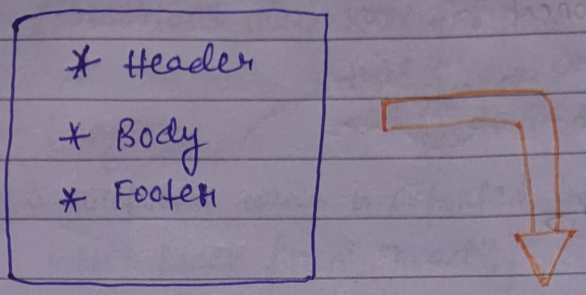
"whenever you are writing code, do planning"

Our app look like this ⟶



Date.....

So, the app layout should have :-



Const Applayout = () => {

return (

- Header
  - Logo
  - Nav Items (on right side)
  - Cart
- Body
  - Searchbar
  - Restaurant List
    - RestaurantCard
      - Image
      - Name
      - Rating
      - Cusines
- Footer
  - Links
  - Copyright

)}

Date.....

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>
    );
  };
```

Date.....

Note:

* JSX expressions must have one parent element.

React.Fragment → tujhe div mein wrap nai karna parega → dom mein extra div

→ 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>

Date.....

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

Giving style inside React

To give inline styles in react, do :-

First method ↓

Eg:-

```
* const styleObj = {
      backgroundColor: "red",
  };
```

styleObj is Normal JS object

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

inside this parenthesis, 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> Hellow </h1>
     <h2> World </h2>
     </div>
   );
```


Teacher's Sign .....

Date.....

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', 'Material UI' etc.

Date.....

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

Ans:- 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.

Date.....

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> Burger, American </h3>
          <h4> 4.2 stars </h4>
        </div>
      );};

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

The name, image and all other data's 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.

Date.....

Making data dynamic

* const burgerKing = {

```
name: "Burger King",
image: "https://someurl",
cuisines: ["Burger", "American"],
rating: "4.2"
};
```

In real world data does not comes like this

```
const RestaurantCard = () => {
    return (
        <div className="card">
            <img src={burgerKing.image} />
            <h2> {burgerKing.name} </h2>
            <h3> {burgerKing.cuisines.join(", ")} </h3>
            <h4> {burgerKing.rating} </h4>
        </div>
    );
};
```

arrays ko string kaise kare → .join(", ")

In real world, data is not like this. There are a number of restaurant.

To render many restaurants :-

(i) Have many restaurant cards by doing :

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

Date.....

Address

ii) Make all these cards dynamic.

In real world, data doesn't come 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

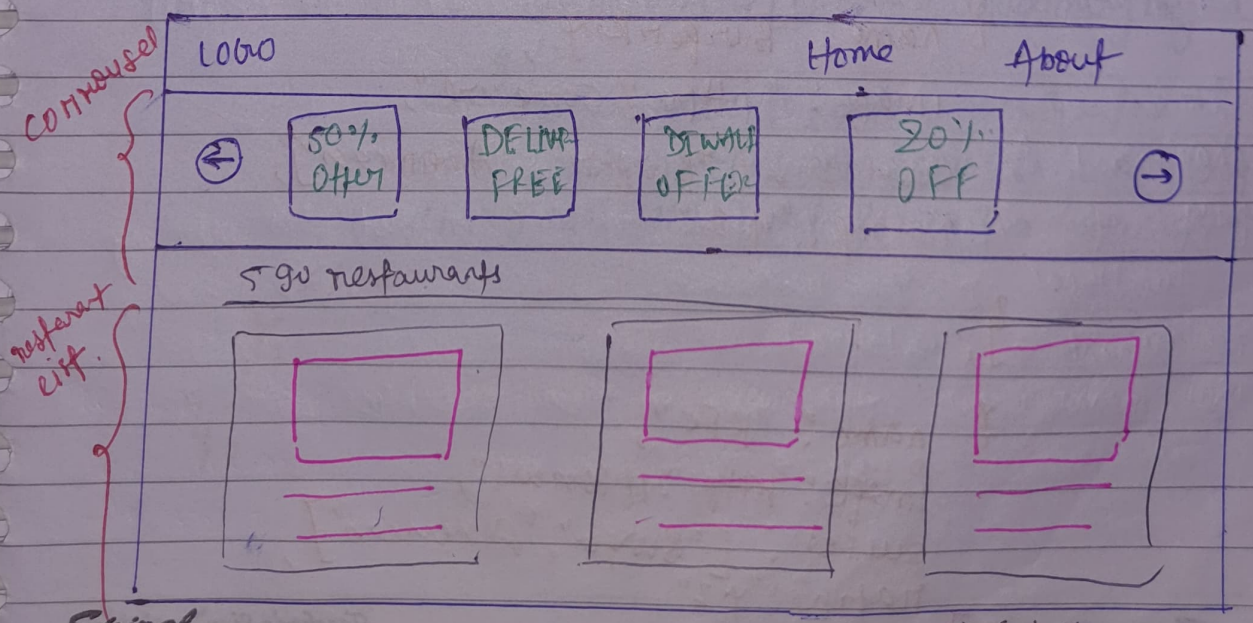
All the UI (lly say, Swiggy home page) is driven by a config which is send by backend (api).

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 ~~cannot~~ 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 ↓



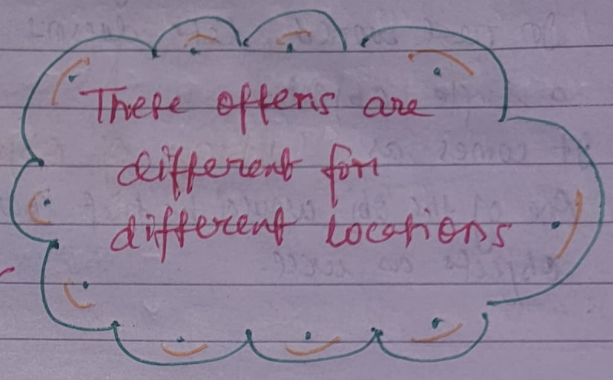
Spiral

Teacher's Sign .....

Date.....

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://someurl",
        cusines : ["Burger", "American"],
        rating : "4.2"
      },
      { name : "KFC",
        image : "https://someurl",
        cusines : ["Burger", "American"],
        rating : "4.2",
      }
    ]}
];
```



Date.....

[Take data from swiggy website and build your own - watch from 01:45:37]

Optional chaining

→ allows us to accer 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://cloudinarycdn.com" +
                resList[1].data?.cloudinaryImageId
            } />
            <h2>{resList[0].data?.name}</h2>
            <h3>{restaurant[0].data?.cuisines.join(", ")}</h3>
            <h4>{resList[0].data?.time} minutes</h4>
        </div>
    );
};
```

Date.....

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]}/>
      <RestaurantCard restaurant={restaurantList[1]}/>
      <RestaurantCard restaurant={restaurantList[2]}/>
      <RestaurantCard 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

→ "pass props" means I'am passing some data or properties into my functional or class component.

→ restaurant = {restaurantList[3]}

This means react wraps up all these properties into a variable known as 'props' → tu naam kuch bhi rakh sakta ha 'props'

Date.....

So our RestaurantCard function will now look like ↓

```
*const RestaurantCard = (props) => {

    return(
      <div className="card">
        <img src={"https://someurl"+
          props.restaurant.data?.cloudinaryImageId} />
        <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, cloudinaryImageId, time} = restraunt.data

then,

{restaurant.data.?.name} ⟹ {name}

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

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

<img src = { "https://someurl" +
cloudinaryImageId } />
<h2> {name} </h2>
<h3> {cuisines.join(", ")} </h3>
<h4> {time} minutes </h4>
 ); };

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}
/>

Teacher's Sign .....

Date.....

```
<RestaurantCard
  name={restaurantList[1].data.name}
  cuisines={restaurantList[1].data.cuisines}
/>
</div>);};
```

This has all the props like name, cuisines, 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() function will now look like

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


Teacher's Sign.....

→ 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 & reduce video on YT]

→ H.W.: 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>
    );
  };
```

Date.....

→ restaurantList – is my array of objects

restaurantList.map() ——means :-

This will map my array and I will pass a function (callback function). This callback function takes each object ie, (restaurant).
So, for each object in that array, I want 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}
/>
```

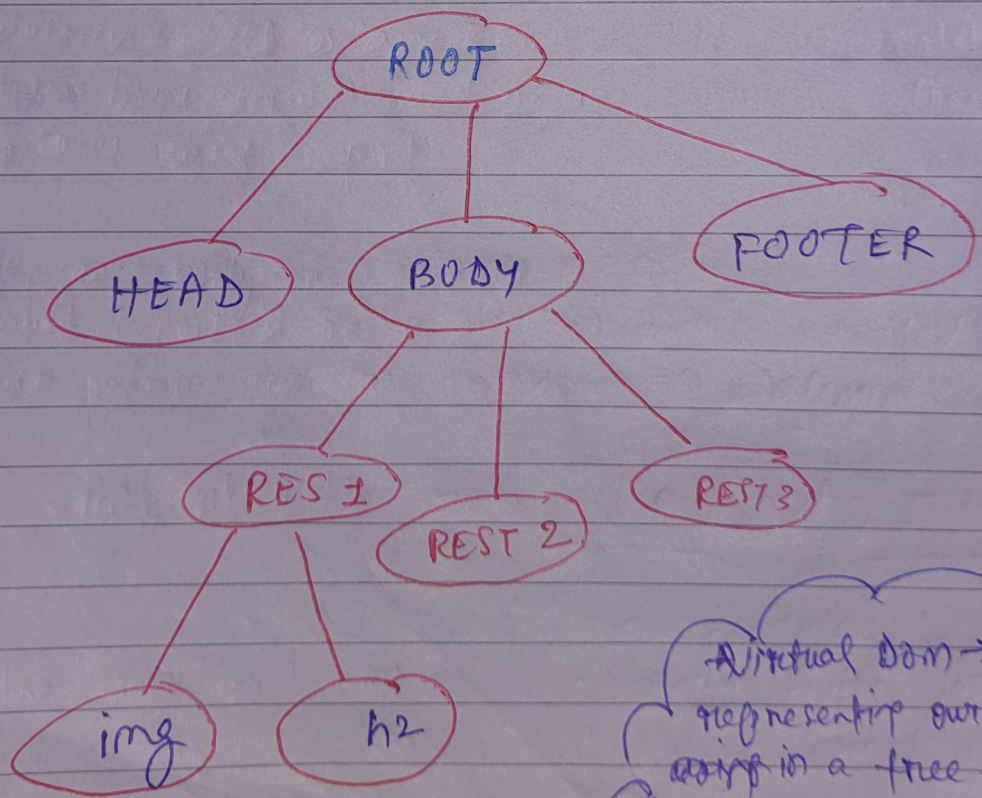
Date.....

VIRTUAL DOM

→ Let the structure of our DOM look like ↓

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

→ We keep a representation DOM with us, which is know as Virtual DOM.



Virtual DOM → representing our DOMs comp in a tree like structure.

Teacher's Sign .....

Date.....

→ 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 determines 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, which is responsible for DIFF algorithm.

[Read about React fiber]