5 - Let's get Hooked

12 April 2023 07:47 PM

- Each component has each file.
- Generally we create "src" folder.

App.js → src folder.

src → components (folder)

→ Inside this all components file.

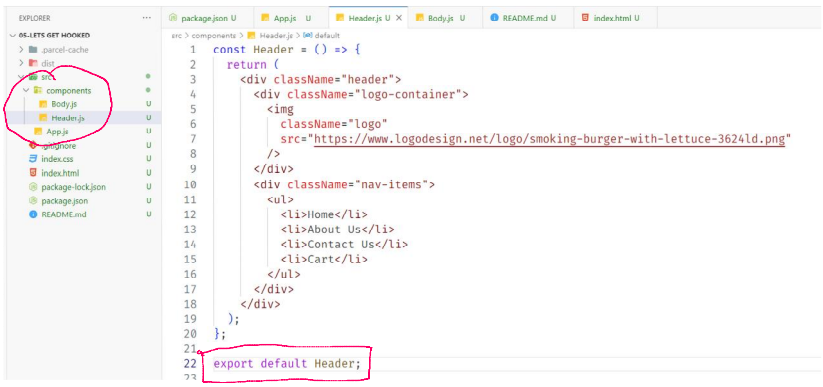
So now for Header Component create a Header.js file

or

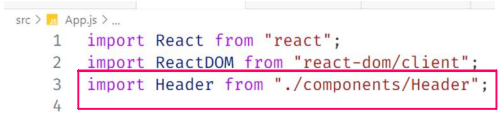
Header.jsx.

Now import Header to the App.js before that export the Header code.

To export ⇒ export default name of the Component.

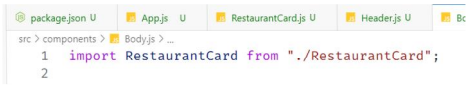


To import:



To the same for other components.

for the RestaurantCard import it to the Body.js.

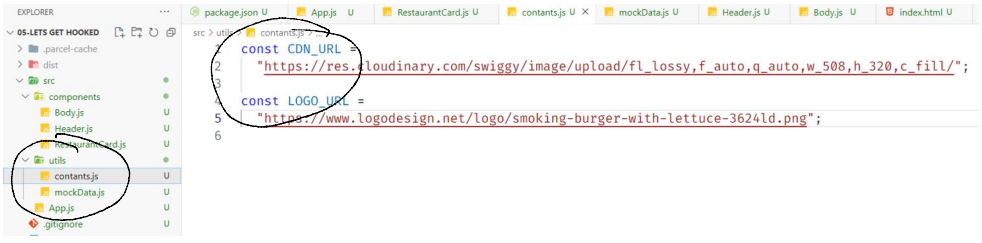


→ Harded coded data and URL don't keep it in the components.

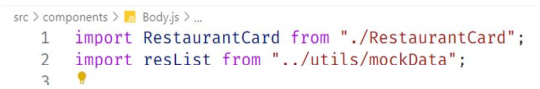
→ keep it in the common folder (util, config) and create a file name as contants.js. in smaller letter as it's not components.

Variable should be in captial letter

→ For the dummy data we creak mockData.js



now again you need to export and import



Other way to export. and import (Named)

Other way to export [illegible]

- this is used when in single file you need to export multiple things. for eg. in constants.js you have CDN_URL and LOGO_URL.

```
src > utils > constants.js > ...
1  export const CDN_URL =
2    "https://res.cloudinary.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_5
3
4  export const LOGO_URL =
5    "https://www.logodesign.net/logo/smoking-burger-with-lettuce-3624ld.png";
```

Now to import,

import {Name} from '.....';

```
src > components > RestaurantCard.js > RestaurantCard
1  import { CDN_URL } from "../utils/constants";
2
3  const RestaurantCard = (props) => {
4    const { resData } = props;
5    const { cloudinaryImageId, name, cusines, totalRatingsString, deliveryTime } =
6      resData?.data;
7    return (
8      <div
9        className="res-card"
10       style={{
11         backgroundColor: "#f0f0f0",
12       }}
13     >
14       <img className="res-logo" src={CDN_URL + cloudinaryImageId} />
15       <h3>{name}</h3>
16       <h4>{Array.isArray(cusines) ? cusines.join(",") : ""}</h4>
17       <h4>{totalRatingsString} stars</h4>
18       <h4>{deliveryTime}</h4>
19     </div>
20   );
21 };
22
23 export default RestaurantCard;
24
```

```
src > components > Header.js > Header
1  import { LOGO_URL } from "../utils/constants";
2
3  const Header = () => {
4    return (
5      <div className="header">
6        <div className="logo-container">
7          <img className="logo" src={LOGO_URL} />
8        </div>
9        <div className="nav-items">
10         <ul>
11           <li>Home</li>
12           <li>About Us</li>
13           <li>Contact Us</li>
14           <li>Cart</li>
15         </ul>
16       </div>
17     </div>
18   );
19 };
20
21 export default Header;
22
```

→ as object pass it.

H.W.

Can you use two default ?

Hooks :

let make the app dynamic (like interactive)

- lets create a button, on click of that we should top-rated restaurant. (rating > 4 star) (for us 1000+ rating)

Code for on-click : (Click-handler)

- create a attribute onClick = { }

- It take a callback f().

```
<div className="filter">
  <button className="filter-btn" onClick={() => {}}>
    Top Rated Restaurants
  </button>
</div>
```

```
<button
  className="filter-btn"
  onClick={() => {
    console.log("Button Clicked");
  }}
>
```

For rating code:

- The cards are coming from reslists (it has mockData)

  let's not use reslists for now.

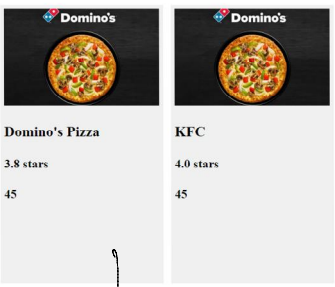
- Create our own list of Restaurant. from scratch

  Very Important

```
const listOfRestaurants = []; // Empty for now
return (
  <div className="body">
    <div className="filter">
      <button
        className="filter-btn"
        onClick={() => {
          console.log("Button Clicked");
        }}
      >
        Top Rated Restaurants
      </button>
    </div>
    <div className="res-container">
      {listOfRestaurants.map((restaurant) => (
        <RestaurantCard key={restaurant.id} resData={restaurant} />
      ))}
    </div>
```

Let's use this data

```
const Body = () => {
  let listOfRestaurants = [
    {
      data: {
        id: "1",
        name: "Domino's Pizza",
        cloudinaryImageId: "bz9zkh2aqywjhpankb07",
        cuisines: ["Pizzas"],
        deliveryTime: 45,
        avgRating: "3.8",
      },
    },
    {
      data: {
        id: "2",
        name: "KFC",
        cloudinaryImageId: "bz9zkh2aqywjhpankb07",
        cuisines: ["Burger"],
        deliveryTime: 45,
        avgRating: "4.2",
      },
    },
  ];
```



```
    },
  },
];
```

So on click on top Rated Restaurant only Top rated should display.

→ So you need to write filter logic.

```
return (
  <div className="body">
    <div className="filter">
      <button
        className="filter-btn"
        onClick={() => {
          // Fliter logic here
          listOfRestaurants = listOfRestaurants.filter(
            (res) => res.data.avgRating > 4
          );
          console.log(listOfRestaurants);
        }}
      >
        Top Rated Restaurants
      </button>
    </div>
    <div className="res-container">
      {listOfRestaurants.map((restaurant) => (
        <RestaurantCard key={restaurant.data.id} resData={restaurant} />
      ))}
    </div>
  </div>
);
```

The code above filters a list of restaurants based on a condition that checks if the totalRatingsString property of each restaurant's data object is greater than 4. The `filter()` function in JavaScript creates a new array containing all elements that pass the test implemented by the provided function.

In this case, the provided function is an arrow function that takes one argument, `res`, which represents a single restaurant object from the original array. The function checks if the `totalRatingsString` property of the `data` object of the restaurant has a value greater than 4. If the value is greater than 4, the function returns `true`, indicating that the restaurant should be included in the filtered array. If the value is less than or equal to 4, the function returns `false`, indicating that the restaurant should be excluded from the filtered array.

Therefore, after the code executes, `listOfRestaurants` will contain a new array with only the restaurants that have a `totalRatingsString` value greater than 4.

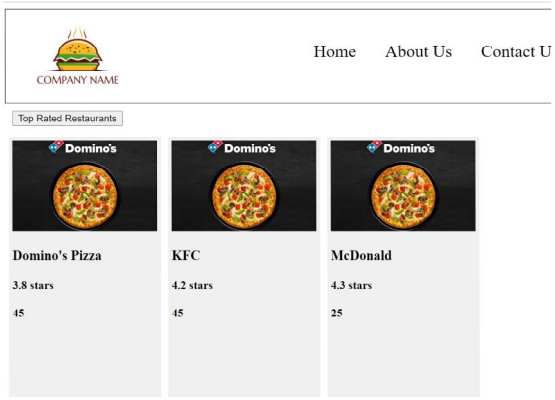
```
▼[{…}]
  ▼0:
    ▶data: {id: '2', name: 'KFC', cloudinaryImageId: 'bz9zkh2aqywjhpankb07', cuisines: Array(1), deliveryTime: 45, …}
    ▶[[Prototype]]: Object
  length: 1
  ▶[[Prototype]]: Array(0)
```

→ only one restaurant came

```
    data: {
      id: "2",
      name: "KFC",
      cloudinaryImageId: "bz9zkh2aqywjhpankb07",
      cuisines: ["Burger"],
      deliveryTime: 45,
      avgRating: "4.2",
    },
  },
  {
    data: {
      id: "3",
      name: "McDonald",
      cloudinaryImageId: "bz9zkh2aqywjhpankb07",
      cuisines: ["Meals"],
      deliveryTime: 25,
      avgRating: "4.3",
    },
  },
];
```

```
▼(2) [{…}, {…}]
  ▼0:
    ▶data: {id: '2', name: 'KFC', cloudinaryImageId: 'bz9zkh2aqywjhpankb07', cuisines: Array(1), deliveryTime: 45, …}
    ▶[[Prototype]]: Object
  ▼1:
    ▶data: {id: '3', name: 'McDonald', cloudinaryImageId: 'bz9zkh2aqywjhpankb07', cuisines: Array(1), deliveryTime: 25,
    ▶[[Prototype]]: Object
  length: 2
  ▶[[Prototype]]: Array(0)
```

but our UI didn't update.



So React has superpower,

now if you want

- data, UI layer consistent with each other

that's were React comes into picture.

- Updating the DOM.

Introduction of Hooks starts:

we use State Variable - Super Powerful Variable

- we use React Hooks (Use State)

- React Hooks is basically a normal JS f(), which is pre-built which has some superpower.

Two hooks:

- useState (80%) we use
- use Effect (20%)

Use State Hooks:

- Superpowerful State Variable
- So you need to import from react as named import

  import {useState} from "react";
- It's known as State variable because it maintain state of the components.

You get state variable inside an array

const [listOfRestaurant]

Like in normal JS we created let listOfRestaurant.

```
// Local State Variable - Super powerful variable
const [listOfRestaurants] = useState();
// normal JS variable
// let listOfRestaurants = [ ...
// ];
```

State variable

normal variable

const [listOfRestaurant] = useState([])

↳ default value

if you want to pass null

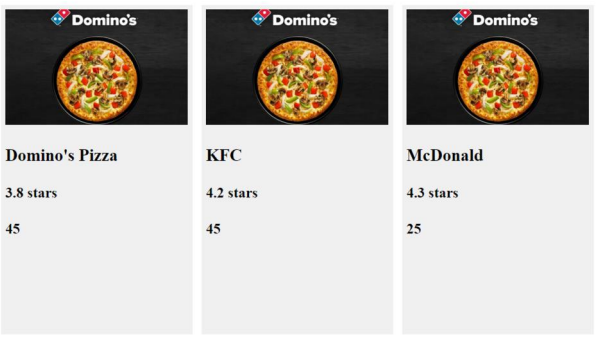
const [listOfRestaurant] = useState(null)

Here the data is passed:

```
// Local State Variable - Super powerful variable
const [listOfRestaurants] = useState([
  {
    data: {
      id: "1",
      name: "Domino's Pizza",
      cloudinaryImageId: "bz9zkh2aqywjhpankb07",
      cuisines: ["Pizzas"],
      deliveryTime: 45,
      avgRating: "3.8",
    },
  },
  {
    data: {
      id: "2",
      name: "KFC",
      cloudinaryImageId: "bz9zkh2aqywjhpankb07",
      cuisines: ["Burger"],
      deliveryTime: 45,
      avgRating: "4.2",
    },
  },
  {
    data: {
      id: "3",
      name: "McDonald",
      cloudinaryImageId: "bz9zkh2aqywjhpankb07",
      cuisines: ["Meals"],
      deliveryTime: 25,
      avgRating: "4.3",
    },
  },
]);
```

```
const Body = () => {
  // Local State Variable - Super powerful variable
  const [listOfRestaurants] = useState([
    {...
    },
    {...
    },
    {...
    },
  ]);
```

1:29:



Now, to modify the 'list of Restaurant' variable

Because it's a React special variable we need to use React Hooks. (Use State Hook which give array)

To modify List Of Restaurant,

ListOfRestaurant = []; X (You can't modify like this)

we modify by a function(), it comes as 2nd parameter in an array. (setListOfRestaurant) // You can name it anything but according to industry if name is listOfRestaurant then it better to put setListOfRestaurant, if it 'res' then setRes. Basically add 'set'.

So the setListOfRestaurant is to update the lists.

```
const Body = () => {
  // Local State Variable - Super powerful variable
  const [listOfRestaurants, setListOfRestaurant] = useState([
    {...
    },
    {...
    },
    {...
    },
  ]);
```

Now on click on button, you need to update. So to update the state variable is. You need to call the setListOfRest and pass in the new data which you need to push inside it.

```
  ]);

  return (
    <div className="body">
      <div className="filter">
        <button
          className="filter-btn"
          onClick={() => {
            setListOfRestaurant();

            // listOfRestaurants = listOfRestaurants.filter(
            //   (res) => res.data.avgRating > 4
            // );
            // console.log(listOfRestaurants);
          }}
        >
```

Now what you need to push is

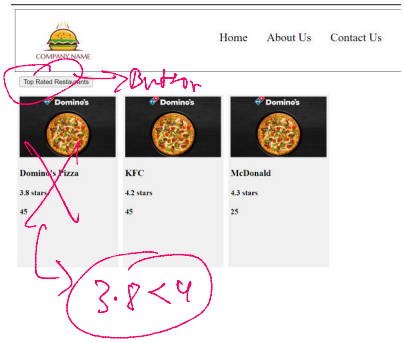
```jsx
  ]);

  return (
    <div className="body">
      <div className="filter">
        <button
          className="filter-btn"
          onClick={() => {
            const filteredList = listOfRestaurants.filter(
              (res) => res.data.avgRating > 4
            );
            setListOfRestaurant(filteredList);
          }}
        >
          Top Rated Restaurants
        </button>
      </div>
      <div className="res-container">
        {listOfRestaurants.map((restaurant) => (
          <RestaurantCard key={restaurant.data.id} resData={restaurant} />
        ))}
      </div>
    </div>
  );
};
```

So basically, your updating list of Restaurant with this filtered list.

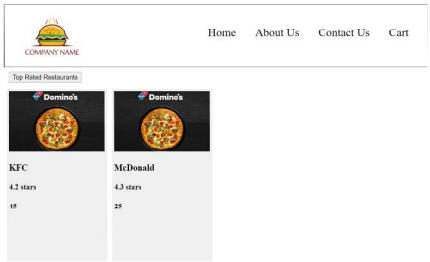
Suppose you need to make listOfRestaurant empty you can do this,

```jsx
const Body = () => {
  // Local State Variable - Super powerful variable
  const [listOfRestaurants, setListOfRestaurant] = useState([
    {...
    },
    {...
    },
    {...
    },
  ]);

  setListOfRestaurant([]);
```

So now on click of button (Top Rated Restaurant)



UI got filtered automatically



Wherever a set variable updates, React render the components, or changes

So in the above eg. on click of the button, react render the components.

This is the power of useState.

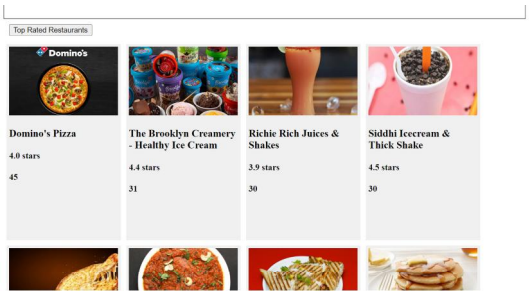
Now let's use our mock data,

```jsx
import RestaurantCard from "./RestaurantCard";
import { useState } from "react";
import resList from "../utils/mockData";

const Body = () => {
  // Local State Variable - Super powerful variable
  const [listOfRestaurants, setListOfRestaurant] = useState(resList);
```

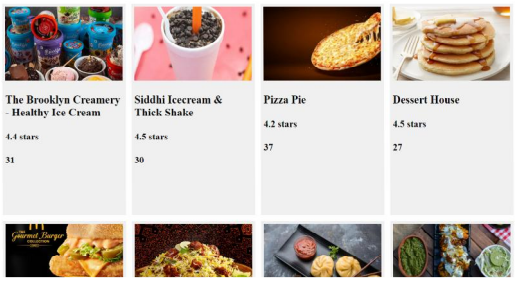
mockdata file

```jsx
const resList = [ ...
];

export default resList;
```

Original UI



on click of button it get update.

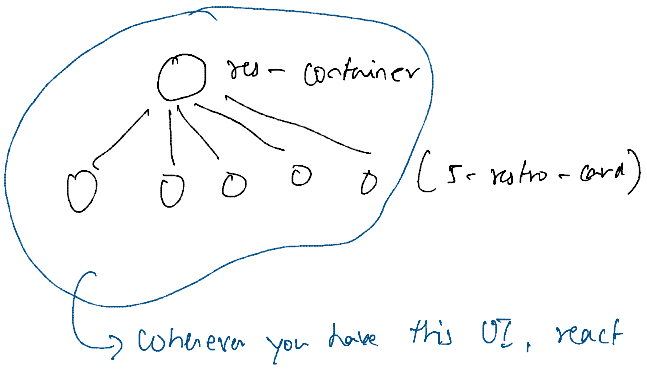


React make the DOM operation super fast.

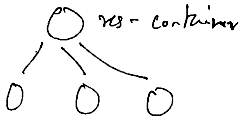
How React is doing all this?

- React use something known as Reconciliation Algorithm.

Reconciliation Algorithm (React Fibre)
(React 16)



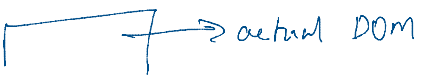
UI changes from
⇒ 5 - card
to 3 card.



→ Wherever you have this UI, react create virtual DOM.

Virtual DOM (not a actual DOM) : It's a representation of actual DOM.

Actual DOM are tags.



Virtual DOM

Let's see virtual DOM in code:

```
App.js U
src > App.js > AppLayout
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import Header from "./components/Header";
4  import Body from "./components/Body";
5
6  const AppLayout = () => {
7    console.log(<Body />);
```

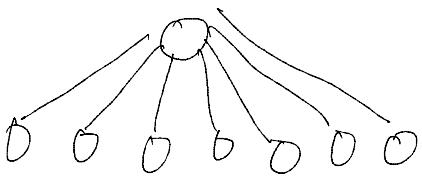
↳ it will print object

```
App.js:
▼ {$$typeof: Symbol(react.element), key: null, ref: null, props: {…}, type: ƒ, …}
    $$typeof: Symbol(react.element)
    key: null
  ▶ props: {}
    ref: null
  ▶ type: ()=> {…}
  ▶ _owner: FiberNode {tag: 0, key: null, stateNode: null, elementType: ƒ, type: ƒ, …}
  ▶ _store: {validated: false}
    _self: undefined
  ▶ _source: {fileName: 'src/App.js', lineNumber: 7, columnNumber: 15}
  ▶ [[Prototype]]: Object
```

This object is virtual DOM.

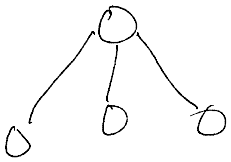
Just like React.createElement (three parameter) // this is object

Diff Algo



OLD Virtual DOM

Clicked the button



New Virtual DOM

Diff Algo: It basically try to find the difference b/w old virtual DOM and new virtual DOM.

→ The difference here is 'li' node

– Then it calculate the difference and then actual update the DOM on every render cycle.

Read more on this

https://github.com/acdlite/react-fiber-architecture

The React is fast because of virtual DOM, diff algo, React fiber.

Homework:

1) What is the difference b/w named exports, default exports and * as exports?

The usage and behaviour only differ in this.

Named Exports:

- It's used to export multiple components, functions, or objects from a module.
- You can define multiple component in a single file and export them using named exports.

for e.g.

```
export const Button = () => {
  return <button> Click me </button>;
};
export const Input = () => {
  return <input type="text" />;
};
```

In another module, you can import

```
import { Button, Input } from './myComponents';
```

Default Exports:

- Used to export a single component, fn() or object from a module.

for e.g.

```
const Button = () => {
return <button> Click me </button>;
};
export default Button;
```

In another module, you can import

```
import Button from './Button';
```

Note: when importing a default import, you don't need the curly braces.

* n Export:

- Used to import all named exports from a module into a single object.

for e.g.

```
export const Button = () => {
  return <button> Click me </button>;
};
export const Input = () => {
  return <input type="text" />;
};
export const Checkbox = () => {
  return <input type="checkbox" />;
};
```

In another module, you can import as

```
import * as MyComponents from './myComponent';
<MyComponents.Button />
<MyComponents.Input />
<MyComponents.Checkbox />
```

2) What are the important of config.js file.

- It can help to improve the maintainability, security and deployment of a React Application by providing a centralized location for configuration variable.

Suppose you have an app, that retrieves data from an API and you want to store the API end point

```
// config.js

const config = {
```

```
// config.js

const config = {
  apiUrl: "https://api.example.com",
};

export default config;
```

you want to store the API endpoint URL in the configuration file.

So now in your app, you can then import config.js file and use 'apiURL' property to retrieve data from the API.

```
import React, { useState, useEffect } from "react";
import config from "./config";

const MyComponent = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch(`${config.apiUrl}/data`)
      .then((response) => response.json())
      .then((data) => setData(data));
  }, []);

  return (
    <div>
      {data.map((item) => (
        <div key={item.id}>{item.name}</div>
      ))}
    </div>
  );
};

export default MyComponent;
```

By using 'config.apiURL' property you can easily change the API endpoint URL.

3) What are React Hooks?

- They are way to add state and other features to functional components in React, without having to use class component.
- Hooks are special f() provided by React that let you use state, lifecycle methods, context etc.
- It can make your code simpler and easier to understand, since you don't have to deal with complex class component

```
import React, { useState } from "react";

const MyComponent = () => {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
};

export default MyComponent;
```

In the code, we're using the useState hook to add state to the MyComponent f() component.

- Initializing the count state = 0
- Providing a f(), setCount that allow us to to update the count whenever the button is clicked.

4) Why do we need useState Hook ?

- State is a way to store data that can change over time, such as user input or the result of the API call

Syntax of the State Hook :

```
const [state, setState] = useState(initialState);
```

```jsx
import React, { useState } from "react";

const MyComponent = () => {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
};

export default MyComponent;
```

In the code, we're using the useState hook to add state to the MyComponent fC) component.

- Initializing the count state = 0
- Providing a f() , setcount that allow us to to update the count whenever the button is clicked.