Chapter 5 – Let's get hooked

Before we get into hooks, lets discuss about Folder structure in react.

Is there a recommended way to structure React projects?

React doesn't have opinions on how you put files into folders.

Grouping files by features

```
common/
  Avatar.js
  Avatar.css
  APIUtils.js
  APIUtils.test.js
feed/
  index.js
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  FeedAPI.js
profile/
  index.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
  ProfileAPI.js
```

Grouping files by file types

```
api/
  APIUtils.js
  APIUtils.test.js
  ProfileAPI.js
  UserAPI.js
components/
  Avatar.js
  Avatar.css
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
```

We can customise our own folder structure as well.

Import and Export in React -

Single Responsibility principle (SRP) is a design Patten which states each component in an application should have one responsibility / purpose.

In our application we created three components header, footer and body inside the layout app component. The app component is violating SRP. As per SRP, we should create component for which the component is designed for. We can create components for header, footer and body and maintain them in separate files. Later we can use them inside app layout component whenever we need. This is where import and export come in. Import and export in react will help us write modular code i.e. splitting the code into modular files.

Import –

Import statement in a file allows us to use contents of another file from where the contents are exported. There are two type of imports in react. Named import and default import

Export –

Export statement in a file allows us to export the file contents to another file where the contents are imported. There are two type of exports in react. Named export and default export

Let's consider few cases of exports and imports.

Default Export – (Export Single component)

This is the default way of exporting a component in react. We can have only one default export per module. syntax is export default <componentName>

RestaurantCard.js

src > components > RestaurantCard.js > ...

```
1  import { IMG_CDN_URL } from "../constants";
2  const RestaurantCard = ({
3    name,
4    cuisines,
5    cloudinaryImageId,
6    lastMileTravelString,
7  }) => {
8    return (
9      <div className="card">
10       <div className="img-div">
11         <img src={IMG_CDN_URL + cloudinaryImageId} alt="" />
12       </div>
13       <div className="divider"></div>
14       <div>
15         <div className="restaurantNameDesign">{name}</div>
16         <div className="cusines"></div>
17         <div className="rating">{lastMileTravelString} minutes </div>
18       </div>
19     </div>
20   );
21 };
22
23 export default RestaurantCard;
24
```

Default Import – (Import single component)

This is importing a component in react in default way. We can use any component name in default import. If we replace RestaurantCard with XYZ we won't get any error but make sure to use XYZ everywhere in the Body.js module. import componentName from "FilePath"

RestaurantCard.js App.js Body.js

src > components > Body.js > ...

```
1  import { restaurantList } from "../constants";
2  import RestaurantCard from "./restaurantCard";
3
4  const Body = () => {
5    const searchText = "KFC"
6    return (
7      <>
8        <div className="search-container">
9          <input
10           type="text"
11           className="search-input"
12           placeholder="Search"
13           value={searchText}
14         onChange={(e)=>console.log(e.target.value)}
15
16         />
17         <button className="search-btn">Search</button>
18       </div>
19       <div className="restaurant-List">
20         {restaurantList.map((restaurant, index) => {
21           return (
22             <RestaurantCard
23               {...restaurant.data.data}
24               key={restaurant.data.data.id}
25             />
26           );
27         })}
28       </div>
29     </>
30   );
31 };
32
33 export default Body;
```

In the above code snippet, we are importing RestaurantCard component into Body.js file, while importing we can add js extension to the file in the file path without affecting the functionality.

Named Export – (Export Single component)

Exporting a component in react using named export. We use an export key word before exporting a component from a module. We are exporting Footer component from Footer.js module.

App.js Footer.js

src > components > Footer.js > ...

```
1  export const Footer = () => {
2    return <div>Footer</div>;
3  };
4
```

Named Import – (Import Single component)

While importing in a module we use componentName within parenthesis such as {componentName}. Syntax is import {componentName} from "FilePath"

App.js | Footer.js

src > App.js > ...

```
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  import { Footer } from "./components/Footer";
6
7  const AppLayout = () => {
8    return (
9      <>
10       <Header />
11       <Body />
12       <Footer />
13     </>
14   );
15 };
16
17 const root = ReactDOM.createRoot(document.getElementById("root"));
18 root.render(<AppLayout />);
```

Note: {componentName} is not object destructuring.

Default Export – (Export multiple components)

Not possible

Default Import – (Import multiple components)

Not possible

Named Export – (Export Multiple components)

Exporting two footer components in react.

App.js | Footer.js

src > components > Footer.js > ...

```
1  export const Footer1 = () => {
2    return <div>Footer 1</div>;
3  };
4  export const Footer2 = () => {
5    return <div>Footer 2</div>;
6  };
```

App.js | Footer.js

src > components > Footer.js > ...

```
1
2  const Footer1 = () => {
3    return <div>Footer 1</div>;
4  };
5  const Footer2 = () => {
6    return <div>Footer 2</div>;
7  };
8  export {Footer1,Footer2}
9
```

Named Import – (Import Multiple components)

Importing two footer components in react.

App.js | Footer.js

src > App.js > ...

```
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  import { Footer1, Footer2 } from "./components/Footer";
6
7  const AppLayout = () => {
8    return (
9      <>
10       <Header />
11       <Body />
12       <Footer1 />
13       <Footer2 />
14     </>
15   );
16 };
17
18 const root = ReactDOM.createRoot(document.getElementById("root"));
19 root.render(<AppLayout />);
```

Another way

App.js | Footer.js

src > App.js > ...

```
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  import * as obj from "./components/Footer";
7
8  const AppLayout = () => {
9    return (
10     <>
11       <Header />
12       <Body />
13       <obj.Footer1 />
14       <obj.Footer2 />
15     </>
16   );
17 };
18
19 const root = ReactDOM.createRoot(document.getElementById("root"));
20 root.render(<AppLayout />);
```

We can mix default imports exports with named imports exports but the rules are same.

What is Config File?

We create a config file or constant file in our application to store hard coded information. Config file data are named exported data.

constants.js

```
src > constants.js > ...
1    export const IMG_CDN_URL = "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_660/"
2
3  > export const restaurantList = [ ···
1196   ];
```

RestaurantCard.js

```
src > components > RestaurantCard.js > RestaurantCard
1   import { IMG_CDN_URL } from "../constants"; // named import
2
3   const RestaurantCard = ({
4     name,
5     cuisines,
6     cloudinaryImageId,
7     lastMileTravelString,
8   }) => {
9     return (
10      <div className="card">
11        <div className="img-div">
12          <img src={IMG_CDN_URL + cloudinaryImageId} alt="" />
13        </div>
14        <div className="divider"></div>
15        <div>
16          <div className="restaurantNameDesign">{name}</div>
17          <div className="cusines"></div>
18          <div className="rating">{lastMileTravelString} minutes </div>
19        </div>
20      </div>
21    );
22  };
```

In previous chapter we built a skeleton of our application. Let’s add functionalities to it. Let’s build a search bar.

Building Search bar functionality -

Body.js App.js

src > components > Body.js > ...

```
1  import { restaurantList } from "../constants";
2  import RestaurantCard from "./restaurantCard";
3
4  const Body = () => {
5    const searchText = "KFC";
6    return (
7      <>
8        <div className="search-container">
9          <input
10           type="text"
11           className="search-input"
12           placeholder="Search"
13           value={searchText}
14           onChange={(e) => console.log(e.target.value)}
15         />
16       </div>
17       <div className="restaurant-List">
18         {restaurantList.map((restaurant) => {
19           return (
20             <RestaurantCard
21               {...restaurant.data.data}
22               key={restaurant.data.data.id}
23             />
24           );
25         })}
26       </div>
27     </>
28   );
29 };
30
31 export default Body;
```

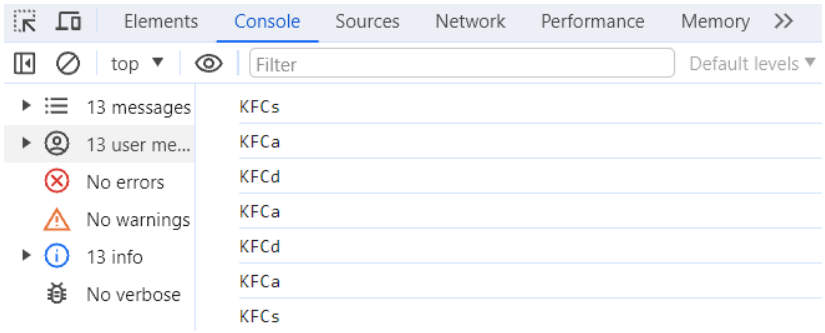
We have added a search bar Text control. We have created a local JavaScript variable searchText inside Body component and placed this to the value property of search bar. That's the reason we can see KFC printed in the search text box.

Now if we try to edit this value, nothing happens in the UI. Why?

* Because it all depends on how the data is flowing. In our case data flow is in one direction i.e. from local variable to the search textbox value. We are simply writing the data from local variable to UI search textbox control. To see the updated change, we got to read the data. Data has to flow backwards i.e. from search textbox to local variable. It sounds complex but it's not.

* Another reason is the way react renders the component. when a change event takes place, when we type anything in search bar, react quickly rerenders the Body component. When this happens the value, KFC is assigned to searchText variable and then the variable value is mapped inside textbox value. Bottom line is no matter how many times we type in the search box, react triggers its render cycle and displays KFC in the search box in every render phase.

We have written the code in such a way that on every text change, the value will be logged in the console.



This is not what we want. we want to see the updated text rather than seeing a hard-coded text.

To achieve this functionality, normal local JavaScript variables are not good enough. We need to maintain state of the variable within the component. That's why react state variables are introduced.

Why React/State variables? Why not local JS variables?

Because React variables provide state which JavaScript local variable cannot provide.

What is State in react?

Every component in react maintains a state so that we can put all variables into the state.

Every time we create local variable, we use state in it.

A state is an object that hold data and information related to a component.

State is mutable. State in a component can change over time. The change in state over time can happen as a response to user action or system event.

State provides Two-way data binding

Why do I need state or state variables when I have local JS variables?

Because React only tracks state variables. React cannot track local variables.

In our code, if local variable searchText gets modified by a function, react will have no clue about the change. In code the variable might be used in so many places, react will not know why the local variable got modified and it won't even display the updated change in UI. Because these variables are stateless. They don't maintain their state and as we already know React only tracks state variables.

Note: If we want our variables to be in sync with the UI state, we need to use state variables so that React can keep track of these state variables by maintaining a state.

Data Binding in react

In react we have 2 types of data binding

* One-way data binding where date binding happens in one direction

* Two-way data binding where data binding happens in both direction

In our code example, we have just placed a JavaScript local variable into the textbox. We are writing the local variable data into the textbox. This is React's one-way data binding. But our objective is to edit this textbox value, which means our local variable should get notified and change its value when the text box value gets changed. In other words, we are reading the data from the text box and updating our local variable.

This concept of reading and writing the data simultaneously is called Two-way data binding.

Up to this point, we have seen one-way data binding in action. Let's introduce two-way data binding and to implement Two-way data binding we use a hook named use state hook.

What's a Hook?

A normal JavaScript function written by Facebook developers which gives us some functionalities. There are different types of hooks. One of the common hooks is useState Hook.

What is a useState Hook?

* useState hook is a hook that react provides to create state variables in a functional component.

* useState hook returns an array of which first element is Name of the variable, second element is a setter function that changes the variable.

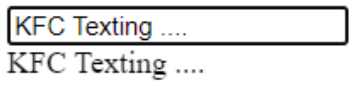
* UseState function expects an argument which initialises the variable.

Using react state variable using useState hook

Body.js App.js

src > components > Body.js > Body > restaurantList.map() callback

```
1   import { useState } from "react";
2   import { restaurantList } from "../constants";
3   import RestaurantCard from "./restaurantCard";
4
5   const Body = () => {
6     const [searchText, setSearchText] = useState("KFC");
7     return (
8       <>
9         <div className="search-container">
10          <input
11            type="text"
12            className="search-input"
13            placeholder="Search"
14            value={searchText}
15            onChange={(e) => setSearchText(e.target.value)}
16          />
17        </div>
18        {searchText}
19        <div className="restaurant-List">
20          {restaurantList.map((restaurant) => {
21            return (
22              <RestaurantCard
23                {...restaurant.data.data}
24                key={restaurant.data.data.id}
25              />
26            );
27          })}
28        </div>
29      </>
30    );
31  };
32
33  export default Body;
```



KFC Texting ....

I have placed react variable next to the search input text

setSearchText is the setter function given by useState hook to update the react variable searchText.

React watches/tracks this searchText.

when searchText is updated or changed, on every change react rerenders the entire body component and updates only the changed portion in the DOM (which is the searchText next to the search button). Reconciliation is happening behind the scene. Diffing algorithm does all the hard work behind the scene.

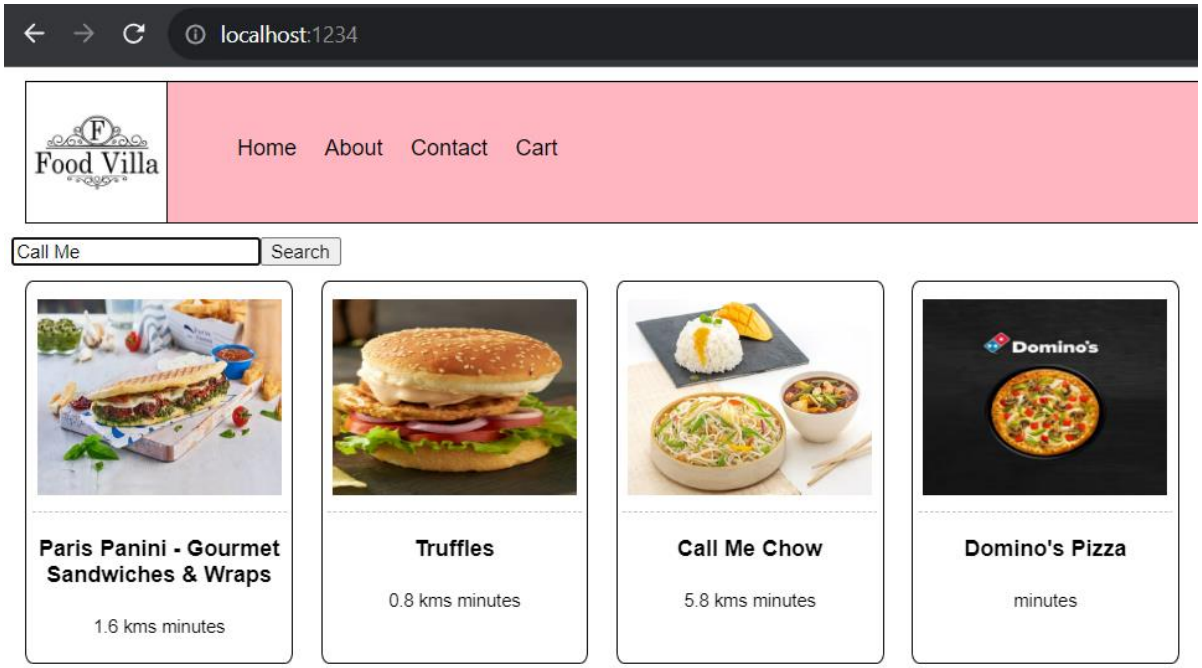
Note - React rerenders body component means react destroys the body component and creates the body component once again and it does it very fast. Now if I type anything in the search textbox in the UI the searchText got updated which we can see visually. Two-way data binding is happening here.

Search functionality Logic –

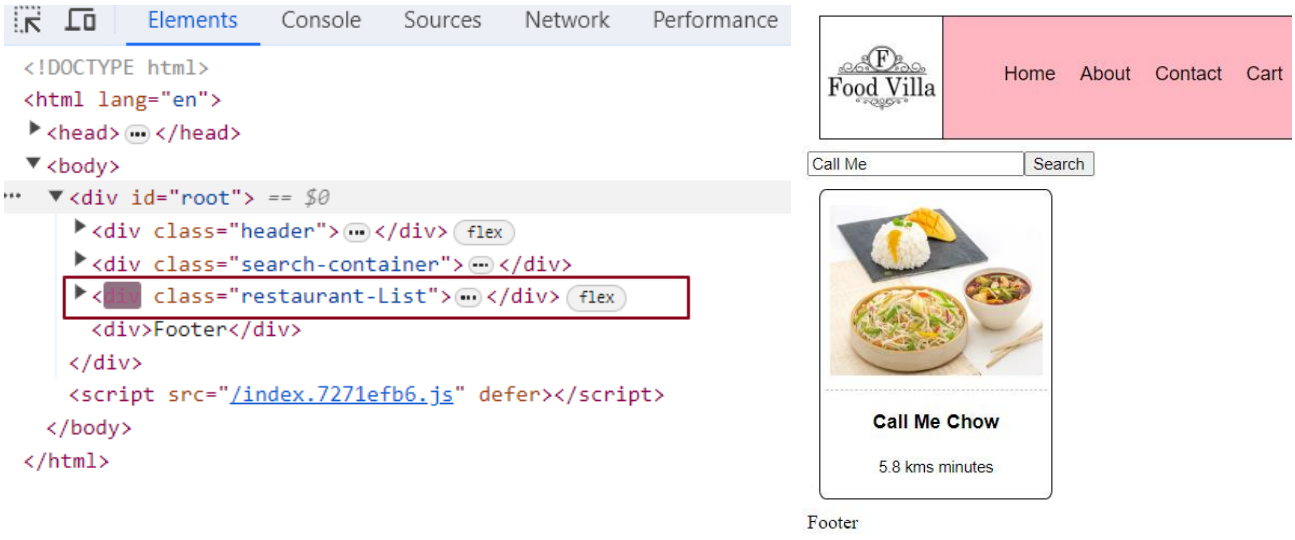
Body.js

src > components > Body.js > ...

```
import { useState } from "react";
import { restaurantList } from "../constants";
import RestaurantCard from "./restaurantCard";

// logic for filtering restaurants based on searchText value
function filterData(searchText, restaurants) {
  const filteredData = restaurants.filter((restaurant) =>
    restaurant.data.data.name.includes(searchText)
  );
  return filteredData;
}

const Body = () => {
  const [restaurants, setRestaurants] = useState(restaurantList);
  const searchvar = useState("KFC");
  const [searchText, setSearchText] = searchvar; // Array destructuring
  return (
    <>
      <div className="search-container">
        <input
          type="text"
          className="search-input"
          placeholder="Search"
          value={searchText}
          onChange={(e) => setSearchText(e.target.value)}
        />
        <button
          className="search-btn"
          onClick={() => {
            // Filtering restaurant data
            const filteredData = filterData(searchText, restaurants);
            // Updating restaurants state variable with filtered data
            setRestaurants(filteredData);
          }}
        >
          Search
        </button>
      </div>
      <div className="restaurant-List">
        {restaurants.map((restaurant, index) => {
          return (
            <RestaurantCard
              {...restaurant.data.data}
              key={restaurant.data.data.id}
            />
          );
        })}
      </div>
    </>
  );
};

export default Body;
```



When user type Call Me in the search bar and click on search button, filter data logic is invoked which returned the list of filtered restaurant data. Then we are updating the restaurantList state variable using setRestaurants(filteredData). When this code is run, restaurantList state is updated. On every state change react triggers its reconciliation process which uses differing algorithm to differentiate between actual and virtual DOM.In our case only the updated restaurantList is rerendered in the dom.



In this approach we have one issue i.e. Search functionality will work only once. If we search for the restaurant for second time, this functionality won't work. Why? We will discuss in the upcoming chapters

Note -

On every change in state react rerenders the component where the state is used

On every change in props react rerenders the components where the props are passed.