

Let's Get Hooked! (Namaste-React)

🔗 Please make sure to follow along with the whole **"Namaste React"** series, starting from Episode-1 and continuing through each subsequent episode. The notes are designed to provide detailed explanations of each concept along with examples to ensure thorough understanding. Each episode builds upon the knowledge gained from the previous ones, so starting from the beginning will give you a comprehensive understanding of React development.


I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-5** first. Understanding what **"Akshay"** shares in the video will make these notes way easier to understand.

So far, here's what we've learned in the previous episode

- We built a food ordering app that displayed restaurant cards using real-time data from an API (like Swiggy's).
- We also explored the concept of config-driven UI.

Before we start to learn today's episode, a common question many of us may encounter is:

Q) Why do we use React? Some of us might wonder why we don't just stick to HTML, CSS, and JAVASCRIPT for everything we've been doing?



Of course! It's absolutely possible to accomplish everything using regular **HTML**, **CSS** and **JAVASCRIPT** without using **REACT**. However, we chose React because it enhances our developer experience, making it more seamless and efficient.

Part-1

Introducing React-Hooks.

Before we begin, the first thing we need to do is clean up our app. Up until now, we've placed all our components inside a single `App.js` file, but this isn't considered good practice. It's best to create separate files for each component.

Q) How can we achieve this?

To achieve this, Let's discuss the folder structure. Currently, all our files are located at the root level of our project. (If you're following along with the course, you can view the current structure of your project on the left side of your code editor)

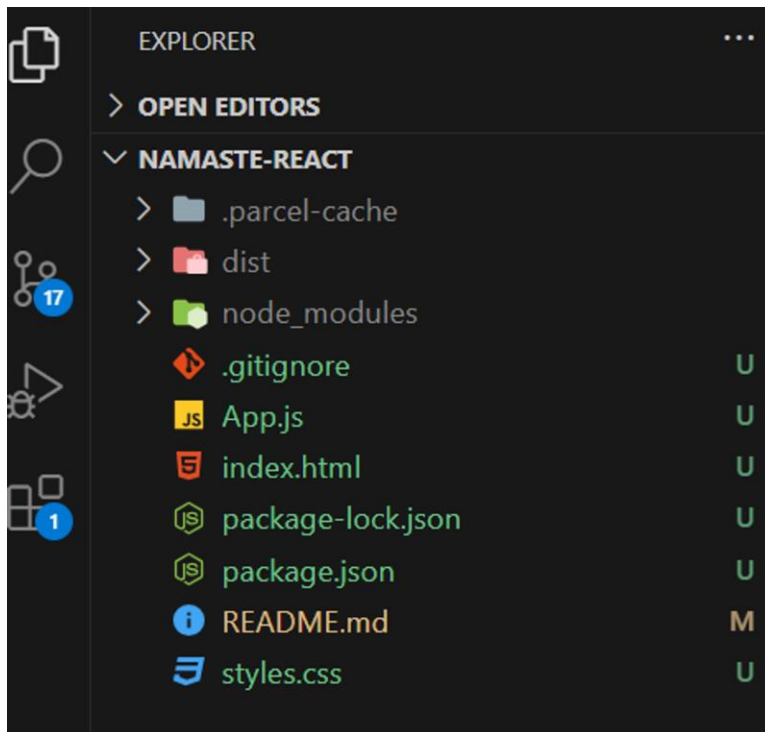


Fig 6.1

We are going to Restructure our project folder **"NAMASTE-REACT"**

- there is a very good convention in the industry that all the code in a React project is kept in a '**src**-folder', there is no compulsion to use a '**src**-folder' in a project. But here we are following what the industry follows.
- We are creating and moving our `App.js` file in the '**src**-folder' and whatever new files we create we put them in the '**src**folder'.

[?] NOTE: Don't forget to update the path of the `App.js` file in `index.html` other we will get an Error.

The best practice is to make separate files for every component.

We have the following components.

1. Header
2. RestaurantCard
3. Body

We put all the above components inside the folder named '**components**' (child-folder) which has been placed inside the '**src**- folder' (parent folder). When we are creating separate component files inside the '**components**-folder' always start with a capital letter like.

In this course, we are using (.js) as an extension.

1. Header.js
2. RestaurantCard.js
3. Body.js

NOTE: We can use (.jsx) as an extension instead of (.js) it's

up to the developer's wish.

1. Header.jsx
2. RestaurantCard.jsx
3. Body.jsx

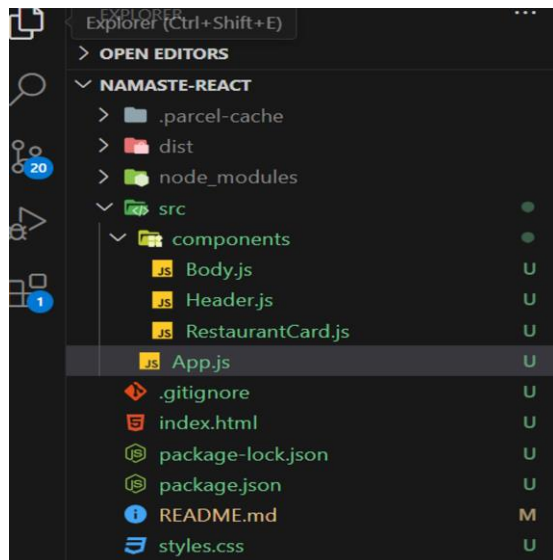



Fig 6.2

 NOTE: NO Thumb Rule to use this convention, we could use any folder structure you wish.

Understanding Export and Import in React.

Two types of export/import in React, We will understand each of them in details.

1.Default export/import.

2.Named export/import.

As we know we move each component's code and create new files individually. Still, Our React project throws an error because our `App.js` file doesn't have components in it and we are using components inside the `App.js` to solve this we need to import the components from their respective files which have been kept inside the *components-folder* in `src`. (refer fig 6.2)

to understand it well let's take an example of the `Header.js` component.

Example:

The `Header.js` component is missing in `App.js` so we are not able to use it, if we try to use it, It throws an error, To solve this, here we have to import `Header.js` inside the `App.js` and before the import, we have to 1st initially export `Header.js` component.

1.Default export/import.

Step-1 (export)————>

We use the 'export' and 'default' keywords with the component name at the end of the component file. In (figure 6.3) on line-no-19 we see that we are exporting Header component. Understand the Syntax properly.

```
// Syntax
export default Header;

// Or we can write with extension.
export default Header.js;
```

A screenshot of a code editor with a dark background. It shows a JavaScript file with several lines of code. Line 3 is a comment '//Header'. Line 4 is a const declaration for navItems. Line 10 is a closing brace for a function. Line 11 is a blank line. Line 12 is a const declaration for Header as a function. Line 17 is a closing brace for the Header function. Line 18 is a blank line. Line 19 is the line 'export default Header;' which is highlighted in blue. Line numbers 3, 4, 10, 11, 12, 17, 18, and 19 are visible on the left side of the editor.

```
3 //Header
4 > const navItems = (...
10 );
11
12 > const Header = () => (...
17 );
18
19 export default Header;
```

Fig 6.3

Step-2 (import)————>

In (fig 6.4), we import the Header on line-no-3, inside `App.js` We use 'import' with the component name at the start of the file. Understand the Syntax properly.

```
// Syntax
import Header from "../components/Header"
```

```

1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import Header from "../components/Header";
4
5
6
7
8

```

Fig 6.4

NOTE:

1. If you are using Vs-Code Editor during import it automatically tracks the path of the component and gives suggestions to us. so we don't have to worry about the path.
2. We don't have to put an extension in the file in the import statement. If want to put then completely Fine.

Follow the same method for the rest of the components.

In the case of RestaurantCard, we are using it inside the body component.

follow the same steps

Step-1 (export) —————> exporting RestaurantCard.js

Step-2 (import) —————> Importing RestaurantCard.js inside the Body.js

If we attempt to run our project, we'll encounter the '**resLists not defined**' error.

This happens because '**resList**' is being used inside '**Body.js**' without being defined in that file. While one solution is moving '**resLists**' inside '**Body.js**', it's not considered the best practice.

❓ NOTE:

resList contains hard-coded data and we never put any hard-coded data like the *source-URL* of *logo* and *images* inside the component file. That's not the best practice the industry follows.

Q) So where should we keep it?

- We've established a new directory within the '**src**' folder called '**utils**', and inside it, we've created a file named '**constants.js**' to store all hard coded data. You can choose any name for this file, but we've opted for lowercase letters since it's not a component. Additionally, we've included the source of logos and images in this file.
- We've stored our mock data for '**resList**' in a file named '**mockData.js**', which resides within the '**utils**' folder.
- We need to export data from '**constants.js**' and '**mockData.js**', and then import it into the necessary component files where it will be used

But there's a catch: If we intend to export multiple items simultaneously, from single file 'default export/import' won't work; it'll result in an error (refer Fig 6.5). For instance, in our '**constants.js**' file, we've stored **URLs** for logos and images using separate variables, which means default export/import won't be feasible (refer Fig 6.5).

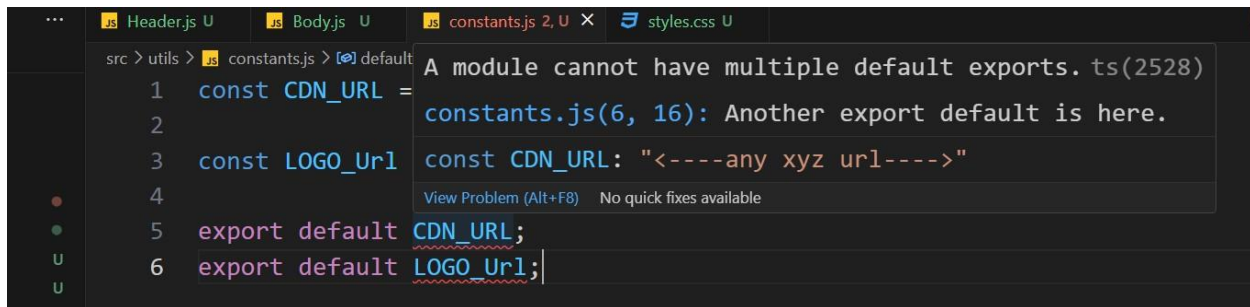


Fig 6.5

Instead, we can employ 'named export/import' to handle this scenario effectively.

2.Named export/import.

just write the 'export' keyword before the variables we want to export (refer Fig 6.6). We won't get any error.

Step-1 (export)—————>

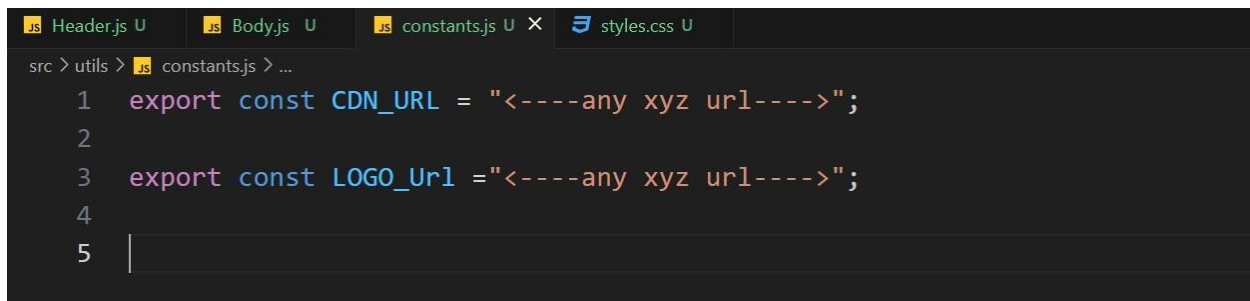


Fig 6.6

Step-2 (import)—————> when we import there is a slight difference in syntax we use curly braces.

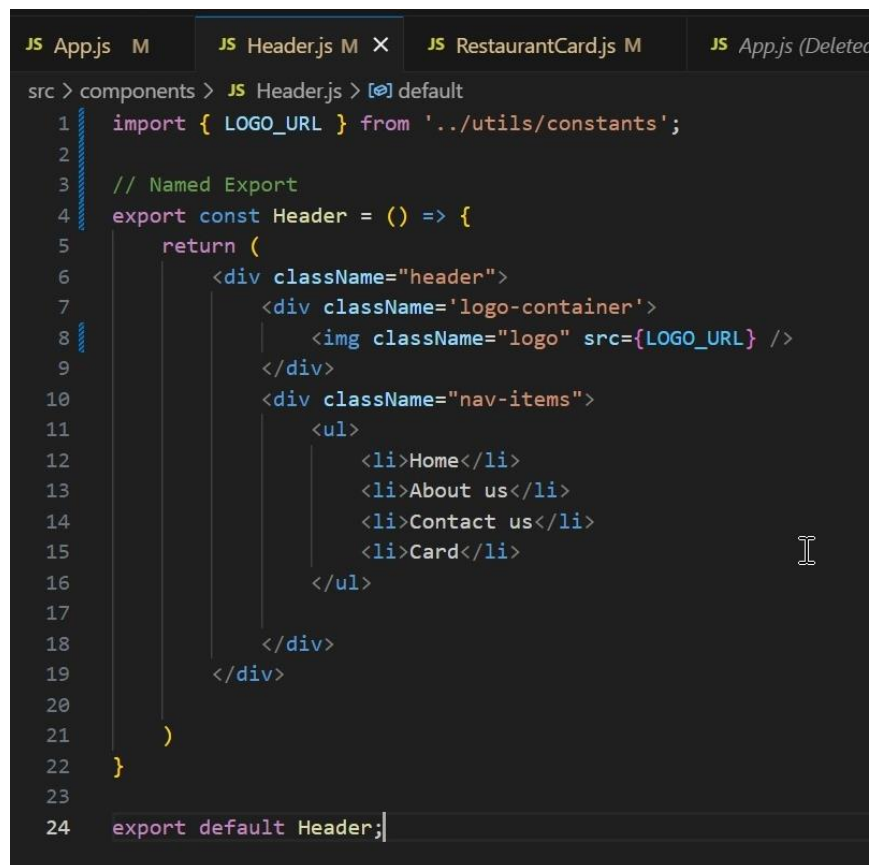
- We import LOGO_URL inside the `Header.js` and
- Import the CDN_URL inside the `RestaurantCard.js`

```
import { LOGO_URL } from "../utils/constants";
```

```
import { CDN_URL } from "../utils/constants";
```

Q) Can we use default export with named export ?

yes



```
JS App.js M JS Header.js M X JS RestaurantCard.js M JS App.js (Deleted)
src > components > JS Header.js > [0] default
1 import { LOGO_URL } from '../utils/constants';
2
3 // Named Export
4 export const Header = () => {
5   return (
6     <div className="header">
7       <div className='logo-container'>
8         <img className="logo" src={LOGO_URL} />
9       </div>
10      <div className="nav-items">
11        <ul>
12          <li>Home</li>
13          <li>About us</li>
14          <li>Contact us</li>
15          <li>Card</li>
16        </ul>
17      </div>
18    </div>
19  )
20 }
21
22 export default Header;
```

Part-2

Let's make our app more lively and engaging.

In this tutorial series, we're learning by actually building things. As we go along, we'll keep adding new features to our app.

For now, we're going to add a button. When you click on this button, it will show you the best-rated restaurants.

Inside the body component, instead of having a search box, let's replace it with a `<button>`. We'll add this button inside a `<div>` and give it any class name we want. Similarly, we'll also give a class name to the button, as per our preference.

We are adding `onclick` evenhandler inside the button, `onclick` has call-back function which will be called when we clicked on the button, there is condition when we clicked on the restaurant we will get a restaurant which avg-rating is more than 4.2 (avg-rating > 4.2)

Q) How these cards are coming on the screen?

- These cards are appear on the screen because we're using the `map()` method to go through each restaurant in our mock data (`resList`). This method helps us display the information from each restaurant on individual cards. so any thing changes in reslist the cards displayed on the screen will also change.
- We're simplifying our mock data to improve our understanding, or you can use the data provided below (which is the actual data from the Swiggy API).

```

let listOfRestaurant = [
  {
    data: {
      id: "255655",
      name: "Cake & Cream",
      clouinaryImageId: "ac57cc371e73f96f812613f58457aca3",
      areaName: "Jairaj Nagar",
      costForTwo: "₹200 for two",
      cuisines: ["Bakery", "Hot-dog", "pastery", "Cake", "Thi
ck- shake"],
      avgRating: 4.3,
      veg: true,
      parentId: "54670",
      avgRatingString: "4",
      totalRatingsString: "20+",
    },
  },
  {
    data: {
      id: "350363",
      name: "Haldiram's Sweets and Namkeen",
      clouinaryImageId: "25c3a7d394d6c5556b134385f7d665b0",
      avgRating: 4.6,
      veg: true,
      cuisines: [
        "North Indian",
        "South Indian",
        "Chinese",
        "Pizzas",
        "Fast Food",
      ],
    },
  },
]

```

```

        parentId: "391465",
        avgRatingString: "4.6",
        totalRatingsString: "100+",
    },
    {
        data: {
            id: "154891",
            name: "Rasraj Restaurant",
            clouinaryImageId: "egbr63ulc8h1zglliivd8",
            locality: "Civil Line",
            areaName: "Civil
Lines",
            costForTwo: "₹250 for two",
            cuisines: [
                "North Indian",
                "South Indian",
                "Street Food",
                "Chinese",
                "Pizzas",
                "Fast Food",
            ],
            avgRating: 4.2,
        },
    },
    {
        data: {
            id: "745961",
            name: "Balaji
Restaurant",
            clouinaryImageId:
            "b8672fe52944c3599ea324d99d608300",
            locality: "Sai
Rubber Stamp",
            areaName: "Jairaj Nagar",
            costForTwo: "₹149 for two",
            cuisines: ["South Indian",
"North Indian"],
            avgRating: 4.8,
            veg: true,
        },
    },

```

```

}, {
  data: {
    id: "798745",
    name: "Friends
Restaurant",
    cloudinaryImageId:
    "b14cd9fc40129fcfb97aa7e621719d07",
    locality: "Gayatri
Nagar",
    areaName: "Jairaj Nagar",
    costForTwo:
    "₹150 for two",
    cuisines: ["North Indian", "Chinese",
    "Biryani", "Tando or", "Kebabs"],
    avgRating: 4.2,
    parentId: "84308",
  },
}, {
  data: {
    id: "314737",
    name: "RASOI
the KITCHEN",
    cloudinaryImageId:
    "yjjymo9nhyn7rhvafsr3",
    locality: "Sriram Chowk",
    areaName: "Bazar Ward",
    costForTwo: "₹200 for two",
    cuisines: ["North Indian", "Maharashtrian", "Chinese",
    "Thalis"],
    avgRating: 3.9,
    parentId: "167341",
  },
}, {
  data: {
    id: "201454",
    name:
    "Morsels restaurants",
    cloudinaryImageId:
    "aafe71251ef5328784652dc838cd91f3",
    locality: "Bazar
Ward",
    areaName: "Chandrapur Locality",
  },
},

```

```

        costForTwo: "₹300 for two",          cuisines:
["North Indian", "South Indian"],
avgRating: 3.2,          veg: true,          parentId:
"139266",          avgRatingString: "4.2",
    },
    {
        data: {          id: "266124",          name:
"Trimurti Restaurant",          clouinaryImageId:
"8135c0066b06e2925c66930be4e9ffb5",          locality: "Bazar
Ward",          areaName: "Chandrapur Locality",
costForTwo: "₹150 for two",          cuisines: ["Desserts"],
avgRating: 3.9,          veg: true,          parentId: "217751",
avgRatingString: "4.4",
    },
    {
        data: {          id: "509254",          name: "Saha
Restaurant",          clouinaryImageId: "z1ez4uc9idul2uj2v87g",
areaName: "Jairaj Nagar",          costForTwo: "₹300 for two",
cuisines: ["North Indian", "Biryani", "Thalis", "Bevera
ges"],          avgRating: 3.7,          parentId: "174585",
avgRatingString: "3.7",
    },

```

```
    },  
  ];
```

We utilize the data mentioned above, incorporating a condition within the callback function. This condition triggers when we click on the button, displaying only the restaurants whose average rating is above 4.2. That's what we are expecting.

```
<button  
  onClick={() => {  
    const filterLogic = listOfRestaurant.filter((res)  
    => {  
      return res.info.avgRating > 4.2;  
    });  
    console.log(filterLogic);  
  }}  
>  
  Top Restaurant  
</button>
```

Fig 6.7

Despite implementing the code in Figure 6.7, no visible changes occur on the screen. However, we successfully filter the data, which we can confirm by checking the filtered results using `console.log(filterlogic).`



Note:

whenever we have react App we have a UI layer and data layer, UI layer will display what is being sent by the data layer.

Q) How can we display filtered restaurants dynamically on UI(display screen) ?

Here, we're utilizing data retrieved from the `listOfRestaurant` variable, which stores an array of objects. It's treated as a regular variable within our codebase. However, for this functionality, we require a superpowerful React variable known as a 'state variable'.

Q) How do we create Super-powerful variable?

for that we use 'React Hooks'.

Q) What is Hook?

It's simply a regular JavaScript function. However, it becomes powerful when used within React, as it's provided to us by React itself. These pre-built functions have underlying logic developed by React developers. When we install React via npm, we gain access to these superpowers.

Two crucial hooks we frequently utilize are:

1. `useState()`
2. `useEffect()`

1. `useState()`

(import)————>

- We are using `useState()` inside the body component to create a 'state variable '. Look at the syntax below.

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

First, we have to import as a named import from 'react'.

```
// syntax of useState()
const [listOfRestaurant] = useState([]);
```

In the provided code, we pass an empty array `[]` as the initial value inside the `useState([])` method. This empty array serves as the default value for the `listOfRestaurant` variable.

If we pass the `listOfRestaurant` were we stored all the restaurant data inside the `useState()` as the default data, it will render the restaurants on the screen using that initial data.

Q) How could we modify the list of restaurant?

To modify we pass the second argument `setListOfRestaurant` we can name as we wish. `setListOfRestaurant` used to update the list.

```
import { useState } from "react";
// syntax of useState()
const [listOfRestaurant , setListOfRestaurant] = useState
([]);
```

Q) How can we display filtered restaurants dynamically on UI(display screen)? I am repeating the same question which we had previously.

We are using `setListOfRestaurant` inside the call-back function to show the filtered restaurant. on clicking the button, we will see the Updated top-rated restaurant.

```

<button
  className="filter-btn"
  onClick={() => {
    const filtertheRestaurant = listOfRestaurant.filter((res)
=> {
      return ( res.data.avgRating > 4));
    });
    setListOfRestaurant(filtertheRestaurant);
  }}
>
  Top Rated Restaurant
</button>;

```



NOTE:

- The crucial point about State variables is that whenever they update, React triggers a reconciliation cycle and re-renders the component.
- This means that as soon as the data layer changes, React promptly updates the UI layer. The data layer is always kept in sync with the UI layer.
- To achieve this rapid operation, React employs a [reconciliation algorithm](#), also known as the [diffing algorithm](#) or [React-Fibre](#) which we will delve into further below.

React is often praised for its speed, have you ever wondered why? ?

At the core lies `React-Fiber` - a powerhouse reimplementation of React's algorithm. The goal of React Fiber is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is incremental rendering: the ability to split rendering work into chunks and spread it out over multiple frames.

These days, we can use ***JavaScript*** and ***React*** alongside popular libraries like ***GSAP*** (GreenSock Animation Platform) and ***Three.js***. These tools allow us to create animations and 3D designs using the capabilities of ***JavaScript*** and ***React***..

But how does it all work behind the scenes?

When you create elements in React, you're actually creating ***virtual DOM objects***. These virtual replicas are synced with the ***real DOM***, a process known as "Reconciliation" or the React

"diffing" algorithm.

Essentially, every rendering cycle compares the new UI blueprint (updated VDOM) with the old one (previous VDOM) and makes precise changes to the actual DOM accordingly.

It's important to understand these fundamentals in order to unlock a world of possibilities for front-end developers!

Do you want to understand and dive deep into it?

Take a look at this awesome React Fiber architecture repository on the web: <https://github.com/acdlite/react-fiberarchitecture>

Self-Notes

Chapter 05 - Let's get Hooked!

Why do we use react?

It makes your developer experience easy, it makes you write less code and do more on the web pages. This is the major job of ui library. Using react you can build large scalable production ready performant application. React makes coding experience very fast and optimize something on web page so that things happen very fast.

File structure-

<https://legacy.reactjs.org/docs/faq-structure.html>

Hardcoded data is the mock data ex- Json

If you have to export multiple things how do you do that so then we use something known as named export. It is very easy you can write export in front of your variable or constant. You have to export multiple things from the same file. You will always have to use a named export.

There are of two types of export/import –

Named export and the default export.

Or

Default export/import-

Export default component.
Import component from 'path'

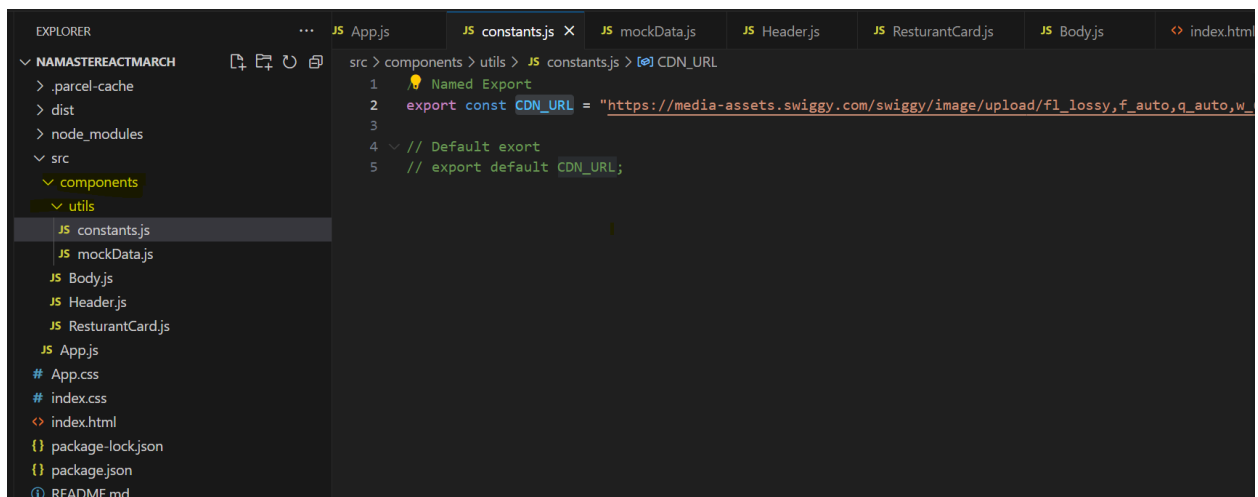
Named export/import-

Export const component;
Import {component} from "path"

When we will import there will be a slight difference. So, whenever you have your named exports, you have to write curly braces and then whatever you need to import. This is how you import export a named export from your file.

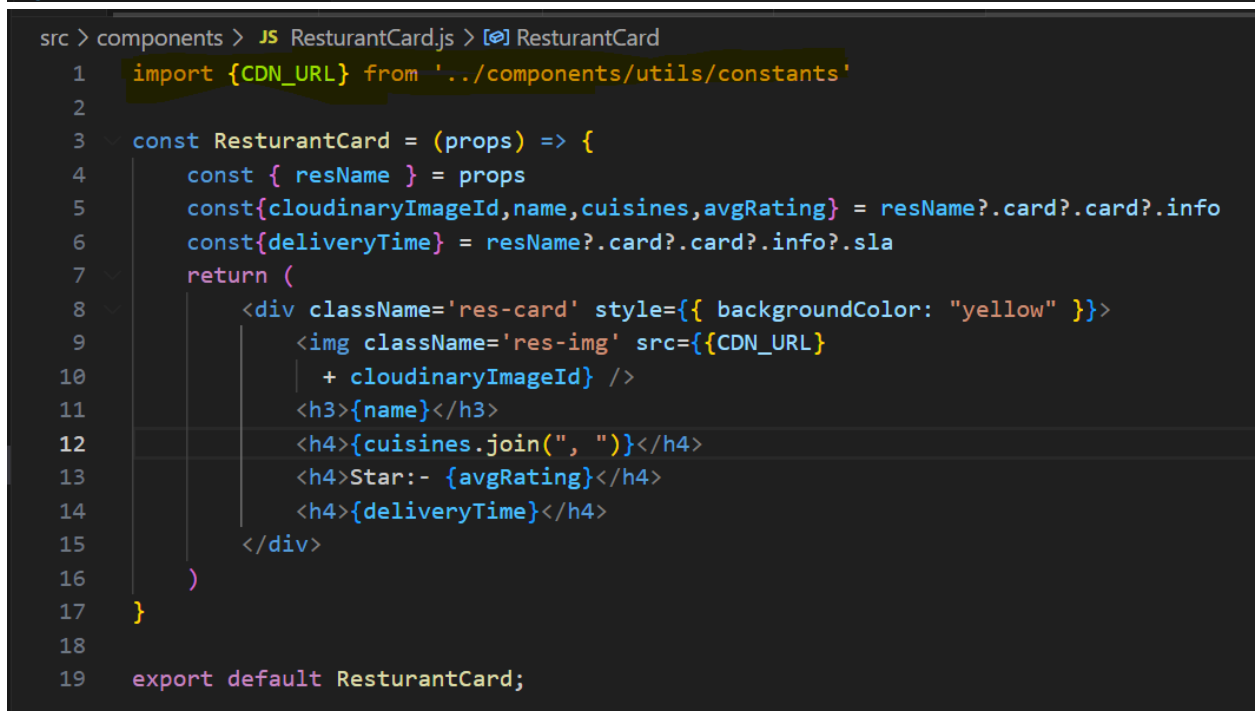
If we use default export, then no need to use curly bracket at the time of import.

Note- In one file only we can have one export otherwise it will throw an error.



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure for 'NAMASTEAREACTMARCH'. The 'src' directory is expanded, showing 'components' and 'utils'. The 'constants.js' file in the 'utils' folder is selected. The main editor area shows the content of 'constants.js', which defines a named export 'CDN_URL' and a default export.

```
src > components > utils > JS constants.js > [0] CDN_URL
1  ⚡ Named Export
2  export const CDN_URL = "https://media-assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_
3
4  // Default export
5  // export default CDN_URL;
```



The screenshot shows the VS Code editor with the 'ResturantCard.js' file open. The file path in the breadcrumb is 'src > components > JS ResturantCard.js > ResturantCard'. The code defines a functional component 'ResturantCard' that takes 'props' and returns a JSX element. It imports 'CDN_URL' from the constants file and uses it to set the 'src' attribute of an image. The component also displays the restaurant's name, cuisines, average rating, and delivery time.

```
src > components > JS ResturantCard.js > ResturantCard
1  import {CDN_URL} from '../components/utils/constants'
2
3  const ResturantCard = (props) => {
4    const { resName } = props
5    const{cloudinaryImageId,name,cuisines,avgRating} = resName?.card?.card?.info
6    const{deliveryTime} = resName?.card?.card?.info?.sla
7    return (
8      <div className='res-card' style={{ backgroundColor: "yellow" }}>
9        <img className='res-img' src={{CDN_URL}
10          + cloudinaryImageId} />
11        <h3>{name}</h3>
12        <h4>{cuisines.join(", ")}</h4>
13        <h4>Star:- {avgRating}</h4>
14        <h4>{deliveryTime}</h4>
15      </div>
16    )
17  }
18
19  export default ResturantCard;
```

Other example-

```

export const Header = () => {
  return (
    <div id="heading">
      <h1>Logo</h1>
      <div className='nav-item'>
        <ul className='nav-item-ul'>
          <li>Home</li>
          <li>About Us</li>
          <li>Contact Us</li>
          <li>Card</li>
        </ul>
      </div>
    </div>
  )
};

```

```

JS App.js > ...
import React, { Component } from 'react';
import ReactDOM from 'react-dom/client';
import {Header} from './components/Header'
import Body from './components/Body'

```

Implementation- we will have a button. That button is like top rated restaurant. It is like a filter. If you will click on that filter. So, our page filters out all top-rated restaurants. The function has some logic written behind it.

Hook-

Hook is just a normal JavaScript function which is given to us by react. It is the pre-built. That function has some logic written behind it scenes inside react. That function is a utility function given by react. We have to import these utility functions.

There are two very important hook-

1. **useState** – super powerful state variable in react

First we have to import useState with name import

```
import {useState} from 'react'.
```

react is import as the default import

```
import React from 'react'.
```

// this react comes from the node module.

Use state is to create state variable that is why the name is use state variable.

Why it is called state variable because it maintains state of you're your component.

We are creating a local state variable inside the component.

```
//local state variable - super powerful variable
const [listOfRestaurant] = useState([]);
const [listOfRestaurant] = useState(null);

//Normal JS Variable
let listOfRestaurant = [];
let listOfRestaurant = null;
```

Pass the json data inside the use state. It works absolutely just like the normal js variable.

```
//local state variable - super powerful variable
const [listOfRestaurant] = useState([
  {
    "card": {
      "card": {
        "@type": "type.googleapis.com/swiggy.presentation.food.v2.Restaurant",
        "info": {
          "id": "331999",
          "name": "Shri Chamunda Purohit Dinning Hall",
          "cloudinaryImageId": "prum3zpw3sfbvoog94xu",
          "locality": "3rd Phase",
          "areaName": "Hinjawadi",
          "costForTwo": "₹200 for two",
          "cuisines": [
            "Indian"
          ],
          "avgRating": 4.2,
          "veg": true,
          "parentId": "186312",
          "avgRatingString": "4.2",
          "totalRatingsString": "100+",

```

So, this setlistofrestaurant the update the list.

In normal js how we can update the value –

let list = [];

list = ["abc"] or list.push("abc")

Suppose I have to modify list of restaurants in react. How do I have to modify it by a function and function is the second parameter in this array.

```
//local state variable - super powerful variable
const [listOfRestaurant, setlistOfRestaurant] = useState([
  {

```

Now I want to update the list of variables by onclick on UI.


```

return (
  <div id="search">
    <div className='filter'>
      <button className='filter-btn' onClick={() => {
        const filteredList = listOfRestaurant.filter(
          (res) => res.card.card.info.avgRating > 4
        )
        setlistOfRestaurant(filteredList);
      }}>
        Top Rated Resturant</button>
      </div>
    </div>
  )
)

```

useState is powerful it keeps the UI in sync with the data layers.

Whenever state variable updates or changes react re-render the components.

Whenever basically anywhere the setListOfResturant of restaurant is called react will remove this body and update it properly. It will very quickly the UI.

We can use mock data inside of it.

```

import { useState } from 'react'
import RestaurantCard from './RestaurantCard'
import restList from './utils/mockData';
const Body = () => {
  //local state variable - super powerful variable
  const [listOfRestaurant, setlistOfRestaurant] = useState(restList);
  // const[listOfRestaurant] = useState(null);

```

React will keep your UI in sync with the data layer, when you have a local state variable, *as soon as data layer update as soon as you UI layer update*. How it will update – by re-rendering the component. It is the whole logic of react it does fast and very optimized way.

Why react is the most popular library and why it is fast – react will make these DOM operation super-fast and efficient. Or we can say that react is efficient DOM manipulation. Because it has the virtual DOM and it the object representation. React took that and built its code algo. over that virtual DOM. It has the diff algo. which is very efficient Dom manipulation. It can find out the diff and update the UI

React using reconciliation algo. and this reconciliation algo. is also known as react fiber. (it comes up in react16)- <https://github.com/acdlite/react-fiber-architecture>

When somethings changes on the UI, this is known as the reconciliation and this new algo. came in react 16. This is known as react fiber.

- This react fiber is a new way of finding the diff. and updating the DOM
- It is basically findout the difference between virtual DOM and (it is the obj). then it will update the real DOM and that's how react becomes faster.

You know finding the diff. between the html code is tough and finding out the difference between the objects is fast. So it find out the diff between the object.

React keeps a track of all UI, all this DOM nodes, all the HTML as a virtual DOM.

Virtual DOM is kind of like an object representation. It represent HTML.

So when I click on the button new object will create, new object is formed and react find out the difference between two objects then it will update the actual DOM. It will not find out the difference HTML

Ex- In DOM we have dom, dom is like a tree. Suppose we have the res-container and it has 15 res card and suppose my UI changes to filtering these cards to 3 cards.

Whenever you have this UI react creates a virtual DOM.

What is actual DOM – actual DOM are the tags

What is virtual DOM – virtual DOM is the representation of the actual DOM. It is basically that object. It is basically those element

When I console.log-

```
const AppLayout = () => {
  console.log( <Body />)
  return (
    <div id="container1">
      <Header />
      <Body />
    </div>
  )
}
```

This is virtual DOM. It creates the object. This object is basically the react virtual DOM. It is basically the JavaScript object.

```
App.js:55
{
  $$typeof: Symbol(react.element), key: null,
  ref: null, props: {}, type: f, ...
}
i
  $$typeof: Symbol(react.element)
  key: null
  ▶ props: {}
  ref: null
  ▶ type: ()=> {...}
  ▶ _owner: FiberNode {tag: 0, key: null, s
  ▶ _store: {validated: false}
  _self: undefined
  ▶ _source: {fileName: 'src/App.js', lineN
  ▶ [[Prototype]]: Object
```

Diff algorithm –

It basically finds out the difference between two virtual DOM.

It finds the difference between the updated virtual DOM and previous virtual DOM.

Example – suppose I have the restaurant container had seven restaurant cards, and I clicked the button on the UI.

It basically tries to find out the difference between old virtual DOM and the new virtual DOM. What it will do it will calculate the difference and it will then actually update the DOM on every render cycle.

Whenever there is a change in any state variable, react will find out the diff between virtual DOM and it will re-render our component, it will update the DOM.

As soon as call the `setlistofrestaurant` function calls react start the reconciliation algo., it will re-render the application, it will find out the diff update the UI.

Array destructuring-

```
// This array destructuring, usestate return the array
const [listOfRestaurant, setlistOfRestaurant] = useState(restList);

const [listOfRestaurant, setlistOfRestaurant] = arr;
const listOfRestaurant = arr[0]
const setlistOfRestaurant = arr[1]
```

2. `useEffect` – refer next video

Assignment

- What is the difference between **Named** Export, **Default** export and *** as** export?
- What is the importance of `config.js` file
- What are React Hooks?
- Why do we need a **`useState`** Hook?

Coding Assignment:

- Clean up your code
- Create a Folder Structure for your app
- Make different files for each Components

- Create a config file
- Use all types of import and export
- Create a Search Box in your App
- Use useState to create a variable and bind it to the input box
- Try to make your search bar work

References

- Code Link - <https://bitbucket.org/namastedev/namaste-react-live/src/master/>