





Exploring The World! (Namaste-React)


 I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-6** first. Understanding what **"Akshay"** shares in the video will make these notes way easier to understand.

Here are two quick stories:

 In the web development world, we often hear that the trend is leaning from **'Monolithic'** towards the **'Microservices'** architecture.

 **Atlassian's Shift:** Back in 2018, **'Atlassian'** faced some growing pains. To keep up with demand and stay flexible, they switched to microservices. This move helped them stay agile and scale up smoothly.

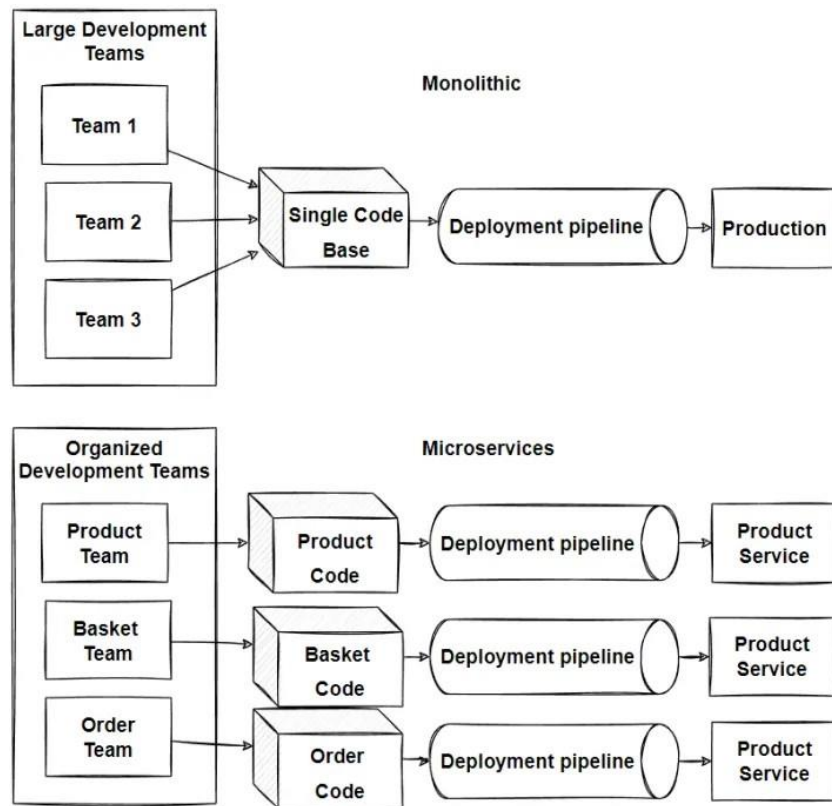
 **Netflix's Transformation :** The popular video streaming platform over in the world **'Netflix'** runs on AWS. They started with a monolith and moved to microservices.

 In today's episode, we discussed the trend towards lighter and more adaptable web architectures.

PART-1

Q) What are 'Monolithic' and 'Microservices' architectures exactly?

Understanding **'Monolith'** and **'Microservices'** architectures is a big deal in software development, but as developers, it's important to grasp the basics. So, in this episode, we'll break it down into simple terms.



Monolithic Architecture

In the past, we used to build large projects where everything was bundled together. Imagine building an entire application where all the code-APIs, user interface, database connections, authentication, even notification services—resides in one massive project with single code base.

- **Size and Complexity Limitation:** Monolithic applications become too large and complex to understand
- **Slow Startup:** The application's size can slow down startup time.
- **Full Deployment Required:** Every update requires redeploying the entire application.
- **Limited Change Understanding:** It's hard to grasp the full impact of changes, leading to extensive manual testing.
- **Difficult Continuous Deployment:** Implementing continuous deployment is challenging.
- **Scaling Challenges:** Different modules may have conflicting resource needs, making scaling difficult.
- **Reliability Concerns:** Bugs in any module can crash the whole application, affecting availability.

- **Adoption of New Technologies:** Making changes in frameworks or languages is expensive and time-consuming since it affects the entire application.

Microservices Architecture

The idea is to split your application into a set of smaller, interconnected services instead of building a single monolithic application. Each service handles a specific job, like handling user accounts or managing payments. Inside each service, there's a mini world of its own, with its own set of rules (business logic) and tools (adapters). Some services talk to each other in different ways, like using REST or messaging. Others might even have their own website!

- **Simpler Development:** Microservices break down complex applications into smaller, easier-to-handle services. This makes development faster and maintenance easier.
- **Independent Teams:** Each service can be developed independently by a team focused on that specific task.
- **Flexibility in Technology:** Developers have the freedom to choose the best technologies for each service, without being tied to choices made at the project's start.
- **Continuous Deployment:** Microservices allow for independent deployment, enabling continuous deployment for complex applications.
- **Scalability:** Each service can be scaled independently, ensuring efficient resource usage.
- **Separation of Concerns:** With each task having its own project, the architecture stays organized and manageable.
- **Single Responsibility:** Every service has its own job, following the principle of single responsibility. This ensures focused and efficient development.

Q) Why Microservices?

Breaking things down into microservices helps us work faster and smarter. We can update or replace each piece without causing a fuss. It's like having a well oiled machine where each part does its job perfectly.

Q) How do these services interact with each other?

In our setup, the UI microservice is written in React, which handles the user interface.

Communication Channels

These services interact with each other through various communication channels. For instance, the UI microservice might need data from the backend microservice, which in turn might need to access the database.

Ports and Domain Mapping

Each microservice runs on its specific port. This means that different services can be deployed independently, with each one assigned to a different port. All these ports are then mapped to a domain name, providing a unified access point for the entire application.

PART-2

Connecting to the External World

In this episode, we're going to explore how our React application communicates with the outside world. We'll dive into how our application fetches data and seamlessly integrates it into the user interface. It's all about understanding data exchange that makes our app come alive.

In our Body component, we're displaying a list of restaurants. Initially, we used mock data inside the `'useState()'` hook to create a state variable. However, in this episode, we're stepping up our game by fetching real-time data from Swiggy's API and displaying it dynamically on the screen. How cool is that? 🤖

Before diving in, let's understand two approaches to fetch and render the data :

1. Load and Render:

We can make the API call as soon as the app/page loads, fetch the data, and render it.

2. Render First Fetch Later:

Alternatively, we can quickly render the UI when the page loads we could show the structure of the web page, and then make the API call. Once we get the data, we re-render the application to display the updated information.

In React, we're opting for the second approach. This approach enhances user experience by rendering the UI swiftly and then seamlessly updating it once we receive the data from the API call.

PART-3

Today, we're diving into another important topic `useEffect()`. We've mentioned it before in a previous episode. Essentially, `useEffect()` is a *Hook* React provides us, it is a regular JavaScript function, to help manage our components. To start exploring its purpose, let's first import it from React.

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

`useEffect()` takes two arguments .

1. Callback function.
2. Dependency Array.

```
// Syntax of useEffect()  
// We passed Arrow function as callback function.  
  
useEffect(() => {}, []);
```

Q) When will the callback function get called inside the `useEffect()`?

Callback function is getting called after the whole component get rendered.

In our app we are using `useEffect()` inside Body component. So it will get called once Body component complete its render cycle.

If we have to do something after the rendercycle complets we can pass it inside the `useEffect()`. this is the actual use case of *useEffect*. It is really helpful to render data which we will get after the `fetch()` operation and we are going to follow second approach which we have discussed already.

Q) Where we fetch the data?

inside the `useEffect()` we use `fetchData()` function to fetch data from the external world. don't worry we will see each and every steps in detail. logic of fetching the data is exactly the same that we used to do in javascript. here we are fetching the swiggy's API.


IMPORTANT:


If getting difficulty to understand

`fetch()`, Don't worry please read about how `fetch()` works.

Fetch basic concepts - Web APIs | MDN

The Fetch API provides an interface for fetching resources (including across the network). It will seem familiar to anyone who has used XMLHttpRequest, but it provides a more powerful and

 https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Basic_concepts

 mdn web docs

Q) How can we use Swiggys API in our App?

We know that '`fetch()`' always return promise to us. we can handle response using '`.then()`' method. but here we are using newer approach using '`async/await`' to handle the promise.

we convert this data to javascript object by using '`.json()`'

```
// here once the body component would have been rendered , we will fetch the data
useEffect(() => {
  fetchData();
}, []);

const fetchData = async () => {
  const data = await fetch(
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=19.9615398&lng=79.2961468&is-seo-homepage-enabled=true&page_type=DESKTOP_WEB_LISTING"
  );

  const json = await data.json();
  console.log(json);
};
```

Q) By using above code let's see can we able to call swiggy's api successfully or not?

We got an error  (refer fig 6.1)

```
Access to fetch at 'https://www.swiggy.com/dapi/restaurants/list/v5?lat=19.9615398&lng=79.2961468&is-seo-homepage-enabled=true&page_type=DESKTOP_WEB_LISTING' from origin 'http://localhost:1234' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
GET https://www.swiggy.com/dapi/restaurants/list/v5?lat=19.9615398&lng=79.2961468&is-seo-homepage-enabled=true&page_type=DESKTOP_WEB_LISTING net::ERR_FAILED 200 (OK)
Uncaught (in promise) TypeError: Failed to fetch
    at fetchData (Body.js:15:22)
    at Body.js:11:3
    at commitHookEffectListMount (react-dom.development.js:23150:26)
    at commitPassiveMountOnFiber (react-dom.development.js:24926:13)
    at commitPassiveMountEffects_complete (react-dom.development.js:24891:9)
    at commitPassiveMountEffects_begin (react-dom.development.js:24878:7)
    at commitPassiveMountEffects (react-dom.development.js:24866:3)
    at flushPassiveEffectsImpl (react-dom.development.js:27039:13)
    at flushPassiveEffects (react-dom.development.js:26984:14)
    at react-dom.development.js:26769:9
    at workLoop (scheduler.development.js:266:34)
    at flushWork (scheduler.development.js:239:14)
    at MessagePort.performWorkUntilDeadline (scheduler.development.js:533:21)
```

fig 6.1

Q) What is the reason we got that error?

Basically, calling swiggy's API from local host has been blocked due to CORS policy.

Q) What exactly the CORS policy is?

(Cross-Origin Resource Sharing) is a system, consisting of transmitting HTTP headers, that determines whether browsers block frontend JavaScript code from accessing responses for cross-origin requests.

In simpler terms, CORS (Cross-Origin Resource Sharing) is a security feature implemented by browsers that restricts web pages from making requests to a different origin (domain) than the one from which it was served. Therefore, when trying to call Swiggy's API from localhost, the browser blocks the request due to CORS restrictions.

IMPORTANT:
If getting difficulty to understand CORS, Don't worry please read the below document

CORS - MDN Web Docs Glossary: Definitions of Web-related terms | MDN
CORS (Cross-Origin Resource Sharing) is a system, consisting of transmitting HTTP headers, that determines whether browsers block frontend JavaScript code from accessing responses for cross-origin requests.
 <https://developer.mozilla.org/en-US/docs/Glossary/CORS>

 mdn web docs

<https://www.youtube.com/watch?v=tcLW5d0KAYE>

IMPORTANT!

To prevent CORS errors when using APIs, utilize a CORS extension and activate it.

IMPORTANT!

In future swiggy definately change their API data so always remember go to swiggy's website and copy the updated URL of API to fetch data.

To show the new data on our page, we just need to update the `'listOfRestaurant'` with the fresh info. React will then refresh the page to display the updated data.

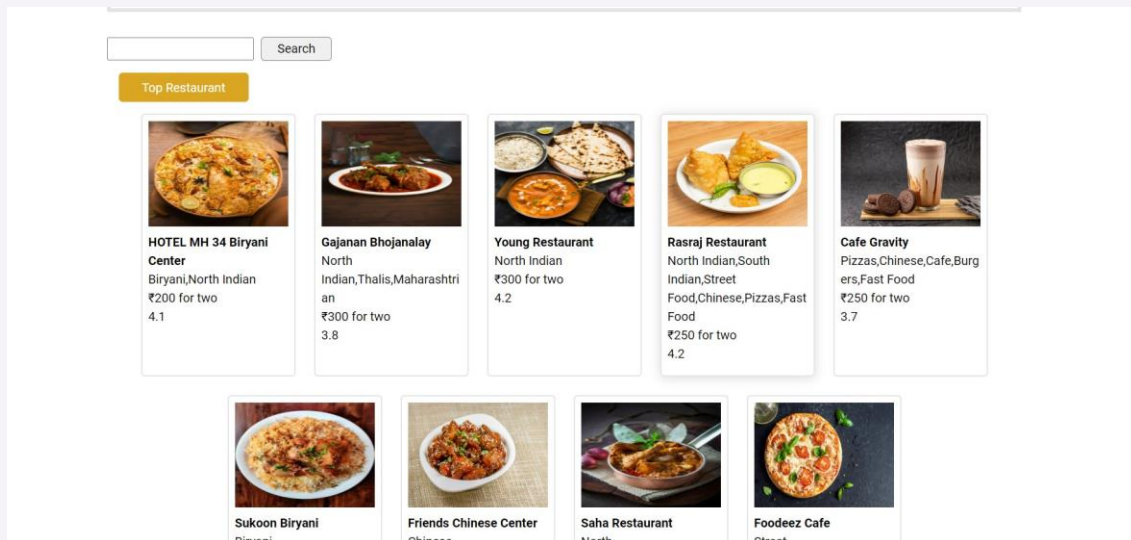
Q) How do we Update the data?

We're updating the `'listOfRestaurant'` using a state variable we've already defined. We simply use the `'setlistOfRestaurant()'` function to replace the old data with the new.

```
const fetchData = async () => {
  const data = await fetch(
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=19.9615398&lng=79.2961468&is-seo-homepage-enabled=true&page_type=DESKTOP_WEB_LISTING"
  );
  const json = await data.json();
  setListOfRestaurant(
    json.data.cards[4].card.card.gridElements.infoWithStyle.restaurants
  );
};
```


Our App making live call to the external world getting the data and render on the screen. Amazing

Congratulations we have successfully fetch and render the data 🍕



🔔 IMPORTANT:

As we delve into the JSON data, it's essential to note its complexity. Our focus lies solely on extracting cards which have restaurant information for our project.

Attempting to directly implement the code snippet provided here definitely results in errors due to potential changes in Swiggy's API structure. Your understanding is greatly appreciated during this phase. Focus on the concept of whatever Akshay taught us in this Episode.

NOTE: In the upcoming episode, 'Akshay' addresses all API-related issues, ensuring a smoother experience. So, don't stress—everything will be resolved in the upcoming episode.

Happy coding!

PART-4

After fetching the data, there's a noticeable one-second delay before it appears on the screen. This delay occurs because the APIs take some time to load. Improving this can enhance the user experience.

Q) How could we improve it?

To enhance the user experience, we could add a spinning loader that appears while we wait for the data to load from the APIs. This provides visual feedback to the user and indicates that the application is working to retrieve the information.

We could implement a condition to display a spinning loader if our list of restaurants hasn't received any data yet.

```
if (listOfRestaurant.length === 0) {  
  return <h1>loading. . .</h1>;  
}
```

Refreshing the page to see the result, but this isn't an ideal approach. Instead, we can enhance the user experience by implementing a '*Shimmer UI*'.

Shimmer UI

Shimmer UI is a technique that shows placeholder content while data is loading, reducing wait time and keeping users engaged.

Instead of displaying a generic "loading" message, we'll integrate a `<shimmer/>` component within our app to provide visual feedback while data is loading. this concept is known as 'conditional rendering'.

```
// conditional rendering  
if (listOfRestaurant.length === 0) {  
  return <Shimmer/>;  
}
```

PART-5

Use of ternary operator in our code base.

```
return listOfRestaurant.length === 0 ? (
  <Shimmer />
) : (
  <div className="container body">
    <div className="filter">
      <button
        className="filter-btn"

        onClick={() => {
          const filterLogic = listOfRestaurant.filter((res) => {
            return res.info.avgRating > 4.2;
          });
          setListOfRestaurant(filterLogic);
        }}
      >
        Top Restaurants
      </button>
    </div>
    <div className="RestaurantContainer">
      {listOfRestaurant.map((restaurant) => (
        <RestaurantCard key={restaurant.info.id} resData={restaurant} />
      ))}
    </div>
  </div>
);
```

Q) Why do we need State variable?

Many developers have this confusion today we will see that Why with the help of following example:

to understand this we will introduce on feature in our app is a **'login/logout'**

button

Inside Header component we are adding the button look at the code given below. also we want to make that login keyword dynamic it should change to logout after clicking. **step1 →**

We create `'btnName'` variable with login string stored in it and we are going use that btnName as a button text look at the code below **step2 →**


Upon clicking this button, it changes to 'logout'.

```
// Step 1-->
const btnName = "Login";

return (
  <div className="container header">
    <a>logo</a>
    {navItems}
    <button
      className="login"
      onClick={() => {
        btnName = "Logout";
      }}
    >
    {btnName}
  </button>
</div>
);
```

It's frustrating that despite updating the '

```
    <button
      className="login"
      onClick={() => {
        btnName = "Logout";
      }}
    >
    {btnName}
  </button>
</div>
);
```

But it will not change 

`btnName` value and seeing the change reflected in the console, the UI remains unchanged. This happens because we're treating '`btnName`' as a regular variable. To address this issue, we need a mechanism that triggers a UI refresh whenever '`btnName`' is updated

To ensure UI updates reflect changes in '`btnName`', we may need to use state management that automatically refreshes the UI when data changes. that the reason we need state variable '`useState()`'.

Let's utilize `reactBtn` as a state variable using `useState()` instead of `btnName`. Here's the code:

```
const [reactBtn, setReactBtn] = useState("login");
```

To update the default value of `reactBtn`, we use `setReactBtn` function.



NOTE:

In React, we can't directly update a state variable like we would use a normal JavaScript variable. Instead, we must use the function provided by the `useState()` hook. This function allows us to update the state and triggers a re-render of the component, ensuring our UI is always up-to-date with the latest state.

With the code provided below, we've enhanced the functionality of our app. Now, we can seamlessly toggle between "login" and "logout" states using a ternary operator. This addition greatly improves the user experience.

```
const [reactBtn, setReactBtn] = useState("login");

return (
  <div className="container header">
    <a>logo</a>
    {navItems}
    <button
```

```

        className="login"
        onClick={() => {
            reactBtn === "login"
              ? setReactBtn("logout")
              : setReactBtn("login");
        }}
      >
        {reactBtn}
      </button>
    </div>
  );

```



NOTE :

The interesting aspect of the above example is how we manage to modify a `const` variable like `'reactBtn'`, which traditionally isn't possible. However, because React rerenders the entire component when a state variable changes, it essentially creates a new instance of `'reactBtn'` with the updated value. So, in essence, we're not updating `'reactBtn'`; instead, React creates a new one with the modified value each time the state changes. This is the beauty of React.

PART-6

lets add another feature in our React app , search functionality.

When you input text into the search field, it provides suggestions based on the data related to restaurants that we already have.

step 1 →

Let's create a search bar within a `<div>` element and assign any class name of your choice to it. Additionally, we'll give class names to the input field and button inside the search bar.

step 2 →

Upon clicking the button, filter the restaurant cards and update the UI to retrieve data from the input box. To link our input to the button, we'll use the `value` attribute within the input field and bind that value to a local state variable. We'll create a local state variable named `searchText` along with a function named `setSearchText` to update the value. lets see below code will work or not by simple puting the call back function.

```
const [searchText, setSearchText] = useState("");

<div className="search">
  <input type="text" className="search-box" value={searchText} />
  <button className="searchBtn" onClick={() => {
    console.log(searchText);
  }}>
    Search
  </button>
</div>;
```

we could see that our input not taking value. we unable to type any thing.

- we knew already we have bind this searchText to the input field. what ever is inside the searchText variable will inside the value attribute of the input field.
- when we will change the value of input field by typing on it still it will tied to the searchText but searchText is not Updating . because default value of search text is empty string .this is most import point to understand whole concept. this input box not changed unless we change the search text

Q) How could we solve this problem ?

to solve this we have to add 'onChange' eventHandler inside the input field, so as soon as input changes the onChange call back function should also be changed the input text.

inside the onchange event handler we have event 'e' inside the call back . so access that typed input by using event 'e' see the code

```
<input type="text" className="search-box" value={searchText}
  onChange={(e) => {setSearchText(e.target.value)}} />
```

based on the on change we have make in the code now we can type inside the search box and see the output inside the console.



NOTE: when ever search text is change on the every key press state variable re render the component. its find the difference between every updated V-DOM with new text added inside the input field with older one.

Step 3 —>

We're currently filtering the list of restaurants to update the UI. When we type a word in the input field, it filters out the restaurant cards based on whether the typed word matches any restaurant names. However, we're facing a challenge with the input field being case-sensitive. We want the suggestions to be based solely on the word typed, without considering whether it's in uppercase or lowercase.

Q) How could we solve this problem ?

To fix the problem, we just need to use the code provided. It uses '`toLowerCase()`' to make our search bar insensitive to capitalization.

```
<button
  className="searchBtn"
  onClick={() => {
    // filter the Restaurant and update the UI
    const filtertheRestaurant = listOfRestaurant.filter((res) => {
      return res.info.name.toLowerCase().includes(searchText.toLowerCase());
    });

    setListOfRestaurant(filtertheRestaurant);
  }}
>
  Search
</button>;
```

Step 4 —>

We've encountered another issue in our app: after searching for a restaurant, the UI doesn't render anything when we search again. Instead, we only see the Shimmer UI.

How could we solve this problem ?

here problem is when we search 1st time we are updating '`listOfRestaurants`'. If we try to search it again it is searching from previous updated list thats the problem. simple solution for this instead of filtering the original data we simple make a copy to of that original data in our case it is nothing but a '`listOfRestaurant`' and stored the copy with new variable `filteredRestaurant`.

```
//original const [listOfRestaurant, setListOfRestaurant] =
useState([]);

//copy
const [filteredRestaurant, setFilteredRestaurant] = useState([]);
```


In our code, when we fetch data using the `'fetchData'` function, it's important to update the rendering to display the new data. We achieve this by updating the state variables `'listOfRestaurant'` and `'filteredListOfRestaurant'` using functions provided by the `'useState()'` hook. Initially, both arrays are empty, but after fetching data, we fill them with the retrieved information. otherwise we won't see any thing on the page.

```
const fetchData = async () => {
  const data = await fetch(
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=19.9615398&lng=79.2961468&is-seo-homepage-enabled=true&page_type=DESKTOP_WEB_LISTING"
  );
  const json = await data.json();
  // here we are filling both the variable with new data with the help of their functions.
  setListOfRestaurant(
    json.data.cards[4].card.card.gridElements.infoWithStyle.restaurants
  );
  setFilteredRestaurant(
    json.data.cards[4].card.card.gridElements.infoWithStyle.restaurants
  );
};
```

IMPORTANT:

here there is two important points to remember

- 1) When we need to modify the list Of Restaurants based on certain conditions, we're essentially using the original data we fetched and stored within the `'listOfRestaurant'` variable. (original)
- 2) To display data on the UI, we use a copy of `'listOfRestaurant'` called `'filteredRestaurant'`. (copy)

Final code base given below.

```
import { useState, useEffect } from "react";
import RestaurantCard from "../RestaurantCard";
import Shimmer from "../Shimmer";

const Body = () => {
  const [listOfRestaurant, setListOfRestaurant] = useState([]);
  const [filteredRestaurant, setFilteredRestaurant] = useState([]);
```

```

const [searchText, setSearchText] = useState("");

useEffect(() => {
  fetchData();
}, []);

const fetchData = async () => {
const data = await fetch(
  "https://www.swiggy.com/dapi/restaurants/list/v5?lat=19.9615398&lng=79.2961468&is-seo-homepage-enabled=true&page_type=DESKTOP_WEB_LISTING"
);
const json = await data.json();

  setListOfRestaurant(
    json?.data?.cards[4]?.card?.card?.gridElements?.infoWithStyle?.restaurants
  );
  setFilteredRestaurant(
    json?.data?.cards[4]?.card?.card?.gridElements?.infoWithStyle?.restaurants
  );
};

return listOfRestaurant.length === 0 ? (
  <Shimmer
) : (
  <div className="container body">
    <div className="filter-btn">
      <div className="search">
        <input type="text"
          className="search-box"
          value={searchText}
          onChange={(e) => {
            setSearchText(e.target.value);
          }}
        />
        <button
          className="searchBtn"
          onClick={() => {
            // filter the Restaurant and update the UI.
            const filteredRestaurant = listOfRestaurant.filter((res) => {
              return res.info.name

```

```

        .toLowerCase()
        .includes(searchText.toLowerCase());
    });

    setFilteredRestaurant(filteredRestaurant);
  }}
  >
    Search
  </button>
</div>

<button
  onClick={() => {
    const filterLogic = listOfRestaurant.filter((res) => {
      return res.info.avgRating > 4;
    });
    setFilteredRestaurant(filterLogic);
  }}
  >
    Top Restaurants
  </button>
</div>
<div className="RestaurantContainer">
  {filteredRestaurant.map((restaurant) => (
    <RestaurantCard key={restaurant.info.id} resData={restaurant} />
  ))}
</div>
</div>
);
};

export default Body;

```

THANK YOU!

Self-Notes

Monolith- when the apps develops traditionally all were using the monolith architecture, what does it means- Earlier we have a huge big project and the project has code where APIs has written, we have UI in the same projects, we have authentication inside the same project, we have database connectivity in the same projects, we have sms inside the same project, so basically we have do everything inside the same project/in the same service.

Disadvantage - Even we have do the single change, suppose I have change the color of the button, what we used to do we have to build this whole project, we have to compile the whole project, we have deploy the whole big bulky project.

Now the word moving to the microservices architecture.

Microservices- In microservices we have different services for different job, like we have a UI service, we have authentication services, service which connect to db, we have a service for SMS sending, we have a service for email notification, so there are different services, and all the services combine together and form a big app.

All the services talking to each other depending upon the use cases and for each and every thing we have a different project like we have a separate UI project, we have separate db project and so on and this is known **as separation of concerns and single responsibility principle.** Where each and every services have there own job.

UI developer, db team and everyone working in the same project and same git repo. and with the microservices architecture all the team works on independent services.

How the services are connected - All the services run there on own specific port. Suppose port :1234- we have UI service, port:1000- we have backend port and on :3000 – sms service, on different port deploy different services and at the end all the ports can map domain name. Suppose backend map to /api and all the api is map to the same url.

How the services interact, they call to different urls. Suppose UI connect to backend they may make the call to /api or we can say that it call to different port. That's how the services are connected and that's how they interacted to each other.

How do this react application talk to different microservices outside of this world?

How the react application make the backend api call and fetch the data?

How we can fetch dynamically data from the Api and populate are page dynamically?

There are two approaches how web app or ui application fetch the data from backend –

1. Load and Render-

Loads -> **API** -> **Render**

As soon as our page loads, we can make Api call and wait for data to come and render to it to the UI.

If the API call takes 500ms so what will happen our page will load and page will wait 500ms and then after 500ms it will render the UI. You must have seen we you open the web page you suddenly don't seen anything and suddenly as soon as api response it quickly shows lots of stuff on the screen.

2. Render First Fetch Later-

Loads -> **Render** -> **API** -> **Re-render**

As soon as the page loads, just quickly render the UI (render the skeleton) and as soon as skeleton will render, now will make the Api call and as soon as we get result back to API, will now again re-render the application with the new data.

In react always we use second approach, and this will give us the better UX.

Why are we using second approach and why it is better?

With the first approach till 500ms our page are kind of frozen, we don't not seen anything on to the page and after 500ms will suddenly see everything. So that's the poor user experience.

In the second approach we load the page and render the skeleton , so we can see something on the website and slowly website will load. It is the better user experience. Use don't see lot of lack all the random stuff, the zitriiness when we load the page. This experience much better.

React render cycle is very fast, it has the best render mechanism. So it render the UI very fast. So we are not bother how much time we are rendering, but two render here are ok.

Hooks-

UseEffect()-

First import it from react

```
import { useEffect } from 'react';
```

useEffect() takes two arguments-

1. Callback function
2. Dependency Array

Syntax- `useEffect(()=>{},[])`

When will this useEffect callback function be call-

useEffect callback function will call after the components will renders

When we return this useEffect inside the body component, so when the body component will load it will render the component and as soon as the render cycle finished it will quickly call the callback function which will pass to the useEffect,

Now how the code will work , the body component will render and then this callback function will be call and useEffect printed to the console.

If we do something after the rendering, we have write it down inside the useEffect. As soon as the body loaded useEffect will load.

```
useEffect(()=>{  
  console.log("useEffect called")  
},[])  
  
console.log("body rendered")
```

body rendered

[Body.js:13](#)

useEffect called

[Body.js:10](#)

How the code is executed , as soon as the body component will rendered, It renders line by line. It will execute all the code as soon as see `useEffect`, it will keeps the callback function/ save the callback function to call it after render and it will moves a head, then it will call `console.log("Body rendered")` and it will render JSX and once the render is done then it will call `useEffect` callback function and it will print `console.log("useEffect called")`. This is how `useEffect` will works. `useEffect` will be helpful when we have to implement the **Render First Fetch Later** approach.

Fetch the data-

So we are fetch the data inside the `useEffect`. So to fetch the data we are creating the function and write it down the function logic to fetch the data.

How do we fetch the data in javascript - > same we can fetch in react.

So we use the method `fetch` , `fetch` is the super power that is give to us by browsers/ which has JS engine and this `fetch` method will fetches the data from API.

What will `fetch` return -> `fetch` will return the promise, so we can handle the promise either we can use `then` and `catch` or we can use `async` and `await`.

Use `async` `await` is the standard practice in the industry and have the much better syntax. So, make the function `async` and wait data to come. such way we can resolve the promise. Once we have data , we have to convert the data into json.

```
const Body = () => {
  const [listOfRestaurants, setListOfRestraunt] = useState([]);

  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async () => {
    const data = await fetch(
      "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.62448069999999&page_type=DESKTOP_WEB_LISTING"
    );

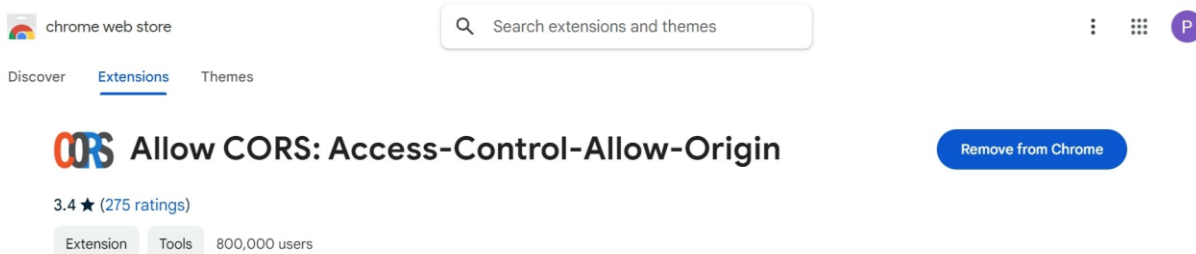
    const json = await data.json();
    console.log(json)

    // Optional Chaining
    setListOfRestraunt(json?.data?.cards[1]?.card?.card.gridElements?.infoWithStyle?.restaurants);
  };
};
```


Error- access to fetch swiggy.com from local host has been blocked by cors policy. What is CORS policy- are browser that is chrome not allowing us to call swiggy API in the local host, from one origin to another origin. If the origin mismatches the browser blocks the API call. That is the cross policy.

How do I bypass the cors?

We can install CORS chrome extension in our browser, it allow to bypass the CORS. There is option turn on and off and every browser have there diff. extension, we can install it.



Got the data –

```
Body.js:19
{statusCode: 0, data: {...}, tid: '2097ded1-2ef8-45ff-a80d-
d5edb292922f', sid: 'gpwf25a2-4923-4d86-89c4-d9003636487
9', deviceId: '723e2588-fc3c-a1e3-3538-eb988682d798', ...}
  i
  csrfToken: "eD6rxhgtxsv-mJZUVt1Q4-e1jXY1Cb0tdocn0Gc"
  data: {statusMessage: 'done successfully', pageOffset: {
    deviceId: "723e2588-fc3c-a1e3-3538-eb988682d798"
    sid: "gpwf25a2-4923-4d86-89c4-d90036364879"
    statusCode: 0
    tid: "2097ded1-2ef8-45ff-a80d-d5edb292922f"
  }}
  [[Prototype]]: Object
```

Now we want the json swiggy data populate in our page-

- **We need to rendering the cards-** so we already rendering the cards using map. We did `listOfResturant.map` and this `listOfResturant` contains `reslist(mockdata)`.
- Now after received the data from json , we want to update the `listOfResturant` with new data and what will happen is react will re-render the component and the new data will populate on the page. So, how do I update the `listOfResturant`, what we will do in `setListOfResturant` and in this update the new data comes up.

```
setListOfRestraunt(json?.data?.cards[1]?.card?.card.gridElements?.infoWithStyle?.restaurants)
```

After that we will get the new data and `listOfResturant` should be empty-

```
const [listOfRestaurants, setListOfRestraunt] = useState([]);
```

Optional Chaining –

Optional chaining (?.) in JavaScript and React is a concise way to safely access deeply nested properties in objects, without having to check each level for null or undefined. If any part of the chain is null or undefined, it returns undefined instead of throwing an error.

```
const user = {
  name: 'John',
  address: {
    city: 'New York',
  },
};

// Without optional chaining
const city = user && user.address && user.address.city; // 'New York'

// With optional chaining
const city = user?.address?.city; // 'New York'
```

How it works in React:

1. **Props Access:** When passing props to a component, some properties might not always be available. Optional chaining ensures that you can access deeply nested props without worrying about runtime errors if a certain prop doesn't exist.

```
function UserProfile({ user }) {
  return (
    <div>
      <p>Name: {user?.name ?? 'Guest'}</p>
      <p>Email: {user?.email ?? 'No Email Provided'}</p>
    </div>
  );
}
```

In this example, if user or user.name/user.email is undefined, it won't throw an error. Instead, it safely returns a fallback like 'Guest' or 'No Email Provided'.

2. **State Access:** When accessing state in a React component, especially when it's asynchronous or fetched from an API, certain properties might be missing or undefined

```
const [user, setUser] = React.useState(null);

React.useEffect(() => {
  // Simulating fetching user data
  setTimeout(() => {
    setUser({ name: 'John', address: { city: 'New York' } });
  }, 1000);
}, []);

return (
  <div>
    <p>Name: {user?.name ?? 'Loading...'}</p>
    <p>City: {user?.address?.city ?? 'Unknown'}</p>
  </div>
);
```

Here, `user?.name` and `user?.address?.city` will safely handle the case where `user` or its properties aren't immediately available due to the asynchronous nature of fetching data.

3. **API Responses:** Optional chaining is useful when handling responses from APIs, especially when the structure is inconsistent or incomplete.

```
function Post({ postData }) {
  return (
    <div>
      <h1>{postData?.title ?? 'Untitled'}</h1>
      <p>{postData?.body ?? 'No content available.'}</p>
      <p>Author: {postData?.author?.name ?? 'Anonymous'}</p>
    </div>
  );
}
```

This ensures that even if `postData`, `postData.title`, `postData.body`, or `postData.author` is missing, the app doesn't break. Instead, fallback values are provided.

After fetching the data, there's a noticeable one-second delay before it appears on the screen. This delay occurs because the APIs take some time to load. Improving this can enhance the user experience.

Either we can do we can shoe the spinning loader or use the simmer effect-

1. Spinning loader -> we will write the condition

```
if (listOfRestaurants.length === 0) {  
  return <h1>Loading.....</h1>  
}
```

Note: But its not the recommended way

2.Shimmer UI – It is like we load fake page until the actual data from API, So instead of showing loading (spinning loader) we can show the skeleton, we can show fake cards over here till the data is loaded.

Conditional rendering – rendering based on condition is known as conditional rendering.
Example – simmer effect

```
return listOfRestaurants.length === 0 ? <Shimmer /> : (  
  <div className="body">...  
  </div>  
)  
}  
  
export default Body;
```

Dive deep in state variable-

When we have normal variable why do we use state variable/ Why and when do we need state variable?

Build the feature login/ logout for understating -

If we want to make the component dynamic (here we will make the login button dynamic).

1. We will create the local variable and use inside the JSX, initially button is login

```
const btnNameReact = "Login"
<button className='Login' >{btnNameReact}</button>
```

2. On click of login button, will change in logout button and it should work.

```
<button className='Login' onClick={()=>{ btnNameReact = "Logout"
console.log(btnNameReact)}} >{btnNameReact}</button>
```

When we see in console , this variable name btnNameReact will update and return Logout , but the UI does not update that means UI didn't render/ didn't not refresh, there is some way refresh the header component so will get the latest value of btnNameReact on UI, In such cases normal JS variable not working, If you want to make component dynamic, if you want something should changed in your component we use local state variable, here is where we use useState variable.

Create react variable –

```
const [btnNameReact] = useState("Login")
// This behave like the normal JS variable like - const btnNameReact = "Login"
```

Now on click on the button I want to update this new updated variable, but we can't directly modify like above example , we will have to use second variable (That will become the function) and if we want to update the variable we can do it with the second variable.

```
const [btnNameReact, setBtnNameReact] = useState("Login")

<button className='Login' onClick={() => {setBtnNameReact("Logout")
console.log(btnNameReact )}}>{btnNameReact}</button>
```

When the state variable will be changed, react will re-render the component and refresh the component and trigger the re-consolation cycle.

Que- React will refresh the whole component i.e. Header or it will just refresh the button?

Ans- It will re-render the whole header component that means whole component refresh it.

First time is the initial render i.e. header render and as soon as I click on the login button header will render once again, that means whole component refresh quickly.

as soon as I click on this button and as soon as onclick will call and this callback will call and setBtnNameReact will call, react will keep in the track btnNameReact and this btnNameReact react will update this variable btnNameReact and then it will call the header component, it will render the header component once again.

What do you mean by rendering the component – means it will just call the header function and will call the header component once again. It will trigger it once again. But now this time we have updated values inside it.

Thousands of developers get confused that if is `this const [btnNameReact, setBtnNameReact] = useState("Login")` is the const variable, how is getting updated, and the const variable updated the new value, Is it defending the JS principles. If the `btnNameReact` is the const variable how can we modify the const variable Why and How this `setBtnNameReact` how it is able to change `btnNameReact` value to logout, how ?

Whenever we update `setBtnNameReact` value, basically react is updating this `btnNameReact` and it will calling this header function once again , It is rendering , Calling the function equivalent to the rendering function once again but this time you invoke this function and this `btnNameReact` is the new variable then it is before. Initially time state variable is login and as soon as I do `setBtnNameReact` it will call the header function once again and it will create the new instance, this `btnNameReact` is different than the older `btnNameReact` and when the new `btnNameReact` is created it will

created with the default value, it will created with the updated value with the `setBtnNameReact`

In short –

```
const [btnNameReact, setBtnNameReact] = useState("Login")

<button className='Login' onClick={()
=>{setBtnNameReact("Logout")}}>{btnNameReact}</button>
```

As soon as call `setBtnNameReact` it will update the `btnNameReact` value reference and then it will render the header component once again, it will find the diff between the older version and new version and it will see in that div, only this button getting updated, there is no change in ul li , only this button has changed , this is happening with the diff algo, this is the re-consolation process. Here only the button value is changed and it change very fast, it is all dynamic that is why we need state variable.

Toggle functionality-

```
<button className='Login' onClick={() => {
  btnNameReact == "Login" ? setBtnNameReact("Logout") : setBtnNameReact
  ("Login")
}}>{btnNameReact}</button>
```

Every time click on button reconciliation process is trigger, react find out the difference between the older version and the new version of the header and it find out the only the button change, in this html only update the button. This is why react is fast. React is best in the dom manipulation because it exactly knows what to change. So it only changes the button. This is how whole render cycle is work. Code is render again and again on click of button.

Implement the search functionality-

If I onclick on the search button we want to implement the Filter the restaurant cards and updated the UI, where will get the search text from so I will need the search text, whatever in the search box to get the data from input box I need to take this value of this inputbox and we have to bind this inputbox to the local state variable.

To track the value of the inputbox whatever users is typing in, to get that value in I have to bind this value with local state variable of react. So will create one more state variable and bind that variable to my inputbox. Initially I will give searchText empty and also have setSearchText to updated ad then bind it to the inputbox.

```
const [searchText, setSearchText] = useState("");
```

```
<input type="text" className='search-box' value={searchText} onChange={(e) => {  
  setSearchText(e.target.value) }} />
```

Issue- I am typing in the seachbox and nothings is happening over here. Why so?

Because we bind value inside the input by searchText variable vice versa , That means whatever is there in the searchText variable will be the inside the value={searchText} of input box and now when I change the value of inputbox still is tight to the searchText so my searchText not getting updated but I am trying to update by input box, so it can't happen because value is bind to the seachText and searchText is empty . so that the input box does not change unless we change the searchText. So to fix this we will write the onChange Handler there. So as soon as my input changes what will happen my onChange function also change/update my searchText with the new value. How will the new value, I will get it from the callback method and we will do setSearchText over here and we will give e.target.value

```
return listOfRestaurants.length === 0 ? <Shimmer /> : (  
  <div className="body">  
    <div className="filter">  
      <input type="text" className='search-box' value={searchText} onChange={(e) => { setSearchText(e.target.value) }} />  
      <button onClick={() => {
```


Whenever I am typing something as soon as I type the letters what will happen behind the scene, we are changing local state variable and what happens when we are changing the local state variable, **react will re-rendering and refresh the component**

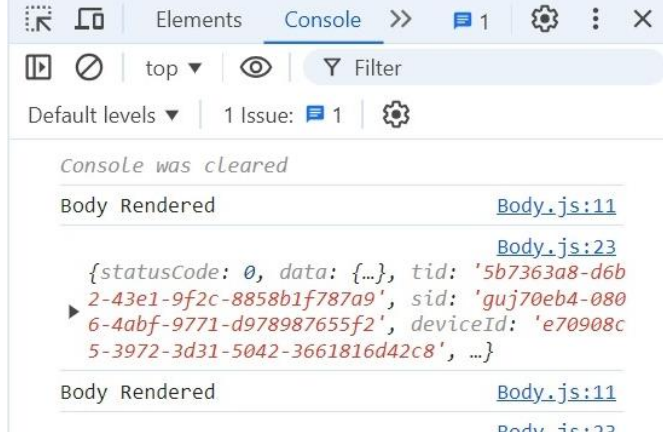
Whenever the search text getting updated every keypress, component getting re-render. The whole-body component getting re-render, react is efficiently rendering it.

```
const Body = () => {
  const [listOfRestaurants, setListOfRestraunt] = useState([]);
  const [searchText, setSearchText] = useState("");

  console.log("Body Rendered")

  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async () => { ...
};
```



Console was cleared

Body Rendered [Body.js:11](#)

[Body.js:23](#)

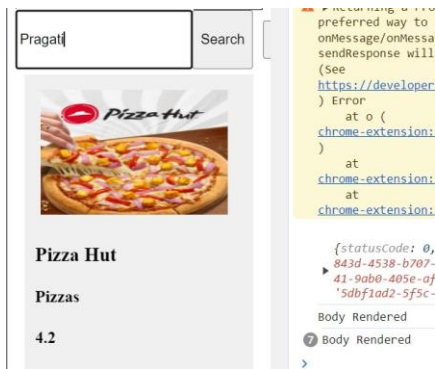
```
{statusCode: 0, data: {...}, tid: '5b7363a8-d6b2-43e1-9f2c-8858b1f787a9', sid: 'guj70eb4-0806-4abf-9771-d978987655f2', deviceId: 'e70908c5-3972-3d31-5042-3661816d42c8', ...}
```

Body Rendered [Body.js:11](#)

Initially the body got rendered, why it is render once again because it follows-

Loads -> **Render** -> **API** -> **Re-render**

And whenever state variable update, react triggers a reconciliation cycle(re-renders the component)



Here in every letter component will re-render

Even that we are not realizing body is re-render 7 times. React does the render process very fast, efficient rendering, efficient DOM manipulation.

How it is working - whenever you are typing anything (P, R, A, G, A, T, I), its calling onchange method and onchange method taking this target and it is just updating the state variable(searchText) and body component just re-render. But the best part is react is triggering the re-consolation cycle and finding the difference between the old virtual DOM and new virtual DOM and see the input value has changed so it will update the input box. React is checking the difference every time when I hitting the keypress and react is find the diff between the two virtual doms and it will only update the inputbox.

React is re-rendering the whole-body component but it is only updating the input box value inside the DOM. DOM manipulation is very expensive and react is very efficient of doing it.

Implement the filter logic

```
<button onClick={() => {
  //Filter the resturant carads and updated the UI
  //searchText
  console.log(searchText)

  const filterRestaurant = listOfRestaurants.filter((res) => res.info.name.includes(searchText))
  setListOfRestraunt(filterRestaurant)
}}>Search</button>
```

When I click on the search button it will filter out all the restaurant which includes pizza, coffee and just updating listOfResturant with the updated data and as soon as data updated react will re-render the body component and this time it will re-render with the filter data.

Bug for capital and small letters –

```
const filterRestaurant = listOfRestaurants.filter((res) => res.info.name.toLowerCase().includes(searchText.toLowerCase()))  
setListOfRestraunt(filterRestaurant)
```

Here problem is when we search 1st time we are updating 'listOfRestaurants'. If we try to search it again it is searching from previous updated list that's the problem and lost all the other data and update it with the new data that's the problem.

simple solution for this instead of filtering the original data we simple make a copy to of that original data in our case it is nothing but a 'listOfRestaurant ' and stored the copy with new variable filteredRestaurant.

The issue is come because we are setListOfRestraunt(filterRestaurant) instead what we should do is we will keep in the another copy of the filter list. Create the another variable of filter restaurant.

```
const [listOfRestaurants, setListOfRestraunt] = useState([]); //Original
```

```
const [filteredRestaurant, setFilteredRestaurant] = useState([]); //Copy
```

instead doing ->	setListOfRestraunt(filterRestaurant)
use this ->	setFilteredRestaurant(filterRestaurant)

```
{  
  instead doing ->           // listOfRestaurants.map((restaurant) =>  
(<RestaurantCard resData={restaurant} key={restaurant?.info.id} />))  
  
  use this ->  
    filteredRestaurant.map((restaurant) => (<RestaurantCard  
resData={restaurant} key={restaurant?.info.id} />))  
}
```

and when whenever we are doing filter instead doing setListOfRestraunt I will update in setListOfRestraunt and when we will rendering , will rendering in filteredRestaurant

Issue facing – initially card will not showing



To resolve the issue will update the data in `setFilteredRestaurant` (Here basically we are extracting the list of cards from the data)

```
// Optional Chaining
setListOfRestraunt(json?.data?.cards[1]?.card?.card.gridElements?.infoWithStyle?.restaurants)
setFilteredRestaurant(json?.data?.cards[1]?.card?.card.gridElements?.infoWithStyle?.restaurants)
};
```

So when the body component loaded for the first time it makes the API call and it will update the `setListOfRestraunt` and as well as the `setFilteredRestaurant`. Whenever I have to filter something will use `ListOfRestraunt` to filter out and I will update the `FilteredRestaurant` and to display to in the UI using 'FilteredRestaurant'

Whenever I search any word and I will click on the button, now what happened when I click on this `FilteredRestaurant` got updated and because whenever the state variable changes it will automatically refreshes. So filter restaurant getting displayed. But is the value inside `ListOfRestraunt`, so `ListOfRestraunt` will still have the data which was coming from the API, so next time I will filter I will filter from the `ListOfRestraunt`.

Episode-6.1 | Swiggy API Issue Resolved

API returning the data in different format, the structure of data has changed and the destructuring we are doing in the last episode has changed, this is how we are facing the issue.

Problem of working with the live api is keep on changing

How to find the API-

Inspect -> Network -> refresh the page -> Select Fetch/XHR (give all the API call) -> Copy link address **or** just double click so it will open on the new tab **or** in preview we can also see the data.

If we click on all it will give us result of all the URL that has been fetch for a page like photos, JS files, CSS files are present in this network

We can see the data in the systemic format by installing the JSON viewer extension / plugin on my system.

Extracting data from API-

https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.62448069999999&page_type=DESKTOP_WEB_LISTING

setListOfRestraunt(json?.data?.cards[1]?.card?.card.gridElements?.infoWithStyle?.restaurants)

we can also console it and see the data

console.log(json?.data?.cards[1]?.card?.card.gridElements?.infoWithStyle?.restaurants)

Episode-6.2 | CORS Plugin Issue solved

Without CORS Plugin how we can resolve the issue because CORS plugin we have put it up in your laptop but suppose somebody using the website they might not be CORS plugin.

There is one website **corsproxy.io** this basically use to bypass the cors error

What is cors- when you are trying to make a api call from one origin to different origin (one domain name to another domain name) or we can say that you are trying to api call from local host to swiggy.com api so browser does not allow. So, to the buypass this cors we have something know as **corsproxy.io**.

How to use it-

Use the Corsproxy.io URL (<https://corsproxy.io/>?) just before the API like this way-

https://corsproxy.io/?https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.62448069999999&page_type=DESKTOP_WEB_LISTING

So what it will do you are not directly going to swiggy website, you are going to cors proxy.io(acting as the proxy) and internally it is making call to swiggy and getting to the data and returning back to you. So, Corsproxy handling CORS issue. It will work even the CORS plgin is not activated.

But this Corsproxy has a limit, after 50 60 request per minute. I think 40 apis calls per minutes. So, if we trying to call more than 40 api calls. so it may give you the eroor.