

Today lets start with higher order component.

Whats a higher order component??

It's a function that takes in a component and returns a component.

Lets assume we have video card ,

We can make a function that takes in a video card and returns a component with some additional functionality.

Suppose I want to do some modification on otop thee already present component we can use this HOC

GitHub Copilot: Sure, here's an example of how you can create a higher-order component that adds a `clickCount` prop to a component and renders a `div` element with an `h1` element and a `button` element:

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

const withClickCount = (WrappedComponent) => {
 const WithClickCount = (props) => {
 const [clickCount, setClickCount] = useState(0);

 const handleClick = () => {
 setClickCount(clickCount + 1);
 };

 return (
 <div>
 <h1>Click Count: {clickCount}</h1>
 <button onClick={handleClick}>Click Me</button>
 <WrappedComponent clickCount={clickCount} {...props} />
 </div>
);
 };
 return WithClickCount;
};
```

```

);
 };

 return WithClickCount;
};

export default withClickCount;
...

```

In this example, we define a higher-order component called `withClickCount` that takes a `WrappedComponent` as an argument. The `withClickCount` component returns a new component called `WithClickCount` that adds a `clickCount` prop to the `WrappedComponent` and renders a `div` element with an `h1` element and a `button` element.

To use the `withClickCount` higher-order component, you can wrap any component with it like this:

```

```javascript
import React from "react";
import withClickCount from "../withClickCount";

const MyComponent = ({ clickCount }) => {
  return (
    <div>
      <h2>My Component</h2>
      <p>Click Count: {clickCount}</p>
    </div>
  );
};

export default withClickCount(MyComponent);
...

```

In this example, we define a `MyComponent` component and wrap it with the `withClickCount` higher-order component. The `MyComponent` component receives the `clickCount` prop from the `withClickCount` component and renders a `div` element with an `h2` element and a `p` element that displays the `clickCount` prop.

Today we will be building search bar.(suggestions , debouncing)

For each word we are typing it is making an api call.

So what is happening is at first letter it is making a call , if I write a word immediately , very fast it will not make call.

So whats happening is its making api calls on some unit time

When u write slow – 4 letters – 4 api calls.

Very fast 7 letters – 4 api calls

Why is this happening??

U can go to network tab , for utube filter search

For flipcart – filter – auto

In payload you can see at what exact data its trying auto suggest on.

This is called Debouncing.

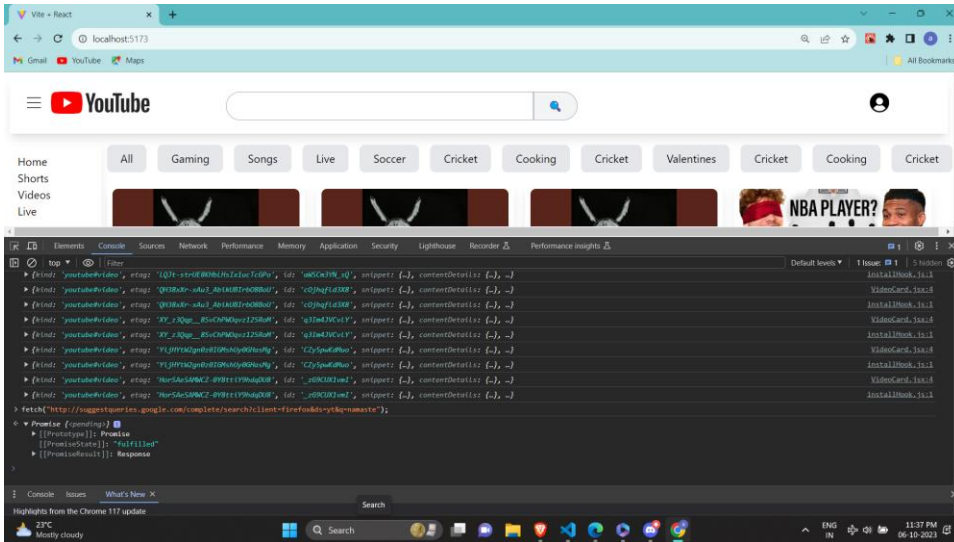
When I say debouncing of 2 milli sec –

If difference between 2 key strokes – do not make api call.

If its greater than 2ms – make api call

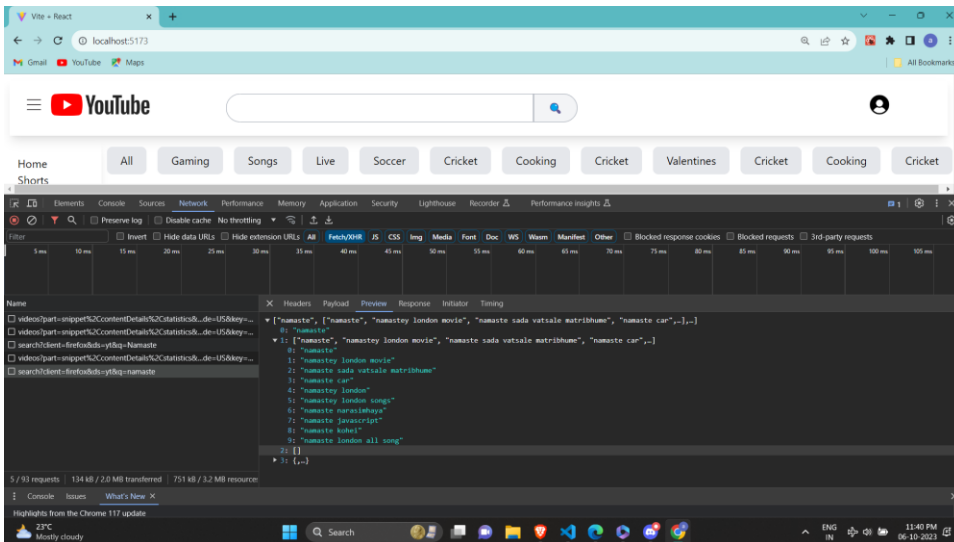
So lets start this by doing google search api.

So we will get whatever we will write in search bar – useRef – to get the value or u can also use handle event.
Once that's done we will make api calls for each words with some delay.



Made an api call in console.log.

Go to network , go for search api and click on it – go to preview you will get the search result data.



-----constants-----

```
const GOOGLE_API_KEY = "AIzaSyA0diAmgUiVwmrm_-Xrm93PJTVC12i1Tk";
```

```
export const YOUTUBE_VIDEOS_API =
```

```
"https://youtube.googleapis.com/youtube/v3/videos?part=snippet%2CcontentDetails%2Cstatistics&chart=mostPopular&maxResults=50&regionCode=US&key=" +
```

```
GOOGLE_API_KEY;
```

```
export const YOUTUBE_SEARCH_API =  
http://suggestqueries.google.com/complete/search?client=firefox&ds=yt&q=;
```

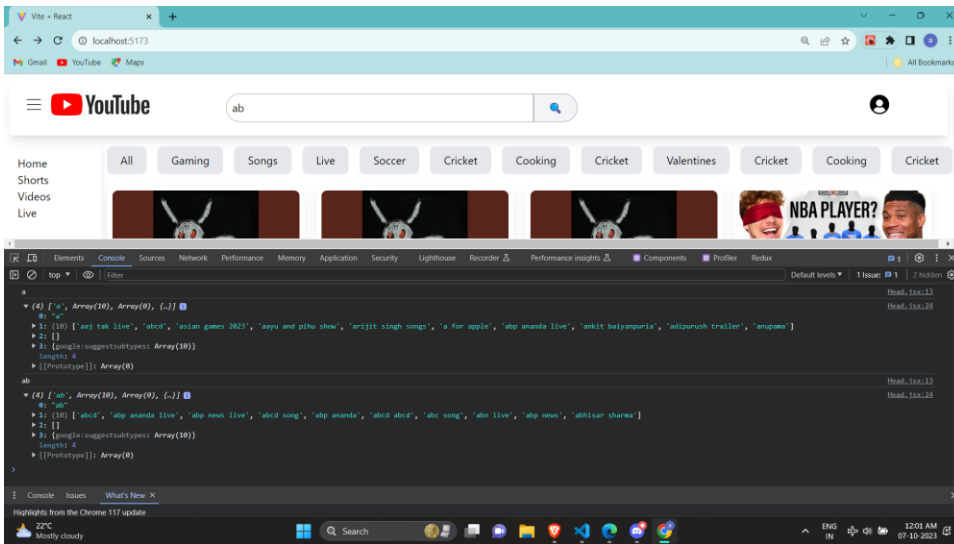
Now lets try to write debouncing.

```
<input  
  className="w-1/2 border border-gray-400 p-2 rounded-l-full"  
  type="text"  
  placeholder="Search"  
  value={searchQuery}  
  onChange={(e) => setSearchQuery(e.target.value)}  
/>
```

This is my search box - I have taken its value and updating it to my state variable.

```
useEffect(() => {  
  // api call  
  console.log(searchQuery);  
  // make api call ovwer each press , but if diff is less than <200 ms decline the cl  
  GetSearchSuggesion()  
}, [searchQuery]);
```

```
const getSearchSuggestions = async (searchQuery) => {  
  const data = await fetch(  
    YOUTUBE_SEARCH_API + searchQuery  
  );  
  const json = await data.json();  
  console.log(json);  
};
```



Now if I just go into search bar and try to search some api I can see so many api calls been made.

Now I want to make this api call only after 200 ms – debouncing

```
setTimeout(() => {
  getSearchSuggestions();
}, 200);
```

Key press 1 :

- Render comp
- useEffect()
- Start timer => make api call after 200 ms.

Key – 2

- Render the component
- useEffect()
- Start timer – is it the same timer or a new one??
- **It's a new timer**

But we need to clear the setTimeout also – as soon as my component rerenders I need to reset the time .

In class based component -

Unmounting was thought – return() at end of use effect. Rewatch if u don't understand.

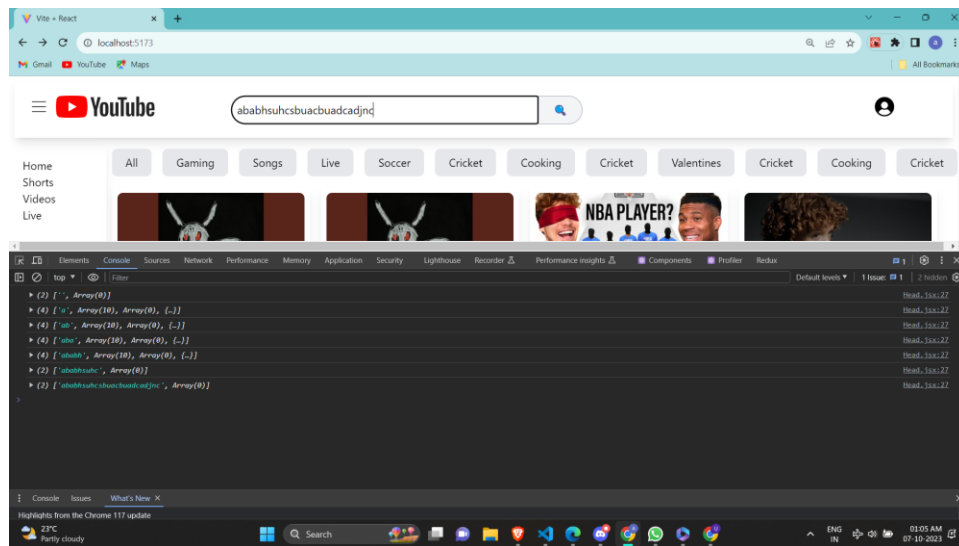
Key – 2

Destroy the component once you press a new key , old timer needs to be destroyed.(use Effect Return method)

- Render the component
- UseEffect()
- Start timer – is it the same timer or a new one??
- **It's a new timer**

If no key pressed till 200ms it will make api call

Look at the screen , now its making api call in 200ms only , we are not getting api call for each



So in youtube when you try to do some search it shows auto suggestion its shows , we will build that now.

Now we can optimise it even more by caching the fetched data.

When I search , it should store the suggestion data in store , and if the key is present in slice it should not make a new slice.