Debouncing

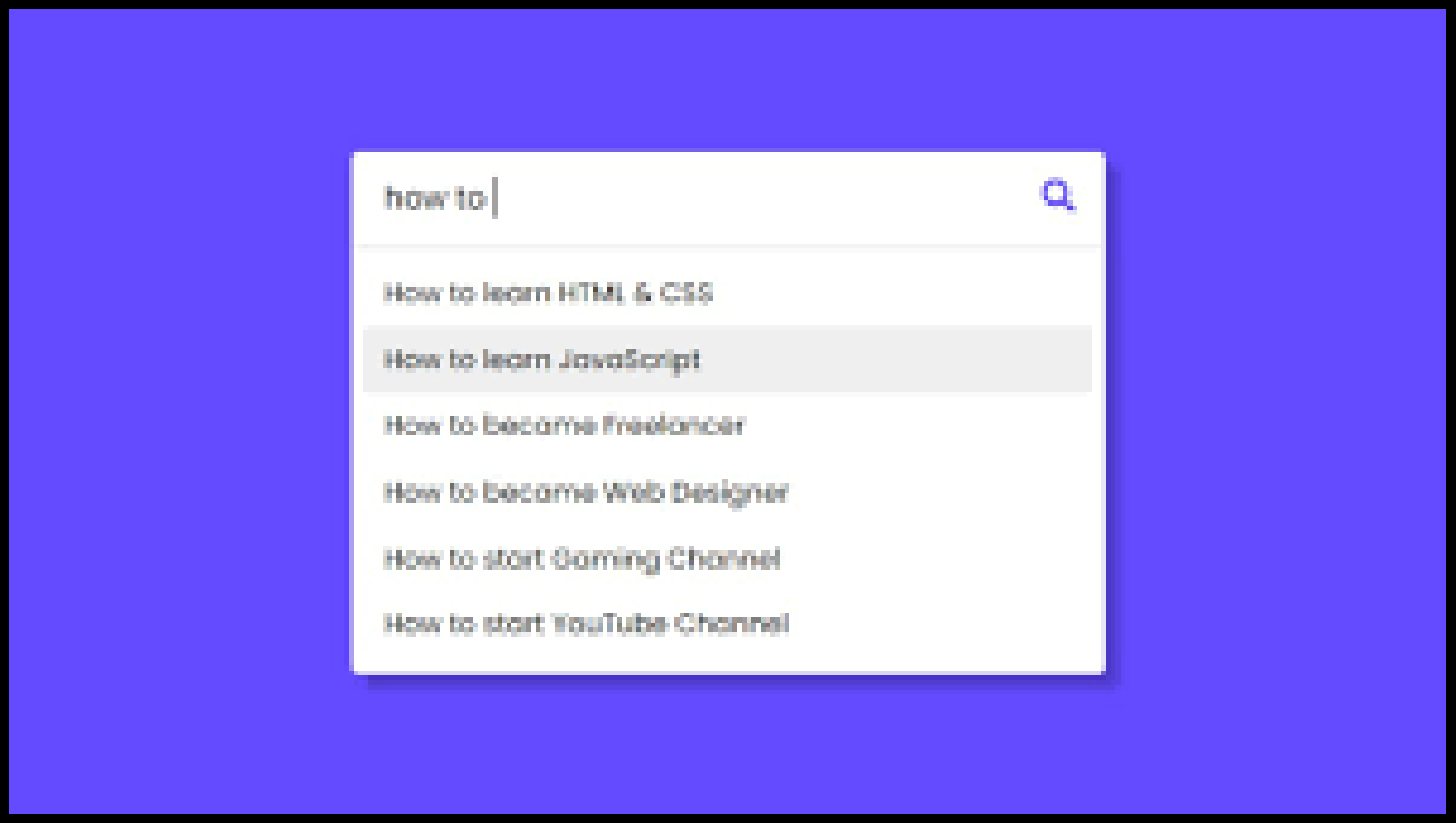
Give a space to learn

Human brain understand the concept only when it knows the problems solved by the concept

we will go to the concept of Debouncing after knowing the common problem solved by debouncing

- Search box suggestions,
- Text-field auto-saves, and
- double-button clicks.

Search box suggestions,



-> After each letter typed in search bar it will trigger the api function

->Here we triggered the api function 5 times by typing 5 letters "how to".

-> which will slow down the application as it is calling the function to many times

Debouncing solves the issue

Text field auto Save

- A dev created a web application which auto saves the form when user enters the details.
- Input Component will trigger an API call on every single character change.
- If the user types quickly, that means a lot of executions in a short amount of time:

which slow downs the application.

```
<input
  type="text"
  name="firstname"
  value={firstName}
  onChange={firstNameChange}
/>
```

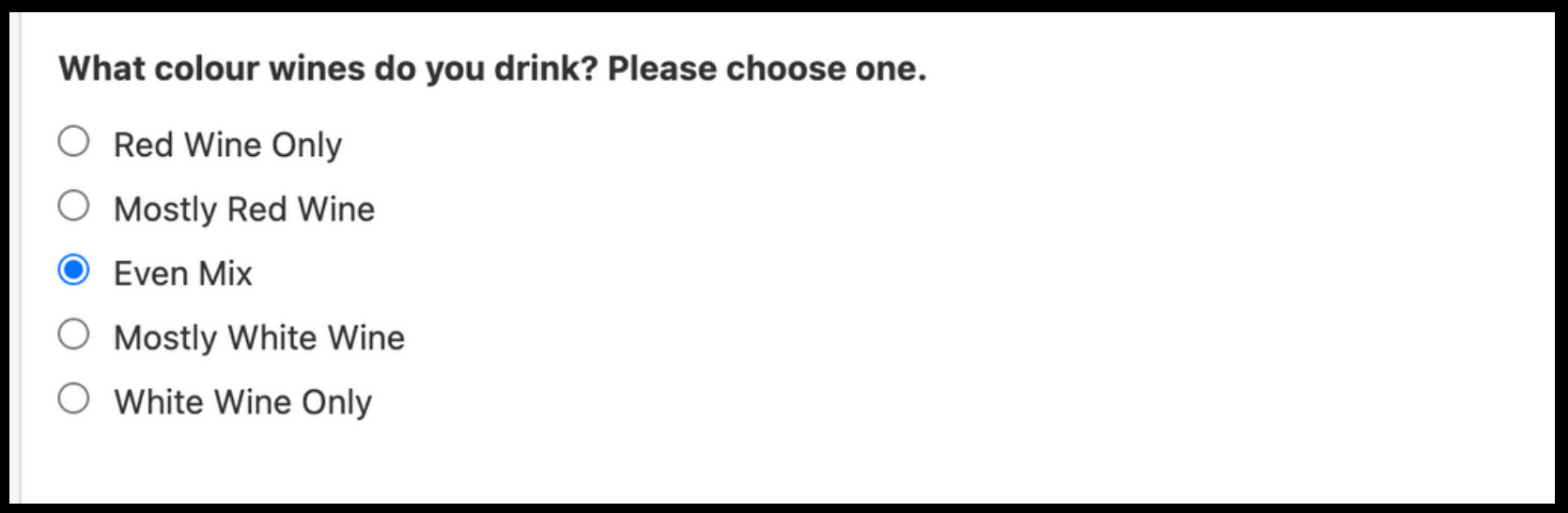
Debouncing solves the issue

Double-button clicks.

- A web application have radio buttons ,where user choose his preferred choice
- User clicks on the item(radio button) ,which triggers api update and as update is not happened immediately , user is clicking the radio button again as his previous update is not updated .

What happens is

- Here second api call fails or it will create a duplicate data in the database.



To avoid this we need Debouncing

Debouncing

Debouncing is a programming pattern or a technique to restrict the calling of a time-consuming function frequently,

- **By delaying the execution of the function until a specified time to avoid unnecessary CPU cycles, and API calls and improve performance.**

Simple definition

Debounce limits the times an event fires. No matter how many times the user fires the event, the function will be executed only after a specific time after the last fired.

Debouncing program example

```
function debounce(func, timeout = 500) {
    let timer;
    return (...args) => {
        clearTimeout(timer);
        timer = setTimeout(() => {
            func.apply(this, args); //our custom function
        }, timeout);
    };
}

function fetchResults() {
    console.log("Fetching input suggestions");
}
const processChange = debounce(() => fetchResults());
```

Explanation 👉

As we discussed debouncing limits the number of function calls.

- Here in the program we are using **setTimeout** in order to limit the function calls with in the duration specified, which is 500ms in our case.
- if the function is called 5 times with in 500ms, it will be triggered only once with the help of setTimeout.
- In this way we can have less function calls.
- **clearTimeout** is used to clear the previous timeout before executing the next timer .
- Inside the timer we can have our **own function call**, which will be triggered after a period of time