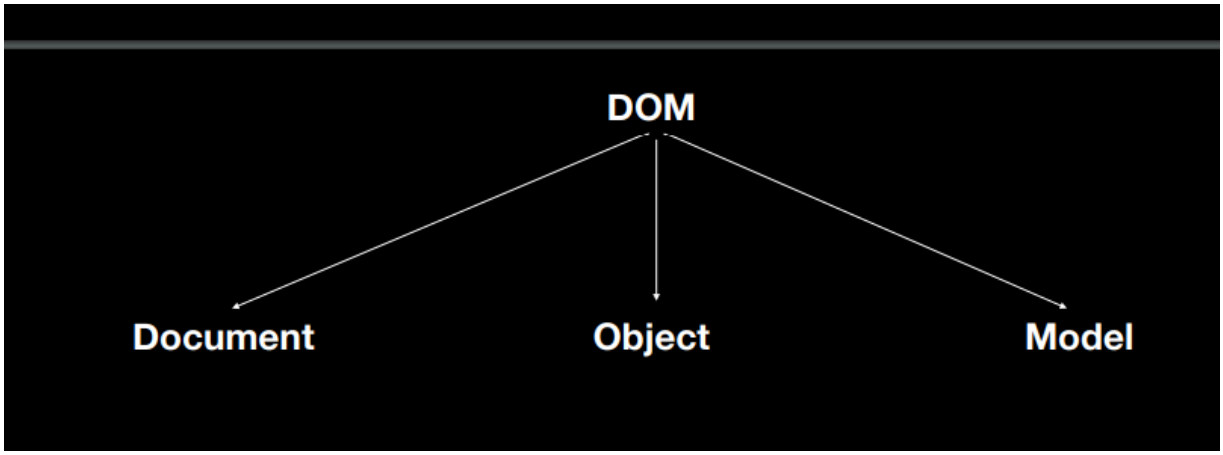
DOM

Week 4

4.1 - DOM, Dynamic frontends, Connecting FE to BE

4.2 - Chrome developer tools, Why frontend frameworks

The **DOM** (Document Object Model) API is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as a tree of objects; each object represents a part of the page.



How to create frontends without react?

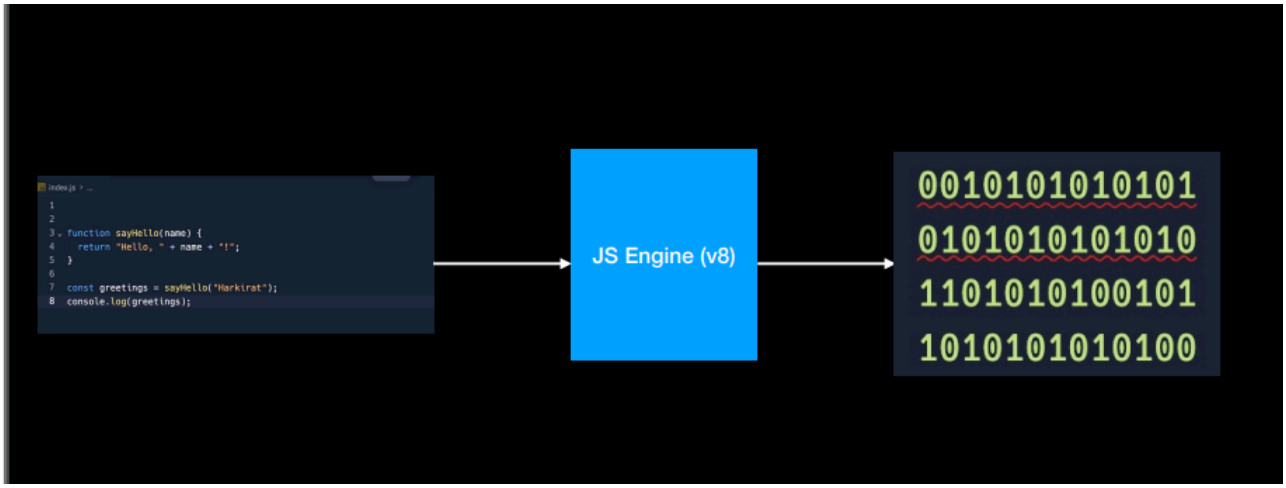
DOM is very important part of it.

What is Javascript??

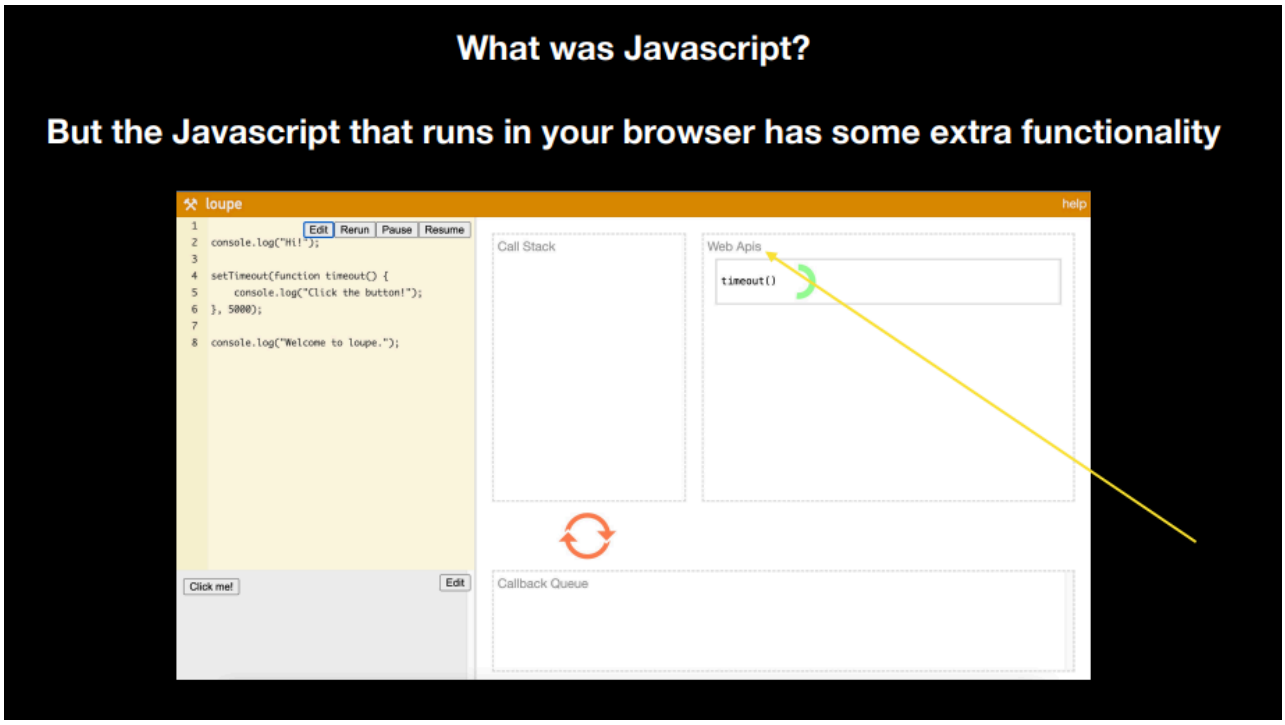
It is an implementation of the ECMAScript spec:

Google chrome has writtent he v8 engine , which is one of the compiler of the JS.

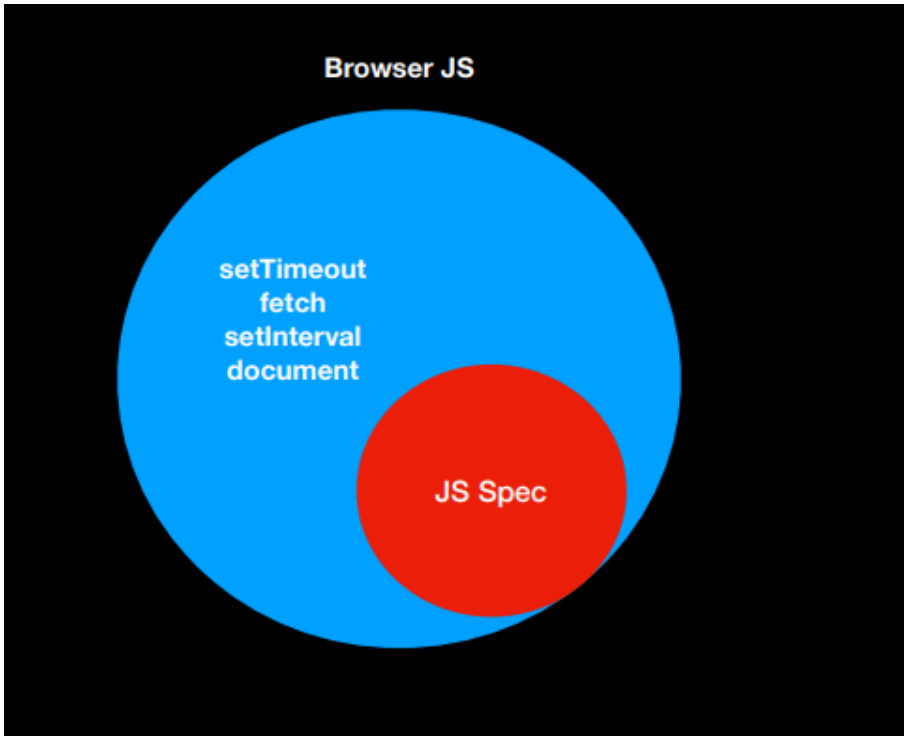
V8 engine get some Js code and convert it into 0 and 1’s.



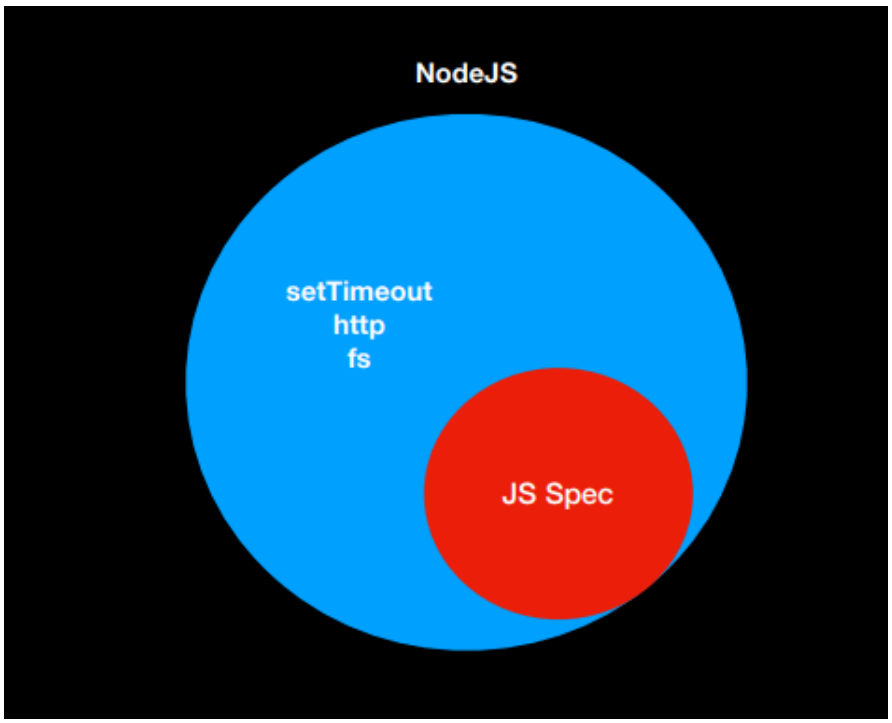
Browsers have some extra functionality



Example: setTimeout() (Web Apis) it was never part of ECMAScript



Even Nodejs has extra functionality

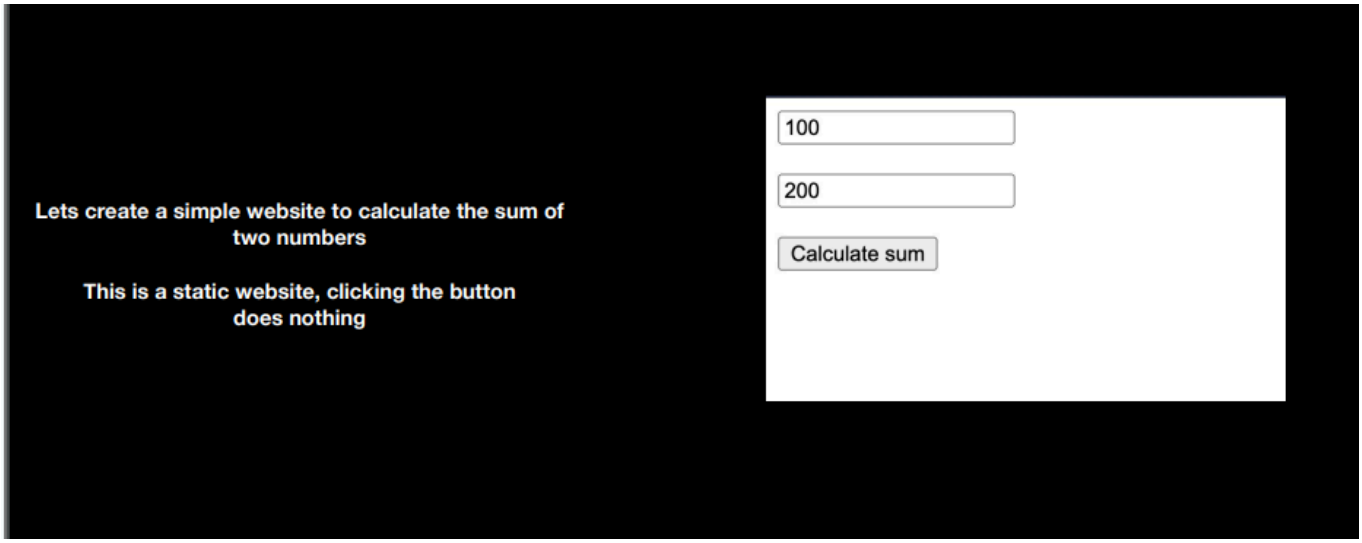


Http server on backend

A backend project need to access file system

These are the auxiliary libraries.

DOM / Document object



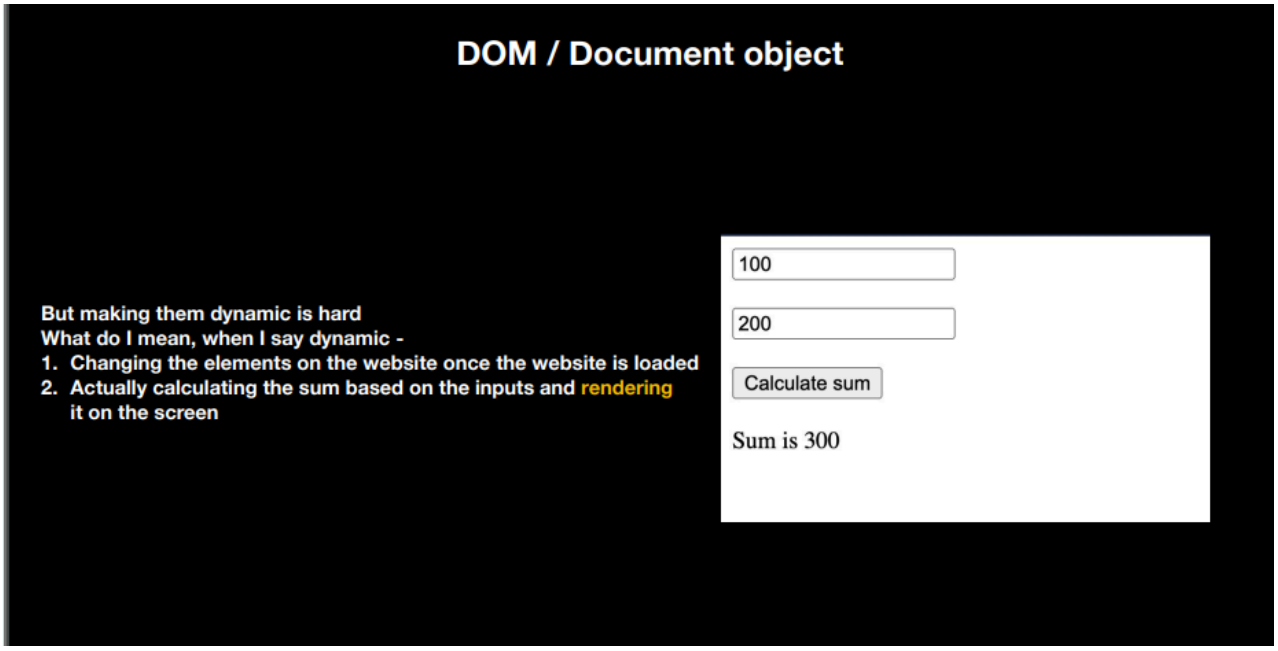

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
        <input type="text" placeholder="First number"><br></br>
        <input type="text" placeholder="Second number"><br></br>
    <button>Calculate sum</button><br>
</body>
</html>
```

But making this dynamic is hard (This is kind of why we need **React**)

:

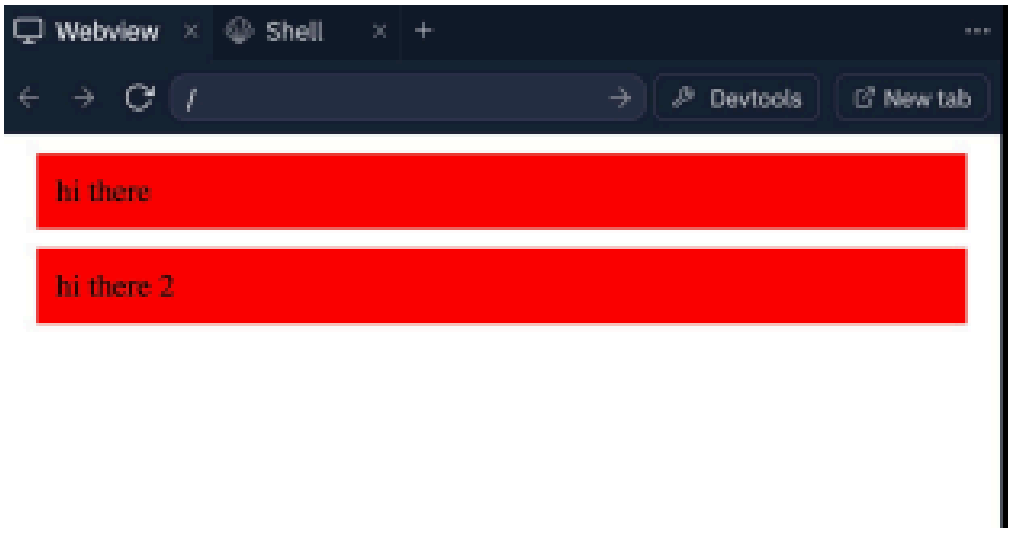
Changing the page once it is loaded makes a page dynamic

In our example putting a specific text once clicked on the button is making website dynamic or we can say adding a functionality to the website



Classes and Ids

Lets say we want similar div in two places.



Brute force way: (Violating the **DRY** principle)

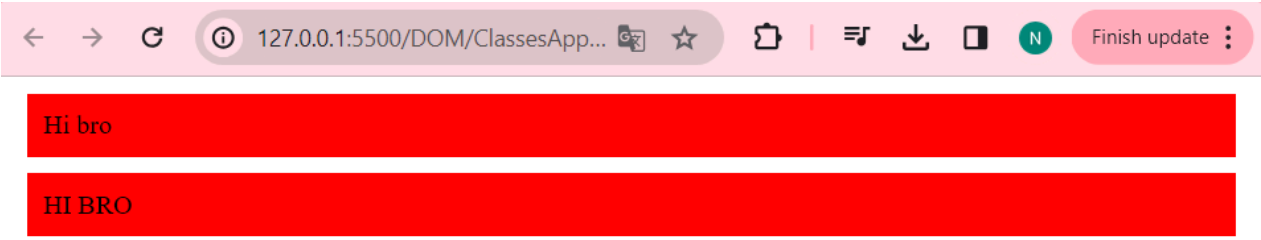
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
</head>
<body>
    <div style="background-color: red; margin: 10px;padding: 10px;">
        Hi there
    </div>
    <div style="background-color: red; margin: 10px; padding: 10px;">
        Hi there broo
    </div>
</body>
</html>
```

Classes

We just attach the class to div

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container{
            background-color: red;
            margin: 10px;
            padding: 10px;
        }
    </style>
</head>
<body>
    <div class="container">
        Hi bro
    </div>
    <div class="container">
        HI BRO
    </div>
</body>
</html>
```



Two container can have same id but it is **bad practice.**

- Why are Ids useful?
- Classes let you get rid of code repetition
- But what do ids do?

They let you access elements via the DOM API

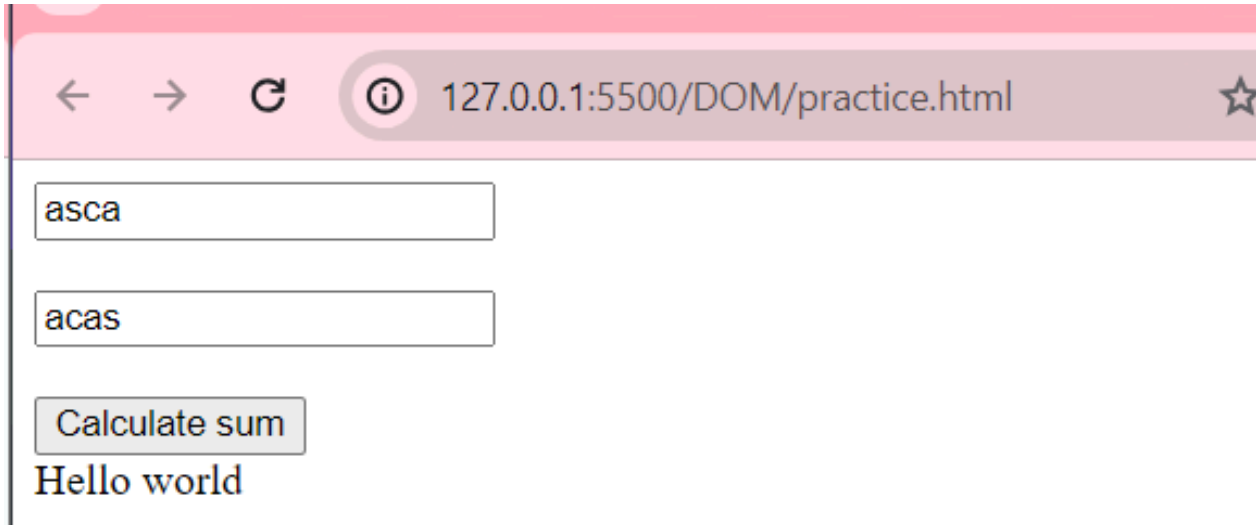
Can you use ids to do CSS? Yes

But we usually use it for JS

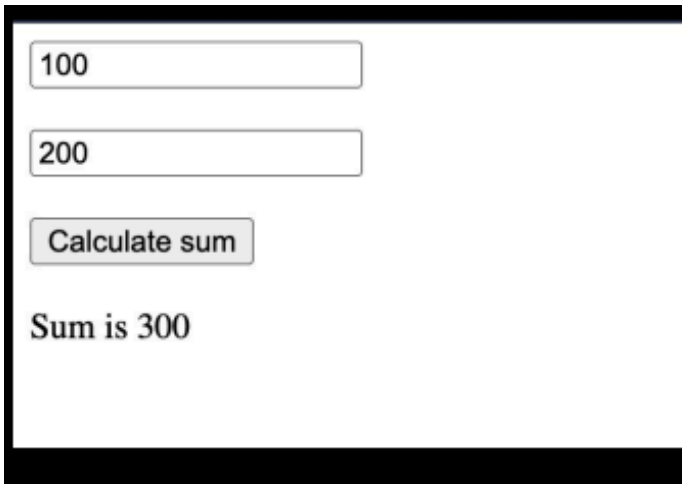
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<script>
    function populateDiv(){
        // document
        // hello world
        const element= document.getElementById("finalSum")
        element.innerHTML="Hello world";
    }
</script>
<body>
    <input type="text" placeholder="First number"><br></br>
    <input type="text" placeholder="Second number"><br></br>
    <button onclick="populateDiv()">Calculate sum</button><br>
    <div id="finalSum"></div>
```

```
    <!-- to make it dynamic we will use DOM Api -->
</body>
</html>
```

After clicking on Calculate sum we render something new



Now lets complete this assignment



How can i access the internal value of both the internal boxes?

Browser return a string when we use .value

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<script>
    function populateDiv(){
        const a=document.getElementById("firstNumber").value;
        const b=document.getElementById("secondNumber").value;
        const element= document.getElementById("finalSum")
        let c = parseInt(a)+parseInt(b);
        element.innerHTML=c;
    }
</script>
<body>
    <input id="firstNumber" type="text" placeholder="First
number"><br></br>
    <input id="secondNumber" type="text" placeholder="Second
number"><br></br>
    <button onclick="populateDiv()">Calculate sum</button><br>
    <div id="finalSum"></div>

    <!-- to make it dynamic we will use DOM Api -->
</body>
</html>
```

Q/Na:

1. **How to attach an event to input box change**

onchange

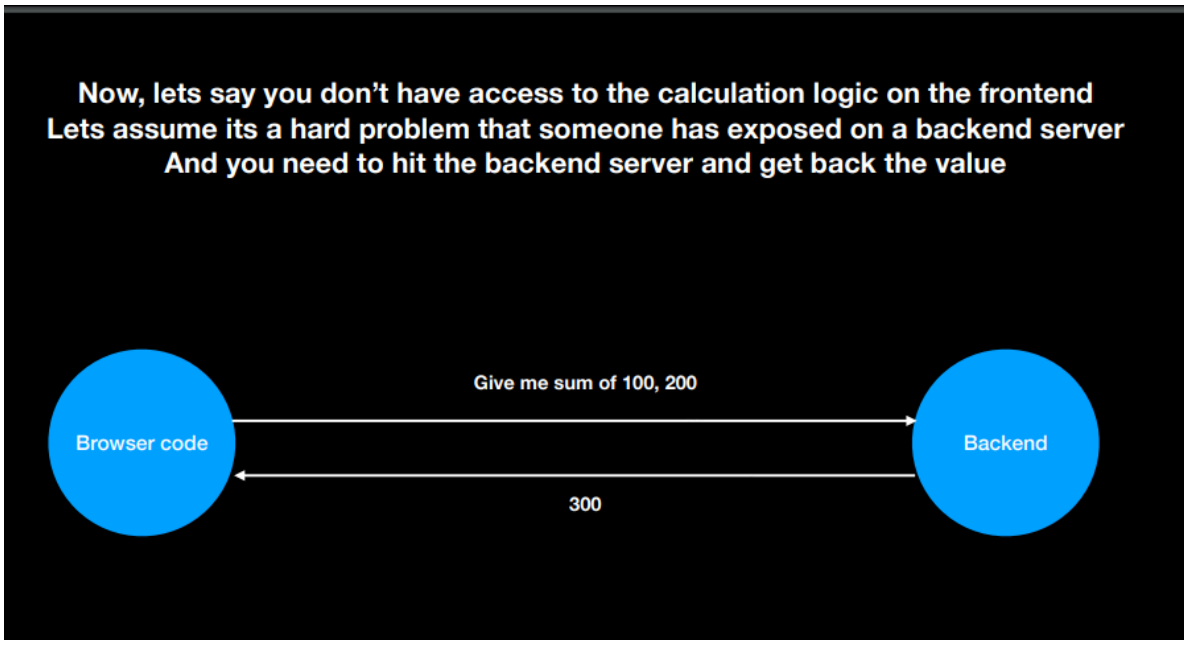
2. **DOM is foundation of react.**
3. **If a function return promise it is async**
4. **Doing the above example without button is onInput**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
</head>
<script>
    function populateDiv(){
        const a=document.getElementById("firstNumber").value;
        const b=document.getElementById("secondNumber").value;
        const element= document.getElementById("finalSum")
        let c = parseInt(a)+parseInt(b);
        element.innerHTML=c;
    }
</script>
<body>
    <input id="firstNumber" oninput="populateDiv()" type="text"
placeholder="First number"><br></br>
    <input id="secondNumber" oninput="populateDiv()" type="text"
placeholder="Second number"><br></br>
    <div id="finalSum"></div>

    <!-- to make it dynamic we will use DOM Api -->
</body>
</html>
```

```
    at <anonymous>:1:1
> document.querySelector('div')
<- <div class="container"> Hi bro </div>
> document.querySelectorAll('div')
<- ▼ NodeList(2) [div.container, div.container] i
    ▶ 0: div.container
    ▶ 1: div.container
      length: 2
    ▶ [[Prototype]]: NodeList
> document.querySelector(".dsz")
<- null
> document.querySelector("#dec")
<- null
>
```

5. **document.getElementsByClassName[“inputs”][0]**

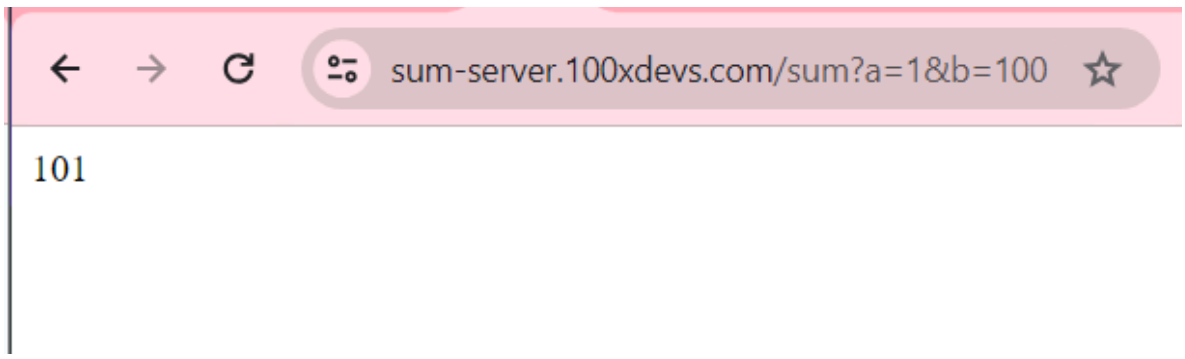


Our logic of addition is in the backend , logic is usually in backend so that world doesn't know the logic . For example chatGpt.

Lets say this calculation of adding two no. is expensive it is backend , we arent allowed to run this logic in the browser.

Backend Server

https://sum-server.100xdevs.com/sum?a=1&b=2



Result depends on the query parameter.

Here it is a=1 and b=100

Lets checkout the backend code:

```
const express = require("express");

const app = express();
app.get("/sum", (req, res) => {
    const a = parseInt(req.query.a);
    const b = parseInt(req.query.b);
    const sum = a + b;
    res.send(sum.toString());
  });

  app.get("/interest", (req, res) => {
    const principal = parseInt(req.query.principal);
    const rate = parseInt(req.query.rate);
    const time = parseInt(req.query.time);
    const interest = (principal * rate * time) / 100;
    const total = principal + interest;
    res.send({
      total: total,
      interest: interest,
    })

  });
```

We arent allowed to do

element.innerHTML = parseInt(a) + parseInt(b);

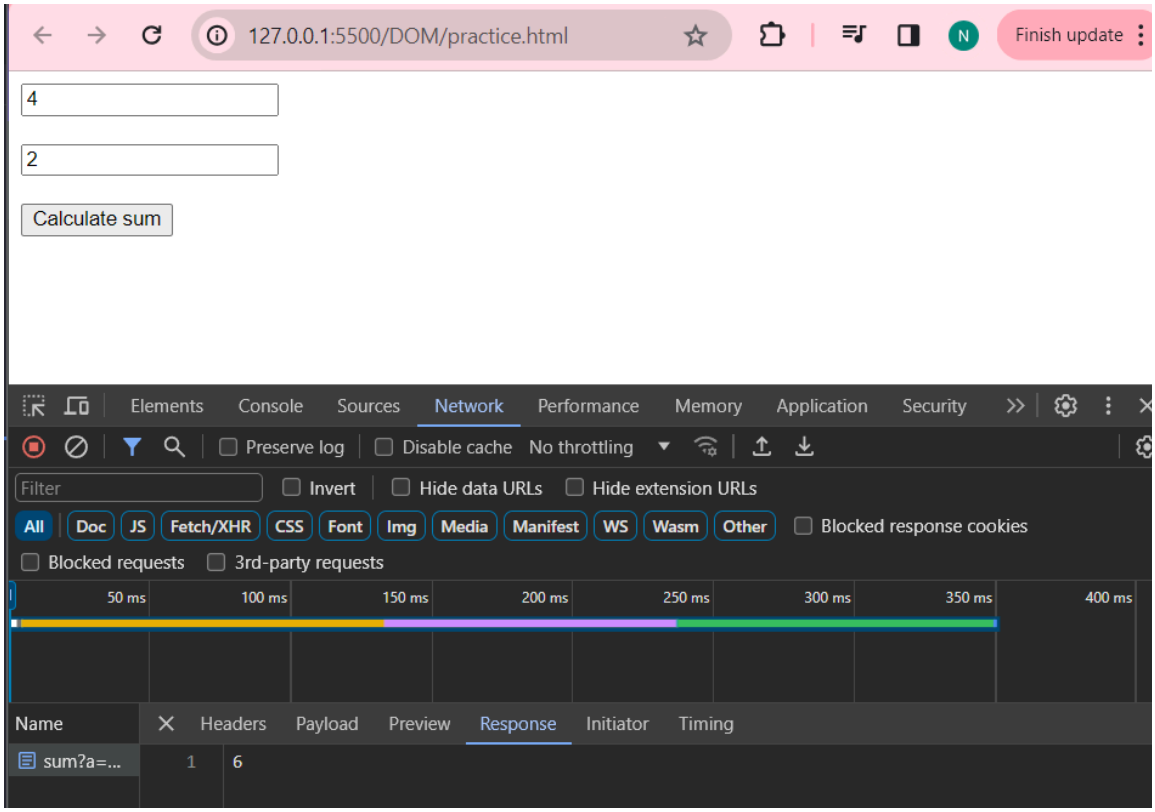
We have to hit the backend server and get a response.

Using fetch we will send request.

Fetch : a function to which we give endpoints to hit , these are method which we have to hit and these are the query parameters then it will give us a response.

Here the endpoint is already deployed.
We just have to hit it using fetch and get back a response.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<script>
    function populateDiv(){
        const a=document.getElementById("firstNumber").value;
        const b=document.getElementById("secondNumber").value;
        // const element= document.getElementById("finalSum")
        // fetch
        // a silent request goes: we dont have to refresh.
        fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
        // let c = parseInt(a)+parseInt(b);
        // we arent allowed to do this, we have to hit the backend server
and get the response
    }
</script>
<body>
    <input id="firstNumber"  type="text" placeholder="First
number"><br></br>
    <input id="secondNumber" type="text" placeholder="Second
number"><br></br>
    <button onclick="populateDiv()">Calculate sum</button><br>
    <div id="finalSum"></div>
</body>
</html>
```



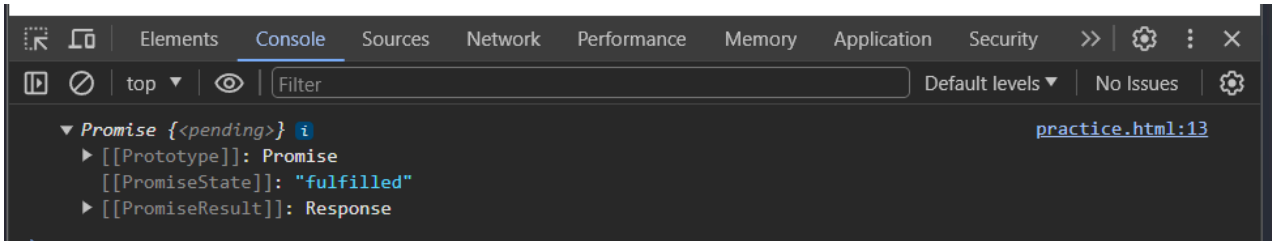
Our frontend is able to hit the backend.

Now the question arises is . How can we get back the response in a variable so that we can put it back in the div.

Fetch is asynchronous for example our code runs in HP and our backend server is located at the Kolkata.

Now lets see what will happen when we log the response

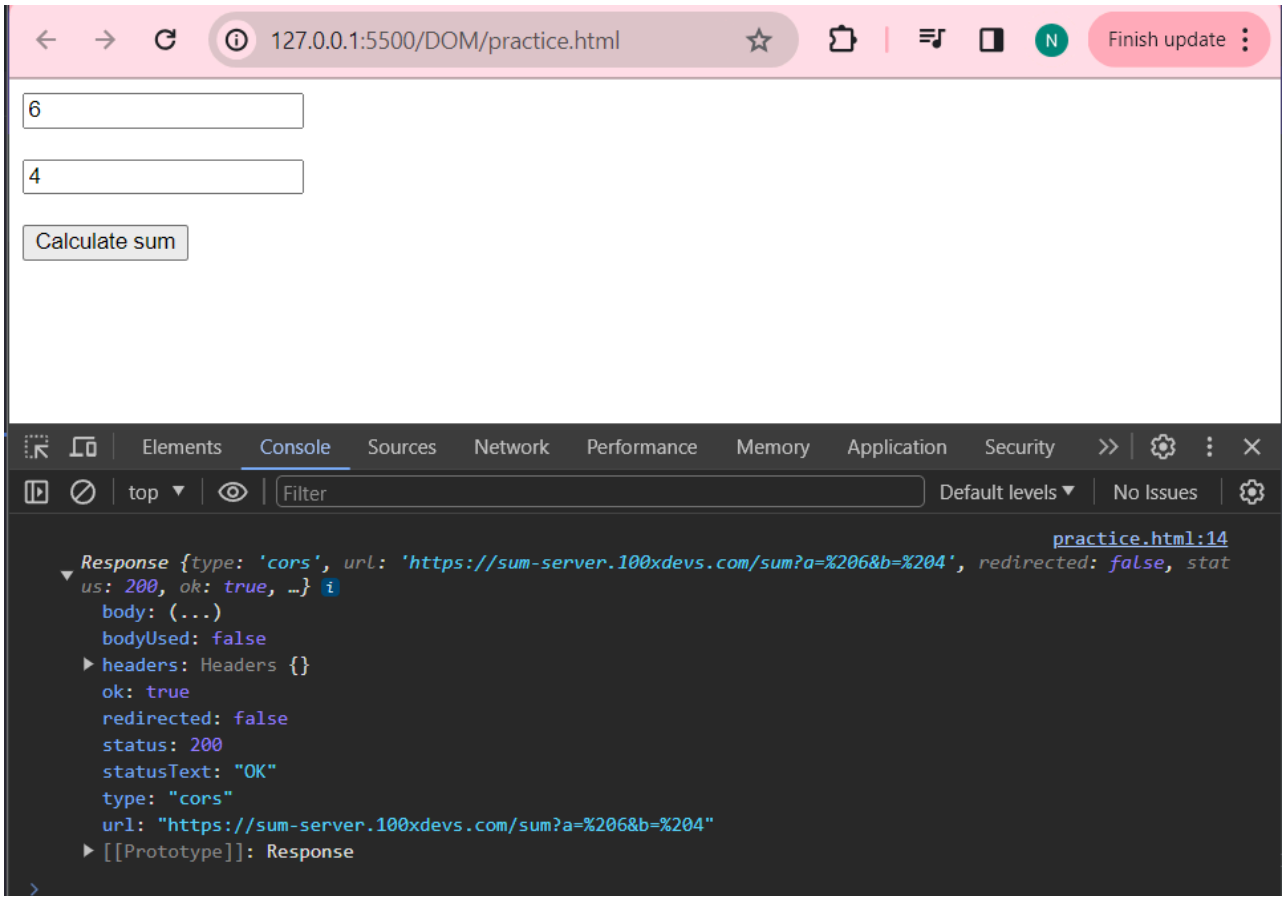
```
const res = fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
console.log(res);
```



When we dealing with the promise we add **.then**

When the promise resolve and when we get the data then we do something.

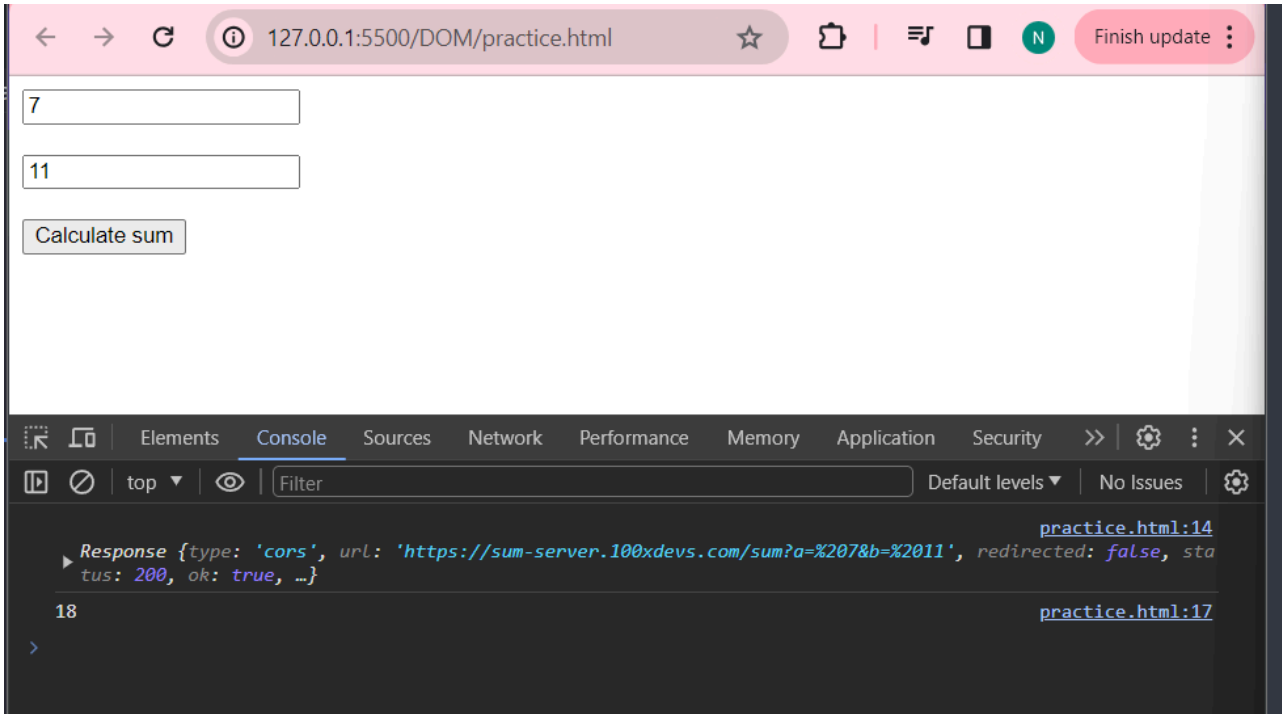
Now lets use .then along with the fetch




```
fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
        .then(function(response){
            console.log(response);
        })
```

Still we didn't get the answer 10.

It is the syntax of the fetch. (The alternative of the fetch is **Axios**)




```
        fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
        .then(function(response){
            console.log(response);
            response.text()
            .then(function(ans){
                console.log(ans);
            })
        })
```

Even response.text() is promise , it is the syntax of fetch.

If we try to access without using .then() we will get promise. But if we are using .then() , then we will get the actual data since we are waiting for the response

Final code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
</head>
<script>
    function populateDiv(){
        const a=document.getElementById("firstNumber").value;
        const b=document.getElementById("secondNumber").value;
        fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
        .then(function(response){
            console.log(response);
            response.text()
            .then(function(ans){
                console.log(ans);
                document.getElementById("finalSum").innerHTML = ans;
            })
        })
    }

</script>
<body>
    <input id="firstNumber"  type="text" placeholder="First
number"><br></br>
    <input id="secondNumber" type="text" placeholder="Second
number"><br></br>
    <button onclick="populateDiv()">Calculate sum</button><br>
    <div id="finalSum"></div>
</body>
</html>
```

Another way to do is async await syntax

```
    async function populateDiv(){
        const a = document.getElementById("firstNumber").value;
        const b = document.getElementById("secondNumber").value;
        const response = await
fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
        const ans = await response.text();
        document.getElementById("finalSum").innerHTML = ans;
    }
```

In any full-stack application in the long run, the Frontend's main job is to send a backend bunch of requests and render whatever came.

How we render something, it is done by DOM

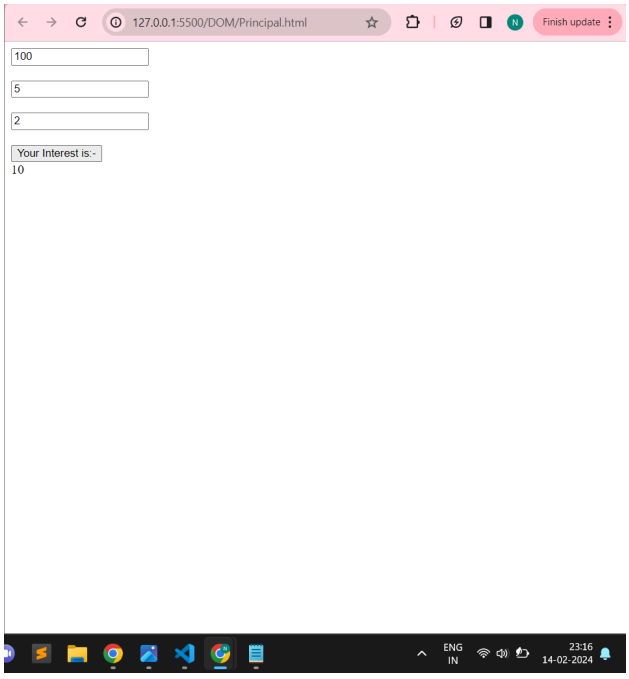
Assignment

We have to build a simple UI to calculate the interest in a principal amount after some time(in years)

Our backend which gives us the answer(does all the computation) is

https://sum-server.100xdevs.com/interest?principal=100&rate=5&time=2

Basically we have to print this




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

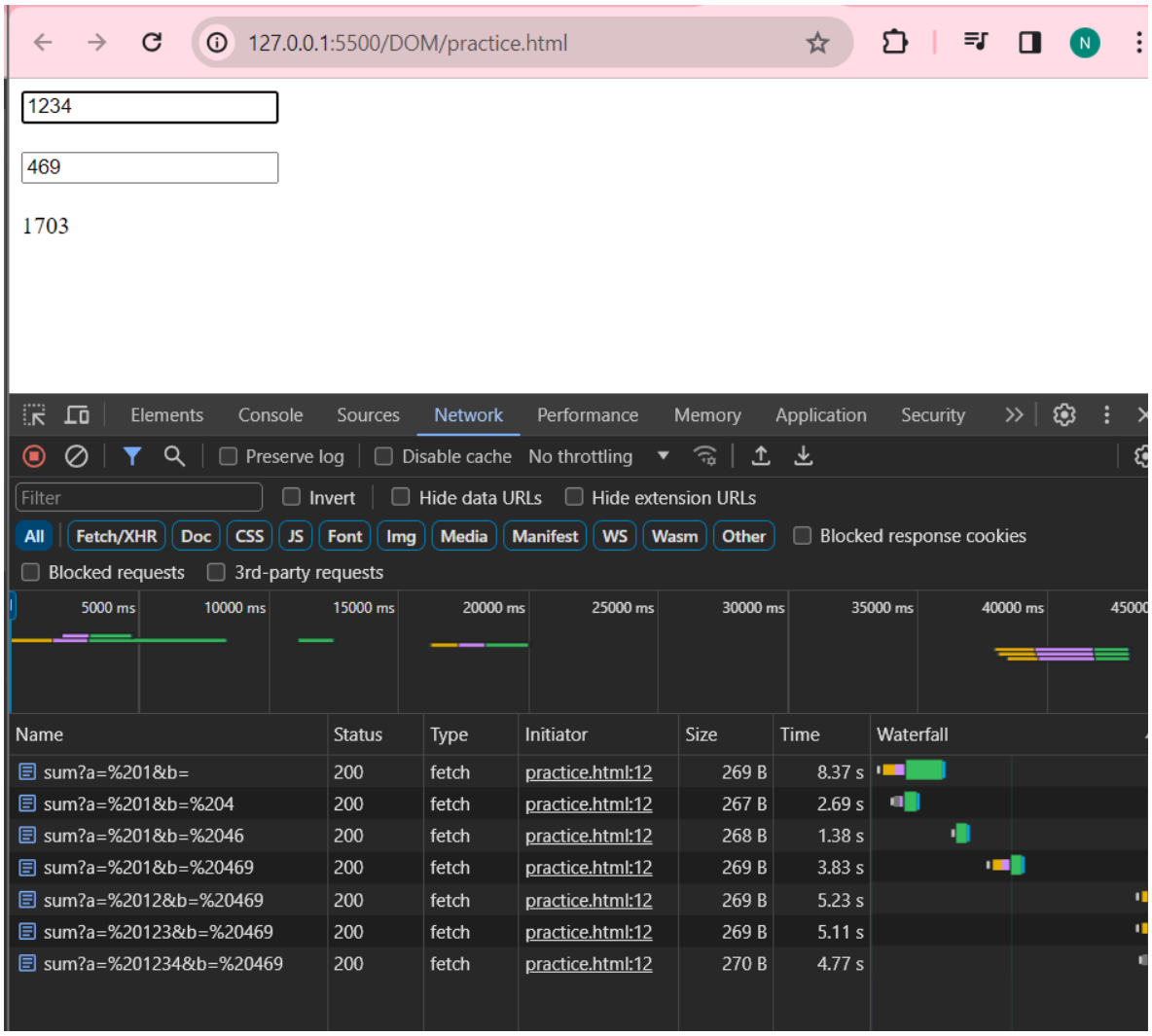
```
    <title>Document</title>
</head>
<script>
    function calculateInterest(){
        var p = document.getElementById("principal").value;
        var r = document.getElementById("rate").value;
        var t = document.getElementById("time").value;

fetch("https://sum-server.100xdevs.com/interest?principal="+p+"&rate="+r+"
&time="+t)
        .then(function(response){
            response.text()
            .then(function(ans){
                var ansObject = JSON.parse(ans);
                var fInterest = ansObject.interest;

document.getElementById("finalInterest").innerHTML=fInterest;
            })
        })
    }
</script>
<body>
    <input type="text" placeholder="Principal Amount"
id="principal"><br></br>
    <input type="text" placeholder="Rate" id="rate"><br></br>
    <input type="text" placeholder="Time" id="time"><br></br>
    <button onclick="calculateInterest()">Your Interest is:- </button>
    <div id="finalInterest"></div>
</body>
</html>
```

Throttling and Debouncing

Now we want that any time the input changes we need to send the request to the backend , we will not use the button

```
<body>
    <input oninput="populateDiv()" id="firstNumber"  type="text"
placeholder="First number"><br></br>
```

```
    <input oninput="populateDiv()" id="secondNumber" type="text"
placeholder="Second number"><br></br>
    <div id="finalSum"></div>
</body>
```



The problem is that one re-quest was sent to the backend server for each number.

Suppose our num1 = 100 and num2 = 213, we are sending 6 requests to the backend instead of one.

In amazon we can see that if we are typing slowly then they will suggest us product in each letter (request to backend on each letter). But if we are typing

very fast in amazon then the request to backend for the suggestion will be send at the end. They are somehow **debouncing** the request.
Example: If the user hasn't type for 100ms then we will send request to backend.

Hence we got why we need to **debouncing** or delay sending of the request.

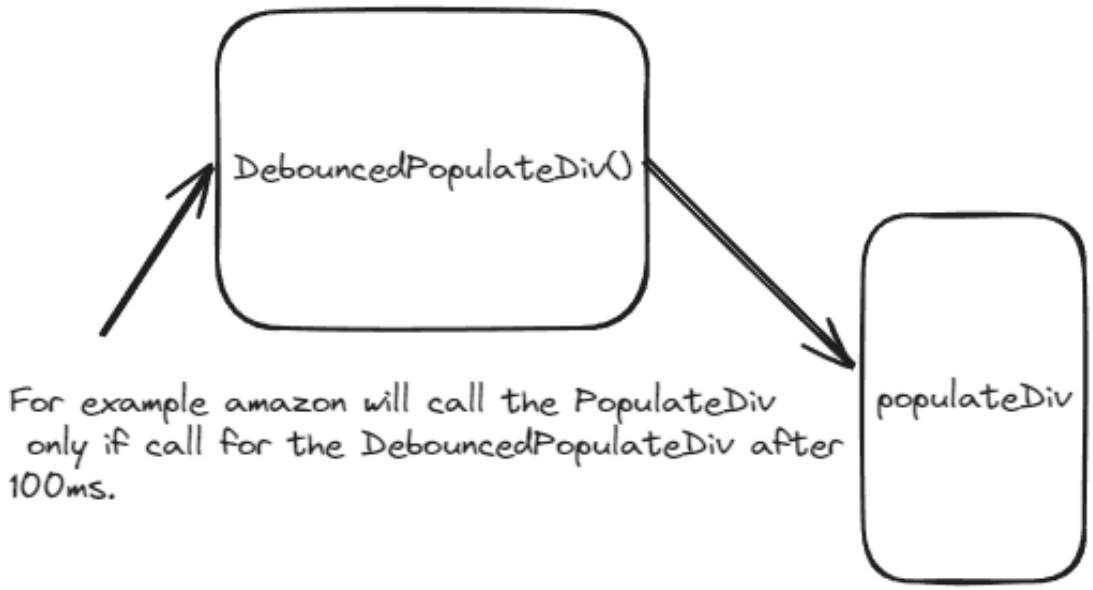
How to implement debouncing??

```
    function debouncePopulateDiv(){
        // delay the call to populate div untill ive not been called for 100ms.
        // and I've been called atleast once.
        setTimeout(function(){
            populateDiv()
        },100)
        // if we make this one second then the request will go after one second
        // but the problem is that when during that 100ms if we send another request then we will close this timer and start another timer. Which we havent done yet?
        // How do we cancel the clock
        // clearTimeout() let us clear the clock
        // Suppose when the clock is running we call the clearTimeout() then the clock will be stopped

    }
```

We are canceling the clock

```
        const timeout = setTimeout(function(){
            populateDiv()
        },100);
        clearTimeout(timeout);
```

Now if we observe the network tab in inspect we will observe that the request in not being send to the backend since as soon as the setTimeout() is started the clock is getting cancelled by the clearTimeout().

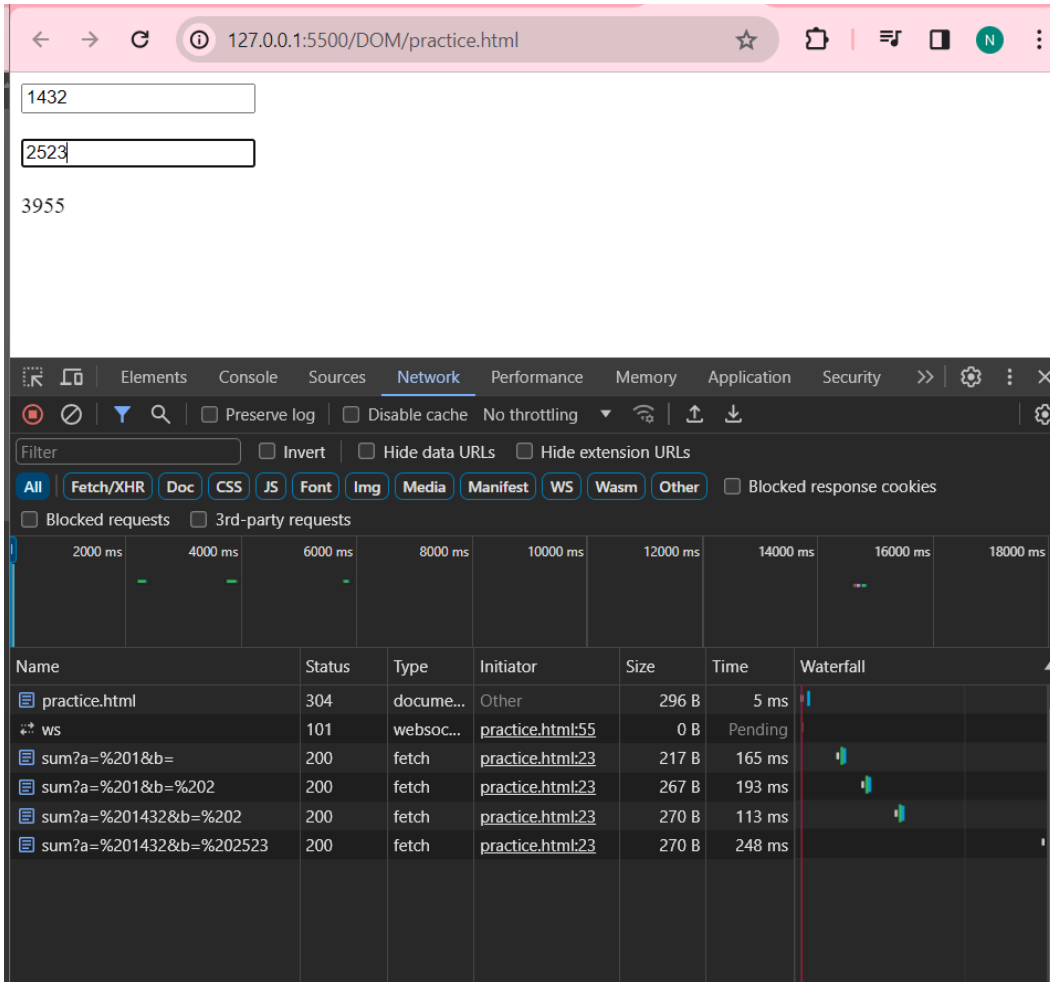


Here we have implemented the debouncing since we have made the request to the backend in less no. of time than the previous

.

```
    let timeout;
    function debouncePopulateDiv(){
        // delay the call to populate div untill ive not been called for
100ms.
        // and I've been called atleast once.
        clearTimeout(timeout);
        timeout = setTimeout(function(){
            populateDiv()
        },100)
    }
```

With introducing the debouncing we save the unnecessary calls to the backend



Q/Na

- **response.text() :**

Fetch (by default get) doesn't know what type of data (json,video, binary data)the backend has send to the user.

Hence it says the user to specify what type of data example:Textual data response.text() Json: response.json();

- **Axios Library:**

It is like a wrapper above the fetch library if we use axios it will look like this:

const response = await axios.get("https://sum-server.100xdevs.com/sum?a="+a+"&b"+b)
response.data

- **React under the hood is doing DOM manipulation only.**

Hence under hood we dont need it much.

When are creating chat it is better not store in react state and directly update the DOM element (by using refs)

- **Throttling:**

It is done in backend. We usually do the rate-limiting.

Throttling is done in case of video .

Slowly responsing

- **What if our fetch request require the authentication**

```
        const response = await
fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b,{
            headers: {
                "authorization":""
            }
        })
```

- **How does the text get slowly rendered in ChatGPT??**

Can we slowly send slowly request from the backend? No. Specially in the http

We dont get slow request from the backend coming in pieces.

- **In our example we are getting into callback hell . How to prevent it?**

```
    function populateDiv2(){
        const a = document.getElementById("firstNumber").value;
        const b = document.getElementById("secondNumber").value;
        fetch("https://sum-server.100xdevs.com/sum?a= "+a+"&b= " + b)
        .then(function(response){
            return response.text();
        }).then(function(ans){
            document.getElementById("finalSum").innerHTML = ans;
        })
    }
```

Promise chaining (await async : is better)

- **We dont want to overwhelm backend with the unnecessary request.**

We try to minimize the unnecessary request.

- Why in react we don't call the function just give the function name , but in DOM we will call the function
- If suppose a backend server can handle a million requests at a time . And if we want or expect users to send around a billion requests at a time then we will start a thousand machines and load balance them.
-

```
function debouncePopulateDiv(){
    clearTimeout(timeout);
    timeout = setTimeout(function(){
        populateDiv()
    },100)
}
```

  In this the id (timeout) of the setTimeout() function is sent immediately hence we can reset the clock almost immediately.
- Before website were not made dynamic , they were once written and they were as it is.
  For example dynamic things in linkedln is :
  message coming
  New post loading
  Like count increasing
  New Comments showing
- **cors**
  Allowing any frontend to access our backend