AJAX

Topics

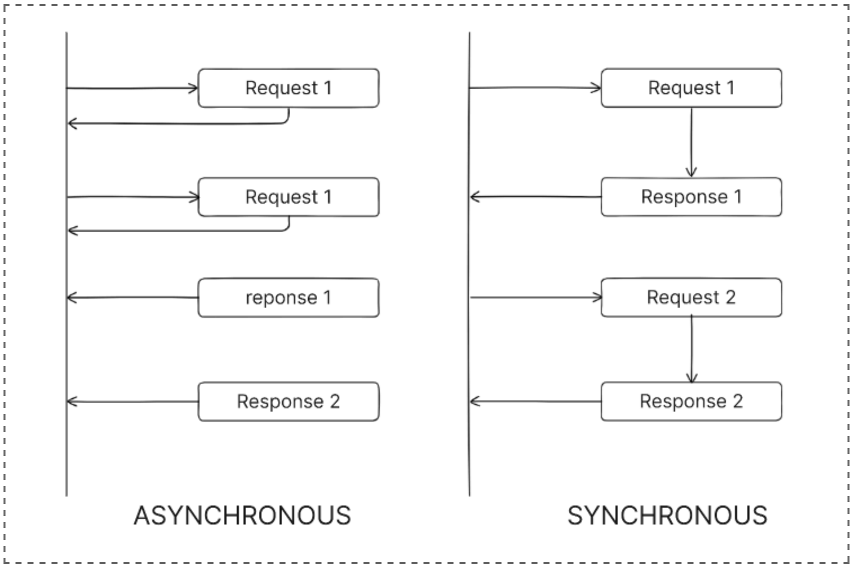
- Introduction to Async JS
- Introduction to AJAX
- The XMLHttpRequest object
- Making API request

Introduction to Async JS

Async JS which stands for Asynchronous JavaScript is a powerful programming concept that allows JavaScript to perform tasks concurrently, without blocking the main execution flow. It enables handling multiple operations simultaneously, leading to more efficient and responsive web applications.

One way to understand async JavaScript is to think of it as a way to run multiple tasks simultaneously. When you start an asynchronous operation, JavaScript does not wait for the operation to complete before continuing. Instead, JavaScript starts the operation and then continues to execute other code.

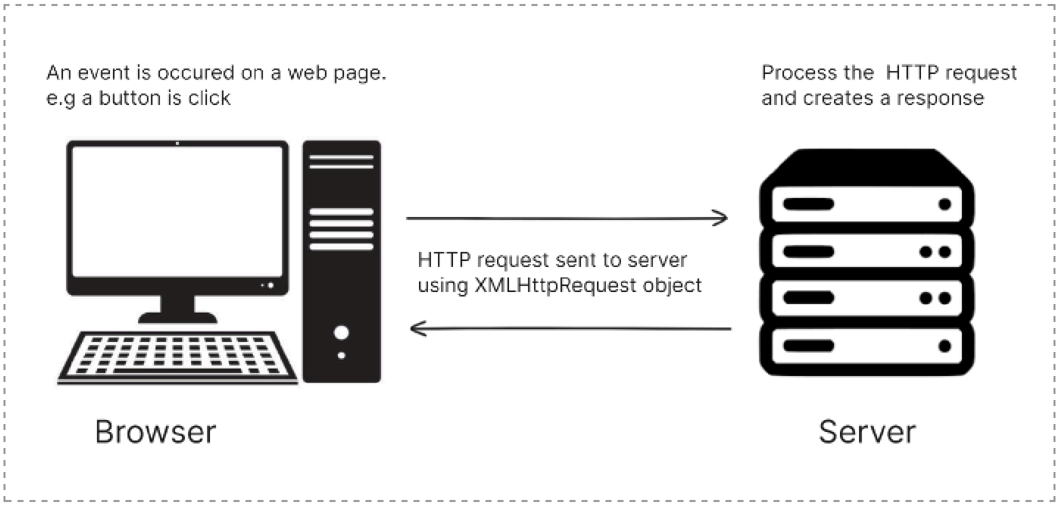
Here is a comparison diagram of Synchronous and Asynchronous Javascript programming concepts -



Introduction to AJAX

AJAX stands for Asynchronous JavaScript and XML. It is a set of techniques used to create interactive web applications. It allows web pages to be updated asynchronously without reloading the page. This means that parts of a web page can be updated without affecting the rest of the page, leading to faster and more responsive user interfaces, It allows for more efficient use of server resources since only the necessary data is fetched and updated and it also helps in loading dynamic content dynamically, making the web application more interactive and responsive.

Overview of how AJAX works -


An event is occured on a web page.
e.g a button is click
Process the HTTP request
and creates a response
HTTP request sent to server
using XMLHttpRequest object
Browser
Server

The XMLHttpRequest object

The XMLHttpRequest object is a fundamental part of modern web development. It's a JavaScript object that allows us to make HTTP requests to a server and retrieve data without having to refresh the entire webpage. Some of the reasons why XMLHttpRequest is used -

- Asynchronous Requests - It enables sending requests to the servers in the background, allowing the web page to remain responsive to user interactions.
- Update Parts of a Page - It allows the user to update specific parts of a webpage without reloading the entire page, creating a smoother user experience.

- Load data - Retrieve data from a server and use it to update the content dynamically.

To use the XMLHttpRequest object, you first need to create a new instance of it. You can do this by using the code below -

```
// creating new instance of the XMLHttpRequest
const xhrInstance = new XMLHttpRequest();
```

Once the new XMLHttpRequest object is created, it can be configured by calling the **open()** method. The **open()** method takes three arguments.

1. The HTTP method to use, such as GET, POST, or PUT
2. The URL of the web server resource to request
3. A boolean value indicating whether the request should be asynchronous

Example -

```
const xhrInstance = new XMLHttpRequest();
xhrInstance.open('GET', '/demo/url', true);
```

Some of the other methods and Properties of XMLHttpRequest are -
Methods -
1. abort() - cancels the current request.
2. getAllResponseHeaders() - Returns header information
3. getResponseHeader() - Returns specific header information
4. send() - send the request to the server, used for the GET request
5. send(string) - Sends the request to the server, used for POST requests
6. setRequestHeader() - Adds a label/value to the header to be sent

Properties -
1. onreadystatechange - defines a function to be called when the readyState property changes
2. readyState - holds the status of the XMLHttpRequest.
0: request not initialized
1: server connection established
2: request received

3: processing request
4: request finished and response is ready

3. responseText - returns the response data as a string
4. responseXML - returns the response data as XML data
5. Status - returns the status number of a request

200: “OK”
403: “Forbidden”
404: “Not Found”

6. statusText - returns the status-text (e.g. “ok” or “Not Found”)
7. onload - it allows you to specify a function that will be executed when the response from the server has been completely received and the HTTP status indicates a successful request.

Making API request

To make a simple API request, here a simple steps to follow -

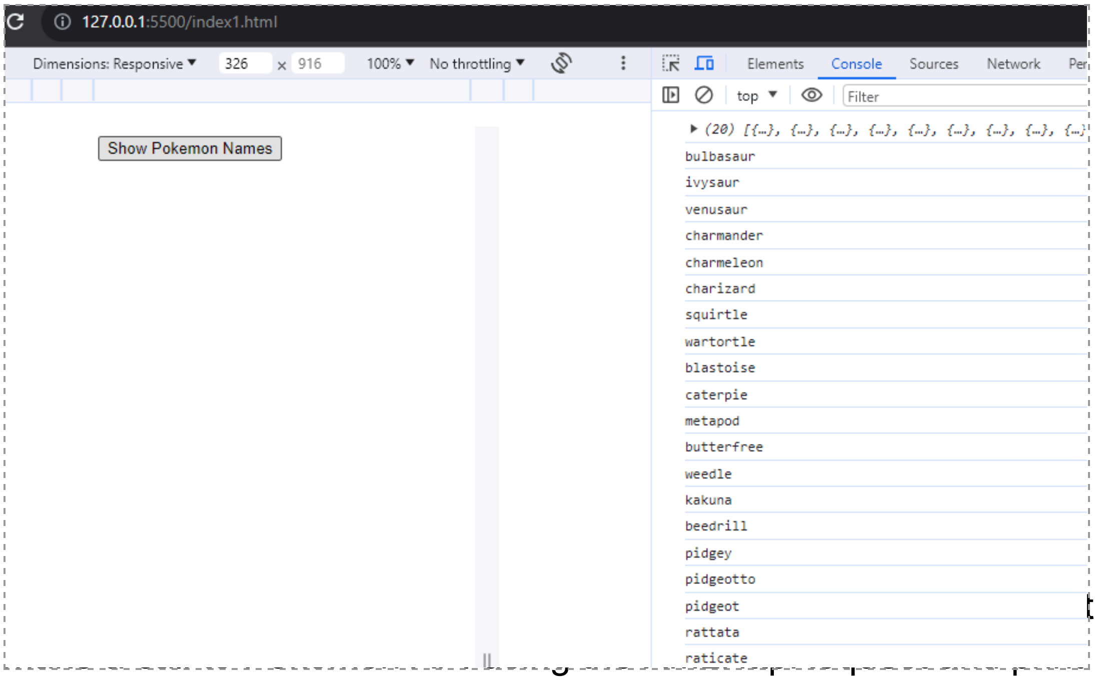
1. Create a new XMLHttpRequest object
2. Open a connection to the API endpoint
3. Set any request headers if needed.
4. Send the request by calling the send() method
5. Listen for the load event to be fired
6. Check the status

Example -

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>AJAX demo </title>
</head>
<body>
 <button onclick="getPokemonName()"> Show Pokemon Names</button>
 <script>
  function getPokemonName(){
   let xhr = new XMLHttpRequest()

   xhr.open("GET", "https://pokeapi.co/api/v2/pokemon/", true)
   xhr.onload = function() {
    if(xhr.status === 200){
     let pokemon = JSON.parse(this.response)
     console.log(pokemon.results)
     pokemon.results.forEach(element => {
      console.log(element.name)
     });
    }
   }
   xhr.send()
  }
 </script>
</body>
</html>
```

Browser console output -



```
▸ (20) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
bulbasaur
ivysaur
venusaur
charmander
charmeleon
charizard
squirtle
wartortle
blastoise
caterpie
metapod
butterfree
weedle
kakuna
beedrill
pidgey
pidgeotto
pidgeot
rattata
raticate
```

ton, will
all the
Pokemon names in the console inside the browser developers tools.

Note - to successfully incorporate with the example code, flux and Axios should be force installed with legacy peers
i.e -
npm install flux --legacy-peer-deps or npm install axios --legacy-peer-deps
npm install --force axios or npm install --force flux

*************** End ***************