AJAX & PROMISES

Topics:

- When we are dealing with get and post requests, we can check it in the network tab in our browser.
- Promises (Understanding)

Till now we have only accessed the file(s) which are not hosted anywhere, it only exits in a local system. In order to open that local file, it will use different protocol. If we use https before the address, it will not work because index.html file is not being served through a server currently. The local files open with ‘file’ protocol.

Once we refresh, we can see the index.html and script.js file in the Network tab. If we highlight or click on it, it shows the basic file path.

When we click on it, we can see the complete file in the Response Tab.

Since it had a dependency of another file script.js which is also again in my browser system, so it will also be fetched from the index.html file.

Lets add console.log(‘fetch’), we can see it in the Response Tab of script.js file.

The same thing repeats when we add the style.css file in the order that we have written it in the html file. Depending on which is coming first depending on the browser, you will see the files listed. There will be too many requests if we open a website. Everything that is fetched can be seen in the Network Tab.

FETCH:

Fetch is a top-level function which comes in with ES6. The fetch() method take sone mandatory argument, the path to the resource you want to fetch. It returns a promise that resolves to the response to that request, whether it is successful or not. You can also optionally pass in an init options object as the second argument

Once a response is retrieved, there are number of methods available to define what the body content is and how it should be handled.

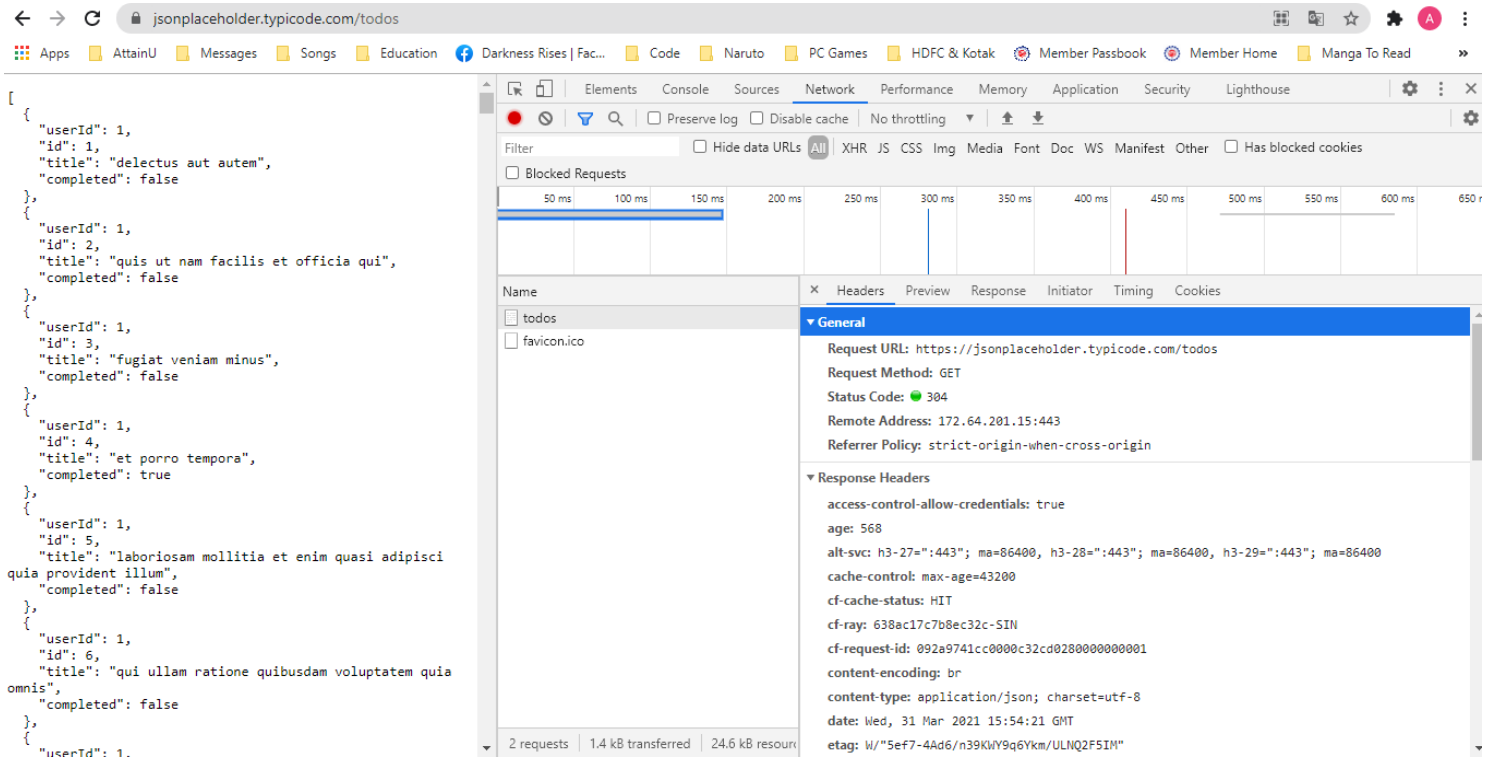
You can create a request and response directly using the request() and response() constructors, but it’s uncommon to do this directly. Instead, these are more likely to be created as results of other API actions.

So, coming back to JSON placeholder, in order to get some data, we have to give the url mentioned.

```
fetch ('https://jasonplaceholder.typicode.com/users')
```

Using the above url we are getting some data from there server. We have an array of objects in it.

https://jsonplaceholder.typicode.com/todos



The main idea is to get the data programmatically. We have to manually put the url in the browser to get the data, but we can do it using the **`fetch`** command as well.

Earlier, we used to have xml format to share the data. We had some custom defined tag between which the user data was given. It was wasting a lot of bandwidth because for each data, we have to give 2-tags. On the other hand, JSON using no tags.

JSON stands for “JavaScript Object Notation”. Let create a javascript object

```
const obj {
    name: "abc"
}
console.log(obj)
```

In the console tab we will have many properties for the object that we have defined. But if this was a json object, it would not have the ***`__proto__`***. Because when we are creating the object, we are creating a javascript object in the memory. In the abc.json file, it will have the same sort of notation just as JavaScript.

```
{
    "name": "Yash"
    "age": 23
    "fav": ["computer", "music"]
}
```

We would be using same sort of notation to the .json file as well. All **the keys** should have double quotes and unless the **value** is an integer, they will also have double quotes. Even while nesting, the **key** name should always be in double quotes. If we remove the quotes, the javascript will start throwing an error. The same format is being used in the https://jsonplaceholder.typicode.com/todos.

We can understand, the JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

Back to FETCH:

The fetch function will call the url depending on the way the data has been configured. The fetch actually returns a promise instead of the data. Meaning, the url does not directly give me a data but a promise/commitment, that the data will fetched depending up the internet. So,

```
const responsePromise = fetch ('https://jasonplaceholder.typicode.com/users')
console.log(responsePromise)
```

We can see the data in the Response tab, but at the time when data was fetched, it would not be available. Once the data is fetched, in the console tab we can see in the __proto__, *[[PromiseState:]] "fulfilled"*

All the promises have two functions attached to it (then & catch).

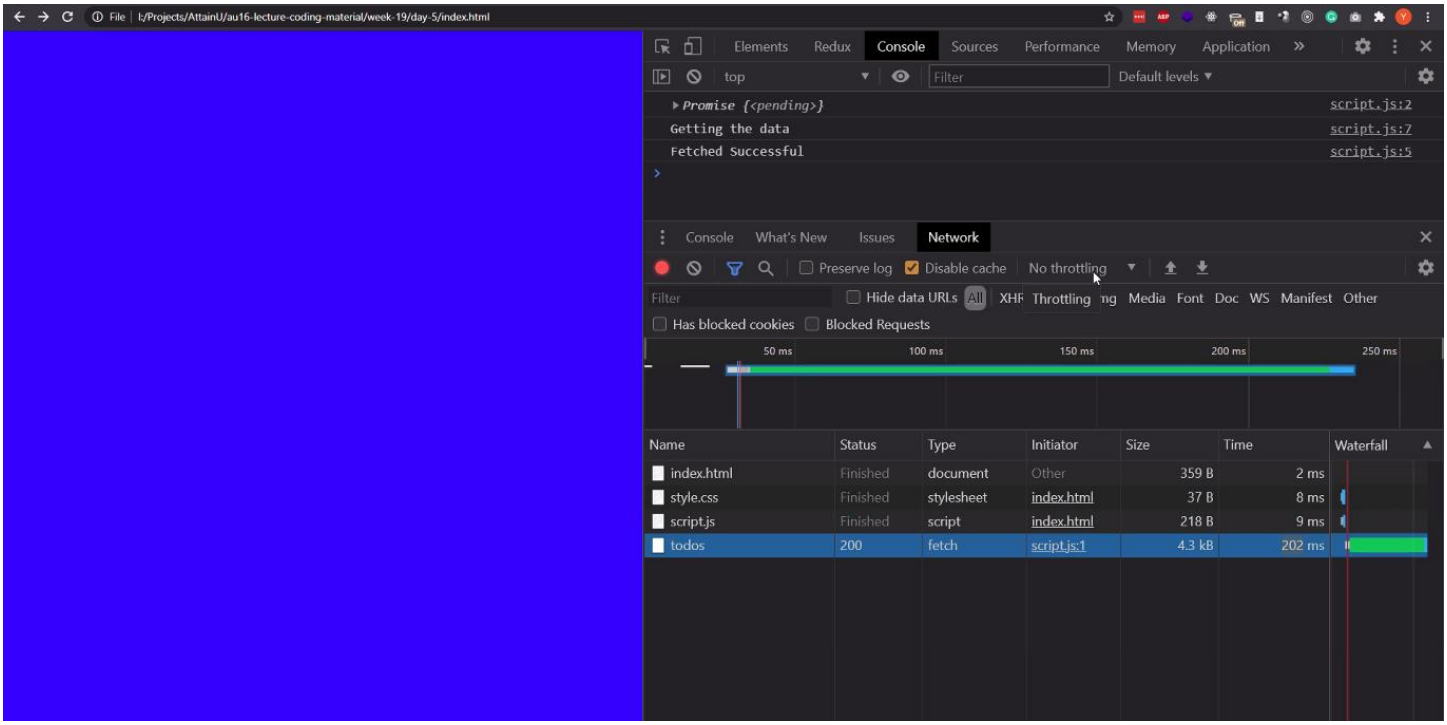
- The **then()** method returns a **promise**. It takes up to two arguments: callback functions for the success and failure cases of the promise. Once a **promise** is **fulfilled** or **rejected**, the respective handler function will be called asynchronously. The behavior of the handler function follows a specific set of rules.
- The **catch()** method returns a promise and deals with rejected cases only. It behaves the same as calling **then() function.** The promise returned by **catch()** is rejected if **onRejected** throws an error or returns a promise which is itself rejected; otherwise, it is resolved.

```
const responsePromise = fetch ('https://jasonplaceholder.typicode.com/users')
console.log(responsePromise)

console.log('Getting the data')

responsePromise.then(function() {
    console.log('Fetched Successful')
})
```

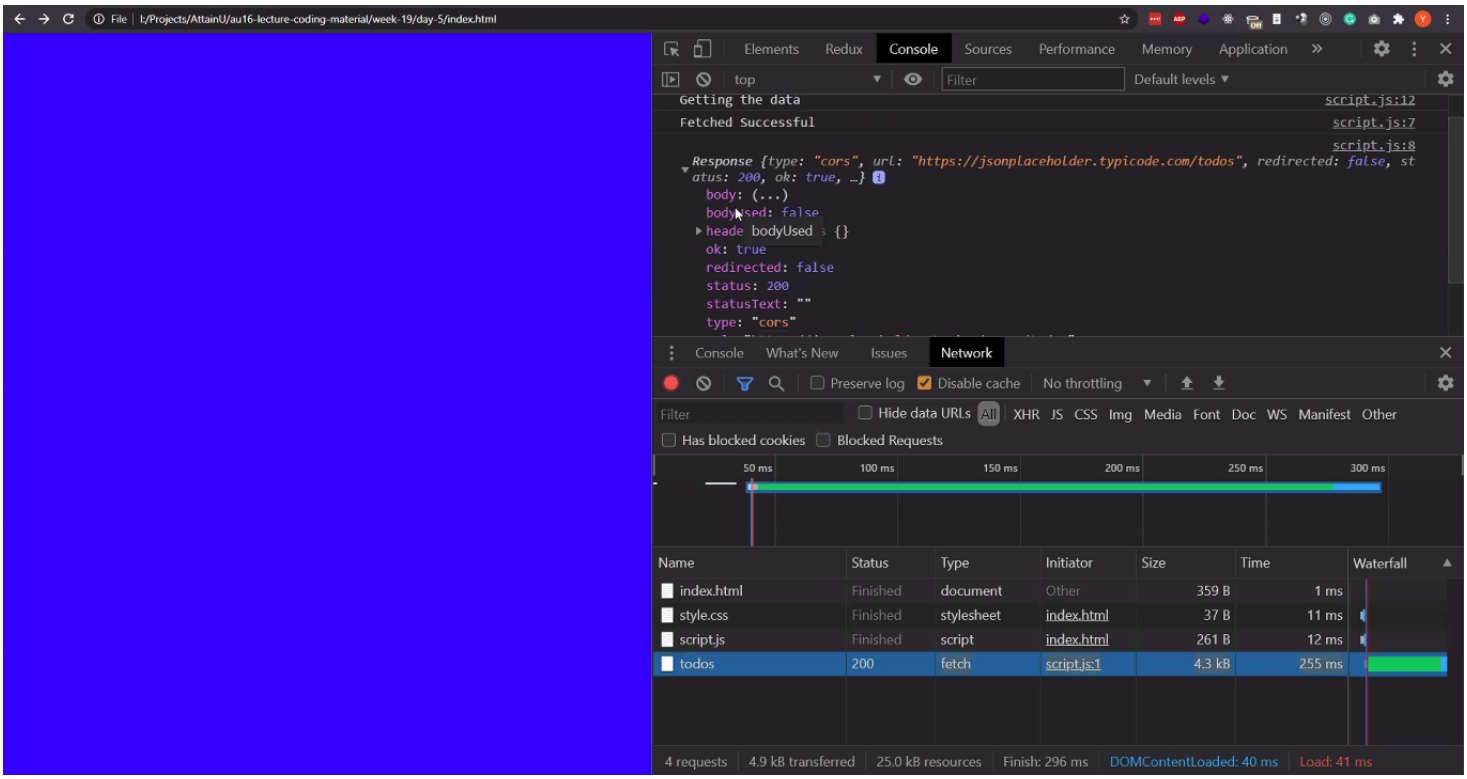
Output:

Getting the Data
Fetched Successful

`console.log('Getting the data')` is asynchronous in nature so it will handled after the first even call and after the data has been fetched then **`console.log('Fetched Successful')`** will be displayed in the console.



When the **`'then'`** function or the data is properly fetched, then this function will also be giving a response object. The response itself is also another promise.

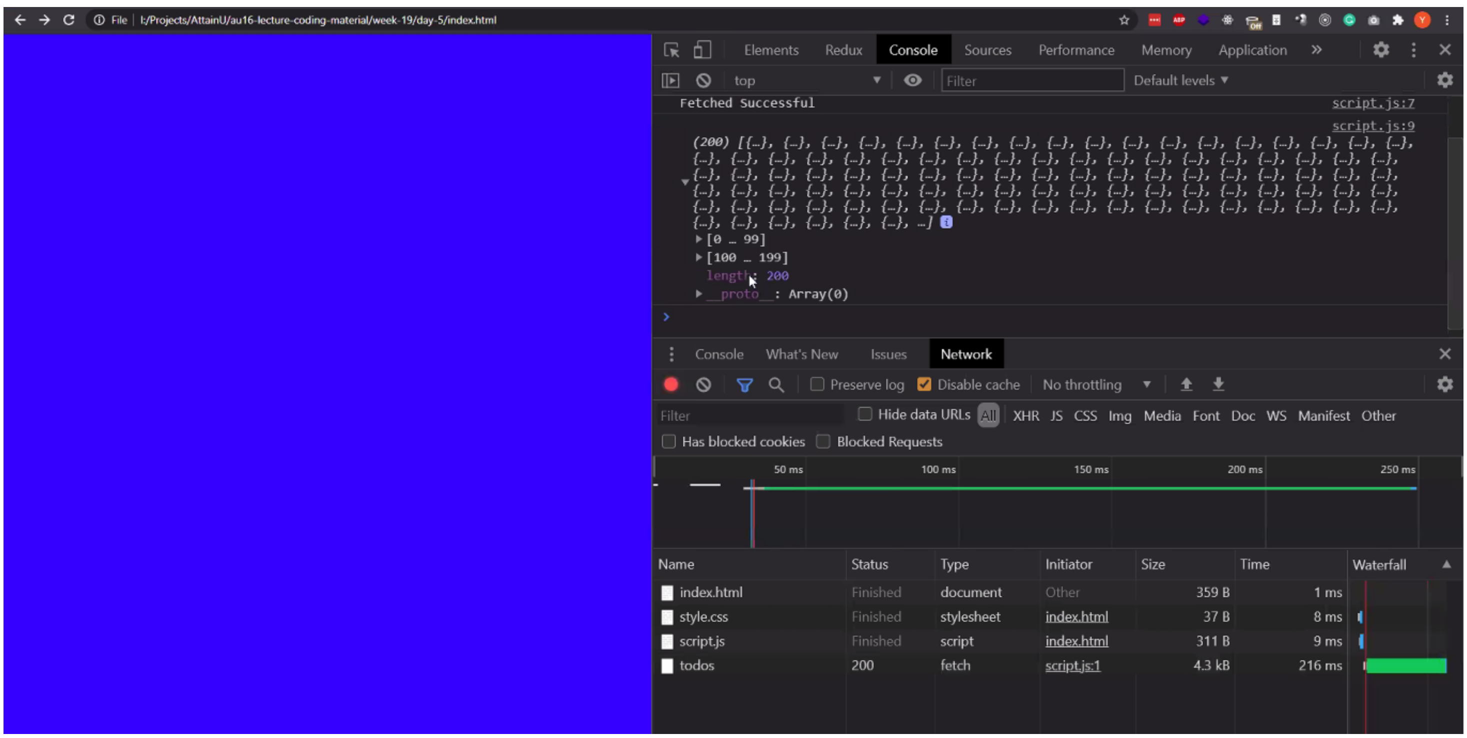

File | I:/Projects/AttainU/au16-lecture-coding-material/week-19/day-5/index.html
Elements
Redux
Console
Sources
Performance
Memory
Application
top
Filter
Default levels
Getting the data
script.js:12
Fetched Successful
script.js:7
script.js:8
Response {type: "cors", url: "https://jsonplaceholder.typicode.com/todos", redirected: false, status: 200, ok: true, …}
body: (...)
bodyUsed: false
headers: {}
ok: true
redirected: false
status: 200
statusText: ""
type: "cors"
Console
What's New
Issues
Network
Preserve log
Disable cache
No throttling
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
Has blocked cookies
Blocked Requests
50 ms
100 ms
150 ms
200 ms
250 ms
300 ms
Name
Status
Type
Initiator
Size
Time
Waterfall
index.html
Finished
document
Other
359 B
1 ms
style.css
Finished
stylesheet
index.html
37 B
11 ms
script.js
Finished
script
index.html
261 B
12 ms
todos
200
fetch
script.js:1
4.3 kB
255 ms
4 requests
4.9 kB transferred
25.0 kB resources
Finish: 296 ms
DOMContentLoaded: 40 ms
Load: 41 ms

Since it a promise, we have to

```javascript
const responsePromise = fetch ('https://jasonplaceholder.typicode.com/users')
console.log(responsePromise)

responsePromise.then(function(response) {

    console.log('Fetched Successful')
    response.json().then(function() {
        console.log(data)
    })
})

console.log('Getting the data')
```

Output:

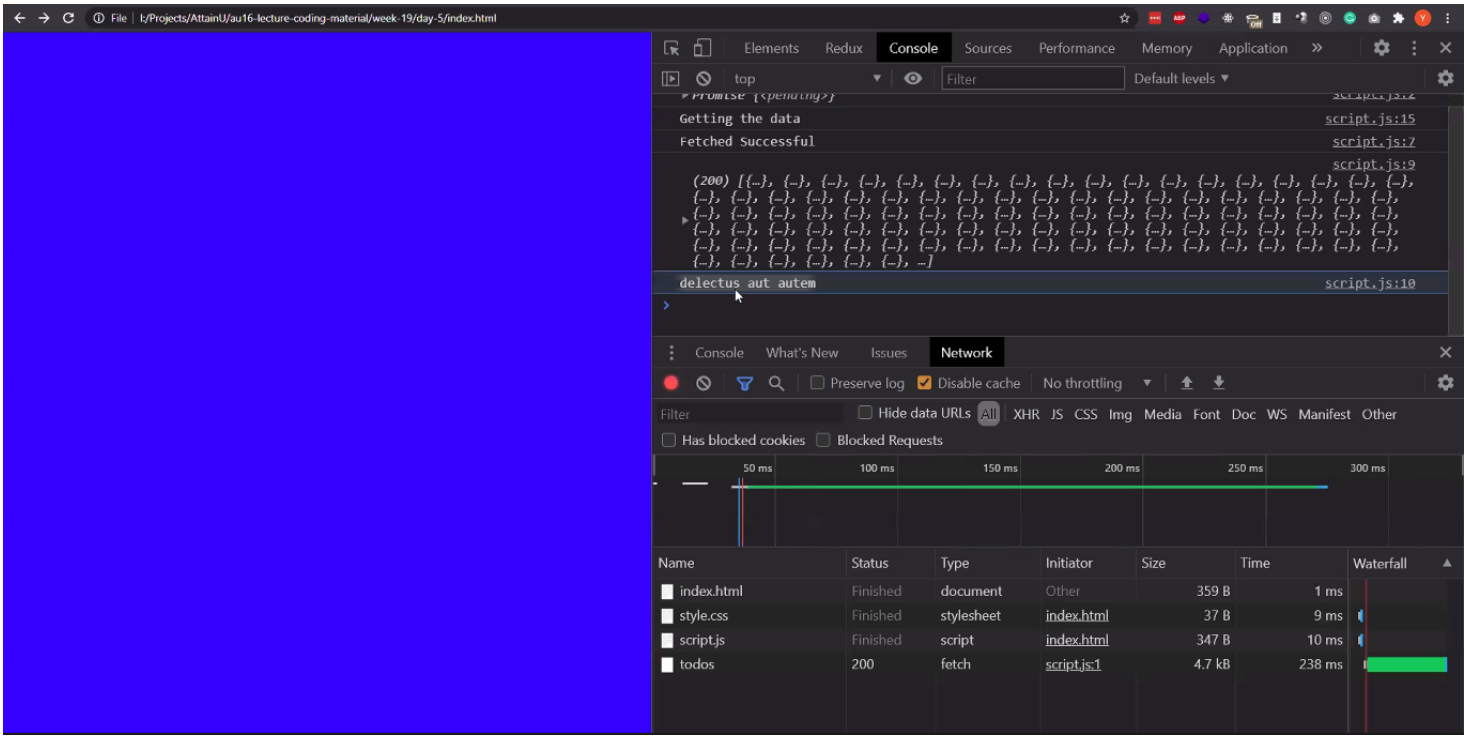
The .json() is converting my data into javascript object.



To access the first object of the json array,

```
const responsePromise = fetch ('https://jasonplaceholder.typicode.com/users')
console.log(responsePromise)

responsePromise.then(function(response) {

    console.log('Fetched Successful')
    response.json().then(function() {
        console.log(data)
        console.log(data[0].title)
    })
})

console.log('Getting the data')
```

Output:



In case there is an error

```
const responsePromise = fetch ('https://jasonplaceholder.typicode.com/users')
console.log(responsePromise)

responsePromise.then(function(response) {

    console.log('Fetched Successful')
    response.json().then(function(data) {
        console.log(data)
        console.log(data[0].title)

    })

}) catch (function (){
    console.log('ERROR')
})

console.log('Getting the data')
```

This is a very simplistic approach to get a data from a server.

To convert my javascript notation to json, we need to use JSON.stringify(obj). IT will convert the object into a string. This is what fetch is also doing behind the scenes.

Since data is in array, we can use all the array functions. So, we will create

```
const responsePromise = fetch ('https://jasonplaceholder.typicode.com/users')
console.log(responsePromise)

responsePromise.then(function(response) {

    console.log('Fetched Successful')
    response.json().then(function() {
        const paragraphs = data.map(todoItem => `<li>${todoItem.title}</li>`)

        document.body.innerHTML = `<ul>${paragraphs.join('')}</ul>`
    })

})
responsePromise.catch (function() {
    console.log('Error')
})
console.log('Getting the data')
```

We will get all the title elements from the objects. We can manipulate the dom, and even though we don't have anything in our html file, we are getting the data from the script.js file.

```
const getDataBtn = document.getElementById('getData')
getDataBin.addEventListener('click', handleBtnClick)

function handleBtnClick() {
    getDataFromServer()
}

async function getDataFromServer() {
    const responseObj = await fetch('https://jasonplaceholder.typicode.
com/users')
    console.log(responseObj)

    const jsonData = await responseObj.json()
    console.log(jsonData)
}
```

In an ideal world, everything works perfectly, but the moment we go offline, we will be getting errors. So to handle it, we can do **`try catch`** block

```
const getDataBtn = document.getElementById('getData')
getDataBin.addEventListener('click', handleBtnClick)

function handleBtnClick() {
    getDataFromServer()
}

async function getDataFromServer() {
    try {
        const responseObj = await fetch('https://jasonplaceholder.typico
de.com/users')
        console.log(responseObj)

        const jsonData = await responseObj.json()
        console.log(jsonData)

        const paragraphs = jsonData.map(todoItem=> `<li>${todoItem.title
}</li>`).join('')
        document.getElementById('todoItem').innerHTML = `<ul>${paragraph
s}</ul>`

    } catch (error) {
        document.getElementById('todoItem').innerHTML = 'ERROR'
    }
}
```

Async and await always work with promises.