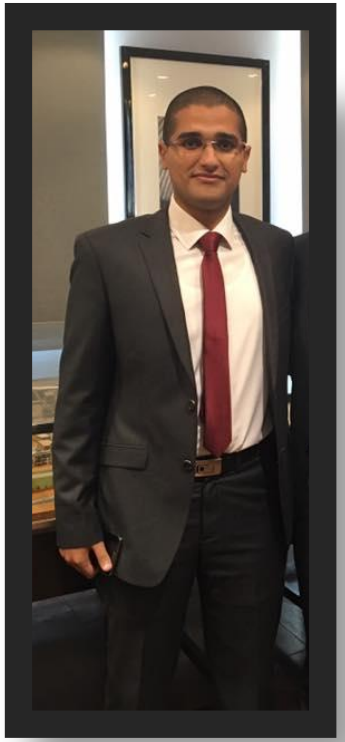
<Hello />



I' AM

MOATAZ

SANAD

Full-Stack Development Manager

I work at @ezzsteel

Keep Learning 💪

Starts 8:00 PM

Session Agenda

- Async JS, Promises, Async/Await & Fetch.
- React Router DOM.
- Project Walkthrough.

What is Async JS

The goal is to deal with long running task, like fetching data from a server

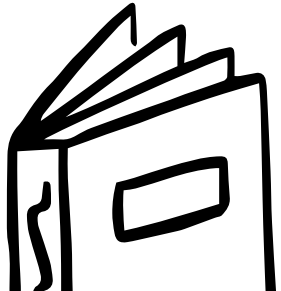
First What is Synchronous code: -

1- Most of the code you write is synchronous.

2- Synchronous code is executed line by line in order.

3- This means that we will not move to the next line until we are sure that the previous line is completed

```
let x = 1;
x = x + 2;
console.log(x);
```



What happen in Async JS

If JS engine detected asynchronous code, it will not block the synchronous code execution, instead it will run the asynchronous code in the background and wait until the synchronous code finishes then it executes **any register call back functions.**

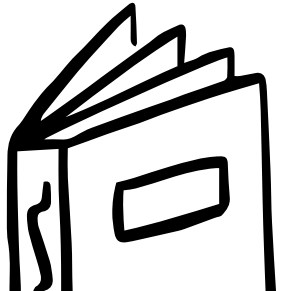
```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 3000);
console.log(3)
//OUTPUT will be 1,3,2
```

Execution Context

1- logging 1
2- detected async setTimeout then move it to background.
4- logging 3
5- Executing the register call back logging 2

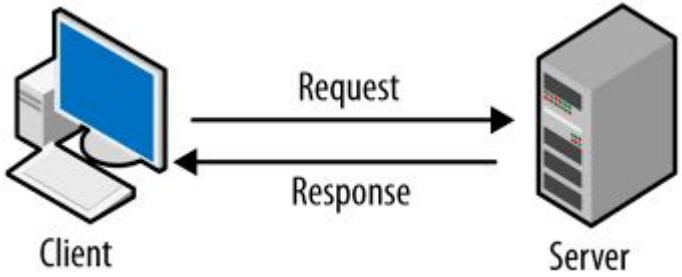
Background

3- Timer execution and registering the call back function



Ajax

Is an important example of Async JS



1- When you send Request JS will not block the execution waiting for the response from the server.

2- Instead you will register a callback function to be called when the server send the response.

Callback Hell / Pyramid of Doom

```
setTimeout(() => {
  console.log(1)
  setTimeout(() => {
    console.log(2)
    setTimeout(() => {
      console.log(3)
      setTimeout(() => {
        console.log(4)
      }, 1000);
    }, 1000);
  }, 1000);
}, 1000);
```

Promise & Fetch API

1- Promises & Fetch API will allow us to write a better code escaping the callback hell by chaining promises.

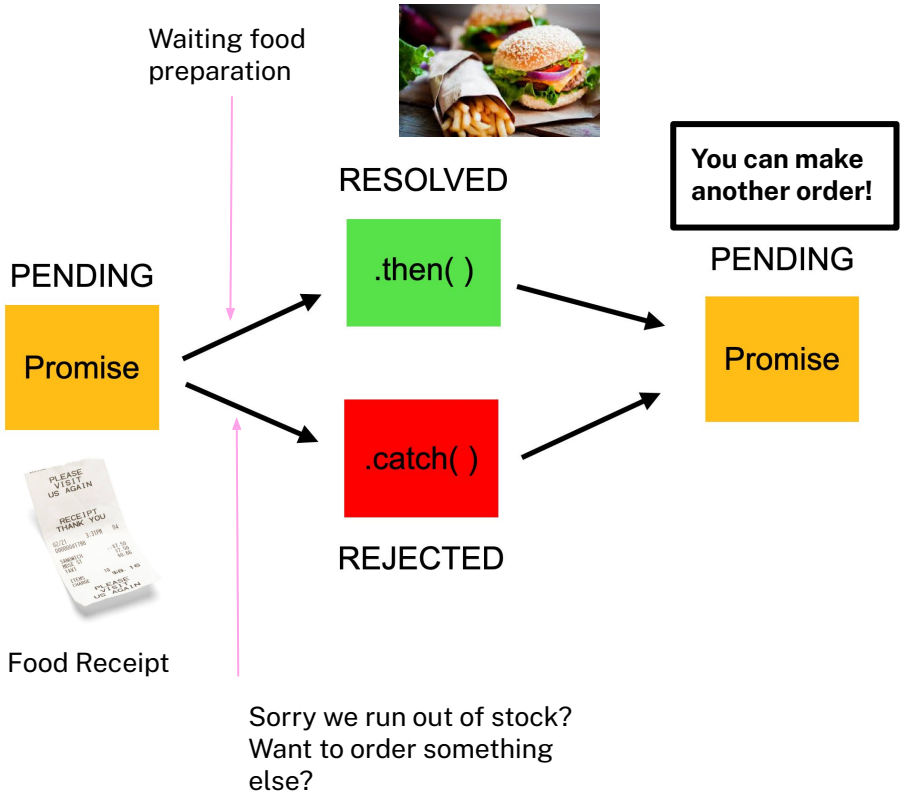
2- Fetch API immediately return Promise object.

Can You Think of an Example of a Promise in Real Life ? 🤔

What is a Promise 🤔

- **For simplicity you can treat a promise as an empty container that holds a future value.**
- **Example of future value is the response coming from the ajax call**
- **Read more here https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise**

Promise Life Cycle



- Fetch -> Creates and return a promise for you.
- You consume the promise using .then(cb)
- You pass callback to .then() and js will inject the response when it comes back from the server .then(response => log(response))
- To access data from response you need to call response.json() which returns a new promise.
- You handle errors in .catch()
- You can return another promise from .then()
- Now you can start chaining promises using .then()

Example ☺

```javascript
//Fetch create and return a promis
fetch('https://jsonplaceholder.typicode.com/todos/1')
//To access the future result, which will be response we need
//to call . then() passing a callback where js will inject
//the response object when it comes back from the server
.then(response => {
  //Response is not the data it hold alot of information about
  //server and status as well as the data.
  //To get the data you need to call .json() method, this method
  //return a promise, so we do the return statement to chain
  //the new promise to get the data
  return response.json()
})
// Since in the last .then() method we return a promise, now we can
// chain .then() to access the data when it is ready
.then(data => console.log(data))
//Handling error
.catch( e => console.log(e) )
```

```javascript
//Fetch create and return a promis
fetch('https://jsonplaceholder.typicode.com/todos/1')
//Since we only have one line of code
//we can omit { } and return keywork
.then(response => response.json())
.then(data => console.log(data))
.catch( e => console.log(e + 💀) )
//This code always runs after sucess or reject
.finally( ()=>{ //Code } )
```

Good VS Bad

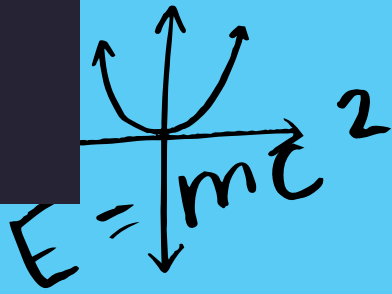
```
1  const normalFunction = () => {
2    fetch('https://jsonplaceholder.typicode.com/todos')
3      .then((response) => response.json())
4      .then((data) => {
5        console.log(data);
6        //We returned the promis to continue chaining .then()
7        return fetch(
8          `https://jsonplaceholder.typicode.com/users/${data[0]?.userId}`
9        );
10     })
11     .then((userResponse) => userResponse.json())
12     .then((user) => console.log(user))
13     .catch((error) => console.log(error + '😱'));
14 };
```

```
1  const badFunction = () => {
2    fetch('https://jsonplaceholder.typicode.com/todos')
3      .then((response) => response.json())
4      .then((data) => {
5        console.log(data);
6        //Take care you are doing the pyramid of doom or callback hell
7        fetch(`https://jsonplaceholder.typicode.com/users/${data[0]?.userId}`)
8          .then((userResponse) => userResponse.json())
9          .then((user) => console.log(user))
10         .catch((error) => console.log(error + '😱'));
11     });
12 };
```

Async / Await - Write Async code that looks like Sync code !

```
(async () => {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/todos');
    const data = await response.json();
    //Here we have all todos data
    console.log(data);

    //Now instead of nesting with .then() we continue and fire another request
    const userResponse = await fetch(
      `https://jsonplaceholder.typicode.com/users/${data[0]?.userId}`
    );
    const user = await userResponse.json();
    console.log(user);
  } catch (error) {
    console.log(error + 😱);
  }
})();
```



DEMO TIME

- https://github.com/mtzSanad/udacity-session-3-demo
- https://github.com/mtzSanad/udacity-session3-async-js-example.git


LIKES
THANK
YOU!