Async Await >> Promise Chains >> call Back Hell

Synchronous:

Codes runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

Asynchronous:

Due to sync programming, sometimes imp instructions get blocked due to some previous instructions , which causes delay in the UI. Async code execution allows to execute next instr immediately and doesn’t block the flow.

Callback : It is an argument to another function.

When we pass a callback, we never use paranthesis. If we do so, then we are just executing the function then and there itself.

In callbacks, we can pass just the name of the function whose definition is already written and also we can use arrow functions to do so.

Callbacks work both synchronously and asynchronously.

**When we use callbacks in the programming, then there are scenarios where callback hell may arise.**

Callback hell:

Nested callback scenario -> Nested callbacks stacked below one another forming a pyramid structure (pyramid of Doom)

Let’s, consider a real-life scenario, If we are entering username and password to login to Instagram, then the username will be searched first in the database. If the username is available, then only the control goes to search for the corresponding password. This can be implemented using callbacks.

If we want a particular flow to get executed in a specific interval of time, we can use callbacks.

To solve the problem of Callback Hell, The concept of promises into action.

Promises:

Promise is for “eventual” completion of task. It is an object in JS.

Example:

Amazon Delivery:

Item Reached – Promise Resolved

Order Cancelled – Promise Rejected

let promise = new Promise((resolve, reject) => {

    console.log("I am a Promise");

});

Resolve and Reject are handlers which are callbacks.

We have a prototype in the promise object.

Promise has 3 states:

1. Pending. Result - undefined
2. resolved. – the result is a value(fulfilled)
3. Rejected. – the result is an error object.

Example – Amazon Delivery

Resolve and Reject are default handlers (callbacks) created by JavaScript.

In general, we do not create a promise object, when we send a request to an API, it returns a promise which we use to play with the data. We just handle the promises.

As we just manage the returned promises , we just want to handle them. So we need some mechanism to do so.

We have two methods to handle fulfilled promise and rejected promise.

Methods to handle promise:

.then() and .catch()

Fulfilled promise is handled with .then()

Rejected promise is handled with .catch()

If we have a pending promise and we handled it with then() and wrote some code in that, that code executes only when the promise is fulfilled.

In then() , the default parameter is res

In catch(), the default parameter is err

function asyncFunction()

{

   return new Promise((resolve,reject) =>

   {

    setTimeout(() => {

        console.log("some data1");

        resolve("success")

    }, 4000);

   });

}

function asyncFunction2()

{

   return new Promise((resolve,reject) =>

   {

    setTimeout(() => {

        console.log("some data2");

        resolve("success")

    }, 4000);

   });

}

console.log("fetching data1")

let p1 = asyncFunction();

p1.then((res) =>

{

    console.log(res);

});

console.log("fetching data2")

let p2 = asyncFunction2();

p2.then((res) =>

{

    console.log(res);

});

What we ultimately want here is to fetch data1 and then data2. But the above code will give you both at a time which is not we wanted. So , promise chain come into play.

Promise Chain:

Using then () inside the other then ()

function asyncFunction()

{

   return new Promise((resolve,reject) =>

   {

    setTimeout(() => {

        console.log("some data1");

        resolve("success")

    }, 4000);

   });

}

function asyncFunction2()

{

   return new Promise((resolve,reject) =>

   {

    setTimeout(() => {

        console.log("some data2");

        resolve("success")

    }, 4000);

   });

}

console.log("fetching data1")

let p1 = asyncFunction();

p1.then((res) =>

{

    console.log(res);

    console.log("fetching data2")

    let p2 = asyncFunction2();

    p2.then((res) =>

{

    console.log(res);

});

});

Actual Chaining:

getData(1).then((res)=>

{

    return getData(2);

}).then((res)=>

{

    console.log(res);

});

Async & Await:

Async is a keyword and we can make any function async by just adding async keyword infront of it.

Async always returns a promise.

So, when we write async keyword infront of the function, then it is the duty of the function to return a promise mandatorily.

promise1-> resolved , promise2-> resolved, promise3->rejected, promis4-> rejected, Promise5 -> resolved.

Java Script Validations with forms

With email, text, number