Ramadhan Salman Alfarisi (35/XIRPL3)

Synchronous adalah yang paling umum dan mudah di mengerti. Setiap perintah di eksekusi satu persatu sesuai urutan kode yang anda tuliskan. Contoh :

Output dari kode diatas dijamin akan sesuai urutan, karena setiap perintah harus menunggu perintah sebelumnya selesai. Proses seperti ini disebut ‘blocking’.

Dalam dunia nyata ini mirip seperti antrian di BANK. Jika anda berada antrian nomor 4, maka anda akan dilayani setelah antrian 1–3 sampai selesai.

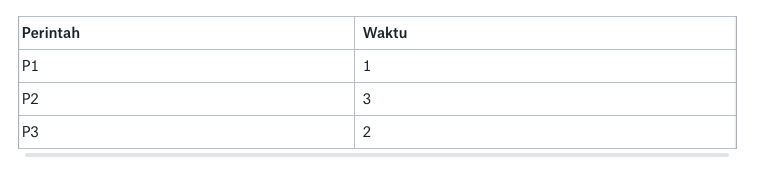
Sedangkan Asynchronous hasil eksekusi atau output tidak selalu berdasarkan urutan kode, tetapi berdasarkan waktu proses. Eksekusi dengan asynchronous tidak akan membloking atau menunggu suatu perintah sampai selesai. Daripada menunggu, asynchronous akan mengeksekusi perintah selanjutnya. Wait, sampai disini mungkin tidak masuk akal

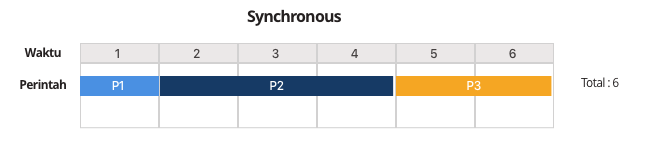
Contoh :

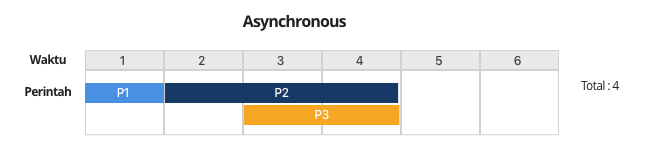
Catatan :  
Pada baris ke 2 setTimeout digunakan untuk menunda eksekusi dalam satuan milisecond dalam hal ini untuk simulasi prosess async.

Perhatikan bahwa outputnya tidak berurutan sesuai input (kode). Karena cara kerja asynchronous adalah berdasarkan waktu proses. Jika ada salah satu eksekusi membutuhkan proses yang agak lama, maka sembari menunggu prosess tersebut javascript mengeksekusi perintah selanjutnya.

Mari kita lihat performancenya, Sebagai contoh ada 3 perintah dengan waktu proses masing-masing.







Dari contoh simulasi diatas model eksekusi asynchronous lebih effisien.

Tapi ada yang menjadi pertimbangan yang disebut race condition. Race Condition terjadi ketika ada satu perintah yang bergantung pada output eksekusi asynchronous sebelumnya.Dengan kata lain kejar-kejaran. Contoh :

Dari contoh kode diatas besar kemungkinan displayUser menampilkan data kosong, karena belum tentu output dari ekskusi requestAjax sudah selesai.

Callback Pada Synchronous

1. Callback sebagai Injeksi sebuah function

function calculate(x,y){

result = x + y

return result

}

calculate(3,2) // 5

Dengan cara umum kita bisa menyelesaikanya dengan bantuan if atau switch untuk menguji operatornya. Tapi ini akan membuat code lebih panjang dan kurang dinamis. Dengan callback kita dapat membuat function diatas menjadi lebih dinamis

function calculate(param1,param2,callback){

//default operation

result = param1 + param2

// callback is function ?

if (typeof callback == 'function'){

result= callback(param1,param2)

}

return result

}

//execute

a=calculate(2000,4000, function(x,y){return "$ " + (x + y) })

b=calculate(7000,2000, function(x,y){return "Rp " + (x \* y) })

console.log(a) // $ 6000

console.log(b) // $ 14000

2. Callback sebagai Event Listener

Event Listener adalah function yang di eksekusi karena suatu event contoh ketika berinteraksi dengan DOM seperti event click, focus, keydown,keypress dan lain sebagainya.

Contoh pada Native Javascript

document.getElementById("my\_button").addEventListener("click",function(){

alert('Ouhh aku di klik!')

})

Jika anda pengguna jquery, pasti sudah terbiasa dengan event listener seperti ini :

$('#my\_button').on('click', function(e) {

console.log('Ouhh aku di klik!');

})

Callback Pada Asynchronous

Proses asynchronous identik dengan delay, dimana hasil dari proses tersebut membutuhkan selang waktu tertentu untuk menghasilkan output. Kita akan menemukan proses asynchronous pada proses Ajax, komunikasi HTTP, Operasi file, timer, dsb.

Pada synchronous output di prosess berdasarkan urutan kode.

Contoh :

function p1() {

console.log('p1 done')

}

function p2() {

console.log('p2 done')

}

function p3() {

console.log('p3 done')

}

p1()

p2()

p3()

/\* Output :

p1 done

p2 done

p3 done

\*/

Tetapi pada proses asynchronous output dari kode yang tuliskan tidak selalu berurutan. Hasilnya tergantung yang mana yang lebih dulu selesai. Perhatikan contoh berikut :

function p1() {

console.log('p1 done')

}

function p2() {

//setTimeout or delay for asynchronous simulation

setTimeout(

function() {

console.log('p2 done')

},100

)

}

function p3() {

console.log('p3 done')

}

p1()

p2()

p3()

/\* Output :

p1 done

p3 done

p2 done

\*/

Catatan : setTimeout digunakan untuk simulasi asynchronous. Karena sebenarnya kita tidak bisa membuat proses asynchronous murni.

Perhatikan output dari kode diatas tidak lagi berurutan. Kerena javascript mengerjakan mana yang lebih dulu selesai. Mungkin pada contoh diatas tidak terlalu masalah tapi pada kasus tertentu ini menjadi problem, Contohnya kita ingin menampilkan data yang harus di request terlebih dahulu dengan proses ajax.

Kemungkinan besar hasilnya undefined. Karena belum tentu data sudah tersedia ketika function showResult( ) di eksekusi. Teknik callback dapat kita gunakan untuk problem ini.

Baik sebelum membahas ajax lebih , kita akan coba memperbaiki challange asynchronus di atas dengan memastikan output p1,p2,p3 sesuai urutan.

Solusi :

Solusinya adalah dengan membuat p3 menjadi callback bagi p2.

function p1() {

console.log('p1 done')

}

function p2(callback) {

setTimeout(

function() {

console.log('p2 done')

callback()

},100

)

}

function p3() {

console.log('p3 done')

}

p1()

p2(p3)

Apa sih itu promise ?

Promise adalah salah fitur terbaru dari ES6. Jika anda sebelumnya sudah pernah menggunakan .then maka anda sudah menggunakan promise. Mari kita mulai dari analogi sederhana. Anda janjian ketemuan dengan salah satu kolega anda, tiba-tiba kolega tersebut bertanya anda sudah dimana ? Ada beberapa kemungkinan jawaban disini : dalam perjalanan, sudah sampai atau janjinya di batalkan.

Dalam dunia promise analogi di atas juga sama, ketika melakukan request asynchronous seperti Ajax, maka ada 3 kemungkinan state :

* Pending ( sedang dalam proses )
* Fulfilled ( berhasil )
* Rejected ( gagal )

Lalu bagaimana implementasinya dalam javascript ? Untuk sekarang ingat saja bahwa promise itu adalah object. Object yang merepresentasikan state diatas.

2. Callback vs Promise

Promise umumnya digunakan sebagai alternative callback. Salah satu tantangan di callback adalah callback hell. Disebut neraka ketika ada callback didalam callback didalam callback lagi dan di dalam callback lagi. Problemnya adalah kode sulit dibaca dan penanganan error nya juga menjadi sulit. Disaat seperti ini maka promise menjadi solusi.

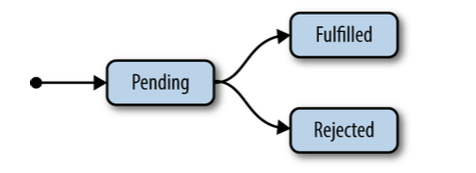
Sebelum mendalami promise sebaiknya pahamilah terlebih dahulu konsep callback. Promise bukan untuk menggantikan callback, karena promise akan selalu berjalan asynchronous sedangkan callback bisa digunakan untuk synchronous maupun asynchronous. Benefit utama dari promise adalah membuat code lebih readable dan manajemen error yang lebih baik.

Beberapa hal penting perbedaan callback dan promise adalah :

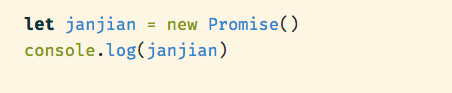
* Callback adalah function sedangkan promise adalah object.
* Callback di kirim melalui parameter, sedangkan promise mengembalikan object
* Callback digunakan untuk menghandle succes dan failure,sedangkan promise tidak
* Callback dapat digunakan untuk beberapa event sekaligus, sedangkan promise hanya untuk satu event

2. Membuat & Menggunakan Promise

Masih ingat dengan 3 state yang kita bahas sebelumnya ? Pending, Fulfilled, Reject.



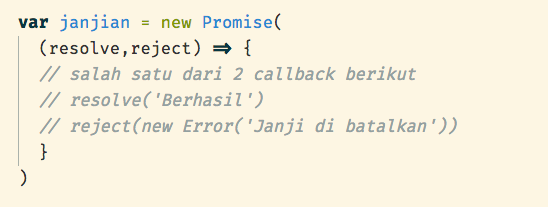
Untuk membuat promise cukup dengan memanggil constructor nya :



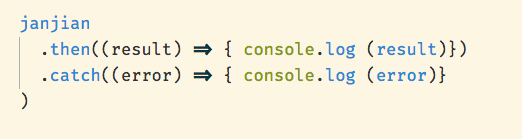
Sampai disini output dari code atas adalah Promise { <pending> }

Lalu bagaimana untuk mengatur state Fullfilled dan Reject ?

Untuk state ini gunakan salah satu listener, resolve() atau reject()

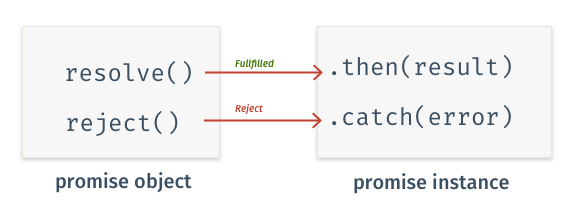


Untuk menggunakan promise diatas gunakan method then dan catch



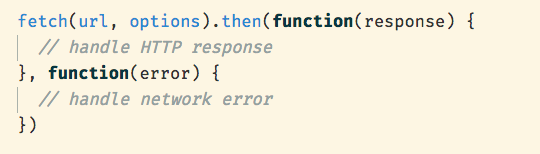
Output dari code diatas ada 2 kemungkinan,

* Jika comment pada resolve( ) di hapus maka hasilnya “berhasil”
* Jika comment pada reject( ) di hapus maka hasilnya “Janji di batalkan”

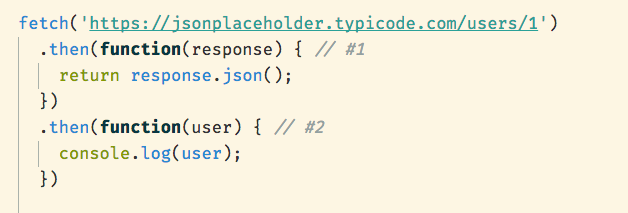


3. Request Ajax dengan Fetch

fetch adalah API yang di perkenalkan sejak ES6, untuk melakukan request ajax. Berikut boilerplate untuk penggunaan fetch.



Mari kita coba request ke API public.



Ouput dari code diatas json data user.



Tapi kenapa method .then di panggil 2 kali ?

Hal ini sering membuat bingung, apalagi bagi yang pertama kali mencoba fetch.

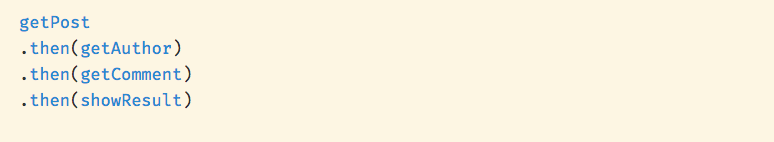
1. Response fetch pada #1 hasilnya adalah object Response, untuk lebih detailnya adan bisa lihat [disini](https://developer.mozilla.org/en-US/docs/Web/API/Response). Untuk mendapat response dalam bentuk json() maka digunakan method [Body.json()](https://developer.mozilla.org/en-US/docs/Web/API/Body/json)
2. Hasil dari method [Body.json()](https://developer.mozilla.org/en-US/docs/Web/API/Body/json) dibaca pada #2. Selain json kita juga menerima respon dalam format plain text menggunakan method [Body.text()](https://developer.mozilla.org/en-US/docs/Web/API/Body/text)

4. Promise Chaining

Seperti namanya promise chaining berarti promise berantai. Berikut contoh kasus untuk menampilkan post dengan alur seperti berikut :

1. request post
2. request author berdasarkan Id post
3. request comment berdasarkan Id post
4. tampilkan hasil

dalam bentuk pseude-code kira-kira seperti ini :



Berikut source code lengkapnya :

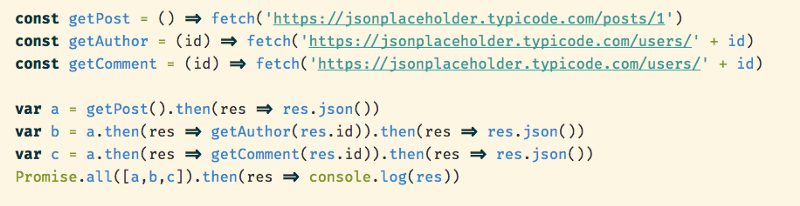


dirasa kode diatas sulit dibaca, mari kita per simple lagi dengan cara seperti berikut :



5. Promise.all

Promise.all sedikit lebih mudah, Cara kerjanya adalah menunggu hingga semua eksekusi promise selesai dan menghasil output dalam bentuk array. Sebagai contoh untuk kasus diatas menggunakan dengan Promise.all



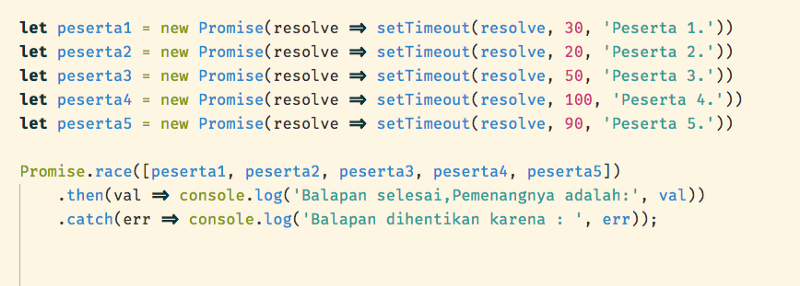
6. Promise in Looping

Salah satu kasus menarik di promise adalah mengeksekusi beberapa promise dalam looping sesuai urutan. Untuk memproses promise dalam looping, gunakan array.map dan Promise.all



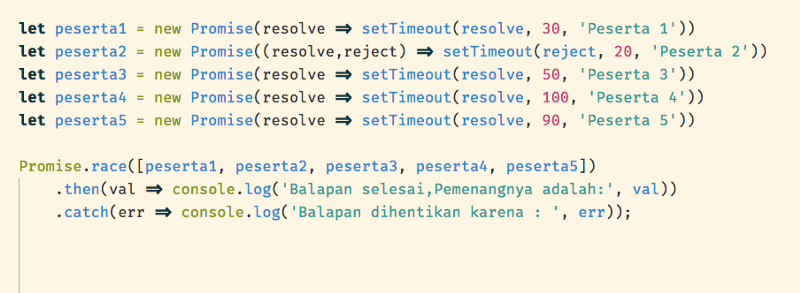
7. Promise.race

Berbeda cara eksekusi promise sebelumnya. Promice.race hanya menghasilkan promise yang lebih dulu selesai. Sebagai contoh kasus mari kita ambil contoh olahraga kegemaran kita semua yaitu balap karung.



Output : Balapan selesai,Pemenangnya adalah: Peserta 2.

Tapi bagaimana kalau satu peserta terjatuh ? ( ada promise yang reject )

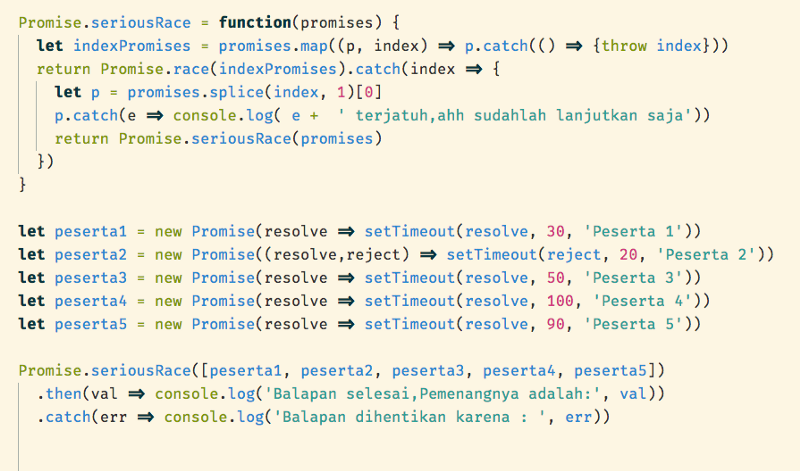


Output : Balapan dihentikan karena : Peserta 2

Seharusnya jika peserta 2 terjatuh maka pemenang adalah peserta 1, tapi karena jiwa mereka ini sangat nasionalis terpaksa balapan dihentikan.

Tapi karena kebetulan ini adalah pertandingan piala dunia balap karung, peraturan balapnya di ubah. Pertandingan tetap dilanjutkan meskipun ada peserta yang terjatuh.

Untuk ini method promise di extend untuk mendeteksi promise yang reject.



Output :  
Peserta 2 terjatuh,ahh sudahlah lanjutkan saja   
Balapan selesai,Pemenangnya adalah: Peserta 1

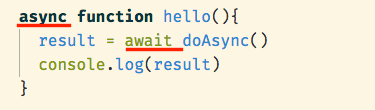
8. Library Promise

Semua fungsi promise yang di gunakan disini menggunakan standar ES6, jika anda ingin mengeksplor lebih dalam lagi tentang promise atau fitur promise standar belum cukup, anda bisa mencoba library-library promises yang menawarkan fitur yang lebih banyak seperti [Bluebird](https://github.com/petkaantonov/bluebird),[Q](https://github.com/kriskowal/q), [RSVP](https://github.com/tildeio/rsvp.js), [Catiline](https://github.com/calvinmetcalf/catiline) dan masih banyak lagi.

Async/await

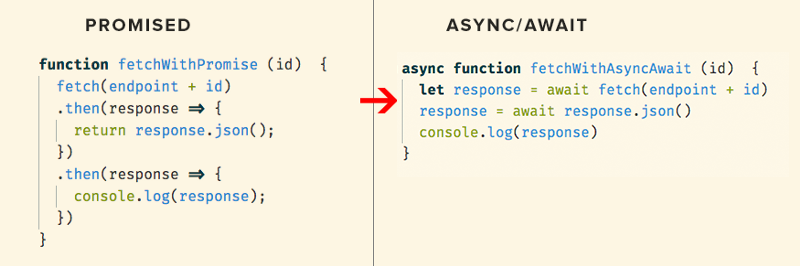
Async/await adalah fitur yang hadir sejak ES2017. Fitur ini mempermudah kita dalam menangani proses asynchronous. Untuk memahami async/await sebaiknya anda harus memahami [promise](https://medium.com/coderupa/panduan-komplit-asynchronous-programming-pada-javascript-part-3-promise-819ce5d8b3c?source=collection_home---4------0----------------) terlebih dahulu.

Ada 2 kata kunci disini yaitu async dan await, mari kita lihat contohnya :



Keterangan :  
async → mengubah function menjadi asynchronous  
await → menunda eksekusi hingga proses asynchronous selesai, dari kode di atas berarti console.log(result) tidak akan di eksekusi sebelum prose doAsync( ) selesai. await juga bisa digunakan berkali-kali di dalam function

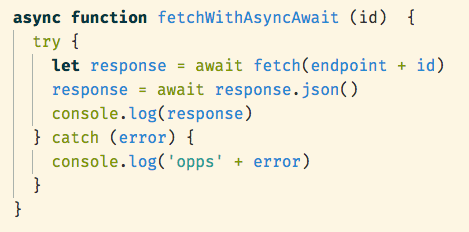
Sekarang mari kita coba untuk fetch dengan promise dan async/await



Output nya sama tapi async/await lebih mudah dibaca. Lalu bagaimana dengan manajemen error nya ?

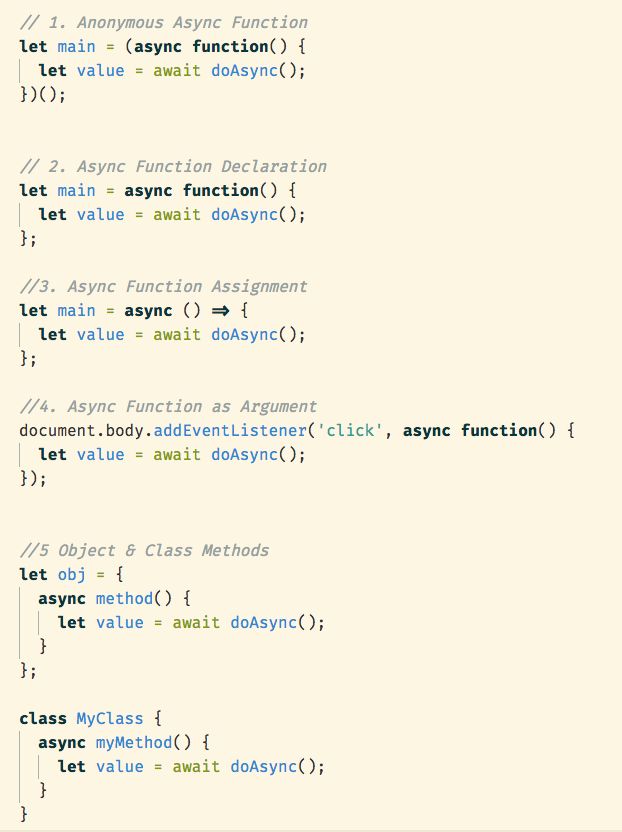
2. Async/await Error Handling

Error handling pada async/await menggunakan [try…catch](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)



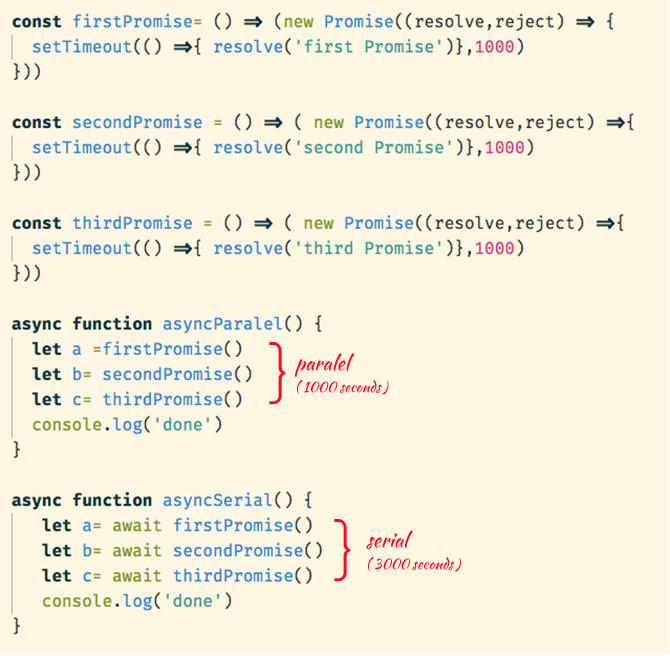
3. Async/await Pattern

Ada beberapa pattern untuk membuat async/await :



4. Serial & Paralel

Pada saat mengeksekusi beberapa proses asynchronous, ada kalanya kita harus memilih eksekusi secara serial atau parallel. Serial biasanya digunakan jika kita ingin mengeksekusi proses asynchronous secara berurutan. Sedangkan paralel jika ingin di eksekusi secara bersamaan, dalam hal ini urutan tidak menjadi prioritas tapi hasil dan performa.



5. Promise.all & Promise.race dalam Async/await

Salah satu favorite saya di promise adalah promise.all dan promise.race. Karena async/await di desain untuk bekerja dengan promise, maka dengan mudah bisa kita gunakan pada async/await, contohnya :



Perbedaan map dengan forEach

map() method menghasilkan array baru dari hasil masing-masing elemen array yang diaplikasikan pada fungsi map itu sendiri, tanpa merubah array aslinya. dan method ini punya return value sehingga bisa di terapkan pada sebuah variabel

var arr = [1,2,3,4,5];

var newArr = arr.map(function(a) {

return a\*10;

});

console.log(arr); // [1,2,3,4,5]

console.log(newArr); // [10,20,30,40,50]

sedangkan forEach() method hanya menjalankan sebuah fungsi sebanyak jumlah elemen array dengan variabel masing2 elemen itu sendiri. dan method ini tidak memiliki return value.

var arr = [1,2,3,4,5];

var newArr = arr.forEach(function(a) {

return a\*10;

});

console.log(arr); // [1,2,3,4,5]

console.log(newArr); // undefined

arr.forEach(function(a) {

console.log(a\*10); // fungsi console.log bejalan sebanyak 5 kali

});

// output

// 10

// 20

// 30

// 40

// 50

Menampilkan data object di JSON

