**JAVASCRIPT LANJUTAN 11. Promise**

Promise ini adalah sebuah solusi untuk menangani apa yang kita sebut dengan callback hel di video sebelum nya, untuk mengetahui apa itu Promise kita langsung ke codingan aja, sekarang saya udah puny halaman index.html yg didalam nya udah saya hubungin ke JQUERY.

Di video sebelum nya kita udah tau kalo misalnya kita mau melakukan Request ke sebuah API itu kita bisa menggunakan Jquery Ajax ya, misalkn kita ambil contoh ngambil data API nya dari omdb api simpel aja cara manggil nya gini :

$.ajax({

  url: "http://www.omdbapi.com/?apikey=dca61bcc&s=avengers",

  success: (*movies*) => console.log(*movies*),

});

Penjelasannya Jquery jalankan ajax terus didalam nya ada sebuah object yg url nya itu mengarah ke omdb api nya, lalu property berikut nya kita bisa tulis success untuk ngasih tau itu, kalo berhasil data nya udah di ambil kita mau ngapain nah kita bisa pake sebuah function ya kita pake arrow funtion ya karena itu mengembalikan satu parameter.

Kalo sudah itu kalo saya jalanin kalian bisa lihat di console.log nya bahwa data nya sudah di kembalikan, simpel aja ada sepuluh data film avengers, nah itu kalo menggunakan ajax simpel aja. Tapi perlu kalian inget itu kita menggunaan Liblary Jquery kita mengandalkan Liblary nya Jquery.

Nah kalo kita gamau mengandalkan Jquery kita bisa pake ajax versi vanila javascript atau javascript murni, seperti ini :

const xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

  if (xhr.status === 200) {

    if (xhr.readyState === 4) {

      console.log(JSON.parse(xhr.response));

    }

  } else {

    console.log(xhr.responseText);

  }

};

xhr.open("get", "http://www.omdbapi.com/?apikey=dca61bcc&s=avengers");

xhr.send();

Jadi kita jalankan Object Ajax lalu kita kasih pengkondisian kalo berhasil bagaimana kalo gagal bagaimana, dan data yg dikembalikan masih berupa text sehingga kita harus parsing ke dalam JSON yg bagian ( JSON.parse ) itu parsing, dan kalo kita jalanin hasil nya sama aja dengan menggunakan Jquery, kita udah ga ngandelin liblary external tapi code yg kita tulis lebih banyak.

Nah saat ini di javascript modern ada cara yg lebih simpel dari pada menuliskan object ajax nya seperti di atas, yaitu menggunakan fetch, gini cara nulis nya :

fetch('http://www.omdbapi.com/?apikey=dca61bcc&s=avengers');

tinggal kita kasih url api nya aja itu mau kemana, dan kalo kita msukin kedalam const dan liat di console.log nya. Gini :

const movies = fetch("http://www.omdbapi.com/?apikey=dca61bcc&s=avengers");

console.log(movies);

nah dengan begitu di console.log nya sudah melakukan request tapi problem nya adalah yg dikembalikan bukan data film nya, tapi bentuk nya promise, nah misalkan kita mau liat isinya si film nya itu, gini aja :

fetch("http://www.omdbapi.com/?apikey=dca61bcc&s=avengers")

  .then((*response*) => *response*.json())

  .then((*response*) => console.log(*response*));

Kalian cukup tulis .then yg isinya berupa callback terus itu kita jalankan ke sebuah method json() dan karena method json() itu mengembalikan promise, kita bisa tulis .then lagi terus kita kasih response lagi baru console.log response nya.

Dengan kita tulis seperti itu hasil di console nya itu sama dengan menggunakan ajax maupun object ajax vanila javascript. Nah karena fetch mengembalikan Promise makannya kita pelajarin promise dulu sebelum masuk ke fetch.

**Promise**

Promise itu artinya janji, definisi sederhana nya promise itu adalah sebuah object yang merepresentasikan keberhasilan atau kegagalan dari sebuah event yang asynchronous, yg akan terjadi di masa mendatang, janji kan ditepati nya dimasa yg akan datang ya.

Dan yg namanya janji itu pasti ada yg terpenuhi atau ingkar, dan kalo di javascript si janji ini kita sebut keadaan nya states dan kalo terpenuhi kita sebut dengan fulfilled kalo di ingkari janji nya itu rejected dan ada satu keaadan yg namanya pending ini waktu tunggu nya sebelum janji nya terpenuhi atau tidak.

Dan untuk menjalankan keadaan itu ada 3 fungsi callback yg harus kalian inget, resolve ketika janji nya terpenuhi, reject ketika janjinya tidak terpenuhi, finally ketika waktu tunggu nya selesai, baik itu janjin nya terpenuhi atau ingkar finally akan tetap terlaksanakan.

Dan didalam promise itu ada aksi yg akan kita lakukan ketika janji nya terpenuhi atau tidak, kalo terpenuhi ada aksi then artinya ketika terpenuhi lakukan aksi yg ada, kalo janji nya ingkar yg dilakukan catch. Kalo then akan menjalankan resolve kalo catch akan menjalankan reject.

Promise kita pake biasa nya ketika kita mau request ke sebuah API, contoh promise ini hanya untuk contoh aja tidak request ke API, :

let ditepati = true;

const janji1 = new Promise((*resolve*, *reject*) => {

  if (ditepati) {

    resolve("janji ditepati");

  } else {

    reject("ingkar janji");

  }

});

console.log(janji1);

Nah karena promise object kita bisa bikin pake kyword new tapi object nya khusus yaitu Promise dan object nya dia punya dua parameter kita bikin nya bisa pake callback, disitu kita gunakan arrow function. Dan didalam nya kita kasih kondisi jadi kita bisa isikan kondisinya ketika resolve kita mau lakukan apa dan ketika reject kita mau lakukan apa.

Misal kalo resolve kita mau lakukan request ke API itu bisa saja, bisa menjalankan proses yg kompleks, dan kalo reject panggil aja function callback nya reject mau melakukan apa ketika reject.

Jika mau menjalakan secara langsung bisa menggunakan console.log, tapi kalo mau menangkap resolve and reject nya kalian bisa console.log nya didalam method than dan catch, gini :

let ditepati = true;

const janji1 = new Promise((*resolve*, *reject*) => {

  if (ditepati) {

    resolve("janji ditepati");

  } else {

    reject("ingkar janji");

  }

});

janji1

  .then((*response*) => console.log(*response*))

  .catch((*response*) => console.log(*response*));

Baca nya gini janji1 ketika then maka dia akan ngambil resolve nya, kalo resolve kita ambil response nya (note response itu hanya parameter nama nya bisa apa aja bebas), berarti apapun yg dihasilkan dari fungction resolve itu kita jalankan ke arrow function lalu console.log aja.

Jadi parameter arrow function (response) nya itu menerima hasil dari object Promise, kalo hasilnya resolve dia ngejalanin then nya kalo hasil dari object Promise nya reject dia ngejalanin yg catch nya.

**Contoh 2**

Kita mau bikin janji nya supaya ga langsung di tepati nunggu dulu pake waktu gitu, kan biasanya nunggu dulu ya karena emang proses nya kompleks atau data nya banyak jadi dibuat nunggu dulu.

let ditepati = true;

const janji2 = new Promise((*resolve*, *reject*) => {

  if (ditepati) {

    setTimeout(() => {

      resolve("Janji telah ditepati setelah beberapa waktu");

    }, 2000);

  } else {

    setTimeout(() => {

      reject("janji tidak ditepati setelah beberapa waktu");

    }, 2000);

  }

});

Jadi kalo ditepati janji nya ga langsung jalan kita harus nunggu dulu 2detik kan itu pake setTimeout() kita ngejalanin function resolve() nya di dalam setTimeout(), jadi kita coba jalanin supaya keliatan Asynchronous, kita panggil gini :

console.log("mulai");

console.log(janji2);

console.log("selesai");

Nah kalo cara manggil nya gitu keliatan kalo prosesnya lagi pending tapi ga selesai” pending nya karena kita belum jalanin then nya, tapi kalo cara manggil nya gini :

console.log("mulai");

console.log(janji2.then(() => console.log(janji2)));

console.log("selesai");

Nah kita disitu jalanin Promise nya lagi, ketika promise nya lagi pending terus tambahin then didalam nya manggil lagi si promise nya nah itu pas promise nya udah selesai, jadi awal nya pending terus nunggu 2detik baru selesai.

Karena kalo saya langsung tulis begini kalian gabakal bisa liat pending nya :

console.log("mulai");

*// console.log(janji2.then(() => console.log(janji2)));*

janji2

  .then((*response*) => console.log(*response*))

  .catch((*response*) => console.log(*response*));

console.log("selesai");

kalo kita liat di console, dia hanya ada mulai, selesai, terus nunggu 2 detik baru nongol hasilnya pending nya ga keliatan.

**Finally**

Kalo kalian mau bisa tambahin method finally, method ini dijalankan ketika salah satu dari then atau catch ini sudah selesai dijalankan, jadi kalo Promise nya udah ga pending lagi finally dulu dijalanin baru hasilnya. Gini :

console.log("mulai");

*// console.log(janji2.then(() => console.log(janji2)));*

janji2

  .finally(() => console.log("Selesai menunggu"))

  .then((*response*) => console.log(*response*))

  .catch((*response*) => console.log(*response*));

console.log("selesai");

Jadi kalo udah selesai pending nya yg pertama keluar itu finally baru hasilnya then atau catch, biasanya finally ini digunakan kalo kalian mau pake animasi loading jadi pada saat mulai jalan promise nya animasi load nya jalanin begitu finally matiin animasi nya.

**Promise All**

Ini digunakan ketika kalian mau menjalankan lebih dari satu Promise sekaligus, contoh nya gini pahamin kode nya ya :

const film = new Promise((*resolve*) => {

  setTimeout(() => {

    resolve([

      {

        judul: "Avengers",

        sutradara: "Sandhika Galih",

        pemeran: "Doddy, Erik",

      },

    ]);

  }, 1000);

});

const cuaca = new Promise((*resolve*) => {

  setTimeout(() => {

    resolve([

      {

        kota: "Bandung",

        temp: 26,

        kondisi: "Cerah Berawan",

      },

    ]);

  }, 500);

});

film.then((*response*) => console.log(*response*));

cuaca.then((*response*) => console.log(*response*));

Misal kalian mau konek ke 2 API yg pertama data film misalnya yg ke dua data cuaca misalnya, nah itu kita udah jalanin nya secara satu-satu ya, nah yg muncul duluan di console.log nya itu yg data cuaca karena waktu tunggu dia lebih dikit.

Nah kalo mau jalanin sekaligus kalian bisa gini :

Promise.all([film, cuaca]).then((*response*) => console.log(*response*));

Jadi kalian panggil terus didalam kurung nya kasih array untuk merepsentasikan promise-promise yg ingin kalian panggil dan baru deh kita chaining menggunakan then nya itu dan hasilnya itu berbentuk array jadi ada didalam array.

Nah kalo kalian mau hasilnya terpisah bisa pake spread operator, gini :

Promise.all([film, cuaca])

*// .then((response) => console.log(response));*

  .then((*response*) => {

    const [film, cuaca] = *response*;

    console.log(film);

    console.log(cuaca);

  });

Ini kan harusnya udah tau materinya ada di sebelum-sebelum ini mengenai spread operator.