Front End Bootcamp
PT. Mahardika Solusi Teknologi

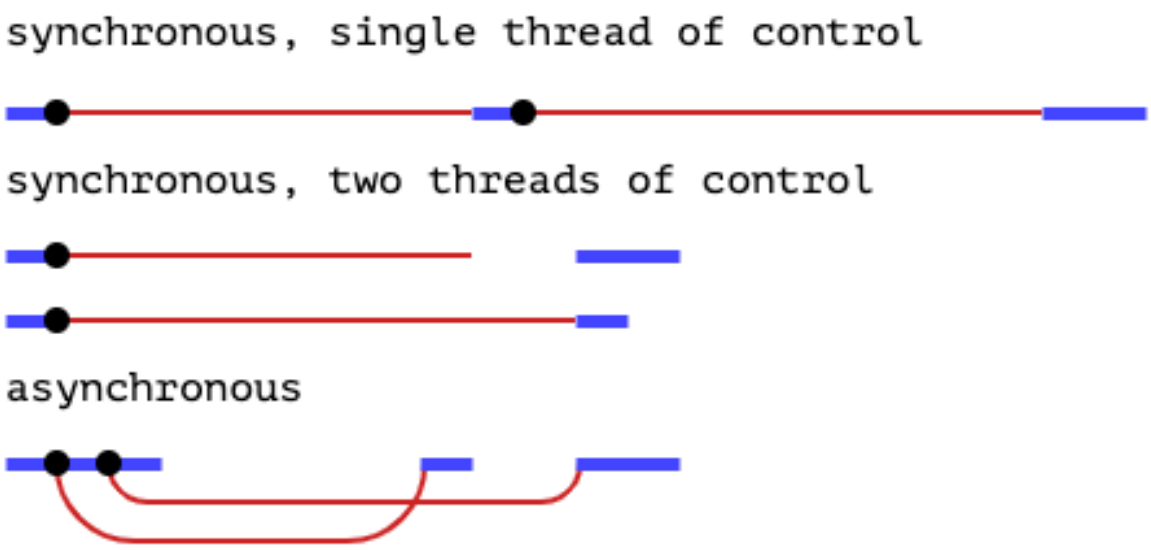
Ariel Arliyanus, S.Kom

Chapter 12 – JavaScript Asynchronous

Chapter 12
JavaScript Asynchronous

Di dalam dunia pemrograman terdapat dua cara dalam menjalankan program: Synchronous dan Asynchronous. Synchronous artinya program berjalan secara berurutan sedangkan Asynchronous artinya program berjalan bersama-sama.

Terkadang di dalam program yang kita buat terdapat suatu sintaks yang mengharuskan code pada baris tersebut untuk dijalankan terlebih dahulu sebelum menjalankan sintaks pada baris selanjutnya. Hal ini dikenal dengan istilah **blocking**. Sebaliknya **non-blocking** artinya program berjalan dengan mengeksekusi sintaks dari baris ke baris secara paralel (bersama-sama) .




```
setTimeout(function() {
  console.log("saya dijalankan belakangan")
}, 3000)

console.log("saya dijalankan pertama")
```

Jika kita jalankan program di atas, maka yang akan tampil terlebih dahulu di console adalah “saya dijalankan pertama” walaupun sintaksnya ditulis belakangan setelah function setTimeout. Function setTimeout di atas merupakan salah satu contoh function asynchronous di Javascript.

Cara untuk mengatasi Asynchronous seperti function setTimeout adalah dengan **Callback** atau dengan **Promise**.

Callback

Membuat Callback

Callback adalah function yang dipanggil ketika function lain selesai menjalankan programnya. Contoh sederhana Callback adalah pada function setTimeout di atas. function setTimeout menerima dua parameter yaitu callback dan waktu tunggu (timeout). function tersebut menjalankan timeout terlebih dahulu lalu ketika waktu yang diset sudah dilewati maka function callback akan dipanggil.

Contoh untuk membuat callback seperti berikut:

```
// Deklarasi function yang memilik callback sebagai parameter
function periksaDokter(nomerAntri, callback) {
    if(nomerAntri > 50 ) {
        callback(false)
    } else if(nomerAntri < 10) {
        callback(true)
    }
}
```

Misalkan kita ingin periksa ke dokter yang antriannya sering panjang dan memakai nomer antri melalui pemesanan online. Setelah registrasi online lalu kita melakukan pemesanan dan menunggu nomer antrian. Function di atas menerima parameter nomer antri dan sebuah callback. Dilakukan pengecekan kondisi jika nomor antriannya lebih dari 50 maka lebih baik jalan-jalan dulu daripada menunggu, tapi jika nomor antriannya kurang dari 10 tentunya kita harus standby lagi di klinik untuk dipanggil.

Menjalankan Callback

Setelah dideklarasi function yang memiliki callback, kini kita jalankan function tersebut.

```
// Menjalankan function periksaDokter yang sebelumnya sudah dideklarasi
periksaDokter(65, function(check) {
    if(check) {
        console.log("sebentar lagi giliran saya")
    } else {
        console.log("saya jalan-jalan dulu")
    }
})
```

Karena kita tidak ingin menunggu maka kita gunakan callback untuk mengecek apakah nomer antriannya masih lama atau tidak. Jadi ketika dicek ternyata nomer antriannya masih lama maka hasil return (callback) dari function periksaDokter bernilai false (check = false), sebaliknya jika sudah sebentar lagi akan diberikan callback dengan nilai true (check = true) .

Ketika menjalankan function periksaDokter, diberikan parameter pertama yaitu nomer antrian 65 dan parameter kedua adalah sebuah deklarasi function yang merupakan callback. Seperti sudah dideklarasikan di periksaDokter bahwa callback dipanggil dengan satu parameter bernilai boolean.

contoh lainnya:

terdapat fungsi callback seperti di bawah ini

```
function periksaAntrianDokter(nomerAntri, callback) {
  console.log(`sekarang antrian ke-${nomerAntri}`)
  setTimeout(function () {
    if(nomerAntri === 10 ) {
      console.log("saya masuk ruangan dokter")
      callback(0)
    } else {
      console.log("saya masih menunggu")
      callback(nomerAntri+1)
    }
  }, 1000)
}
```

cara menggunakan callback dengan setTimeout di dalamnya adalah seperti ini:

```
var nomorAntriSekarang = 7

// contoh menggunakan callback hell
periksaAntrianDokter(nomorAntriSekarang, function(nomorAntriBaru){
  periksaAntrianDokter(nomorAntriBaru, function(nomorAntriBaru1){
    periksaAntrianDokter(nomorAntriBaru1, function(nomorAntriBaru2){
      periksaAntrianDokter(nomorAntriBaru2, function(nomorAntriBaru3){
        return nomorAntriBaru3
      })
    })
  })
});

// atau dengan
 menggunakan function recursive
function execute(nomorAntri){
  periksaAntrianDokter(nomorAntri, function(nomorAntriBaru){
    if (nomorAntriBaru !== 0){
      execute(nomorAntriBaru)
    }
  })
}

execute(nomorAntriSekarang)
```

Promise

Sesuai dengan namanya, Promise berarti janji. Seperti janji yang biasanya memakan waktu dan janji bisa ditepati (resolve) atau diingkari (reject). Misalkan pada contoh di bawah ini seorang anak dijanjikan mendapatkan sebuah handphone baru oleh Ibunya. Jika Ibunya sedang bahagia maka janji sebuah handphone baru ditepati, selain itu jika ibunya sedang marah maka janji diingkari.

Membuat Promise

Cara membuat promise adalah dengan menginstance sebuah class Promise. class Promise tersebut sudah disediakan di Javascript. Parameter yang dikirim ketika melakukan instance class Promise yaitu sebuah function yang menerima dua parameter yaitu resolve dan reject.

```
var isMomHappy = false;

// Promise
var willIGetNewPhone = new Promise(
    function (resolve, reject) {
        if (isMomHappy) {
            var phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone); // fulfilled atau janji dipenuhi
        } else {
            var reason = new Error('mom is not happy');
            reject(reason); // reject (ingkar)
        }

    }
);
```

Menjalankan Promise

Menjalankan promise seperti kita menagih janji yang sudah dibuat.

```
function askMom() {
    willIGetNewPhone
        .then(function (fulfilled) {
            // yay, you got a new phone
            console.log(fulfilled);
         // output: { brand: 'Samsung', color: 'black' }
        })
        .catch(function (error) {
            // oops, mom don't buy it
            console.log(error.message);
         // output: 'mom is not happy'
        });
}
```

```
// Tanya Mom untuk menagih janji
askMom()
```

Untuk menagih janji dibuat sebuah function dengan nama askMom yang isinya adalah menagih janji willIGetNewPhone. Ketika anak menagih janji menggunakan function askMom() maka promise willIGetNewPhone dipanggil dan terdapat dua methods yaitu `then` dan `catch` . Method .then() dan .catch() keduanya menerima parameter function. Resolve yang dijalankan di pendeklarasian promise akan mengirim handphone baru dan ditangkap di method `.then()`. Sedangkan reject pada pendeklarasian promise akan mengirim pesan error atau alasan kenapa janji diingkari dan ditangkap di method `.catch()`.

contoh lainnya:

terdapat dua fungsi promise seperti di bawah ini:

```
// promise periksa antrian dokter
function periksaAntrianDokterPromise(nomerAntri) {
  console.log(`sekarang antrian ke-${nomerAntri}`)
  return new Promise( function (resolve, reject){
    setTimeout(function () {
      if(nomerAntri === 10 ) {
        console.log("saya masuk ruangan dokter")
        reject(0)
      } else {
        console.log("saya masih menunggu")
        resolve(nomerAntri+1)
      }
    }, 1000)
  })
}


// promise periksa data pasien
function periksaDataPasien(nomorIdPasien) {
  var dataPasien = [
    {id: 1, nama: "John", jenisKelamin: "Laki-laki"},
    {id: 2, nama: "Michael", jenisKelamin: "Laki-laki"},
    {id: 3, nama: "Sarah", jenisKelamin: "Perempuan"},
    {id: 4, nama: "Frank", jenisKelamin: "Laki-laki"}
  ]
  return new Promise( function (resolve, reject){
    var pasien = dataPasien.find(x=> x.id === nomorIdPasien)
    if (pasien === undefined){
      reject("data pasien tidak ada")
    }else{
      resolve(pasien)
    }
  })
}
```

cara menggunakan promisenya adalah seperti ini:

```
// untuk promise periksa antrian dokter
var nomorAntriSekarang = 7
function execute(nomorAntri){
 periksaAntrianDokterPromise(nomorAntri).then(function(nomorAntriBaru){
    if (nomorAntriBaru !== 0){
      execute(nomorAntriBaru)
    }
  }).catch(function(err){
    console.log(err)
  })
}

execute(nomorAntriSekarang)


// untuk promise periksa data pasien
periksaDataPasien(5).then(function(data){
  console.log(data)
}).catch(function(err){
  console.log(err)
})
```

Async/await

1. Apa itu async/await ?

Async/await adalah fitur yang hadir sejak ES2017. Fitur ini mempermudah kita dalam menangani proses *asynchronous.*

Ada 2 kata kunci disini yaitu ***async*** dan ***await****, m*ari kita lihat contohnya :

jika kita memiliki promise seperti ini:

```
function doAsync() {
  return new Promise( function (resolve, reject){
    var check = true
    if (check){
      resolve("berhasil")
    }else{
      reject("gagal")
    }
  })
}
```

maka cara menggunakan nya pada async/await adalah seperti ini:

```
async function hello(){
   var result = await doAsync()
   console.log(result)
}

hello()
```

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 proses *doAsync( )* selesai . *await* juga bisa digunakan berkali-kali di dalam function

sekarang kita coba gunakan promise pada async/await :

```
// promise periksa data pasien
function periksaDataPasien(nomorIdPasien) {
  var dataPasien = [
    {id: 1, nama: "John", jenisKelamin: "Laki-laki"},
    {id: 2, nama: "Michael", jenisKelamin: "Laki-laki"},
    {id: 3, nama: "Sarah", jenisKelamin: "Perempuan"},
    {id: 4, nama: "Frank", jenisKelamin: "Laki-laki"}
  ]
  return new Promise( function (resolve, reject){
    var pasien = dataPasien.find(x=> x.id === nomorIdPasien)
    if (pasien === undefined){
      reject("data pasien tidak ada")
    }else{
      resolve(pasien)
    }
  })
}
```

jika promise diatas kita gunakan dalam async/await maka akan menjadi seperti di bawah ini:

```
async function periksaPasien(){
  const dataJohn = await periksaDataPasien(1)
  console.log(dataJohn)
}
```

perbedaan saat kita menggunakan then dan catch disini kita bisa memasukkan hasil promise nya ke dalam variabel

2. Error Handling

ketika menggunakan promise maka pasangan dari then adalah catch yang di mana catch itu adalah error handling dari promise, tapi bagaiamana dengan async/await, async/await menggunakan try dan catch untuk error handlingnya seperti contoh di bawah ini:

```
async function hello(){
  try {
   var result = await doAsync()
   console.log(result)
  } catch(err){
    console.log(err)
  }
}

hello()
```

3. 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.

```
const firstPromise = () =>{
  return new Promise ((resolve,reject) =>{
    setTimeout(()=>{
      resolve("first promise")
    },1000)
  })
}

const secondPromise = () =>{
  return new Promise ((resolve,reject) =>{
    setTimeout(()=>{
      resolve("second promise")
    },1000)
  })
}

const thirdPromise = () =>{
  return new Promise ((resolve,reject) =>{
    setTimeout(()=>{
      resolve("third promise")
    },1000)
  })
}

//ini paralel selama satu detik
const asyncParalel = async () =>{
  firstPromise().then(res=>{
    console.log(res)
  })
  secondPromise().then(res=>{
    console.log(res)
  })
  thirdPromise().then(res=>{
    console.log(res)
  })

}

// ini berseri selama tiga detik
const asyncSerial = async () =>{
  let a = await firstPromise()
  console.log(a)
  let b = await secondPromise()
  console.log(b)
  let c = await thirdPromise()
  console.log(c)
}

asyncParalel()
asyncSerial()
```

jika kita jalankan kode diatas maka terlihat bahwa asyncParalel diatas menjalankan semua promisenya serentak sedangkan asyncSerial menjalankan kodenya satu persatu, maka dari itu permasalahan asynchronous dapat di selesaikan dengan ini

Tugas

Soal 1

Kita mempunyai tumpukan buku untuk dibaca. Setiap buku memiliki waktu yang dibutuhkan untuk menghabiskan buku tersebut. Sudah disediakan function readBooks yang menerima tiga parameter: waktu, buku yang dibaca, dan sebuah callback. Salin code berikut ke dalam sebuah file bernama `callback.js` .

```
// di callback.js
function readBooks(time, book, callback ) {
    console.log("saya membaca " + book.name )
    setTimeout(function(){
        let sisaWaktu = 0
        if(time >= book.timeSpent) {
            sisaWaktu = time - book.timeSpent
            console.log("saya sudah membaca " + book.name + ", sisa waktu saya " +
sisaWaktu)
            callback(sisaWaktu) //menjalankan function callback
        } else {
            console.log('waktu saya habis')
            callback(time)
        }
    }, book.timeSpent)
}

module.exports = readBooks
```

Masih satu folder dengan file `callback.js`, buatlah sebuah file dengan nama `index.js` lalu tuliskan code seperti berikut.

```
// di index.js
var readBooks = require('./callback.js')

var books = [
    {name: 'LOTR', timeSpent: 3000},
    {name: 'Fidas', timeSpent: 2000},
    {name: 'Kalkulus', timeSpent: 4000},
    {name: 'komik', timeSpent: 1000}
]

// Tulis code untuk memanggil function readBooks di sini
```

lanjutkan code pada `index.js` untuk memanggil function readBooks. Buku yang akan dihabiskan adalah buku-buku di dalam array `books`. Pertama function readBooks menerima input waktu yang dimiliki yaitu 10000 ms (10 detik) dan books pada indeks ke-0. Setelah mendapatkan callback sisa waktu yang dikirim lewat callback, sisa waktu tersebut dipakai untuk membaca buku pada indeks ke-1. Begitu seterusnya sampai waktu habis atau semua buku sudah terbaca.

Soal 2

Setelah no.1 berhasil, implementasikan function readBooks yang menggunakan callback di atas namun sekarang menggunakan Promise. Buatlah sebuah file dengan nama `promise.js`. Tulislah sebuah function dengan nama readBooksPromise yang me-return sebuah promise seperti berikut:

```
// di file promise.js
function readBooksPromise (time, book) {
  console.log("saya mulai membaca " + book.name )
  return new Promise( function (resolve, reject){
    setTimeout(function(){
      let sisaWaktu = time - book.timeSpent
      if(sisaWaktu >= 0 ){
          console.log("saya sudah selesai membaca " + book.name + ", sisa waktu saya "
+ sisaWaktu)
          resolve(sisaWaktu)
      } else {
          console.log("saya sudah tidak punya waktu untuk baca "+ book.name)
          reject(sisaWaktu)
      }
    }, book.timeSpent)
  })
}

module.exports = readBooksPromise
```

Masih di folder yang sama dengan promise.js, buatlah sebuah file dengan nama `index2.js`. Tuliskan code sebagai berikut

```
var readBooksPromise = require('./promise.js')

var books = [
    {name: 'LOTR', timeSpent: 3000},
    {name: 'Fidas', timeSpent: 2000},
    {name: 'Kalkulus', timeSpent: 4000}
]

// Lanjutkan code untuk menjalankan function readBooksPromise
```

Lakukan hal yang sama dengan soal no.1, habiskan waktu selama 10000 ms (10 detik) untuk membaca semua buku dalam array books.

Soal 3

Buatlah sebuah file dengan nama `promise2.js`. Tulislah sebuah function dengan nama filterBookPromise yang me-return sebuah promise seperti berikut:

```
function filterBooksPromise(colorful, amountOfPage){
  return new Promise(function(resolve, reject){
    var books=[
        {name: "shinchan", totalPage: 50, isColorful: true},
        {name: "Kalkulus", totalPage: 250, isColorful: false},
        {name: "doraemon", totalPage: 40, isColorful: true},
        {name: "algoritma", totalPage: 250, isColorful: false},
    ]
    if (amountOfPage >= 40) {
        resolve(books.filter(x=> x.totalPage >= amountOfPage && x.isColorful
== colorful));
      } else {
        var reason= new Error("Maaf buku di bawah 40 halaman tidak tersedia")
        reject(reason);
      }
  });
}

module.exports = filterBooksPromise
```

Masih di folder yang sama dengan promise2.js, buatlah sebuah file dengan nama `index3.js`. Tuliskan code sebagai berikut:

```
var filterBooksPromise = require('./promise2.js')

// Lanjutkan code untuk menjalankan function filterBookPromise
```

gunakan promise dengan kondisi seperti di bawah ini:

- bukunya berwarna dan jumlah halamannya 40
- bukunya tidak berwarna dan jumlah halamannya 250 (**gunakan async/await untuk kondisi ini**)
- bukunya berwarna dan jumlah halamannya 30 (**gunakan async/await untuk kondisi ini**)

berikut ini contoh output soal 3:

```
Soal 1
saya membaca LOTR
saya sudah membaca LOTR, sisa waktu saya 7000
saya membaca Fidas
saya sudah membaca Fidas, sisa waktu saya 5000
saya membaca Kalkulus
saya sudah membaca Kalkulus, sisa waktu saya 1000
saya membaca komik
saya sudah membaca komik, sisa waktu saya 0
```

```
Soal 2
saya mulai membaca LOTR
saya sudah selesai membaca LOTR, sisa waktu saya 7000
saya mulai membaca Fidas
saya sudah selesai membaca Fidas, sisa waktu saya 5000
saya mulai membaca Kalkulus
saya sudah selesai membaca Kalkulus, sisa waktu saya 1000
```

```
[
  { name: 'shinchan', totalPage: 50, isColorful: true },
  { name: 'doraemon', totalPage: 40, isColorful: true }
]
[
  { name: 'Kalkulus', totalPage: 250, isColorful: false },
  { name: 'algoritma', totalPage: 250, isColorful: false }
]
Maaf buku di bawah 40 halaman tidak tersedia
```