Coding Subuh #20

Promise

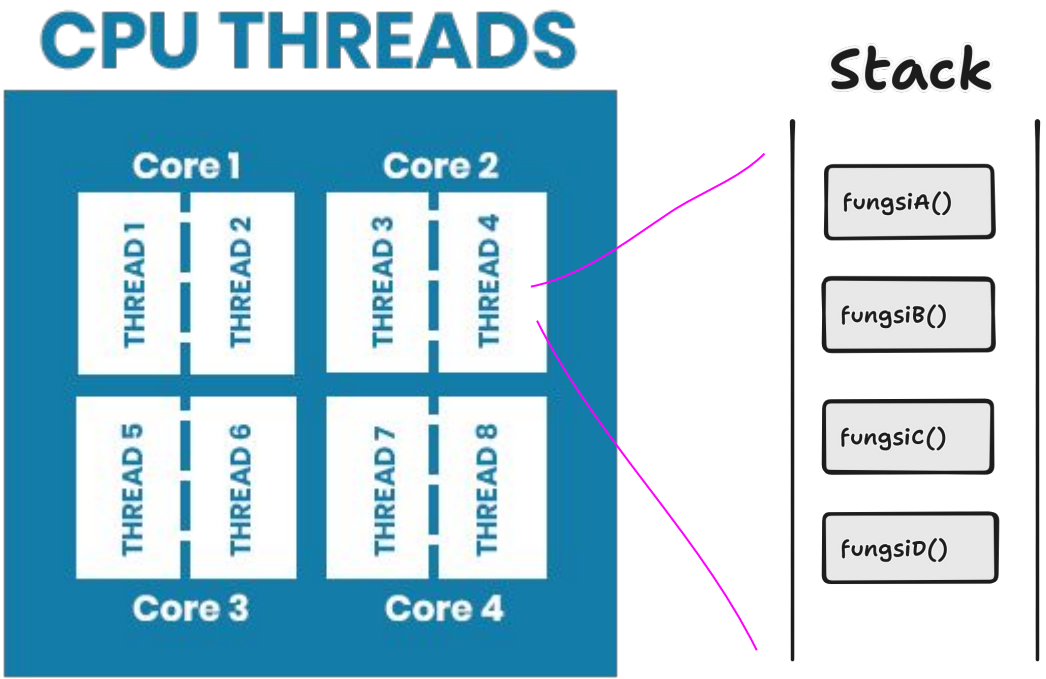
Ahmad Muhardian

Agenda kita..

- Memahami Single-Thread Javascript
- Fungsi Callback (again)
- Promise
  - Then..catch
  - Async/Await pattern
- Latihan

Single-Thread

Javascript itu Single-Thread



Fungsi-fungsi javascript dijalankan dalam satu thread CPU, tapi kok tidak saling block atau menunggu? 🤔

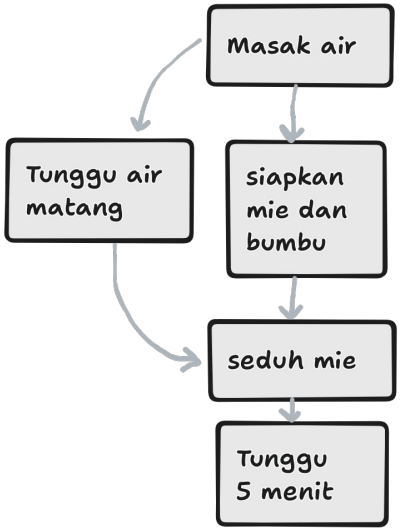
Synchronous vs Asynchronous

Seduh Mie Instan secara

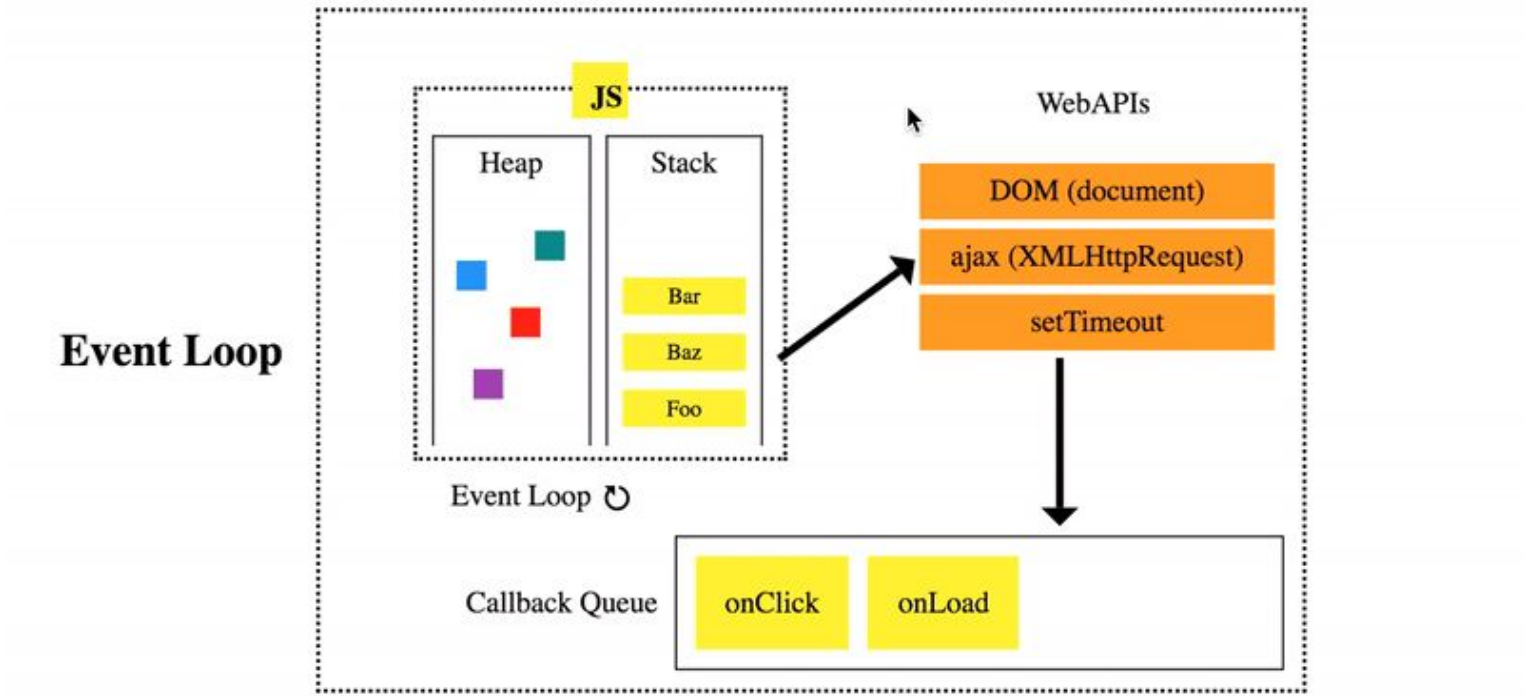
Synchronous

Seduh Mie Instan secara

Asynchronous



Cara Javascript Handle Proses Asynchronous



Callback again

Sebelum itu.. Kita bahas dulu Callback

- Callback adalah **fungsi yang dipanggil kembali setelah sebuah fungsi dikerjakan**
- Callback di-inject atau dimasukan melalui parameter
- Callback biasanya dipakai untuk handle event listener, handle proses asynchronous dan AJAX.

```
function doPayment(amount, callback){
    if (amount < 0 || isNaN(amount) || amount > myBalance){
        throw new Error("Saldo tidak cukup");
    }

    // lakukan pembayaran
    myBalance -= amount;

    // panggil fungsi callback
    callback();
}


doPayment(10000, function(){
    alert("Pembayaran Berhasil");
});
```

Callback dipakai di Event Listener

```
// mendeteksi event load (request sukes dan selesai)
xhr.addEventListener("load", function () {

  // ubah JSON ke Object Javascript
  const products = JSON.parse(xhr.responseText);

  console.log(products);

  // show product ke HTML
  showProducts(products);
});
```

Callback di Proses Asynchronous

- Callback akan dipanggil ketika proses asynchronous selesai dikerjakan supaya hasil dari proses tersebut bisa kita proses lebih lanjut.
- Pada contoh di samping, **`tugas2()`** akan ditunda sampai 1 detik. **`tugas3()`** tidak akan menunggu **`tugas2()`** selesai.

```
function tugas1(){
    console.log("Tugas 1 done");
}

function tugas2(callback){
    // delay tugas 1 detik
    setTimeout(function(){
        console.log("Tugas 2 done");
        const hasil = "Coding Subuh";
        callback(hasil);
    }, 1000);
}

function tugas3(){
    console.log("Tugas 3 done");
}

// pemanggilan fungsi
tugas1();
tugas2(hasil => console.log(hasil));
tugas3();
```

Callback Hell

```
getArticles(20, (user) => {
    console.log("Fetch articles", user);
    getUserData(user.username, (name) => {
        console.log(name);
        getAddress(name, (item) => {
            console.log(item);
            // this goes on and on...
        })
    })
})
```

Callback hell terjadi ketika ada callback di dalam callback dan ada callback lagi di dalam callback.. Dan seterusnya. Callback hell membuat kode jadi sulit dipahami

Inilah alasan mengapa kita butuh Promise.

Promise

Apa itu Promise?

- Promise adalah object yang mewakili **hasil akhir** (keberhasilan & kegagalan) dari proses asynchronous.
- Promise punya tiga state:
  - *Pending* -> proses async sedang dikerjakan
  - *Fulfilled* -> proses async selesai dan berhasil dikerjakan
  - *Rejected* -> proses async gagal

Cara Buat Object Promise

Parameter callback dijalankan saat berhasil

Parameter callback dijalankan saat gagal

```
var janji = new Promise((resolve, reject) => {
    // body fungsi kode promise
})
```

Contoh Promise

```
const getTHR = new Promise((resolve, reject) => {
    const lamaKerja = 30;
    if(lamaKerja < 30){
        reject("Lama kerja harus udah 1 bulan untuk dapat THR");
    }
    resolve({thr: 1000000});
})

const janjianBukber = new Promise((ok, tolak) => {
    const isRaining = true;
    if(isRaining){
        tolak("Ada Hujan, gak jadi!");
    }
    ok("OK, aku ikut bukber");
})
```

```
// mengakses data dari promise
janjianBukber.then((hasil) => {
    console.log(hasil);
}).catch(reason => {
    console.log(reason);
});
```

Mengambil data dari Promise

1. Menggunakan **then..catch** atau **then..catch..finally**
2. Menggunakan **async/await**

```
// mengakses data dari promise
janjianBukber.then((hasil) => {
    console.log(hasil);
}).catch(reason => {
    console.log(reason);
});
```

```
async function showTHR(){
    const hasil = await getTHR;
    console.log(hasil.thr);
}

showTHR();
```

Mengambil data dari Promise Berantai (Chained Promise)

```
myPromise
  .then((value) => `${value} and bar`)
  .then((value) => `${value} and bar again`)
  .then((value) => `${value} and again`)
  .then((value) => `${value} and again`)
  .then((value) => {
    console.log(value);
  })
  .catch((err) => {
    console.error(err);
  });
```

Di dalam ***callback then***, kita bisa return promise dan akan diporses di then berikutnya

Cara ini Lebih terlihat rapi dibandingkan Callback Hell

Static Method di Promise()

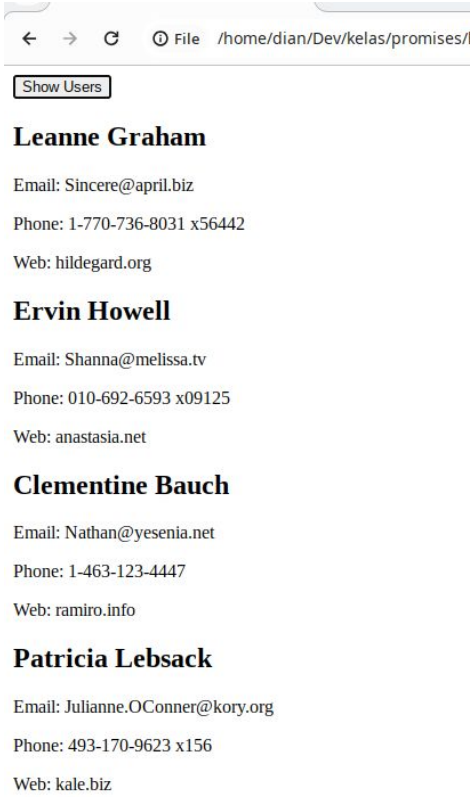
- `Promise.all()` -> untuk menjalankan semua promise di dalam array atau iterable dan menggabungnya jadi satu promise. Semua promise harus fullfiled (berhasil)
- `Promise.any()` -> sama seperti `Promise.all()` tapi dia akan tetap menganggap berhasil walaupun ada cuma ada 1 promise yang berhasil.
- `Promise.race()` -> mengambil satu promise yang paling awal berhasil saja.

Latihan 💪

Latihan: AJAX Upload dengan Promise

- Buat form upload file dan kirim filenya dengan AJAX ke server: https://codesandbox.io/p/sandbox/coding-subuh-ajax-server-33zfhf

Latihan: Promise Fetch Data



Selamat

Kita sudah selesai hari ini..

Resources & Referensi

- XMLHttpRequest: responseType property - Web APIs | MDN
- Using the Fetch API
- https://www.petanikode.com/javascript-ajax/
- Belajar Nodejs #12: Menggunakan Database SQLite pada Nodejs