ASYNCHROUNOUS JAVASCRIPT

PROMISE & FETCH

PENGERTIAN PROMISE

- Promise adalah object (container) yang ditujukan sebagai penampung (placeholder) sebuah nilai yang belum ada, tapi ditunggu untuk tampil
- Promise menggunakan callback yang diregistrasi, hasil eksekusi adalah satu dari 2 kondisi, **berhasil atau gagal**
- Bila berhasil, maka callback “resolve( )” akan dijalankan
- Bila gagal, maka callback “reject()” akan dijalankan

CONTOH SIMULASI PROMISE

```
let promise = new Promise(function(resolve, reject) {
    //do stuff
    let isSuccessful = true;
    setTimeout( function( ) { //asynchronously setelah 2 detik
      if (isSuccessful) { //Jika berhasil
          resolve('berhasil!');
      }
      else{                    //Terjadi error
            reject(Error("terjadi error"))
        }
    }, 2000);
});
```

THEN - CATCH

```
promise.then(function(val){
    //val merepresentasikan nilai keberhasilan
    console.log(val);      // logs "berhasil"
    })
.catch(function(val){
    //val merepresentasikan nilai penolakan //(rejected)
    console.log(val);  //tidak terjadi bila sukses
});
```

TRANSFORMASI NILAI

- Nilai balik Promise dapat ditransformasikan dengan memanggil instruksi return dalam blok *then()*
- metode then() selanjutnya memberikan nilai balik sebuah Promise baru atau bentuk obyek lainnya dari hasil yang ditransformasikan tersebut

CONTOH

```
let promise1 = Promise.resolve("INIX");
let promise2 = promise1.then(function(result) {
    console.log(result) //logs "test promise"
    return result + "INDO"
});

promise2.then(function(result){
    console.log(result);
});
```

CONTOH VERSI 2

```
Promise.resolve("INIX")
    .then(function(result) {
        console.log(result);
        return result + "INDO"
    })
    .then(function(result){
        console.log(result);
});
```

CONTOH LAIN

```
let promise = Promise.resolve([1,2,3,4]);
promise.then(function(result) {
   console.log(result) //logs [1,2,3,4]
   return result.map(x => x * x);

   }).then(function(result2){
   console.log(result2) //logs [1,4,9,16]
   return result2.filter( x => x > 10); //filters out

         //elements that are not larger than 10
   }).then(function(result3){
   console.log(result3) //logs [16]
   return result3.toString() + "!!"; //converts result3 to a

         //string and adds "!!"

}).then(function(result4){
console.log(result4) //logs "16!!"
return result4; //returns a promise with "16!!" as the

         //fulfillment value
}).catch(function(error){

         console.log(error)

});
```

RANGKUMAN

```
let p = Promise.resolve (argumen) ;

p.then (function(param) {
    ...
    return hasil;
})
.then ( function(argumen) { return x; }
```

PROMISE.ALL

- Menggunakan array yang terdiri atas Promise sebagai parameter dan menunggu sampai semuanya sukses (resolve)
- Jika salah satu Promise ditolak (reject), maka metode Promise.all() akan mengembalikan *rejected* Promise sebagai nilai balik

CONTOH

```
var promise1 = Promise.resolve('test promise');
var promise2 = Promise.resolve({nama:"Dastan",jabatan:"manager"})
var promise3 = Promise.reject('failure'); //rejected promise


Promise.all([promise1,promise2,promise3])
  .then(function(result) {
     console.log(result) // tidak muncul, promise3 ditolak
}).catch(function(error){
    console.log(error)  //logs 'failure.'
});
```

PROMISE.RACE

- Promise.race menggunakan daftar promises dan program selesai, bila ada salah satu promise yang didalam daftar tersebut telah selesai
- Promise.race() digunakan untuk memilih sumber yang paling cepat, jika ada dua atau lebih sumber yang ada

CONTOH

```
var promise1 = new Promise(function(resolve,reject){
    setTimeout(function(){
        resolve("finished in two seconds");
    },2000) //return setelah 2 detik
});

var promise2 = new Promise(function(resolve,reject){
    setTimeout(function(){
        resolve("Selesai dalam 3 detik");
    },3000) //return setelah 3 detik
});


// Mana yang paling cepat?
Promise.race([promise1,promise2])
  .then(function(result) {
     console.log(result)
}).catch(function(error){
     console.log(error)
});
```

FETCH

GET REQUEST & RESPONSE

- **GET Request**: sebuah permintaan data di internet. Request dikiri dari client ke server
- **Response**: Sebuah jawaban dari server atas request dari client. Response dikirim dari server ke client. Response umunya menyertakan data dala format tertentu, mislanya format teks, json, atau blob.
- Hasil response (data) tersebut kemudian ditampilkan di web

AJAX CALL DENGAN FETCH API

- Fetch adalah API baru menggantikan XHR
- Fetch API berbasis Obyek Promise

FUNGSI FETCH

- Metode **fetch()** API merupakan interface untuk membuat request ke Server melalui jaringan
- Fungsi fetch() ditargetkan sebagai pengganti API XMLHttpRequest
- Metode fetch() didefinisikan pada obyek *window* , request dilakukan dari browser
- Metode ini memberikan **Promise** sebagai nilai balik

SYNTAX FETCH

```
fetch('/myfile.txt);
```

- Fetch memberikan nilai balik berupa Obyek Promise
- Secara asynchonous fetch mengirim request
- Object promise memberikan nilai .then bila positif, dan error (catch) bila gagal

CONTOH FETCH

```
<html>
<head>
<script>
function logFetch(url) {
  return fetch(url)
    .then(response => response.text())
    .then(text => {
      console.log(text);
      let myid= document.getElementById("myid");
      myid.contentText= text;
    })
    .catch(err => {
      console.error('fetch failed', err);
    });
}
</script>
</head>
<body>

Test Asycn

<p id="myid"> </p>
<button onClick="logFetch('/myfile.txt');">Fetch This</button>
</body>
</html>
```

ASYNC WAIT - FETCH

```
<html>
<head>
<script>
async function logFetch2(url) {
  try {
    const response = await fetch(url);
    let teks= await response.text();
    console.log(teks);
    document.getElementById("myid2").contentText= teks;
  }
  catch (err) {
    console.log('fetch failed', err);
  }
}
</script>
</head>
<body>

Test Asycn

<p id="myid2"> </p>
<button onClick="logFetch('/myfile.txt');">Fetch dengan Async Await</button>
</body>
</html>
```

CONTOH FETCH DENGAN JSON

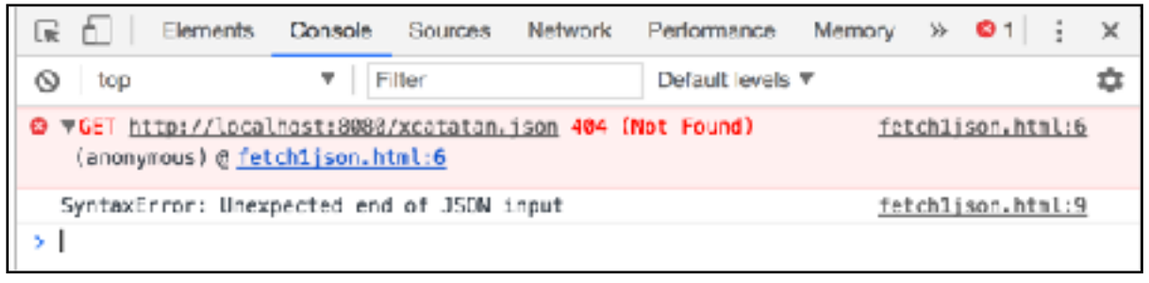
```
var URL="catatan.json";
fetch(URL)
.then(function(response){
    return response.json()
})
.then(function(text){
    console.log(text);
    . . .
})
.catch(function(err){
    console.log(err);
});
```

CONTOH FETCH DENGAN JSON

```
var URL="catatan.json";
fetch(URL)
.then(function(response){
    return response.json()
})
.then(function(text){
    console.log(text);
    document.write(text);
})
.catch(function(err){
    console.log(err);
});
```

```
var URL="catatan.json";
fetch(URL)
.then((response)=> response.json())
.then((rsp) => console.log(rsp.judul))
.catch((err) => console.log(err));
```

SIMULASI ERROR




```
Simulasi Error: URL="xcatatan.json"; // non existence
```

Ekstraksi Data dari Obyek Response

Metode pada Obyek Response untuk mengambil data (fetching):

json()	extract json object	// response.json()
text()	extract text string	// response.text()
blob()	extract file-like object	// response.blob()

Mengubah Metode HTTP

- Metode default HTTP untuk request Fetch adaah GET.
- Mengubah metode dapat dilakukan melalui parameter/ opsi.

```
fetch('http://localhost/myfile,json', {method: ‘POST’} );
```