

Fundamen PWA Aplikasi Offline



Obyektif:

- Membuat Aplikasi tetap berjalan pada saat Offline

INIXINDO

Daftar Isi

Aplikasi Offline	1
Registrasi Service-Worker	2
Persiapan Cache	5
Lab: Intersepsi Jaringan.....	7
Solusi tugas MWS - Aplikasi Offline	10

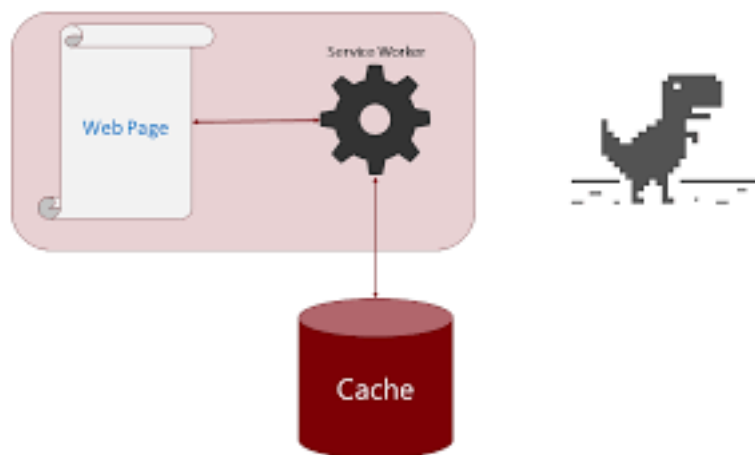
Aplikasi Offline

Aplikasi disebut "Offline" karena aplikasi tersebut dapat tetap aktif walaupun tidak tersambung dengan jaringan.

Ada 2 alasan yang menyebabkan hal itu terjadi.

Yang pertama, karena Browser mempunyai "Cache" yaitu memory yang dapat menampung file teks, images, dan lainnya, sehingga dapat mensuplai Browser dengan data yang telah disimpan di Cache lebih dahulu.

Yang kedua, Browser mempunyai Service-Worker, yaitu thread (light-weight proces) yang berada diantara jaringan, Cache, dan Browser.



Yang terjadi jika jaringan terputus, pada saat Service-Worker menerima event "fetch" **Request** dari Browser, maka Service-Worker tidak meneruskan ke jaringan, melainkan berusaha untuk memenuhi request tersebut dengan membaca Cache. Apa bila file yang diminta berada pada Cache, maka Service-Worker memberikan file tersebut sebagai "**Response**" langsung ke Browser.

Yang dibicarakan disini adalah "Request" dan "Response", dimana Service-Worker bertanggung jawab dalam menerima Request tersebut dan kemudian memberikan Response.

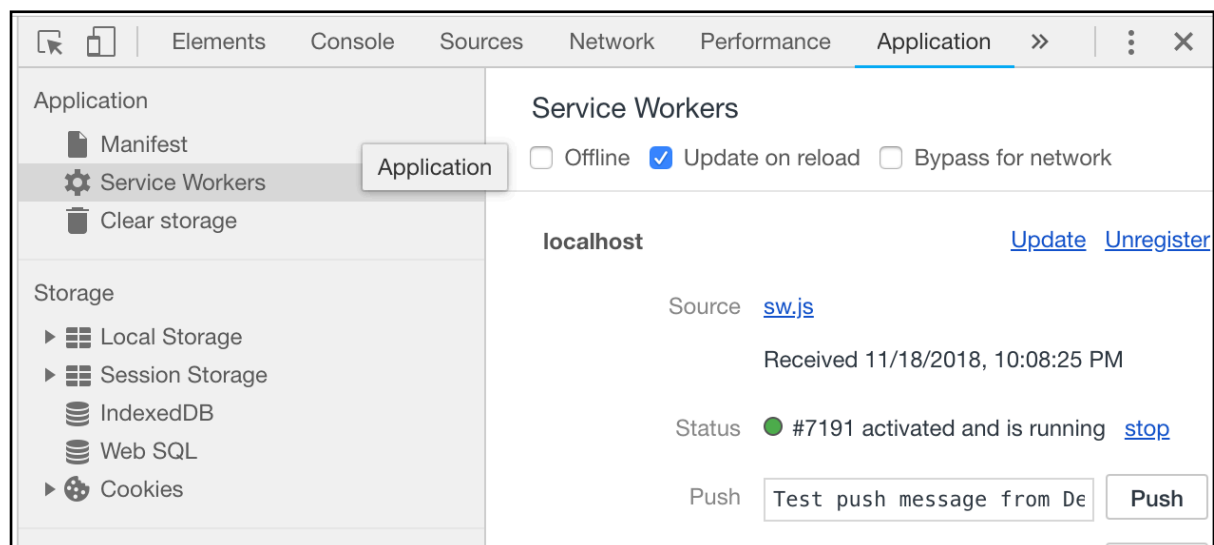
Beberapa hal yang harus dipersiapkan agar aplikasi dapat berjalan secara offline, yaitu registrasi service-worker, persiapan cache, dan intersepsi request browser.

Registrasi Service-Worker

Siapkan Service-Worker dari file index.html. File ini akan meregistrasi service-worker pada saat Browser diaktifkan.

File: index.html

```
<html>
<head>
<title>Service Worker dan Fetch</title>
</head>
<body>
Service Worker dan Fetch
<script>
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    navigator.serviceWorker.register('sw.js')
      .then(reg => console.log('SW terdaftar', reg))
      .catch(err => console.log('SW Gagal: ', err));
  });
}
</script>
</body>
</html>
```



Service-Worker dapat dilihat dari Chrome DevTools, pilih tab *Application*, kemudian *Service Workers*.

Buat file sw.js dan isi dengan program sederhana berikut:

```
// File: sw.js
self.addEventListener('fetch', event => {
  console.log( "event.request" );
})
```

```

SW terdaftar                                                                    (index):12
ServiceWorkerRegistration {scope: "http://localhost:8080/", active: ServiceWorker, ins
▼talling: null, navigationPreload: NavigationPreloadManager, sync: SyncManager, ...} ⓘ
  ▶ active: ServiceWorker {scriptURL: "http://localhost:8080/sw.js", state: "activated"...
    installing: null
  ▶ navigationPreload: NavigationPreloadManager {}
    onupdatefound: null
  ▶ paymentManager: PaymentManager {instruments: PaymentInstruments, userHint: ""}
  ▶ pushManager: PushManager {}
    scope: "http://localhost:8080/"
  ▶ sync: SyncManager {}
    updateViaCache: "imports"
  ▶ waiting: ServiceWorker {scriptURL: "http://localhost:8080/sw.js", state: "installed...
  ▶ __proto__: ServiceWorkerRegistration

```

Pada hasil console terlihat bahwa Service Worker terdaftar, dan obyek SW tersebut dapat dilihat dengan membuka icon segitiga pada ServiceWorkerRegistration.

Untuk memberikan response kepada Browser, maka `event.respondWith()` dapat digunakan dengan obyek Response sebagai parameter.

```

self.addEventListener('fetch', event => {
  event.respondWith( new Response('intersepsi jawaban!'));
})

```

Apapun request dari Browser, selalu dijawab dengan 'intersepsi jawaban' . Bila browser masih menunjukkan hasil yang lama, maka refresh browser, atau matikan, kemudian hidupkan kembali (restart).

Jawaban dapat juga diberikan dalam format html, namun Response harus diberikan header berupa `content-type: text/html` seperti contoh berikut:

Name	× Headers Preview Response Timing
xxxxxx	<div>▼ General</div> <div>Request URL: http://localhost:8080/xxxxxx</div> <div>Request Method: GET</div> <div>Status Code: 200 OK (from ServiceWorker)</div> <div>Remote Address: 127.0.0.1:8080</div> <div>Referrer Policy: no-referrer-when-downgrade</div> <div>▼ Response Headers view source</div> <div>content-type: text/html</div>

```
self.addEventListener('fetch', event => {
  event.respondWith( new Response('<b>intersepsi</b> jawaban!',
    {headers: {'content-type': 'text/html'}}));
})
```

Selanjutnya "fetch" dapat digunakan untuk langsung mengakses *request-file* dari server dan memberikan data tersebut ke browser.

```
self.addEventListener('fetch', event => {
  event.respondWith(fetch('/images/checkmark.png'));
})
```

fetch() disini melakukan normal-request ke server. File yang diminta dapat dilihat dengan properti request (event.request).

```
self.addEventListener('fetch', event => {
  event.respondWith(fetch('/images/checkmark.png'));
})
```

Lebih lengkap lagi dengan mengakses server, jika file tersebut tidak ada, maka akan diberikan jawaban status 404. Atau bila terjadi error di jaringan, maka dijawab dengan 'catch(error)'.

```
self.addEventListener('fetch', event => {
  event.respondWith(
    fetch(event.request).then ( resp => {
      if (resp.status == 404) {
        return new Response ("File tidak ditemukan");
      }
      return resp; // jawaban normal tanpa error
    })
    .catch ( error => {
      return new Response("Terjadi Error: " + error);
    })
  )
})
```

Perhatikan bahwa kasus pertama, service worker menerima jawaban positif dari web-server, tapi setelah status jawaban diperiksa, ternyata file tersebut tidak dapat diakses.

Sedangkan kasus kedua terjadi, bila ada gangguan dengan data stream, misalnya jaringan sedang offline, atau terjadi error dengan sistem. Jawaban dapat diperiksa melalui Obyek Error.

Persiapan Cache

Sebelum cache dapat digunakan, lebih dahulu dilakukan inisialisasi. Cache harus dideklarasikan dengan memberikan nama. Bila diperlukan, bisa dibentuk lebih dari satu Cache, dan setiap Cache mempunyai nama yang berbeda.

Metode yang digunakan adalah :

cache.open(NamaCache) Membuka NamaCache, jika belum ada, maka cache tersebut akan diciptakan. Nilai balik adalah obyek Promise.

```
const NAMACACHE = 'mws-v1';
cache.open(NAMACACHE)
  .then(cache => {
    .....
  })
```

cache.addAll (arrayOfFile) menambahkan semua nama file dalam array untuk disimpan di Cache.

Inisiasi cache dilakukan pada tahap "install" disaat browser pertama kali mengaktifkan service-worker.

```
self.addEventListener('install', event => {
  // persiapkan cache
  ...
})
```

Contoh:

Berikut adalah penyimpanan nama files yang akan disimpan pada Cache, hanya files yang diperlukan untuk membentuk "application shell" saja yang didaftarkan (tidak seluruh aset didaftarkan).

```
// file: sw.js
const NAMACACHE = 'mws-v1';
let filesToCache = [
  '.',
  'index.html',
  '404.html',
  'css/mygrid.css',
  'pl/add2numbers.html',
  'pl/js/add2numbers.js'
];

self.addEventListener('install', event => {
  console.log('Persiapan Cache');
  event.waitUntil(
```

```

    caches.open(NAMACACHE)
      .then(cache => {
        return cache.addAll(filesToCache);
      })
    );
  });
});

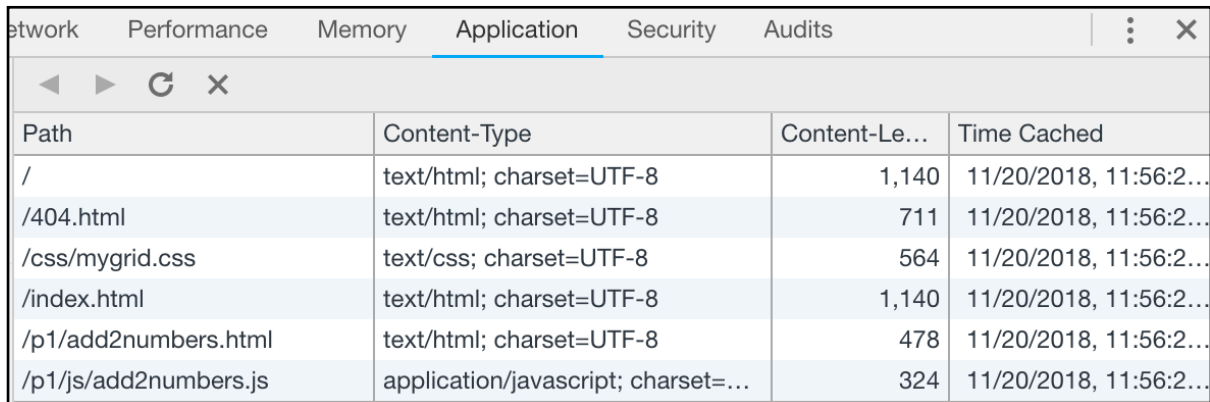
```

`event.waitUntil()` akan melakukan loop hingga semua file yang terdaftar selesai disimpan dalam cache.

Pada tahap "install", service worker mempersiapkan Cache yang terdiri atas array nama file yang akan disimpan. Perhatikan bahwa karena keterbatasan memory, sebaiknya hanya file yang terkait dengan "application shell", yaitu layar awal aplikasi, semua file yang terkait tersebut yang disimpan dalam Cache.

File lain bila dibutuhkan akan dapat disimpan ke Cache, sesuai dengan strategi aplikasi yang dibuat (cache first, atau network first).

Berikut adalah Dashboard Application -> Cache-Storage. Bila daftar file belum terlihat, ada baiknya dilakukan refresh dengan klik kanan mouse pada icon Cache Storage (refresh cache).



Path	Content-Type	Content-Le...	Time Cached
/	text/html; charset=UTF-8	1,140	11/20/2018, 11:56:2...
/404.html	text/html; charset=UTF-8	711	11/20/2018, 11:56:2...
/css/mygrid.css	text/css; charset=UTF-8	564	11/20/2018, 11:56:2...
/index.html	text/html; charset=UTF-8	1,140	11/20/2018, 11:56:2...
/p1/add2numbers.html	text/html; charset=UTF-8	478	11/20/2018, 11:56:2...
/p1/js/add2numbers.js	application/javascript; charset=...	324	11/20/2018, 11:56:2...

Pada awalnya hanya file yang terdaftar di "filesToCache" saja yang akan disimpan dalam cache. Program dapat diatur untuk file yang tidak terdaftar, akan disimpan kemudian setelah dibaca dari web-server.

Caches berisi pasangan **Request**, **Response**, sehingga bila service-worker mencari sebuah **event.request**, maka bila ada "match" dengan isi Cache, berdasarkan Request tersebut akan ditemukan **Response**.

Metode `put` dapat mengisi cache dengan request, response :

`cache.put (request, response)`

Mengisi cache juga dapat dilakukan dengan `cache.addAll()`, berdasarkan array of files seperti pada contoh sebelumnya.

Hati-hati bila salah satu dari request terjadi error, maka `cache.addAll()` tidak berfungsi sama sekali. Error yang sering terjadi adalah salah penulisan nama file atau folder di array tersebut! Metode `addAll()` ini bersifat atomic (all or nothing).

Untuk mencari request,response di Cache, digunakan 2 metode:

`cache.match(request)` : mencari request, response di Cache

`caches.match(request)`: mencari request di semua Cache yang ada

Lab: Intersepsi Jaringan

Setelah cache terpasang dengan baik, maka Service-Worker masuk ke tahap "idle", dimana Service-Worker menunggu "**Request**" dari Browser. *Request* terjadi pada saat program melakukan `fetch(nama-file)`.

Bila browser meminta file, maka ServiceWorker akan memeriksa Cache lebih dahulu. Bila ternyata ada match, maka ServiceWorker mengambil file tersebut dari Cache dan memberikannya ke Browser. Request dari Browser dapat dideteksi oleh Service-Worker melalui event *fetch*.

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then( ada_response => {
        return ada_response;
      })
      .catch(error => {
        return new Response("Waduh " + error);
      })
  );
});
```

Logic program diatas dibaca sebagai berikut, apabila file yang diminta (event.request) ada di cache (`caches.match()`) , maka jawab dengan file tersebut (`return ada_response`).

Test program secara offline. Dari DevTools, pilih tab Application, pilih network. Kemudian pilih "offline". Pastikan bahwa cache tidak dalam status "disabled".

Refresh browser, dan pilih "add2numbers". Bila berhasil mengaktifkan tersebut, maka aplikasi berjalan pada saat offline.

Namun, apa yang terjadi dengan file yang tidak ada di Cache? Code tersebut berbahaya, karena untuk setiap fetch harus ada response dari cache, bila tidak ada response, maka terjadi error. Walaupun kondisi ditukar dari offline menjadi online, juga tidak menolong, karena request tetap selalu diarahkan ke cache dan tidak ke jaringan.

Untuk menghindari hal buat program yang mengakses jaringan, jika jawaban dari cache kosong.

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then( ada_response => {
        if (ada_response) {
          return ada_response;
        }
        // tidak ada response, ambil ke jaringan
        else {
          return fetch(event.request)
        }
      })
      .catch(error => {
        return new Response("Waduh " + error);
      })
  );
});
```

fetch(event.request) yang baru dimasukkan diambil dari jaringan, tapi tidak disimpan didalam cache. Sehingga untuk transaksi offline berikutnya, request tersebut akan melempar eksepsi, data tidak ditemukan di cache.

Untuk memasukkan ke cache diperlukan coding sebagai berikut:

```
return fetch(event.request)
  .then( jawaban => {
    // Periksa jika jawaban itu tidak kosong
    if (!jawaban.ok) {
      throw Error( jawaban.statusText);
    }
    // tulis jawaban di cache dan juga berikan jawaban
    // ke browser
    return caches.open(NAMACACHE)
      .then ( cache => {
```

```

        cache.put (event.request, jawaban.clone());
        return jawaban;
    }
})

```

fetch(event.request) , dengan fungsi ini ServiceWorker mengakses web-server dengan alamat yang ada pada parameter url . Metode ini mengembalikan nilai balik berupa **Promise** . Apabila jawabannya adalah negatif, maka proses diteruskan di **.catch()**.

Jika jawaban positif dari Server belum tentu berisi file, bisa terjadi misalnya ijin akses yang tidak sesuai, atau file tidak dapat dibaca, maka server menjawab dengan **jawaban.ok==true** atau **response.ok == false** .

jawaban.ok menyatakan bahwa file tersebut bisa dikirim (deliver), sedangkan **jawaban.ok==false** menyatakan bahwa file tersebut tidak ada.

Asumsi bahwa **jawaban.ok == true**, maka ServiceWorker membuka Cache dan mencari nama file tersebut.

Fungsi **caches.open()** juga memberikan nilai balik Promise.

Bila ternyata ada di Cache, maka response tersebut diberikan ke Browser dengan **return(response)**.

Bersamaan dengan itu, karena response hanya dapat dikonsumsi satu kali, dibuat **response.clone()** yang kemudian ditulis ke Cache dengan menggunakan **cache.put()**.

```

return caches.open(NAMACACHE)
    .then(function(cache) {
        cache.put(url, response.clone());
        return response;
    });

```

Rangkuman:

fetch(url): Mengakses jaringan untuk mengambil file yang ditulis dalam url.

response.ok: true atau false, bila true maka file tersebut dikirim dari server

caches.open: Membuka cache (bila cache belum ada, cache dibuat)

caches.match: Mencari file yang match dengan parameter url

caches.put: Mengisi file kedalam Cache

Catatan: Untuk peta wisata yang menggunakan leaflet disarankan untuk tidak dibuat offline, karena javascript dari leaflet tidak mendukung penggunaan mode offline tersebut. Oleh karena itu , aplikasi offline cukup didemonstrasikan melalui aplikasi **add2numbers** .

Solusi tugas MWS - Aplikasi Offline

Sesuaikan dengan program aktual Anda, nama files dan nama folder. Solusi berikut tidak mencakup semua data, tapi cukup memenuhi syarat dari tugas yang diberikan. Aplikasi offline ini hanya mengakses file yang sebelumnya sudah tersimpan di Cache.

File: sw.js

```
const NAMACACHE = 'mws-v1';
const filesToCache = [
  '.',
  'index.html',
  '404.html',
  'css/mygrid.css',
  'pl/add2numbers.html',
  'pl/js/add2numbers.js'
];

self.addEventListener('install', event => {
  console.log('Persiapan Cache');
  event.waitUntil(
    caches.open(NAMACACHE)
      .then(cache => {
        return cache.addAll(filesToCache);
      })
  );
});

self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then( ada_response => {
        if (ada_response) {
          return ada_response;
        }
        // tidak ada response, ambil ke jaringan
        else {
          return fetch(event.request)
        }
      })
  )
  .catch(error => {
    return new Response("Waduh " + error);
  })
);
});
```

File index.html

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Wisata Kuliner terbaru</title>
<link rel="stylesheet" href="css/mygrid.css">
</head>
<body>
  <div class="container">
    <div class='menu'>Menu Baru</div>
    <div class='header'>Header</div>
    <div class='sidebar'>SideBar</div>
    <div class='konten'>
      <div class='subkonten'>
        <a href='p1/add2numbers.html'>Add 2 Numbers </a>
      </div>
      <div class='subkonten'>
        <a href='p2/peta.html'>Peta Kuliner </a>
      </div>

      <div class='subkonten'>
        <a href='p3/fetchjson.html'>Peta Kuliner v2</a>
      </div>
      <div class='subkonten'>
        <a href='p4/materi.html'>Download Files Latihan</a>
      </div>

    </div>
    <div class='footer'>Footer</div>
  </div>
  <script>
    if ('serviceWorker' in navigator) {
      window.addEventListener('load', function() {
        navigator.serviceWorker.register('sw.js')
          .then(reg => console.log('SW terdaftar', reg))
          .catch(err => console.log('SW Gagal: ', err));
      });
    }
  </script>
</body>
</html>
```