

PWA

PSB 2020
Y. Stroppa

Références

Infos sur les services workers

<https://developers.google.com/web/fundamentals/primers/service-workers/>

https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Youtube :

<https://www.youtube.com/watch?v=ksXwaWHCW6k>

<https://jakearchibald.com/2014/service-worker-first-draft/>

<https://jakearchibald.com/2014/offline-cookbook/>

Tuto de mise en oeuvre

<https://medium.com/james-johnson/a-simple-progressive-web-app-tutorial-f9708e5f2605>

Avec NodeJS

<https://www.twilio.com/blog/2018/03/practical-introduction-pwa-node-hoode-offline-first.html>

Cours sur le mise en oeuvre d'une PWA

<https://github.com/google-developer-training/pwa-training-labs>

<https://developers.google.com/web/tools/workbox/>

<https://codelabs.developers.google.com/dev-pwa-training/>

Sommaire

Introduction

Rappels Javascript :

callbacks,

await,

async,

fetch et promises

Principe de fonctionnement PWA

Étude de cas : shopping List

Réalisations à faire/ projet

PWA Introduction

Les PWA sont conçues pour apparaître comme des applications locales, bien que fondées en ligne.

2017 Google annonce la fin des applications chrome et du store chrome pour laisser la place aux PWA qui fournissent une expérience identique à l'utilisateur et qui ont l'avantage de fonctionner sur tous les navigateurs récents en utilisant les technos du web.

Les PWA s'exécutent dans une application universelle, le navigateur Web.

Les PWA sont en fait des caractéristiques plus précises :

Réactives grâce au ,mode asynchrone des workers, des websockets

Fonctionnant hors ligne grâce aux services workers et à indexedDB

PWA Introduction

L'idée est de développer une seule application pour tous les médias.

moins chère de développer une PWA qu'une application native.

Look Natif de l'interface et l'interaction

Mise à jour immédiate du programme et du contenu par le web
indexable grâce au manifeste qui permet aussi de configurer
localement l'application

sûres quand à l'origine, **elles doivent être en HTTPS.**

Autre avantage , alors qu'elles étaient jusqu'à là réservées aux app natives, les développeurs peuvent désormais activer les notifications push pour les PWA également.

Concepts d'une application PWA

Les utilisateurs profitent d'une expérience sans pareil.

Fluide, intuitive : une expérience App like

Pour tous les utilisateurs récurrents, cela sous entend que les éléments clé de l'app seront affichés immédiatement, dans la mesure du possible où ils ont été mis en cache, pour une performance accrue, le contenu étant lui récupéré à chaque utilisation.

Accessible et légère : une consultation sans contrainte

vos utilisateurs sont seulement à 1 clic de votre contenu et ils accèdent à la seconde, sur n'importe quel appareil. Les PWAs ne nécessitent pas d'installation pour être consultées. Aussi leur légèreté incomparable ôte toute contrainte de mémoire sur les devices au moment de l'installation quand un utilisateur tente d'accéder à une page en particulier, seulement les éléments nécessaires pour afficher cette page sont chargés.

Concepts d'une application PWA

Rapide --> chargement instantané

La mise à jour dynamique du contenu est assurée afin que l'utilisateur ait toujours accès au contenu le plus récent. Les PWAs ont la possibilité d'exécuter du code javascript sans avoir besoin d'être au premier plan, voire sans qu'aucune page web ne soit ouverte dans le navigateur. Ainsi la maj régulière des données de l'app est rendue possible, permettant ainsi l'affichage direct des données à jour à l'arrivée de l'utilisateur. Finis le temps de chargement interminables, vos utilisateurs navigueront sur votre PWA avec une fluidité et une rapidité sans pareil sur le web.

Fonctionnement hors ligne : zéro frustration pour l'utilisateur

Une des caractéristiques majeures d'une PWA, c'est sa capacité à fonctionner en l'absence d'une connexion internet stable. A la première visite de l'utilisateur, le système de mise en cache permettra de stocker localement une partie ou l'intégralité du contenu disponible. Ainsi, à la prochaine visite et même sans aucune connexion, l'utilisateur pourra avoir accès à ce contenu et pourra naviguer entre les pages de la PWA.

Le fonctionnement d'une PWA

Les PWAs sont à mi chemin entre les sites web et les apps natives. Elles utilisent le meilleur de ces 2 concepts. Leur fonctionnement repose donc sur la combinaison inédite de concepts existants :

Comment une web app devient-elle progressive ?

pour être qualifiée de PWA, une application web doit absolument répondre à certains critères :

Progressive : elle doit fonctionner pour tout utilisateur quel que soit le navigateur

Responsive : elle doit pouvoir s'adapter à tout type d'écran.

indépendante du réseau : elle doit fonctionner même si il n'y a pas de connexion internet ou si la communication est de mauvaise qualité.



Concepts d'une application PWA

Une PWA repose sur dix principes clés :

Progressive

Fonctionne pour tous les utilisateurs sans se soucier du navigateur internet qu'ils utilisent.

Responsive

Convient à toutes les tailles d'écrans : téléphone mobile, tablette, ordinateur..

Connectivity independent

Est propulsée par les services workers pour fonctionner en mode déconnecté et sur les réseaux de mauvaise qualité.

App-like

Elle doit fournir la même expérience qu'une application native. Ressemble à une application mobile. Grâce à l'App Shell qui permet de faire la distinction entre le fonctionnement de l'application et son contenu.

Fresh

Est toujours à jour grâce au processus de mise à jour des service workers.

Concepts d'une application PWA

Safe

Utilise le HTTPS pour sécuriser le flux d'échange et pour s'assurer que le contenu n'a pas été altéré.

Discoverable

Est identifiée comme une "application" grâce au Web App Manifest et au service worker registration, permettant aux moteurs de recherche de la trouver facilement.

Engageante

Interpelle l'utilisateur avec des fonctionnalités telles que les notifications push.

Installable

Permet aux utilisateurs d'ajouter les applications qu'ils trouvent pertinentes à leur page d'accueil sans avoir à passer par une boutique d'applications.

Linkable

Se partage facilement via une URL et ne requiert pas une installation complexe

Les technologies

Pour qu'une application web soit similaire à une application locale elle doit être réactive, sans latence ce qui est permis par une série de nouveaux outils :

[Web Assembly](#) : nouveau langage de type bytecode permet d'ajouter des APIs javascript, disponibles dans le navigateur et qui sont générées à partir de C, C++ ou autres langages. Leur vitesse d'exécution est proche de celle du code binaire.

[IndexedDB](#) : une bdd peut être créée sur le poste client avec indexedDB. Elle est uniquement accessible par l'application qui l'a créée. Elle peut copier des données d'une base sur le serveur. Elle peut aussi contenir du code wasm ou JS qui restera stocké en permanence sur le poste client et n'aura plus à être chargé à chaque nouvelle session.

Les technologies

Pour qu'une application web soit similaire à une application locale elle doit être réactive, sans latence ce qui est permis par une série de nouveaux outils :

Web workers: on peut exécuter des scripts en tâche de fond, de façon asynchrones dans un web worker. un moyen de rendre l'application plus fluide. IL communique avec la page en PostMessage.

Service workers : c'est une sorte de web workers dédié aux interactions entre le serveur et l'application. Il fonctionne de manière asynchrone et invisible utilisant l'interface postMessage. C'est un proxy à tout ce qui vient du serveur, il intercepte ce qui est communiqué, le transmet à la page qui l'utilise au moment opportun.

Mode offline : on pourra utiliser les services workers pour indiquer les actions et fichiers à copier en local. Le fichier sw.js.

Le fonctionnement d'une PWA

Les PWAs sont développées à partir de langage web client, qui vont permettre de gérer l'interface utilisateur de chaque page.

Le choix possible est le développement d'applications isomorphiques . Applications capables d'exécuter aussi bien du côté serveur (Server Side Rendering SSR) que du côté client (Client Side Rendering CSR).

Comment ça fonctionne :

la première page de l'application est générée du côté serveur (SSR) permettant ainsi aux robots des moteurs de recherche d'indexer une page entièrement construite. Mais l'application effectue des calculs à l'intérieur du navigateur (CSR) permettant ainsi à l'application de fonctionner sans réseau puisque tout le Javascript de la PWA aura été téléchargé en arrière plan, et stocké dans le cache.

Architecture d'une PWA

App shell

App Shell : Il s'agit du squelette de votre application, à l'intérieur duquel seront diffusées vos données. L'App Shell contient les éléments principaux d'interface et les composants strictement nécessaires pour le fonctionnement de l'interface utilisateur. Ces éléments y sont conservés localement, tandis que les contenus sont récupérés dynamiquement depuis une API.

L'App Shell permet, à partir du deuxième passage dans l'application, un affichage rapide de l'application, une consommation de bande passante la plus faible possible, un chargement des ressources statiques depuis le cache local, et enfin la séparation du contenu de la navigation, permettant ainsi d'obtenir des temps de chargement de l'application extrêmement courts.

Architecture d'une PWA

Services workers

Les service workers ?

Il s'agit de workers JavaScript situés entre l'application et le réseau, et agissant comme un proxy. Exécutes dans un thread séparé, ils peuvent effectuer des actions alors même que la Progressive Web App n'est pas utilisée. De ce fait, ils peuvent gérer la mise en cache et intercepter les requêtes réseaux afin de proposer une expérience utilisateur rapide et fiable, sans se soucier de la qualité du réseau. En ce sens, ils permettent de donner vie à une Progressive Web App.

Détail de la spécifications

<https://w3c.github.io/ServiceWorker/#service-worker-obj>

Architecture d'une PWA

Manifeste

<https://medium.com/@guillaumeandre/progressive-web-app-pwa-fichier-web-app-manifest-7292db378af5>

Votre PWA est décrite dans un fichier JSON appelé le manifeste (Web App Manifest). Il s'agit d'un fichier contenant les métadonnées nécessaires à l'indexation de votre app dans un Store comme le Windows Store, ou à l'installation de votre app sur l'écran d'accueil de votre utilisateur.

Il s'agit donc d'un fichier descriptif qui permettra de donner un rendu plus natif à l'application avec un affichage plein écran, des icônes identifiables, ou encore la possibilité de modifier l'orientation de l'écran.

Architecture d'une PWA

Manifeste

<https://medium.com/@guillaumeandre/progressive-web-app-pwa-fichier-web-app-manifest-7292db378af5>

Site d'info sur tous les meta-données

<https://www.w3.org/TR/appmanifest/#webappmanifest-dictionary>

Et un site pour valider votre manifest si besoin

<https://manifest-validator.appspot.com/>

Liste des propriétés et fonctionnalités non-exhaustive

name : Le nom de la PWA retrouvable au sein de l'environnement ayant installé la PWA

short_name : Le nom utilisé par l'environnement juste en dessous de l'icône de la PWA

start_url : Il faut renseigner l'url de démarrage de la PWA.

Une des bonnes pratiques consiste à tracker cette url en spécifiant une query string notamment avec les paramètres UTM et l'outil Campaign URL Builder de Google (<https://ga-dev-tools.appspot.com/campaign-url-builder/>)

display : Permet de spécifier comment la PWA va être présentée. fullscreen permet d'ouvrir l'application en plein écran, stand-alone de garder la statusbar et le bouton back, minimal-ui avec aucun bouton et browser pour une vue par défaut (navigateur classique). Cette fonctionnalité permet aussi de filtrer certains périphériques qui ne sauraient pas gérer le mode d'affichage renseigné. fullscreen et minimal-ui ne sont pas encore supportés par Safari.

Architecture d'une PWA

Manifeste

<https://medium.com/@guillaumeandre/progressive-web-app-pwa-fichier-web-app-manifest-7292db378af5>

orientation : Permet de préciser le type d'affichage pour la PWA à savoir portrait, paysage ou les deux.

background_color : Force la couleur du background avant le chargement des CSS. On voit le background_color généralement au lancement de la PWA sur une très courte durée.

scope : Par défaut c'est la start_url, il est possible de renseigner par exemple un scope spécifique pour chaque type de Service Workers.

icons : Peut contenir une collection d'icônes avec différentes tailles

serviceworker : Permet de renseigner un Service Worker, mais il est coutume d'enregistrer et instancier directement un Service Worker via JavaScript au sein de l'application.

Rappels Javascript

Callbacks

Async et await

Fetch

Promises

Rappel notion avancée en Javascript

Mise en oeuvre Callback

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta name="viewport" content="width=device-width, minimum-scale=1.0,
initial-scale=1.0, user-scalable=yes">
    <meta charset="utf-8">
    <title>Async JS</title>
  </head>

  <body>
    <header>
      <h1>Première application Callback-Sync-await-promises</h1>
    </header>

    <script src="callbacks.js"> </script>

  </body>
</html>
```

Rappel notion avancée en Javascript

<https://javascript.info/callbacks>

Callbacks, async, await et promises

Plusieurs fonctions dans une application web sont fournies par Javascript qui autorise des actions asynchrones. En d'autres mots, des actions que l'on déclenche maintenant et qui finiront plus tard.

Par exemple : la fonction `setTimeout` et d'autres telles que le chargement de script et de modules.

Soit un petit exemple, on souhaite charger dans la page HTML en dynamique un script en JS et utiliser une des fonctions de ce script. Dans les instructions pour réaliser ce type d'opération sont :

callbacks.js

```
function loadScript(src) {  
  let script = document.createElement('script');  
  script.src = src;  
  document.head.append(script);  
}  
loadScript('callbacks_complement.js') ;  
getPosts() ;
```

Rappel notion avancée en Javascript

Callback_complement.js

```
const posts =[
  {title:"Post One", body:"this is post one"},
  {title:"Post Two", body:"this is post two"},
];
function getPosts() {
  setTimeout(()=> {
    let output="";
    posts.forEach(
      (post,index) => {
        output +=`<li>${post.title}</li>`;
      }
    );
    document.body.innerHTML=output;
  },1000);
}
function createPost(post,callback) {
  setTimeout(() => {
    posts.push(post);
    callback();
  }, 2000  );
}
```

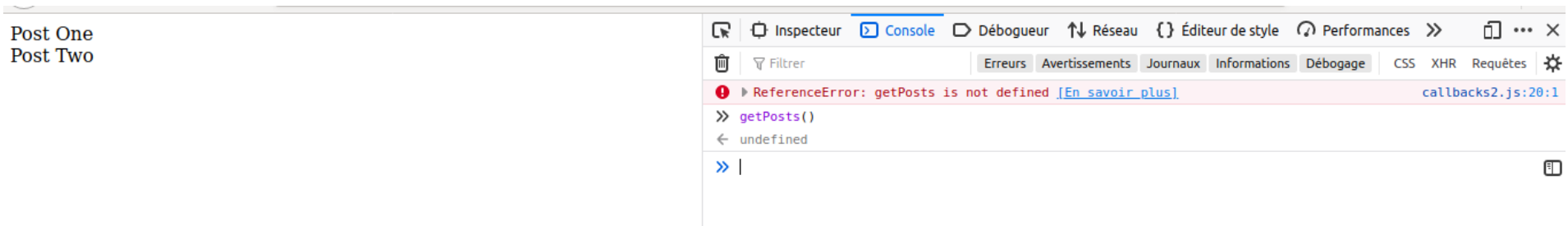
Rappel notion avancée en Javascript

Callback

Si on utilise ce code dans l'état, on obtient du navigateur le message suivant dans la console :



Problème car Javascript n'attend pas que le script soit chargé, et passe à l'interprétation de la deuxième instruction ... pourtant si on essaye dans la console on accède bien à la fonction



Rappel notion avancée en Javascript

Callback

Pour éviter ce problème, on va créer une dépendance dans les enchaînements de ces étapes en utilisant une callback à l'appel de la fonction `loadScript` de la manière suivante :

```
// fonction avec un callback
function loadScriptC(src, callback) {
  let script = document.createElement('script');
  script.src = src;
  script.onload = () => callback(script);
  document.head.append(script);
}

loadScriptC('./callbacks.js', function() {getPosts();});
ou
loadScriptC('./callbacks.js', ()=> {getPosts();});
```

Et là tout se passe comme attendu....

Rappel notion avancée en Javascript

Callback dans des callbacks

Si on souhaite exécuter des scripts de façon séquentiels après le premier, on peut le faire en insérant un deuxième appel dans la fonction callback avec un autre script. De la façon suivante :

```
loadScript('script1.js', function (script) {  
    console.log("Appel premier script") ;  
    loadScript('script2.js', function(script) {  
        console.log("Appel deuxieme script") ;  
    }  
    .....  
})
```

Dans notre exemple :

```
loadScriptC('./callbacks.js', ()=> {  
    createPost({title:'Post Three', body:'This is post three'},getPosts);  
    createPost({title:'Post Four', body:'This is post four'},getPosts);  
    createPost({title:'Post Fixe', body:'This is post five'},getPosts);  
});
```

Rappel notion avancée en Javascript

Callback : gestion des erreurs

Pour gérer les erreurs d'exécution éventuelles, il faut ajouter un gestionnaire d'événement sur l'erreur :

```
function loadScript(src, callback) {  
  let script = document.createElement('script');  
  script.src = src;  
  
  script.onload = () => callback(null, script);  
  script.onerror = () => callback(new Error(`Script load error for ${src}`));  
  
  document.head.append(script);  
}  
  
loadScript('script.js', function(error, script) {  
  if (error) {  
    // handle error  
  } else {  
    // script loaded successfully  
  }  
});
```

Rappel notion avancée en Javascript

Callback de l'enfer ou pyramid of Doom

```
loadScript('1.js', function(error, script) {  
  
  if (error) {  
    handleError(error);  
  } else {  
    // ...  
    loadScript('2.js', function(error, script) {  
      if (error) {  
        handleError(error);  
      } else {  
        // ...  
        loadScript('3.js', function(error, script) {  
          if (error) {  
            handleError(error);  
          } else {  
            // ...continue after all scripts are loaded ...  
          }  
        });  
      }  
    });  
  }  
});
```

```
loadScript('1.js', function(error, script) {  
  if (error) {  
    handleError(error);  
  } else {  
    // ...  
    loadScript('2.js', function(error, script) {  
      if (error) {  
        handleError(error);  
      } else {  
        // ...  
        loadScript('3.js', function(error, script) {  
          if (error) {  
            handleError(error);  
          } else {  
            // ...  
          }  
        });  
      }  
    });  
  }  
});
```



Rappel notion avancée en Javascript

Une solution pour éviter cette spirale

```
loadScript('1.js', step1);

function step1(error, script) {
  if (error) {
    handleError(error);
  } else {
    // ...
    loadScript('2.js', step2);
  }
}

function step2(error, script) {
  if (error) {
    handleError(error);
  } else {
    // ...
    loadScript('3.js', step3);
  }
}

function step3(error, script) {
  if (error) {
    handleError(error);
  } else {
    // ...continue after all scripts are loaded (*)
  }
};
```

Rappel notion avancée en Javascript

https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses

Les promesses :

L'objet Promise est utilisé pour réaliser des traitements de façon asynchrone. Elle représente la complétion ou l'échec d'une opération asynchrone. En résumé, une promesse est un objet qui est renvoyé et auquel on attache des callbacks. Ainsi au lieu d'avoir une fonction qui prend deux callbacks en arguments

```
function faireQqcALAncienne(successCallback, failureCallback){
    console.log("C'est fait");
    // réussir une fois sur deux
    if (Math.random() > .5) {
        successCallback("Réussite");
    } else {
        failureCallback("Échec");
    }
}
function successCallback(résultat) {
    console.log("L'opération a réussi avec le message : " + résultat);
}
function failureCallback(erreur) {
    console.error("L'opération a échoué avec le message : " + erreur);
}
faireQqcALAncienne(successCallback, failureCallback);
```

Rappel notion avancée en Javascript

Promise

On aura une fonction qui renvoie une promesse et on attachera les callbacks sur cette promesse.

```
function faireQqc() {  
  return new Promise((resolve, reject) => {  
    console.log("C'est fait");  
    // réussir une fois sur deux  
    if (Math.random() > .5) {  
      resolve("Réussite");  
    } else {  
      reject("Échec");  
    }  
  })  
}  
  
const promise = faireQqc();  
promise.then(successCallback, failureCallback);  
ou encore  
faireQqc().then(successCallback, failureCallback);
```

On observera l'instruction `const` devant `promise` qui indique que l'état de la promesse est évalué et conservé.

```
>> promise  
← Promise { "fulfilled" }  
  <state>: "fulfilled"  
  <value>: "Réussite"  
  <prototype>: PromiseProto { ... }  
>>
```

```
>> promise.then(successCallback, failureCallback);  
← Promise { <state>: "pending" }  
L'opération a réussi avec le message : Réussite  
debugger eval code:2:11  
>>
```

Rappel notion avancée en Javascript

Promise

La structure d'une promesse est donc :

```
const maPremierePromesse= new Promise( (resolve,reject)=> {  
  // réaliser une tâche asynchrone et appeler :
```

```
  // resolve(uneValeur) ; // si la promesse est tenue
```

```
  // ou
```

```
  // reject("Raison du rejet") ; // si la promesse est rompue
```

```
});
```

Forme classique

```
function maFonctionAsync(url) {  
  ((resolve,reject)=>{  
    const xhr=new XMLHttpRequest() ;  
    xhr.open("GET",url) ;  
    xhr.onload=() => resolve(xhr.responseText) ;  
    xhr.onerror=() =>reject(xhr.statusText) ;  
    xhr.send() ;  
  }) ;
```

```
function reqListener () {  
  console.log(this.responseText);  
}
```

```
var oReq = new XMLHttpRequest();  
oReq.onload = reqListener;  
oReq.open("get", "yourFile.txt", true);  
oReq.send();
```

Permet de mieux gérer les situations lors d'appel
synchrone, car c'est encapsulé

Rappel notion avancée en Javascript

Promise

Cette dernière forme est ce qu'on appelle un appel de fonction asynchrone. Cette convention possède différents avantages dont le premier est le chaînage

Garanties :

A la différence des imbrications de callbacks, une promesse apporte certaines garanties :

Les callbacks ne seront jamais appelés avant la fin du parcours de la boucle d'événements Javascript courante

Les callbacks ajoutés grâce à **then** seront appelés , y compris après le succès ou l'échec de l'opération asynchrone.

Plusieurs callbacks peuvent être ajoutés en appelant **then** plusieurs fois, ils seront alors exécutés l'un après l'autre selon l'ordre dans lequel ils ont été insérés.

Autre exemple

```
// Juste pour la démonstration de l'appel à l'API fetch
function recup_fichier_data() {
  fetch('./data.json')
    .then(response => {return response.json();})
    .then(data => { console.log(data);})
    .catch(err => { console.log(err);})
}
```

Rappel notion avancée en Javascript

Promise

Chaînage des promesses :

Un besoin fréquent d'exécuter deux ou plus d'opérations asynchrones les unes à la suite des autres, avec chaque opération qui démarre lorsque la précédente a réussi et en utilisant le résultat de l'étape précédente. Ceci eut être réalisé en créant une chaîne de promesses.

La méthode `then()` renvoie une nouvelle promesse, différente de la première :

```
faireQqc().then(function(result) {  
  return faireAutreChose(result);  
})  
.then(function(newResult) {  
  return faireUnTroisiemeTruc(newResult);  
})  
.then(function(finalResult) {  
  console.log('Résultat final : ' + finalResult);  
})  
.catch(failureCallback);
```

```
faireQqc()  
.then(result => faireAutreChose(result))  
.then(newResult => faireUnTroisiemeTruc(newResult))  
.then(finalResult => {  
  console.log('Résultat final : ' + finalResult);  
})  
.catch(failureCallback);
```

Analogie avec les Pipes (tubes) en shell

Rappel notion avancée en Javascript

Async/Await

Async et await

Sont encore des mécanismes qui sont utilisées dans des appels de ressources distantes qui nécessitent un certains temps :

```
async function fetchusers() {  
  const res= await fetch('https://jsonplaceholder.typicode.com/users');  
  const data=await res.json();  
  console.log(data);  
}  
fetchusers();
```

On va chercher une ressource distante et on l'a converti en json pour ensuite l'afficher, permet d'avoir une exécution séquentielle dans un bloc déclaré asynchrone

Rappel notion avancée en Javascript

Notion Fetch : en exercice

https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

L'API Fetch fournit une interface JavaScript pour l'accès et la manipulation des parties de la pipeline HTTP, comme les requêtes et les réponses. Cela fournit aussi une méthode globale `fetch()` qui procure un moyen facile et logique de récupérer des ressources à travers le réseau de manière asynchrone.

Ce genre de fonctionnalité était auparavant réalisé avec `XMLHttpRequest`. Fetch fournit une meilleure alternative qui peut être utilisée facilement par d'autres technologies comme `Service Workers`. Fetch fournit aussi un endroit unique et logique pour la définition d'autres concepts liés à HTTP comme `CORS` et les extensions d'HTTP.

Rappel notion avancée en Javascript

Notion Fetch : en exercice

```
var myImage = document.querySelector('img');
```

```
fetch('flowers.jpg')  
.then(function(response) {  
  return response.blob();  
})  
.then(function(myBlob) {  
  var objectURL = URL.createObjectURL(myBlob);  
  myImage.src = objectURL;  
});
```

```
// avec l'option init  
var myHeaders = new Headers();  
var myInit = { method: 'GET',  
               headers: myHeaders,  
               mode: 'cors',  
               cache: 'default' };  
fetch('flowers.jpg',myInit)  
.then(function(response) {  
  return response.blob();  
})  
.then(function(myBlob) {  
  var objectURL = URL.createObjectURL(myBlob);  
  myImage.src = objectURL;  
});
```

Rappel notion avancée en Javascript

Api Notification : exercice de mise en oeuvre

https://developer.mozilla.org/fr/docs/Web/API/notification/Using_Web_Notifications

L'API de Notifications Web permet à une page Web d'envoyer des notifications qui s'affichent hors de la page au niveau du système. Cela permet aux applications web d'envoyer des informations à un utilisateur, même si l'application est inactive. Un des principaux cas d'utilisation évidente est une application de messagerie Web qui informe l'utilisateur à chaque fois qu'un nouvel e-mail est reçu, même si l'utilisateur fait autre chose avec une autre application.

Pour afficher des notifications, il faut commencer par demander la permission appropriée et d'instancier un objet Notification :

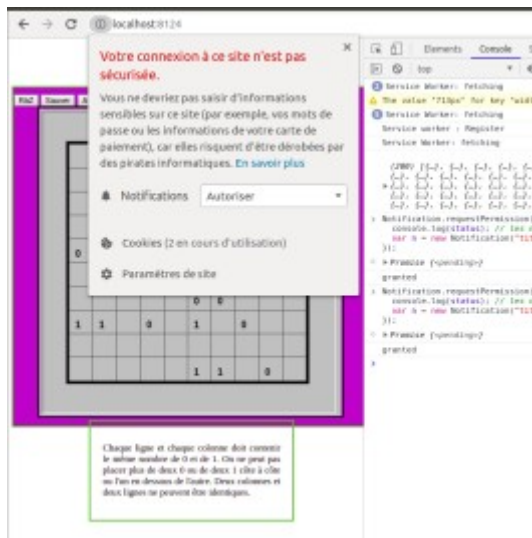
Rappel notion avancée en Javascript

Api Notification : exercice de mise en oeuvre

https://developer.mozilla.org/fr/docs/Web/API/notification/Using_Web_Notifications

Demande d'autorisation que doit valider l'utilisateur

```
Notification.requestPermission( function(status) {  
    console.log(status);  
    var n = new Notification("title", {body: "notification body"});  
    // this also shows the notification  
});
```



Demandé au démarrage de l'application

```
window.addEventListener('load', function () {  
    Notification.requestPermission(function (status) {  
        // Cela permet d'utiliser Notification.permission avec Chrome/Safari  
        if (Notification.permission !== status) {  
            Notification.permission = status;  
        }  
    });  
});
```

Exemple d'envoi

```
var n = new Notification("Bienvenue !");
```

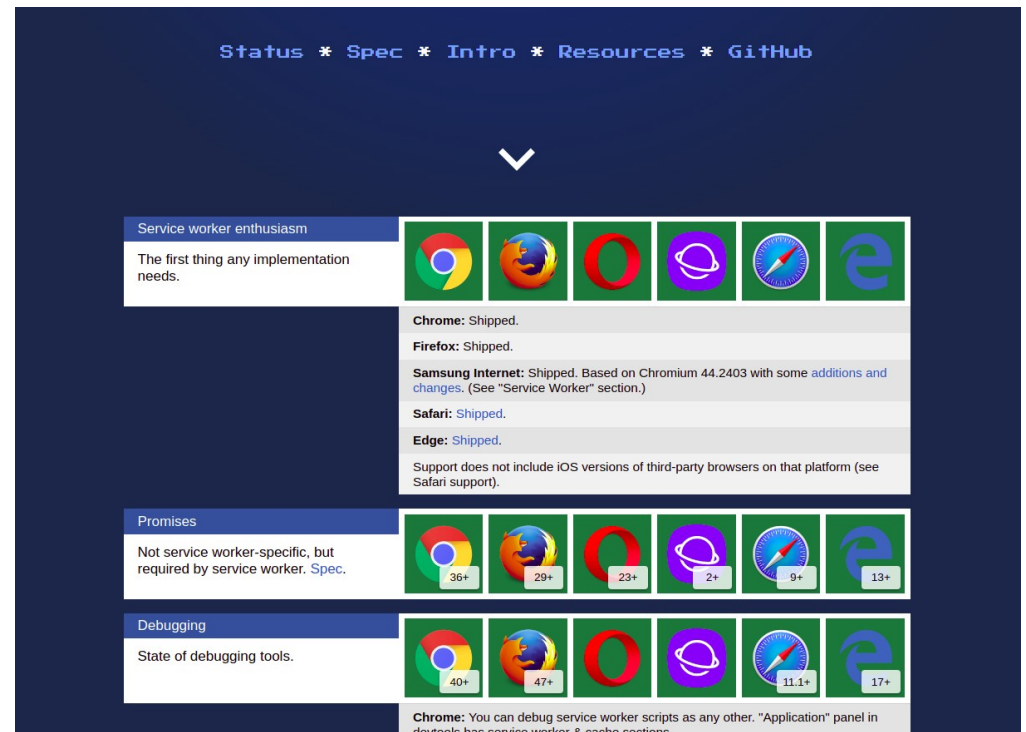
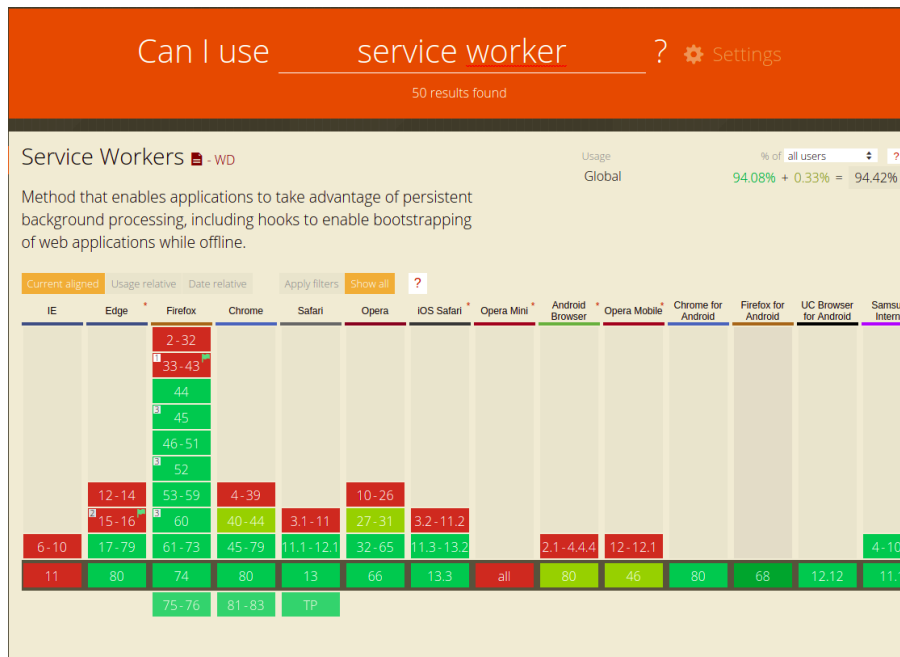

Les services workers

Sites d'infos sur les compatibilités

Sites d'informations :

<https://caniuse.com/#search=service%20worker>

<https://jakearchibald.github.io/isserviceworkerready/>



Services workers

Rappel objectif est d'autoriser le offline

Les restrictions du SW :

ne peut pas accéder à la DOM de la page WEB mais peut communiquer via des envoies de messages grâce à l'interface postMessage

le SW est programmable comme un proxy réseau, qui contrôle les requêtes émises par le serveur pour votre page web.

ce termine lorsqu'il n'est plus utilisé et redémarre dès que l'on en a besoin, ce qui induit qu'il n'a pas d'état global donc on ne peut pas utiliser de la persistance il faut pour cela utiliser indexedDB API.

le SW est une extension des promises

Avantages et inconvénients

Avantages	Désavantages
Les PWA fonctionnent sur différentes plateformes ce qui induit un coût de développement réduit par rapport à des applications natives.	La visibilité des PWA est quasi nulle pour les utilisateurs qui passent par les boutiques d'applications « classiques » de recherche d'applications.
Les PWA utilisent peu de données mobiles et d'espace disque. Pour une application native qui aurait une taille de 10 Mb, son équivalent en PWA aurait une taille de l'ordre de 500 Kb.	Les plugins tels que Facebook Login ou Google Login ne peuvent pas récupérer les données de leurs applications respectives. Il faut obligatoirement s'identifier en ligne.
Les PWA n'ont pas besoin d'être mises à jour par l'utilisateur. Leur mise à jour s'effectue comme pour les pages internet. L'utilisateur est donc assuré d'utiliser la dernière version en ligne de la PWA.	Les PWA ne peuvent pas utiliser certaines fonctionnalités hardware. Par exemple : le lecteur d'empreinte digitale.
Les PWA n'ont pas besoin d'être « installées » pour fonctionner. L'utilisateur choisira lorsqu'il souhaitera faire l'« installation ».	Le support complet des PWA n'est pas disponible dans tous les navigateurs internet dont Internet Explorer.

Les outils pour créer une application PWA

Les outils pour créer une PWA

Vous pouvez créer des PWA à partir du HTML, du CSS et du JavaScript. Si vous souhaitez avoir une première expérience avec les PWA sans aucun framework, Voir les tutos indiqués au début de cette présentation.

Voici quelques outils vous permettant de créer une Progressive Web App :

Vous pouvez utiliser Angular, framework populaire pour le front-end, pour créer une PWA, notamment depuis la version 5 avec ses nouveaux service workers.

Polymer, proposé par Google, met à disposition une collection de composants web, d'outils et des modèles pour construire des PWA.

Ionic est un framework open-source qui vous permet de concevoir des applications mobiles hybrides et web.

vérification compatibilité PWA à l'aide de google chrome F12 - Audit

The screenshot shows the Google Chrome DevTools interface with the 'Audits' tab selected. The browser window displays a game titled 'Jeu de l'âne rouge M2ISN 2020' on a mobile device. The game interface includes a donkey image, a 3x3 grid puzzle, and buttons for 'RàZ', 'Sauver', 'Auto', and 'Pte 1'. The text below the grid explains the game rules and controls.

Jeu de l'âne rouge M2ISN 2020

L'âne rouge est un casse-tête du genre puzzle à glissement.
Il faut amener le grand carré rouge en face de la porte inférieure C'est plutôt difficile.

Pour jouer glisser les pions avec la souris. (Cliquer sur le pion que vous voulez déplacer puis faire glisser le pointeur vers la case libre).

Le bouton **[RàZ]** remet les pions dans la situation de départ.

Le bouton **[Sauver]** permet de sauver une configuration prometteuse. Un click sur le bouton **[Restaurer]** remplace cette configuration sur le plateau.

Le bouton **[Auto]** lance une animation qui reproduit les mouvements des pièces conduisant à une solution.

Audit Results:

- Performance: 100
- Progressive Web App: 38
- Accessibility: 69
- Best Practices: 80
- SEO: 67

Score scale: 0-44 (red), 45-74 (orange), 75-100 (green).

Performance Metrics:

- First Contentful Paint: 750 ms ✓
- Speed Index: 750 ms ✓
- Time to Interactive: 810 ms ✓
- First Meaningful Paint: 750 ms ✓
- First CPU Idle: 810 ms ✓
- Estimated Input Latency: 13 ms ✓

Opportunities:

These are opportunities to speed up your application by optimizing the following resources.

Resource to optimize	Estimated Savings
1 Eliminate render-blocking resources	0,15 s
2 Properly size images	0,15 s

Diagnostics:

More information about the performance of your application.

- 1 Critical Request Chains: 3 chains found

Passed audits: 19 audits

Service Worker

Un service worker est un serveur proxy programmable qui est exécuté de façon séparé du navigateur dans un thread et vous autorise à intercepter des requêtes et processus que vous avez choisis.

Vous pouvez intercepter et mettre en cache des requêtes pour les fournir du côté serveur dès que possible.

L'API Cache est une partie du service worker pour stocker ces requêtes.

Cycle de vie d'un service worker

https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Worker lifecycle

INSTALLING

This stage marks the beginning of registration. It's intended to allow to setup worker-specific resources such as offline caches.

⚡ install

- Use `event.waitUntil()` passing a promise to extend the installing stage until the promise is resolved.
- Use `self.skipWaiting()` anytime before activation to skip installed stage and directly jump to activating stage without waiting for currently controlled clients to close.



INSTALLED

The service worker has finished its setup and it's waiting for clients using other service workers to be closed.



ACTIVATING

There are no clients controlled by other workers. This stage is intended to allow the worker to finish the setup or clean other worker's related resources like removing old caches.

⚡ activate

- Use `event.waitUntil()` passing a promise to extend the activating stage until the promise is resolved.
- Use `self.clients.claim()` in the activate handler to start controlling all open clients without reloading them.



ACTIVATED

The service worker can now handle functional events.



REDUNDANT

This service worker is being replaced by another one.



Events

⚡ install

⚡ activate

⚡ message

Functional events

⚡ fetch

⚡ sync

⚡ push

Activez le service dans votre navigateur

https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Setting up to play with service workers

Many service workers features are now enabled by default in newer versions of supporting browsers. If however you find that demo code is not working in your installed versions, you might need to enable a pref:

Firefox Nightly: Go to `about:config` and set `dom.serviceWorkers.enabled` to `true`; restart browser.

Chrome Canary: Go to `chrome://flags` and turn on `experimental-web-platform-features`; restart browser (note that some features are now enabled by default in Chrome.)

Opera: Go to `opera://flags` and enable `Support for ServiceWorker`; restart browser.

Microsoft Edge: Go to `about:flags` and tick `Enable service workers`; restart browser.

Visualisation du Worker Service

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. The left sidebar contains a tree view with categories: Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, psb_test1 - http://localhost:8080, Application Cache), and Background Services (Background Fetch, Background Sync, Notifications, Payment Handler, Push Messaging). The 'Service Workers' item is highlighted. The main panel displays the 'Service Workers' section for the origin `http://localhost:8080/`. It includes controls for Offline, Update on reload, and Bypass for network. A single service worker is listed with source `sw.js`, received on 12/03/2020 at 17:19:27, and status '#224 activated and is running'. It has a client at `http://localhost:8080/`. Below the worker details are input fields for 'Push' (containing 'Test push message from DevTools.') and 'Sync' (containing 'test-tag-from-devtools'), each with a corresponding button. At the bottom, there is a section for 'Service workers from other origins'.

Application

- Manifest
- Service Workers**
- Clear storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Cache

- Cache Storage
- psb_test1 - http://localhost:8080
- Application Cache

Background Services

- Background Fetch
- Background Sync
- Notifications
- Payment Handler
- Push Messaging

Service Workers

☐ Offline ☐ Update on reload ☐ Bypass for network

http://localhost:8080/ [Update](#) [Unregister](#)

Source [sw.js](#)

Received 12/03/2020 à 17:19:27

Status ● #224 activated and is running [stop](#)

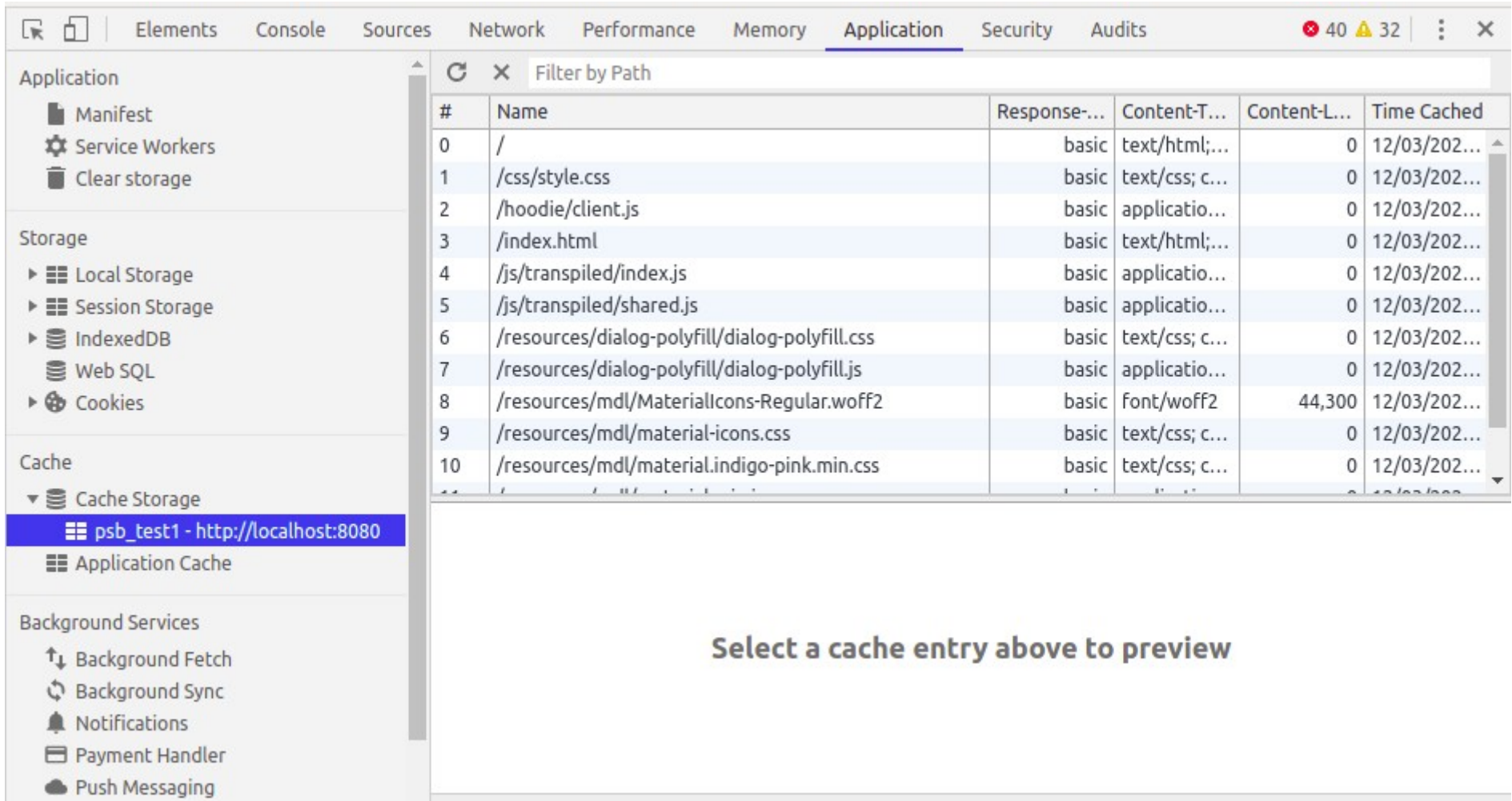
Clients [http://localhost:8080/](#) [focus](#)

Push [Push](#)

Sync [Sync](#)

► Service workers from other origins

Visualisation du Worker Service

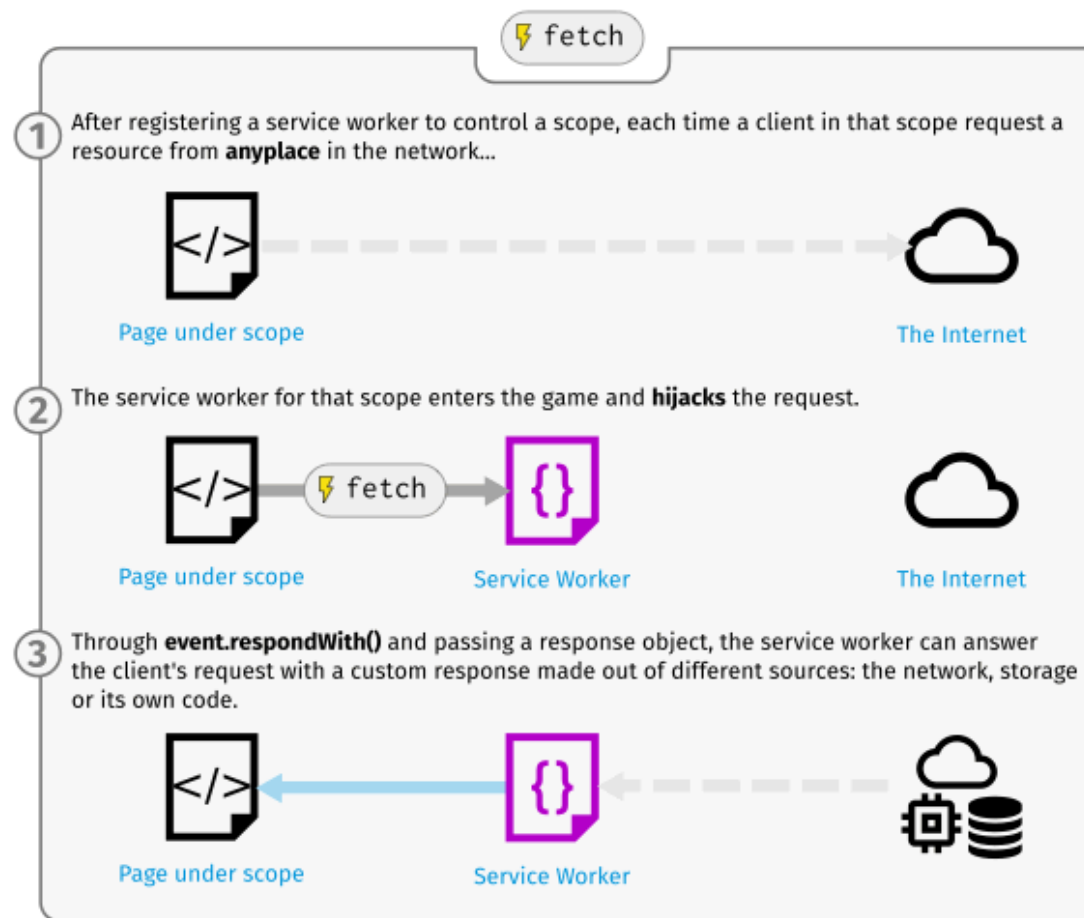


The screenshot shows the Chrome DevTools Application tab. The left sidebar has a tree view with sections: Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, **psb_test1 - http://localhost:8080**, Application Cache), and Background Services (Background Fetch, Background Sync, Notifications, Payment Handler, Push Messaging). The main panel displays a table of cached resources for the selected service worker.

#	Name	Response...	Content-T...	Content-L...	Time Cached
0	/	basic	text/html;...	0	12/03/202...
1	/css/style.css	basic	text/css; c...	0	12/03/202...
2	/hoodie/client.js	basic	applicatio...	0	12/03/202...
3	/index.html	basic	text/html;...	0	12/03/202...
4	/js/transpiled/index.js	basic	applicatio...	0	12/03/202...
5	/js/transpiled/shared.js	basic	applicatio...	0	12/03/202...
6	/resources/dialog-polyfill/dialog-polyfill.css	basic	text/css; c...	0	12/03/202...
7	/resources/dialog-polyfill/dialog-polyfill.js	basic	applicatio...	0	12/03/202...
8	/resources/mdl/MaterialIcons-Regular.woff2	basic	font/woff2	44,300	12/03/202...
9	/resources/mdl/material-icons.css	basic	text/css; c...	0	12/03/202...
10	/resources/mdl/material.indigo-pink.min.css	basic	text/css; c...	0	12/03/202...

Select a cache entry above to preview

Fonctionnement de fetch



Fichier sw.js

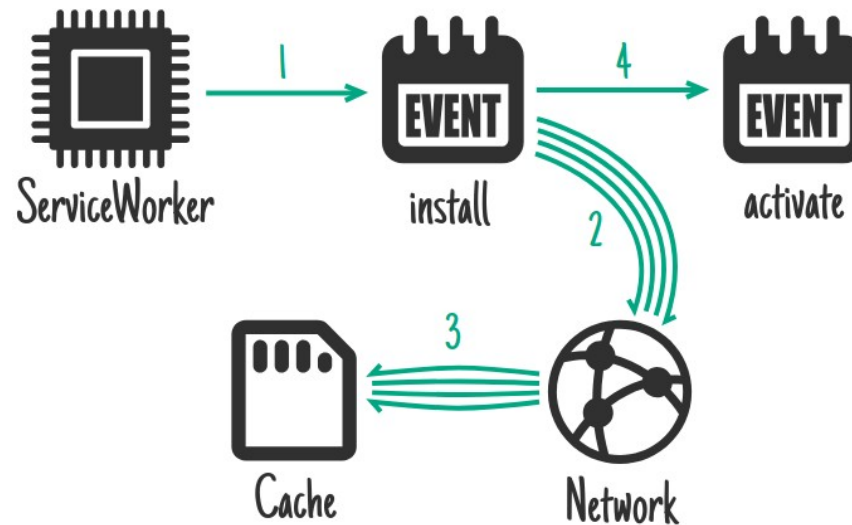
Fichier de configuration du service worker. Indique les actions et événements auxquels est sensible le service worker.

```
const CACHE_NAME = "psb_test1";  
// attention aux fichiers ne mettre que ceux qui existent  
// sinon ca plante n'arrive pas faire la synchro  
const assetToCache = [  
    // liste des fichiers à transférer  
];  
  
// Ecouteurs spécifiques  
self.addEventListener("install", function(event) {  
    """"  
});  
self.addEventListener("activate", function(event) {  
    """"  
});  
self.addEventListener("fetch", function(event) {  
    """"  
});  
self.addEventListener("push", function(event) {  
    """"  
});
```

Fonctionnement SW

On install - as a dependency

<https://iainarchibald.com/2014/offline-cookbook/>



Idéal pour les fichiers : css, images, js, templates et tout ce qui peut être considéré static pour un site

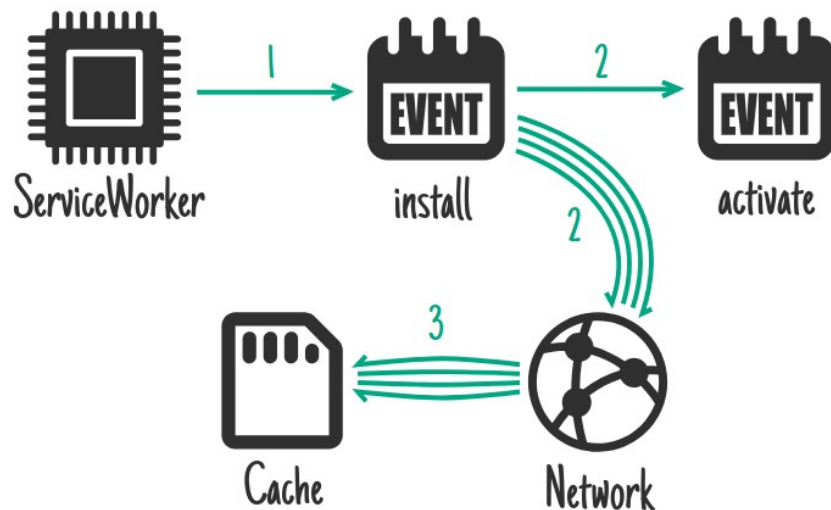
Code :

```
self.addEventListener('install', (event) => {
  event.waitUntil(async function() {
    const cache = await caches.open('mysite-static-v3');
    await cache.addAll([
      '/css/whatever-v3.css',
      '/css/imgs/sprites-v6.png',
      '/css/fonts/whatever-v8.woff',
      '/js/all-min-v4.js'
      // etc
    ]);
  })();
});
```

On passe en mode activate que si la cache se passe bien

Fonctionnement SW

On install - not as a dependency



Idéal pour les grandes ressources qui ne sont pas immédiatement nécessaires c'est comme ci en effectuer les choses en deux temps

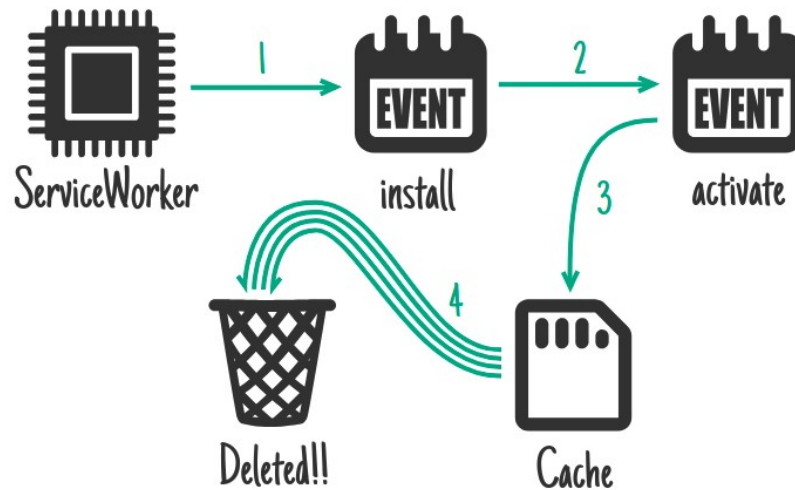
Code :

```
self.addEventListener('install', (event) => {
  event.waitUntil(async function() {
    const cache = await caches.open('mygame-core-v1');
    cache.addAll(
      // levels 11-20
    );
    await cache.addAll(
      // core assets & levels 1-10
    );
  }());
});
```

On passe en mode activate sans être dépendant du cache
Dans ce cas on va attendre que les premiers niveaux soient descendus dans le cache et les autres on n'attend pas. On pourra commencer en offline avec les niveaux inférieurs fr 1 à 10

Fonctionnement SW

On activate



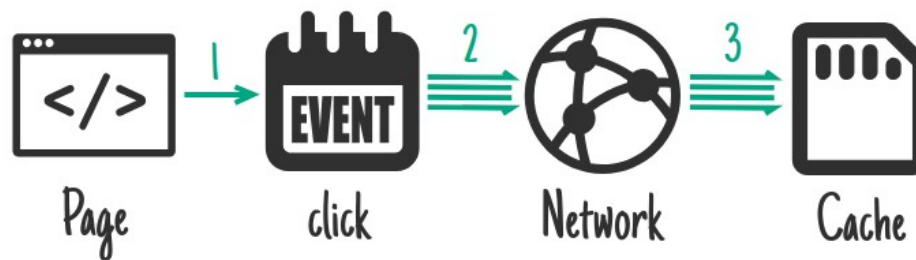
Idéal nettoyage te migration

Code :

```
self.addEventListener('activate', (event) => {
  event.waitUntil(async function() {
    const cacheNames = await caches.keys();
    await Promise.all(
      cacheNames.filter((cacheName) => {
        // Return true if you want to remove this cache,
        // but remember that caches are shared across
        // the whole origin
      }).map(cacheName => caches.delete(cacheName))
    );
  })();
});
```

Cas de figure où le service worker a été installé, et que l'on veut nettoyer les versions précédentes du cache et on pourra conserver le dernier cache/

Fonctionnement SW On user interaction



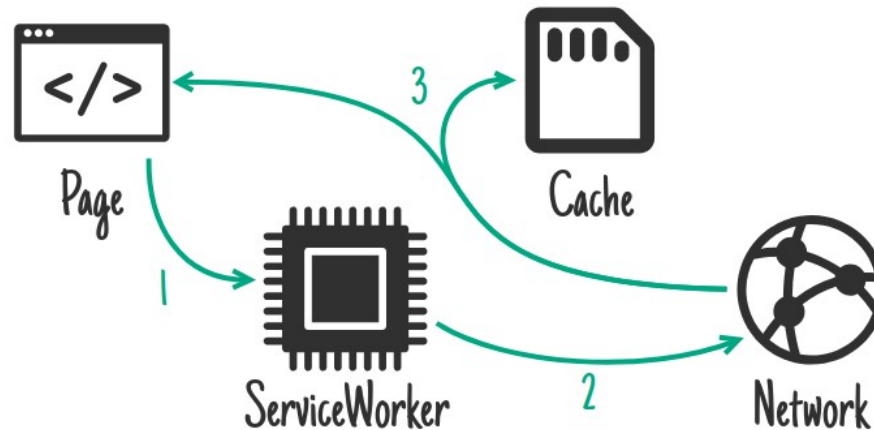
Ideal for: If the whole site can't be taken offline, you may allow the user to select the content they want available offline. E.g. a video on something like YouTube, an article on Wikipedia, a particular gallery on Flickr.

Code :

```
document.querySelector('.cache-article').addEventListener('click', async (event) => {  
  event.preventDefault();  
  
  const id = this.dataset.articleId;  
  const cache = await caches.open('mysite-article-' + id);  
  const response = await fetch('/get-article-urls?id=' + id);  
  const urls = await response.json();  
  await cache.addAll(urls);  
});
```

Fonctionnement SW

On network response

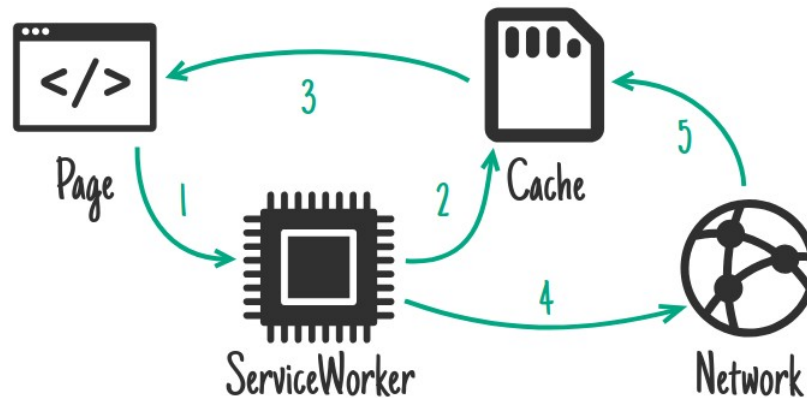


Idéal pour des ressources mises à jour fréquemment (inbox, articles)

Code :

```
self.addEventListener('fetch', (event) => {
  event.respondWith(async function() {
    const cache = await caches.open('mysite-dynamic');
    const cachedResponse = await cache.match(event.request);
    if (cachedResponse) return cachedResponse;
    const networkResponse = await fetch(event.request);
    event.waitUntil(
      cache.put(event.request, networkResponse.clone())
    );
    return networkResponse;
  }());
});
```

Fonctionnement SW Stale-While-revalidate



Ideal for: Frequently updating resources where having the very latest version is non-essential. Avatars can fall into this category.

On répond rapidement à l'utilisateur et on en profite pour réactualiser le contenu du cache.

```
Code :
self.addEventListener('fetch', (event) => {
  event.respondWith(async function() {
    const cache = await caches.open('mysite-dynamic');
    const cachedResponse = await cache.match(event.request);
    const networkResponsePromise = fetch(event.request);
    event.waitUntil(async function() {
      const networkResponse = await networkResponsePromise;
      await cache.put(event.request, networkResponse.clone());
    }());
    // Returned the cached response if we have one, otherwise return the network response.
    return cachedResponse || networkResponsePromise;
  }());
});
```

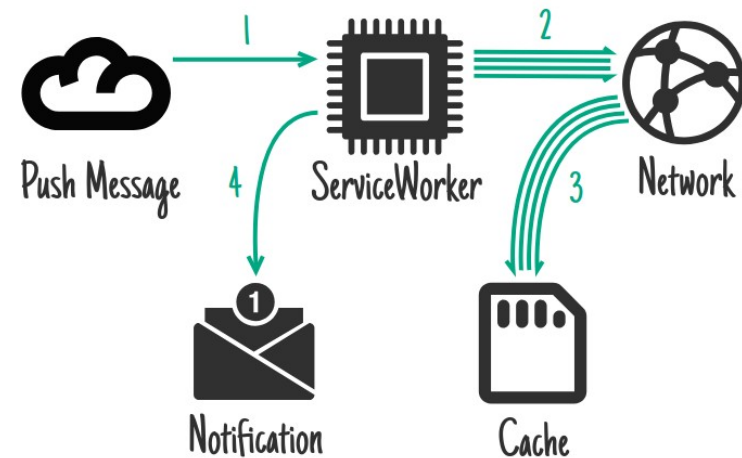
Fonctionnement SW

On push message

Code :

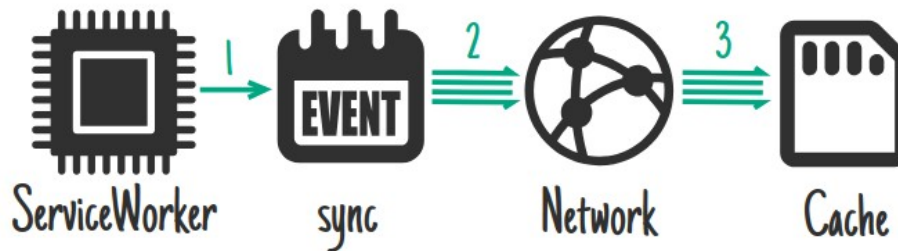
```
self.addEventListener('push', (event) => {
  if (event.data.text() == 'new-email') {
    event.waitUntil(async function() {
      const cache = await caches.open('mysite-dynamic');
      const response = await fetch('/inbox.json');
      await cache.put('/inbox.json', response.clone());
      const emails = await response.json();
      registration.showNotification("New email", {
        body: "From " + emails[0].from.name
        tag: "new-email"
      });
    });
  }
});
```

```
self.addEventListener('notificationclick', function(event) {
  if (event.notification.tag == 'new-email') {
    // Assume that all of the resources needed to render
    // /inbox/ have previously been cached, e.g. as part
    // of the install handler.
    new WindowClient('/inbox/');
  }
});
```



Ideal for: Content relating to a notification, such as a chat message, a breaking news story, or an email. Also infrequently changing content that benefits from immediate sync, such as a todo list update or a calendar alteration.

Fonctionnement SW On background-sync



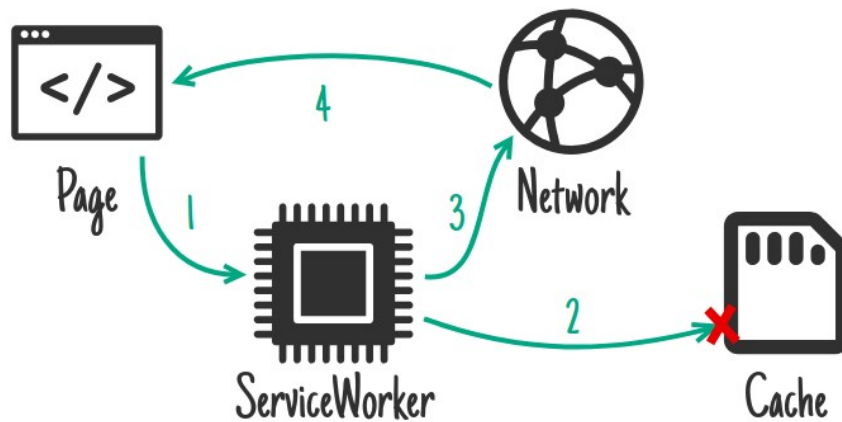
Ideal for: Non-urgent updates, especially those that happen so regularly that a push message per update would be too frequent, such as social timelines or news articles.

Code :

```
self.addEventListener('sync', (event) => {  
  if (event.id === 'update-leaderboard') {  
    event.waitUntil(async function() {  
      const cache = await caches.open('mygame-dynamic');  
      await cache.add('/leaderboard.json');  
    }());  
  }  
});
```

Fonctionnement SW

Cache, falling back to network



Code :

```
self.addEventListener('fetch', (event) => {  
  event.respondWith(async function() {  
    const response = await caches.match(event.request);  
    return response || fetch(event.request);  
  })();  
});
```

Ideal for: If you're building offline-first, this is how you'll handle the majority of requests. Other patterns will be exceptions based on the incoming request.

Fonctionnement SW

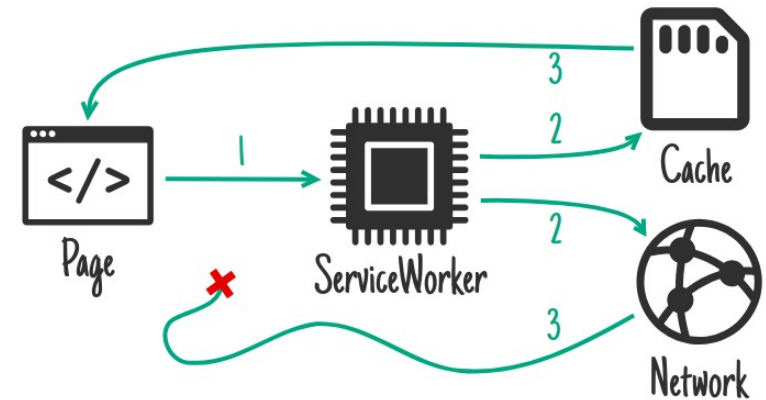
Cache & network race

Code :

```
// Promise.race is no good to us because it rejects if  
// a promise rejects before fulfilling. Let's make a proper  
// race function:
```

```
function promiseAny(promises) {  
  return new Promise((resolve, reject) => {  
    // make sure promises are all promises  
    promises = promises.map(p => Promise.resolve(p));  
    // resolve this promise as soon as one resolves  
    promises.forEach(p => p.then(resolve));  
    // reject if all promises reject  
    promises.reduce((a, b) => a.catch(() => b))  
      .catch(() => reject(Error("All failed")));  
  });  
};
```

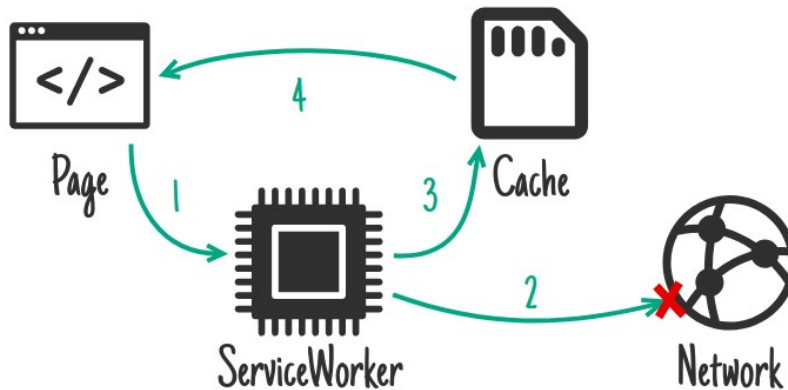
```
self.addEventListener('fetch', (event) => {  
  event.respondWith(  
    promiseAny([  
      caches.match(event.request),  
      fetch(event.request)  
    ])  
  );  
});
```



Ideal for: Small
assets where you're
chasing
performance on
devices with slow
disk access.

Fonctionnement SW

Network falling back to cache



Ideal for: A quick-fix for resources that update frequently, outside of the "version" of the site. E.g. articles, avatars, social media timelines, game leader boards.

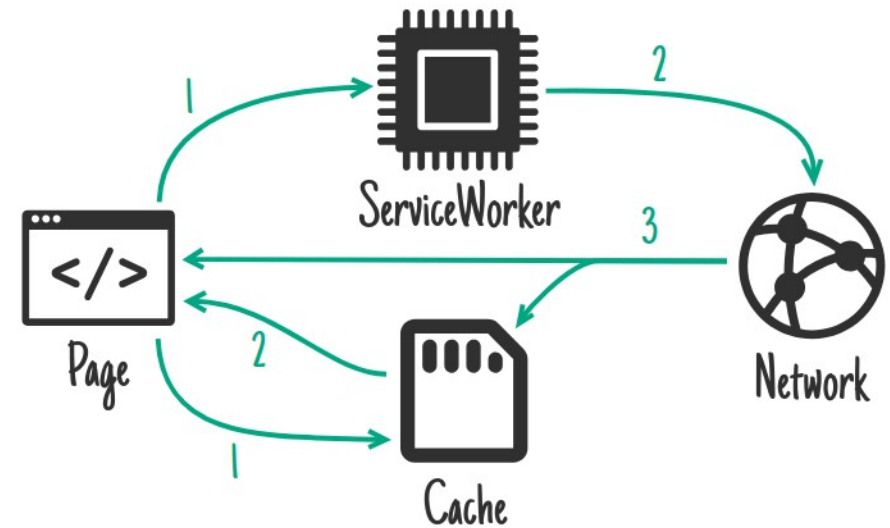
Code :

```
self.addEventListener('fetch', (event) => {  
  event.respondWith(async function() {  
    try {  
      return await fetch(event.request);  
    } catch (err) {  
      return caches.match(event.request);  
    }  
  })();  
});
```


Fonctionnement SW Cache then network

Code :

```
async function update() {  
  // Start the network request as soon as possible.  
  const networkPromise = fetch('/data.json');  
  startSpinner();  
  const cachedResponse = await caches.match('/data.json');  
  if (cachedResponse) await displayUpdate(cachedResponse);  
  try {  
    const networkResponse = await networkPromise;  
    const cache = await caches.open('mysite-dynamic');  
    cache.put('/data.json', networkResponse.clone());  
    await displayUpdate(networkResponse);  
  } catch (err) {  
    // Maybe report a lack of connectivity to the user.  
  }  
  stopSpinner();  
  const networkResponse = await networkPromise;  
}  
  
async function displayUpdate(response) {  
  const data = await response.json();  
  updatePage(data);  
}
```



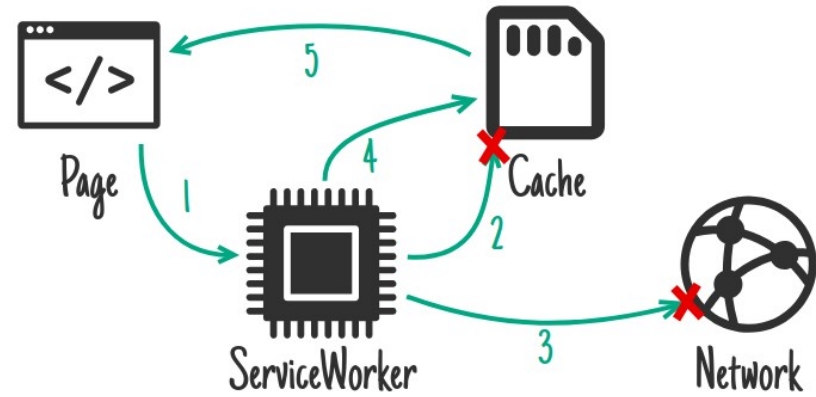
Ideal for: Content that updates frequently. E.g. articles, social media timelines, game leaderboards.

Fonctionnement SW

Generic fallback

Code :

```
self.addEventListener('fetch', (event) => {  
  event.respondWith(async function() {  
    // Try the cache  
    const cachedResponse = await caches.match(event.request);  
    if (cachedResponse) return cachedResponse;  
    try {  
      // Fall back to network  
      return await fetch(event.request);  
    } catch (err) {  
      // If both fail, show a generic fallback:  
      return caches.match('/offline.html');  
      // However, in reality you'd have many different  
      // fallbacks, depending on URL & headers.  
      // Eg, a fallback silhouette image for avatars.  
    }  
  })();  
});
```



Ideal for: Secondary imagery such as avatars, failed POST requests, "Unavailable while offline" page..

API PostMessage

Autorise des envois de message cross-origin sécurisés entre window objects, entre une page et un pop-up entre des pages et un iframe

Pour envoyer un message :

`window.postMessage()` :

`window.addEventListener ("message",callback)`

La syntaxe de la méthode

`targetwindow.postMessage(message,targetOrigin,[transfer]) ;`

targetwindow : handle de la fenêtre d'envoi

message: contenu du message

targetOrigin : handle de l'objet cible

transfer :

API Post

<https://medium.com/javascript-in-plain-english/javascript-and-window-postmessage-a60c8f6adea9>

Mise en oeuvre dans le cadre d'une application

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <meta charset="utf-8" />
<script>
var childwin;
const childname = "popup";
function openChild() {
childwin = window.open('Page2.html', childname, 'height=300px, width=500px');
}
function sendMessage(){
  let msg={pName : "Bob", pAge: "35"};
  // In production, DO NOT use '*', use toe target domain
  childwin.postMessage(msg,'*')// childwin is the targetWindow
  childwin.focus();
}
</script>
</head>
<body>
  <form>
    <fieldset>
      <input type='button' id='btnopen' value='Open child' onclick='openChild();' />
      <input type='button' id='btnSendMsg' value='Send Message' onclick='sendMessage();' />
    </fieldset>
  </form>
</body>
</html>
```

API postMessage

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <meta charset="utf-8" />
  <script>
    window.addEventListener("message", (event)=>{
      let txt=document.querySelector('#txtMsg');
      txt.value=`Name is ${event.data.pName} Age is  ${event.data.pAge}` ;
    });
    function closeMe() {
      try {
        window.close();
      } catch (e) { console.log(e) }
      try {
        self.close();
      } catch (e) { console.log(e) }
    }
  </script>
</head>
  <body>
    <form>
      <h1>Recipient of postMessage</h1>
      <fieldset>
        <input type='text' id='txtMsg' />
        <input type='button' id='btnCloseMe' value='Close me' onclick='closeMe();' />
      </fieldset>
    </form>
  </body>
</html>
```

On peut également le tester directement dans la console

```
window.addEventListener("message",(msg)=>{
  console.log("reception message",msg.data);
});
```

API IndexedBD

https://developer.mozilla.org/fr/docs/Web/API/API_IndexedDB

Est un moyen de stocker des données de manière persistante dans un navigateur. Cela nous permet de créer des applications web avec de riches possibilités de requêtes indépendantes de la disponibilité du réseau puisque vos applications peuvent fonctionner en ligne et hors ligne.

Seule l'application qui a créé la base de données peut y accéder.

API IndexedDB

https://developer.mozilla.org/fr/docs/Web/API/API_IndexedDB

indexedDB : les bases de données IndexedDB stockent des paires clé-valeurs

indexedDB : est construit autour d'un modèle de base de données transactionnelle

The indexedDB API est principalement asynchrone

indexedDB utilise de nombreuses requêtes

indexedDB utilise les événements DOM pour vous informer quand les résultats sont disponibles.

indexedDB est orienté objet

indexedDB n'utilise pas le langage SQL (système NoSQL)

indexedDB adhère au principe de same-origin policy (même origine).

API IndexedBD

https://developer.mozilla.org/fr/docs/Web/API/API_IndexedDB

```
window.indexedDB= window.indexedDB ||  
window.mozIndexedDB || window.webkitIndexedDB ||  
window.msIndexedDB;
```

```
window.IDBTransaction=window.IDBTransaction ||  
window.webkitIDBTransaction || window.msIDBTransaction;
```

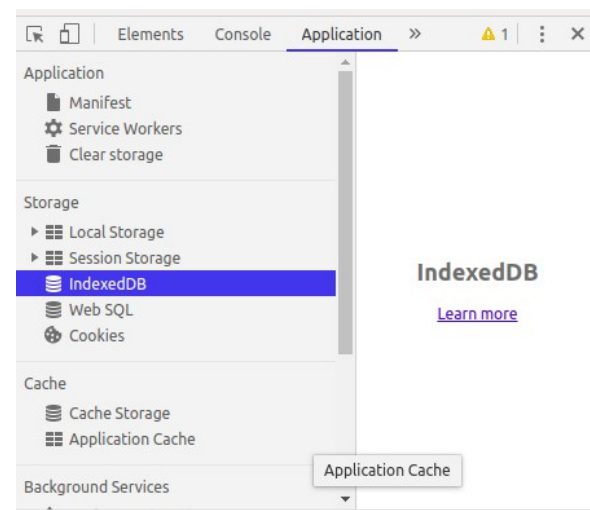
```
window.IDBkeyRange= window.IDBKeyRange ||  
window.webkitIDBKeyRange || window.msIDBKeyRange ;
```


API IndexedDB

https://developer.mozilla.org/fr/docs/Web/API/API_IndexedDB

Plusieurs implémentations en fonction du fournisseur de navigateur. Existence de préfixe en fonction de la base sur laquelle repose le navigateur : base webkit ou gecko (moz). Une fois consensus trouvé, le préfixe saute comme IE, Firefox, Chrome.

```
> window.indexedDB
< IDBFactory {}
  __proto__: IDBFactory
  ▶ open: f open()
  ▶ deleteDatabase: f deleteDatabase()
  ▶ cmp: f cmp()
  ▶ databases: f databases()
  ▶ constructor: f IDBFactory()
    Symbol(Symbol.toStringTag): "IDBFactory"
  ▶ proto: Object
```



Gestion des erreurs

```
request.onerror=function(event) {  
}  
request.onsuccess=function(event) {  
}
```

Illustration et démonstration

Vérification comptabilité navigateur

Pour vérifier si le navigateur supporte

```
if ( !window.indexedDB) {  
    window.alert("Votre navigateur ne supporte pas indexedDB") ;  
}
```

Ouverture d'une base

```
var request=window.indexedDB.open("PSBV1",3) ;
```

==> retourne un objet IDBOpenRequest

IndexedBD

Exemple d'instructions

Ouverture d'une base de données

```
function openDb() {  
    console.log("openDb ...");  
    var req = indexedDB.open(DB_NAME, DB_VERSION);  
    req.onsuccess = function (evt) {  
        db = this.result;  
        console.log("openDb DONE");  
    };  
    req.onerror = function (evt) {  
        console.error("openDb:", evt.target.errorCode);  
    };  
  
    req.onupgradeneeded = function (evt) {  
        console.log("openDb.onupgradeneeded");  
        var store = evt.currentTarget.result.createObjectStore(  
            DB_STORE_NAME, { keyPath: 'id', autoIncrement: true });  
  
        store.createIndex('biblioid', 'biblioid', { unique: true });  
        store.createIndex('title', 'title', { unique: false });  
        store.createIndex('year', 'year', { unique: false });  
    };  
}
```

Quelques méthodes

- result
- onsuccess
- onerror
- onupgradeneeded
- onblocked
- transaction
- source

Une mise en oeuvre shopping list

Dans le cadre d'un TDs fournis sous github , veuillez effectuer la mise en œuvre de cette application. Une fois la mise en œuvre dans l'état, on souhaite à partir de cette application, en décliner une autre qui permettra de faire de la saisie de publications sur les mêmes outils.

Une principale évolution PWA est la capacité à travailler offline aussi bien qu'online.

Hoodie est un backend JavaScript pour le travail offline d'une application web. il fournit des API au frontend pour autoriser et gérer des données et ajouter des autorisations aux utilisateurs. Il stocke des données locales sur son device et dès qu'il "retrouve" le réseau, synchronise avec le serveur et résout les conflits de data. il utilise couchDB sur le client et couchDB et hapi sur le serveur.

Babel

[https://en.wikipedia.org/wiki/Babel_\(transcompiler\)](https://en.wikipedia.org/wiki/Babel_(transcompiler))

Babel is a free and open-source JavaScript transcompiler that is mainly used to convert ECMAScript 2015+ (ES6+) code into a backwards compatible version of JavaScript that can be run by older JavaScript engines. Babel is a popular tool for using the newest features of the JavaScript programming language.

Voir site <http://node.green> pour compatibilité ES6

Conclusion

Les PWA présentent donc l'avantage d'atteindre les appareils mobiles sans développements trop complexes et tout en conservant une version desktop en phase avec les normes actuelles. Cette solution peut également être rapidement mise en œuvre si le besoin est bien identifié. De plus, le projet est porté par Google et les frameworks front-end populaires intègrent les concepts liés aux PWA.

En revanche, mon conseil est de limiter les PWA à des besoins simples qui ne requièrent pas l'accès à des fonctionnalités matérielles de l'appareil car, dans la plupart des cas, il sera impossible d'y accéder.

Pour aller plus loin, n'hésitez pas à consulter cet article : <https://www.pwastats.com/> qui proposent quelques "success stories" autour des PWA dont notamment celles de Uber, Pinterest et Tinder.

Plus d'infos

*** Web App Manifest : le WAM est un fichier JSON qui identifie l'application et définit des propriétés clefs comme les couleurs, les icônes à utiliser... Le but du manifeste est d'installer des applications sur l'écran d'accueil d'un appareil, offrant aux utilisateurs un accès plus rapide et une expérience plus riche. Plus de détails sur le Web App Manifest [ici](#).**

Outils de développement de débogage

En fonction des navigateurs

chrome://serviceworker-internals/

Chromium : chrome://inspect/#service-workers

firefox : about:serviceworkers

Voir

<https://developers.google.com/web/tools/lighthouse>

Autres aspects à couvrir

La synchronisation des données en arrière-plan ;
Répondre aux requêtes depuis d'autres origines ;
Recevoir des données lourdes à calculer comme des données de géolocalisation ou de gyroscope, afin que plusieurs pages puissent y accéder sans refaire les calculs ;
Compilation et gestion de dépendances de CoffeeScript, less, des modules CJS/AMD, etc. pour les besoins de développement ;
Gestion et utilisation de services en d'arrière-plan ;
Templates personnalisés en fonction de patterns d'URL ;
Amélioration des performances, par exemple en pré-chargeant les ressources que l'utilisateur pourrait utiliser plus tard telles que les prochaines photos d'un album.

Liens utiles

Quelques liens

Enfin voici quelques ressources pour aller plus loin.

Introduction to Service Worker, Matt Gaunt : <http://www.html5rocks.com/en/tutorials/service-worker/introduction/>

Service Worker Cookbook, Mozilla : <https://serviceworke.rs/>

Offline Recipes for Service Workers, David Walsh : <https://hacks.mozilla.org/2015/11/offline-service-workers/>

Service Worker API, Mozilla Developer Network :

https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

Using Service Workers, MDN :

https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Service Worker test repository, MDN : <https://github.com/mdn/sw-test>

Can I Use Service Workers?, Alexis Deveria : <http://caniuse.com/#search=service%20worker>

Liens utiles

Quelques liens

Getting started with Service Workers, Ritesh Kumar : <http://www.sitepoint.com/getting-started-with-service-workers/>

Is Service Worker ready?, Jake Archibald : <https://jakearchibald.github.io/isserviceworkerready/resources.html>
<https://jakearchibald.com/2014/offline-cookbook/>

Une nouvelle architecture pour nos applications web mobiles, Julien Wajsberg :
<http://www.24joursdeweb.fr/2015/une-nouvelle-architecture-pour-nos-applications-web-mobiles/>

Background Sync, Web Incubator Community Group :
<https://github.com/WICG/BackgroundSync/blob/master/explainer.md>

Cache, MDN : <https://developer.mozilla.org/en-US/docs/Web/API/Cache>

FetchEvent, MDN : <https://developer.mozilla.org/en-US/docs/Web/API/FetchEvent>

postMessage, MDN : <https://developer.mozilla.org/en-US/docs/Web/API/Client/postMessage>

Using Service Workers, MDN :
https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Exemple d'application interactive

Socket.io

/home/ystroppa/auto_mathon/socketIO/socket.io/examples

Exemple de whiteboard : tableau blanc interactif

- index.js
- node_modules
- package.json
- public
- README.md

- ./public/
 - index.html
 - main.js
 - style.css



Bonjour
à tous!
Bon courage

Démarrer l'application à l'aide de la commande
`#node index.js`
ou
`#npm start`

Exemple d'application interactive

Socket.io /home/ystroppa/auto_mathon/socketIO/socket.io/examples

Exemple de whiteboard : tableau blanc interactif

```
//index.js
const express = require('express');
const app = express();
const http = require('http').Server(app);
const io = require('socket.io')(http);
const port = process.env.PORT || 3000;

app.use(express.static(__dirname + '/public'));

function onConnection(socket){
  socket.on('drawing', (data) =>
    socket.broadcast.emit('drawing', data));
}

io.on('connection', onConnection);

http.listen(port, () => console.log('listening on
port ' + port));
```

```
{
  "name": "whiteboard",
  "version": "1.0.0",
  "description": "A simple collaborative whiteboard
using socket.io",
  "main": "index.js",
  "keywords": [
    "socket.io",
    "whiteboard"
  ],
  "dependencies": {
    "express": "4.9.x",
    "socket.io": "latest"
  },
  "scripts": {
    "start": "node index"
  },
  "author": "Damien Arrachequesne",
  "license": "MIT"
}
```

Exemple d'application interactive

Socket.io /home/ystroppa/auto_mathon/socketIO/socket.io/examples

Exemple de whiteboard : tableau blanc interactif

```
//index.html
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Socket.IO whiteboard</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <canvas class="whiteboard" ></canvas>

  <div class="colors">
    <div class="color black"></div>
    <div class="color red"></div>
    <div class="color green"></div>
    <div class="color blue"></div>
    <div class="color yellow"></div>
  </div>

  <script src="/socket.io/socket.io.js"></script>
  <script src="/main.js"></script>
</body>
</html>
```

```
//main.js
// initialisation de la socket
var socket=io() ;

// réception de message
socket.on("drawing", OnDrawingEvent) ;

// envoi de message
socket.emit('drawing', {
  x0: x0 / w,
  y0: y0 / h,
  x1: x1 / w,
  y1: y1 / h,
  color: color
});
```

Exemple d'application interactive

Socket.io /home/ystroppa/auto_mathon/socketIO/socket.io/examples

Chat voir exemple fourni avec socket.io sous nodejs

Travail à faire et à rendre (Par 3)

Choisir une application exemple sur le site de <https://pwa.rocks> et fournir un rapport de retro conception de l'application

Analyse à faire :

Services workers (les modes utilisés)

Cache : localStorage ou/et indexedDB