

Formation Programmation Multiplateformes (Approche PWA)

Rossi Oddet

Table des matières

Approche Progressive	1
Définition	1
Techniques	1
Le cas Twitter	3
Media queries	5
Fetch API	6
Support navigateur	6
Exemple de requête :	6
Objet Response	6
Exemple avec un objet JSON	7
Exemple avec une image	7
Personnaliser la requête	8
Pour aller plus loin	9
Service Worker	10
Notion de Web Worker	10
Service Worker	11
Web App Manifest	15
Inclure le fichier dans une page HTML	15
Exemple de fichier Manifest	16
Propriétés	16
Événement beforeinstallprompt	19
Outils PWA	20
Lighthouse	20
WorkBox	20
TP #01-mini-pwa	22
Partie 1 - Service Worker	22
Partie 2 - Web App Manifest	32
Frameworks UI	34
Pourquoi utiliser un framework ?	34
Quelques frameworks UI ?	34
Ionic	34
Inclure Ionic Core	36
ion-app	36
ion-header , ion-footer , ion-content	37
TP #02-ionic-core	40

Approche Progressive

L'approche progressive vise à créer une application web avec des techniques classiques (HTML, JavaScript et CSS) qui sur mobile s'adapte aux règles de celui-ci.

Définition

En 2015, Alex Russell (ingénieur chez Google) va écrire [un article](#) listant les caractéristiques d'une application web progressive (Progressive Web Apps en sigle PWA) :

- **Adaptative** : l'application s'adapte à toutes les formes.
- **Indépendante de la connectivité** : l'application supporte un mode hors-ligne. Elle fournit des services même sans connexion à un réseau.
- **Interagit comme une application** : la navigation et la présentation de l'information suivent les codes de la mobilité.
- **Actualisée** : l'application est toujours à jour et se met à jour de manière transparente.
- **Sûr** : l'application est servie avec TLS (un prérequis du Service Worker).
- **Découvrable** : l'application peut être trouvée via les moteurs de recherche.
- **Re-engageante** : les utilisateurs sont plus susceptibles de réutiliser leurs applications avec des fonctionnalités telle que le *push*.
- **Installable** : l'application est identifiable comme une application sur mobile.
- **Liens** : Utiliser l'URI pour conserver l'état d'une application.



L'application n'est pas déployée dans un store quelconque, il s'agit d'un site internet.

Techniques

Google propose [une checklist](#) en plusieurs parties.

Voici les éléments jugés essentiels dans une application web progressive.

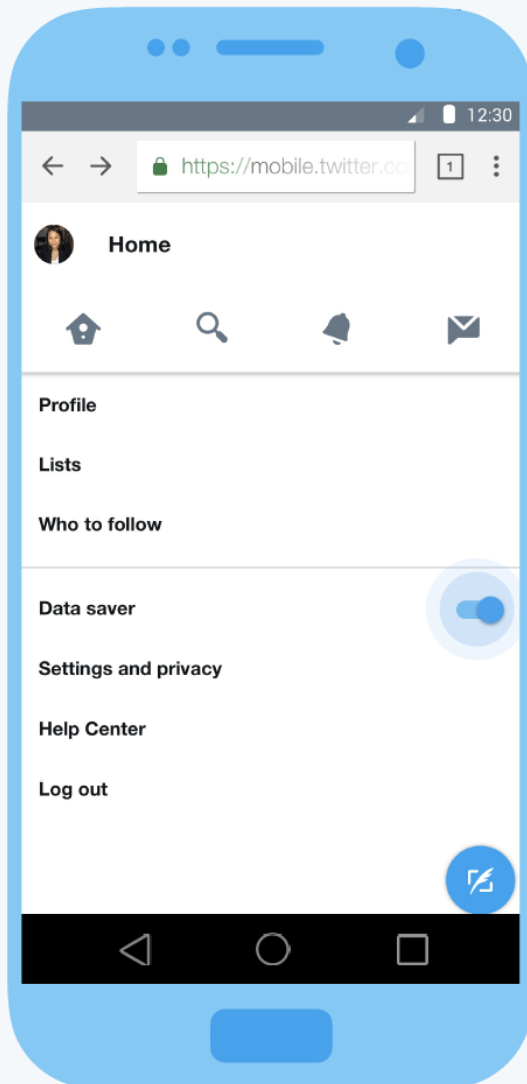
	Critère	Technique
01	Le site est servi en HTTPS	Activer HTTPS sur votre serveur. Utiliser par exemple letsencrypt.org .
02	Le site s'adapte aux appareils mobiles	Mettre en place un Responsive Web Design via les media queries
03	Toutes les urls du site peuvent fonctionner en mode hors-ligne	Utiliser les Services Workers

	Critère	Technique
04	Fournir des métadonnées pour que le site soit installable	Ajouter un fichier Web App Manifest
05	Premier chargement très rapide en 3G	Optimiser les performances et mettre en cache.
06	Le site marche sur les principaux navigateurs (Chrome, Edge, Firefox et Safari)	Tester l'application sur différents navigateurs
07	Les transitions entre pages sont fluides	Ne pas recharger entièrement la nouvelle page

Le cas Twitter

En 2017, Twitter lance une nouvelle version de son application web dédiée aux mobiles : **Twitter Lite**.

1 Tap your profile picture. Turn on the new “Data saver” setting.



2 Images will now appear blurred for preview. Tap on any image to load and view.



Les objectifs recherchés :

- Temps de chargement plus rapide
- Moins de données consommées
- Meilleur accrochage des utilisateurs

Sources :

- [Article de lancement Twitter Lite](#)
- [Technologies utilisées par Twitter Lite](#)

Media queries

Media queries est un module CSS3 permettant d'adapter le contenu d'une page web aux caractéristiques de l'appareil de l'utilisateur (par exemple, écran de téléphone intelligent versus écran d'ordinateur). De telles adaptations sont devenues une norme W3C recommandée en juin 2012. Il s'agit de la pierre angulaire des sites web adaptatifs.

Exemple :

```
@media screen and (min-width:500px) {  
  
    /* ici une règle CSS */  
  
}
```

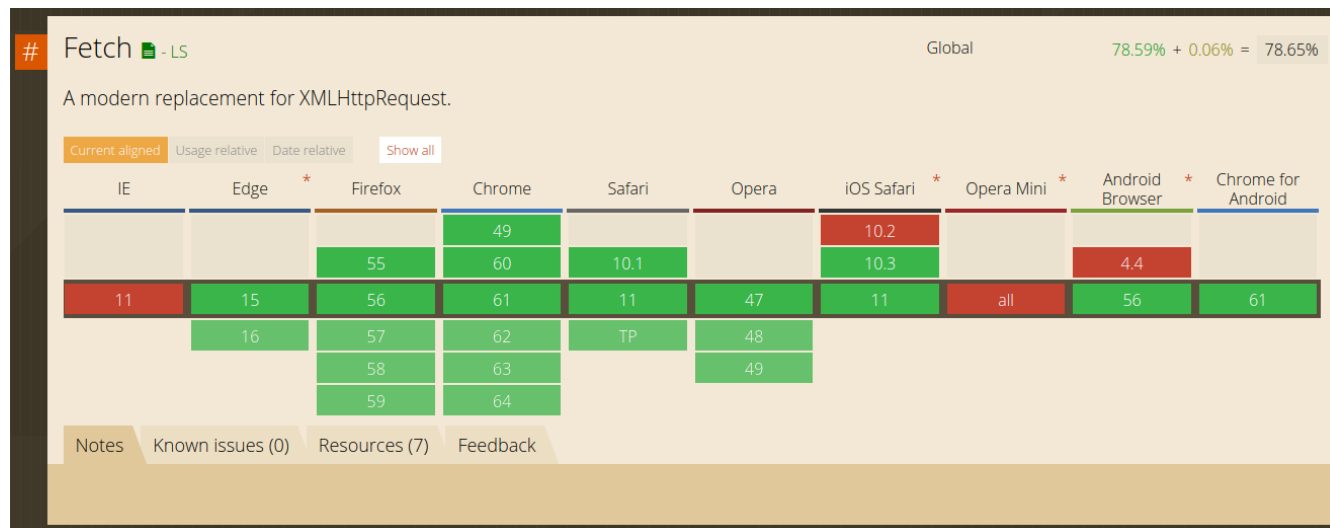
Source : https://fr.wikipedia.org/wiki/Media_queries

Pour aller plus loin : https://developer.mozilla.org/fr/docs/Web/CSS/Requ%C3%AAtes_m%C3%A9dia/Utiliser_les_Media_queries

Fetch API

Fetch API est le remplaçant moderne de *XMLHttpRequest*.

Support navigateur



```
if ('fetch' in window) {  
  // ok  
}
```

Il est possible d'utiliser un *polyfill* (<https://github.com/github/fetch>).

Exemple de requête :

```
fetch('data/data.json')  
  .then(function(response) {  
    // traiter la réponse  
  })  
  .catch(function(error) {  
    console.log('Il semble avoir un soucis...', error);  
  });
```

Objet Response

L'objet *Response* représente la réponse de la requête.

Statut de la réponse

Pour connaître le statut de la réponse :


```
response.status // code de la réponse

response.statusText // libellé du code de la réponse

response.ok // si le code de la réponse est compris entre 200 et 299.
```

Lire le contenu de la réponse

```
response.json() // récupérer un objet JS

response.blob() // récupérer une ressource binaire (images par exemple)

response.text() // récupérer du texte
```

Exemple avec un objet JSON

```
fetch('data/superdata.json')
  .then(function(response) {
    if (!response.ok) {
      throw Error(response.statusText);
    }
    // lecture du corps de la réponse en tant que JSON.
    return response.json();
  })
  .then(function(responseAsJson) {
    // traitement de l'objet
    console.log(responseAsJson);
  })
  .catch(function(error) {
    console.log('Une erreur est survenue : ', error);
  });
```

Exemple avec une image

```

fetch('data/superimage.png')
.then(function(response) {
  // lecture du corps de la réponse en tant que Blob.
  return response.blob();
})
.then(function(responseAsBlob) {
  // dans l'hypothèse qu'une <div> existe avec l'id = 'container'
  var container = document.getElementById('container');
  var imgElem = document.createElement('img');
  container.appendChild(imgElem);
  var imgUrl = URL.createObjectURL(responseAsBlob);
  imgElem.src = imgUrl;
})
.catch(function(error) {
  console.log('Une erreur est survenue : ', error);
});

```

Personnaliser la requête

- L'attribut *method* permet de spécifier la méthode *HTTP*.

```

fetch('data/supertexte.txt', {
  method: 'HEAD'
})

```

- L'attribut *body* spécifie le corps de la requête.

```

fetch('data/supercommentaire', {
  method: 'POST',
  body: 'title=cool&message=voyagecool'
})

```

- L'attribut *headers* permet de spécifier des entêtes.

```

// POST
fetch('data/supercommentaire', {
  method: 'POST',
  headers: {
    "Content-Type": "application/json"
  },
  body: '{"data": "content"}'
})

```

Pour aller plus loin

- <https://fetch.spec.whatwg.org/>
- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

Service Worker

Notion de Web Worker

Les Web Workers fournissent un moyen simple d'exécuter des scripts dans des threads en arrière-plan d'un contenu web. Le thread worker peut réaliser des tâches sans interférer avec l'interface utilisateur. Les workers s'exécutent dans un contexte global différent du contexte de la fenêtre courante.

```
if (window.Worker) {  
  // Worker supporté !  
}
```

Instancier un Worker dédié

```
var myWorker = new Worker("worker.js");
```

Envoyer un message à un Worker

```
var myWorker = new Worker("worker.js");  
  
// Envoie d'un message depuis le script principal  
myWorker.postMessage(["nantes", "rennes", "paris"]);
```

Recevoir un message depuis un worker

Exemple de fichier `worker.js`.

```
onmessage = function(e) {  
  // réception du message  
  var workerResult = 'Result: ' + e.data.join(",");  
  
  // envoie d'un message au script principal  
  postMessage(workerResult);  
}
```

Recevoir le message d'un Worker

```
var myWorker = new Worker("worker.js");

myWorker.onmessage = function(e) {
  // réception du message
  var msg = e.data;
}
```

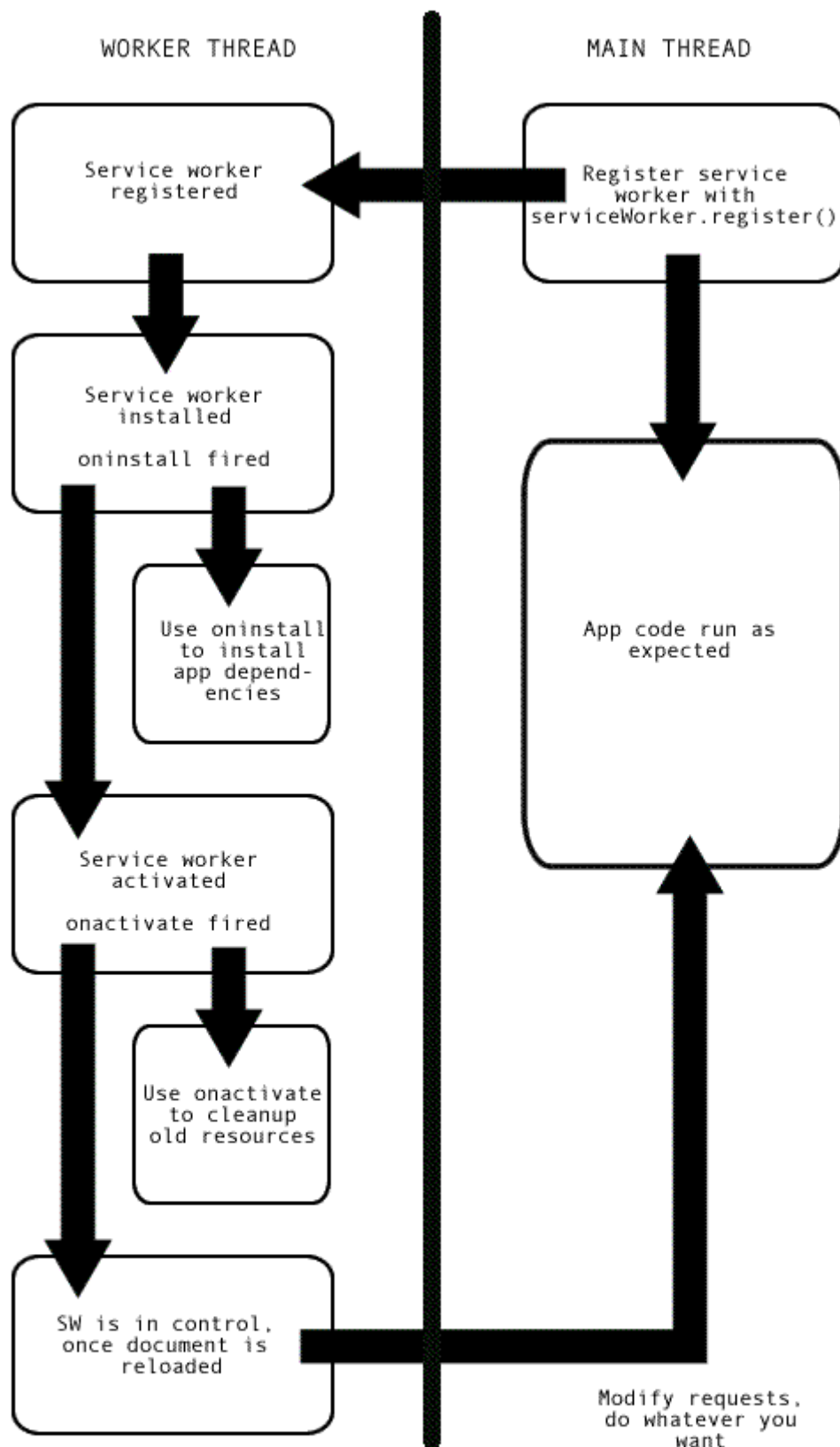
Arrêter un Worker

```
// depuis le script principal
myWorker.terminate();
```

```
// depuis un worker
close();
```

Service Worker

Un service worker est joue essentiellement le rôle d'un serveur proxy placé entre l'application web et le réseau. Il permet d'intercepter toutes les échanges entre une application web et le réseau.



Un *Service Worker* est une catégorie de *Web Worker*.

Il s'agit essentiellement d'un fichier JavaScript qui :

- s'exécute de manière séparée du thread principal du navigateur.
- est capable d'intercepter les requêtes allant vers le réseau.
- permet de mettre en cache des ressources et de les récupérer du cache.

Quelques caractéristiques :

- Un *Service Worker* est conçu pour être fondamentalement asynchrone. Les API bloquantes comme une requête *XHR synchrone* ou le *localStorage* ne peut être utilisé.
- Un *Service Worker* peut recevoir des messages depuis un serveur même si l'application n'est pas active.
- Un *Service Worker* ne peut accéder directement au DOM. La communication avec la page s'effectue via des messages.
- Un *Service Worker* fonctionne uniquement sur HTTPS.

Cycle de vie

Trois étapes du cycle de vie :

- Enregistrement
- Installation
- Activation

Enregistrement

```
if ('serviceWorker' in navigator) {  
  
    // enregistrement d'un service worker  
    navigator.serviceWorker.register('/service-worker.js')  
  
    .then(function(registration) {  
        console.log('Enregistrement Ok, le scope est :', registration.scope);  
    })  
  
    .catch(function(error) {  
        console.log('Enregistrement Ko, erreur:', error);  
    });  
}
```

Lors de l'enregistrement, il est possible de spécifier un *scope* différent :

```
// Les requêtes /off-app/, /off-app/res, ... seront contrôlées par le service worker

navigator.serviceWorker.register('/service-worker.js', {
  scope: '/off-app/'
})
```

Installation

L'installation a lieu après l'enregistrement et si le *service worker* est considéré comme nouveau par le navigateur.

Un événement *install* est déclenché dans cette phase.

```
// Traitement de l'événement 'install' du service worker
self.addEventListener('install', function(event) {
  // mise en cache des ressources
});
```

Activation

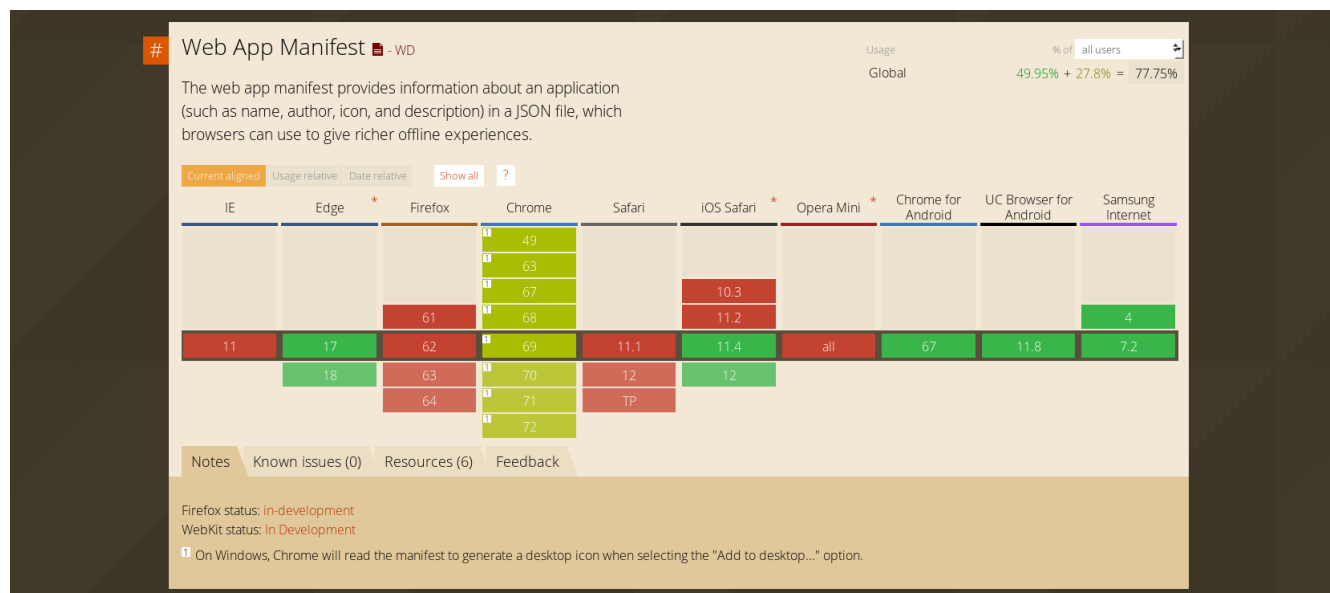
Un *service worker* va osciller entre une phase d'activation et une phase d'attente.

```
// Traitement de l'événement 'activate' du service worker
self.addEventListener('activate', function(event) {
  // ...
});
```


Web App Manifest

Le standard [Web App Manifest](#) permet de rendre installable un site web.

En septembre 2018, voici le support navigateur :



Le *Web App Manifest* fournit les informations à propos de l'application via un fichier texte JSON.

Les informations fournies peuvent être : * le nom * l'auteur * l'icône * la description * ...

Ces informations sont utilisées pour *installer* l'application sur l'écran d'accueil de l'utilisateur.

Inclure le fichier dans une page HTML

```
<!doctype html>
<html>
  <head>
    <!-- inclusion du fichier Web App Manifest -->
    <link rel="manifest" href="/manifest.webmanifest">
  </head>
  <body>
    <!-- ... --->
  </body>
</html>
```

Même si l'utilisation de l'extension `.webmanifest` est précisée dans la spécification, les navigateurs supportent généralement également l'extension `.json`.

Le fichier manifest doit être idéalement servi avec type MIME `application/manifest+json`. Les navigateurs prennent néanmoins en compte ce fichier même si ce n'est pas le cas.

Exemple de fichier Manifest

```
{
  "name": "DevFest App",
  "short_name": "DFA",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#fff",
  "description": "Une super app pour le DevFest",
  "icons": [{
    "src": "images/touch/logo48.png",
    "sizes": "48x48",
    "type": "image/png"
  }, {
    "src": "images/touch/logo72.png",
    "sizes": "72x72",
    "type": "image/png"
  }, {
    "src": "images/touch/logo96.png",
    "sizes": "96x96",
    "type": "image/png"
  }, {
    "src": "images/touch/logo144.png",
    "sizes": "144x144",
    "type": "image/png"
  }, {
    "src": "images/touch/logo168.png",
    "sizes": "168x168",
    "type": "image/png"
  }, {
    "src": "images/touch/logo192.png",
    "sizes": "192x192",
    "type": "image/png"
  }],
  "related_applications": [{
    "platform": "play",
    "url": "https://play.google.com/store/apps/details?id=devfestapp"
  }]
}
```

Propriétés

- **background_color** : définit la couleur de fond du site web. Cette propriété peut être utilisée par le navigateur pour créer une transition visuelle en douceur pendant le changement du site. (le navigateur afficherait en avance un fond avec la couleur souhaitée).
- **description** : fournit une description générale sur l'objet de l'application.
- **dir** : spécifie la direction du texte pour les propriétés **name**, **short_name** et **description**. Il peut

avoir les valeurs suivantes : `ltr` (**l**eft-**t**o-**r**ight), `rtl` (**r**ight-**t**o-**l**eft) ou `auto`. Il s'utilise généralement avec la propriété `lang`.

```
{
  "dir": "rtl",
  "lang": "ar",
  "short_name": "📱 📱 📱📱📱📱!"
}
```

- `display` : définit le mode d'affichage préféré. Les valeurs possibles sont :
 - `fullscreen` : tout l'espace disponible est utilisé (mode plein écran).
 - `standalone` : le site web ressemble à une application mobile.
 - `minimal-ui` : l'application ressemble à une application mobile mais il y a quelques éléments de navigation intégrés.
 - `browser` : le site s'ouvre dans un navigateur.
- `icons` : spécifie un tableau d'images utilisées suivant le contexte.

```
{
  "icons": [
    {
      "src": "icon/1.webp",
      "sizes": "48x48",
      "type": "image/webp"
    },
    {
      "src": "icon/2",
      "sizes": "48x48"
    },
    {
      "src": "icon/3.ico",
      "sizes": "72x72 96x96 128x128 256x256"
    },
    {
      "src": "icon/4.svg",
      "sizes": "72x72"
    }
  ]
}
```

- `lang` : spécifie la langue utilisée dans les propriétés `name` et `short_name`.

```
{
  "lang" : "fr-FR"
}
```

- **name** : fournit un nom lisible pour l'utilisateur.

```
{  
  "name": "DevFest Nantes 2018"  
}
```

- **orientation** : définit une orientation par défaut du site. Les valeurs possibles :
 - any
 - natural
 - landscape
 - landscape-primary
 - landscape-secondary
 - portrait
 - portrait-primary
 - portrait-secondary
- **prefer_related_applications** : définit un booléen qui indique à l'utilisateur que vous recommandez l'utilisation d'une application native. À utiliser uniquement si l'application offre plus de fonctionnalités que l'application web.

```
{  
  "prefer_related_applications": false  
}
```

- **related_applications** : définit un tableau d'applications natives alternatives au site internet.

```
{  
  "related_applications": [  
    {  
      "platform": "play",  
      "url": "https://play.google.com/store/apps/details?id=com.example.app1",  
      "id": "com.example.app1"  
    }, {  
      "platform": "itunes",  
      "url": "https://itunes.apple.com/app/example-app1/id123456789"  
    }  
  ]  
}
```

- **scope** : restreint la portée de l'application web progressive.

```
{  
  "scope" : "/superapp/"  
}
```

- `short_name` : définit un nom court à l'application. Ce nom court est utilisé quand il n'y a pas assez d'espace pour afficher le nom complet (`name`).

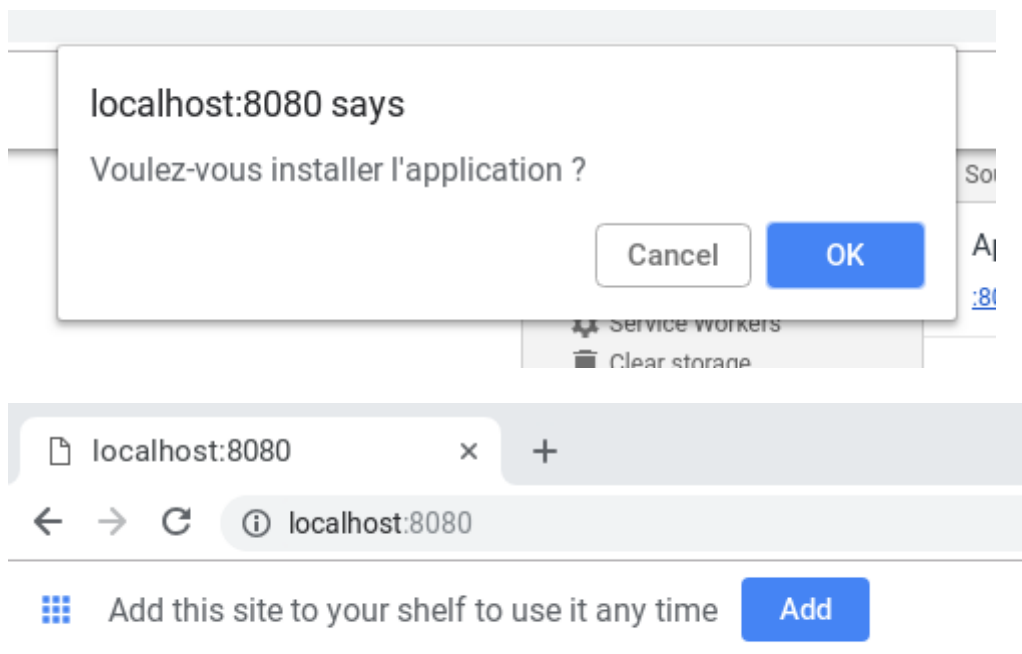
```
{  
  "short_name" : "DFN 2018"  
}
```

- `start_url` : définit l'url de démarrage de l'application.
- `theme_color` : définit la couleur du thème par défaut de l'application.

Événement `beforeinstallprompt`

L'événement `beforeinstallprompt` est émis juste avant de demander la permission d'installer l'application à l'utilisateur.

```
window.addEventListener('beforeinstallprompt', (e) => {  
  e.preventDefault();  
  
  if(confirm("Voulez-vous installer l'application ?")) {  
  
    // déclenche la demande d'installation de l'application.  
    e.prompt();  
  }  
});
```



Web App Manifest

Outils PWA

Lighthouse



Lighthouse est un outil open source d'analyse de la qualité d'une application web.

Il est accessible :

- soit via l'onglet **Audits** de Chrome DevTools.
- soit en ligne de commande
- soit via une extension Chrome

WorkBox



Workbox

Workbox est une librairie JavaScript aidant à implémenter les cas d'utilisation usuels des services workers.

TP #01-mini-pwa

Partie 1 - Service Worker

Installation

- Créer un répertoire `01-mini-pwa` :

```
mkdir 01-mini-pwa
```

- Initialiser un projet `npm` :

```
npm init -y
```

- Installer un serveur :

```
npm i -D https-localhost
```

- Compléter le fichier `package.json` pour ajouter la tâche `start` :

```
{  
  ...  
  "scripts": {  
    "start" : "serve .",  
    ...  
  },  
  ...  
}
```

- Démarrer le serveur :

```
npm start
```

Page par défaut

- Ajouter une page Web `index.html` :


```

<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Mini PWA</h1>

    <script src="app.js"></script>
  </body>
</html>

```

- Ajouter les fichiers `app.js` et `app.css`.

À ce stade, l'arborescence est la suivante :

```

/mini-pwa
  index.html
  app.js
  app.css
  package.json
  /node_modules/

```

Enregistrer un Service Worker

- Compléter le fichier `app.js` comme suit :

```

(function () {
  'use strict';

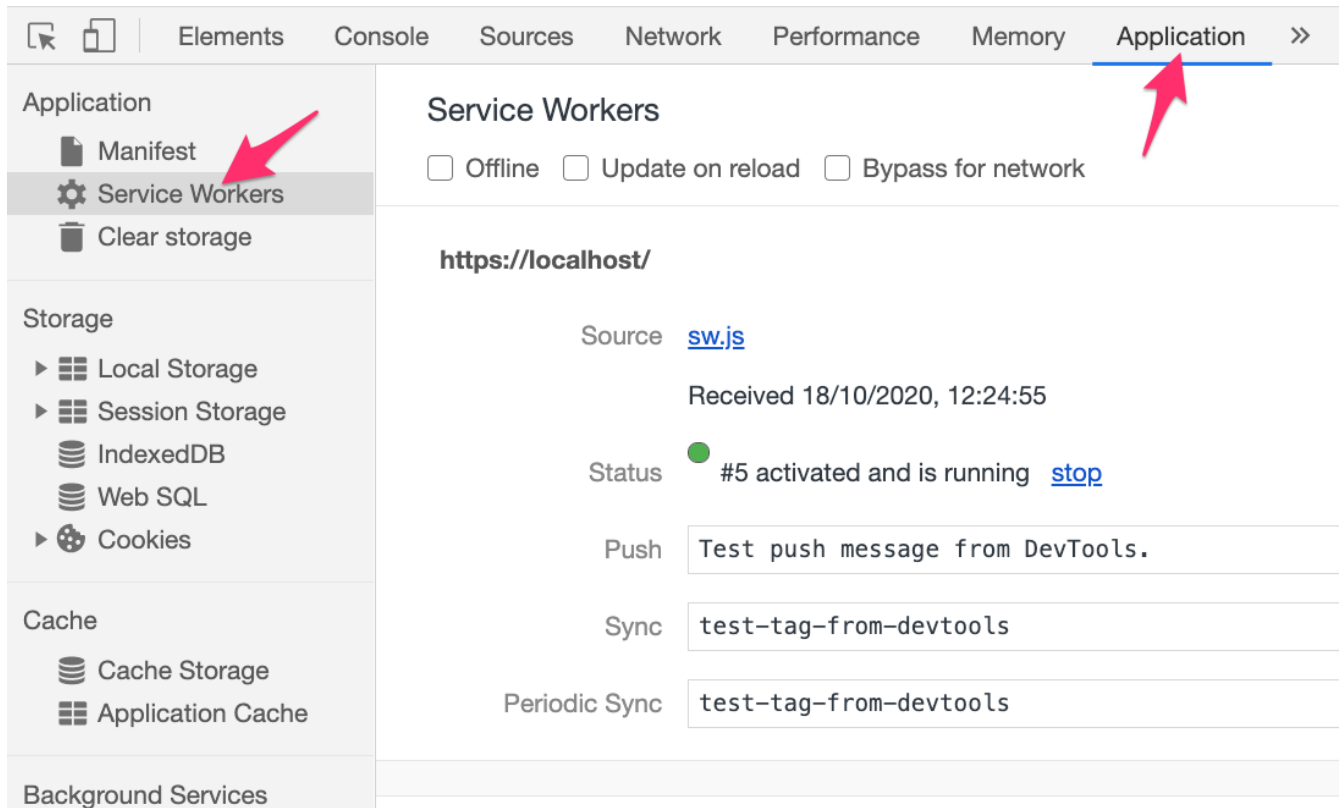
  if (!('serviceWorker' in navigator)) {
    console.log('Service worker non supporté');
    return;
  }
  navigator.serviceWorker.register('sw.js')
    .then(() => {
      console.log('Enregistrement OK');
    })
    .catch(error => {
      console.log('Enregistrement KO :', error);
    });
})();

```

- Créer un fichier `sw.js`.
- Afficher la page d'accueil. Visualiser la console du navigateur :

Enregistrement OK

- Visualiser l'installation du Service Worker dans le navigateur (Chrome : Onglet Application > Services Workers).



Cycle de vie d'un Service Worker

- Compléter le fichier `sw.js` :

```
self.addEventListener('install', event => {
  console.log('Installation du Service Worker...');
});

self.addEventListener('activate', event => {
  console.log('Activation du Service Worker...');
});
```

- Recharger la page :

Enregistrement OK
Installation du Service Worker...

- Compléter le fichier `sw.js` :

```
// ...  
  
// termine le précédent SW  
// et active immédiatement un nouveau service worker  
self.skipWaiting();
```

- Recharger la page :

```
Enregistrement OK  
Installation du Service Worker...  
Activation du Service Worker...
```

Intercepter les requêtes

- Compléter le fichier `sw.js` :

```
// ...  
  
self.addEventListener('fetch', event => {  
  console.log('Fetching:', event.request.url);  
});
```

- Recharger plusieurs fois. Vérifier que les requêtes sont bien interceptées. :

```
Fetching: http://localhost:8080/  
Fetching: http://localhost:8080/app.css  
Fetching: http://localhost:8080/app.js  
Enregistrement OK
```

Pour en savoir plus :

- Événement Fetch Event : <https://developer.mozilla.org/en-US/docs/Web/API/FetchEvent>

Mise en cache des fichiers

La mise en cache des fichiers dans un *Service Worker* se fait l'aide de *Cache API* (<https://developer.mozilla.org/en-US/docs/Web/API/Cache>).

- Modifier le service worker comme suit (modification du traitement lors de l'événement *install*) :

```

const FILES_TO_CACHE = [
  '/',
  'app.css',
  'app.js'
];

const STATIC_CACHE_NAME = 'pages-cache-v1';

self.addEventListener('install', event => {
  console.log('Installation du Service Worker...');
  console.log('Mise en cache des ressources');
  event.waitUntil(
    caches.open(STATIC_CACHE_NAME)
      .then(cache => {
        return cache.addAll(FILES_TO_CACHE);
      })
  );
});

```

- Recharger la page et vérifier le cache :

#	Request	Response
0	http://localhost:8080/	OK
1	http://localhost:8080/app.css	OK
2	http://localhost:8080/app.js	OK

Utiliser le cache

- Créer un fichier *app.no.cache.js*.
- Compléter le fichier *index.html* comme suit :

```

<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Service Worker</h1>

    <script src="app.js"></script>
    <script src="app.no.cache.js"></script> <!-- à ajouter -->
  </body>
</html>

```

- Modifier le service worker comme suit (modification du traitement lors de l'événement *fetch*) :

```

self.addEventListener('fetch', event => {
  console.log('Fetching:', event.request.url);

  event.respondWith(
    caches.match(event.request).then(response => {
      if (response) {
        console.log(event.request.url, 'servi depuis le cache');
        return response;
      }
      console.log(event.request.url, 'servi depuis le réseau');
      return fetch(event.request)
    }).catch(error => {
      console.log("oops");
    })
  );
});

```

- Recharger la page plusieurs fois :

```

...
http://localhost:8080/ servi depuis le cache
http://localhost:8080/app.css servi depuis le cache
http://localhost:8080/app.js servi depuis le cache
http://localhost:8080/app.no.cache.js servi depuis le réseau
...

```

- Modifier le titre de la page *index.html* :

```

<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <!-- modifier le texte -->
    <h1>Service Worker en cache</h1>

    <script src="app.js"></script>
    <script src="app.no.cache.js"></script>
  </body>
</html>

```

- Recharger la page. La modification n'est pas prise en compte.

Mettre une réponse dans le cache

- Créer un fichier *app.1.js*.
- Compléter le fichier *index.html* comme suit :

```

<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Service Worker</h1>

    <script src="app.js"></script>
    <script src="app.no.cache.js"></script>
    <script src="app.1.js"></script> <!-- inclusion du script _app.1.js_ -->
  </body>
</html>

```

- Modifier le service worker comme suit (modification du traitement lors de l'événement *fetch*) :

```

self.addEventListener('fetch', event => {
  console.log('Fetching:', event.request.url);

  event.respondWith(
    caches.match(event.request).then(response => {
      if (response) {
        console.log(event.request.url, 'servi depuis le cache');
        return response;
      }
      console.log(event.request.url, 'servi depuis le réseau');
      return fetch(event.request)

    })

    // rubrique à ajouter
    .then(function (response) {

      return caches.open(STATIC_CACHE_NAME).then(cache => {

        // mise en cache des ressources qui ne contiennent pas no.cache
        if (event.request.url.indexOf('no.cache') < 0) {
          cache.put(event.request.url, response.clone());
        }
        return response;
      });
    })

    .catch(error => {
      console.log("oops");
    })
  );
});

```

- Au premier rechargement :

```

...
http://localhost:8080/app.no.cache.js servi depuis le réseau
...
http://localhost:8080/app.1.js servi depuis le réseau
...

```

- Lors des rechargements suivants :

```
...  
http://localhost:8080/app.no.cache.js servi depuis le réseau  
...  
http://localhost:8080/app.1.js servi depuis le cache  
...
```

Faire évoluer le cache

- Modifier le service worker comme-suit (modification du traitement lors de l'événement *activate*):

```
self.addEventListener('activate', event => {  
  console.log('Activating new service worker...');  
  
  const cacheWhitelist = [STATIC_CACHE_NAME];  
  
  // suppression des caches excepté le cache courant (STATIC_CACHE_NAME)  
  event.waitUntil(  
    caches.keys().then(cacheNames => {  
      return Promise.all(  
        cacheNames.map(cacheName => {  
          if (cacheWhitelist.indexOf(cacheName) < 0) {  
            return caches.delete(cacheName);  
          }  
        })  
      );  
    })  
  );  
});
```

- Modifier le nom du cache :

```
const STATIC_CACHE_NAME = 'pages-cache-v2';
```

- Recharger la page et vérifier que le cache *pages-cache-v1* est bien supprimé.

Mode hors ligne

- Arrêter le serveur.
- Fermer le navigateur.
- Démarrer le navigateur et relancer l'application <http://localhost:8080/>.
- Vérifier que la page s'affiche toujours.

Données métiers

Stockage des données métiers au chargement de l'application

Pour faciliter le stockage des données métiers, nous allons utiliser la librairie *localforage* (<https://localforage.github.io/localForage>).

- Modifier le service worker comme-suit :

```
// import du script _localforage_

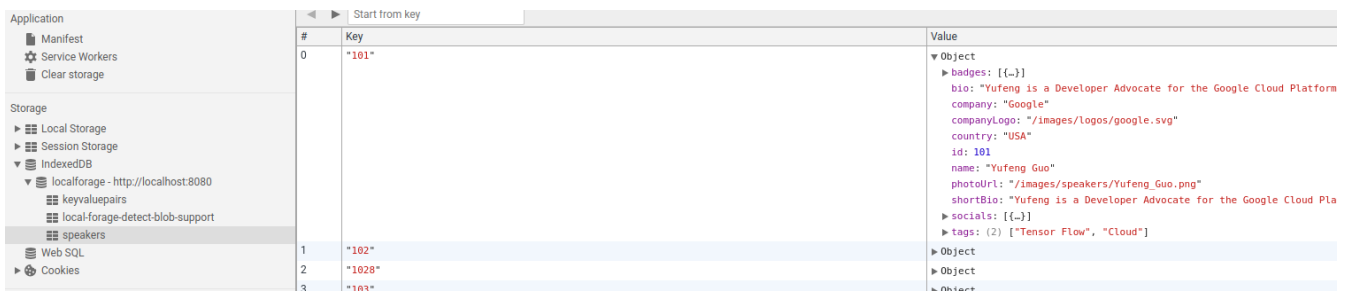
importScripts("https://cdn.rawgit.com/mozilla/localForage/master/dist/localforage.js")
;

// ....

// événement _install_

self.addEventListener('install', event => {
  console.log('Installation du Service Worker...');
  console.log('Mise en cache des ressources');
  event.waitUntil(
    Promise.all([
      caches.open(STATIC_CACHE_NAME)
        .then(cache => {
          return cache.addAll(FILE_TO_CACHE);
        }),
      fetch('https://raw.githubusercontent.com/DevInstitut/conference-
data/master/speakers.json')
        .then(resp => resp.json())
        .then(speakers => {
          localforage.config({storeName: 'speakers'})
          for (key in speakers) {
            localforage.setItem(key, speakers[key])
          }
        })
    ])
  );
});
```

- Recharger la page et vérifier le contenu de la base indexedDb.



#	Key	Value
0	*101*	Object badges: [{-}] bio: "Yufeng is a Developer Advocate for the Google Cloud Platform" company: "Google" companyLogo: "/images/Logos/google.svg" country: "USA" id: 101 name: "Yufeng Guo" photoUrl: "/images/speakers/Yufeng_Guo.png" shortBio: "Yufeng is a Developer Advocate for the Google Cloud Pla" socials: [{-}] tags: (2) ["Tensor Flow", "Cloud"]
1	*102*	Object
2	*1028*	Object
3	*103*	Object

Communication via des messages

Une page et un *Worker* peuvent communiquer par le biais de message.

Côté page (**app.js**) :

```
// recevoir des messages du service worker
navigator.serviceWorker.onmessage = function(event) {
  console.log("Reçu du SW : ", event.data);
}

// envoyer un message au service worker
if (navigator.serviceWorker.controller) {

  navigator.serviceWorker.controller.postMessage({
    "command": "MISE_A_JOUR",
    "message": "Hello je suis un client"
  });
}
```

Côté service worker (**sw.js**) :

```
// ecoute de message provenant d'un client

self.addEventListener('message', event => {
  // traitement du message (event.data)
})

// exemple d'envoi de message à tous les clients (en local)

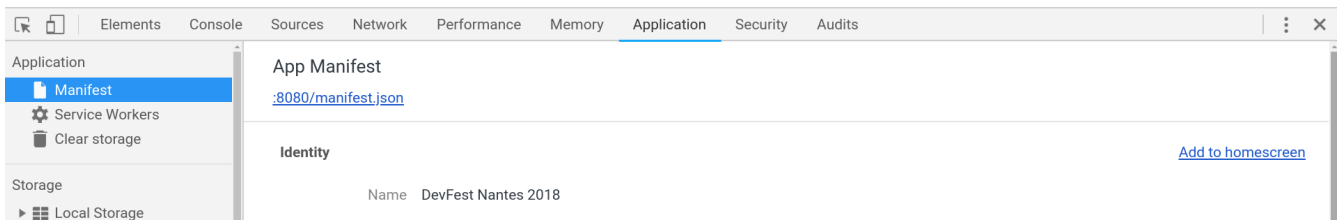
self.clients.matchAll().then(function(clients) {
  clients.forEach(function(client) {
    client.postMessage({
      "command": "HELLO_LES_CLIENTS",
      "message": "Hello je suis un SW"
    });
  })
})
```

Partie 2 - Web App Manifest

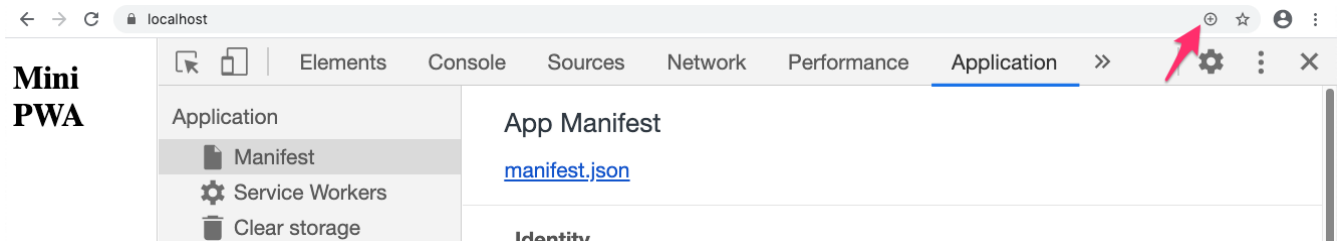
- Créer un fichier **manifest.json** à la racine du répertoire **mini-pwa** avec le contenu suivant :

```
{
  "name" : "Mini PWA"
}
```

- Lancer Chrome Dev Tools.



- Installer l'application sur votre machine.



- Vous devriez avoir une série d'erreurs dans la console. Nous allons valoriser uniquement les propriétés requises. Résolvez toutes les erreurs jusqu'à ce que l'installation s'effectue.

Frameworks UI

Pourquoi utiliser un framework ?

Les applications adaptées aux mobiles doivent obéir à des **codes**.

Un framework met à disposition des catégories de composants graphiques largement utilisées par des applications mobiles. Il vous évite de réinventer les usages courants.

Quelques frameworks UI ?

- JQuery Mobile
- Ionic Framework
- Kendo UI
- Sencha ExtJS
- Onsen UI

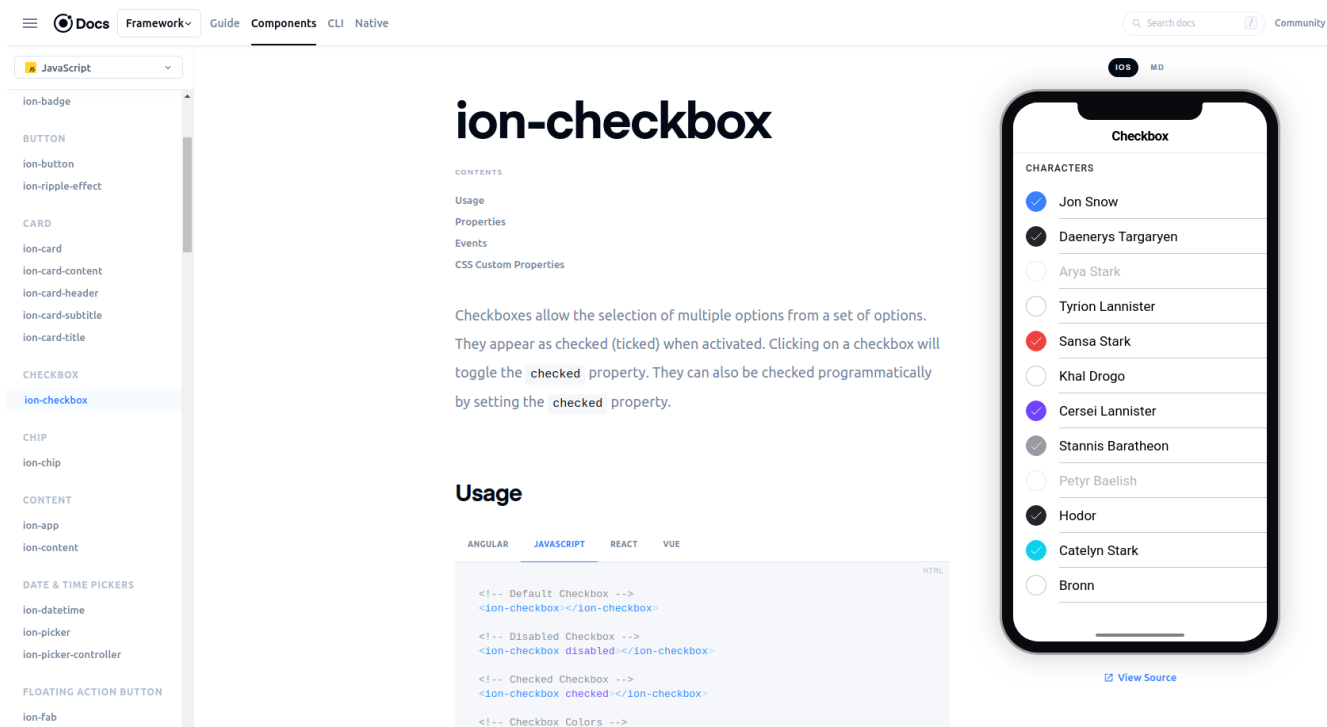
Ionic

Explorons un de ces frameworks : Ionic Core.

Il met à disposition plusieurs composants respectant les codes du mobile :



Visitez la documentation : <https://ionicframework.com/docs/api/>



Inclure Ionic Core

Pour inclure Ionic Core à une page :

```
<script type="module"
src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.esm.js"></script>
<script nomodule src=
"https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.js"></script>
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@ionic/core/css/ionic.bundle.css"/>
```

ion-app

Une application Ionic commence par l'utilisation de la base `<ion-app>` qui contiendra toute l'application.

Il ne doit y avoir qu'un élément `<ion-app>` par projet.

Un élément `<ion-app>` peut avoir plusieurs composants Ionic (menus, entêtes, contenu, pied de page).

Exemple d'application Ionic Core minimaliste :

```

<!doctype html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Démo Ionic Core</title>
  <script type="module"
src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.esm.js"></script>
<script nomodule src=
"https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.js"></script>
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@ionic/core/css/ionic.bundle.css"/>
</head>
<body>

<ion-app>
  <!-- ici les futurs vues de l'application -->
</ion-app>

</body>
</html>

```

ion-header, ion-footer, ion-content

Ionic donne la possibilité de structurer son application en 3 parties :

- entête via `<ion-header>`
- contenu principal via `<ion-content>`
- pieds de page via `<ion-footer>`

Exemple de structure :

```
<ion-app>
  <ion-header>
    <ion-toolbar>
      <ion-title>Entête</ion-title>
    </ion-toolbar>
  </ion-header>

  <ion-content>

</ion-content>

  <ion-footer>
    <ion-toolbar>
      <ion-title>Footer</ion-title>
    </ion-toolbar>
  </ion-footer>
</ion-app>
```

Rendu :

Entête

Footer

TP #02-ionic-core

- Créer un répertoire `02-ionic-core`.
- Ajouter un fichier `index.html`

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Blog DevFest Nantes</title>
  <script type="module"
src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.esm.js"></script>
  <script nomodule
src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.js"></script>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@ionic/core/css/ionic.bundle.css"/>
</head>
<body>

<ion-app>
<!-- TODO application à compléter -->
</ion-app>
<script src="app.js"></script>

</body>
</html>
```

- Créer un fichier `app.js` qui hébergera votre code JavaScript.

Les données du blog du Devfest sont accessibles via l'API GET <https://devfest-nantes-2018-api.cleverapps.io/blog>.

Les images viennent du site du DevFest 2018 : <https://devfest2018.gdgnantes.com/>.

- Créer le visuel suivant à partir des données distantes.



Annnonce du DevFest Nantes 2018

Le GDG Nantes est fier de vous annoncer la 7ème édition du DevFest Nantes! L'événement aura lieu les 18 et 19 Octobre 2018 à la Cité des Congrès de Nantes.



Voyage aux confins du code

