PWA

**SECTION 1 : Installation de notre environnement**

Tout le code source de cette formation qui permet de créer une PWA se trouve sur  :

<https://github.com/codeconcept/pwa_veilletechno>

Le code source de la REST API Node qui permettra de tester les background sync se trouve sur :

<https://github.com/codeconcept/nodetestapi>

Installer les 3 fichiers index.html, style.css et main.js

console.log('hello depuis main');

const technosDiv = document.querySelector('#technos');

let technos = [

    {id: 1, name: 'Angular', description: 'le framework front-end', url: 'https://angular.io/'},

    {id: 2, name: 'Node', description: 'JavaScript côté backe-end', url: 'https://nodejs.org/en/'},

    {id: 3, name: 'MongoDB', description: 'la célèbre base noSQL', url: 'https://www.mongodb.com/'},

    {id: 4, name: 'PWA', description: 'rendre vos applications ++', url: 'https://developer.mozilla.org/en-US/Apps/Progressive'}

];

function loadTechnologies(technos) {

    const allTechnos = technos

        .map(t => `<div><b>${t.name}</b> ${t.description} - site officiel </div>`)

        .join('');

    technosDiv.innerHTML = allTechnos;

}

loadTechnologies(technos);

Installer node.js

Installer live server : <https://www.npmjs.com/package/live-server>

cd /c/Projets/Git/pwa  
npm install -g live-serve   
live-server --port=3000 ou utiliser "go live" dans webstorm en bas à droite  
  
[Veille Techno](http://127.0.0.1:5501/index.html)  
  
npm init -y  
> permet d'ajouter un package.json  
  
dans package.json, ajouter la ligne   
"start": "live-server --port=3000",  
  
Permet de faire la ligne de commande  
npm start  
  
Installer <https://www.npmjs.com/package/json-server>

npm install -g json-server  
  
Créer un fichier db.json comme le second commit de <https://github.com/codeconcept/pwa_veilletechno>  
Taper la commande

json-server --watch db.json

dans package.json, ajouter la ligne

"jsonserver": "json-server --watch db.json"

Dans une seconde console executer   
**npm run jsonserver**  
  
L'API Json peut s'appeler à partir de cette url : <http://localhost:3000/technos>  
  
Cela nous permet d'utiliser notre json comme une API  
Modifier le main.js par

console.log('hello depuis main');

const technosDiv = document.querySelector('#technos');

function loadTechnologies(technos) {

    fetch('http://localhost:3000/technos')

        .then(response => {

            response.json()

                .then(technos => {

                    const allTechnos = technos.map(t => `<div><b>${t.name}</b> ${t.description}  <a href="${t.url}">site de ${t.name}</a> </div>`)

                            .join('');

                    technosDiv.innerHTML = allTechnos;

                });

        })

        .catch(console.error);

}

loadTechnologies(technos);

**SECTION 2 : Le service worker**

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

sw.js : service worker, il est capable de mettre en cache des données, de reception des push notifications.

Ajouter sw.js à la racine du projet

Dans main.js ajouter l'une des 2 conditions :

if ('serviceWorker' in navigator) {

}

if (navigator.serviceWorker) {

}

Enregistrer le service worker dans main.js :

if (navigator.serviceWorker) {

    navigator.serviceWorker.register('sw.js')

    .catch(err => console.error);

}

[Veille Techno](http://127.0.0.1:5501/index.html) : <http://127.0.0.1:5501/index.html> ou <http://localhost:5501/> (sur chrome à cause du https)

Crtl + Maj +I ou F12

Onglet application puis service worker

sw.js :

self.addEventListener('install', evt => {

    console.log('install evt', evt);

});

self.addEventListener('activate', evt => {

    console.log('activer evt', evt);

});

self.addEventListener('fetch', evt => {

    console.log('fetch event sur url', evt.request.url);

});

Notification de connexion internet :

sw.js

self.addEventListener('fetch', evt => {

    if (!navigator.online) {

        const headers = { headers: {'Content-Type': 'text/html;charset=utf-8'}};

        evt.respondWith(new Response('<h1>Pas de connexion internet</h1><p>Essayez de rétablir votre connexion</p>'));

    }

    console.log('fetch event sur url', evt.request.url);

});

CacheStorage :

création d'un cache dans chrome > F12 > Applications > CacheStorage

main.js

if (window.caches) {

    caches.open('veille-techno-1.8');

    caches.open('other-1.0');

    caches.keys().then(console.log);

}

**SECTION 3 : Gestion du cache**

Mise en cache de plusieurs fichiers dans main.js

if (window.caches) {

    caches.open('veille-techno-1.0').then(cache => {

        cache.addAll([

            'index.html',

            'main.js',

            'vendors/bootstrap4.min.css'

        ]);

    });

}

On peut mettre ce script dans le service worker sw.js en listant toutes les pages à mettre en cache

self.addEventListener('install', evt => {

    console.log('install evt', evt);

    caches.open(cacheName).then(cache => {

        return cache.addAll([

            'index.html',

            'main.js',

            'style.css',

            'vendors/bootstrap4.min.css',

            'add\_techno.html',

            'add\_techno.js',

            'contact.html',

            'contact.js',

        ])

    })

});

add\_technos.js

const technonameField = document.querySelector('#techno-name');

const technoDescriptionField = document.querySelector('#techno-description');

const technoUrlField = document.querySelector('#techno-url');

const addTechnoForm = document.querySelector('#add-techno-form');

addTechnoForm.addEventListener('submit', evt => {

    evt.preventDefault();

    const payload = {

        name: technonameField.value,

        description: technoDescriptionField.value,

        url: technoUrlField.value

    }

    fetch('http://localhost:3001/technos', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify(payload)

        })

        .then(resp => {

            console.log(resp);

        })

        .catch(err =>console.error);

})

Permet une fois le formulaire remplis d'ajouter les éléments des champs dans le json et d'afficher sur la home le résultat.

**SECTION 4 : Gestion avancée du cache**

Mise en cache des js, html, dans le service worker :   
Visible dans chrome > f12 > Network > Name

self.addEventListener('fetch', evt => {

    if (!navigator.online) {

        const headers = { headers: {'Content-Type': 'text/html;charset=utf-8'}};

        evt.respondWith(new Response('<h1>Pas de connexion internet</h1><p>Essayez de rétablir votre connexion</p>'));

    }

    console.log('fetch event sur url', evt.request.url);

    evt.respondWith(

        caches.match(evt.request).then(res => {

            if (res) {

                return res;

            }

        })

    )

});

S'il manque un fichier, ajouter un return :

evt.respondWith(

        caches.match(evt.request).then(res => {

            if (res) {

                return res;

            }

            return fetch(evt.request).then(newResponse => {

                caches.open(cacheName).then(cache => cache.put(evt.request, newResponse));

                return newResponse.clone();

        })

        })

    )

**SECTION 5 : Rendre votre application web instable**

<https://developer.mozilla.org/fr/docs/Web/Manifest>

L'utilisateur pourra installer un raccourci sur sa homePage du site, cela permet un accès plus rapide.

<link rel="manifest" href="manifest.json">

manifest.json :

{

    "name":"PWA",

    "short\_name":"pwa",

    "start\_url": "index.html",

    "portée" : "./",

    "display": "standalone",

    "theme\_color":"#fcfc35",

    "background\_color":"#0b2f77",

    "icons":[{

        "src":"assets/img/android-chrome-192x192.png",

        "sizes":"192x192",

        "type":"image/png",

        "purpose": "any maskable"

        },

        {

            "src":"assets/img/android-chrome-512x512.png",

            "sizes":"512x512",

            "type":"image/png",

            "purpose": "any maskable"

        }]

}

Il se visualise dans Chrome > F12 > Application > Manifest

**SECTION 6 : Notifications**

// Ajouter des notifications

if (window.Notification && window.Notification !== 'denied') {

    Notification.requestPermission(perm => {

        if (perm === 'granted') {

            const notif = new Notification('Hello notification')

        } else {

            console.log('Authorisation de notifications à été refusée')

        }

    })

}

Pour modifier les autorisations de notifications, il suffit d'aller dans le navigateur, à coté de l'url [Veille Techno](http://127.0.0.1:5501/), il y  a un **i**. Cliquez dessus et les notifications avec un choix apparait.

Ajout d'un body et d'1 icon à la notification :

// Ajouter des notifications

if (window.Notification && window.Notification !== 'denied') {

    Notification.requestPermission(perm => {

        if (perm === 'granted') {

            const options = {

                body: 'je suis le body de la notification',

                icon: 'images/icons/icon-72x72.png'

            }

            const notif = new Notification('Hello notification', options);

        } else {

            console.log('Authorisation de notifications à été refusée');

        }

    })

}

Ajouter une notification permanente dans le service worker :

self.registration.showNotification('Notif depuis le sw', {

    body: 'je suis une notification dite "persistante"'

});

Permet d'adresser des notifications depuis un serveur par exemple.

Notification fermée :

self.addEventListener('notificationclose', evt => {

    console.log('notification fermée', evt);

});

Des actions de l'utilisateur sur des notifications :

self.registration.showNotification('Notif depuis le sw', {

    body: 'je suis une notification dite "persistante"',

    actions: [

        {action: 'accept', title: 'accepter'},

        {action: 'refuse', title: 'refuser'}

    ]

});

self.addEventListener('notificationclick', evt => {

    if(evt.action === 'accept') {

        console.log('vous avez accepté')

    } else if (evt.action === 'refuse') {

        console.log('vous avez refusé')

    } else {

        console.log ('vous avez cliqué sur la notification (pas sur 1 des boutons)');

    }

})

Fermer une notification :

self.addEventListener('notificationclick', evt => {

    if(evt.action === 'accept') {

        console.log('vous avez accepté')

    } else if (evt.action === 'refuse') {

        console.log('vous avez refusé')

    } else {

        console.log ('vous avez cliqué sur la notification (pas sur 1 des boutons)');

    }

    evt.notification.close();

})

**SECTION 7 : les notifications push**

Push notif :

self.addEventListener('push', evt => {

    console.log('push event', evt);

    console.log('data envoyée par la push notif des devs tools',evt.data.text());

    const title = evt.data.text();

    evt.waitUntil(self.registration.showNotification(title, {

        body : 'ça marche',

        image : 'images/icons/icon-152x152.png'

    }));

})

Architecture :

Firebase Cloud Messaging ou MangoDB : base de donnée  
Serveur Node

Etape 0

vapid key : génération d'une clé public / clé privée

recupération d'une fonction capable de convertir une string (qui est en base 64) en array buffer (Uint8Array)

Etape 1

client (chrome) joint une clé publique pour effectuer une demande de souscription

Firebase Cloud Messaging (FCM)

FCM envoie une souscription

Etape 2

configurer un push serve Node afin qu'il puisse envoyer une push notification à un utilisateur possédant une souscription crée durant l'étape 1

Génération de vapid keys :

<https://github.com/codeconcept/pwa_veilletechno/commit/fe0dcc42bf6bbc96d0e6d30e97afe594aa76772d>

<https://github.com/web-push-libs/web-push>

npm install web-push --save

npm install web-push -g

web-push generate-vapid-keys --json

web-push generate-vapid-keys --json > pushServerKeys.json

Ce fichier est crée et contient les clés

Aller sur <https://github.com/web-push-libs/web-push> et récupérez Vapid Key for applicationServerKey

dans main.js :

//VAPID Key

// retrieved from https://github.com/web-push-libs/web-push readme

// and used to convert base64 string to Uint8Array == to an array buffer

function urlBase64ToUint8Array(base64String) {

    const padding = '='.repeat((4 - base64String.length % 4) % 4);

    const base64 = (base64String + padding)

      .replace(/\-/g, '+')

      .replace(/\_/g, '/');

    const rawData = window.atob(base64);

    const outputArray = new Uint8Array(rawData.length);

    for (let i = 0; i < rawData.length; ++i) {

      outputArray[i] = rawData.charCodeAt(i);

    }

    return outputArray;

}

Récupération ou création d'une souscription auprès d'un push service

dans main.js

if (navigator.serviceWorker) {

    navigator.serviceWorker

    .register('sw.js')

    //.catch(err => console.error);

    .then(registration => {

        // public vapid key generated with web-push

        const publicKey = 'BCVZPt9-NfZxqSQeJX4tTYaSjxQ--NYG4Q72manYl9Usm\_X7KzlJsiPXF-5LJIh8vzlNWNGNmszvr6qPVlbdTWA';

        registration.pushManager.getSubscription().then(subscription => {

            if (subscription) {

                console.log('subscription', subscription);

                // no more keys proprety directly visible on the subscription objet. So you have to use getKey()

                const keyArrayBuffer = subscription.getKey('p256dh');

                const authArrayBuffer = subscription.getKey('auth');

                const p256dh = btoa(String.fromCharCode.apply(null, new Uint8Array(keyArrayBuffer)));

                const auth = btoa(String.fromCharCode.apply(null, new Uint8Array(authArrayBuffer)));

                console.log('p256dh key', keyArrayBuffer, p256dh);

                console.log('auth key', authArrayBuffer, auth);

                return subscription;

            } else {

                // ask for a subscription

                const convertedKey = urlBase64ToUint8Array(publicKey);

                return registration.pushManager.subscribe({

                    userVisibleOnly:true,

                    applicationServerKey: convertedKey

                })

                .then(newSubscription => {

                    console.log('new subscription', newSubscription)

                })

            }

        })

    })

}

Envoyer une notification push depuis Node

Créer le fichier pushClientSubscription.json :

{

    "endpoint": "",

    "keys": {

        "auth": "",

        "p256dh": ""

    }

}

Dans main.js, on ajoute une fonction afin de simplifier le code :

// retrieved from https://github.com/web-push-libs/web-push readme

// and used to convert base64 string to Uint8Array == to an array buffer

function urlBase64ToUint8Array(base64String) {

    const padding = '='.repeat((4 - base64String.length % 4) % 4);

    const base64 = (base64String + padding)

      .replace(/\-/g, '+')

      .replace(/\_/g, '/');

    const rawData = window.atob(base64);

    const outputArray = new Uint8Array(rawData.length);

    for (let i = 0; i < rawData.length; ++i) {

      outputArray[i] = rawData.charCodeAt(i);

    }

    return outputArray;

}

if(navigator.serviceWorker) {

    navigator.serviceWorker

    .register('sw.js')

    .then(registration => {

        // public vapid key generate with web-push command line

        const publicKey = 'BCVZPt9-NfZxqSQeJX4tTYaSjxQ--NYG4Q72manYl9Usm\_X7KzlJsiPXF-5LJIh8vzlNWNGNmszvr6qPVlbdTWA';

        registration.pushManager.getSubscription().then(subscription => {

            if(subscription) {

                console.log('subscription',subscription);

                // no more keys proprety directly visible on the subscription objet. So you have to use getKey()

                const keyArrayBuffer = subscription.getKey('p256dh');

                const authArrayBuffer = subscription.getKey('auth');

                const p256dh = btoa(String.fromCharCode.apply(null, new Uint8Array(keyArrayBuffer)));

                const auth = btoa(String.fromCharCode.apply(null, new Uint8Array(authArrayBuffer)));

                console.log('p256dh key', keyArrayBuffer, p256dh);

                console.log('auth key', authArrayBuffer, auth);

                return subscription;

            } else {

                // ask for a subscription

                const convertedKey = urlBase64ToUint8Array(publicKey);

                return registration.pushManager.subscribe({

                    userVisibleOnly: true,

                    applicationServerKey: convertedKey

                })

                .then(newSubscription => {

                    // TODO post to a subscription DB

                    console.log('newSubscription', newSubscription);

                    // no more keys proprety directly visible on the subscription objet. So you have to use getKey()

                    const key = newSubscription.getKey('p256dh');

                    const auth = newSubscription.getKey('auth');

                    console.log('p256dh key', key);

                    console.log('auth key', auth);

                })

            }

        })

    })

    .catch(err => console.error('service worker NON enregistré', err));

}

Dans pushClientSubscription.json, nous remplissons les valeurs que nous voyons dans la console de chrome :

{

    "endpoint": "https://fcm.googleapis.com/fcm/send/f5sHiiaN17w:APA91bFoNKuR5FIgnfmSPDk2l5JyxXl\_5kZr-KFz\_Jm9Bs-AbcQ9Swntp01u-AE7QchlBLAm0lex8SEBBEpr8hX2krDbX0JF\_\_RmrAp75DwCwAOSQmVyfnAXn-6Pt4deK-jpPu7GRhX1",

    "keys": {

        "auth": "Eb00EiNi50/KSjsXQRQt9w==",

        "p256dh": "BLlT9St6txTzi+ii2U9XgNXdzR3a65lpHJP8gTtiqu/QhfRRZatnYKRSgy+lmw8xL1Tbii+r/luZbs/SZwchiPc="

    }

}

Dans pushServer.js, on ajoute :

const webPush = require('web-push');

const pushServerKeys = require('./pushServerKeys');

const pushClientSubscription = require('./pushClientSubscription');

webPush.setVapidDetails('mailto:bordes\_cecile@hotmail.com', pushServerKeys.publicKey, pushServerKeys.privateKey);

const subscription = {

    endpoint: pushClientSubscription.endpoint,

    keys: {

        auth: pushClientSubscription.keys.auth,

        p256dh: pushClientSubscription.keys.p256dh

    }

};

webPush.sendNotification(subscription, 'Notification envoyée depuis le serveur push node :)')

.then(res => console.log('ma push notif a bien été poussée', res))

.catch(err => console.error);

console.log(pushServerKeys,pushClientSubscription);

Puis dans Git Bash, on ajoute :   
node pushServer.js

**SECTION 8 : Background Sync**

Synchronisation en arrière plan  
Sync Manager doit être installé  
Pour les formulaires hors ligne, sauvegarde dans le localStorage ou indexDB

mlab.com pour avoir une base de donnée MangoDB

Chrome > Application > IndexDB > techno > Key et Value > unsynced: true ou false

Infrastructure :

Application > Service Workers > Offline : ne marche pas toujours. Pour tester, couper le wifi.

zeit.co/now : pour créer un petit serveur et le déployer. <https://vercel.com/home>

<https://github.com/codeconcept/nodetestapi>

Aller copier/coller le fichier <https://raw.githubusercontent.com/codeconcept/nodetestapi/master/server.js>

Et le déployer sur zeit.co/now

Pour avoir des pages en https sur GitHub, il suffit d'aller dans son repo, dans settings, puis dans "GitHub Pages" et de mettre source sur "master branch".

<https://mlab.com/> pour créer une base de données.

dans add\_techno.js :

const payload = {

**id:Date.now()**,

        name: technonameField.value,

        description: technoDescriptionField.value,

        url: technoUrlField.value,

**unsynced: true**,

    }

    //fetch('http://localhost:3001/technos', {

    fetch('**https://nodetestapi-thyrrtzgdz.now.sh/technos**', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify(payload)

        })

        .then(resp => {

            console.log(resp);

        })

        .catch(() => {

            if ('serviceWorker' in navigator && 'SyncManager' in window) {

                console.log('SyncManager supported by browser');

                console.log('we are probably offline');

                navigator.serviceWorker.ready.then(registration => {

                    //put techno in IndexedDB for later syncing

                    console.log('dans le catch de add\_techno car offline');

                    console.log('dans le catch de add\_techno id vaut', payload.id);

                    console.log();

                    return putTechno(payload.id).then(() => {

                        // register a sync with the ServiceWorker

                        return registration.sync.register('sync-technos')

                    });

                })

            }

            else {

                // TODO browser does NOT support SyncManager: send data to server via ajax

                console.log('SyncManager NOT supported by your browser');

            }

        })

        .then(() => {

            clearform();

        })

        .catch(err =>console.error);

<https://glitch.com/edit/#!/dream-catcher?path=public%2Fdatabase.js%3A1%3A0>

> permet d'intéragir avec IndexDB

idb.js : ajouter ces 2 js à notre projet et à notre serviceworker

dans sw.js pour le cache :

self.addEventListener('install', evt => {

    console.log('install evt', evt);

    caches.open(cacheName).then(cache => {

        return cache.addAll([

            'index.html',

**'idb.js',**

**'database.js'**

            'main.js',

            'style.css',

            'vendors/bootstrap4.min.css',

            'add\_techno.html',

            'add\_techno.js',

            'contact.html',

            'contact.js',

        ])

    })

});

dans add\_techno.html :

<script src="idb.js"></script>  
          <script src="database.js"></script>

Notre environnement est prêt

Background Sync coté Service Worker :

dans sw.js :

self.addEventListener('sync', event => {

    if (event.tag === 'sync-technos') {

        console.log('attempting sync', event.tag);

        console.log('syncing', event.tag);

        event.waitUntil(

            getAllTechnos().then(technos => {

                console.log('got technos from sync callback', technos);

                const unsynced = technos.filter(techno => techno.unsynced);

                console.log('pending sync', unsynced);

                return Promise.all(unsynced.map(techno => {

                    console.log('Attempting fetch', techno);

                    fetch('https://nodetestapi-thyrrtzgdz.now.sh/technos', {

                        method: 'POST',

                        headers: {

                            'Accept': 'application/json',

                            'Content-Type': 'application/json'

                        },

                        body: JSON.stringify(payload)

                    })

                    .then(() => {

                        console.log('Sent to server');

                        console.log('id passé à putTechno', techno.id);

                        return putTechno(Object.assign({}, techno, {unsynced: false}), techno.id);

                    })

                }))

            })

        )

    }

});

On peut utiliser index DB API : <https://developer.mozilla.org/fr/docs/Web/API/IndexedDB_API>

<http://localhost:5501/add_techno.html>

Serveur distant

https://nodetestapi-thyrrtzgdz.now.sh/technos

qu'on a déployé grâce à zeit.co/now

Qui lui même s'execute grâce à mlab.com/databases/vtechno/collections/technos (base de donnée)

Ou bien utiliser le js server <https://raw.githubusercontent.com/codeconcept/nodetestapi/master/server.js> qui permet de faire du post