Réalisation d'une mise sen œuvre
Y. Stroppa
PSB 2020 N°1

Cas N°1 : première application PWA

Cette application se veut très simple et a pour objectif de vous montrer l'utilisation des mécanismes PWA. On peut utiliserpython3.0 juste pour démarrer un serveur web ou un autre service de type Apache. pour ce dernier cas il faut mettre les fichier sous la bonne arborescence de votre serveur web.

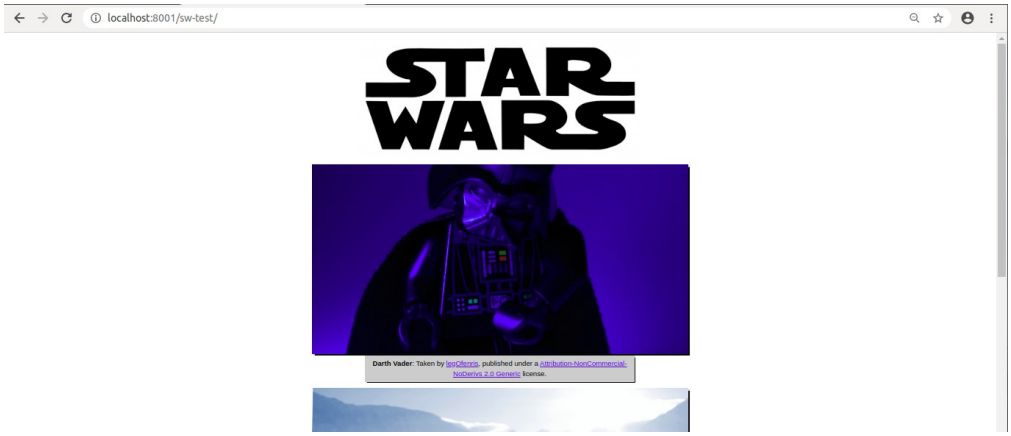
Site de réf : https://github.com/mdn/sw-test
Cloner le site sur un répertoire de votre machine. IL faut au préalable que sur votre machine soit installé git. Normalement accessible à tous les environnements de Windows à Linux en passant par mac OSX.
Cette commande n'a que pour objectif de téléverser les fichiers du site distant sur une arborescence de votre machine.
git clone https://github.com/mdn/sw-test

```
.
├── app.js
├── CODE_OF_CONDUCT.md
├── gallery
│   ├── bountyHunters.jpg
│   ├── myLittleVader.jpg
│   └── snowTroopers.jpg
├── image-list.js
├── index.html
├── LICENSE
├── README.md
├── star-wars-logo.jpg
├── style.css
└── sw.js
```

Liste des fichiers rapatriés

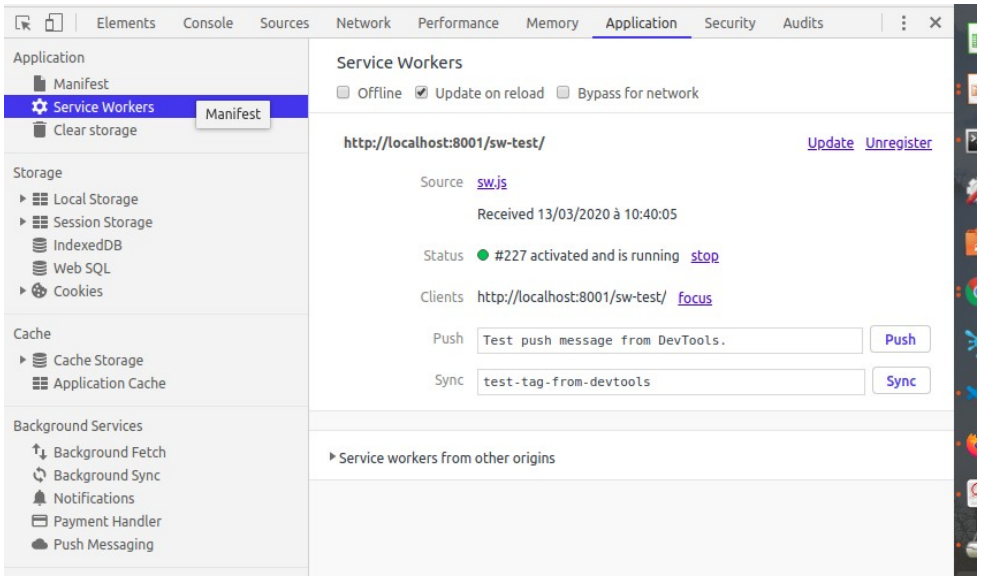
Lancez la commande python pour démarrer un serveur web
#python3 -m http.server 8001

Et se connecter à partir d'un navigateur sur l'url: http://localhost:8001/sw-test/

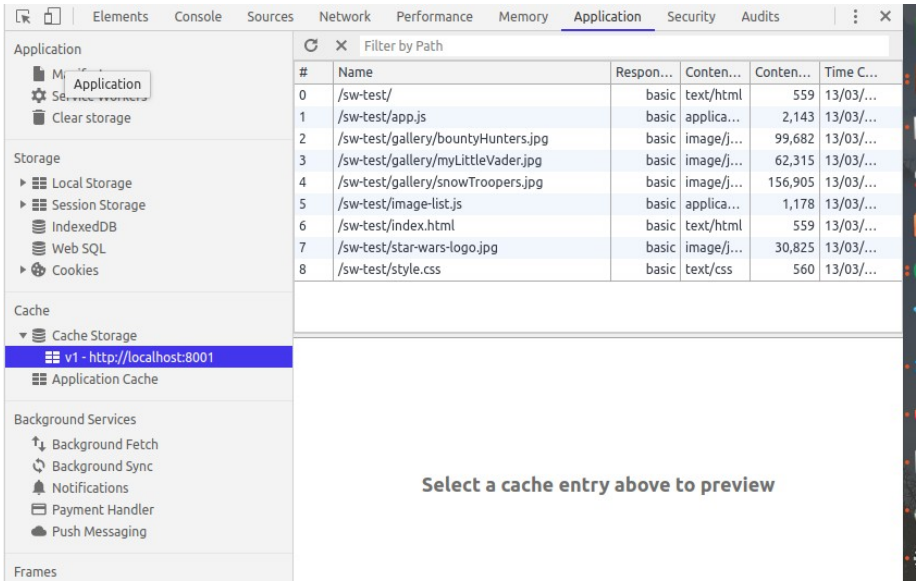


Observation et analyse de l'application :

Se mettre sous l'environnement dev de votre navigateur et regarder le service worker



Et si on regarde du coté du cache-storage



Vous devez avoir ce type de configuration une fois que votre application web est lancée.

Analyse du contenu des fichiers

Détail du fichier sw.js

```
self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open('v1').then(function(cache) {
      return cache.addAll([
        '/sw-test/',
        '/sw-test/index.html',
        '/sw-test/style.css',
        '/sw-test/app.js',
        '/sw-test/image-list.js',
        '/sw-test/star-wars-logo.jpg',
        '/sw-test/gallery/bountyHunters.jpg',
        '/sw-test/gallery/myLittleVader.jpg',
        '/sw-test/gallery/snowTroopers.jpg'
      ]);
    })
  );
});
self.addEventListener('fetch', function(event) {
  event.respondWith(caches.match(event.request).then(function(response) {
    if (response !== undefined) {
      return response;
    } else {
      return fetch(event.request).then(function (response) {
        let responseClone = response.clone();
        caches.open('v1').then(function (cache) {
          cache.put(event.request, responseClone);
        });
        return response;
      }).catch(function () {
        return caches.match('/sw-test/gallery/myLittleVader.jpg');
      });
    }
  }));
});
```

A faire :

Indiquez à quoi sert le fichier sw.js et à quel moment est-il utilisé ? Où est-il référencé dans l'application.
Veuillez tester cette application en mode offline ...
Indiquez les liens entre les fichiers ... qui appel qui ...
Le nom du cache utilisé
Indiquez les fichiers que l'on veut préserver sur le poste client (navigateur) ?
Décrivez le fonction et la gestion de l'événement fetch dans ce fichier...

Maintenant dans cet exemple nous n'avons pas gérer le nettoyage des caches, sin on est amené à changer le cache comment peut-on faire pour nettoyer les anciens caches .... Donnez une solution et l'adapter à cette application ...

Cas N° 2 : application de gestion de produits.

L'objectif de cette application qui fonctionne cette fois-ci sous nodejs, vous permettre d'illustrer le fonction d'une application PWA et une possibilité de travailler en mode déconnecté afin de continuer son activité à l'aide de produit comme Hoodie associé à nodeJS.

Site de réf :
https://www.twilio.com/blog/2018/03/practical-introduction-pwa-node-hoodie-offline-first.html

Dans cet exercice on utilisera nodejs, hoodie, babel.

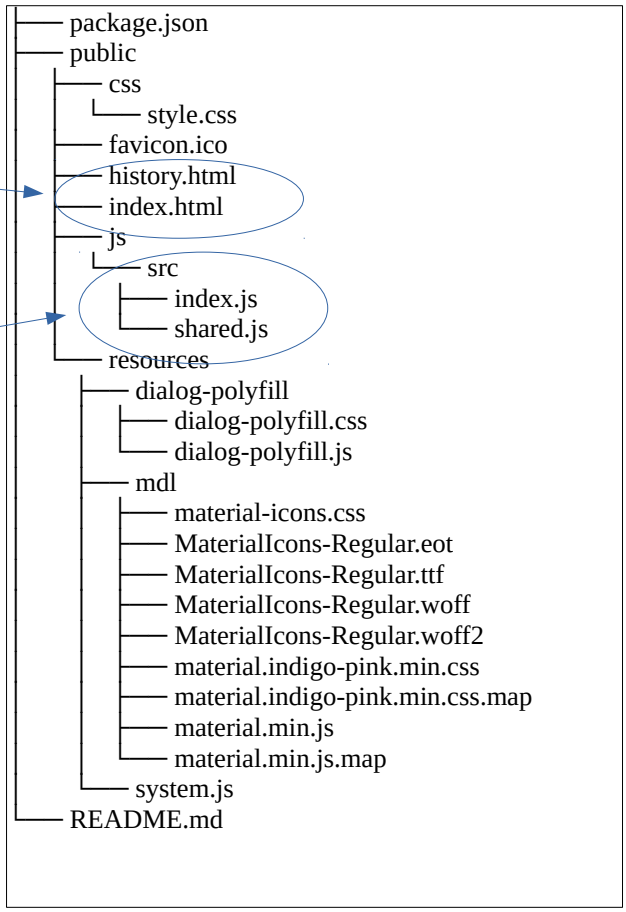
Veuillez récupérer le code source à partir du site à l'aide de l'utilitaire git[1].

```
#git clone https://github.com/pmbanugo/shopping-list-starter.git
```

Se mettre sous le répertoire shopping-list-starter

Le fichier client de démarrage de l'appli.

Les fichiers js qui complète la partie client



1 Git : voir annexes

Description du contenu du fichier package.json

Partie déclarative

Scripts de lancement

Dépendances

```
{
  "name": "shopping-list",
  "type": "module",
  "version": "1.0.0",
  "description": "a progressive web app that you can use to list items to buy and also store the list and total item cost",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
  },
  "author": "Peter Mbanugo",
  "license": "ISC",
  "dependencies": {
    "hoodie": "^28.2.10"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-plugin-transform-es2015-modules-systemjs": "^6.23.0",
    "babel-preset-es2015": "^6.22.0"
  }
}
```

#npm install

Commande npm install :
Effectue la lecture du fichier packages.json (duparagraphe dépendance) et télécharge si pas présent tous les fichiers nécessaires au fonctionnement de nodejs.
Package.json décrit lea configuration de votre application et ses dépendances.

Regardez le contenu du répertoire node_modules pour vérifier la présence de l'ensemble des dépendances de votre application.

Première modification à effectuer Pour sauvegarder les données en local, il nous faut installer la module hoodie à l'aide de la commande
#npm install —save hoodie

Hoodie s'installe dans les modules et construit par convention un répertoire ./hoodie dans le lequel on trouve les fichiers :

```
.hoodie
├── client.js
└── data
    ├── hack.json
    ├── hoodie-config.json
    ├── hoodie-store.json
    ├── pouch__all_dbs__.json
    ├── user
    │   └── 9lah04k.json
    ├── user9lah04k.json
    └── _users.json
```

Ajouter dans le fichier index.html les références vers les scripts js à la fin du body

```
<script src="/hoodie/client.js"></script>
<script src="/js/src/index.js"></script>
</body>
```

Sauvegarde des données

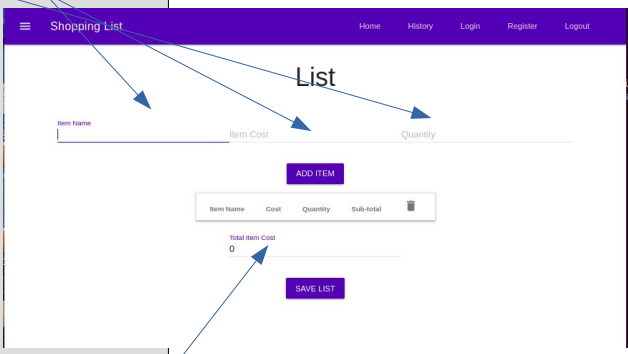
Ouvrir le fichier index.js pour compléter les éléments de stockage que l'on va effectuer à partir de hoodie SAVE DATA

On se préoccupe uniquement pour le moment du stockage des informations pour les nouveaux items. On définit du côté de Hoodie comment stocker ces éléments.

```
function saveNewitem() {
 let name = document.getElementById("new-item-name").value;
 let cost = document.getElementById("new-item-cost").value;
 let quantity = document.getElementById("new-item-quantity").value;
 let subTotal = cost * quantity;

 if (name && cost && quantity) {
   hoodie.store.withIdPrefix("item").add({
     name: name,
     cost: cost,
     quantity: quantity,
     subTotal: subTotal
   });
   document.getElementById("new-item-name").value = "";
   document.getElementById("new-item-cost").value = "";
   document.getElementById("new-item-quantity").value = "";
 } else {
   let snackbarContainer = document.querySelector("#toast");
   snackbarContainer.MaterialSnackbar.showSnackbar({
     message: "All fields are required"
   });
 }
}

function init() {
 hoodie.store.withIdPrefix("item").on("add", addItemToPage);
 document.getElementById("add-item").addEventListener("click", saveNewitem);
 //retrieve items on the current list and display on the page
 hoodie.store
   .withIdPrefix("item")
   .findAll()
   .then(function(items) {
     for (let item of items) {
       addItemToPage(item);
     }
   });
}
init();
```



Partie init qui permet de remplir la page avec la ,liste des éléments présents dans la hoodie.store

Maintenant DELETE DATA

```
function deleteRow(deletedItem) {
        let row = document.getElementById(deletedItem._id);
        let totalCost = Number.parseFloat(
                document.getElementById("total-cost").value
        );
        document.getElementById("total-cost").value =
                totalCost - deletedItem.subTotal;
          row.parentNode.removeChild(row);
}

function deleteItem(itemId) {
        hoodie.store.withIdPrefix("item").remove(itemId);
}
```

Compléter la fonction init() pour rajouter la gestion de l'événement

```
function init() {
 hoodie.store.withIdPrefix("item").on("add", addItemToPage);
 hoodie.store.withIdPrefix("item").on("remove", deleteRow);
 document.getElementById("add-item").addEventListener("click", saveNewitem);

 //retrieve items on the current list and display on the page
 hoodie.store
   .withIdPrefix("item")
   .findAll()
   .then(function(items) {
     for (let item of items) {
       addItemToPage(item);
     }
   });

 window.pageEvents = {
   deleteItem: deleteItem
 };
}
```

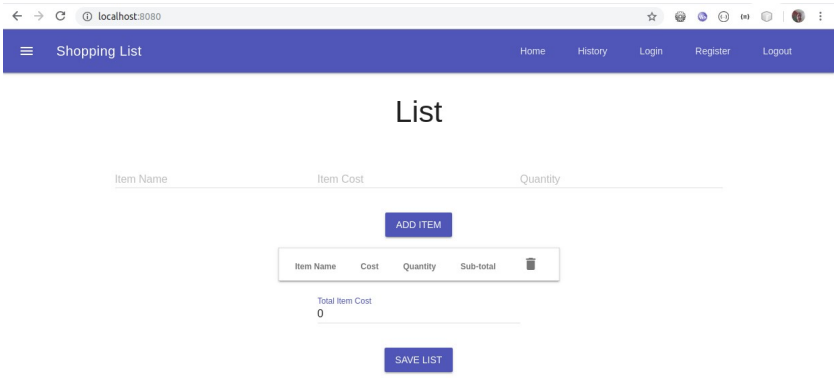
Après ces modifications , on revient sur la racine de l'application et il faut compléter le fichier package.json pour indiquer le démarrage de l'appli.

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

Une fois cette modification effectuée, on peut exécuter l'application à l'aide de la commande #npm start

Démarre l'application sur le port 8080

On démarrant un navigateur on doit pouvoir visualiser la page suivante :



Configuration du login et register de Hoodie pour permettre la saisie et le stockage des infos. C'est dans le fichier shared.js que nous allons rajouter ces éléments :

```
let login = function() {
  let username = document.querySelector("#login-username").value;
  let password = document.querySelector("#login-password").value;
  hoodie.account
    .signIn({
      username: username,
      password: password
    })
    .then(function() {
      showLoggedIn();
      closeLoginDialog();

      let snackbarContainer = document.querySelector("#toast");
      snackbarContainer.MaterialSnackbar.showSnackbar({
        message: "You logged in"
      });
    })
    .catch(function(error) {
      console.log(error);
      document.querySelector("#login-error").innerHTML = error.message;
    });
};
let register = function() {
  let username = document.querySelector("#register-username").value;
  let password = document.querySelector("#register-password").value;
  let options = { username: username, password: password };
  hoodie.account
    .signUp(options)
    .then(function(account) {
      return hoodie.account.signIn(options);
    })
    .then(account => {
      showLoggedIn();
      closeRegisterDialog();
      return account;
    })
    .catch(function(error) {
      console.log(error);
      document.querySelector("#register-error").innerHTML = error.message;
    });
};
```

Idem pour les fonctions de déconnexion :

```
let signOut = function() {
  hoodie.account
    .signOut()
    .then(function() {
      showAnonymous();
      let snackbarContainer = document.querySelector("#toast");
      snackbarContainer.MaterialSnackbar.showSnackbar({
        message: "You logged out"
      });
      location.href = location.origin;//trigger a page refresh
    })
    .catch(function() {
      let snackbarContainer = document.querySelector("#toast");
      snackbarContainer.MaterialSnackbar.showSnackbar({
        message: "Could not logout"
      });
    });
};
let updateDOMWithLoginStatus = () => {
  hoodie.account.get("session").then(function(session) {
    if (!session) {
      // user is signed out
      showAnonymous();
    } else if (session.invalid) {
      // user has signed in, but session has expired
      showAnonymous();
    } else {
      // user is signed in
      showLoggedIn();
    }
  });
};
```

Il faut compléter dans la shared.js la partie

```
export {
  register,
  login,
  signOut,
  updateDOMWithLoginStatus
};
```

Compléter le fichier index.js en ajoutant sur la première ligne l'instruction suivante :
import * as shared from "shared.js";

Réadapter le fichier index.js comme suit :

```
function init() {
 shared.updateDOMWithLoginStatus();
 hoodie.store.withIdPrefix("item").on("add", addItemToPage);
 hoodie.store.withIdPrefix("item").on("remove", deleteRow);



 window.pageEvents = {
   ...
   closeLogin: shared.closeLoginDialog,
   showLogin: shared.showLoginDialog,
   closeRegister: shared.closeRegisterDialog,
   showRegister: shared.showRegisterDialog,
   login: shared.login,
   register: shared.register,
   signout: shared.signOut
 };
}
```

On va utiliser le transcompilateur babel pour convertir le code en ECMA2015 avec les options avancées

Plusieurs manipulations sont nécessaires :

Dans le fichier .babel

```
//file -> .babelrc
{
    "plugins": ["transform-es2015-modules-systemjs"],
    "presets": ["es2015"]
}
```

Dans le ficher package.json

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "hoodie",
    "build": "babel public/js/src --out-dir public/js/transpiled"
  }
```

Dans le fichier index.html

```
<body>
  <script src="/hoodie/client.js"></script>
  <script src="resources/system.js"></script>
  <script>
    System.config({ "baseURL": "js/transpiled" });
    System.import("index.js");
  </script>
</body>
```

Arrêter le service si il est lancé,
On peut maintenant lancer la commande de trans-compilateur à l'aide de npm
#npm run build

Ensuite le relancer
#npm start

Test de l'application :

On vérifie que notre application fonctionne , on s'enregistrant et en utilisant les fonctionnalités d'ajout de de retrait d'items.

Création du service worker

Création du service worker pour ce faire, il est nécessaire de savoir ce qui est important à conserver du côté du client pour continuer à fonctionner en mode offline.

Ajoutez en entête du fichier shared.js les instructions de démarrage du worker service dans le navigateur

```
if ("serviceWorker" in navigator) {
  navigator.serviceWorker
    .register("sw.js")
    .then(console.log)
    .catch(console.error);
}
```

Ce code une fois exécuté, vérifie la présence d'un fichier sw.js dans la répertoire public et l'exécute si présent.

Contenu du fichier sw.js

```
const CACHE_NAME = "cache-v1";
const assetToCache = [
  "/index.html",
  "/",
  "/history.html",
  "/resources/mdl/material.indigo-pink.min.css",
  "/resources/mdl/material.min.js",
  "/resources/mdl/MaterialIcons-Regular.woff2",
  "/resources/mdl/material-icons.css",
  "/css/style.css",
  "/resources/dialog-polyfill/dialog-polyfill.js",
  "/resources/dialog-polyfill/dialog-polyfill.css",
  "/resources/system.js",
  "/js/transpiled/index.js",
  "/js/transpiled/history.js",
  "/js/transpiled/shared.js",
  "/hoodie/client.js"
];
self.addEventListener("install", function(event) {
  console.log("installing");
  event.waitUntil(
    caches
      .open(CACHE_NAME)
      .then((cache) => {
        return cache.addAll(assetToCache);
      })
      .catch(console.error)
  );
});
```

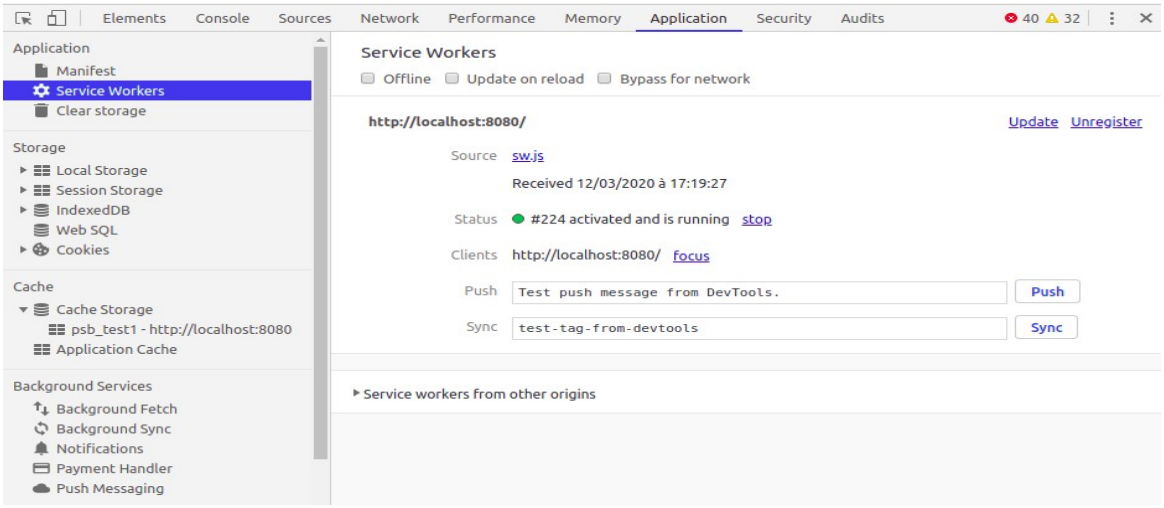
On pourra ajouter la partie suivante dans le fichier sw.js pour permettre une fois le réseau établi, de déverser les données sur le serveur.

```
self.addEventListener("fetch", function(event) {
  event.respondWith(
    caches.match(event.request).then(function(response) {
      if (response) {
        return response; //return the matching entry found
      }
      return fetch(event.request);
    })
  );
});
```

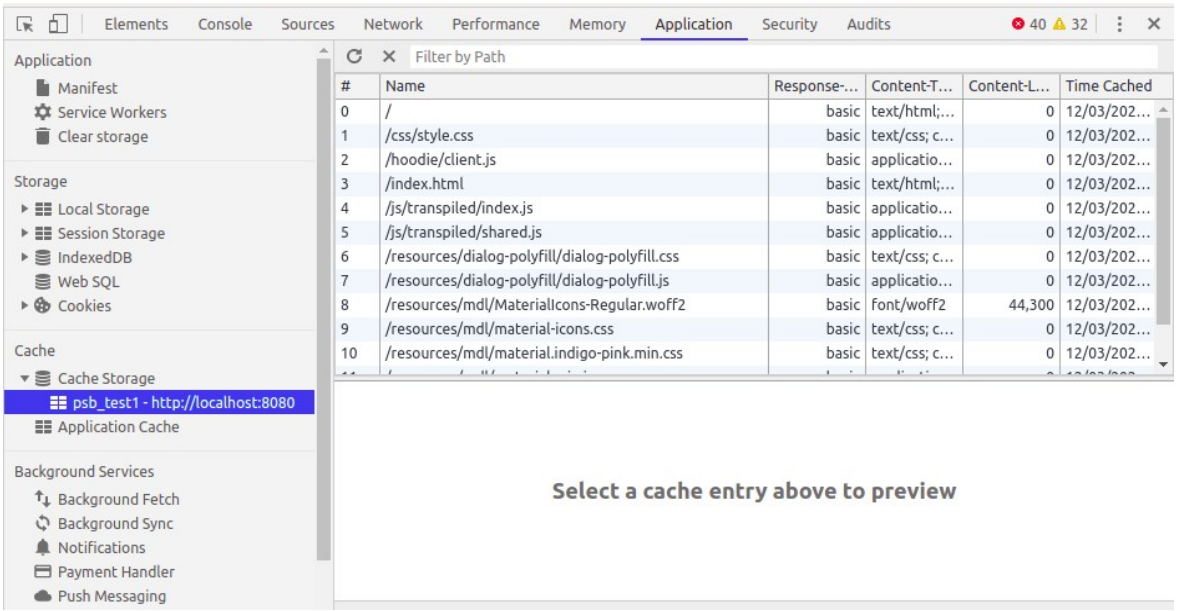
Tests et explications du fonctionnement

Une fois votre application démarrée :

Lancez votre navigateur et la partie développeur pour se mettre sous application :



Si l'initialisation du service worker se passe bien



Ensuite pour effectuer un test Offline, veuillez mettre votre navigateur en mode offline et continuer à saisir des produits ... à partir d'un autre navigateur lui online essayez de voir si vous visualiser les données saisies ... et remettre votre navigateur en mode online ...

On change la nature des données :
Pour aller plus loin, veuillez sur cette racine d'application, l'aménager de telle sorte que l'on puisse saisir des données de type publications :

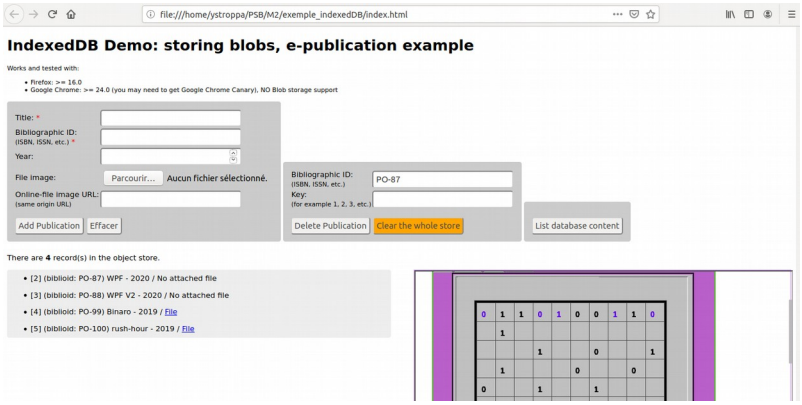
Publication : titre, résumé, date, auteurs ....

Cas N°3 : Utilisation indexedDB

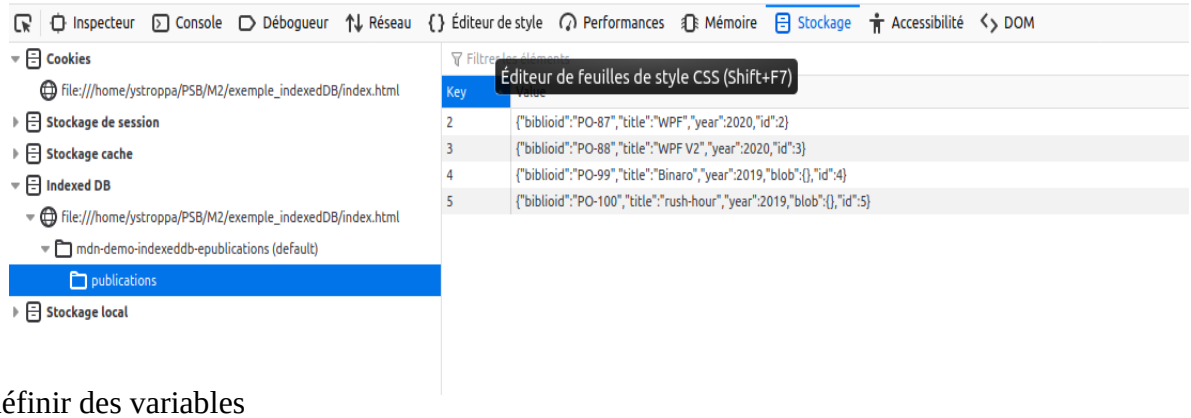
Objectif est d'utiliser la base de données IndexexDB de votre navigateur comme cache de haut niveau pour une application web.

Mise en œuvre de l'exemple fourni par
https://developer.mozilla.org/fr/docs/Web/API/API_IndexedDB/Using_IndexedDB

Une fois démarrée l'application nous affiche la fenêtre suivante :



Détail de la structure de stockage sous votre navigateur



A définir des variables

```
DB_NAME="mdn-demo-psb-M2-2020" ;
DB_VERSION=1 ;
DB_STORE_NAME="publications" ;
```

Fonctions à élaborer

```
function openDb()
function getObjectStore(store_name,mode)
function clearObjectStore(store_name)
function getBlob(key,store,success_callback)
```

function displayPublish(store)
function newViewerFrame()
function setInViewer(key)
function addPublicationFromUrl(biblioid, title,year,url)
function addPublication(biblioid, title,year,blob)
fonction deletePublicationFromBib(biblioid)
function deletePublication(key, store)
function displayActionSuccess(msg)
function displayActionFailure(msg)
function restActionStatus()
function addEventListeners()

fonction invoquées au chargement :

openDb() ;
addEventListeners() ;

L'application est prête pour fonctionner en local.

Annexes

npm : explication et fonctionnement

Node Package module

C'est le gestionnaire de module de nodejs, qui permet de démarrer et d'installer l'ensemble des dépendances d'une application nodejs.

Git : explication et fonctionnement

Un gestionnaire de version, qui permet lors de vos développements d'entretenir et de vous offrir des fonctionnalités de gestion des versions des sources de votre projet. Indispensable dans le développement informatique et vous soulage de cette partie.

Pour démarrez un nouveau projet dans le répertoire de vos sources iol suffit d'exécuter la commande
#git init

Ensuite une fois cette commande exécutée, vous devez indiquez à git quels sont les fichiers à préserver et à gèrer localement ou à distance ...
git add .....

Ensuite on peut commiter l'ensemble à l'aide de la commande
#git commit

#git status pour voir quel est le statut de votre projet et sur quelle branche vous ètes.

Système de

Hoodie : description et fonctionnement

Babel : description et fonctionnement