TP's

PWA Module

ENI, Nicolas Hodicq

Version 1.0, 2019-05-05

Table des matières

Chapitre 1. Installation

Tous les TP seront réalisés avec le navigateur Chrome

1.1. Chrome

- Installez Chrome pour votre plateforme.

1.2. Node.js

Node.js en version LTS doit être installé sur la machine locale. Pour cela, il existe 2 façons de l'installer :

1.2.1. Site officiel

- téléchargez la version correspondant à votre système sur le Site officiel

1.2.2. Node Version Manager

Installer un version manager permet d'avoir une ou n version de Node.js différentes sur la machine.

Mac

- Installation via HomeBrew

```
brew install nvm
```

- Ajoutez ensuite à votre profile (~/.bash_profile, ~/.zshrc, ~/.profile, or ~/.bashrc), le contexte pour nvm

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh" # This loads nvm
```

- Vous pouvez ensuite voir toutes les versions de node.js disponibles. Installez la version v8.1.0

```
nvm ls-remote

nvm install v8.1.0

nvm ls
```

Windows

- Installation de Nodist
- Page de release

1.2.3. Test de l'installation

- Ouvrez un terminal et lancez les commandes suivantes pour vérifier les versions :

```
node -v

npm -v
```

1.3. LightHouse

> Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, and more.
>
> — lighthouse, developers.google.com

- Installez Lightouse

```
npm install -g lighthouse
```

- Testez l'installation

```
lighthouse --help
```

Chapitre 2. Performance

Dans le zip, qui vous a été fourni en début de module, se trouve un dossier **punk-pwa**. Ce projet est un starter d'application web réalisé avec Webpack.

Webpack est utilisé pour compiler des modules JavaScript. Il est extensible via un système de plugins.

- Copiez le projet dans votre espace de travail
- Installez les dépendances Node.js

```
npm install
```

- Vérifiez que l'application se lance correctement

```
# Lance un serveur de développement
npm start

# Il existe d'autres lanceurs pour builder l'application en mode developement ou production
```

2.1. Tree Shaking

L'objectif du Tree Shaking est d'éliminer le code mort ou inutilisé lors de la phase de build de l'application

- Contruisez l'application en mode de développement

```
npm run build:dev
```

- Dans le dossier `./dist`, vous trouverez un fichier `app.bundle.js`. Ouvrez le et effectuez une recherche du nom de la function exporté par le fichier `not-bundle.js`, **treeShaking**
- Vous ne devriez pas trouvez de référence
- Modifiez le fichier `index.js` comme ci-dessous :

```
import _ from 'lodash';
import '../styles/index.css';
import {treeShaking} from './not-bundle';    ①

function component() {

  treeShaking();      ②

  const element = document.createElement('div');
  element.classList.add('main');
  const button = document.createElement('button');
  const br = document.createElement('br');

  button.innerHTML = 'Click me and look at the console!';
  element.innerHTML = _.join(['Hello', 'webpack'], ' ');
  element.appendChild(br);
  element.appendChild(button);

  button.onclick = e => import(/* webpackChunkName: "print" */ './print').then(module
=> {

    const formation = module.default;
    formation('PWA');
  });
  return element;
}

document.body.appendChild(component());
```

① Import de la fonction **treeShaking**

② Appel de la fonction

- Contruisez l'application en mode de développement et effectuez la recherche à nouveau.
- Vous devriez trouver une référence à la fonction que vous venez d'appeler dans l'application

Le Tree Shaking permet donc de ne pas inclure les fichiers qui ne sont pas importés dans l'application. C'est pratique pour nos fichiers de tests à coté de nos scripts, par exemple, mais surtout pour toutes les fonctionnalités que nous n'utilisons pas dans les librairies importées dans nos applications.

2.2. Lazy Loading

Le Lazy Loading doit permettre d'aller chercher une ressource, par exemple un fichier JavaScript, à la demande. L'objectif est de ne charger que l'essentiel pour l'affichage de la home page afin de permettre une expérience utilisateur la plus fluide possible.

- Ouvrez le fichier `index.js`

```
import _ from 'lodash';
import '../styles/index.css';

function component() {

  const element = document.createElement('div');
  const button = document.createElement('button');
  const br = document.createElement('br');

  button.innerHTML = 'Click me and look at the console!';
  element.innerHTML = _.join(['Hello', 'webpack'], ' ');
  element.appendChild(br);
  element.appendChild(button);

  button.onclick = e => import(/* webpackChunkName: "print" */ './print').then(module
=> { ①

    const formation = module.default;
    formation('PWA');
  });
  return element;
}

document.body.appendChild(component());
```

① La syntaxte `import(/* webpackChunkName: "print" */ './print')` indique à Webpack que le fichier `./print` doit être créé comme un chunk lors de la phase de build. Cela créera un fichier JavaScript séparé du app.bundle.js. Lors du click sur le bouton, le fichier sera téléchargé depuis le serveur.

- Démarrez l'application
- Dans le navigateur, ouvrez les dev tools, et affichez l'onglet Network
- Cliquez sur le bouton
- Vous devriez voir une requête passée sur le fichier `print.bundle.js`, avant que le log soit affiché dans la console.

Avec tous les frameworks récents (React, Angular, Vue, ..), ce genre de configuration est possible via les routeurs. N'hésitez pas à vous référer à chaque documentation pour en savoir plus.

2.2.1. Punk API

- Nous allons utiliser cette API, pour manipuler des données dynamiques
- Dans la documentation, identifiez l'appel à réaliser pour afficher une page de 40 Bières

2.2.2. Punk PWA

- Dans l'application, supprimez le fichier print.js
- Créez un fichier `punk.list.js` qui retournera de façon asynchrone et lazy loadé un composant HtmlElement qui affiche la liste d'image des bières
- Créez un fichier `punk.api.js` qui aura la responsabilité de fetch les données sur la Punk API
- Au clic sur le bouton, récupérez la liste de bières et affichez la liste d'image des bières

2.3. Lighthouse

- Dans votre terminal, lancez une analyse Lighthouse sur l'application punk-pwa

```
lighthouse http://localhost:8080/ --view
```

- Le rapport va s'ouvrir dans une page de votre navigateur
- Analysez les retours et corrigez toutes les erreurs liées aux metadatas de la page HTML

Plutôt que d'utiliser le client Node.js, vous pouvez faire cet audit dans les Chrome dev-tools, onglet Audit

- Analysez quelque sites de références, Netflix, Facebook, Google, Airbnb, ... et pourquoi pas aussi la fnac, ...

Chapitre 3. Fiabilité

3.1. Installer un service worker

Pour installez un service worker, il y a 2 étapes :

- Créer le service worker
- Enregistrer le service worker si l'API est disponible

3.1.1. Créer le service worker

Un fichier `sw.js` est déjà présent dans dossier src de l'application

- Editez le fichier comme suit :

```
self.addEventListener('install', function(event) {
  console.log('[ServiceWorker] installed!', event);
});

self.addEventListener('activate', function(event){
  console.log('[ServiceWorker] activated!', event);
});
```

3.1.2. Enregistrer le service worker

- Dans le fichier `src/js/index.js`, ajoutez le code suivant à la fin du fichier

```
    if ('serviceWorker' in navigator) {
      navigator.serviceWorker
        .register('./sw.js')
        .then(function() { console.log('Service Worker Registered'); });
    }
```

- Démarrez l'application, et ouvrez les Chrome dev-tools, dans l'onglet application / Services workers, vous devriez voir votre worker démarré.
- Dans cet onglet je vous conseille d'activer la coche `Update on reload` qui permet d'installer le service worker à chaque, fois que l'on reload la page. C'est très pratique en cours de développement.
- La coche offline vous permet de tester votre application sans connectivité.

3.2. Cacher les ressources de l'application shell

Pour rendre l'application robuste à la perte de connectivité, la première étape est de mettre en cache tous les fichiers de l'application shell.

- Editez le fichier `src/sw.js` comme suit :

```
var cacheNameStatic = 'punk-pwa-static-v1';   ①

var filesStaticToCache = [                    ②
  '/',
  '/index.html',
  '/app.bundle.js',
  '/punk.list.bundle.js',
  '/assets/android-chrome-192x192.png',
  '/assets/android-chrome-512x512.png',
  '/assets/apple-touch-icon.png',
  '/assets/favicon-16x16.png',
  '/assets/favicon-32x32.png',
  '/assets/ic_refresh_white_24px.svg',
  '/assets/mstile-150x150.png',
  '/assets/safari-pinned-tab.svg',
];

self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(
    caches.open(cacheNameStatic).then(function(cache) {       ③
      console.log('[ServiceWorker] Caching app shell');
      return cache.addAll(filesStaticToCache);                ④
    })
  );
});
```

① Nom du cache pour l'application avec sa version

② Listes des fichiers à cacher

③ Récupération ou création du cache

④ Ajout de tous les fichiers dans le cache

C'est une bonne pratique de mettre le numéro de la version dans le nom du cache. Lors d'une montée de version, il se peut que vous souhaitiez supprimer ce cache lié à la version.

- Démarrez l'application et ouvrez les Chrome dev-tools
- Dans l'onglet Application, controllez dans le cache storage que tous les fichiers ont été ajouté

- Mettez l'application en mode offline
- Cliquez sur le bouton, que se passe-t-il ?

3.3. Récupérer les ressources dans le cache

Les fichiers sont bien dans le cache, mais lorsque vous passez offline, les requêtes tombent en erreur. Il est nécessaire de scripter le comportement que l'on souhaite sur le fetch des ressources.

- Editez le fichier `src/sw.js` comme suit :

```
var cacheNameStatic = 'punk-pwa-static-v1';

var filesStaticToCache = [
  '/',
  '/index.html',
  '/app.bundle.js',
  '/punk.list.bundle.js',
  '/assets/android-chrome-192x192.png',
  '/assets/android-chrome-512x512.png',
  '/assets/apple-touch-icon.png',
  '/assets/favicon-16x16.png',
  '/assets/favicon-32x32.png',
  '/assets/ic_refresh_white_24px.svg',
  '/assets/mstile-150x150.png',
  '/assets/safari-pinned-tab.svg',
];

self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(
    caches.open(cacheNameStatic).then(function(cache) {
      console.log('[ServiceWorker] Caching app shell');
      return cache.addAll(filesStaticToCache);
    })
  );
});

self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request)                ①
      .then(function(response) {
          if (response) {
            return response;                   ②
          }
          return fetch(event.request);         ③
        }
      )
  );
});
```

① La ressource est-elle conservée dans le cache ?

② La ressource est dans le cache, on la récupère

③ La ressoure n'est pas dans le cache, on la récupère via un appel réseau

Le pattern implémenté est le `cache then network`

Jake Archibald a écrit un article qui fait office de référence sur les différents patterns. Il a été intégré à la documentation de Google. Vous trouverez tous les patterns ainsi que des exemples d'implémentation et les uses cases pour ces patterns.

- Démarrez l'application et ouvrez les Chrome dev-tools
- Mettez l'application en mode offline
- Cliquez sur le bouton, que se passe-t-il ?

3.4. Cacher les ressources dynamiques

Les données dynamiques, récupération de la liste de bières mais aussi les images que nous devons affichées ne sont pas accessibles offline.

Quand on recoit l'évènnement `fetch` dans le service worker, nous souhaitons :

- Conserver le fonctionnement initial, à savoir récupération d'une ressource depuis le cache si elle y est déjà , sinon on forge un appel réseau pour la récupérer
- Si la ressource est une ressources de l'API Punk (liste de bière ou images), un cache dédié est créé, les ressources récupérées au fur et à mesure de la navigation seront conservées dans ce cache
    - Editez le fichier `src/sw.js` comme suit :

```
var cacheNameStatic = 'punk-pwa-static-v1';
var cacheNameDynamic = 'punk-pwa-dynamic-v1';

var filesStaticToCache = [
  '/',
  '/index.html',
  '/app.bundle.js',
  '/punk.list.bundle.js',
  '/assets/android-chrome-192x192.png',
  '/assets/android-chrome-512x512.png',
  '/assets/apple-touch-icon.png',
  '/assets/favicon-16x16.png',
  '/assets/favicon-32x32.png',
  '/assets/ic_refresh_white_24px.svg',
  '/assets/mstile-150x150.png',
  '/assets/safari-pinned-tab.svg',
];

self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(
    caches.open(cacheNameStatic).then(function(cache) {
      console.log('[ServiceWorker] Caching app shell');
      return cache.addAll(filesStaticToCache);
    })
  );
});

self.addEventListener('fetch', function(event) {
  if (event.request.url.indexOf('punkapi.com') === -1) {      ①
    event.respondWith(
      caches.match(event.request)
        .then(function (response) {
```

```
            if (response) {
              return response;
            }
            return fetch(event.request);
          }
        )
    );
  } else {
    console.log('PUNK API REQUEST', event.request.url)
    event.respondWith(
      caches.open(cacheNameDynamic).then(function(cache) {    ②
        return cache.match(event.request).then(function (response) {      ③
          return response || fetch(event.request).then(function(response) {
            cache.put(event.request, response.clone());       ④
            return response;
          });
        });
      })
    );
  }
});
```

① La ressource provient-elle de l'API Punk

② Création ou récupération du cache dynamique

③ Récupération depuis le cache dynamique

④ Ajout de la ressource dans le cache

- Démarrez l'application et ouvrez les Chrome dev-tools
- Cliquez sur le bouton pour mettre les données dans le cache
- Rechargez la page
- Mettez l'application en mode offline
- Cliquez sur le bouton, l'application devrait fonctionner sans erreur

3.5. TP

En prenant la base de l'application, créez une nouvelle application Github PWA

- Ajoutez un champ de saisie pour récupérer le login d'un utilisateur
- Ajoutez un bouton qui déclenchera un appel vers l'API Github. Récupérez les informations de l'utilisateur, via son login

Documentation de l'API Github

- Affichez dans la page, l'avatar, le nom du compte, la liste de ses repositories
- Le squelette de l'application doit être caché pour être disponible offline
- Les données dynamiques récupérées au fur et à mesure des actions seront persistées en cache
- Si il n'y pas de connectivité
  - un message est affiché sur l'écran 'offline'
  - Seules les données des utilisateurs en cache (déjà récupérée lors d'une précédente utilisation) peuvent être affichées.
  - Si les données ne sont pas présentes dans le cache, un message est affiché sur la page, l'invitant à recommencer lorsqu'il y aura du réseau.

Chapitre 4. Engagement utilisateur

4.1. Manifeste

Pour installer la PWA sur la home screen du device, l'application doit déclarer un manifeste. En fonction des platformes, le comportement varie :

- Sur Android, la bannière d'installation apparaîtra automatiquement. Nous n'avons pas la possibilité de déclencher ce traitement.
- Sur iOS, il vous faudra l'installer vous même en cliquant sur l'icone partager, puis add to home screen

4.1.1. Création du manifest.json

Un fichier `manifest.json` est déjà présent dans dossier src de l'application

- Editez le fichier

```
{
  "short_name": "Punk",
  "name": "Punk PWA",
  "icons": [
    {
      "src": "/assets/android-chrome-192x192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/assets/android-chrome-512x512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/",
  "background_color": "#3367D6",
  "display": "standalone",
  "scope": "/",
  "theme_color": "#3367D6"
}
```

- Dans le fichier `index.html`, ajoutez la commande suivante dans l'en tête du fichier

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


- Démarrez l'application
- Ouvrez les Chrome dev-tools
- Dans l'onglet Application, affichez les informations du Manifest. Vous devriez voir toutes les informations du Manifest.
- Cliquez sur `add to home screen` pour simuler l'installation de l'application
- Validez l'installation, l'application sera installée en tant qu'application Chrome. Vous pouvez la trouvez en allant sur la page `chrome://apps/` dans Chrome
- Supprimez l'application

4.1.2. Différer l'installation

Il est, de temps en temps nécessaire de différer l'installation de l'application, pour le proposer à un moment ou l'utilisateur s'engagera plus facilement. En effet, si il refuse l'installation, celle-ci ne lui sera plus proposée lors de la prochaine visite sur l'application. En fonction des navigateurs, la durée avant que le prompt soit ré-affichée diffère, avec Chrome c'est 3 mois.

Avant d'afficher le prompt, les navigateur dispatchent l'évènnement `beforeinstallprompt`, il nous est possible de modifier le traitement par défaut.

- Editez le fichier `index.js`

```
import _ from 'lodash';
import '../styles/index.css';

function component() {

  const element = document.createElement('div');
  element.classList.add('main');
  const button = document.createElement('button');
  const br = document.createElement('br');

  button.innerHTML = 'Click me and look at the console!';
  element.innerHTML = _.join(['Hello', 'webpack'], ' ');
  element.appendChild(br);
  element.appendChild(button);

  button.onclick = e => import(/* webpackChunkName: "punk.list" */
'./punk/punk.list').then(module => {

    module.beersListComponent()
      .then(beersComponent => {
        console.log('BEERS', beersComponent);
        document.body.appendChild(beersComponent);
      });
  });
```

```
  return element;
}

document.body.appendChild(component());

if ('serviceWorker' in navigator) {
  navigator.serviceWorker
    .register('./sw.js')
    .then(function() { console.log('Service Worker Registered'); });
}

let deferredPrompt;           ①

const addToHome = document.createElement('button');  ②
addToHome.innerText = 'Install PWA';
addToHome.style.display = 'none';                    ③
document.body.appendChild(addToHome);

window.addEventListener('beforeinstallprompt', (e) => {
  e.preventDefault();     ④

  deferredPrompt = e;

  addToHome.style.display = 'block';        ⑤

  addToHome.addEventListener('click', () => {

    addToHome.style.display = 'none';

    deferredPrompt.prompt();      ⑥

    deferredPrompt.userChoice.then((choiceResult) => {    ⑦
      if (choiceResult.outcome === 'accepted') {
        console.log('Utilisateur valide l\'installation');
      } else {
        console.log('Utilisateur refuse l\'installation');
      }
      deferredPrompt = null;
    });
  });
});

window.addEventListener('appinstalled', () => {     ⑧
  console.log('Manifest installed');
});

if (window.matchMedia('(display-mode: standalone)').matches) { ⑨
  console.log('display-mode is standalone');
```

```
}

// For Safari
if (window.navigator.standalone === true) {       ⑩
  console.log('display-mode is standalone');
}
```

① Variable pour conserver une référence vers l'event

② Ajout d'un bouton dans le DOM

③ Le bouton ne sera pas visible

④ Modification du traitement par défaut, bloque l'event

⑤ Affichage du bouton personnalisé

⑥ Affichage du prompt

⑦ Promise pour récupérer le choix de l'utilisateur

⑧ Event pour réaliser un traitement quand l'application sera installée

⑨ Affiche-t-on l'application en mode standalone pour la majorité des navigateurs

⑩ Affiche-t-on l'application en mode standalone pour Safari

- Démarrez l'application
- Ouvrez les Chrome dev-tools
- Dans l'onglet Application, affichez les informations du Manifest
- Cliquez sur `add to home screen` pour simuler l'installation de l'application
- Le bouton doit s'afficher dans la page, cliquez dessus
- Validez l'installation, l'application sera installée en tant qu'application Chrome. Vous pouvez la trouvez en allant sur la page `chrome://apps/` dans Chrome
- Supprimez l'application

4.1.3. Github PWA

- Créer un Manifest pour l'application
- Installez l'application sur le Home screen
- Lancez une analyse avec Lighthouse

4.2. Push Notification

4.2.1. Server Side

Création d'un projet pour envoyer les notifications

- Créez un nouveau dossier punk-pwa-push
- Positionnez vous dans le dossier en ligne de commande et initialisez le projet Node.js

```
npm init -y
```

- Ajoutez la librairie web-push

```
npm install web-push --save
```

- Créez un fichier index.js

```
const webpush = require('web-push');

// TODO : Vous allez générér les clés à l'étape suivante du TP
const vapidKeys = {
  publicKey:'PUBLIC_API_KEY',
  privateKey:'PRIVATE_API_KEY'
};

webpush.setVapidDetails(
  'mailto:nhodicq@bewizyu.com',
  vapidKeys.publicKey,
  vapidKeys.privateKey
);

// TODO : Vous ajouterez la subscription ultérieurement dans le TP
const subscription = {};

const dataToSend = {
  type: 'Engaging',
  data : {
    title : 'PWA Rocks !!',
    message : 'Engaging user with notification',

  }
}

webpush.sendNotification(subscription, JSON.stringify(dataToSend))
  .catch((err) => {
    if (err.statusCode === 410) {
      console.log('Delete subscription from database');
    } else {
      console.log('Subscription is no longer valid: ', err);
    }
  });
```

- Pour jouez le script. Attention pour l'instant ce n'est pas encore prêt !!!

```
node index.js
```

Création des Vapid Keys

La création de Vapid Key est nécessaire pour réaliser des envois de push notifications.

- Créez les clés

```
npx web-push generate-vapid-keys --json
```

- Ajoutez les clés générées dans le fichier index.js

4.2.2. Client Side

Permission utilisateur

- Ajoutez dans le fichier `src/index.js` le code suivant

```
function askPermission() {
  return new Promise(function(resolve, reject) {
    const permissionResult = Notification.requestPermission(function(result) {
①
      resolve(result);
    });

    if (permissionResult) {     ②
      permissionResult.then(resolve, reject);
    }
  })
    .then(function(permissionResult) {
      if (permissionResult !== 'granted') {
        throw new Error('We weren\'t granted permission.');
      }

      return;

    });
}
```

① Historiquement la signature de la fonction requestPermission prenait un callback. Pour la rétro compatibilité, c'est une bonne pratique de toujours la déclarer

② La nouvelle API de requestPermission renvoit aujourd'hui une Promise

Pour gérer la rétrocompatibilité, il est nécessaire de gérer l'appel de cette fonction avec le callback et la promise

- Au click sur le bouton, après l'affichage de la liste de bières, demandez la permission de l'utilisateur

Subscription

- Ajoutez dans le fichier `src/index.js` le code suivant


```
    let swRegistration;         ①

    function urlBase64ToUint8Array(base64String) {              ②
      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;
    }

     function subscribeUserToPush() {       ③
      const subscribeOptions = {
        userVisibleOnly: true,
        applicationServerKey: urlBase64ToUint8Array(
          'YOUR_PUBLIC_API_KEY'
        )
      };
      return swRegistration.pushManager.subscribe(subscribeOptions)
        .then(function (pushSubscription) {
          console.log('Received PushSubscription: ', JSON.stringify(
pushSubscription)); ④
          return pushSubscription;
        });
    }
```

① Variable pour conserver une référence sur la `registration` du service worker

② Fonction utilitaire pour convertir la clé publique

③ Fonction pour enregistrer l'utilisateur au Push

④ Récupération de l'objet pushSubscription, cet objet est requis coté serveur pour envoyer des notifications

N'oubliez pas d'ajouter votre PUBLIC_API_KEY. L'object pushSubscription devra être copié dans votre console, pour le mettre dans le fichier index.js du serveur de push

- Dans le fichier index.js, modifier l'inscription du Service worker

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker
    .register('./sw.js')
    .then((registration) => {
      swRegistration = registration;       ①
      console.log('Service Worker Registered', swRegistration);
    });
}
```

① Conservation de l'objet `registration`

- Chainez la subscription à la demande de permission

```
askPermission()
  .then(subscribeUserToPush);
```

Event push dans le service worker

- Ajoutez le code suivant dans le service worker

```
self.addEventListener('push', function(event) {
  if (event.data) {
    const pushData = event.data.json();
    console.log('Type: ', pushData.type);

    self.registration.showNotification(pushData.data.title, {
      body : pushData.data.message,
    });

  } else {
    console.log('This push event has no data.');
  }
});
```

- Démarrez l'application pwa
- Récupérer la subscription dans la console et ajoutez la dans le fichier index.js coté serveur
- Envoyez une notification (node index.js) pour vérifier que vous les recevez bien.
- Envoyez plusieurs notifications

En vous basant sur les configurations possibles pour afficher un push :

```
{
  "//": "Visual Options",
  "body": "<String>",
  "icon": "<URL String>",
  "image": "<URL String>",
  "badge": "<URL String>",
  "vibrate": "<Array of Integers>",
  "sound": "<URL String>",
  "dir": "<String of 'auto' | 'ltr' | 'rtl'>",

  "//": "Behavioural Options",
  "tag": "<String>",
  "data": "<Anything>",
  "requireInteraction": "<boolean>",
  "renotify": "<Boolean>",
  "silent": "<Boolean>",

  "//": "Both Visual & Behavioural Options",
  "actions": "<Array of Strings>",

  "//": "Information Option. No visual affect.",
  "timestamp": "<Long>"
}
```

- Affichez une icone dans la notification
- Forcez l'utilisateur à interagir avec la notification pour la fermer
- Ajoutez un badge
- Ajoutez une image
- Définissez 2 actions
- En prenant exemple sur ce site de démo, mettez en place le merge notification

4.2.3. Event clicknotification dans le service worker

- Ajoutez dans le service worker le code suivant

```
self.addEventListener('notificationclick', function(event) {
  if (!event.action) {
    console.log('Notification Clicked');
    return;
  }
});
```

- Effectuez un log au click sur les actions

4.2.4. Github PWA

- Mettez en place les notifications
- Affichez une icône sur la notification
- Mettez en place le `focus a window or open`, redirigez vers le site Github au click sur la notification

Chapitre 5. Firebase hosting

Et si on déployait l'application pour enfin la tester sur device mobile. Il existe tout un tas de plateformes permettant de déployer un site rapidement sur un environnement couvert en HTTPS. J'ai choisi Firebase car c'est une plateforme très utilisée pour les applications mobiles et pour sa simplicité.

- Générez l'application de production

```
npm run build:prod
```

- Créez vous un compte sur Firebase
- Créez un projet sur Firebase, par exemple `punk-pwa`
- Suivez la documentation pour déployer votre application.

L'application de production est construite dans le répertoire `dist`

- Testez votre application, offline, push, installation sur le Home screen de votre device mobile.

Chapitre 6. Annexes

6.1. Documentation

- Documentation Google
- Mozilla documentation
- Push Notification
- Démo Notification
- Lighthouse
- Firebase

6.2. Librairies

- Node.js web-push
- ws-toolbox
- Workbox
- sw-precache
- webpack

6.3. Blog

- Progressive Web Apps on iOS are here
- Offline cookbook