MY DIGITAL SCHOOL

iocean
créateur de solutions numériques
simples et écoresponsables

Application web

BDW3 23-24

Objectifs globaux du cours

- Avoir une vue d'ensemble du développement Web
- Connaître les principaux patterns d'architecture (front-end)
- Réaliser une application avec Angular
- Consommer une API publique
- Créer une application PWA

Jour 5

Objectifs Jour 4

- PWA : les principes
- Mettre PWA sur une appli existante
- (Cookies et session)
- (Qualité du code : Linting, Tests unitaires)

PWA

PWA

PWA : Qu’est-ce que c’est ?

- **P**rogressive **W**eb **A**pp
- Site Internet (Web App) qui peut être « installé » sur le device
- Technologie axée Mobile
- Offrir une expérience fluide et performante pour l’utilisateur

➢ Docs MDN PWA

PWA : Forces et faiblesses

Avantages/Inconvénients vs. Site Web :

💪 Utilisation possible hors connexion (CacheStorage, IndexedDB)

💪 Rapide (stockage en local, caching)

❌ Ajoute de la complexité à notre Web app

PWA : Forces et faiblesses

PWA

Avantages vs. Appli Native

- 💪 Moins volumineux
- 💪 Indépendant d'un App Store / Play Store
- 💪 Facile à dev pour différents OS (Single code base)
- 💪 SEO ready
- 💪 Mises à jour servies dès qu'on a accès à internet

PWA : Forces et faiblesses

PWA

Inconvénients vs. Appli Native

- ❌ Moins de fonctionnalités du device accessibles
- ❌ Dépendant des navigateurs (compatibilité avec anciens)
- ❌ Stockage disponible moins grande qu'une appli native

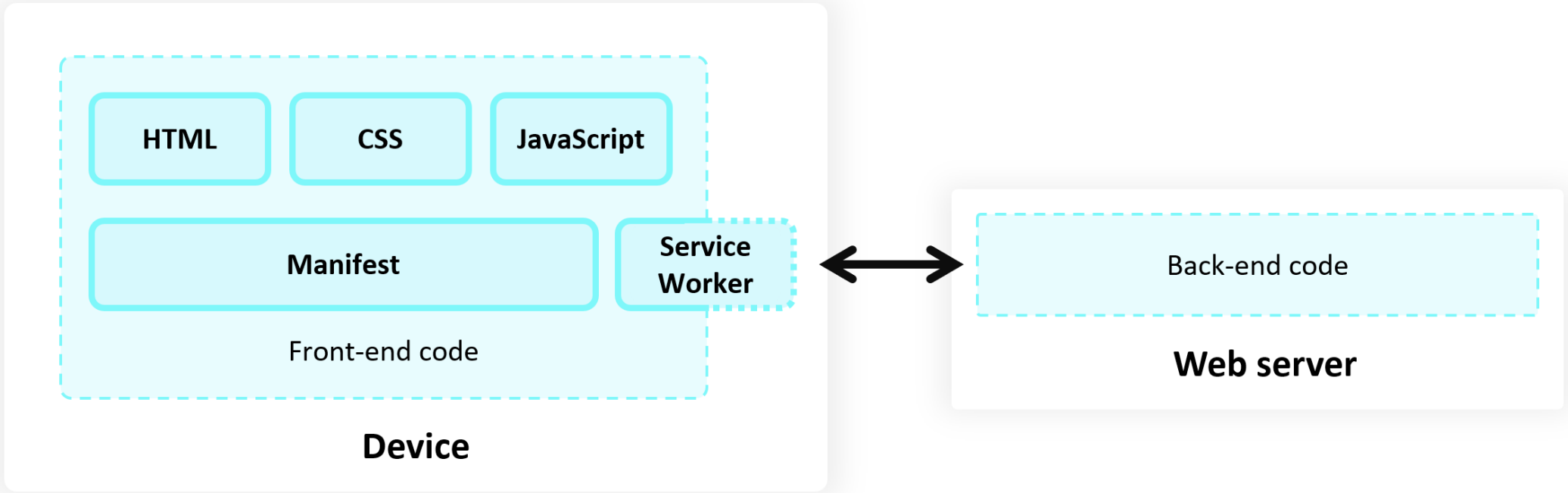
PWA : Structure

Structure requise pour une PWA :

- Une connexion HTTPS
- Un fichier manifest.json
- Un Service Worker en JS

PWA

Schéma



Source : learn.microsoft.com

PWA : Structure

HTTPS est requis pour qu'une PWA soit installable :

- Doit être servie depuis un contexte sécurisé
  - Donc en HTTPS
  - 127.0.0.1 et file:// sont considérés sécurisés.

- Éviter les attaques MitM (Manipulator in the middle)

PWA : Manifest

manifest.json (ou **manifest.webmanifest**)

- Donne des informations sur l'appli au navigateur
- Contient des méta-données
  - Nom
  - Couleur primaire de l'interface
  - Icônes
  - Splash-screen

- MDN Manifest (pour applis web & PWA)

Exemple

```
{
    "name": "Pet Clinic",
    "short_name": "PetClinic",
    "start_url": "index.html",
    "display": "standalone",
    "background_color": "#172554",
    "theme_color": "#172554",
    "orientation": "portrait-primary",
    "icons": [
        {
            "src": "img/icon/logo-72.png",
            "type": "image/png",
            "sizes": "72x72"
        },
        {
            "src": "img/icon/logo-128.png",
            "type": "image/png",
            "sizes": "128x128"
        },
        {
            "src": "img/icon/logo-192.png",
            "type": "image/png",
            "sizes": "192x192"
        },
    ]
}
```

PWA : Manifest

Champs requis pour Chromium :

- `name`
- `icons`
- `start_url`
- `display` et/ou `display_override`

PWA : Manifest

Déployer le manifest

- On le lie via un link (généralement dans index.html) :

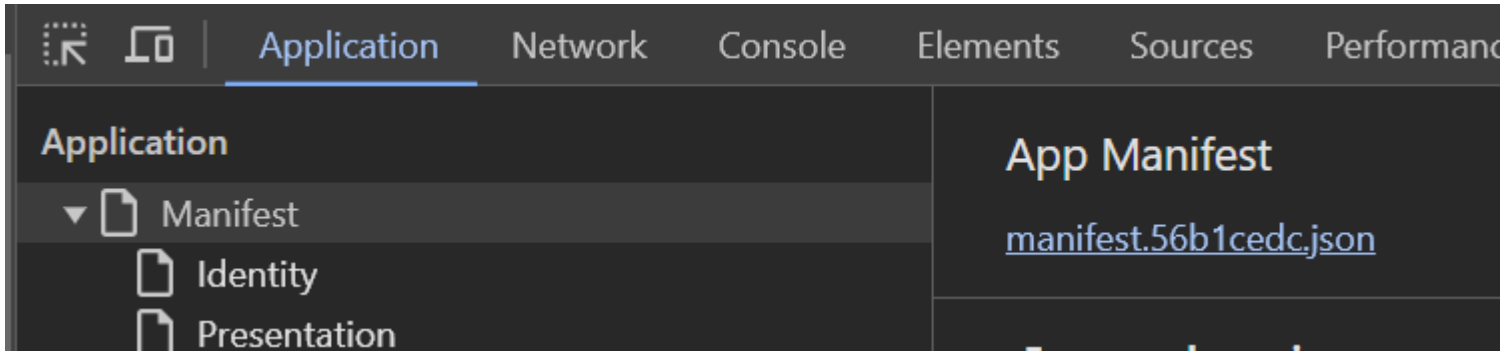
```
<link rel="manifest" href="/manifest.json" />
```

PWA : Manifest

manifest.json

Vérifier le fonctionnement

- S’il est correct, il est visible ici dans les devtools :



PWA

Service Worker

- Fichier JS
- Rend possible le fonctionnement offline
- Sorte d'intermédiaire entre le réseau, le navigateur et la Web App
  - Pourra intercepter les requêtes HTTP
- Responsabilités : caching, content updates, notifications, …

➢ MDN Service Worker

PWA

Service Worker : ses particularités

- C’est un *worker* : n’accède pas au DOM
- S’exécute dans un *thread* différent du *thread* JS principal
- Non bloquant et asynchrone
- Fonctionne uniquement avec le protocole HTTPS

- Il contrôle une page web.
- Il doit être déclaré (slide suivante)

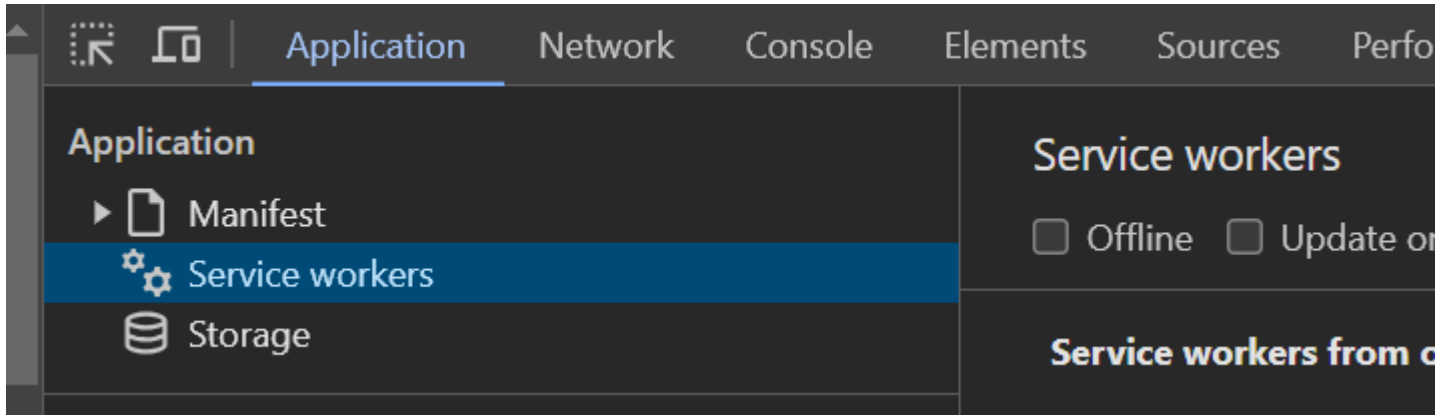
PWA

Service Worker

Déployé de la manière suivante (balise script) :

```
if('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js');
}
```

On peut vérifier s'il est bien présent ici :



MY DIGITAL SCHOOL

iocean
créateur de solutions numériques
simples et écoresponsables

PWA

Service Worker : le lifecycle

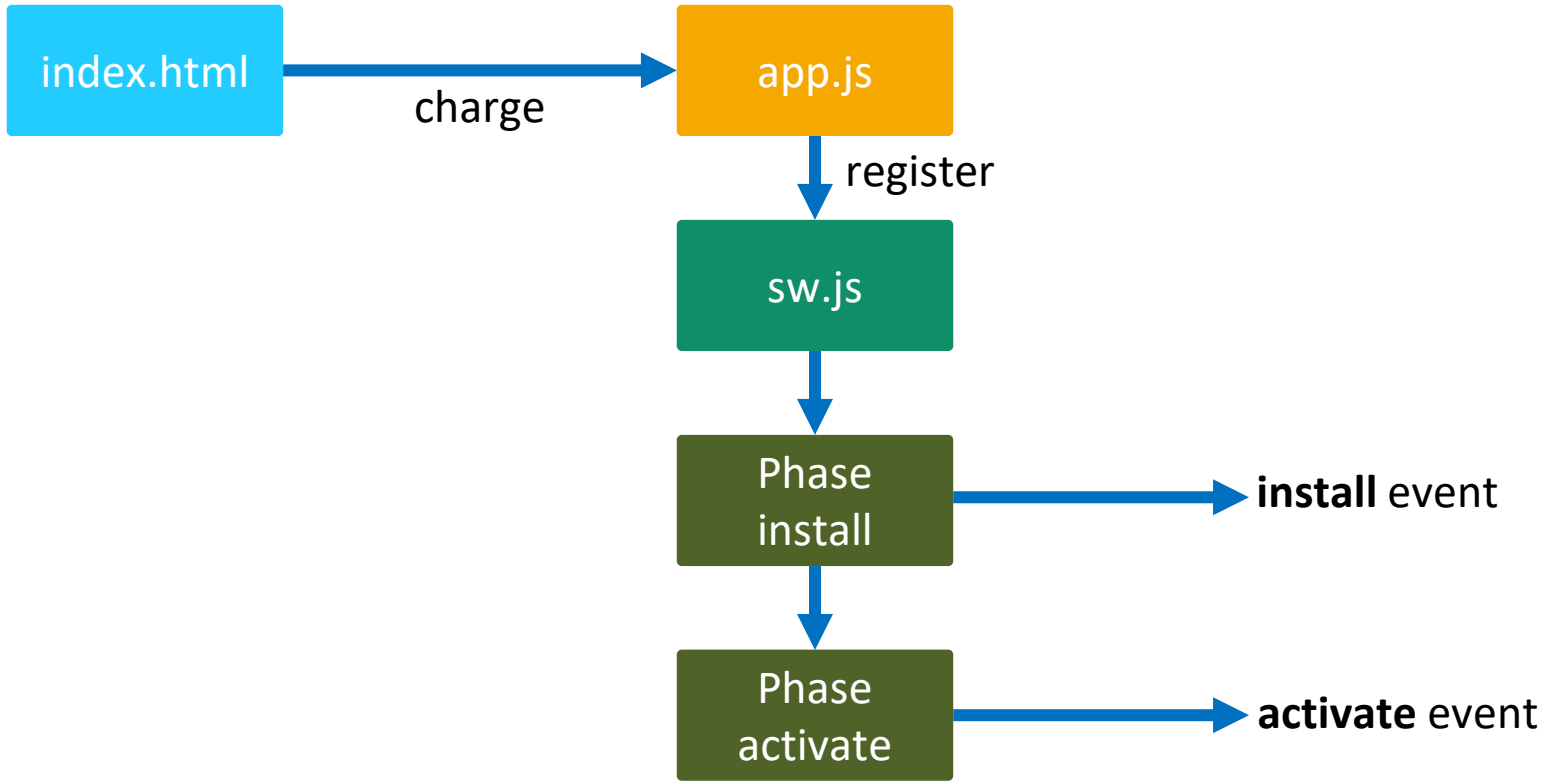
1. Lors de la déclaration, il est **téléchargé** puis lancé.
2. Une fois déclaré, la phase **install** démarre.
   - On met en cache à cette étape, sans toucher à l'existant (versionning)
3. Une fois installé, la phase **activate** démarre.
   - On *prune/update* le cache à cette étape si besoin.

➢ Schéma animé

PWA

Service Worker

Service Worker : le lifecycle



PWA

Service Worker : le lifecycle + update

- Les navigateurs vérifient que le *sw* n'a pas changé à chaque navigation sur une page gérée par le *sw*
  - *Vérification octet par octet*
- Si différence, la nouvelle version sera téléchargée en arrière plan
- Puis, phase *install*, *wait* et dès que l'ancien *sw* a fini d'être utilisé, *activate.*
  - On peut forcer l'utilisation du nouveau sw en écrivant *self.skipWaiting()*

➢ Schéma animé

PWA

Service Worker : son fonctionnement

Écouter certains évènements :

- Install = exécuté 1 fois au moment où on installe la PWA
- Activate = exécuté au moment où un sw s'active
- Fetch = au moment où on fait une requête
- Sync = au moment où le browser regagne une connexion

PWA

Caching et Opérations hors ligne

- Pour que la PWA fonctionne bien hors connexion, il faudra utiliser l'event **fetch**
- Permettra de vérifier dans le cache avant de faire partir la requête.

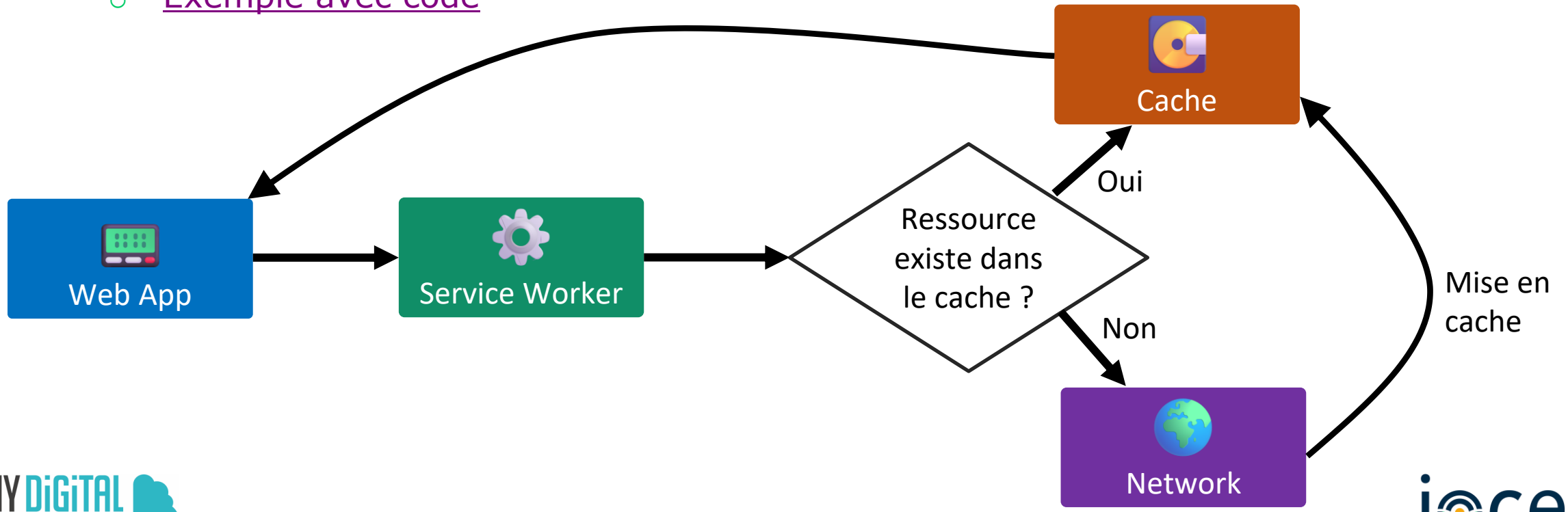
👍 Gains en rapidité (pas de requêtes vers le serveur)

👍 Gains en disponibilité (accessible même si pas de connexion)

👎 Gérer un cache est complexe. Il faudra décider d'une stratégie de mise en cache.

PWA

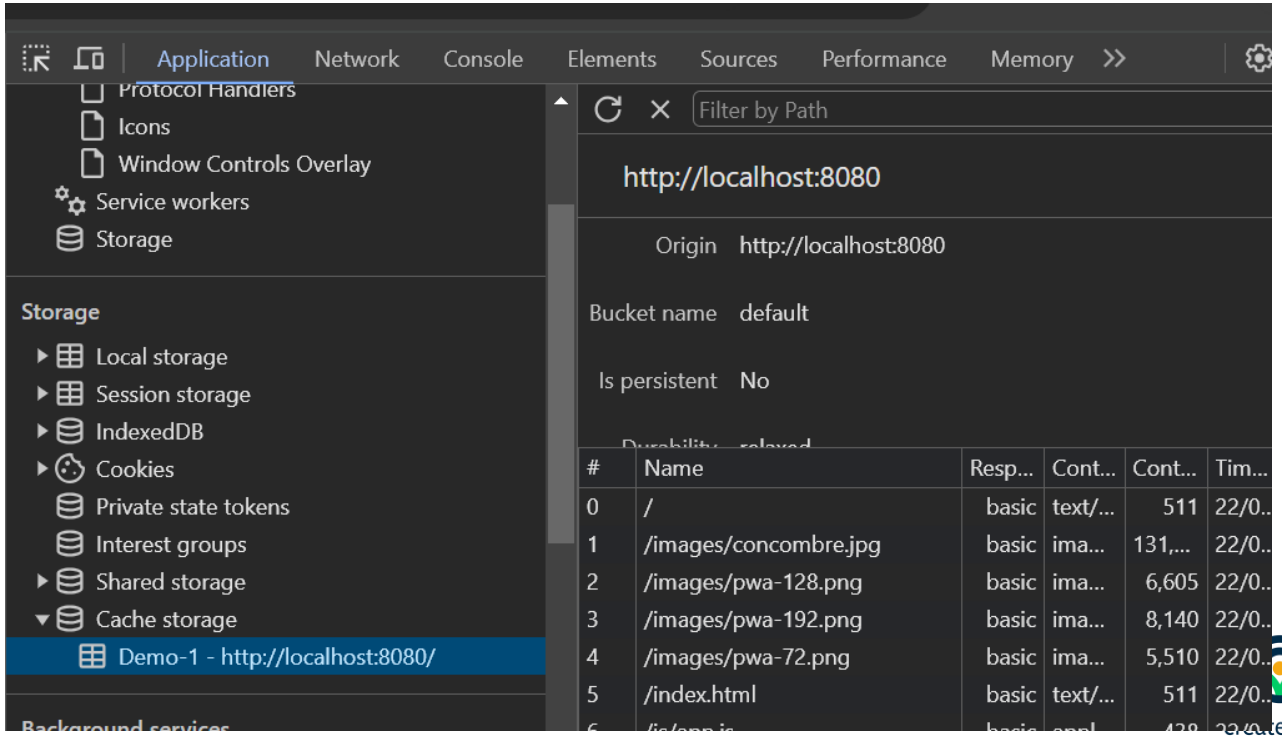
- Exemple d'une stratégie « Cache first, fallback to network »
  - Avantage : gain de rapidité, diminution de bande passante
  - Inconvénient : *staleness* des ressources
  - Bien pour : les ressources statiques qui changent rarement
  - Exemple avec code



PWA

Service Worker

- Se base sur l'interface Cache (MDN) pour mettre en cache
  - Différent du cache HTTP
  - Programmable en JS



PWA + Frameworks

PWA + Frameworks

- La plupart des Frameworks JS modernes contiennent des fonctionnalités prenant avantage de la technologie PWA
- Permet de transformer notre app React, Vue, Angular, etc en PWA facilement
- Les SPAs se prêtent bien à l'utilisation
  - car 1 seule page gérée par 1 seul worker, facile à mettre en cache

PWA avec Angular

PWA avec Angular

Façon de gérer la PWA par Angular

- Angular a une certaine opinion sur la gestion du cache
- Fait beaucoup de choses pour nous
  - Icones, manifest…
  - Embêtant si on veut avoir la main dessus.
- Utilise un *service worker* tout fait
  - On n'aura pas directement accès au sw
  - On y accède via les services injectables SwPush et SwUpdate

PWA avec Angular

PWA + 

Les choses à faire pour convertir une app Angular en PWA

- Commande du CLI :

Terminal

```
$ ng add @angular/pwa
```

PWA avec Angular

Terminal

```
$ ng add @angular/pwa
```

- package.json : ajout de `@angular/service-worker`
- Ajout des fichiers `manifest.webmanifest`, `ngsw-config.json`
- Ajout des icônes dans `assets/icons`
- Modifie app.config.ts pour *register* le *sw*
- angular.json
  - ajout du webmanifest dans assets
  - configuration du build pour prendre en compte le service worker
- Modification de index.html
  - Balise link vers le manifest
  - Balise meta theme-color

PWA avec Angular

PWA + 

Le fichier **ngsw-config.json**

- Configuration du *sw* pour Angular.
- Le gros du boulot : choisir quoi mettre en cache et comment.
- On utilisera les propriétés **assetGroups** et **dataGroups**
  - Chaque item à l'intérieur correspond à un groupe de ressources à mettre en cache

PWA avec Angular

PWA + 

assetGroups

- On y mettra les ressources que l'on veut recharger à chaque nouvelle version de l'app

- Par ressources, on entend les fichiers html, js, css, et les assets dont a besoin notre app
  - Les données provenant d'une API / d'un backend seront généralement mis dans **dataGroups**

- C'est par ce biais que l'on pourra rendre notre app utilisable hors connexion

PWA avec Angular

PWA + 

assetGroups : installMode

- installMode: prefetch
  - Dit à Angular de mettre en cache toutes les ressources listées, **de façon eager**

- installMode: lazy
  - Dit à Angular de mettre en cache les ressources **lorsqu'elles sont utilisées pour la première fois**

PWA avec Angular

PWA + 

assetGroups : updateMode

- Détermine le comportement des ressources en cache lorsqu'une nouvelle version est disponible
  - Nouvelle version = changement de code dans notre appli
- updateMode : prefetch
  - Télécharge et met en cache les ressources qui ont changé immédiatement
- updateMode : lazy
  - Télécharge et met en cache les ressources qui ont changé lorsqu'elles sont demandées

PWA avec Angular

PWA + 

dataGroups

- Contrairement aux *assetGroups*, ils ne sont pas versionnés
  - Càd : ne sont pas rechargés en fonction des versions de l'app
  - On peut cependant mettre une version sur un assetGroup afin de l'invalider à la prochaine update.

- Mis en cache conformément à une politique qui est plus utile pour des appels API
  - Basé sur plusieurs facteurs : âge, timeout, taille du cache…

PWA avec Angular

PWA + 

dataGroups : cache config

- **maxSize** : nombre maximal d'éléments à mettre en cache
  - Si ça dépasse, on expulse les données du cache en trop
- **maxAge** : définit un âge maxi au bout duquel les données seront considérées comme périmées
- **timeout** : durée du *network timeout* si on se base sur le réseau pour la donnée demandée
- **strategy :**
  - **performance** : optimise la rapidité (cache first)
    - Besoin de définir un *maxAge* pour rafraichir le cache au bout de x temps
  - **freshness** : optimise pour diminuer la *staleness* (network first)
    - Besoin de *timeout* pour se rabattre sur le cache en cas de lenteur

PWA avec Angular

PWA + 

Tester en local

- Important : pour tester les fonctionnalités de la PWA et du service worker (mise en cache spécifiquement), il ne faut pas passer par ng serve

- **Pour lancer le serveur, utiliser ces commandes :**

Terminal

```
$ ng build

$ npx http-server -p 8080 -c-1 dist/<project-name>/browser
```

Build le projet puis sers l'app depuis un serveur http sur le port fourni