# Angular PWA

Service workers essentially act as proxy servers that sit between web applications, the browser, and the network (when available). They are intended, among other things, to enable the creation of effective offline experiences, intercept network requests and take appropriate action based on whether the network is available, and update assets residing on the server. They will also allow access to push notifications and background sync APIs.

Service Worker configuration file ngws.service.ts

Set serviceWorker true in angular.json

To disable any caching header of http-server

```
http-server -c-1 -P http://localhost:9000 .
```

**Service Worker** caches static assets but not api responses. Service worker is a Network level http interceptor. A service worker is run in a worker context: it therefore has no DOM access, and runs on a different thread to the main JavaScript that powers your app, so it is non-blocking. It is designed to be fully async; as a consequence, APIs such as synchronous XHR and Web Storage can't be used inside a service worker.

Service workers only run over HTTPS, for security reasons. Having modified network requests, wide open to *man in the middle* attacks would be really bad. In Firefox, Service Worker APIs are also hidden and **cannot be used when the user is in private browsing mode.**

Service Worker kill switch. Just disable service worker from **angular.json.**

Check Cache Storage in Application.

## Caching

```
"dataGroups": [
  {
    "name": "lessons-api",
    "urls": [
      "/api/lessons"
    ],
    "cacheConfig": {
      "strategy": "performance",
      "maxAge": "1d",
      "maxSize": 100
    }
```

```
  }
]
```

**urls** define urls to be cached.

**strategy: performance** defines Service Worker cache first and then network.

**maxAge** is max time/day to be cached.

**maxSize** The maximum number of entries, or responses, in the cache. Open-ended caches can grow in unbounded ways and eventually exceed storage quotas, calling for eviction.

## Fallback to Cache

This strategy states that we hit an api and if the response if response does not come in some time(timeout) then load the cached version and cancel the request.

strategy: freshness

**timeout** This duration string specifies the network timeout. The network timeout is how long the Angular service worker waits for the network to respond before using a cached response, if configured to do so.

```
"dataGroups": [
  {
    "name": "lessons-api",
    "urls": [
      "/api/lessons"
    ],
    "cacheConfig": {
      "strategy": "freshness",
      "timeout":"10s",
      "maxAge": "1d",
      "maxSize": 100
    }
  }
]


After 10seconds, it will cancel the network request and use cached version.
```

# PWA One-Click Installable

We need manifest.json file. Specify manifest.json in assets array and in index.html

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

```
Now, do npm run start:prod. Check F12 > Application > Manifest
```

```
You will be missing, install any icon generator packages
```

# Application Shell (Pre render/Universal)

Adding Application Shell means adding universal to page and adding an app shell to show something before angular loads. This

```
ng generate universal anyName --client-project="project name"
```

`--client-project` map this with `angular.json` value.

Now, do npm install

Changes done by this command:

1. package.json: @angular/platform-server was added.
2. angular.json **apps** key.

ng generate app-shell my-shell --universal-project=ngu-app-shell --route=app-shell-path --client-project=angular-pwa-course

# Push Notifications

```
npm install web-push -g
```

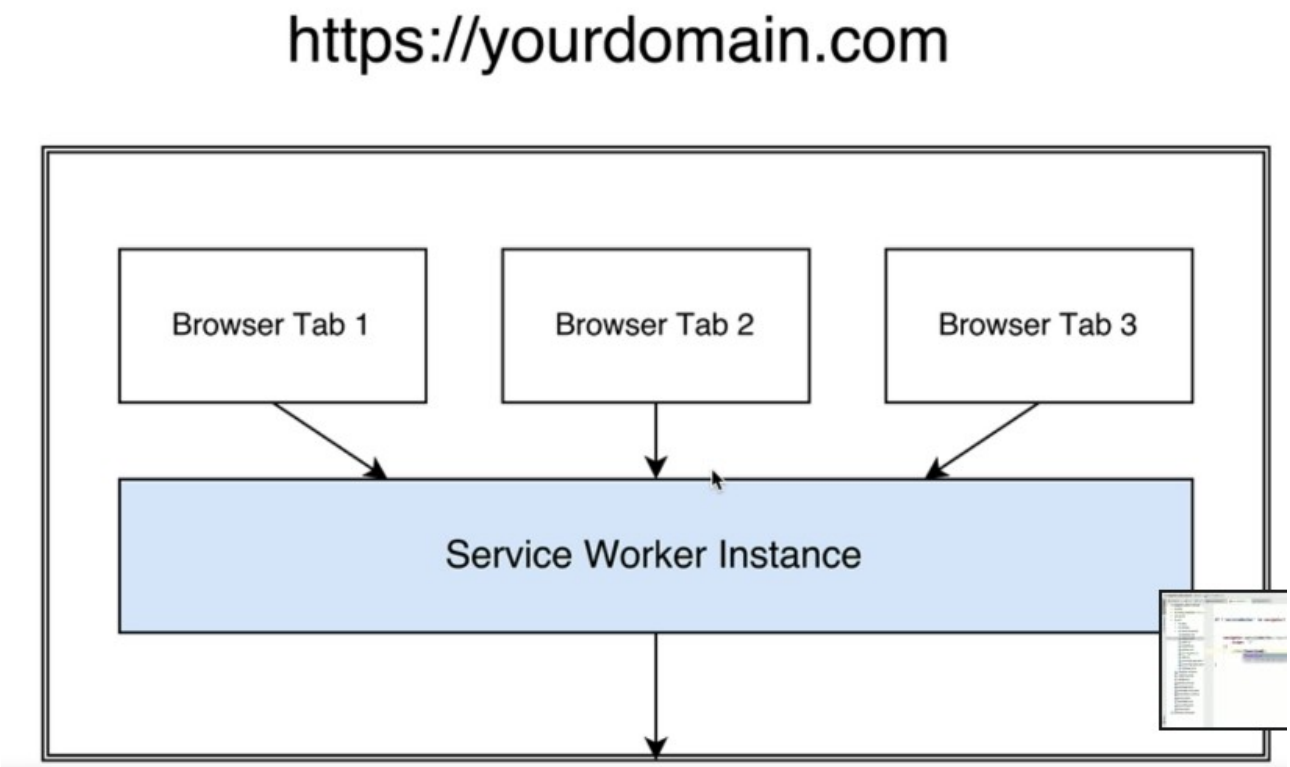
Then

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

This will generate keys.

Now, check the **angular-pwa-course** repository.

# Custom Service Worker




Create a file **sw-register.js**.

Mention it in index.html: `<script src="/sw-register.js"></script>`

***sw-register.js***

```
if('serviceWorker' in navigator) { // if exist
  navigator.serviceWorker.register('/sw.js', {
    scope: '/' // intercept all request starting with /
 }).then(registration => {
    console.log('Service worker registeration completed.');
  });
}
```

***sw.js***

```
const version = 'v1';
function log(messages) {
  console.log(version, mesages);
}
```

```
log('Installing Service worker');
```

Now, mention both the files in  angular.json assets.

Service Worker is started when atleast one tab is open.

Open sw.js, add following:

```
self.addEventListener('install', () => {
  log('version is installed');
});

self.addEventListener('activate', () => {
  log('version is activated');
});
```

**Output:**

Installing Service worker

version is installed

Now, change version=v2 and refresh, it  will be v2 and will print install but not activated.
Now, check application. If you open new tabs then also it will old v1 which is activated.
User consistent of the service worker.

New version will only be activated when user closes all the tabs and ofcorse, *Alt+Shift+R.*

## Cache Storage JS API

Create a temp page. offline.html

<h1>You are offline.</h1>

Now, we will show this page if the network is down.

From service worker perspective, it will be already **installed** but we will **activate** it.

```
self.addEventListener('install', (event) => {
  event.waitUntil(installServiceWorker());
});

async function installServiceWorker() {

  log('SW installation starting');
  const request = new Request('/offline.html');
  const response = await fetch(request); // fetch returns Promise
  if(response.status != 200) {
    throw new Error('testing');
  }

  const cache = await caches.open('app-cache'); // app-cache is random name
  cache.put(request, response);
  log('Cached offline.html');
}
```

Go to **Application > Cache Storage**. Check the **app-cache** will be there.

```
// Intercepting all network request.
self.addEventListener('fetch', event => event.responseWith(showOfflineError()));

async function showOfflineError() {
  log('Calling network: '+ event.request.url);
  let response;
  try {
    let response = await fetch(event.request); // Service worker will make
                                                    request.

  } catch(err) { // Here you will serve offline content
    log('Network request failed', err);

    const cache = await caches.open('app-cache');
    response = cache.match('offline.html'); // Serve this page
```

```
  }
  return response;
}
```

## Service worker in action

Try to register service worker as late as possible.

```
platformBrowserDynamic().bootstrapModule(AppModule).then(() => {
 //serviceworkercode
});
```

1. Try to version the cache.

   ```
   const cache = await caches.open(getCachedName());
   function getCachedName() {
     return "app-cache-"+version;
   }
   ```

2. Cache the files

   ```
   cache.addAll([
     '/',
     '/polyfills.bundle.js',
     .....
   ]);
   ```

3. There is no cache expiration. Check network tab and Cache Storage. You will see request cached by service worker in network tab in cache storage.

4. Try cached data if not then go to network.

   ```
   self.addEventListener('fetch', event =>
         event.respondWith(cacheThenNetwork()
   );

   async function cacheThenNetwork(event) {

         const cache = await caches.open(getCacheName());
         const cachedResponse = await cache.match(event.request);

         if(cachedResponse) {
               log('From Cache: '+ event.request.url);
               return cachedResponse;
         }

         const networkResponse = await fetch(event.request);
   ```

```
        return networkResponse;
    }
```

5. If you open the page firsttime, it will activate the service worker but above mentioned caching strategy will not work. It will happen on next refresh.
6. Remove old caches

```
self.addEventListener('activate', () => {
   const cacheKeys = await caches.keys();
   cacheKeys.forEach(cacheKey => {
      if(cacheKey !== getCacheName()) {
         caches.delete(cacheKey);
      }
   });
});
```

7. First time service worker will activate but will not run. To make it run first time,

```
self.addEventListener('activate', () => {
   const cacheKeys = await caches.keys();
   cacheKeys.forEach(cacheKey => {
      if(cacheKey !== getCacheName()) {
         caches.delete(cacheKey);
      }
   });
   return clients.claim();
});
```

   But this can result in inconsistency.

## skipWaiting

Suppose, there is a service worker waiting and you want to activate it then use

`self.skipWaiting();`

## Manually Checking for new service worker

```
navigator.serviceWorker.register('/sw.js', {
  scope: '/'
}).then(registration => {
  setInterval(() => {
    registration.update();
  });
});
```

Make sure to do **clients.claim()**.