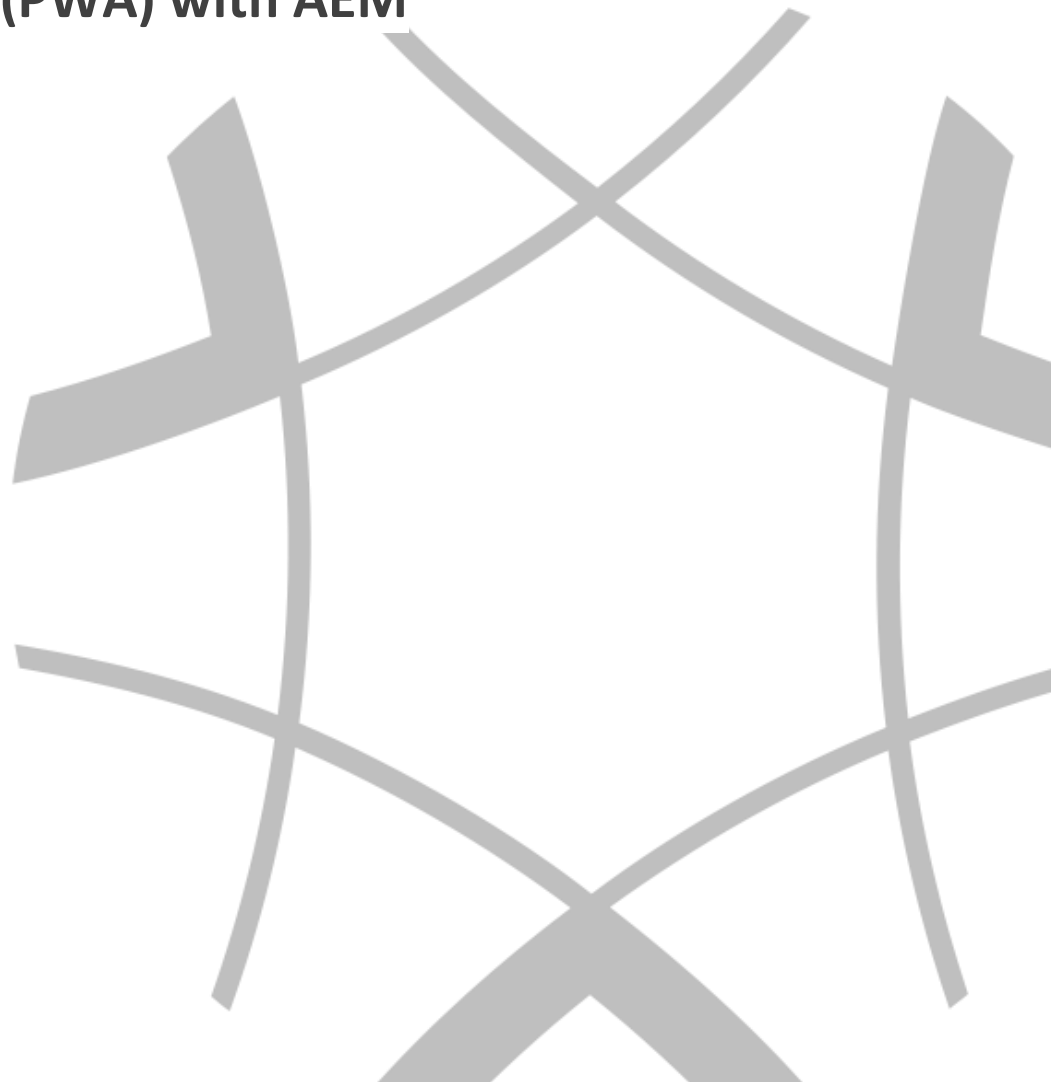
EXPERIENCE LEAGUE

LEARNING LABS

L30 – Building a Progressive Web Application (PWA) with AEM



Adobe Summit 2019

EXPERIENCE LEAGUE

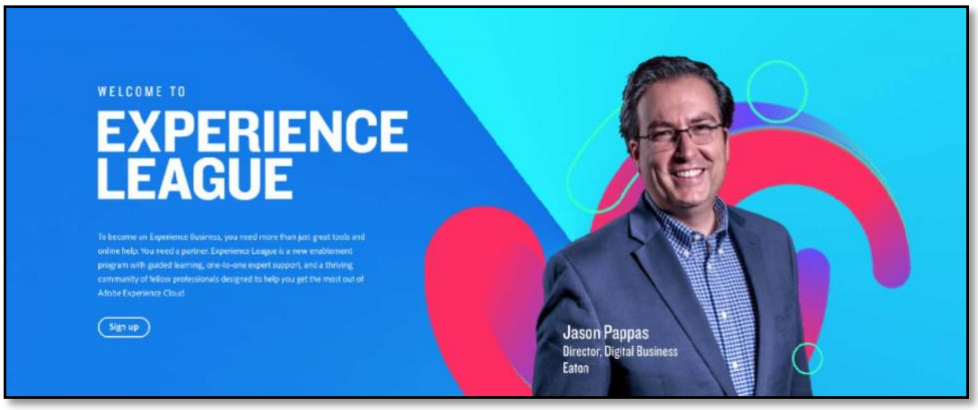
Adobe

Table of Contents

EXPERIENCE LEAGUE

Experience League

Experience Makers are made with Experience League. Kickstart your Customer Experience Management abilities with personalized learning to develop your skills, engage with a global community of your peers and earn career advancing recognition.



Intelligent Guidance

The Intelligent Guidance site is full of **recommended step-by-step learning and events** that are based on the preferences selected in your profile

Community

Get and stay connected, meet other experts like yourself, and **get answers in minutes** from our global community of practitioners. No more waiting for support tickets. Now you can **share your ideas with us** and instantly **vote with your peers on future product enhancements**. Connect to a community of 150,000+ peers to get you answers in minutes.

Connect with Experts

When you need more support, Experience League connects you with Adobe experts ready to work with you in a 1:1 setting.

Join Now

Join the League today at ***experienceleague.adobe.com***

EXPERIENCE LEAGUE

Adobe

Experience League Learning Labs

Overview

Experience League Learning Labs at Summit are guided learning sessions that connect you with Adobe experts and your peers. It's another way Experience League is helping you kickstart your Adobe know-how to get the most out of Adobe Experience Cloud. Sign up for Experience League at experienceleague.adobe.com

Environment

Fast-track your Adobe expertise with 90-minute guided learning sessions. We empower you with preloaded software on each computer, and walk you through the lab with step-by-step guidance.

Tools

Go URL

In the footer of this lab manual you'll find an **Adobe "go" URL** that will redirect you to a **dedicated thread** on our Experience League community where you'll find **all the resources & links for this lab**. In addition, we'll continue to monitor these threads for Q&A, discussion, and guidance to continue your learning.

Q&A

Each lab has reserved time for question and answer. However, if you aren't able to get your question answered during the lab or you'd prefer a written response, please use the GO URL in the footer to post your question on the community thread for this lab. Adobe has experts monitoring these threads prepared to answer your questions and ensure your success in mastering the content contained within this lab.

Continue the Conversation

Don't let this lab coming to an end be the end of our conversation. We've created a dedicated space to collaborate with you and your peers around this lab's content. Join us as you implement these concepts into your business and share your success and trails.

Community

In addition to Adobe experts from Customer Care, Product, Engineering, and other groups, we have over 150,000 of your peers on our communities.

General Questions

Search through our ever-growing goldmine of questions or ask your own and get answers in minutes.

Ideas

Have a great idea that you'd like to see Adobe implement? Come share or vote on ideas in our Experience League communities to influence product roadmap.

Feedback

Our product team often seeks your feedback by posting polls on our community. Ensure your voice is heard by participating.

All links available at adobe.com/go/summit2019-l780

Lab Overview

Marketing and Technical teams need to deliver fast, engaging and reliable experiences on different channels, but even more on mobile as customers are more likely to use their mobile for their daily activities like site searches and shopping. Progressive Web Applications bring mobile-app-like experiences to end users without requiring them to install an app from a store (Apple store/ Play store).

Many brands like Alibaba, Flipkart, Forbes, Lancome, the Financial Time, the Washington Post, Virgin America have built progressive web applications (PWA), and they have seen and still see the benefit in their conversion rate. They deliver reliable, fast and engaging content to customers that drive customer engagement and accelerate transaction processing.

This lab will help you explore Progressive Web Applications (PWA), learn how to make an AEM application become PWA to deliver compeling mobile experiences. This could be a good starting point for extending an existing website or building a new one as a progressive web application.

Key Takeaways

- **Learn how to use service workers** (JavaScript). Using the Sync Manager will help achieve a consistent user experience offline and performs well on slow connections.
- **Learn how to leverage browser caching capabilities** . Caching content will ensure the application loads quickly and ensure that guests will stay on the site as they go through different pages from page to page.
- **Learn how to send web push notifications** . They can show incredible effectiveness and help your marketing team engage new and re-engage your existing users.
- **Test your PWA on a Virtual Device** (Android). A PWA feels like a mobile app with app-style interactions but without the storage requirements or manual update issues.
- **Learn how to access to native device features** : Camera , Background sync **(Bonus)**

Prerequsites

- Google Chrome v54+
- Adobe Experience Manager (6.5)
- An Android Virtual Device (emulator) : Nexus 5 XL
- Basic knowledge of HTML, CSS, JavaScript, and Chrome DevTools
- Good understanding of AEM technical platform (OSGi & ClientLibrairies)

Lesson 0 – Get started with PWA

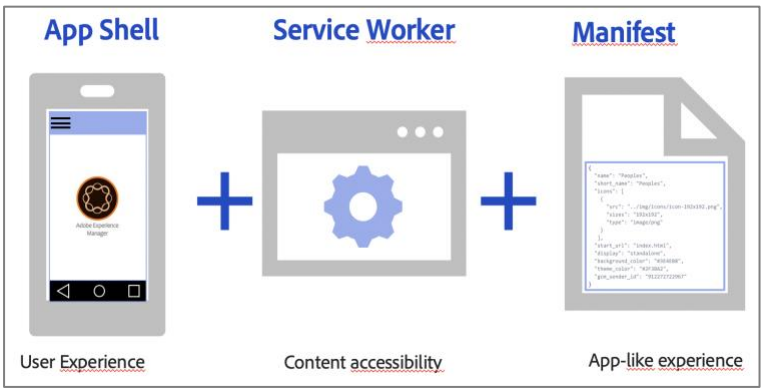
What is a Progressive web Application ?

A progressive web application (PWA) is a website that looks and behaves in the same way as a mobile application, which means that it can be added to the main screen of a smartphone, send push notifications, access the hardware of the device, and work offline. PWA are truly mobile-ready and when installed onto a mobile they will leverage a service worker in background to cache the “app shell” intercepts AJAX call via fetch API and synchronize data when network is unavailable or poor.

The benefits of a PWA

- **Progressive :** Works for every user, regardless of browser choice because it's built with progressive enhancement as a core tenet
- **Responsive** - Fits any form factor: desktop, mobile, tablet, or whatever is next
- **Connectivity** independent - Enhanced with service workers to work offline or on low-quality networks
- **App-like** - Feels like an app, because the app shell model separates the application functionality from application content.
- **Linkable** - Easily share the application via URL, does not require complex installation.
- **Fresh** - Always up-to-date thanks to the service worker update process.
- **Safe** - Served via HTTPS to prevent snooping and to ensure content hasn't been tampered with.
- **Discoverable** - Is identifiable as an "application" thanks to W3C manifest and service worker registration scope, allowing search engines to find it.
- **Re-engageable** - Makes re-engagement easy through features like push notifications.
- **Installable** - Allows users to add apps they find most useful to their home screen without the hassle of an app store.

PWA building blocks



- A web app **manifest**
- A **service worker**
- Caching **app shell**
- Web push **notifications**
- Bonus
    - Devices features : **Camera**
    - Background **synchronization**

Lab Description

In this lab we will build a simple blog website. This website will provide to the end users the following features :

- List pictures on the pwa home page pulled from an AEM DAM Folder using fetch request and a Sling Model Exporter.
- Create a post with the following items :
    - a picture
    - a title
    - some tags : coma separated list
- Authenticate on the PWA as Sylvia Burke (the credentials are provided below).
- Suscribe to push notifications and receive them from AEM.

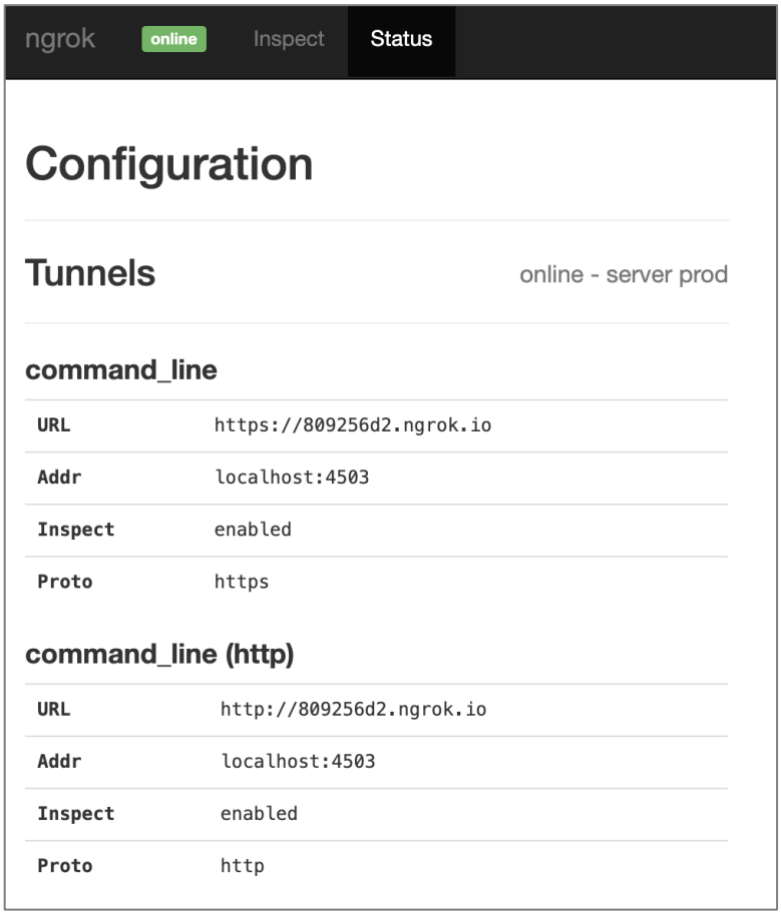
The lab will executed onto an **AEM publish** instance. As AEM uses a cookie-based authentication by default, we will use the local IP address (127.0.0.1) for updating the code and the local DNS (localhost) for authenticating users on the blog. This will avoid HTTP cookies conflicts. You will use these users during the lab :

Hostname	Credentials	Role
http://localhost:4503	Sylvia Burke - username : **sburke** / password : **sburke**	End user who will authenticate on the PWA both in the desktop browser and on the mobile PWA.
http://127.0.0.1:4503	Administrator – username : **admin**/ password : **admin**	Developper building a PWA for Sylvia Burke

Setup

- Open Chrome (Desktop) and navigate to http://127.0.0.1:4503/crx/packmgr/index.jsp
- Check if the lab package (**aem-pwa-blog.ui.reactor-1.3.zip**) is installed.

- Navigate to http://127.0.0.1:4040/status and verify if the ngrok proxy is started.



Lesson 1 – Create the Web App Manifest

Objectives

1. Understand the web app manifest usage within a PWA.
2. Deploy a manifest file using AEM capabilities.

Lesson Context

Web browsers do not consider all websites as Progressive Web Applications (PWA). **The first requirement for a web application to become a PWA is to have a web app manifest**.
The manifest is a file read by the browser to check if the application could be installed onto a user's device. The informations provided help to know if the website could considered as a PWA. The manifest follows a W3C standard and has a set of values expected.

Technical Context

The manifest is a JSON file which gives some informations about the web application and how it should behave when 'installed' on mobile device or desktop.

The manifest should be exposed as a JSON resource by AEM. The manifest has a clear structure imposed by the W3C. The easiest approach from an AEM perspective to have a manifest compliant with W3C standard is to expose the manifest as a Sling servlet ***(com.adobe.summit.emea.core.servlets.ManifestServlet)*** *that will be configured with an OSGi R6 configurations interface.*

Exercise 1.1 – Check the Manifest Servlet

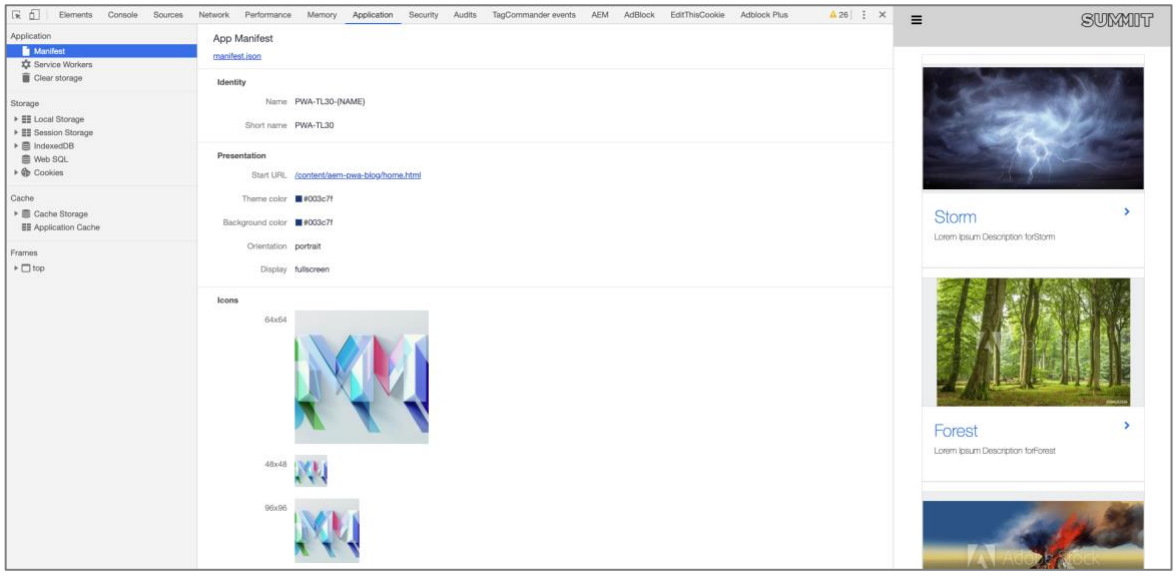
1. Check the manifest availability : http://localhost:4503/content/manifest.json , You will see some JSON content.

Exercise 1.2 – Enable the manifest file

1. Navigate to CRXDE, authenticate on http://127.0.0.1:4503/crx/de/index.jsp with the admin user's credentials.
2. Navigate to CRXDE, update the page component customheaderlibs.html file (**/apps/aem-pwa-blog/components/structure/page/customheaderlibs.html**) file : http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/components/structure/page/customheaderlibs.html by adding the following code:

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

3. Open the home page (**/content/aem-pwa-blog/home.html**) of TL30 blog : http://localhost:4503/content/aem-pwa-blog/home.html
    - Open the Chrome Developer Console by navigating like this :
        i. Click on the **"Application"** tab
        ii. Select "**Manifest"** on the left panel



The servlet has been configured with a `sling:osgiConfig` node (**/apps/aem-pwa-blog/config/com.adobe.summit.emea.core.servlets.ManifestServlet**) available at this location http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/config/com.adobe.summit.emea.core.servlets.ManifestServlet

The icons seen in the screenshot above are loaded from this folder (**/etc/clientlibs/aem-pwa-blog/icons**) http://127.0.0.1:4503/crx/de/index.jsp#/etc/clientlibs/aem-pwa-blog/icons

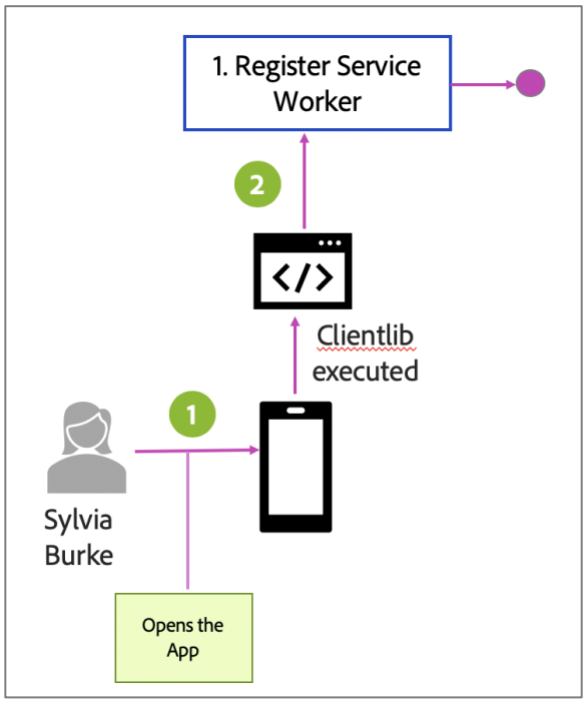
Lesson 1 completion status

The first requirement for the AEM application to become a PWA is met. The application is installable as a Progressive Web Application in a mobile device. Lesson 1 is completed.

1.3 - Go further

[0] https://www.w3.org/TR/appmanifest/
[1] https://developers.google.com/web/fundamentals/web-app-manifest/

Lesson 2 – Adding a service worker



Objectives

1. Understand what a service worker is and its usage within a PWA.
2. Deploy a service worker file using AEM capabilities.

Lesson Context

Customers use their mobile applications on a daily baisis to perform many “straightforward” operations like searching content and purchasing products. Mobile applications perform many background operations like push notifications and content synchonization. PWA uses browser APIs to perform some of these operations by leveraging service workers. The service worker can use these APIs :

- the Cache API for caching content
- the Fetch API for network calls
- the Sync API for data synchronization
- the Push API for managing notifications.

These APIs consume less resources than a native application and give to PWA great engaging capabilities on mobile.

Technical Context

A service worker is a script that your browser runs in the background, separate from a web page. Service Workers run in a different context, thus have no access to DOM elements or JavaScript variables in the main thread. Service workers work asynchronously.
The service worker allows your application to access features that don't need a web page or user interaction.Several events can be caught by a service worker.

AEM will render the service worker as a clientlib from the following paths :

- /etc/designs/aem-pwa-blog/sw.js
- /etc/clientlibs/ aem-pwa-blog /sw.js
- /etc.clientlibs/ aem-pwa-blog /sw.js

These files are not at the root location of the AEM website as web browsers expected them for this path /content/aem-pwa-blog. As per service worker requirements in web browsers, we need to put the server worker at this location: /content/sw.js. This is the reason why we have an OSGi Servlet exposed at /content/sw.js that will read the content of /etc/clientlibs/aem-pwa-blog/sw.js

2.1 – Check the Service Worker status

1. Check the service worker clientlib availability : http://localhost:4503/content/sw.js , You will see some Javascript content.

2.2 – Registering the Service Worker

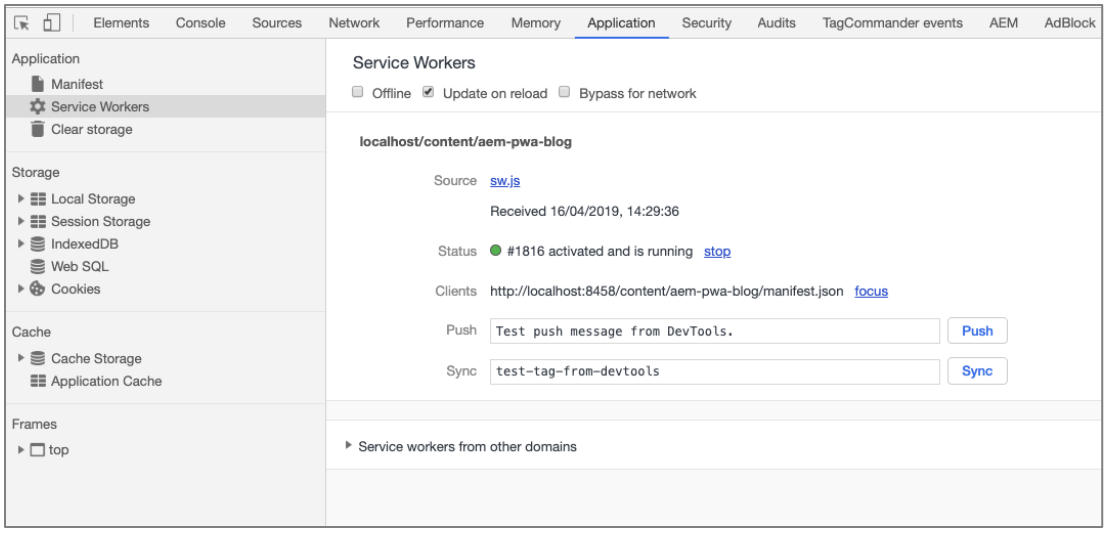
In CRXDE, Copy the “register service worker” code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-02/ex02-code-to-paste.txt**) from http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/code-snippets/exercise-02/ex02-code-to-paste.txt into the clientlib which register the service worker (**/apps/aem-pwa-blog/clientlibs/clientlib-exercises/scripts/ex02_register-service-worker.js**) http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/clientlibs/clientlib-exercises/scripts/ex02_register-service-worker.js by adding the following code in the init function:

a. Location for service worker (SW) registration

```
$(window).load(function() {
    /**
     ================================================================================================

     Exercise 02 : Register the service worker
     -----------
     Copy the code from this file : /apps/aem-pwa-blog/code-snippets/exercise-02/ex02-code-to-paste.txt
     below this commented block  :

     ================================================================================================
     **/

});
```

What does this source code do ?
It will register the file at /content/sw.js as service worker and initialize the other scripts which rely on the service worker object.

2. Open the home page (**/content/aem-pwa-blog/home.html**) of TL30 blog : http://localhost:4503/content/aem-pwa-blog/home.html
    a. Open the Chrome Developer Console by navigating like this :
        i. Click on the **"Application"** tab
        ii. Select **"Service Worker"**



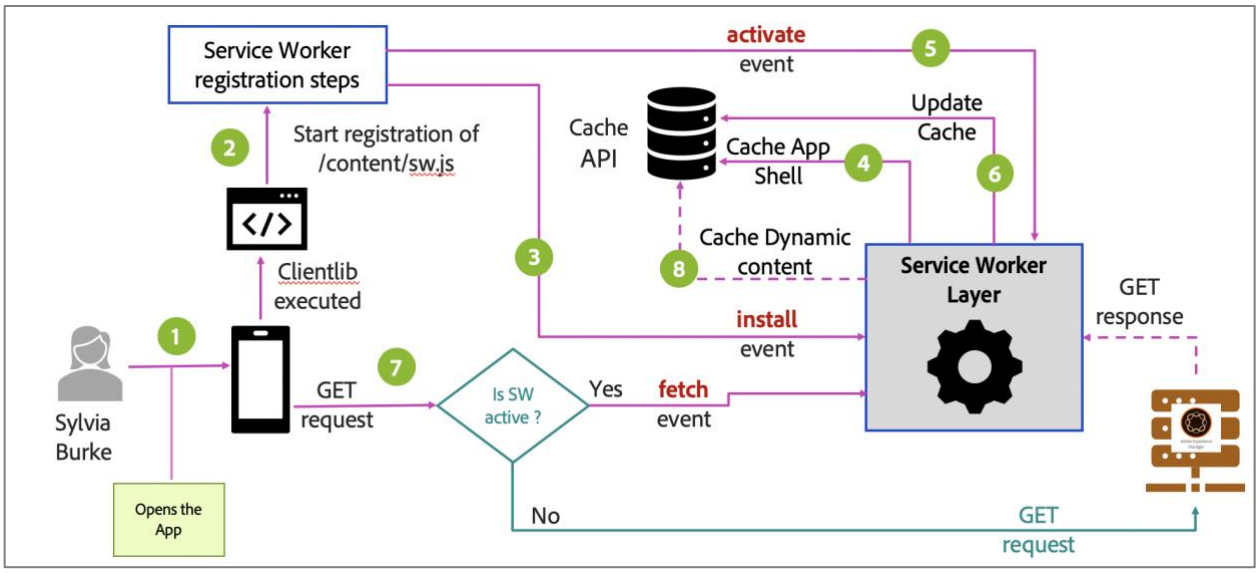
2.3 – Lesson 2 completion status

The second building block of a PWA is ready to use. The application uses a service worker. The service worker will be used in the Progressive Web Application installed on a mobile device.

2.4 - Go further

- [0] https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers
- [1] https://www.w3.org/TR/service-workers-1/
- [2] https://jakearchibald.github.io/isserviceworkerready/#moar

Lesson 3 – Caching the app shell



Objectives

1. Understand how to cache files with a service worker and keep a consistent display even offline.

Lesson Context

The app "shell" is the minimal HTML, CSS and JavaScript required to power the user interface. This shell will be by the browser ensuring to the end user a consistent offline experience and fast loading times for users repeating visits.. This means the application shell is not loaded from the network every time the user visits. This lesson will allow you to understand how to user service worker events and browser Cache API.

3.1 – Caching static content

In CRXDE, Copy the “static cache” code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-03/ex03-code-to-paste-static.txt**) from http://127.0.0.1:4503/crx/de/index.jsp - /apps/aem-pwa-blog/code-snippets/exercise-03/ex03-code-to-paste-static.txt into the service worker file (**/etc/clientlibs/aem-pwa-blog/sw.js**) : http://127.0.0.1:4503/crx/de/index.jsp#/etc/clientlibs/aem-pwa-blog/sw.js**.**The code should be added into the callback function of the **“install”** event listener:

```
self.addEventListener('install', function (event) {
    console.log('[TL30-PWA][install] Installing Service Worker ...', event);

});
```

If the browser determines that a service worker is outdated or has never been registered before, it will proceed to install it. This is a good event to prepare the Service Worker to be used, by initializing a cache, and cache the App Shell and static assets using the Cache API.

3.2 – Caching dynamic content

In CRXDE, Copy the "dynamic cache" code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-03/ex03-code-to-paste-dynamic.txt**) from http://127.0.0.1:4503/crx/de/index.jsp - /apps/aem-pwa-blog/code-snippets/exercise-03/ex03-code-to-paste-dynamic.txt into the service worker file (**/etc/clientlibs/aem-pwa-blog/sw.js**) at http://127.0.0.1:4503/crx/de/index.jsp#/etc/clientlibs/aem-pwa-blog/sw.js **.**The code should be added into the callback function of the **"fetch"** event listener :

```
self.addEventListener('fetch', function (event) {
    console.log('[TL30-PWA][fetch] >>>>> Catching an HTTP  request ['+event.request.url+'] by the Service Worker ....');

});
```

A fetch event is fired when a resource is requested on the network. The service worker will be able to look in the cache before making network requests. The lab codebase uses the Cache API to check if the request URL was already stored in the STATIC-CACHE, and return the cached response if this is the case. Otherwise, it executes the fetch request, cache the response into the DYNAMIC-CACHE and then return the response.

3.3 – Deleting caches when a service worker is updated

In CRXDE, Copy the "delete cache" code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-03/ex03-code-to-paste-delete.txt**) from http://127.0.0.1:4503/crx/de/index.jsp - /apps/aem-pwa-blog/code-snippets/exercise-03/ex03-code-to-paste-delete.txt into the service worker file (**/etc/clientlibs/aem-pwa-blog/sw.js**) available at : http://127.0.0.1:4503/crx/de/index.jsp#/etc/clientlibs/aem-pwa-blog/sw.js**.** The code should be added into the callback function of the **"activate"** event listener :

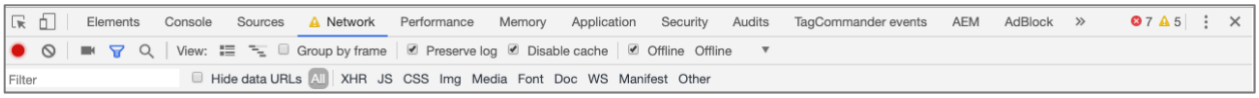
```
self.addEventListener('activate', function (event) {
    console.log('[TL30-PWA][activate] Activating Service Worker ....', event);

});
```

The activation stage is the third step in a service worker lifecycle, once the service worker has been successfully registered and installed. The service worker will be able to work with new page loads. This is the good event to cleanup old caches and things associated with the old version but unused in the new version of the service worker.

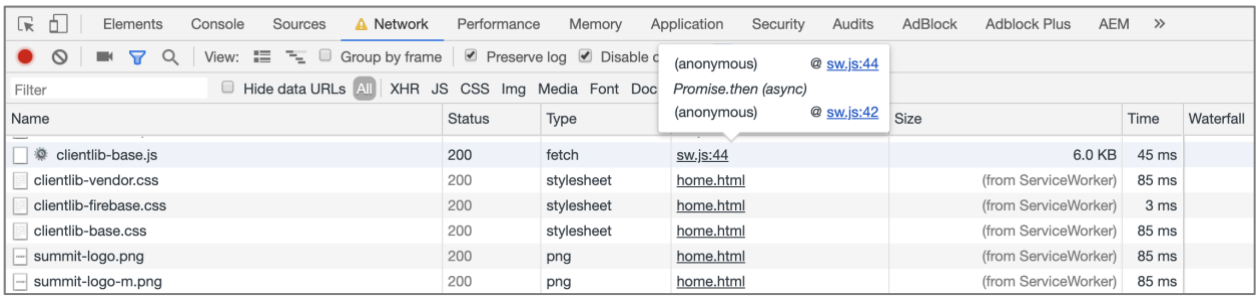
1. **Update the cache version number, by increasing the VERSION variable value in the service worker file (/etc/clientlibs/aem-pwa-blog/sw.js) : this will force the service worker to update the caches in the "activate" state of its lifecycle.**

3.4 - Test caching capabilities

1. Open the home page (**/content/aem-pwa-blog/home.html**) of TL30 blog : http://localhost:4503/content/aem-pwa-blog/home.html
    b. Open the Chrome Developer Console by navigating like this :
        i. Click on the **"Network"** tab
        ii. Check the offline checkbox

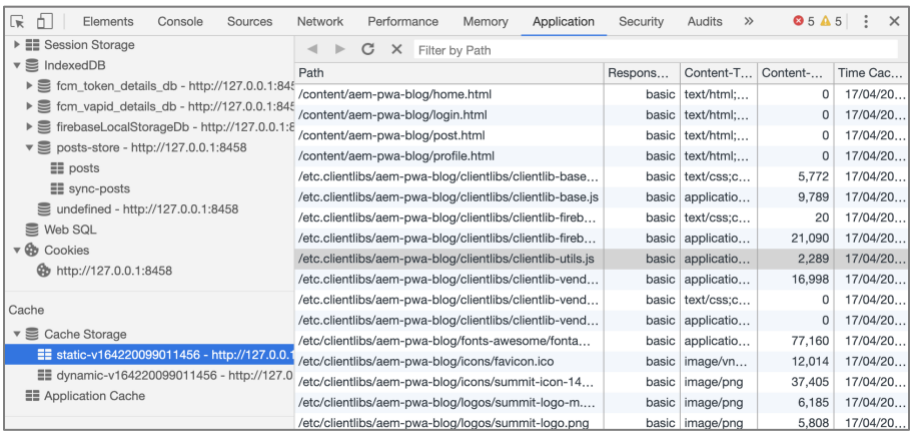


        iii. Reload the page



You will notice that in the size column the origin of the request will be **(From ServiceWorker).**

        i. Check cache content in the cache storage from the **"Application"** tab

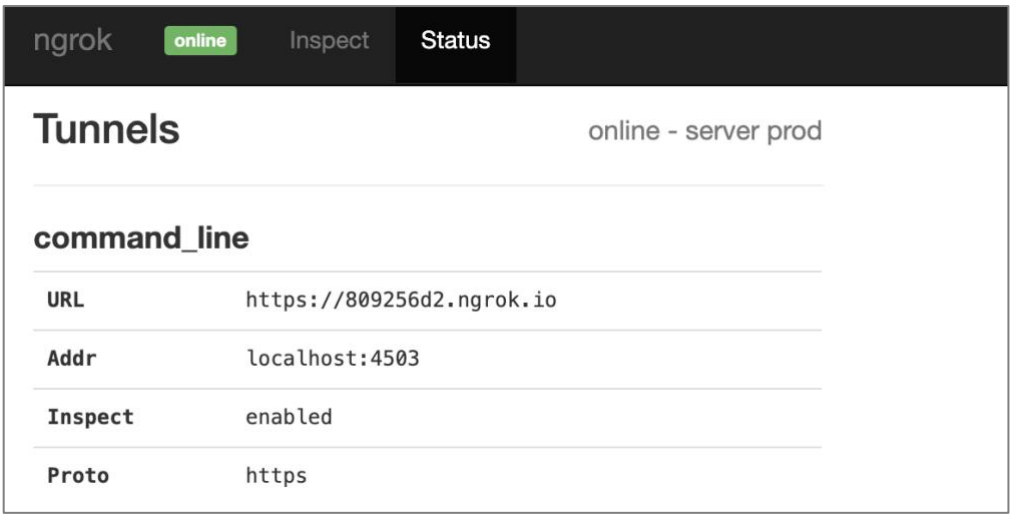


3.5 – Lesson 3 completion status

The third building block of a PWA is ready to use. The application uses the cache API from a service worker. The three main requirements (manifest, service worker, app shell) for our AEM website to become a PWA are met. We will now install it into an Android mobile device.

3.6 Installating the PWA

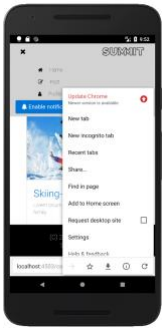
The AEM instance is exposed on the internet using ngrok.io as a web proxy. The url of the instance can be retrieved using this link http://localhost:4040/status . You will see a page



There will be an HTTP link available but in this lab we will user the HTTPS link because this a service worker requirement.

Open Chrome in the Android emulator

Copy this link https://your-ngrok-id.ngrok.io/content/aem-pwa-blog/home.html into the url input field and validate

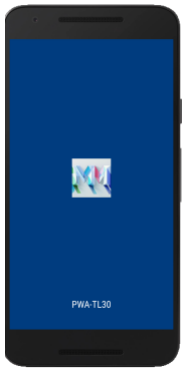


In Chrome, when the site is loaded a popup window should appear, if not click on the menu navigation, then click on **"Add to Home Screen"**

You will see a popup window appearing at the bottom of the screen, with an "**ADD**" button. Click on it.

You will see a modal window in the middle of the screen. Click on **''ADD AUTOMATICALLY"**

The **PWA-TL30** will be installed in the home screen. Click on the PWA icon to start the application.



The home screen of you PWA will appear according to the informations from the manifest.

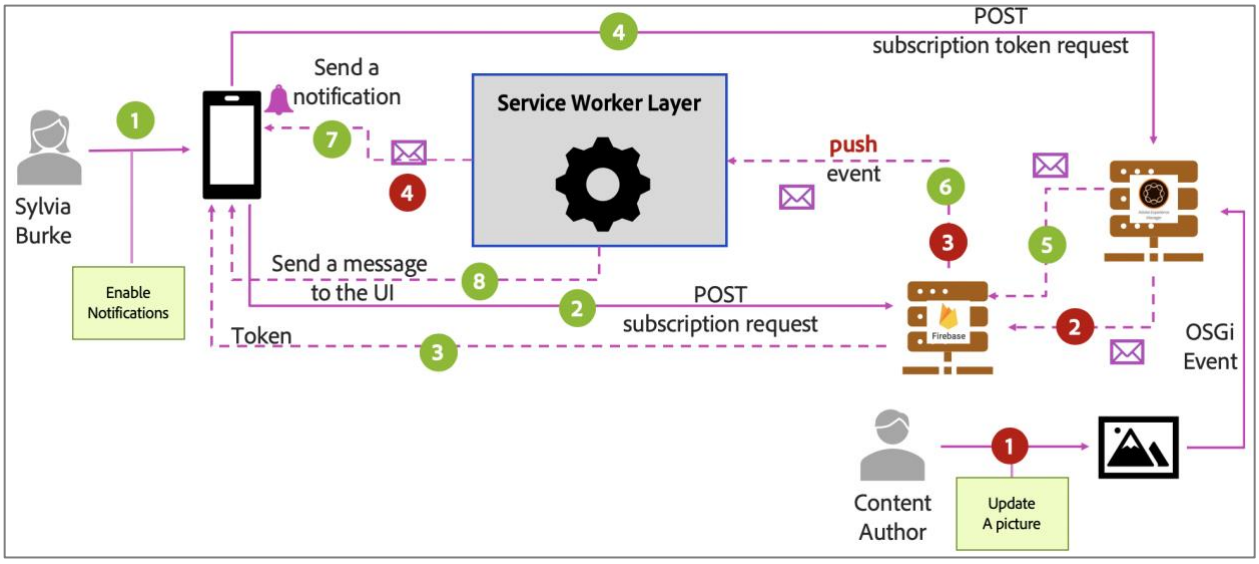
You will see a PWA powerded by AEM displayed.

Well done, you have created your first PWA with AEM

3.7 - Go further

- [0] https://developer.mozilla.org/en-US/docs/Web/API/Cache
- [1] https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker

Lesson 4 – Push notifications



Objectives

1. Understand how to send push notifications using Firebase Cloud Messaging service with a service worker and AEM.

Lesson Context

Push notifications help to engage with application : notifications are delivered to the user as short messages popping up on the user's device.. The message can be triggered locally by the opened application or can be "pushed" from the server to the user even when the app is not running by a service worker. This Lesson will teach you how to deliver a personalised message to an end user based on his profile and AEM modified content.

Technical Context

PWA can be used to send Push Notifications to users by leveraging the Push API and the Notification API. AEM will send data to a service worker, and Notifications will be used by the service workers to show some informations to the user .Service workers listen to "push" events. Those events are conveyed from AEM to the PWA by the Firebase Cloud Messaging APIs.

This lab uses a dedicated OSGi bundle “Firebase Admin SDK OSGi” which has been created to leverage the firebase admin sdk (Java). (https://github.com/firebase/firebase-admin-java).
When a user allow his device to receive notifications, a firebase token is created and then sent to AEM. AEM will check the token integrity and create subscriptions based on the user’s profile.

4.1 – Firebase subscription

In CRXDE, Copy the “web push” code snippet available (**/apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-01.txt**) at http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-01.txt into the exercise 04 client library (**/apps/aem-pwa-blog/clientlibs/clientlib-exercises/scripts/ex04_push-notifications.js**) available at http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/clientlibs/clientlib-exercises/scripts/ex04_push-notifications.js at the location shown below :

```
if($pushButton){
    $pushButton.click( function() {

        /**
         ================================================================================================

         Exercise 04 :  Push notifications
         -----------
         Copy the code from this file : /apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-01.txt
         below this commented block  :

         ================================================================================================
         **/


    });
}
```

4.2 – Listen to push notifications

In CRXDE, Copy the “service worker push” code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-02.txt**) available at http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-02.txt into the service worker file (**/etc/clientlibs/aem-pwa-blog/sw.js**) available at http://127.0.0.1:4503/crx/de/index.jsp#/etc/clientlibs/aem-pwa-blog/sw.js . The code should be added into the callback function of the **“push”** event listener :

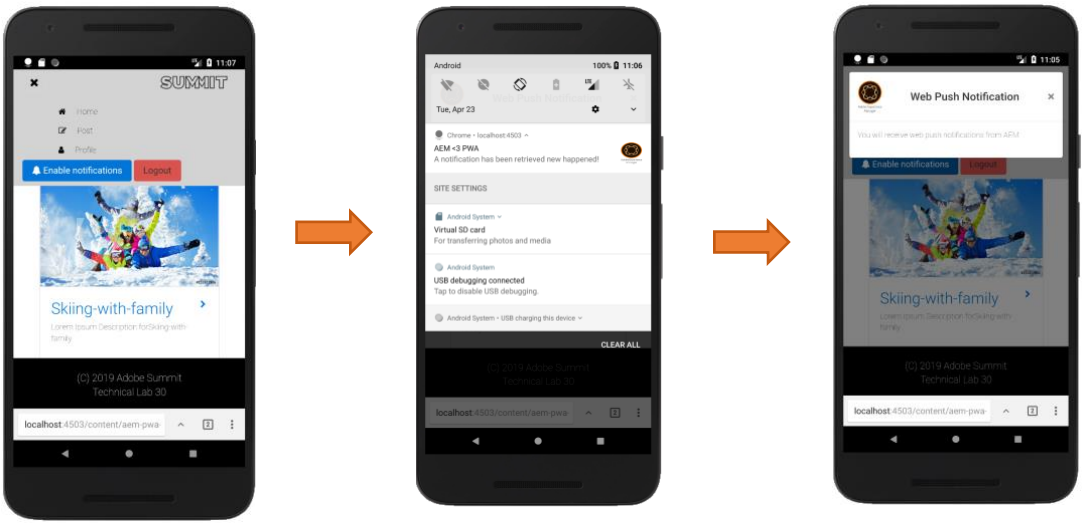
```
self.addEventListener('push', function(event) {
    console.log('[TL30-PWA][push] Push Received.');
    /**
     ======================================================

     Exercise 04 : Push notifications
     -----------
     Copy the code from this file : /apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-02.txt
     below this commented block  :

     ======================================================
     **/


});
```

5 Update the cache version number, by increasing the VERSION variable value in the service worker file (/etc/clientlibs/aem-pwa-blog/sw.js) : this will force the service worker to update the caches in the "activate" state of its lifecycle.

4.3 – Notifications

1. Open your PWA in the android emulator.
2. Navigate to the login page and authenticate yourself with Sylvia Burke's credentials : sburke/sburke
3. Click on the menu and click on **"Enable notifications"**

4.3.1 - Enable notifications :

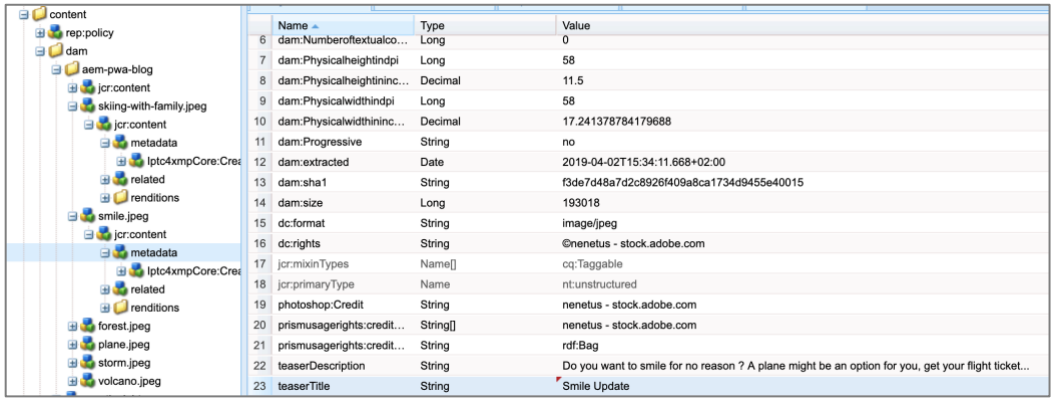


A web push notification has been sent to the PWA from AEM using Firebase Cloud Messaging server and an AEM Event handler.

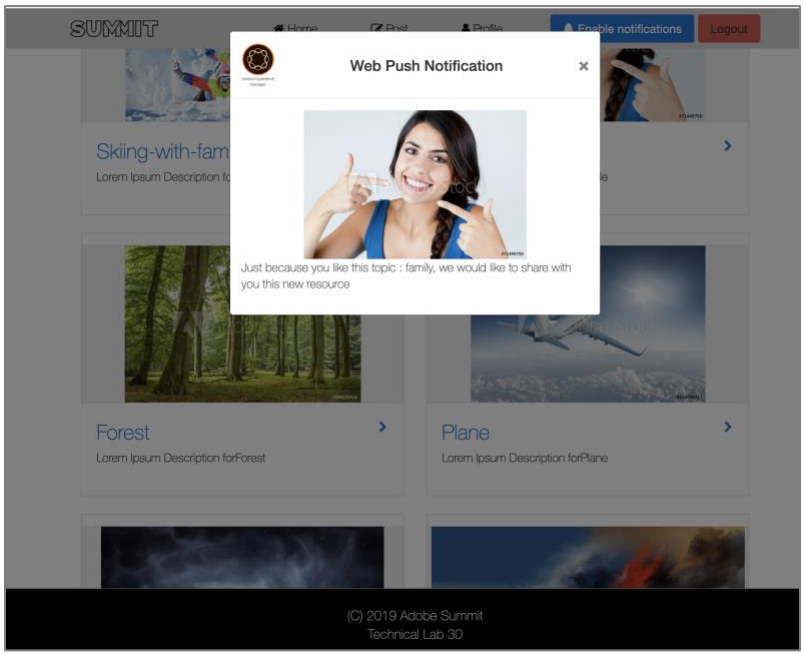
4.3.2 – User's profile based notifications

1. Open the PWA with Chrome browser in the desktop.

2. Authenticate with Sylvia Burke credentials (sburke/sburke) if you are not authenticated yet.
3. Click on the menu and click on **"Enable notifications"**
4. Navigate to CRXDE and update the smile.jpg's metadata (**/content/dam/aem-pwa-blog/smile.jpeg/jcr:content/metadata**) at [http://127.0.0.1:4503/crx/de/index.jsp - /content/dam/aem-pwa-blog/smile.jpeg/jcr:content/metadata](http://127.0.0.1:4503/crx/de/index.jsp#/content/dam/aem-pwa-blog/smile.jpeg/jcr:content/metadata) , you can update the **teaserTitle** property



5. A web push notification will be sent by AEM to the PWA



4.4 - Lab completion status

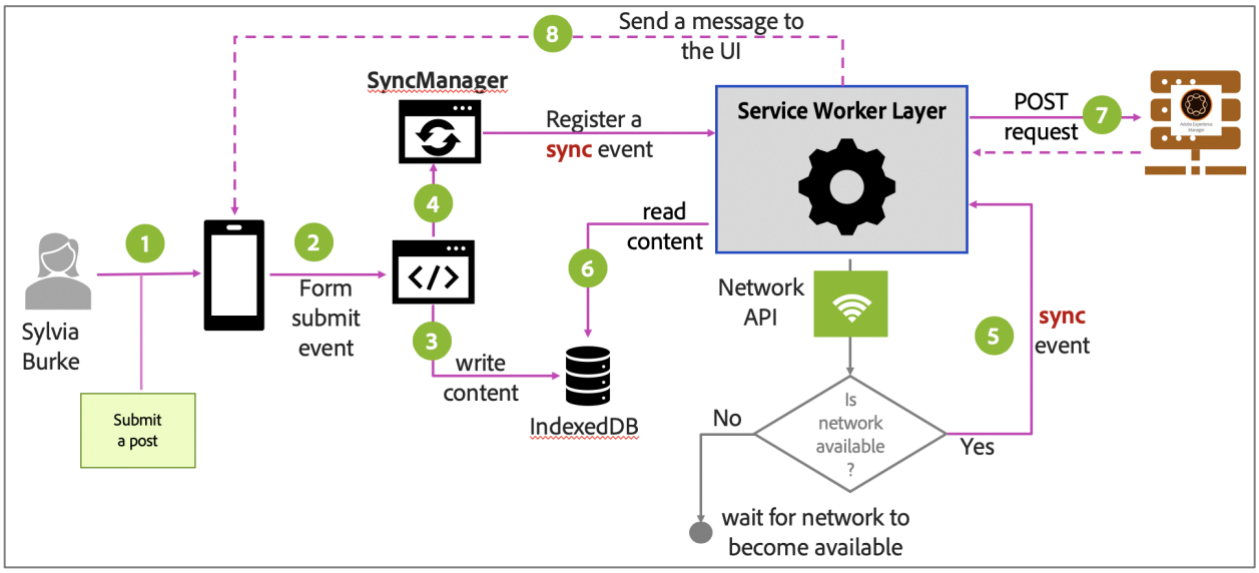
The application uses the Web Push API from a service worker to send push notifications to mobile and desktop. These notifications can be triggered based on content and user's profile to boost their engagement and increase their retention. Our AEM website is now a fully W3C

compliant PWA. It's an "App-like" application because it is installable on mobile and can trigger push notifications as native apps do.

4.5 - Go further

- [0] https://developer.mozilla.org/fr/docs/Web/API/Push_API
- [1] https://developer.mozilla.org/fr/docs/Web/API/Push_API/Best_Practices

Lesson 5 – Background synchronization



Objectives

1. Synchronize data from the PWA with AEM.
2. Understand how to use background synchronization with browser IndexDB database and a service worker.

Lesson Context

The web browser Sync Manager helps to provide a consistent offline user experience and allow your application to perform well on slow connections. Users might interact with AEM by posting User Generated Content(UGC). A example of UGC action is to submit a contact form or subscribe to a newsletter. These actions are performed using HTTP requests if these ones fail because of the network availability then user's data is lost. Using a sync manager allows the service worker to execute the HTTP request based on the content stored in a database of requests.

5.1 – Save data into IndexDB before syncing with AEM

In CRXDE, Copy the "exercise 05" code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-05/ex05-code-to-paste-01.txt**) from http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/code-snippets/exercise-05/ex05-code-to-paste-01.txt to the exercise 05 clientlib (**/apps/aem-pwa-blog/clientlibs/clientlib-exercises/scripts/ex05_background-sync.js**) at http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/clientlibs/clientlib-exercises/scripts/ex05_background-sync.js at the location shown below :

```
if ('serviceWorker' in navigator && 'SyncManager' in window) {
    /**
     ========================================================================================================

     Exercise 04 :  Background Sync
     -----------
     Copy the code from this file : /apps/aem-pwa-blog/code-snippets/exercise-05/ex05-code-to-paste-01.txt
     below this commented block  :

     ========================================================================================================
     **/


} else {
    AdobeSummit.sendData();
}
```

5.2 – Sync data with AEM

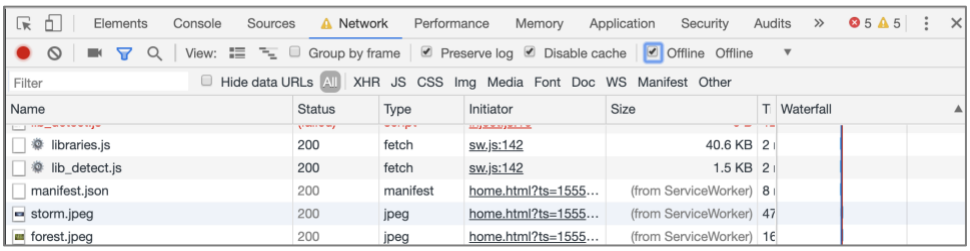
1. In CRXDE, Copy the "sync data" code snippet (**/apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-02.txt**) available at http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/code-snippets/exercise-04/ex04-code-to-paste-02.txt into the service worker file (**/etc/clientlibs/aem-pwa-blog/sw.js**) http://127.0.0.1:4503/crx/de/index.jsp#/etc/clientlibs/aem-pwa-blog/sw.js in the callback function of **"sync"** event :

```
self.addEventListener('sync', function(event) {
    console.log('[Service Worker] Background syncing', event);


});
```

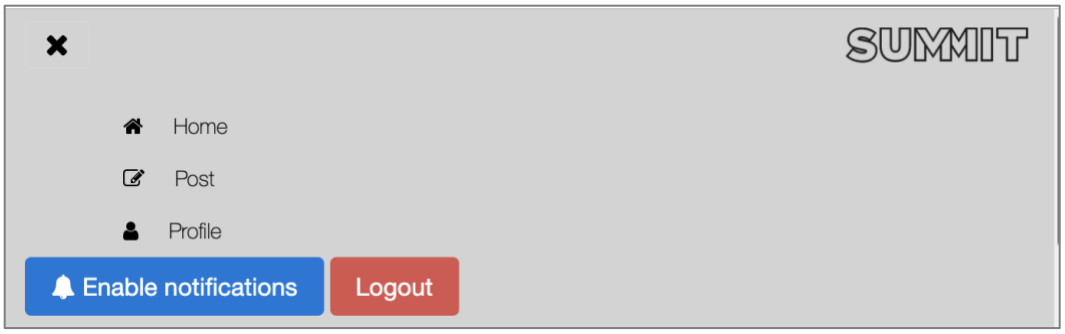
2. **Update the cache version number, by increasing the VERSION variable value in the service worker file (/etc/clientlibs/aem-pwa-blog/sw.js) : this will force the service worker to update the caches in the "activate" state of its lifecycle.**

5.3 – Test background sync

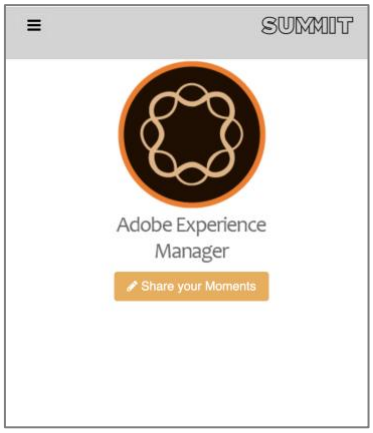
- Disable network connectivity
    - Open the home page (**/content/aem-pwa-blog/home.html**) of TL30 blog : http://localhost:4503/content/aem-pwa-blog/home.html
        c. Open the Chrome Developer Console by navigating like this :
            i. Click on the **"Network"** tab
            ii. Check the offline checkbox

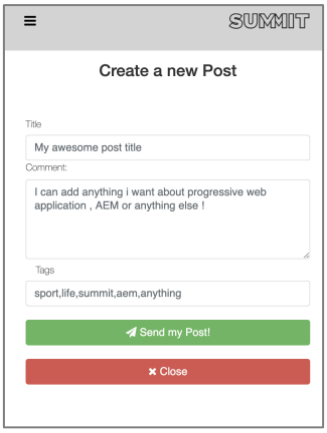


- Create a new post : Click on menu button > Click on Post

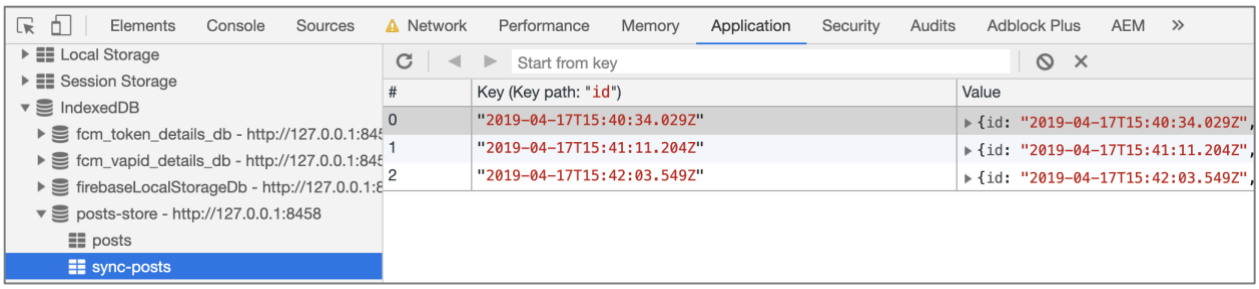


- Click on share **"your moments"** you be redirected to a new section for creating posts





- Fill the form in, and click on **"Send my Post"**
- Open the Chrome Developer Console by navigating like this :
    - Click on the **"Application"** tab
    - Click on **"IndexedDB"**



- Click on the **"Network"** tab
- Uncheck the offline checkbox

All pending requests will be executed by the service worker.

5.4 – Enable device camera (Bonus)

In CRXDE, open this file http://127.0.0.1:4503/crx/de/index.jsp#/apps/aem-pwa-blog/components/content/post/post.html and uncomment the code to include the camera component

```
<!--
<sly data-sly-resource="${'camera' @ resourceType='aem-pwa-blog/components/content/camera'}"></sly>
-->
```

Update the cache version number, by increasing the VERSION variable value in the service worker file (/etc/clientlibs/aem-pwa-blog/sw.js) : this will force the service worker to update the caches in the "activate" state of its lifecycle.

Create a new post by following the steps from 5.3 – Test background sync

5.5 – Lab completion status

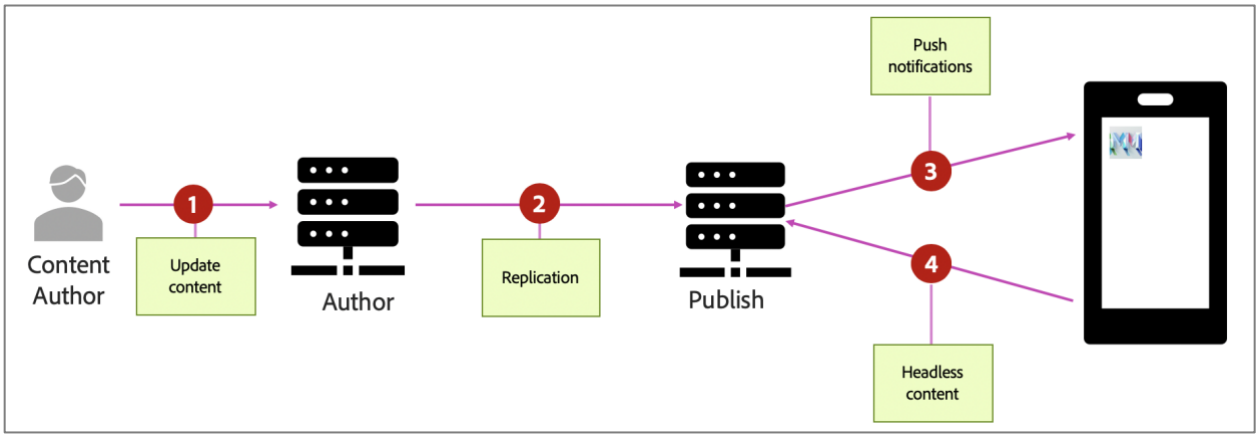
The fourth building block of a PWA is ready to use. The application reacts to background synchronization, using the browser sync manager. The service worker knows when the mobile device goes offline and when the network connectivity is back. The **sync** event is triggered when network becomes available. The user can work offline without having any concerns about data consistency, because these data will be sync up automatically.

5.6 - Go further

[0] https://developer.mozilla.org/fr/docs/Web/API/SyncEvent
[1] https://developer.mozilla.org/fr/docs/Web/API/SyncManager
[2] https://developers.google.com/web/updates/2015/12/background-sync

6 - Key learnings

6.1 - PWA as a channel for AEM



A PWA could be considered as a new channel for sharing content with end users. This is compliant with AEM headless capabilities. AEM will still be used for managing content in the authoring instance. However, the same way this content will be pushed to a publish instance loaded in a web browser, it will also be available in an “App-like” application installed onto a user’s device : a PWA.

6.2 Why having PWA capabilities is a good fit for AEM ?

The PWA will be able to consume content in headless mode. This content could be :

- Pages
- Assets
- Experience Fragment
- Content Fragment via the content services

Creating a dedicated content for the PWA (XF variation, selectors with components) is possible if this content is meant to target a specific population of users via push notifications.
The content will be modified in the users instance and sent to the publish instance, that will trigger some actions like sending a push notifications as we saw in lesson 5.

6.3 Why should you watch out for PWA enhancements ?

PWA uses a set of APIs to access native device features. There are still many features missing like using the filesystem, Bluetooth and other capabilities, but browser keep improving and PWA has a bright future coming. The sooner we embrace it, the better it will be when most of the native devices APIs will/might be ready.