**PWA – PROGRESSIVE WEB APPLICATION**

**KEY POINTS**

1. PWA basic requirements.
2. Guide to Move your existing web application to PWA.
3. Steps to develop progressive web application using angular.

**PWA Basic Requirements**

1. Go Https to convert your website to progressive web application.

**Guide to move your existing web application to PWA**

The first requirement by simply hard coding the content and the second by adding a ViewPort meta tag in your index.html <head> section.

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**Necessary Steps to follow to convert into Progressive web application.**

1. Create an App Manifest.
2. Add it to your base HTML template.
3. Create the service Worker.
4. Serve the service worker on the root of the scope you used in the manifest
5. Add a <script> block to your base HTML template to load the service worker.
6. Deploy your progressive web application
7. Use Your progressive web app.

* **Create an App Manifest**  src/manifest.json

An App Manifest is a json file containing a following information:

* The canonical name of the website.
* A short version of the name (for icons).
* The theme color of the website for OS integration.
* The background color of the website for OS integration.
* The URL scope that the progressive web app is limited to.
* The Start URL that new instances of the progressive web app will implicitly load.
* A human readable description.
* Orientation restrictions.
* Any icons for your website to be used on the home screen.

This information will be used as a OS level metadata for your progressive web app when it is installed.

Example of Manifest file

**{**

**“name”: “EPPM Mobile App”,**

**“short\_Name”: “EPPM”,**

**“theme\_color”: “#ffffff”,**

**“background\_color”: “fa99ca”,**

**“display”: standalone,**

**“scope”: “/”,**

**“start\_url”: “**[**https://eppm.com/**](https://eppm.com/)**”,**

**“description”: “This is Project Management application”,**

**“orientation”: “any”,**

**“icons”: [**

**{**

**“src”: “path for the icon according to your project folder”,**

**“sizes”: “1024 X 1024”,**

**“type”: “image/png”**

**}**

**]**

**}**

* **Add it to your base HTML template**

Assuming you are hosting this manifest at path **static/manifest.json,** Simply add it to the **<head>** section.

Ex: <link rel=”manifest” href=”/static/manifest.json”>

you should also declare the theme color to match the one set in your manifest by adding a meta tag inside the <head>

**<head>**

**<meta name=”theme-color” content=”white”>**

**</head>**

* **Create the Service Worker**

A Service Worker is a JavaScript file, placed in your app’s root, that plays the role of a “middleman” between the browser and host. The one which, once installed in the supported browsers, intercepts and responds to network request in different ways.

Create a file named sw.js in your root folder and enter the content of the script below.

The reason it’s saved in the app root is to give it access to all of the app’s files. This is because service workers only have permission to access files in the same directory and sub- directories.

var cacheName = 'hello-pwa';

var filesToCache = [

'/',

'/index.html',

'/css/style.css',

'/js/main.js'

];

**Start the service worker and cache all of the app’s content.**

self.addEventListener('install',function(e)

{ e.waitUntil(

cache.open(cacheName).then(function(cache) {

return cache.addAll(filesTocache);

})

);

**Server cached content when offline**

self.addEventListener(‘fetch’, function(e) {

e.respondWith(

catches.match(e.request).then( function(response) {

return response || fetch(e.request);

})

);

});

**Add a <script> block to your base HTML template to load the service worker.**

Now that the service worker script is created we need to register it with our app. Create a file named main.js in the js folder and enter the following code:

window.onload = () => {

'use strict';

if ('serviceWorker' in navigator) {

navigator.serviceWorker

.register('./sw.js');

}

}

This code simply loads up the service worker script and get it started.

Add the code to your app by including the script just before the closing </body>

tag in index.html.

….

<script src=”js/main.js”></script>

</body>

After completion of the all the step deploy application build on https server and test it .

For Testing of manifest file and service worker :

1. open your application in browser.
2. Go to inspect using F12 key or ctrl+shift+I
3. Click on Application menu
4. After clicked on application you will see multiple option on left side bar
5. Then click on manifest after that you can check all the details about your application that you have mentioned in manifest file, like app icon.
6. You can also check in Service Worker section, all the details about service worker. Like it is registered or not .
7. You can also check the number of cache that you have mentioned in service worker file.

**Steps to develop progressive web application using angular.**

1. Install Angular Cli:

**npm install -g @angular/cli**

1. Create New Project:

**ng new pwaTestDemo**

1. Go to this project folder

**cd pwaTestDemo**

1. Add angular pwa to your projects

**ng add @angular/pwa**

**What happened when we add angular pwa**

* **package.json**

1. “@angular/pwa”
2. “@angular/service-worker”

In **package.json** file dependencies have been added to necessary bundles.

* **angular.json**
  1. “serviceWorker”: true

In **angular.json** file a notification including service workers has been added to section related to configuration.

* **ngsw-config.json**

ngsw is a file where we can configure behaviors related to caching particular resources in an application.

In **assetGroup** section we have two objects where in the first one we configure caching the index.html,JS and CSS files

The second one is related to caching local resources from **assets folder**.

Entries differ in setting options installMode and updateMode. Official Angualr documentation very well explains the difference between specific modes.

For caching API we use section **dataGroups.**

* **app.module.ts**

In app.module.ts, ServiceWorkerModule has been added. This module gives us opportunity to inject SwUpdate (it allows to manage the update of our application, for example inform the user about a new version) or SwPush services (related to push notifications) and register Angular Service Worker in the browser. Using enabled parameter we may navigate when SW are supposed to be registered.

1. **How to launch it:**

**npm install -g http- server**

**ng build –prod**

After creation of this production build, You will found a **dist** folder inside your

**project name folder or directory**. Now upload this folder to your server and serve your application.

**FEATURES :**

* **New Build update:**

If you are creating a progressive web application using angular and in that you need to install the package **“@angular/pwa”** it content angular pwa as well as angular service Worker.

Angular service Worker have the many services for handling features and **SwUpdate** is one of them, using this service we can give the permission access popup to user to access updated version of the application.

Steps to achieve this feature:

1. Need to import SwUpdate from @angular/service-worker
2. Add instant variable for this import in constructor.

private swUpdate: SwUpdate;

1. if(this.swUpdate.isEnabled) {

this.swUpdate.available.subscribe( () => {

if(confirm("New version available. Load New Version?")){

window.location.reload();

}

});

}

* **Push Notification**

To achieve this feature, you can use service Worker **SwPush** service.