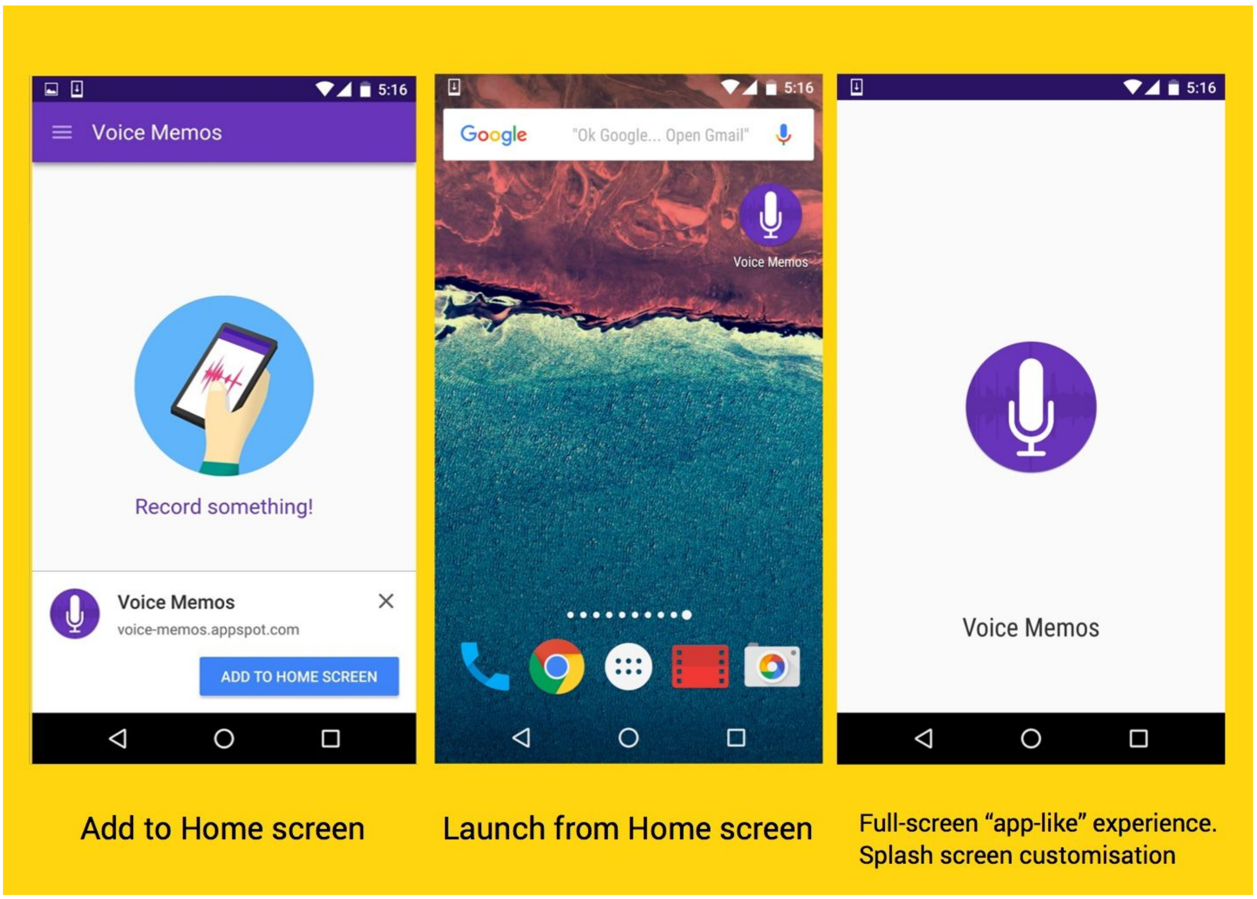
Progressive Web Apps

A Progressive Web App (PWA) is a web app that uses modern web capabilities to deliver an app-like experience to users. These apps meet certain requirements, are deployed to servers, accessible through URLs, and indexed by the search engines.



Features

- *Install on Homescreen*
- *Offline Accessibility*
- *Push Notifications*
- *Access Device Camera and Location*
- *Background Sync*

To be considered a Progressive Web App, your app must be:

➔ ***Progressive*** *- Work for every user, regardless of browser choice, because they are built with progressive enhancement as a core tenet.*
➔ ***Responsive*** *- Fit 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*** *- Use the app-shell model to provide app-style navigation and interactions.*

- ➔ ***Fresh** - Always up-to-date thanks to the service worker update process.*
- ➔ ***Safe** - Served via HTTPS to prevent snooping and ensure content has not been tampered with.*
- ➔ ***Discoverable** - Are identifiable as "applications" thanks to W3C manifests and service worker registration scope allowing search engines to find them.*
- ➔ ***Re-engageable** - Make re-engagement easy through features like push notifications.*
- ➔ ***Installable** - Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store.*
- ➔ ***Linkable** - Easily share via URL and not require complex installation.*
- ➔ ***Offline Support** - Apps should be able to work offline. Whether that be displaying a proper "offline" message or caching app data for display purpose.*

Four minimum requirements for an application to be a PWA:

Web App Manifest
This is just a json file that gives meta information about the web app. It has information like the icon of the app (which a user sees after installing it in their app drawer), background color of the app, name of the app, short name, and so on. If you link to the manifest file in your index.html, browsers will detect that and load the resources for you.

Service Worker
Service Workers are event-driven workers that run in the background of an application and act as a proxy between the network and application. They are able to intercept network requests and cache information for us in the background. This can be used to load data for offline use. They are a javascript script that listens to events like fetch and install, and they perform tasks.

Icon
This is used to provide an app icon when a user installs the PWA in their application drawer. A jpeg image will just be fine. The manifest tool I highlighted above helps in generating icons for multiple formats, and I found it very useful.

Served over HTTPS
In order to be a PWA, the web application must be served over a secure network. With services like Cloudflare and LetsEncrypt, it is really easy to get an SSL certificate. Being a secure site is not only a best practice, it also establishes your web application as a trusted site for users demonstrating trust and reliability, and avoiding middleman attacks.

For an app to be progressive, it needs to have the following requirements:

- *a manifest file — manifest.json*
- *service worker with at least a fetch event — serviceworker.js*
- *icon — icon.jpeg*
- *served over HTTPS — https://www.example.com*

Converting django app into a progressive web app

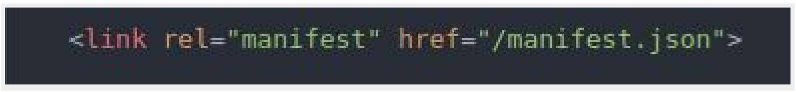
Manifest file

Generate web app manifest with icons. A useful online tool to create manifest file is given below

[*https://app-manifest.firebaseapp.com/*](https://app-manifest.firebaseapp.com/)

Move manifest.json file into /templates folder and place icons at static/img/icons

Link the manifest file in base.html



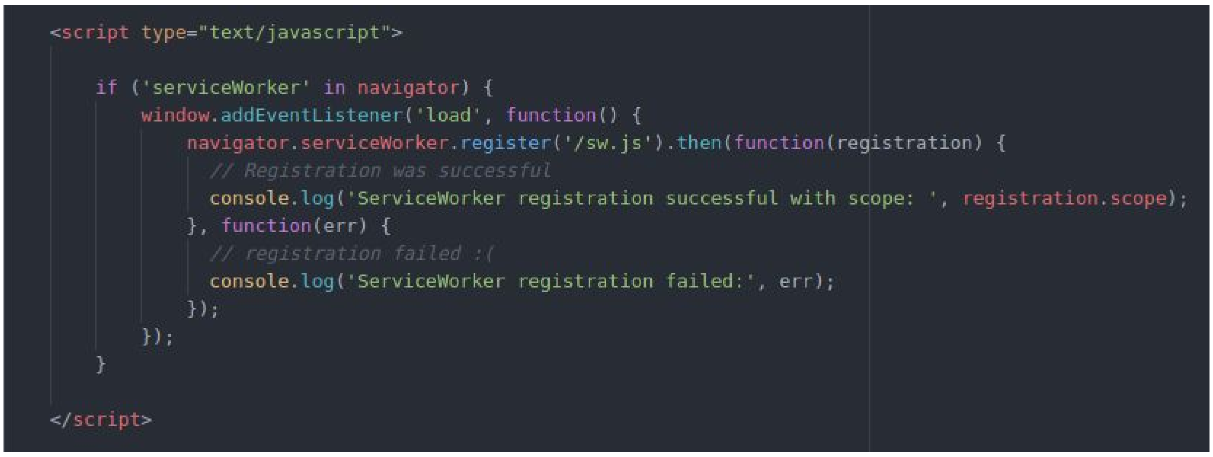
```
<link rel="manifest" href="/manifest.json">
```

Service Worker

First, we are going to register the service worker on install. Then we will cache some static assets such as style.css and script.js. Next, we need to provide offline capability using fetch .

To Register service worker, add following snippet in base.html

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    navigator.serviceWorker.register('/sw.js').then(function(registration) {
      // Registration was successful
      console.log('ServiceWorker registration successful with scope: ',
registration.scope);
    }, function(err) {
      // registration failed :(
      console.log('ServiceWorker registration failed:', err);
    });
  });
}
```



```
<script type="text/javascript">

    if ('serviceWorker' in navigator) {
        window.addEventListener('load', function() {
            navigator.serviceWorker.register('/sw.js').then(function(registration) {
              // Registration was successful
              console.log('ServiceWorker registration successful with scope: ', registration.scope);
            }, function(err) {
              // registration failed :(
              console.log('ServiceWorker registration failed:', err);
            });
        });
    }

</script>
```

Next,

Move sw.js file into /templates folder

A sample manifest.json file and sw.js file is given below

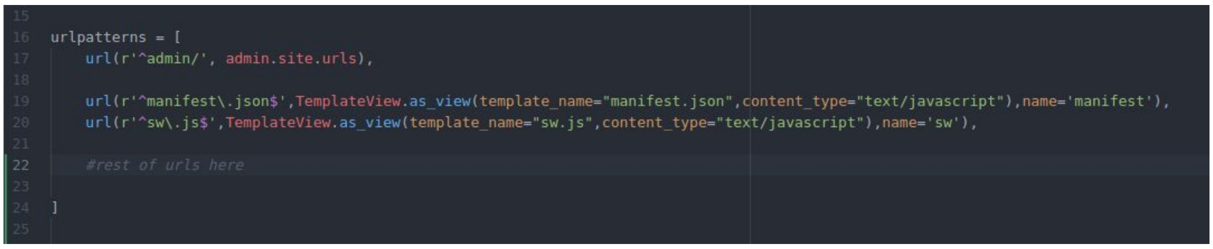
Manifest.json ***sw.js***

Now import and define urls to both files in your project root url

from django.views.generic import TemplateView

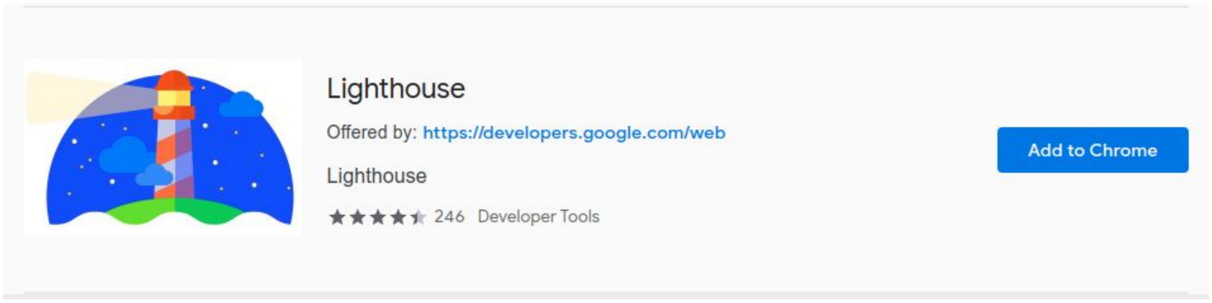
url(r'^manifest\.json$',TemplateView.as_view(template_name="manifest.json",content_type="text/javascript"),name='manifest'),

url(r'^sw\.js$',TemplateView.as_view(template_name="sw.js",content_type="text/javascript"),name='sw'),



```

urlpatterns = [
    url(r'^admin/', admin.site.urls),

    url(r'^manifest\.json$',TemplateView.as_view(template_name="manifest.json",content_type="text/javascript"),name='manifest'),
    url(r'^sw\.js$',TemplateView.as_view(template_name="sw.js",content_type="text/javascript"),name='sw'),

    #rest of urls here

]

```

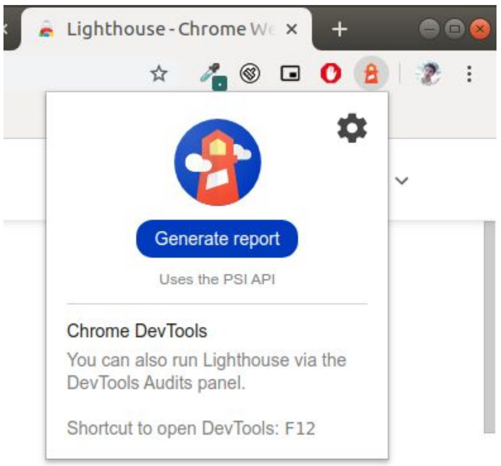
You are all set,
Now you can analyse your app's performance and generate report using google lighthouse chrome extension



Lighthouse
Offered by: https://developers.google.com/web
Lighthouse
★★★★☆ 246 Developer Tools

After adding into chrome,

You can access lighthouse extension from chrome toolbar



Chrome DevTools
You can also run Lighthouse via the DevTools Audits panel.
Shortcut to open DevTools: F12

For Further Reference

Google has published a checklist of items for Progressive Web apps

A comprehensive set of service worker samples
https://github.com/GoogleChrome/samples/tree/gh-pages/service-worker