1. **What is PWA?**

Progressive Web Applications (PWAs) are apps built with web technologies that we probably all know and love, like HTML, CSS, and JavaScript. But they have the feel and functionality of an actual native app.
A Native App is a software application built in a specific programming language for a specific device platform, either IOS or Android.
PWAs are built with the capabilities like push notifications and the ability to work offline. They are also built on and enhanced with modern APIs which makes it easy to deliver improved capabilities along with reliability and the ability to install them on any device.
PWAs takes advantage of the huge web ecosystem that is inclusive of the plugins, and community and the relative ease of deploying and keeping a website contrary to a native application which is pretty difficult to develop. This means you can build a PWA quickly and easily.

2. **Importance of PWA.**

A. They are responsive and work with many different screen sizes.
B. They function just like normal Native Apps.
C. The updates are independent, you don't need to visit the play store for an update.
D. They're built with common web technologies.
E. They're fast and lightweight.
F. They work offline unlike other sites.
G. They are discoverable via search engines.
H. They are easily installable.
I. Low maintenance cost.

3. **Technical components of PWA: Web App Manifest, Service Worker ,Service worker lifecycle.**

<u>Manifest file</u>
This is a JSON file that is created with a generator. This file contains the information that tells how your PWA should appear and function. It allows you to determine the name, description, icon, colors and other features of your PWA.

<u>Service workers</u>

This is one of the key technologies behind PWAs. They help support your app work offline, and they perform advanced caching and run background tasks. Service workers can complete tasks even when your PWA is not running.
Some other functions associated with Service Worker include:

- Sending push notification
- Badging icons
- Running background fetch tasks etc.

Service Worker lifecycle-

The service worker lifecycle has 3 steps; Registration, Installation, and Activation.

#1 Registration
To install a service worker, you have to register it on your background page. It informs the browser regarding the location of the service worker in a JavaScript file.
You can use the following code to check if the Service Worker API is available or not. If it is there, then service worker /sw.js is registered when the page is loaded.

```
// app.js
if (‘serviceWorker’ in navigator) {
   navigator.serviceWorker.register(‘/sw.js’)
   .then(function (registration) {
      console.log(‘Service worker registered!’);
   })
   .catch(function (err) {
      console.log(‘Registration failed!’);
   })
}
```

#2 Installation (Install Event)
After the successful completion of the registration process, the service worker script is downloaded and the installation event is initiated.
The service worker will only be installed if it hasn’t been registered before or if its script gets altered even by 1 byte.
You can define a callback for the install event and decide which files to cache.

Following is the example of installing Cache API event:-

```
// sw.js
const assetsToCache = [
   ‘/index.html’,
   ‘/about.html’,
   ‘/css/app.css’,
   ‘/js/app.js’,
]
self.addEventListener(‘install’, function (event) {
   event.waitUntil(
      caches.open(‘staticAssetsCache’).then(function (cache) {
           return cache.addAll(assetsToCache);
      })
     );
});
```

#3 Activation
After successful installation, the service worker enters an installed state. It is not active yet but takes control of the page from the current service worker.
A service worker doesn’t get active immediately after installation. It can be activated if no other service worker is currently active, the user refreshes the page, or if the self.skipWaiting () is called the script of the installed service worker.
This can be further understood with the following example:-

```
/ sw.js
self.addEventListener(‘install’, function (event) {
   self.skipWaiting();
   event.waitUntil(
        // static assets caching
     );
});
```

4. **Framework tools: Webpack :- webpack-pwa-framework, Lighthouse : - lighthouse pwa-framework.**

**webpack–pwa-framework**

As I mentioned before, webpack-pwa-framework is not a single thing, but a term that can refer to different things related to webpack and PWAs.

- Webpack is a tool for building PWAs that are web apps that work offline and have native-like features1.
- Webpack-pwa is an example of a super simple PWA with webpack that uses no framework and only simple JavaScript and DOM2.
- Webpack-pwa-manifest is a plugin for webpack that generates a manifest file for PWAs with icons, splash screens and more3.

You can also use other frameworks and tools to build PWAs, such as Ionic, Angular or React3.

**Lighthouse-pwa-framework**

Lighthouse is not a PWA framework, but a tool for improving the quality of web pages and PWAs. It is an open-source and automated tool designed by Google that audits web pages and PWAs for performance, accessibility, SEO and other factors123. You can run Lighthouse in Chrome DevTools, from the command line or as a Node module3.