Progressive Web App for Educational System

Aditya Bhattad, Amit Mhaske, Priyanka Khamkar and Radhika More

Abstract

With the invent of Smartphone especially after the launch of Android OS which is free the popularity of using smartphone is very huge. Close to 60% user across the globe are using smartphones and the transaction are increasing day by day. Most of the users use native mobile application to browse the contents of the particular industry. Another way is to browse the contents is through web browser. But both have the limitation. The first one which is the native app, user need to download the app first and then they have to use it as per their requirement. This has major two disadvantages, one is it takes some space on local smartphone device and to operate it smoothly the network connection has to be strong. It is a very slow process to access this native app where only 2G or lesser bandwidth 3G network is available. The second approach which is through the web browsing has disadvantages as the user experience is not that great compared to native app. To overcome above limitations, Google had provided a solution through a term called as Progressive Web App (PWA) which gives you a rich experience just like native apps. You don't require installing this PWA and can be operated through a splash screen. Our Paper majorly focuses on the building a progressive app for an educational system.

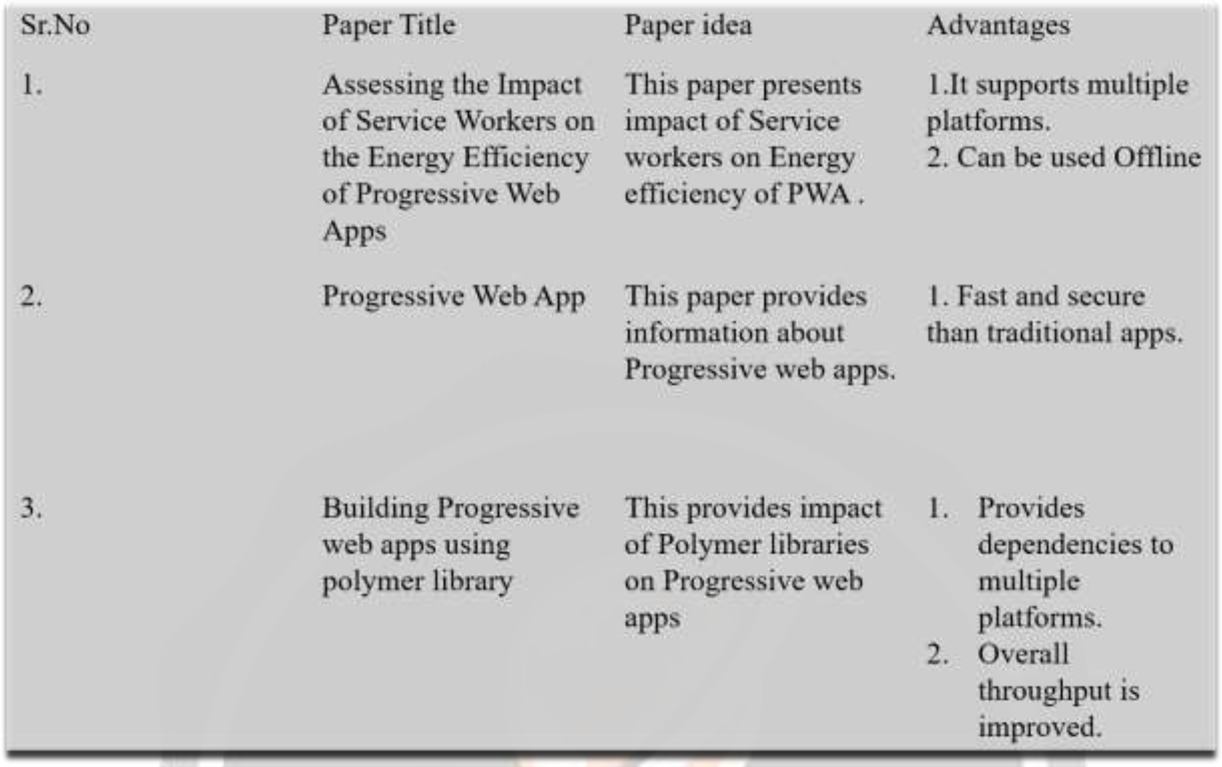
***Keywords**: Progressive web app, Splash Screen, service work, network, native app experience, web library.*

Introduction: As we had discussed in beginning, the world is moving towards smartphone applications and total activity on mobile phone through web browsing or

- 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 navigations and interactions.
- Fresh - Always up-to-date thanks to the service worker update process.
- Safe - Served via TLS to prevent snooping and ensure content hasn't 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.

In this paper we are putting focus on creating progressive web app for an Educational System.

Literature Survey:



Sr.No	Paper Title	Paper idea	Advantages
1.	Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps	This paper presents impact of Service workers on Energy efficiency of PWA .	1.It supports multiple platforms. 2. Can be used Offline
2.	Progressive Web App	This paper provides information about Progressive web apps.	1. Fast and secure than traditional apps.
3.	Building Progressive web apps using polymer library	This provides impact of Polymer libraries on Progressive web apps	1. Provides dependencies to multiple platforms. 2. Overall throughput is improved.

Proposed System Architecture:

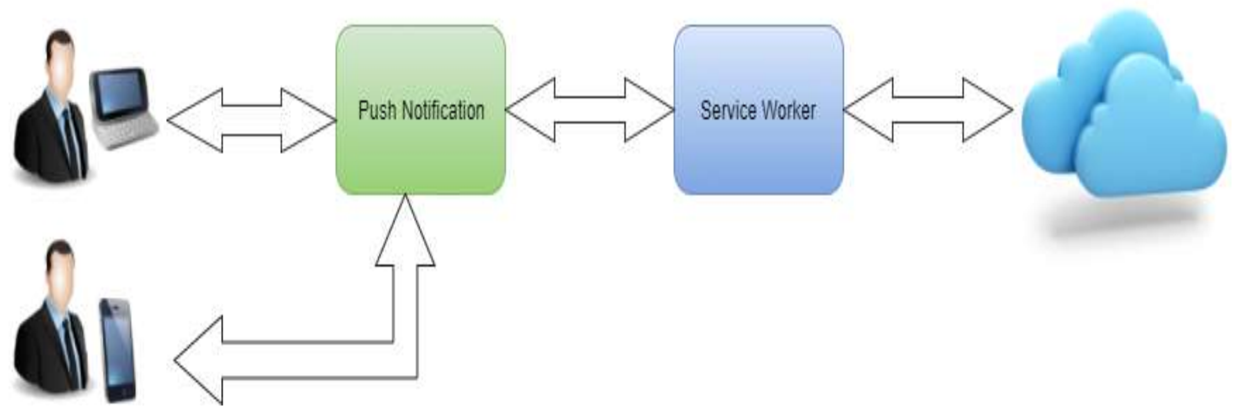


Figure 1: System Architecture of PWA for an Educational System

There are various ways by which this all in approach of progressive web app model goes, but one of the most common ways is using Application Shell. This is not a hard requirement, but does come with several benefits. The application shell architecture uses the user interface so that it can work offline and generates its own contents by using technology called JavaScript. When user goes on multiple repeat visits to same app then it gives u the meaningful pixel positions so that screen can be loaded fastly without the network. This is how we can increase the performance gains.

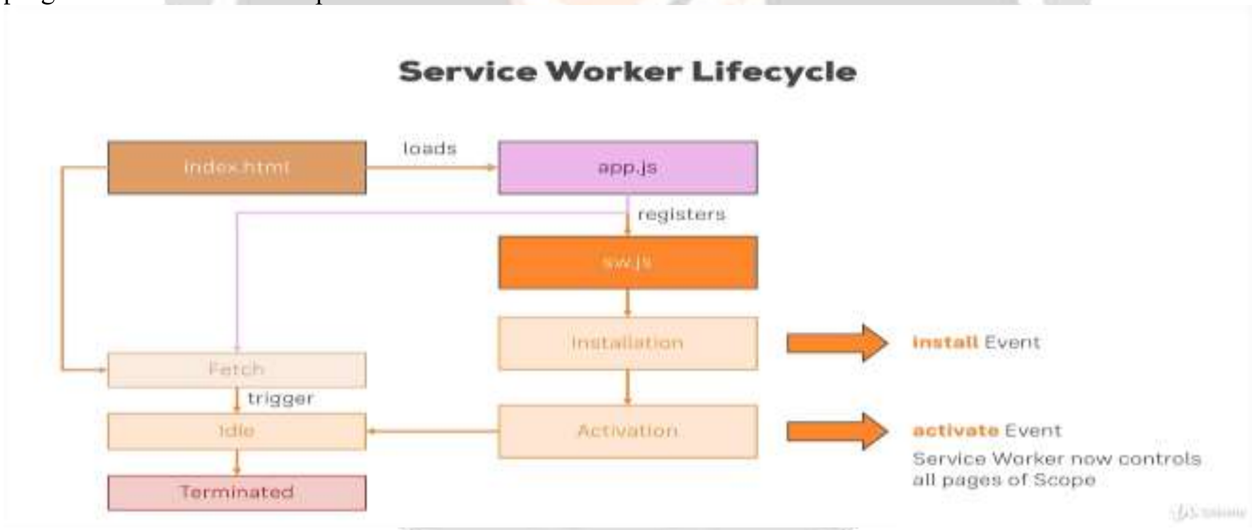
Some of features we used in this project are as follows:

Service Worker:

Service workers are the main part of PWA which runs in background separately from the web pages. All the response to events, network request made from server and client is managed by service workers. The lifetime of Service workers are generally kept for a short time. It wakes up when it gets an event and runs only as long as it needs to process it.

The API which we are going to use in Service Workers is generally limited when we compare with the full functional JavaScript. This is standard for workers on the web. The DOM Stucture of a web page is not accessible by a Service worker can't access the DOM but can access things like the network request, fetch API and Cache API, The Indexed DB API and postMessage() are also available to use for data persistence and messaging between the service worker and pages it controls. Push events sent from your server can invoke the Notification

API to increase user engagement. A service worker can intercept network requests made from a page (which triggers a fetch event on the service worker) and return a response retrieved from the network, or retrieved from a local cache, or even constructed programmatically. Effectively, it's a programmable proxy in the browser. The neat part is that, regardless of where the response comes from, it looks to the web page as though there were no service worker involvement. Service Workers are a way to increase Web app performance by helping to cache and deliver content and background functionality (like push notifications). Service workers can make sites work offline or help speed up the content by, "intercepting network requests to deliver programmatic or cached responses."



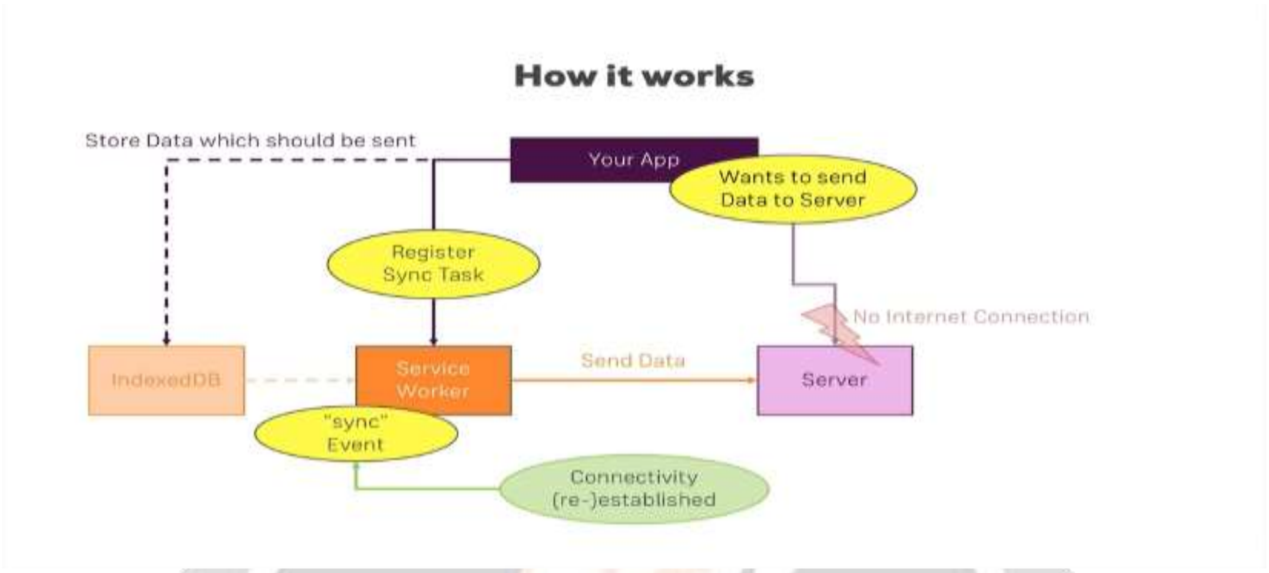
Offline Access:

Service workers can use the Cache interface to cache an application's assets. A service worker script can implement a number of caching strategies, allowing fine tuning of an app's offline and low-connectivity performance.The Cache interface's storage is controlled programmatically and **is independent** of the browser's HTTP cache. Unlike the browser's HTTP cache, the Cache interface's storage is available offline. The service worker can use this to enable offline support in browsers.

Service workers can also use IndexedDB to store data locally. This enables new features such as capturing user actions while offline and delivering them once connectivity returns.

Background Sync:

Background sync is a new web API that lets you defer actions until the user has stable connectivity. This is useful for ensuring that whatever the user wants to send, is actually sent.

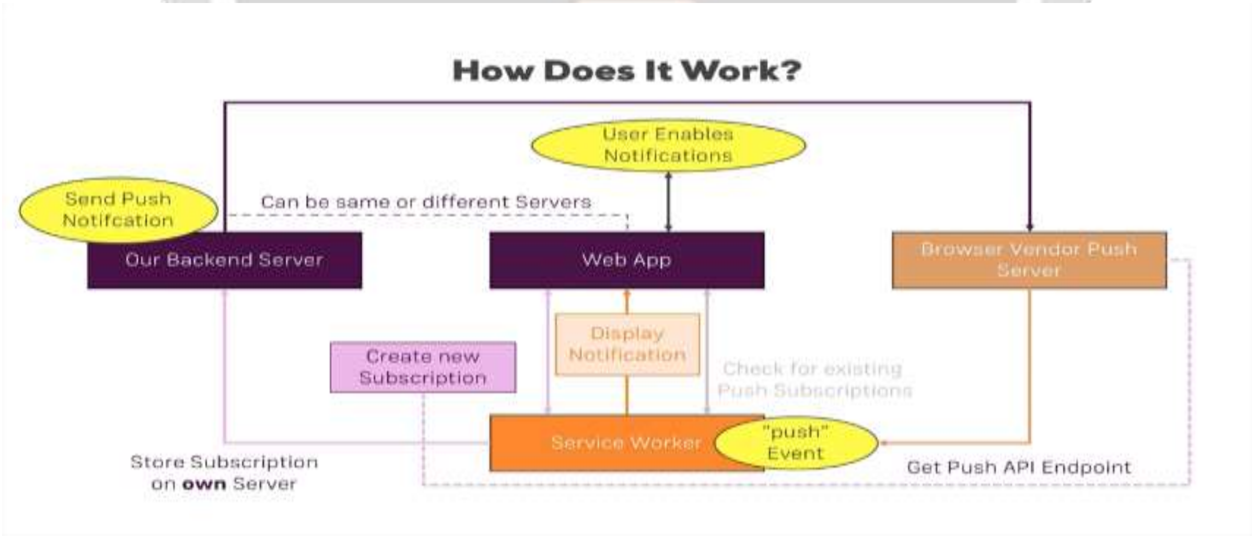


Native application:

This feature provides access to the native applications like camera and geolocation. We use geolocation in this project.

Push Notification:

Push messaging provides a simple and effective way to re-engage with your users and in this code lab you'll learn how to add push notifications to your web app.


How Does It Work?
User Enables Notifications
Send Push Notifcation
Can be same or different Servers
Our Backend Server
Web App
Browser Vendor Push Server
Create new Subscription
Display Notification
Check for existing Push Subscriptions
Service Worker
"push" Event
Store Subscription on own Server
Get Push API Endpoint

Conclusion:

Progressive web app is a mid-way approach for native app and web application. It reduces lot of burden of user about poor network connectivity and rich interface just like native app. They don't require any installation. The app loads quickly, even when the user is on bad networks. It can send relevant push notifications to the user and has an icon on the home screen and loads as top-level, full screen experience.

Progressive web apps are an interesting forward look into the future of mobile apps. It will become a key factor in the world of apps.

References:

[1]http://www.ness-ses.com/progressive-web-apps-the-new-future-of-mobile-apps/
[2]http://blog.cloudfour.com/android-instant-apps-progressive-web-apps-and-the-future-of-the-web
[3]https://arc.applause.com/2015/11/30/application-shell-architecture [4]http://digiday.com/platforms/wtf-progressive-web-apps/