Progressive Web Apps

Garcia Gonzalez Christia Andres
January 2024l

I. Progressive Web Apps

Progressive Web Apps (PWAs) are applications built using web technologies that can be installed and run on all devices from a single codebase.

PWAs provide native-like experiences on supporting devices, adapting to the capabilities of each device and also running in web browsers, similar to websites.

To start building a PWA, refer to "Get started with Progressive Web Apps."

A. PWA Benefits

1) Native-like Experiences: When installed on a device, PWAs function like other apps, including:

- Having their own application icons for the device's home screen or taskbar.
- Launching automatically when an associated file type is opened.
- Running when the user signs in.
- Being submitted to application stores, such as the Microsoft Store.

2) Advanced Capabilities: PWAs have access to advanced capabilities, including:

- Functioning offline.
- Supporting push notifications.
- Performing periodic updates even when not running.
- Accessing hardware features.

3) Web-related Advantages: PWAs can run in web browsers like websites, providing advantages such as:

- Being indexed by search engines.
- Being shared and launched from a standard web link.
- Ensuring user safety through secure HTTPS endpoints and other safeguards.
- Adapting to the user's screen size, orientation, and input method.
- Using advanced web APIs such as WebBluetooth, WebUSB, WebPayment, WebAuthn, or WebAssembly.

4) Lower Development Cost: PWAs have a significantly lower cross-platform development cost compared to compiled apps requiring separate codebases for each platform. A single codebase can be shared between the website, mobile app, and desktop app across operating systems.

5) Cross-device Compatibility: PWAs are built using HTML, CSS, and JavaScript code hosted on web servers. They can be used directly in a web browser or installed on a device from an app store. The user experience scales with device capabilities and ensures compatibility with various browsers, operating systems, and devices.

II. Bridging the Gap Between Web and Native

Web technologies now enable features that were once achievable only with device-specific languages. Examples include:

- Handling files.
- Sharing content with other apps.
- Accessing the clipboard.
- Syncing data and fetching resources in the background.
- Accessing device hardware like Bluetooth and USB.
- Storing content in databases.
- Utilizing hardware-accelerated graphics.
- Using CSS layouts, animations, and filters for advanced designs.
- Running near-compiled performance code with WebAssembly.

III. Technical Requirements for PWAs

Currently compatible with Google Chrome but not with iOS, PWAs must fulfill four technical requisites:

A. Site Visits Requirement

A site must be visited twice within a five-minute interval to qualify.

In Google Chrome, users must have visited the site hosting the PWA twice before the browser shows the message to install it. While not the most reliable form of verification, it serves as a simple way to gauge user interest. This parameter may be replaced by a better one in the future, but for now, Google's developers find it satisfactory.

B. Valid Secure HTTPS Connection

As PWAs have a secure connection, users can feel relatively safe allowing necessary permits. Routing network requests via a service worker script and adding HTTPS to the server help mitigate vulnerabilities, allowing heavy code execution in the background. A secure connection contributes to building trust among users and adds a small SEO benefit.

C. Valid Installed JSON Manifest

Providing a data extract in JSON format allows caching of information with the help of the service worker. The shell app, a model when creating apps, loads CSS rules and delivers an offline version with complete UI capabilities. The service worker caches the JSON extract each time a visitor loads a new page, storing it physically in the shell app.

D. Installed Service Worker

The service worker is responsible for caching files, push notifications, content updates, and data handling. It functions independently of any app or website on the web server, handling network requests on the server while residing as a .js file in the user's device. The service worker manages requests with appropriate responses based on internet connectivity, enabling customized offline pages.

Once these four requirements are correctly implemented, Android users and other compatible devices will be prompted to add the PWA to their home screen. An icon similar to a regular app will open in a browser.

The key concepts of a PWA are defined by:

- Universality: PWAs should seamlessly work for every user, regardless of their web browser.
- Responsive: They should function on any device, be it a computer, laptop, tablet, or smartphone.
- Design: PWAs should mimic native mobile apps, featuring simplified and easily discoverable menus with straightforward interactivity for advanced functions.
- Security: PWAs must always use HTTPS to keep user data secure.
- Updates: PWAs stay updated, offering the latest versions of a service or site.
- Discovery: Users should easily find PWAs and identify them as an application rather than a website.
- Installation: They can be installed from any browser without additional steps and without the need for downloads from app stores.
- Sharing: PWAs require only a single URL for sharing, with no installation needed.

As you can see, Progressive Web Apps aim to provide users with a complete website experience with optimized features and the interface design of a native app.

IV. How They Work

The key to Progressive Web Apps is the use of service workers. A service worker is a controllable script that runs between the browser and the server, acting as a "separate entity from a web page, opening the door to functions that don't need a page or user interaction." These service workers form the foundation of the PWA standard, utilizing web caching for nearly instant results.

Service workers only last as long as their action is required. In a PWA, when you click on something or use a function, one of them comes into action, processing the event and deciding if the offline cache can fulfill the request. The idea is to have multiple offline caches for PWAs to select from, providing a much broader range of offline functionalities.

Moreover, the cache isn't just for offline speed boosts. For instance, if you run a PWA, but your connection is extremely irregular, the service workers can serve a previous cache in full operation, without disrupting the experience. In fact, there are websites so well-implemented that they load incredibly fast, even faster than a native app.

V. Browser Support

There are two requirements for using a Progressive Web App: a compatible browser and an enabled service. Websites like IsServiceWorkerReady allow checking browser compatibility with PWAs, and others like CanIUse specialize in listing the implementation of this technology for each version and browser. For example, if you search for service workers in the search bar, you'll find a table showing the version number with which each browser implemented them.

Progressive Web Apps:

Breaking down the current support status:

Desktop Browsers (Full Support): Chrome, Firefox, Opera, Edge, Safari

Desktop Browsers (Partial Support / Outdated Version): QQ, Baidu

Mobile Browsers (Full Support): Chrome, Firefox, Safari, UC Browser, Samsung Internet, Mint Browser, Wechat

Mobile Browsers (Partial Support / Outdated Version): QQ, Android Browser, Opera Mobile

We confirm that major web browsers are compatible with PWAs. Edge Chromium and Safari are the most recent additions to the full support list. On the contrary, alternative browsers like QQ and Baidu now use outdated versions and have fallen to the second tier.

VI. Finding Progressive Web Apps

PWAs are already everywhere. Many companies have adapted their websites and services to offer a Progressive Web App version. In many cases, you will find them directly while browsing the internet. For example, when you visit the mobile site of Twitter, a dialog box will pop up to "Add to Home Screen." Of course, visiting countless sites and waiting to see a trigger on the screen is not practical.

There are also several websites that catalog them. One of them is outweb, listing a fairly decent range of PWAs, with new options appearing frequently. Another good compilation can be found on pwa.rocks, with a lower number than the former but perhaps a more useful selection.

Additionally, starting from Chrome 72 for Android, Google added the Trusted Web Activity (TWA) feature, allowing Chrome tabs to open independently. This, in turn, enables PWAs to appear on the Google Play app store. The first PWAs that appeared on Google Play were Twitter Lite, Instagram Lite, and Google Maps Go. Many others have accumulated since then.

VII. References

References

[1] Ranchal, J. (2019, September 26). *Qué son, cómo funcionan y ejecutan las aplicaciones web progresivas*. MuyComputer. Recuperado de https://www.muycomputer.com/2019/09/26/aplicaciones-web-progresivas/

[2] *Minimum requirements for implementing Progressive Web Apps*. (2020, September 29). BBVA API_Market. Recuperado de https://www.bbvaapimarket.com/en/api-world/minimum-requirements-implementing-progressive-web-apps/

[3] MSEdgeTeam. (2023, January 24). *Overview of Progressive Web Apps (PWAs) - Microsoft Edge Development*. Microsoft Learn. Recuperado de https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/