APLICACIONES WEB PROGRESIVAS

CAIHUARA SOSSA FABIAN DARIO [1]

[1] Universidad Autónoma Juan Misael Saracho

Correo electrónico: fabiancaihuarasossa@gmail.com

RESUMEN

Las aplicaciones web progresivas combinan las ventajas de la web y las aplicaciones móviles nativas ofreciendo al usuario casi la misma funcionalidad de una aplicación con mayor accesibilidad de navegación, cabe diferenciar con una aplicación hibrida, que utiliza tecnologías web envuelta en una aplicación nativa. En el presente artículo se describen las características técnicas, un conjunto de conceptos y tecnologías desde un punto de vista de unificar el desarrollo de aplicaciones web-nativas.

PALABRAS CLAVE

Web movil, Aplicaciones Web Progresivas(PWA), Service Workers, aplicaciones interpretadas, aplicaciones hibridas, aplicaciones multiplataforma.

INTRODUCCIÓN

Tradicionalmente en el desarrollo de aplicaciones móviles, la reusabilidad de código fuente entre aplicaciones nativas y web no es posible generando como resultado proyectos y entornos de desarrollo diferentes cuando se desea que un aplicativo soporte diferentes plataformas móviles y sistemas operativos.

Las aplicaciones web progresivas, es un término de un nuevo tipo de aplicaciones que acerca a la unificación de aplicaciones web-nativas, incrementando su funcionalidad, conforme las capacidades del dispositivo en el que se ejecuta, de ahí la palabra progresiva, web por qué se hace referencia a su desarrollo basado en tecnologías web.

Existen muchos enfoques para el desarrollo multiplataforma, además de una gran cantidad de frameworks o plataformas bajo licencia de código abierto como productos patentados. Ejemplo de frameworks populares están Ionic, PhoneGap, ReactNative y Xamarin, los frameworks anteriormente mencionados representan tres enfoques tecnológicamente distintos.

Ionic y PhoneGap representan el enfoque hibrido, basado en la web (PhoneGap) y en Cordova (Ionic), donde los componentes de la interfaz del usuario son exclusivamente desarrollados con tecnología web. ReactNative representa el enfoque de interpretación, donde los componentes son desarrollados con ReactJs y aprovecha de un intérprete de JavaScript lo que se obtiene como resultado componentes nativos en lugar de componentes web. El enfoque de Xamarin se conoce como compilación multiplataforma, puesto que compila código fuente en C# y lo convierte en binarios nativos para cada plataforma compatible generando aplicaciones nativas que no dependen de intérpretes o componentes web.

Un nuevo conjunto de estándares propuestos por un grupo de investigación de Google busca unificar, mediante la introducción de funciones, como soporte sin conexión, sincronización en segundo plano e instalar el home-screen en el navegador, este enfoque se conoce como aplicaciones web progresivas (PWA), un término acuñado por Russel y Berriman (2015) en un artículo que cubre ideas de diseño iniciales. La contribución de las PWA en la unificación de

la experiencia móvil y la visión multiplataforma es posible al Service Worker API.

2. CARACTERISTICAS

Una aplicación web progresiva es:

- **Estándar:** utiliza la misma plataforma y tecnología que se utiliza para crear páginas web: HTML, CSS y Javascript.
- **Progresiva:** funciona para todos los usuarios, independientemente de cuál navegador web o sistema operativo utilice, porque está construida para mejorar progresivamente desde el principio.
- **Responsiva:** se ajusta a cualquier resolución y formato de pantalla: escritorio, móviles, tabletas o televisiones.
- **Independiente de la conexión:** está mejorada con service workers para funcionar sin conexión o en redes lentas con conexiones intermitentes.
- **Como una aplicación nativa:** el usuario la usará como una aplicación, con soporte para navegación e interacción con gestos.
- **Fresca:** siempre estará actualizada gracias al proceso automático de actualización del service worker.
- **Segura:** trabaja sobre HTTPS para prevenir que alguien intercepte datos y para asegurarse de que el contenido no ha sido manipulado por otros.
- **Descubrible:** es identificable como una "aplicación" gracias al manifiesto de la W3C y al registro de funciones del service worker, permitiendo a los buscadores web encontrarlas.
- **Interactiva:** hace fácil interactuar con ella incluso cuando está cerrada con características como notificaciones tipo "push".
- **Instalable:** le permite a sus usuarios crear accesos directos en la pantalla de su teléfono.
- **Enlazable:** se pueden compartir fácilmente usando su dirección en la web (URL) y no requiere procesos de instalación complejos.

3. SERVICE WORKERS

Los Service workers son una tecnología interesante y a la vez compleja que nos permite ejecutar servicios en segundo plano en los navegadores.

Van más allá de lo que ofrece un Web Worker. Éstos últimos nos permiten ejecutar código pesado en segundo plano (en un subproceso dedicado) y comunicarnos con ellos, de modo que una o varias tareas largas no bloqueen la interfaz de usuario. Pero los Service Workers son más potentes y complejos, puesto que pueden ejecutarse de manera independiente a la aplicación (es decir, estar en ejecución aunque la página de la aplicación web esté cerrada) y ofrecen capacidades avanzadas como la intercepción de las comunicaciones, el cacheado de información, la descarga en segundo plano de contenidos, el trabajo sin conexión o la posibilidad de enviar notificaciones.

En realidad para algunas cosas no es necesario utilizar un Service Worker. Por ejemplo, es posible crear aplicaciones web que funcionen off-line (sin conexión) utilizando la API de AppCach, pero un Service Worker nos puede dar más funcionalidad, sobre todo si necesitamos tomar decisiones a la hora de cachear la información y no solo asegurar que se encuentra almacenada.

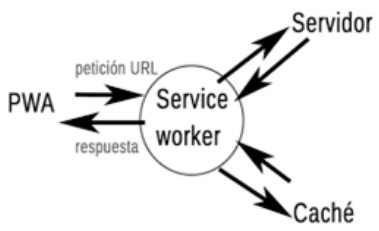

Figura 1: Funcionamiento Service Worker

Actualmente los únicos navegadores que permiten el uso de Service Workers son Chrome (el promotor de los mismos) y Firefox. El primer navegador móvil que ofrece soporte para esta tecnología es Chrome 51, recientemente el navegador Safari está implementando el soporte de esta tecnología.

Ciclo de vida del service worker

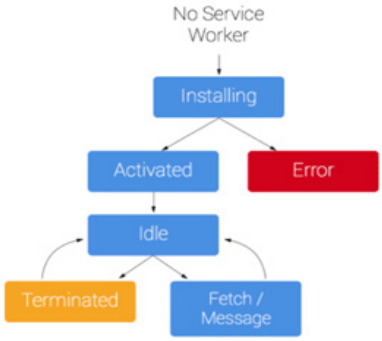

Figura 2: Ciclo de vida de un service worker

Install: Este es el primer evento, ocurre solo una vez por Service Worker. Si la funcion que se llama en este evento falla, el navegador no lo registra y no deja que el Service Worker tome control del cliente.

Activate: Cuando el Service Worker está controlando al cliente y está listo para usar y manejar eventos, se pasa al estado Activate. Se podría entender como que el Service Worker está activo.

Terminated, Fetch: Una vez se toma el control de las páginas, el service worker podrá estar en uno de estos dos estados. Terminated se aplica para ahorrar memoria. Por otro lado, si está haciendo manejo de peticiones de red, su estado será Fetch.

4. APP SHELL

Una arquitectura de app shell (app shell) es una forma de crear una aplicación web progresiva que se carga al instante y de manera confiable en la pantalla del usuario, en forma similar a lo que se observa en una aplicación nativa.

La "shell" es la mínima cantidad de HTML, CSS y JavaScript requeridos para activar la interfaz de usuario, y cuando se almacena en caché sin conexión puede asegurar un rendimiento instantáneo y de alta confiabilidad para los usuarios que realizan visitas repetidas veces. De esta manera, la app shell no se carga desde la red en cada visita del usuario. Solo se carga el contenido necesario de la red.

Para aplicaciones de una sola página con mucho código JavaScript, una app shell es un enfoque acertado. Este enfoque se basa en almacenar la shell agresivamente en caché (utilizando un service worker para lograr que la aplicación funcione). Luego, el contenido dinámico carga cada página a través de JavaScript. Es útil para enviar el HTML inicial a la pantalla de forma rápida y sin generar tráfico en la red.

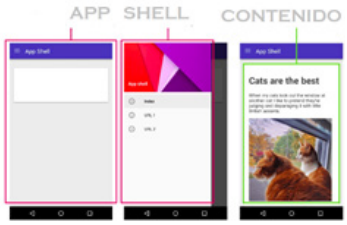

Figura 3: App shell

En otras palabras, la app shell es similar al paquete de código que se publicaría en una tienda de aplicaciones al compilar una aplicación nativa. Es el esqueleto de la interfaz de usuario y contiene los componentes principales necesarios, pero probablemente no contenga los datos.

Una arquitectura de app shell es realmente útil para las aplicaciones y los sitios con navegación relativamente sin cambios pero con contenido cambiante. Cierto número de bibliotecas y marcos JavaScript modernos ya promueven la división de la lógica de su contenido, ya que de esta manera la arquitectura se aplica más directamente. Para cierto tipo de sitios web que solo tienen contenido estático, también puedes seguir el mismo modelo, pero el sitio es 100 % shell de aplicación.

Algunos de los beneficios de una arquitectura de app shell con un service worker son:

- **Rendimiento confiable y rápido en todo momento:** Las visitas repetidas son extremadamente rápidas. Los recursos estáticos y la interfaz de usuario (p. ej. HTML, JavaScript, imágenes y CSS) se almacenan en caché en la primera visita, y luego se cargan instantáneamente en las visitas repetidas. El contenido se puede almacenar en caché en la primera visita, pero generalmente se carga cuando se necesita.
- **Interacciones similares a las aplicaciones nativas:** Al adoptar un modelo de app shell, puedes crear experiencias con navegación e interacciones completas con soporte sin conexión.
- **Uso económico de datos:** Diseñado para hacer un uso de datos mínimo y con criterio en lo que almacena en caché, ya que guardar archivos que no son esenciales (imágenes grandes que no se muestran en todas las páginas, por ejemplo) provocará que los navegadores descarguen más datos de lo estrictamente necesario.
- **App Shell no es una tecnología:** sino un modelo o patrón a la hora de crear las aplicaciones. La idea es muy sencilla: separar la aplicación entre funcionalidad y contenido y cargarlos por separado.

5. MANIFIESTO DE APLICACIÓN

Desde los primeros teléfonos inteligentes, ha sido posible anclar al inicio una página web desde el navegador para luego poder ir directamente a ella. Para controlar el aspecto que tendrá el icono que los usuarios van a anclar es posible utilizar diversas técnicas dependiendo del navegador y el sistema operativo. Así, en iOS o Windows Phone eso se controla a través de unas cabeceras de tipo "meta" que podemos añadir a la página principal de la aplicación web. En el caso de Android y Chrome se utiliza un archivo llamado "Manifiesto" cuyo nombre es manifest.json (que funciona hace años, desde la versión 38 de Chrome en 2014).

Hace poco Google ha hecho que cuando se añade una aplicación al menú de inicio de Android salga una pantalla de instalación como el de una aplicación real, todo ello conducente a que la experiencia cada vez sea más parecida a la de las aplicaciones nativas.

El manifiesto de las aplicaciones web es un archivo JSON simple que permite que el desarrollador, pueda controlar cómo se muestra su aplicación al usuario en áreas donde normalmente ven aplicaciones nativas (por ejemplo, la pantalla de inicio de un dispositivo móvil), además de indicar lo que el usuario puede ejecutar y definir parámetros de apariencia al iniciarse.

Los manifiestos permiten guardar un marcador de sitio en la pantalla de inicio de un dispositivo. Cuando un sitio se inicia de esta manera:

Tiene un ícono y un nombre único para que los usuarios puedan diferenciarlo de otros sitios.

Muestra una interfaz al usuario mientras se descargan los recursos o se restauran a partir del caché.

Proporciona características predeterminadas de visualización al navegador para evitar una transición demasiado brusca cuando los recursos del sitio se encuentren disponibles.

Hace todo esto a través del simple mecanismo de metadatos en un archivo de texto.

```
{
  "short_name": "AirHorner",
  "name": "Kinlan's AirHorner of Infamy",
  "icons": [
    {
      "src": "launcher-icon-1x.png",
      "type": "image/png",
      "sizes": "48x48"
    },
    {
      "src": "launcher-icon-2x.png",
      "type": "image/png",
      "sizes": "96x96"
    },
    {
      "src": "launcher-icon-4x.png",
      "type": "image/png",
      "sizes": "192x192"
    }
  ],
  "start_url": "index.html?launcher=true"
}
```

Figura 4: Ejemplo manifiesto de aplicación

6. COMPARACIÓN DE CARACTERISTICAS Y METRICAS ENTRE LOS DIFERENTES ENFOQUES

La Tabla 1 muestra una comparación de características disponibles en las aplicaciones web progresivas, interpretadas, hibridas y nativas.

Característica	Nativa	Interpretada	Hibrida	PWA
Instalable	Si	Si	Si	Si (a)
Uso sin conexión	Si	Si	Si	Si
Usable antes de la instalación	No	No	No	Si
Disponible en tiendas de aplicación	Si	Si	Si	Si (b)
Notificaciones Push	Si	Si	Si	Si (c)
Multiplataforma	No	Si	Si	Si
Acceso al hardware y sistema operativo mediante APIs	Si	Si	Si (e)	Limitado (d)
Sincronización en segundo plano	Si	Si	Si	Si

Tabla 1: Comparación características de enfoques

a). La instalación es mediante la creación de Home-screen, un acceso directo que permite el uso sin conexion.

b). Las aplicaciones web progresivas actualmente pueden ser buscados en la tienda de aplicaciones de windows 10, por el momento no cuenta con una propia.

c). Las notificaciones están disponibles por intermedio de Push API

d). Las aplicaciones web progresivas pueden usar APIs basadas en HTML5 en conjunto con características y funcionalidades posibles por service workers.

e). Las aplicaciones hibridas obtienen acceso a hardware y al sistema operativo mediante la librería Cordova

De la tabla se puede destacar las diferencias en las características de compatibilidad entre los enfoques para el desarrollo de aplicaciones. Hay diferencias bien marcadas, algunas son características inherentes.

Los PWA son la única opción entre los enfoques mencionados que, naturalmente, permiten probar una aplicación antes de la instalación, debido a que es accesible en los navegadores web. Si es necesario estar disponible en tiendas de aplicaciones, los otros tres enfoques son recomendables, sin embargo con una posible entrada de PWA en la tienda de aplicaciones de Microsoft abre la posibilidad de posicionar al enfoque de igual manera que los anteriores.

Las capacidades de trabajo sin conexión, las notificaciones push y la sincronización en segundo plano están disponibles independientemente del enfoque. La compatibilidad entre plataformas se encuentra en los enfoques interpretado, híbrido y de PWA, el último con algunas limitaciones. Como se mencionó, esta es una de las razones principales por las cuales los enfoques multiplataforma se han convertido en alternativas populares al desarrollo nativo.

Para las aplicaciones que dependen de funciones que no se encuentran o que requieren un rendimiento que no se puede alcanzar en un navegador actual, los enfoques alternativos son más adecuados. Las aplicaciones web progresivas están restringidas a las API de las plataformas disponibles a través del

navegador, por lo que dependen del W3C y de los proveedores de navegadores.

Métrica	Interpretada	Hibrida	PWA
Tamaño de instalación	4,53MB	16,39MB	104KB
Tiempo de ejecución	860ms	246ms	230ms

Tabla 2: Métricas comparativas de los enfoques

La tabla 2 presenta una comparación de dos medidas: tamaño de instalación y tiempo de ejecución. El tamaño de instalación de una aplicación web progresiva es aproximadamente 157 veces más pequeño que una aplicación interpretada y aproximadamente 43 veces más pequeño que una aplicación hibrida basada en Ionic.

En tiempo de ejecución la aplicación hibrida tarda casi 4 veces más que una aplicación web progresiva y posee un tiempo similar al de una aplicación interpretada.

CONCLUSIONES

Una evidente diferencia entre aplicaciones web y aplicaciones móviles normales es la experiencia de navegación. Una aplicación móvil regular necesita ser buscada e instalada por una tienda de aplicaciones. Las aplicaciones web progresivas ofrecen lo mejor de ambos enfoques, una interfaz fácil y cómoda, la instalación mediante Home-Screen un acceso directo y que permite el uso de la aplicación en modo sin conexión.

La experiencia de pantalla completa de las aplicaciones web progresivas, la forma de instalación puede considerarse un avance en la unificación de la experiencia y percepción del usuario final con respecto a la web móvil. En lugar de obligar a los usuarios a descargar una aplicación de una tienda de aplicaciones, pueden experimentar el producto en su navegador web como un sitio web regular antes de instalar el Home-Screen.

Desde la perspectiva de la unificación de las aplicaciones web y nativas, hay ciertas limitaciones principales del enfoque de aplicaciones web progresivas comparadas con las hibridas, interpretadas y nativas.

Malavolta menciona que una aplicación web progresiva tiene un acceso limitado a hardware y sistema, el mismo que el navegador web que lo soporta, por ejemplo no se puede acceder al calendario y a la lista de contactos, sin embargo el crecimiento del conjunto de APIs deja abierta la posibilidad de que los navegadores webs tenga acceso sin restricciones a un dispositivo.

La industria está invirtiendo recursos en aplicaciones web progresivas (PWA) y en el desarrollo de material de aprendizaje. La falta de participación académica denota una brecha de conocimiento significativa, pero al mismo tiempo proporciona un potencial de investigación. El grupo Google Web Fundamentals es una de las principales fuerzas impulsoras detrás de la defensa de los PWA. Podrían ser considerados como el principal editor de material de aprendizaje. El estado actual de las aplicaciones web progresivas implica la falta de ciertas API de plataforma y hardware y características a las que solo (ciertas) aplicaciones nativas y multiplataforma pueden acceder. Los avances recientes en los navegadores han sido fuerzas de unificación para la experiencia de la aplicación para el usuario final, incluidas, entre otras, las aplicaciones web instalables y nativas. Si bien Chrome está marcando el camino para la compatibilidad con PWA, los demás están siguiendo los pasos haciendo sus plataformas compatibles con service workers, fundamental para el funcionamiento de las aplicaciones web progresivas.

Descubrimos que existe un gran potencial para que los PWA se vuelvan unificadores para el desarrollo nativo de la web sin el uso de marcos multiplataforma. Como usuario final, el proceso de instalación de PWA se vuelve más similar a las aplicaciones regulares a través

de nuevos avances en los aspectos de experiencia del usuario. Las aplicaciones web pueden verse, sentirse y actuar de forma similar a las aplicaciones nativas, híbridas e interpretadas. Si bien existen limitaciones de APIs de hardware y plataforma para los PWA que no se encuentran en los otros enfoques, los requisitos y las especificaciones del producto al final serán el factor decisivo para la elección del enfoque.

Se concluye con, una cita de Archibald (2016) de la Conferencia Google I/O 2016: "Queremos que la web sea parte de primera clase del sistema operativo, en la mente del usuario".

BIBLIOGRAFÍA O REFERENCIAS

Archibald, J. (2016). Instant loading: Building offline-first progressive web apps.

Ater, T. (2017). Building Progressive Web Apps: Bringing the Power of Native to the Browser. O'Reilly.

Pedersen, M. (2016). Progressive web apps: Bridging the gap between web and mobile.

Perchat, J., Desertot, M., and Lecomte, S. (2013). Component based framework to créate mobile cross-platform.

Service Worker Google Developers: https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle [consultado 25/05/2018]

Manifiesto Web Google Developers: https://developers.google.com/web/fundamentals/web-app-manifest/?hl=es [consultado 25/05/2018]

Rinaldi, B., Holland, B., Looper, J., Motto, T., and VanToll, T. J. (2016). Are progressive web apps the future?

Rossi, J. (2016). The progress of web apps - microsoft edge dev blog.

Russel, A. and Berriman, F. (2015). Progressive web apps: Escaping tabs without losing our soul.