Esquemas de Recuperación de Datos en Aplicaciones Web Progresivas

Alumno: Cifuentes, Franco Ignacio

Tutora técnica: Rodríguez, Rocío Andrea

Profesora de Trabajo Final: Samela, Marcela

Trabajo Final de Carrera presentado para obtener el título de
Licenciatura en Gestión de Tecnología Informática
Universidad Abierta Interamericana

Marzo, 2022

Resumen

En los últimos años existió un avance en el desarrollo de aplicaciones web y nativas en los dispositivos móviles, como consecuencia del uso masivo por parte de los usuarios y de la mejora de los recursos tecnológicos. Sin embargo, ninguna de éstas resolvió la problemática de multiplataforma, tiempo de carga, uso de batería y manejo del ancho de banda en conjunto. Por este motivo, ha sido de interés el estudio de las aplicaciones web progresivas puesto que han resuelto los inconvenientes planteados. Las mismas han sido herramientas promovidas por Google para el desarrollo de sitios web ya que poseen características de aplicaciones nativas o híbridas sin ser una de ellas.

El presente trabajo de investigación tuvo como objetivo implementar un esquema de recuperación de datos, mediante secuencias de comandos que el navegador ejecuta en segundo plano y el análisis de las estrategias existentes. Por consiguiente, se diseñó un prototipo basado en una arquitectura donde se combinó el uso de versionado de la cache y el guardado de fechas de actualización de los datos.

Finalmente, luego de las pruebas realizadas, se identificó que el esquema propuesto otorga un mejor rendimiento en el manejo de la información y datos móviles en una aplicación web progresiva.

Palabras Clave: análisis de rendimiento, aplicación web progresiva, cache, recuperación de información

Abstract

In recent years there has been an advance in the development of web and native applications for mobile devices, because of the massive use by users and the improvement of technological resources. However, none of these applications solved the problems of multiplatform, load time, battery usage and bandwidth management as a unit. For this reason, the study of progressive web applications has been of interest since they have overcome these problems. These have been tools developed by Google for the creation of web sites because they have native or hybrid application features without being one of them.

The present research work had the objective of implementing a data retrieval scheme, by using scripts that the browser executes in the background and the analysis of existing strategies. Therefore, a prototype was designed based on an architecture that combined the use of cache versioning and the saving of data update dates.

Finally, after the tests performed, it was determined that the proposed scheme provides better performance in the management of information and mobile data in a progressive web application.

Keywords: cache, data recovery, performance analysis, progressive web app, service worker

Dedicatoria

Dedico este trabajo final de carrera a mi hermana Daiana, mi madre Susana y mi padre Ignacio por la ayuda, ánimo y espíritu brindado. A mis amigos, por su motivación y acompañamiento constante. A mi querida novia, por su cariño y amor durante mis últimos años de carrera. A mi gran referente Horacio, por su empuje y motivación.

De igual manera, quiero agradecer a mis maestros y la institución por todos los años de enseñanza. Para ellos es esta dedicatoria, pues es a ellos a quienes les agradezco por su apoyo incondicional.

Reconocimientos

La realización de este trabajo final fue posible, en primer lugar, a la cooperación brindada por la tutora Dra. Rocío Andrea Rodríguez del CAETI (Centro de Altos Estudios en Tecnología Informática), por su dirección y ayuda constante, en especial por su orientación metodológica y continuo estímulo durante todo el proceso hasta al final de este. Asimismo, quiero expresar mi agradecimiento al Dr. Pablo Martín Vera también investigador del CAETI quién había trabajado previamente en la temática de esta investigación.

Me gustaría reconocer también al Ing. Darío Guillermo Cardacci, director de carrera de la facultad de Tecnología Informática de la Universidad Interamericana, por su constante apoyo a lo largo de la carrera.

Se agradece a todas aquellas personas que en forma directa o indirecta contribuyeron a que este trabajo de investigación pudiera llevarse a cabo.

Índice General

Resumen ..... - 1 -
Abstract ..... - 2 -
Dedicatoria ..... - 3 -
Reconocimientos ..... - 4 -
Índice General ..... - 5 -
Índice de Figuras ..... - 7 -
Índice de Tablas ..... - 9 -
Capítulo 1 ..... - 10 -
Introducción ..... - 10 -
1.1 Justificación de la Elección del Problema ..... - 10 -
1.2 Problemas y Soluciones ..... - 10 -
1.3 Hipótesis ..... - 11 -
1.4 Propuesta ..... - 11 -
1.5 Cronograma Operativo y Recursos Necesarios ..... - 11 -
1.6 Objetivo General ..... - 12 -
1.7 Objetivos Particulares ..... - 12 -
1.8 Contribuciones Principales ..... - 12 -
1.9 Estructura General ..... - 12 -
Capítulo 2 ..... - 14 -
Marco Teórico y Estado del Arte ..... - 14 -
2.1 Introducción ..... - 14 -
2.2 Marco Teórico ..... - 14 -
2.2.1 Progressive Web Apps ..... - 14 -
2.2.2 Service Worker ..... - 15 -
2.3 Estado del Arte ..... - 16 -
2.3.1 Arquitecturas Básicas ..... - 17 -

2.3.2 ReWAP ..... - 17 -
2.3.3 Service Worker Infusion Framework ..... - 18 -
Capítulo 3 ..... - 19 -
Aplicaciones Web Progresivas ..... - 19 -
3.1 ¿Qué son? ..... - 19 -
3.2 Estructuración de Datos ..... - 19 -
3.3 Esquemas de Recuperación de Datos ..... - 19 -
3.4 Resumen ..... - 23 -
Capítulo 4 ..... - 24 -
Propuesta de Manejo de la Cache ..... - 24 -
4.1 Esquema de Recuperación de Datos ..... - 24 -
4.2 Definición del Esquema ..... - 24 -
4.3 Rendimiento ..... - 25 -
4.4 Resumen ..... - 25 -
Capítulo 5 ..... - 27 -
Aplicación Práctica ..... - 27 -
5.1 Introducción ..... - 27 -
5.2 Implementación ..... - 27 -
5.3 Validación ..... - 29 -
5.4 Resumen ..... - 41 -
Conclusiones ..... - 42 -
Líneas Futuras de Investigación ..... - 44 -
Acrónimos ..... - 45 -
Referencias ..... - 47 -

Índice de Figuras

Figura 1 - Flujo de información y uso de cache en una arquitectura SWIF ..... - 18 -

Figura 2 - Arquitectura de solo cache ..... - 20 -

Figura 3 - Arquitectura de solo red sin cache ..... - 20 -

Figura 4 - Arquitectura de utilización de cache y recurrir a la red ..... - 20 -

Figura 5 - Arquitectura de red y recurrir a la cache ..... - 21 -

Figura 6 - Arquitectura en respuesta de la red ..... - 21 -

Figura 7 - Arquitectura de cache luego red ..... - 22 -

Figura 8 - Arquitectura con respuesta genérica ..... - 22 -

Figura 9 - Arquitectura con envío de plantilla del lado del SW ..... - 23 -

Figura 10 - APP Shell y versión de cache ..... - 25 -

Figura 11 - Evento install del SW ..... - 28 -

Figura 12 - Evento fetch del SW ..... - 29 -

Figura 13 - App Shell de la prueba 1 ..... - 31 -

Figura 14 - Archivos obtenidos en la primera interacción con el servidor de la prueba 1 . - 31 -

Figura 15 - Cache versión 58 ..... - 32 -

Figura 16 - Llamado con archivos en cache versión 58 ..... - 32 -

Figura 17 - Funcionamiento offline de la home ..... - 32 -

Figura 18 - Funcionamiento offline de página interna clientes.html ..... - 32 -

Figura 19 - App Shell de la prueba 2 ..... - 33 -

Figura 20 - Archivos servidos en el llamado de la prueba 2 ..... - 33 -

Figura 21 - Cache versión 2 ..... - 34 -

Figura 22 - App Shell de la prueba 3 ..... - 34 -

Figura 23 - Archivos servidos en el llamado de la prueba 3 ..... - 35 -

Figura 24 - Cache versión 3 ..... - 35 -

Figura 25 - App Shell de la prueba 4 ..... - 35 -

Figura 26 - Archivos servidos en el llamado de la prueba 4 ..... - 36 -

Figura 27 - Cache versión 5 ..... - 36 -
Figura 28 - App Shell de la prueba 5 ..... - 37 -
Figura 29 - Archivos servidos en el llamado de la prueba 5 ..... - 37 -
Figura 30 - Cache versión 7 ..... - 37 -
Figura 31 - Cliente.html y botón de carga de clientes ..... - 38 -
Figura 32 - Evento fetch de los clientes ..... - 38 -
Figura 33 - Fecha de datos guardados en la cache storage de la prueba 6 ..... - 38 -
Figura 34 - Fecha de guardado de los datos en cache ..... - 39 -
Figura 35 - Fecha de actualización de los datos cliente en la base de datos ..... - 39 -
Figura 36 - Fecha de actualización obtenida desde la API ..... - 39 -
Figura 37 - Respuesta del llamado a la API Clientes ..... - 39 -
Figura 38 - Fecha de guardado de datos en cache ..... - 40 -
Figura 39 - Fecha de actualización de los datos cliente en la base de datos ..... - 40 -
Figura 40 - Fecha de actualización obtenida desde la API ..... - 40 -
Figura 41 - Desconexión de internet ..... - 40 -
Figura 42 - Datos y cartel al estar offline ..... - 41 -
Figura 43 - Evento fetch interceptado y servido desde el SW ..... - 41 -

Índice de Tablas

Tabla 1 - Cronograma operativo ..... - 11 -

Tabla 2 - Descripción de los casos de prueba ..... - 30 -

Capítulo 1

Introducción

1.1 Justificación de la Elección del Problema

La alta inserción de dispositivos móviles en la sociedad (Vorderer & Klimmt, 2020), junto con la evolución de los mismos, expandió las posibilidades de desarrollar un sinfín de aplicaciones con distintos objetivos que pudieran estar al alcance de forma masiva. Hace algunos años, el análisis del desarrollo de aplicaciones web estaba basado en la conveniencia de realizar aplicaciones nativas (instalables y realizadas para un sistema operativo en particular). Sin embargo, los nuevos estándares ofrecidos por el World Wide Web Consortium (W3C) permitieron expandir las capacidades de la web, reduciendo la brecha que las separaba de las aplicaciones nativas (Rodríguez, Vera, Vallés, F., & & Giulianelli, 2014).

En la actualidad, existe la posibilidad de que las aplicaciones web puedan ser instaladas y tengan las mismas prestaciones que las aplicaciones nativas si se desarrollan como Progressive Web Apps (PWA). De esta manera, la cuestión central reside en el rendimiento de las mismas y cómo optimizarlas para consumir la menor cantidad de recursos. En el presente trabajo se analiza y evalúa la implementación de un esquema combinado para la recuperación de datos, con el objetivo de optimizar las aplicaciones web móviles mediante el desarrollo de un service worker (SW).

1.2 Problemas y Soluciones

Los esquemas para manejo de la cache están enfocados en aplicaciones con características particulares, sin embargo, la creciente complejidad en la estructura de las aplicaciones requiere un desarrollo de esquemas combinados más sofisticados.

Un sitio web contiene muchos elementos recurrentes que se pueden almacenar en la cache para obtener datos eficientemente. Pero la mayoría de los desarrolladores de páginas web no se centran en optimizar la política de la cache de los recursos. Por este motivo, se propuso determinar una estructura de recuperación para el almacenamiento de recursos, manejo de la cache y solicitudes que pueden depender del tipo de dato, estado de la red y dispositivo utilizado.

Por lo mencionado en el párrafo previo, se desarrolló e implementó una propuesta capaz de otorgar una solución mediante la utilización de una estructura web, acceso a datos remotos, envío de datos y manejo de versiones.

1.3 Hipótesis

Este es un trabajo de carácter técnico experimental que pretende validar las siguientes hipótesis mediante el uso de un prototipo:

- El uso de SW permite implementar estrategias de almacenamiento en la cache en PWA.
- La implementación de versionado de código, esquemas de recuperación y llamada al servidor otorga mejoras en el rendimiento de aplicaciones web.

1.4 Propuesta

Para definir los límites y alcances de la presente investigación cualitativa se establecen como técnicas de recolección de datos a emplear el estudio de contenidos mediante la revisión de documentos.

En el presente trabajo se propone seleccionar una metodología que permita integrar una estrategia de recuperación de datos que sea utilizable en aplicaciones con características disímiles y demostrar su aplicabilidad por medio de un prototipo.

Para lograr este objetivo se analiza el rendimiento de las alternativas existentes en el mercado: cache, red y sus combinaciones, reserva genérica, ReWAP y Service Worker Infusion Framework, determinando la más adecuada a ser utilizada en cada contexto.

1.5 Cronograma Operativo y Recursos Necesarios

En la tabla 1, se presentan las tareas realizadas con el tiempo estimado de duración de las mismas.

Tabla 1

Cronograma operativo

N.º Tarea	Descripción	Tiempo estimado
1	Revisión del marco teórico/estado del arte	5 semanas
2	Recolección de información	4 semanas
3	Análisis de los esquemas existentes	6 semanas
4	Diseño de la propuesta	4 semanas
5	Construcción del prototipo	6 semanas
6	Implementación y validación del prototipo	4 semanas
7	Informe de resultados	3 semanas

Por otro lado, los recursos necesarios son mínimos ya que sólo se requiere una computadora y acceso a internet para desarrollar la investigación.

1.6 Objetivo General

Lograr la implementación de una solución integral que permita mejorar el uso de la cache en una PWA.

1.7 Objetivos Particulares

Este trabajo final de carrera tiene como objetivos particulares los siguientes:

- Analizar soluciones existentes para el manejo de datos en las PWA que se adapten a aplicaciones de diferentes características.
- Estudiar y evaluar los procedimientos existentes para el manejo de datos en aplicaciones web progresivas.
- Analizar posibles soluciones que brinden mejoras en el rendimiento de las PWA.

El presente trabajo ofrece un aporte al campo de investigación de las PWA teniendo en cuenta la arquitectura de recuperación de datos y los beneficios obtenidos en el rendimiento.

1.8 Contribuciones Principales

Las ventajas en distintos aspectos de las PWA, en contraposición de las aplicaciones nativas, son varias: conversión de retorno, compromiso de la interacción del cliente con el sitio web, velocidad de carga, tiempo de desarrollo, costo de mantenimiento y el peso de la aplicación (Beezer, 2020).

El desafío en el presente trabajo fue desarrollar e implementar el código necesario para realizar un aporte significativo considerando las ventajas mencionadas en el párrafo previo y teniendo en cuenta el trabajo “Estrategias de Manejo de Cache para Aplicaciones Web Progresivas” (Vera & Rodríguez, 2020).

El trabajo de investigación que se presenta a continuación establece la base para demostrar, mediante una lógica correctamente implementada, el impacto de un rendimiento superior en una aplicación web.

1.9 Estructura General

En este capítulo se desarrolla la justificación de la selección del problema a resolver junto con las particularidades para el abordaje del mismo.

En el capítulo 2 “Marco Teórico y Estado del Arte” se presenta una definición detallada de las PWA y sus características principales. Posteriormente, se realiza una revisión

sistemática de los trabajos académicos y avances existentes hasta la actualidad. Las diferentes estrategias descriptas en el capítulo llevan a definir los caminos transitados en el trabajo realizado.

En el capítulo 3 “Aplicaciones Web Progresivas” se describe con detalle la definición de una PWA y sus características principales orientando el contenido al manejo de información dentro de la cache. Se brindan los usos y definiciones de las arquitecturas, para poder tener una imagen general del estado actual del ecosistema tecnológico relacionado a las PWA. Cabe mencionar que este apartado es fundamental para el entendimiento del presente trabajo.

En el capítulo 4 “Propuesta de Manejo de la Cache”, teniendo en cuenta las arquitecturas existentes mencionadas en el capítulo previo, se define el esquema a implementar para obtener mejoras en el rendimiento mediante la utilización de dos funcionalidades: versionado de código y llamadas al servidor.

En el capítulo 5 “Aplicación Práctica” se presenta la implementación del esquema de la cache mediante un prototipo. Asimismo, puede visualizarse el código en funcionamiento y las pruebas realizadas permitiendo presentar un resultado que demuestra lo planteado a lo largo del trabajo, evidenciándose la validación de las hipótesis establecidas. Cabe aclarar que los estudios se han efectuado mediante la utilización de un navegador web, un servidor y una base de datos con datos de prueba.

Capítulo 2

Marco Teórico y Estado del Arte

2.1 Introducción

Con el correr de los años se ha puesto en evidencia el mayor uso de aplicaciones web respecto a aplicaciones nativas (Kaplan, 2020). Sin embargo, el usuario utiliza más tiempo navegando una aplicación nativa en comparación con una aplicación web (Sayal & Abhishek, 2018). De acuerdo con Rodríguez et al. (2020) "una gran parte de los usuarios están acostumbrados a instalar y utilizar distintas aplicaciones … aun cuando la versión web esté adaptada para ser utilizada correctamente en la pantalla reducida" (p. 140). Por este motivo, las aplicaciones web progresivas llegaron para ofrecer una experiencia de usuario similar a las nativas con el objetivo principal de lograr una fidelización y conversión online mayor.

2.2 Marco Teórico

Una persona o máquina puede captar un símbolo, signo o medida en determinada forma (Clarke, 1999) y ésto deriva en la definición de ciertos tipos de datos que se procederán a usar en este trabajo: datos alfanuméricos, imágenes, audios y videos.

Reynolds & Stairs afirman que "la información es una colección de hechos sin procesar organizados y procesados para que tengan un valor adicional más allá del valor de los hechos individuales" (pág. 5), manifestando la necesidad de tener una estructura de recuperación acorde a la información solicitada por el usuario. En adición, la cache es esencial en la rapidez del procesamiento de información puesto que logra una reducción de costos de tiempo y otros recursos en cada interacción con el servidor.

Dependiendo el área en estudio, la recuperación de los datos puede tener diferentes significados, pero en sistemas se refiere a aquellas rutinas algorítmicas ejecutadas por el sistema de información en respuesta a una solicitud del usuario. (Ribeiro Neto & Baeza Yates, 1999).

2.2.1 Progressive Web Apps

Este nuevo tipo de aplicación web fue creada por Google y utiliza HTML, CSS y JavaScript (Google Developers, 2019). Éstas funcionan en cualquier navegador, siempre y cuando exista una conexión HTTPS, un SW y un manifiesto.

Las PWA no son aplicaciones nativas, híbridas o móviles. Son una nueva forma de diseñar sitios web teniendo en cuenta la seguridad, mantenimiento, portabilidad y

funcionalidad. La empresa Mozilla enumera las características principales de la siguiente manera (Mozilla Developers, 2021):

- **Responsiva:** funciona en cualquier dispositivo sin importar la resolución del dispositivo.
- **Independiente de la conectividad:** con la ayuda del SW, puede funcionar con baja conectividad a internet y offline.
- **Adaptable:** posee características similares a las de una aplicación en términos de interacción y navegación.
- **Actualizada:** por el proceso de actualización del SW, se obtiene una información actualizada constantemente.
- **Segura:** se sirven mediante HTTPS.
- **Detectable:** son aplicaciones, pero pueden ser indexadas por los motores de búsqueda.
- **Reactivable:** la utilización de notificaciones push nos permite reactivar la aplicación.
- **Instalable:** un ícono en la pantalla de inicio permite al usuario guardar la aplicación sin necesidad de usar la tienda.
- **Enlazable:** la URL es fácil de compartir.

Las PWA son actualmente muy utilizadas y en diversos campos se encuentran desarrollos específicos: redes sociales (Gallagher, 2017) (Osmani, Pinterest Progressive Web App Performance, 2017) (Osmani, Tinder Progressive Web App Performance, 2017), empleos (Chi-Tsai, Tzuo-Ming, & Ming-Chih, 2019), eventos (Yberg, 2018), tiendas (Starbucks, 2015), educación (Yuliawan, Widyandana, & Nur Hidayah, 2020), comercio electrónico (Quynh, 2018).

2.2.2 Service Worker

Uno de los requisitos de las PWA es tener siempre un SW y como ha sido definido por Gambhir & Raj “es un script de fondo que se ejecuta independientemente de la capa de aplicación y puede recibir mensajes cuando no está activo, ya sea porque su aplicación está en el fondo o no abierto, o el navegador está cerrado” (pág. 295). Este script es la pieza fundamental de las PWA debido a su enorme poder en la administración de información, solicitudes de red de la página, notificaciones y manejo del DOM.

2.3 Estado del Arte

El objetivo de este apartado es lograr una revisión y análisis que permita demostrar la situación actual de las PWA para poder definir un esquema de recuperación de datos.

Para el abordaje del material se utilizó la información provista por Google en el manejo de recursos dentro de una PWA mediante la utilización del SW y, a su vez, se obtuvo información de artículos de la base de datos IEEE Xplore.

En un estudio dirigido por Singla et al. se demostró que "las mejoras en el ancho de banda también son necesarias pero el ancho de banda ya no es el cuello de botella para una fracción significativa de la población" (pág. 1). Por lo tanto, la reducción de latencia es un aspecto valioso debido al ahorro de dinero generado por la mayor velocidad entre transacciones. Adicionalmente, es evidente la mejora de la experiencia de usuario web ante la posibilidad de ofrecer respuestas instantáneas.

Las arquitecturas básicas definidas por la empresa Google (Google Developers, 2019) proporcionan las herramientas necesarias para poder plantear, desarrollar e implementar diferentes metodologías en torno al uso de la cache. El análisis de los estudios (Gambhir & Raj, 2018) (Malavolta, Chinnappan, Jasmontas, Gupta, & Ali Karam Soltany, 2020) demuestra que el rendimiento PWA es mejor que las aplicaciones nativas teniendo en consideración el uso de la cache y el impacto en el consumo energético.

Por otro lado, se analizaron diferentes investigaciones de arquitecturas de la cache teniendo en cuenta el uso de un proxy server (Pande, Somani, Prasad Samal, & Kakkirala, 2018) y un package engine (Liu, et al., 2018) para mejorar el uso de la misma en aplicaciones existentes.

Así como se ha visto la importancia de usar óptimamente la cache en (Biørn-Hansen, Majchrzak, & Grønli, 2017), se advierten las diferentes funcionalidades de las PWA (Alterxsoft, 2019) y cómo pueden jugar un papel importante al momento de decidir qué, cómo y cuales se pueden incluir en nuestra aplicación. La habilidad de las PWA de ser multiplataforma (del inglés, *cross-platform*) otorga ventajas en consideración a las aplicaciones nativas de modo que es un gran momento para aprovecharlas (Ilyukha, 2020).

Las implementaciones pueden ser sencillas, como un sistema de organización (Bhilare, Gaikwad, Kokare, Kumbhar, & Ekatpure, 2019) hasta un sistema complejo (Sheppard, 2017) con una gran variedad de características propias de las PWA (Padmanabhan, 2020), por lo tanto, puede ser llevado a la práctica en cualquier ámbito.

2.3.1 Arquitecturas Básicas

Las arquitecturas existentes en el manejo de recursos de una PWA ayudan a entender los recursos que se deben almacenar en la cache, su duración de guardado y la lógica a utilizar para actualizarlos. Por lo tanto, para construir una arquitectura óptima, se deben realizar las siguientes preguntas: ¿Qué tipo de archivo es? ¿Se está conectado a internet? ¿Qué tamaño tiene el archivo? ¿Es un archivo muy usado? ¿Cuándo conviene guardar el recurso?

Como respuestas a estos interrogantes, Google Developer (2019) define las arquitecturas básicas según el orden de recuperación de recursos:

- **Sólo cache:** se utiliza para recursos estáticos.
- **Sólo red:** se utiliza para aplicaciones que necesiten conexión.
- **Cache y recurrir a la red:** está orientado para pensar una aplicación en términos de primero sin conexión.
- **La carrera de la cache y la red:** para dispositivos con acceso al disco muy lento.
- **Red y recurrir a la cache:** no es recomendable porque, cuando el usuario tenga una conexión lenta, tomará mucho tiempo en mostrar el contenido de la cache y empeorará la experiencia de usuario.
- **Cache y después red:** contenido que se actualiza con frecuencia.
- **Contenido de reserva genérico:** imágenes secundarias, llamadas sin éxito o páginas como no disponible sin conexión.

2.3.2 ReWAP

La arquitectura de reducción de transferencias redundantes mediante empaquetamiento de recursos se define como:

> Una técnica de normalización para identificar el mismo recurso, pero con diferentes URL, un aprendizaje basado en técnicas para predecir con precisión las actualizaciones de recursos, y un algoritmo para minimizar la frecuencia de actualización de paquetes de recursos, así como reducir la sobrecarga. (Li et al., 2018, pág. 10)

La reducción de transferencia de recursos duplicados entre el cliente y servidor permite un ahorro de latencia significativa al momento de realizar pruebas de rendimiento. Para lograr esto, se desarrolló un servicio en el servidor proxy que racionaliza los recursos duplicados, filtra por HTTP y selecciona los más relevantes. A continuación, el gestor de

paquetes multidispositivo agrupa los recursos seleccionados, empaqueta y envía al generador de SW para almacenarlos en caché.

2.3.3 Service Worker Infusion Framework

Este servicio está orientado a la clasificación de recursos, generación de archivos JavaScript y la implementación de un manejo de paquetes para poder tener presencia en dispositivos que no soportan PWA.

Su algoritmo está enfocado en resolver de manera eficiente los cuellos de botella considerando los principios listados a continuación:

- Racionalización de recursos duplicados.
- Filtrado de recursos por encabezados HTTP.
- Captación previa de recursos.

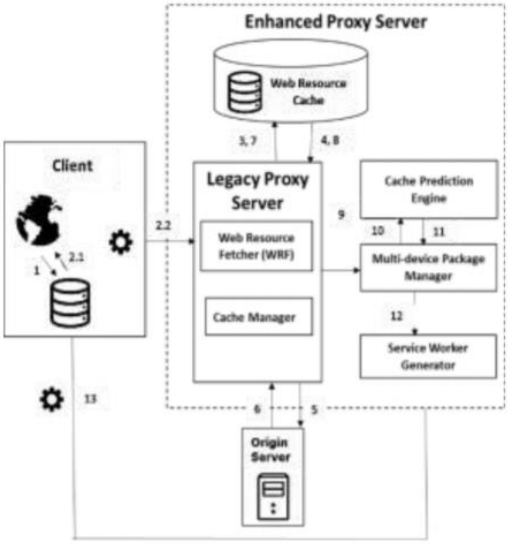
Pande, Somani, Prasad Samal y Kakkirala (2018) afirman lo siguiente:

> El almacenamiento en cache inteligente de recursos no obsoletos y la agrupación de recursos web junto con el service worker en el proxy server puede ayudar en la reducción significativa de innecesarias transferencias de datos entre el servidor proxy y los dispositivos del cliente. (pág. 201)

En la Figura 1 extraída de (Pande, Somani, Prasad Samal, & Kakkirala, 2018) se observa el flujo de datos en una arquitectura Service Worker Infusion Framework (SWIF).

Figura 1

Flujo de información y uso de cache en una arquitectura SWIF



Nota. Adaptado de *Working flow of proposed solution*, de Pande et al., 2018, Enhanced Web Application and Browsing performance through Service-worker Infusion Framework

Capítulo 3

Aplicaciones Web Progresivas

3.1 ¿Qué son?

De acuerdo con Jiyeon Lee (2018) las PWA son consideradas como “una nueva generación de aplicaciones web diseñadas para proporcionar experiencias de navegación similares a las de una aplicación nativa” (pág. 1). Las mismas se encuentran impulsadas por la empresa Google desde el año 2015 y permiten transformar plataformas web en aplicaciones web con una semejanza a las aplicaciones nativas basadas en sistemas operativos como iOS o Android.

Como se indicó en el capítulo anterior, las PWA se encuentran en pleno auge y en la actualidad se encuentran desarrollos en diversos campos como redes sociales, empleos, eventos, tiendas, educación o comercio electrónico.

3.2 Estructuración de Datos

Los tipos de archivos que se pueden guardar en la cache son los siguientes: HTML, JS, CSS, imágenes (.png, .jpg, .jpeg, etc.) y JSON (respuesta de las APIs). En la mayoría de los casos, las cuatro primeras estructuras no cambian con frecuencia por lo que el manejo de éstas no debería generar un problema. En el caso de la información obtenida mediante las llamadas a distintas APIs, se corre el riesgo de mostrar inconsistencias al cliente durante la navegación, producto del manejo y guardado en cache.

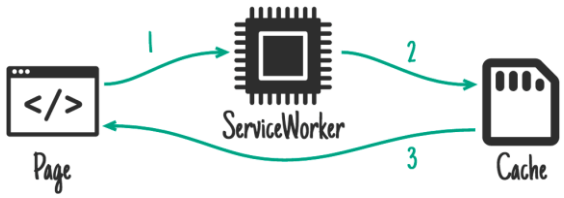
3.3 Esquemas de Recuperación de Datos

El SW no usa la cache a menos que se le indique hacerlo. Por lo tanto, se tienen diferentes formas de manejar las llamadas. En la Figura 2 extraída de (Archibald, 2020) se debe guardar en la cache, mediante el uso del método *install*, la información que no se desea servir en interacciones posteriores. En la Figura 3 extraída de (Archibald, 2020) no existe la opción del método *offline* por lo que se pierde una gran característica de las PWA y, por lo tanto, no es útil para el desarrollo del trabajo.

Las estrategias mencionadas previamente son la base de las estrategias posteriores más complicadas que se verán, por lo que no es muy común que sean objeto de estudio de casos específicos.

Figura 2

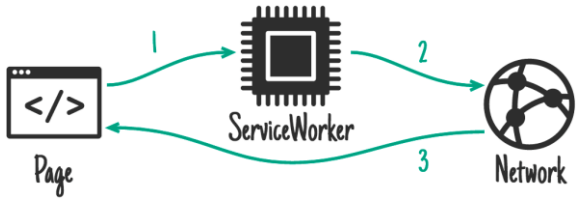
Arquitectura de solo cache



Nota. Adaptado de *Solo cache*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

Figura 3

Arquitectura de solo red sin cache

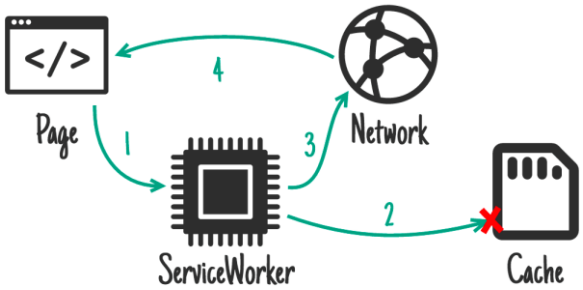


Nota. Adaptado de *Solo red*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

En la Figura 4 extraída de (Archibald, 2020) se puede percibir que la estructura planteada permite la búsqueda del llamado en la cache y servirlo en caso de encontrarlo. De lo contrario, el llamado será servido por la web y la respuesta se guardará en la cache para futuras llamadas. Esta estrategia se denomina cache y recurrir a la red (del inglés, *cache falling back to network*) y permite obtener un comportamiento combinado de las estrategias iniciales por el hecho de solicitar los datos al cache cuando se necesiten y, por otro lado, obtener desde la web información que no se desee o no se pueda tener guardada en la cache.

Figura 4

Arquitectura de utilización de cache y recurrir a la red



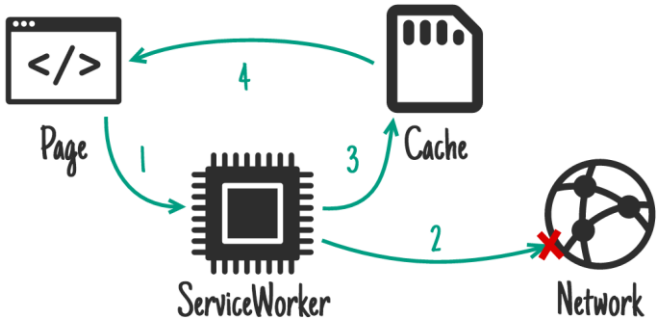
Nota. Adaptado de *Cache primero y luego vuelta a la red*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

La estructura de la Figura 5 extraída de (Archibald, 2020) es ideal para llamados que se encuentran actualizando constantemente debido a que inicialmente se intentará obtener una respuesta desde la web y en caso de ser exitosa se servirá. De lo contrario, se buscará la respuesta guardada previamente en la cache. El eje central es darle al usuario con conexión a internet el contenido más actualizado y ofrecerles a los usuarios sin conexión una experiencia aceptable mediante la utilización de información guardada en la cache. Para lograr esto es

necesario renovar los registros de información contenidos en la cache, como se observa en la Figura 6 extraída de (Archibald, 2020). La finalidad planteada en la última estrategia es obtener y mostrar información desde la red mientras la misma se guarda en la cache. Ésto es sumamente útil para posteriores llamados, aunque es necesario considerar el tamaño del disco porque se va guardando toda la información nueva constantemente. Por lo tanto, es recomendable deshacerse de los elementos de la cache que no se necesiten.

Figura 5

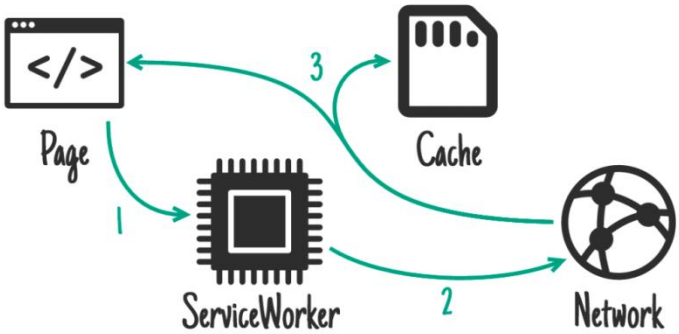
Arquitectura de red y recurrir a la cache



Nota. Adaptado de *Red primero y luego vuelta a la cache*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

Figura 6

Arquitectura en respuesta de la red



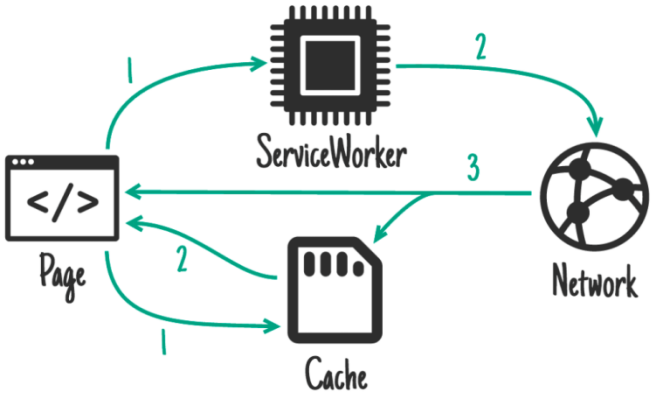
Nota. Adaptado de *Cuando la red responde*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

La arquitectura de red y recurrir al cache (del inglés, *network falling back to cache*) de la Figura 5 tiene una falencia muy importante: la baja señal o intermitencia en la conexión provoca que el usuario deba esperar un tiempo para que el llamado a la red falle. Ésto sugiere que la experiencia de usuario se vea afectada y el cliente se sienta frustrado. En cambio, con la estrategia de la Figura 7 extraída de (Archibald, 2020) llamada cache luego red (del inglés, *cache then network*), se ejecutan dos llamados, uno al cache y otro a la red. En una primera instancia, se muestran los datos obtenidos en la cache para poder brindar una experiencia de usuario más placentera. En el instante que la información se obtiene desde la red, la página web actualizará la información y así el cliente podrá estar a tiempo con la misma. Aunque la estrategia sea muy recomendable, la interfaz de usuario debe ser desarrollada de manera

correcta para no estar actualizando, moviendo o borrando información en la web que el usuario pudiese estar navegando.

Figura 7

Arquitectura de cache luego red

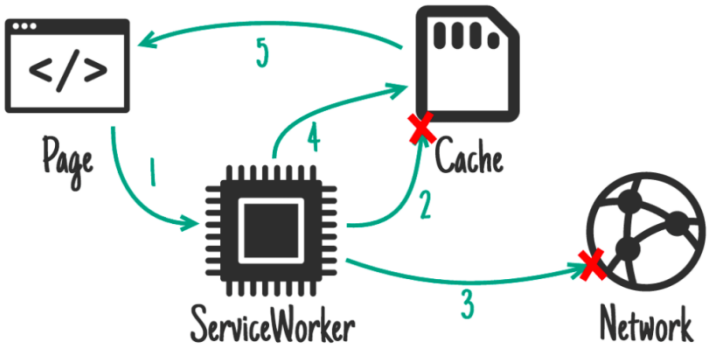


Nota. Adaptado de *Cache luego red*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

Se pueden combinar de muchas maneras la información de la red y la guardada en la cache pero el caso donde no se obtiene información de ningún lugar conduce a la estrategia de la Figura 8 extraída de (Archibald, 2020). En esta estrategia se describe una respuesta genérica en cada solicitud que se realice. Normalmente, es un mensaje o una imagen indicando la leyenda de “Sin contenido”, “Sin conexión a internet”, “No disponible sin internet”, etc.

Figura 8

Arquitectura con respuesta genérica

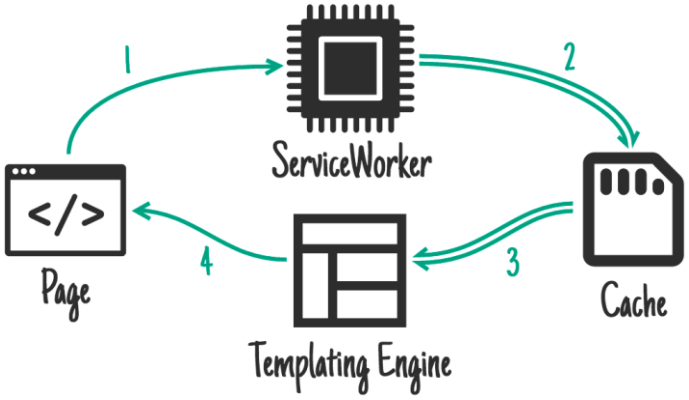


Nota. Adaptado de *Método de respaldo genérico*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

El renderizado del lado del servidor mejora la velocidad de la web y se tiene constancia que el SW es un intermediario entre la web y el servidor con un comportamiento similar a éste último. Para este caso se tiene en cuenta la arquitectura de la Figura 9 extraída de (Archibald, 2020), en donde se utiliza una plantilla junto con los llamados de información, para que el SW combine ambas y luego sean servidas en conjunto.

Figura 9

Arquitectura con envío de plantilla del lado del SW



Nota. Adaptado de *Plantillas del lado del service worker*, de Archibald, 2020, Offline Cookbook (https://web.dev/offline-cookbook/)

3.4 Resumen

En el presente capítulo se describió la visión general de una PWA y su propósito con el desarrollo de aplicaciones web. La estructuración de datos se define como en la mayoría de las páginas web modernas pero lo más destacable es la capacidad y definición que tiene de mutar en base a la solución que se necesite.

Para lograr esta tarea se plantean diversas arquitecturas, desde las más simples hasta las más complejas. La combinación de las mismas es la clave del éxito y se recomienda hacer énfasis en la experiencia del usuario y la interfaz de la web.

Como desarrolladores y usuarios se deben tener en cuenta las numerosas herramientas y la mejor manera de combinarlas para lograr un producto con una calidad excepcional.

Capítulo 4

Propuesta de Manejo de la Cache

4.1 Esquema de Recuperación de Datos

Luego de haber analizado en el capítulo anterior los esquemas de cache existentes, se extenderá el análisis de manejo de cache basado en el trabajo “Estrategias de Manejo de Cache para Aplicaciones Web Progresivas” (Vera & Rodríguez, 2020).

En primera instancia, la propuesta consiste en implementar el manejo de versiones a la cache presentada en el trabajo previamente mencionado. El versionado de la cache es utilizado dentro del SW para que los recursos previamente guardados y modificados sean servidos y almacenados de forma eficiente mediante el uso de arreglos. Ésto implica que, si el usuario se encuentra atrasado una versión, descargará solamente los archivos que fueron actualizados. Por otro lado, si el usuario difiere en dos o más versiones, será penalizado con la descarga de todos los archivos.

En segunda instancia, se implementará el código necesario para poder reducir el consumo de datos, apoyándose en la propuesta del trabajo mencionado al principio de la sección. La puesta en funcionamiento se basa en un llamado previo a la obtención de los datos a una API desarrollada en C#, que proporciona la fecha de actualización de los datos que se utilizará para comparar con la fecha de guardado de los datos almacenados en la cache. Esta situación plantea dos posibilidades:

- Tener los datos actualizados, por lo tanto, servir los datos de la cache y ahorrar el llamado a la API.
- Tener los datos desactualizados, por lo tanto, servir los datos desde el servidor.

La propuesta adhiere un llamado al servidor, pero se asume que, en la mayoría de los casos, los datos se encontrarán actualizados. Es importante remarcar que, en primera instancia, se deberán cargar los datos desde el servidor. (Vera & Rodríguez, 2020).

4.2 Definición del Esquema

La base del esquema consta en tener una cache con versión 1, definida luego del guión (ver primer renglón de la Figura 10), para que en posteriores implementaciones se aumente el valor de uno en uno. Los arreglos definidos indican los recursos que se deben descargar desde la web, inevitables en el primer caso, y los recursos existentes que se pueden corroborar en la cache con versión N-1, donde N es la versión actual del SW (ver arreglos definidos en la Figura 10).

Figura 10

APP Shell y versión de cache

```
const CACHE_ACTUAL = 'cache-1';
const API_URL_ACTUALIZACION = 'https://localhost:44348/api/DatosActualizacion/';

const paginasModificadas = [
    'clientes.html'
];

const recursosACopiar = [
    '/',
    'css/estilos.css',
    'favicon.ico',
    'img/banner.jpeg',
    'img/no-img.jpg',
    'js/app.js',
    'https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'
];
```

4.3 Rendimiento

Los usuarios usualmente navegan la web para recopilar información, de manera que tener el contenido disponible es fundamental en el armado del sitio. El mismo puede no estar disponible constantemente y por esta razón, el uso de la cache provee herramientas para mejorar este aspecto.

Por otro lado, el tiempo de carga del sitio es sumamente importante dado que el porcentaje de abandono de un sitio está ligado a este tema (Everts, 2013). Realizar un buen uso de la cache brinda la ventaja de optimizar los tiempos de carga, no solo en las primeras visitas, sino también en las subsecuentes.

Por último, se encuentra un tema donde hubo mejoras en los últimos años, mediante la utilización de servicios como Amazon Web Services o Azure, denominado congestión de la red. En esta ocasión, el servidor cumple un rol fundamental y el uso de la cache del lado del cliente brinda la seguridad de poder satisfacer necesidades de consumo, sin importar la cantidad de personas que accedan al sitio web.

4.4 Resumen

En el presente capítulo se describe el almacenamiento en la cache y su mejora de la experiencia del usuario, confiabilidad y rendimiento. La razón principal del aumento del tráfico web se debe a la popularidad de la web en sí misma.

El almacenamiento en la cache intenta mejorar el rendimiento de internet de muchas maneras. En primer lugar, intenta reducir el tiempo de espera de los usuarios, asociado con la obtención de datos. La espera de éstos se puede disminuir porque la cache está más cerca del

cliente que del servidor, que brinda el contenido. En segundo lugar, el almacenamiento en la cache intenta reducir el tráfico de red de los servidores. Por lo tanto, la congestión de la red se puede disminuir ya que las páginas y datos servidos desde la cache no deben ir hasta el servidor. Por último, el almacenamiento en la cache puede reducir la cantidad de solicitudes provenientes de los servidores permitiendo un ahorro no solo en términos de tiempo, sino también monetarios.

La propuesta planteada en el presente capítulo permite una mejora de la experiencia de usuario a través del versionado de la cache y la utilización de un llamado previo al servidor.

Capítulo 5

Aplicación Práctica

5.1 Introducción

Thakur señala que “la aplicación [debe tener] todos los componentes de una PWA: Service Worker, Web App Manifest [and] App Shell.” (pág. 3). En adición, para realizar la tarea del versionado, es necesario un editor de texto, un lenguaje como TypeScript (Javascript tipado), un paquete de nuget capaz de brindar las dependencias de una PWA y una conexión HTTPS. Por otro lado, para reducir el consumo de datos servidos desde el servidor, se requiere un editor de texto -puede ser el mismo usado previamente-, una base de datos y una API Rest desarrollada con un lenguaje que brinde comodidad.

5.2 Implementación

El versionado de la cache se realiza teniendo en cuenta la existencia de una versión anterior a la misma y los recursos que deben ser copiados están definidos en dos arreglos llamados *paginasModificadas* y *recursosACopiar*. Adicionalmente, los archivos usados como ejemplos en las pruebas son descargados de dos formas diferentes dependiendo de la existencia de la cache en el navegador:

- Desde el servidor mediante la interacción con la API Rest y la base de datos, cuando la aplicación se inicie o se presente un aumento de dos versiones o más en la cache.
- Desde la cache, en aquellos casos donde la información de los arreglos no tenga modificaciones.

El código fuente de este procedimiento se realiza en el evento *install* del SW, permitiendo una implementación de una estrategia de la cache que brinda mejoras en la eficiencia del uso de la misma. El proceso se observa en la Figura 11 y se encuentra implementado mediante el uso de código JavaScript.

Figura 11

Evento install del SW

```
self.addEventListener("install", function(event) {

    const cacheSplit = CACHE_ACTUAL.split('-');
    const cacheAnterior = `${cacheSplit[0]}-${(cacheSplit[1] - 1)}`;

    // Pregunto si existe el cache anterior
    const response = caches.has(cacheAnterior).then(async existeCacheAnterior => {
            return existeCacheAnterior;
        })
        .then(existeCacheAnterior => {

            caches.open(CACHE_ACTUAL).then(async cache => {

                // Si existe un cache con version -1
                if (existeCacheAnterior) {

                    // Abrimos el cache anterior y recorremos los recursos a copiar para ver si los encontramos
                    caches.open(cacheAnterior).then(async cacheVersionAnterior => {

                        var recursosNoEncontrados = [];

                        await Promise.all(recursosACopiar.map(async(url) => {
                            // solo analizamos si existe en el cache viejo
                            const response = await cacheVersionAnterior.match(url);
                            if (response) {
                                // Existe el recurso
                                cache.put(url, response);
                            } else {
                                recursosNoEncontrados.push(url);
                                return Promise.resolve();
                            }
                        }));

                        return cache.addAll(recursosNoEncontrados.concat(paginasModificadas));
                    })
                } else {

                    // No existe cache con versión -1 - Por lo que cacheamos todo
                    return cache.addAll(recursosACopiar.concat(paginasModificadas));
                }
            })
        })

    event.waitUntil(response);
});
```

La reducción del consumo de datos se lleva a cabo en la intercepción del evento fetch, en donde se dispondrá a realizar un llamado a la API con conexión a la base de datos para obtener el momento en el que fueron actualizados los datos por última vez. El código fuente necesario se puede ver en la Figura 12.

Figura 12

Evento fetch del SW

```
self.addEventListener('fetch', function(event) {

    const fetchRequest = caches.match(event.request)
        .then(cachedResponse => {

            if (!event.request.url.includes('api')) { return cachedResponse; }

            if (!navigator.onLine) { return cachedResponse; }

            if (!cachedResponse) {

                return caches.open(CACHE_ACTUAL).then(cache => {
                    return fetch(event.request).then(newResponse => {
                        cache.put(event.request, newResponse.clone());
                        return newResponse;
                    });
                });
            }

            const cachedDate = new Date(cachedResponse.headers.get('date'));

            const apiRequested = event.request.url.split('/').pop();

            if (!apiRequested) { return fetch(event.request); }

            return fetch(API_URL_ACTUALIZACION + apiRequested)
                .then(r => r.text())
                .then(response => {

                    // si no existe información guardada en la base (no debería pasar)
                    if (!response) { return cachedResponse; }

                    const fechaActualizacion = new Date(JSON.parse(response));

                    // si la información fue actualizada posteriormente a la fecha del cache, tengo que hacer un fetch y guardarla
                    if (cachedDate < fechaActualizacion) {

                        //Cache with Network Fallback
                        return fetch(event.request).then(newResponse => {
                            caches.open(CACHE_ACTUAL)
                                .then(cache => {
                                    cache.put(event.request, newResponse);
                                });
                            return newResponse.clone();
                        });
                    }
                    return cachedResponse;
                })
        });

    event.respondWith(fetchRequest);
});
```

5.3 Validación

Para poder validar las mejoras en el esquema de recuperación de datos propuesto, se dispusieron los casos de prueba y sus resultados esperados en la tabla 2.

Tabla 2

Descripción de los casos de prueba

Identificación del caso de prueba	**Descripción del caso de prueba**	**Resultados Esperados**
Caso de Prueba 1	Activación de SW y posterior obtención de archivos.	La aplicación debe funcionar estando offline.
Caso de Prueba 2	Agregar archivo con modificaciones e incrementar una vez la versión cache storage.	Descarga de los archivos modificados desde el servidor. Los archivos sin modificaciones deben descargarse de la cache.
Caso de Prueba 3	No realizar cambios en el listado de *paginasModificadas* y añadir 1 versión a la cache storage.	Descarga de todos los archivos desde la cache.
Caso de Prueba 4	Agregar archivo con modificaciones y añadir 2 versiones a la cache storage.	Descarga de todos los archivos desde el servidor.
Caso de Prueba 5	No realizar cambios en el arreglo de *paginasModificadas* y añadir 2 versiones a la cache storage.	Descarga de todos los archivos desde el servidor.
Caso de Prueba 6	Obtener lista de clientes en una primera instancia.	Descarga de listado de clientes desde el servidor.
Caso de Prueba 7	Utilización de fecha de cache menor a fecha de actualización para obtener listado de clientes.	Descarga de listado de clientes desde el servidor.
Caso de Prueba 8	Utilización de fecha de cache mayor a fecha de actualización para obtener listado de clientes.	Descarga de listado de clientes desde la cache.
Caso de Prueba 9	Anular servicio de internet y obtener listado de clientes.	Descarga de listado de clientes desde la cache y cartel de advertencia de datos desactualizados.

Caso de prueba 1 – Activación de SW y posterior obtención de archivos.

Utilizando como base el siguiente nombre y versión de cache: *cache-1* (ver renglón inicial de Figura 13), en conjunto con los recursos (ver ítems de los arreglos de la Figura 13), se disponen las pruebas a realizar.

Figura 13

App Shell de la prueba 1

```
const CACHE_ACTUAL = 'cache-1';
const API_URL_ACTUALIZACION = 'https://localhost:44348/api/DatosActualizacion/';

const paginasModificadas = [
    'clientes.html'
];

const recursosACopiar = [
    '/',
    'css/estilos.css',
    'favicon.ico',
    'img/banner.jpeg',
    'img/no-img.jpg',
    'js/app.js',
    'https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'
];
```

En el primer llamado se observa la descarga de los archivos mencionados en los arreglos paginasModificadas y recursosACopiar (ver Figura 14).

Figura 14

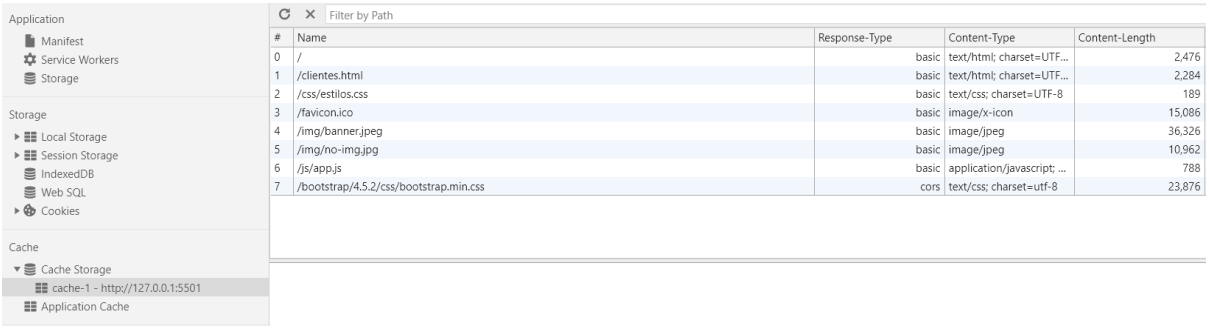
Archivos obtenidos en la primera interacción con el servidor de la prueba 1

Name	Status	Type	Initiator	Size	Time
127.0.0.1	304	document	Other	273 B	3 ms
bootstrap.min.css	200	stylesheet	(index)	(disk cache)	3 ms
estilos.css	304	stylesheet	(index)	272 B	3 ms
app.js	304	script	(index)	273 B	3 ms
banner.jpeg	304	jpeg	(index)	274 B	2 ms
ws	101	websocket	(index):59	0 B	Pending
favicon.ico	200	x-icon	Other	15.4 kB	2 ms
⚙ sw-ejemplo.js	304	javascript	Other	0 B	4 ms
⚙ 127.0.0.1	304	fetch	sw-ejemplo.js:56	273 B	3 ms
⚙ estilos.css	304	fetch	sw-ejemplo.js:56	272 B	4 ms
⚙ favicon.ico	304	fetch	sw-ejemplo.js:56	274 B	4 ms
⚙ banner.jpeg	304	fetch	sw-ejemplo.js:56	274 B	5 ms
⚙ no-img.jpg	304	fetch	sw-ejemplo.js:56	274 B	5 ms
⚙ app.js	304	fetch	sw-ejemplo.js:56	273 B	5 ms
⚙ bootstrap.min.css	200	fetch	sw-ejemplo.js:56	(disk cache)	3 ms
⚙ clientes.html	304	fetch	sw-ejemplo.js:56	273 B	5 ms

La cache storage *cache-1* de la Figura 15 queda definida con la totalidad de los archivos definidos en el App Shell de la Figura 13.

Figura 15

Cache versión 58



#	Name	Response-Type	Content-Type	Content-Length
0	/	basic	text/html; charset=UTF...	2,476
1	/clientes.html	basic	text/html; charset=UTF...	2,284
2	/css/estilos.css	basic	text/css; charset=UTF-8	189
3	/favicon.ico	basic	image/x-icon	15,086
4	/img/banner.jpeg	basic	image/jpeg	36,326
5	/img/no-img.jpg	basic	image/jpeg	10,962
6	/js/app.js	basic	application/javascript; ...	788
7	/bootstrap/4.5.2/css/bootstrap.min.css	cors	text/css; charset=utf-8	23,876

Al recargar la página se verifica que los llamados son servidos desde el service worker (ver Figura 16).

Figura 16

Llamado con archivos en cache versión 58

Name	Status	Type	Initiator	Size	Time
127.0.0.1	200	document	Other	(ServiceWorker)	9 ms
bootstrap.min.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
estilos.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
app.js	200	script	(index)	(ServiceWorker)	3 ms
banner.jpeg	200	jpeg	(index)	(ServiceWorker)	3 ms
ws	101	websocket	(index):59	0 B	Pending
favicon.ico	200	x-icon	Other	(ServiceWorker)	1 ms

Al dejar offline la página, la misma sigue funcionando con normalidad (ver Figura 17 y Figura 18).

Figura 17

Funcionamiento offline de la home

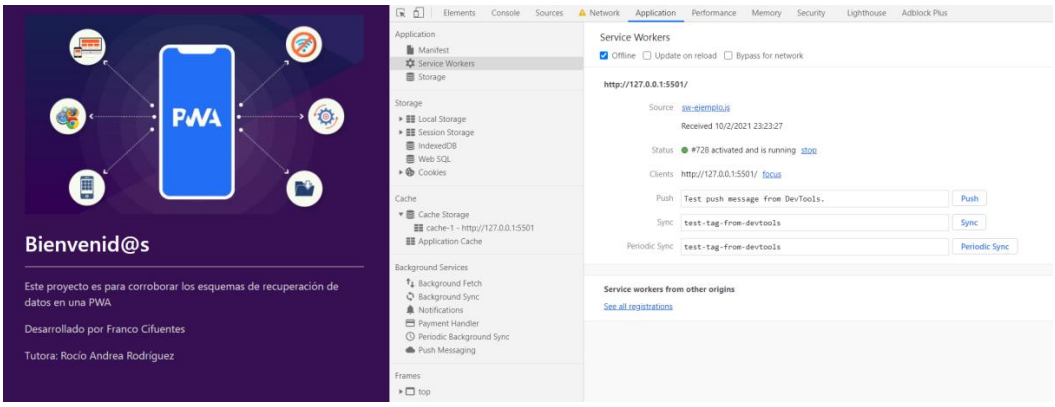
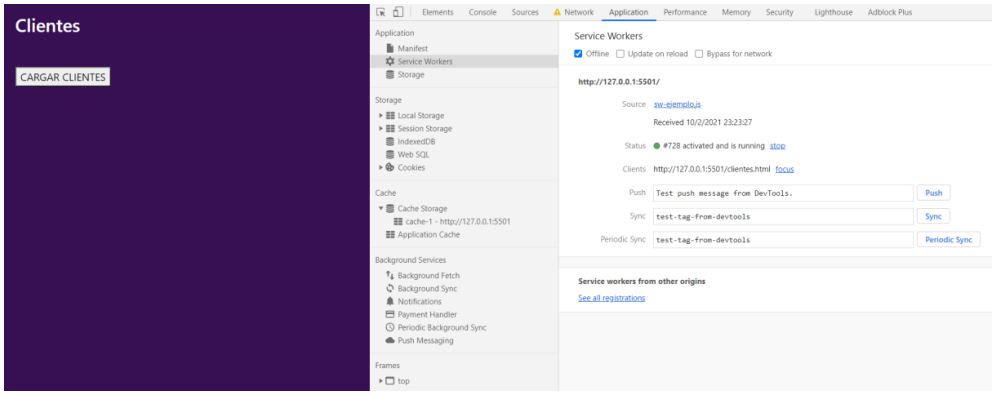


Figura 18

Funcionamiento offline de página interna clientes.html



Caso de prueba 2 – Agregar archivo con modificaciones e incrementar una vez la versión cache storage.

Previamente se tiene la cache con versión 1 y el arreglo paginasModificadas contiene un archivo nuevo: *about.html* (ver Figura 19). Para la realización de esta prueba se aumenta la versión en 1 y se agrega el archivo *clientes.html* sin modificaciones en el arreglo de recursosACopiar (ver último elemento del segundo arreglo de la Figura 19).

Figura 19

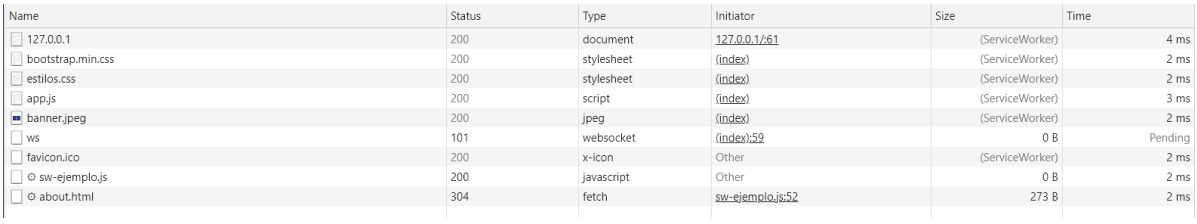
App Shell de la prueba 2

```
const CACHE_ACTUAL = 'cache-2';
const API_URL_ACTUALIZACION = 'https://localhost:44348/api/DatosActualizacion/';

const paginasModificadas = [
    'about.html'
];

const recursosACopiar = [
    '/',
    'css/estilos.css',
    'favicon.ico',
    'img/banner.jpeg',
    'img/no-img.jpg',
    'js/app.js',
    'https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css',
    'clientes.html'
];
```

Se determina que el archivo *about.html* fue descargado desde la web y el resto de los recursos fueron servidos desde el service worker (ver Figura 20).

Figura 20

Archivos servidos en el llamado de la prueba 2

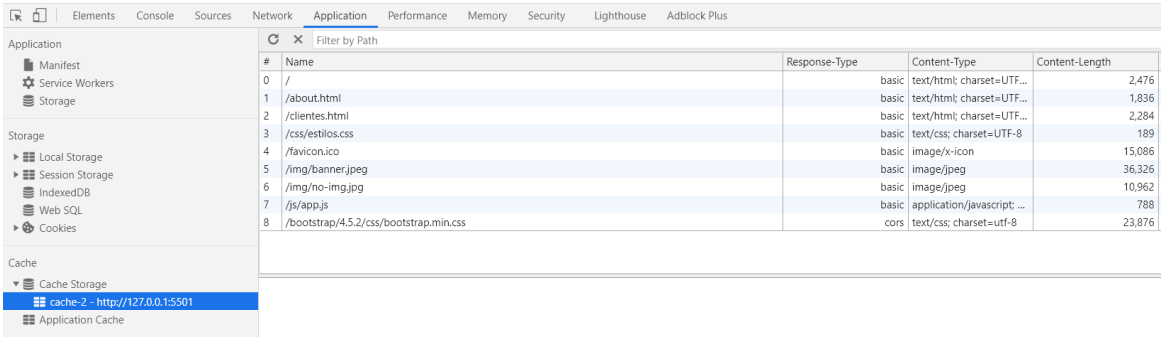


Name	Status	Type	Initiator	Size	Time
127.0.0.1	200	document	127.0.0.1/:61	(ServiceWorker)	4 ms
bootstrap.min.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
estilos.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
app.js	200	script	(index)	(ServiceWorker)	3 ms
banner.jpeg	200	jpeg	(index)	(ServiceWorker)	2 ms
ws	101	websocket	(index):59	0 B	Pending
favicon.ico	200	x-icon	Other	(ServiceWorker)	2 ms
⚙ sw-ejemplo.js	200	javascript	Other	0 B	2 ms
⚙ about.html	304	fetch	sw-ejemplo.js:52	273 B	2 ms

La cache storage *cache-2* de la Figura 21 muestra la totalidad de los archivos definidos en el App Shell de la Figura 17.

Figura 21

Cache versión 2



Caso de prueba 3 – No realizar cambios en el listado de *paginasModificadas* y añadir 1 versión a la cache storage.

Se dispone a aumentar la cache una versión (ver primer renglón de la Figura 18) y no realizar modificaciones en los arreglos (ver Figura 18) para confirmar que los elementos del arreglo paginasModificadas son descargados mientras que los archivos del arreglo recursosACopiar son servidos desde la cache con versión 2.

Figura 22

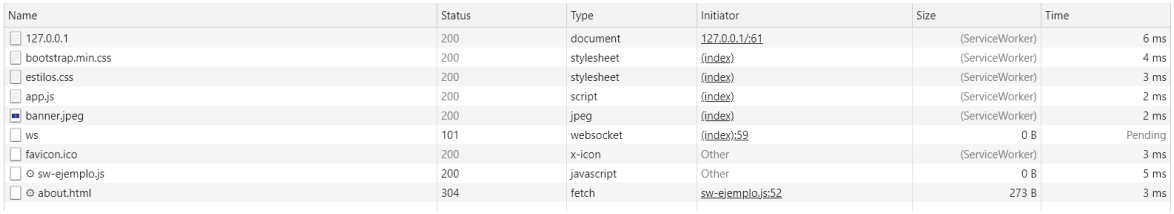
App Shell de la prueba 3

```
const CACHE_ACTUAL = 'cache-3';
const API_URL_ACTUALIZACION = 'https://localhost:44348/api/DatosActualizacion/';

const paginasModificadas = [
    'about.html'
];

const recursosACopiar = [
    '/',
    'css/estilos.css',
    'favicon.ico',
    'img/banner.jpeg',
    'img/no-img.jpg',
    'js/app.js',
    'https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css',
    'clientes.html'
];
```

En la Figura 23, se observa que los archivos son servidos por el service worker, a excepción del archivo *about.html* que vuelve a ser descargado por estar en el arreglo de paginasModificadas (ver Figura 22).

Figura 23

Archivos servidos en el llamado de la prueba 3

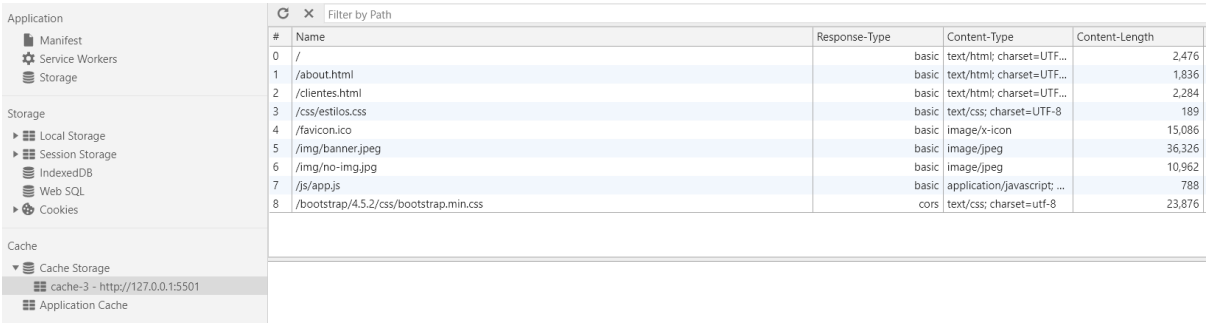


Name	Status	Type	Initiator	Size	Time
127.0.0.1	200	document	127.0.0.1/:61	(ServiceWorker)	6 ms
bootstrap.min.css	200	stylesheet	(index)	(ServiceWorker)	4 ms
estilos.css	200	stylesheet	(index)	(ServiceWorker)	3 ms
app.js	200	script	(index)	(ServiceWorker)	2 ms
banner.jpeg	200	jpeg	(index)	(ServiceWorker)	2 ms
ws	101	websocket	(index):59	0 B	Pending
favicon.ico	200	x-icon	Other	(ServiceWorker)	3 ms
⚙ sw-ejemplo.js	200	javascript	Other	0 B	5 ms
⚙ about.html	304	fetch	sw-ejemplo.js:52	273 B	3 ms

La cache storage *cache-2* de la Figura 24 muestra la totalidad de los archivos definidos en el App Shell de la Figura 22.

Figura 24

Cache versión 3



#	Name	Response-Type	Content-Type	Content-Length
0	/	basic	text/html; charset=UTF...	2,476
1	/about.html	basic	text/html; charset=UTF...	1,836
2	/clientes.html	basic	text/html; charset=UTF...	2,284
3	/css/estilos.css	basic	text/css; charset=UTF-8	189
4	/favicon.ico	basic	image/x-icon	15,086
5	/img/banner.jpeg	basic	image/jpeg	36,326
6	/img/no-img.jpg	basic	image/jpeg	10,962
7	/js/app.js	basic	application/javascript; ...	788
8	/bootstrap/4.5.2/css/bootstrap.min.css	cors	text/css; charset=utf-8	23,876

Caso de prueba 4 – Agregar archivo con modificaciones y añadir 2 versiones a la cache storage.

Se decide aumentar la cache dos versiones (ver primer renglón de la Figura 25) y agregar un nuevo archivo HTML (ver primer arreglo de la Figura 25) para corroborar la descarga total de los archivos de ambos arreglos.

Figura 25

App Shell de la prueba 4

```
const CACHE_ACTUAL = 'cache-5';
const API_URL_ACTUALIZACION = 'https://localhost:44348/api/DatosActualizacion/';

const paginasModificadas = [
    'faq.html'
];

const recursosACopiar = [
    '/',
    'css/estilos.css',
    'favicon.ico',
    'img/banner.jpeg',
    'img/no-img.jpg',
    'js/app.js',
    'https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css',
    'clientes.html',
    'about.html'
];
```

En la Figura 26, se observa que la totalidad de los archivos se descargan desde la web debido al aumento de dos versiones definido en la Figura 25.

Figura 26

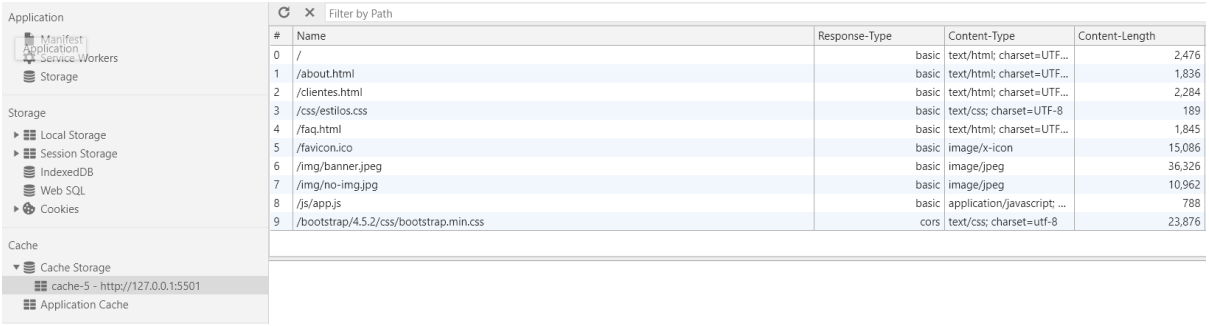
Archivos servidos en el llamado de la prueba 4

Name	Status	Type	Initiator	Size	Time
127.0.0.1	200	document	127.0.0.1/:61	(ServiceWorker)	5 ms
bootstrap.min.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
estilos.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
app.js	200	script	(index)	(ServiceWorker)	2 ms
banner.jpeg	200	jpeg	(index)	(ServiceWorker)	2 ms
ws	101	websocket	(index):59	0 B	Pending
favicon.ico	200	x-icon	Other	(ServiceWorker)	1 ms
⚙ sw-ejemplo.js	200	javascript	Other	0 B	4 ms
⚙ favicon.ico	304	fetch	sw-ejemplo.js:58	274 B	7 ms
⚙ banner.jpeg	304	fetch	sw-ejemplo.js:58	274 B	7 ms
⚙ no-img.jpg	304	fetch	sw-ejemplo.js:58	274 B	9 ms
⚙ app.js	304	fetch	sw-ejemplo.js:58	273 B	7 ms
⚙ bootstrap.min.css	200	fetch	sw-ejemplo.js:58	(disk cache)	6 ms
⚙ clientes.html	304	fetch	sw-ejemplo.js:58	273 B	8 ms
⚙ about.html	304	fetch	sw-ejemplo.js:58	273 B	8 ms
⚙ faq.html	200	fetch	sw-ejemplo.js:58	2.2 kB	8 ms
⚙ 127.0.0.1	304	fetch	sw-ejemplo.js:58	273 B	4 ms
⚙ estilos.css	304	fetch	sw-ejemplo.js:58	272 B	6 ms

La cache storage *cache-5* de la Figura 27 muestra la totalidad de los archivos definidos en el App Shell de la Figura 25.

Figura 27

Cache versión 5



Caso de prueba 5 – No realizar cambios en el arreglo de *paginasModificadas* y añadir 2 versiones a la cache storage.

En este caso, se dispone a aumentar dos versiones en la cache (ver primer renglón de la Figura 28) y no realizar ningún cambio en ninguna página. Por lo tanto, se tiene el arreglo de paginasModificadas sin ningún elemento (ver primer arreglo de la Figura 28).

Figura 28

App Shell de la prueba 5

```
const CACHE_ACTUAL = 'cache-7';
const API_URL_ACTUALIZACION = 'https://localhost:44348/api/DatosActualizacion/';

const paginasModificadas = [
];

const recursosACopiar = [
    '/',
    'css/estilos.css',
    'favicon.ico',
    'img/banner.jpeg',
    'img/no-img.jpg',
    'js/app.js',
    'https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css',
    'clientes.html',
    'about.html',
    'faq.html'
];
```

Se verifica la descarga de todos los archivos debido al salto de dos versiones de la cache (ver Figura 29).

Figura 29

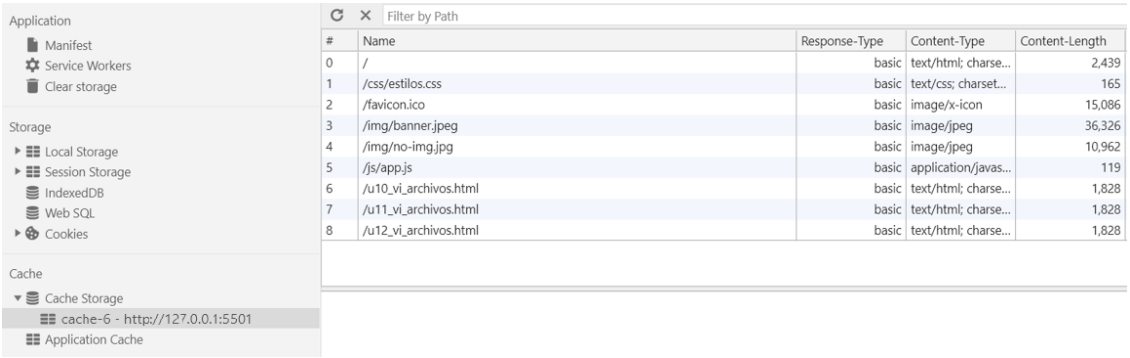
Archivos servidos en el llamado de la prueba 5

Name	Status	Type	Initiator	Size	Time
127.0.0.1	200	document	127.0.0.1/:61	(ServiceWorker)	3 ms
bootstrap.min.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
estilos.css	200	stylesheet	(index)	(ServiceWorker)	2 ms
app.js	200	script	(index)	(ServiceWorker)	4 ms
banner.jpeg	200	jpeg	(index)	(ServiceWorker)	2 ms
ws	101	websocket	(index):59	0 B	Pending
favicon.ico	200	x-icon	Other	(ServiceWorker)	2 ms
⚙ sw-ejemplo.js	200	javascript	Other	0 B	5 ms
⚙ 127.0.0.1	304	fetch	sw-ejemplo.js:57	273 B	11 ms
⚙ estilos.css	304	fetch	sw-ejemplo.js:57	272 B	9 ms
⚙ favicon.ico	304	fetch	sw-ejemplo.js:57	274 B	10 ms
⚙ banner.jpeg	304	fetch	sw-ejemplo.js:57	274 B	12 ms
⚙ no-img.jpg	304	fetch	sw-ejemplo.js:57	274 B	10 ms
⚙ app.js	304	fetch	sw-ejemplo.js:57	273 B	10 ms
⚙ bootstrap.min.css	200	fetch	sw-ejemplo.js:57	(disk cache)	9 ms
⚙ clientes.html	304	fetch	sw-ejemplo.js:57	273 B	10 ms
⚙ about.html	304	fetch	sw-ejemplo.js:57	273 B	11 ms
⚙ faq.html	304	fetch	sw-ejemplo.js:57	273 B	11 ms

La cache storage *cache-7* de la Figura 30 muestra la totalidad de los archivos definidos en el App Shell de la Figura 28.

Figura 30

Cache versión 7

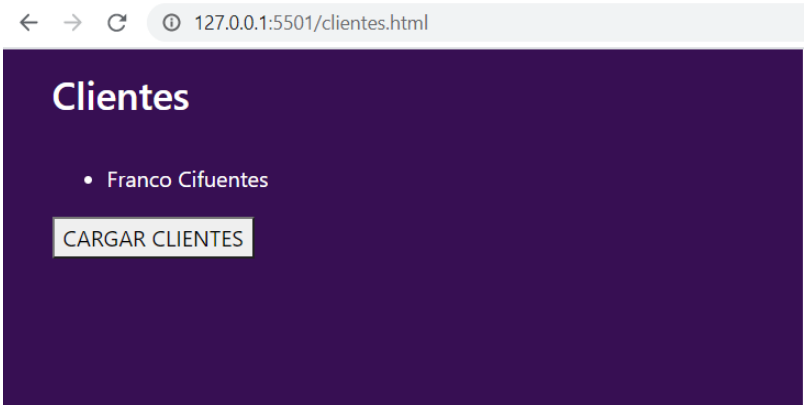


Caso de prueba 6 – Obtener lista de clientes en una primera instancia.

Para poder realizar esta prueba se plantea la obtención del listado de clientes mediante el evento *clic* dentro del archivo *clientes.html*. En la Figura 31, se contempla el botón implementado con la leyenda “Cargar Clientes”.

Figura 31

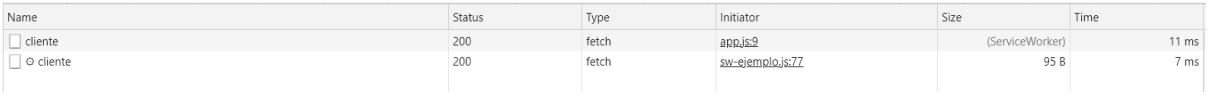
Cliente.html y botón de carga de clientes



El llamado al servidor observado en la Figura 32 destaca la intercepción del evento fetch del service worker.

Figura 32

Evento fetch de los clientes

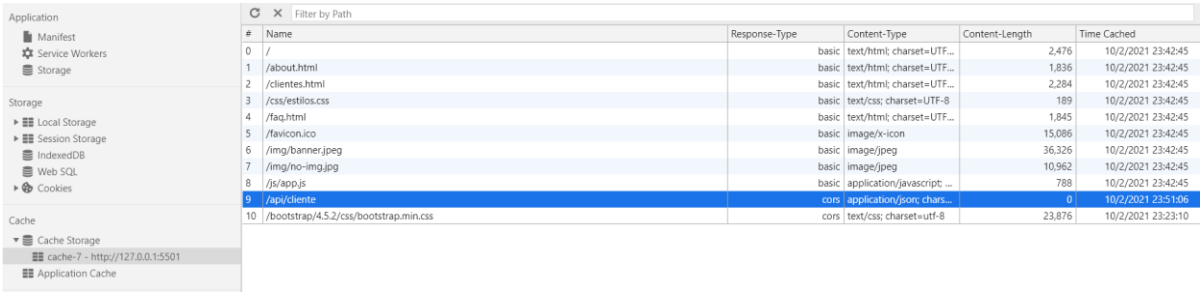


Name	Status	Type	Initiator	Size	Time
cliente	200	fetch	app.js:9	(ServiceWorker)	11 ms
⚙ cliente	200	fetch	sw-ejemplo.js:77	95 B	7 ms

Por último, es importante tener en cuenta, para las pruebas posteriores, el guardado de la información en la cache storage y la fecha en la columna *Time Cached* (ver renglón resaltado en la Figura 33).

Figura 33

Fecha de datos guardados en la cache storage de la prueba 6



#	Name	Response-Type	Content-Type	Content-Length	Time Cached
0	/	basic	text/html; charset=UTF...	2,476	10/2/2021 23:42:45
1	/about.html	basic	text/html; charset=UTF...	1,836	10/2/2021 23:42:45
2	/clientes.html	basic	text/html; charset=UTF...	2,284	10/2/2021 23:42:45
3	/css/estilos.css	basic	text/css; charset=UTF-8	189	10/2/2021 23:42:45
4	/faq.html	basic	text/html; charset=UTF...	1,845	10/2/2021 23:42:45
5	/favicon.ico	basic	image/x-icon	15,086	10/2/2021 23:42:45
6	/img/banner.jpeg	basic	image/jpeg	36,326	10/2/2021 23:42:45
7	/img/no-img.jpg	basic	image/jpeg	10,962	10/2/2021 23:42:45
8	/js/app.js	basic	application/javascript; ...	788	10/2/2021 23:42:45
9	/api/cliente	cors	application/json; chars...	0	10/2/2021 23:51:06
10	/bootstrap/4.5.2/css/bootstrap.min.css	cors	text/css; charset=utf-8	23,876	10/2/2021 23:23:10

Caso de prueba 7 – Utilización de fecha de cache menor a fecha de actualización para obtener listado de clientes.

Para el análisis de este caso acontece tener en cuenta las siguientes variables: la fecha del guardado de la información en la cache storage (ver Figura 34) y la fecha de actualización de los datos de la base de datos (ver Figura 35).

Figura 34

Fecha de guardado de los datos en cache

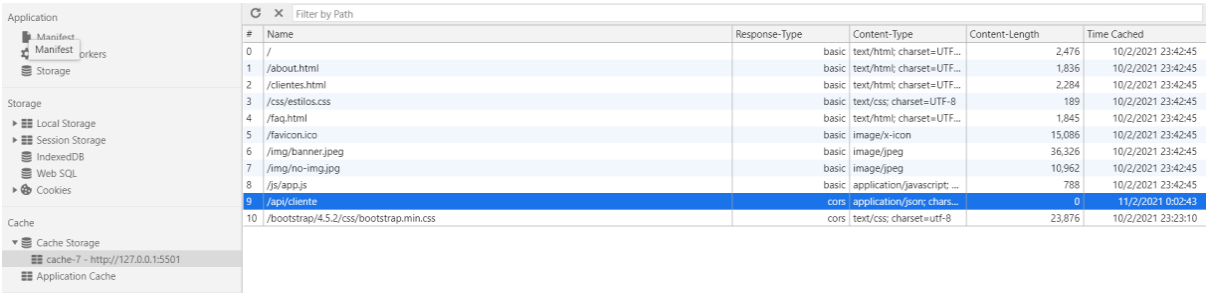


Figura 35

Fecha de actualización de los datos cliente en la base de datos

	Id	DatosNombre	FechaActualizacion
	1	cliente	2021-02-10 23:55:00.000
▶*	NULL	NULL	NULL

Al hacer *clic* en el botón de cargar clientes, se generan dos llamados a la API. En un primer momento, como se menciona en el apartado 5.2, debe realizarse una consulta a la base de datos utilizando la fecha de actualización de los datos. La fecha definida en la Figura 36 coincide con la fecha de la Figura 35 y, al comparar la fecha de la Figura 34 y la Figura 35, se concluye que: como la primera es menor a la segunda, se debe ejecutar un llamado al servidor para traer el listado de clientes debido a la existencia de modificaciones (ver Figura 37).

Figura 36

Fecha de actualización obtenida desde la API

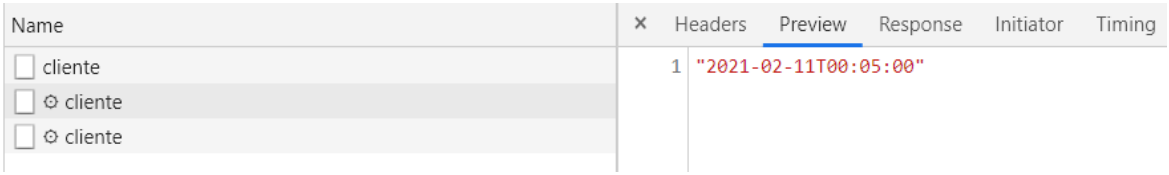
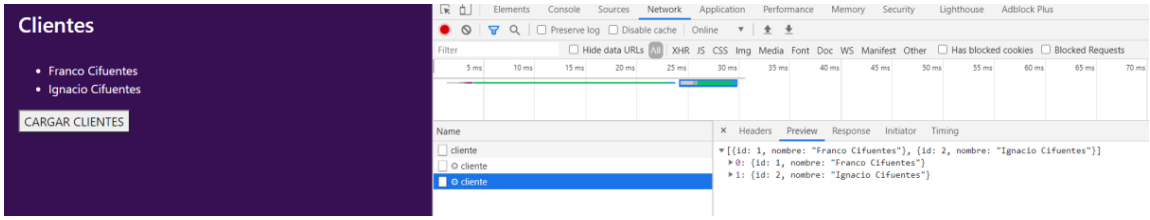


Figura 37

Respuesta del llamado a la API Clientes



Caso de prueba 8 – Utilización de fecha de cache mayor a fecha de actualización para obtener listado de clientes.

En este caso, al igual que el anterior, acontece tener en cuenta las siguientes variables: la fecha del guardado de la información en la cache storage (ver Figura 38) y la fecha de actualización de los datos de la base de datos (ver Figura 39).

Figura 38

Fecha de guardado de datos en cache

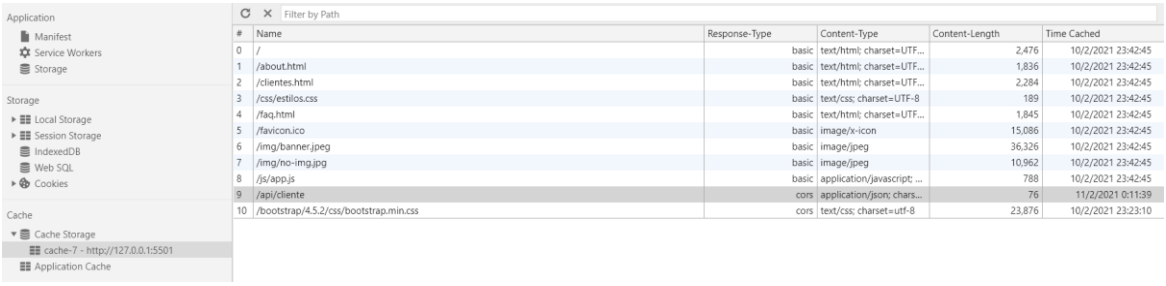


Figura 39

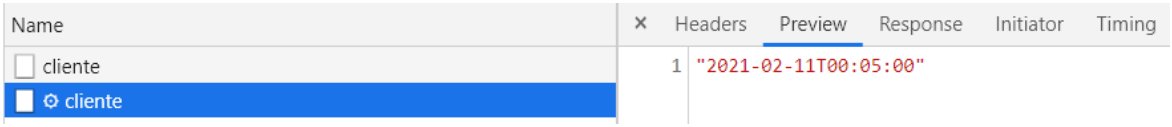
Fecha de actualización de los datos cliente en la base de datos

	Id	DatosNombre	FechaActualizacion
	1	cliente	2021-02-11 00:05:00.000
▶*	NULL	NULL	NULL

Al hacer *clic* en el botón de cargar clientes, la conexión a la API genera un solo llamado en comparación con los dos llamados del caso anterior (ver Figura 40). Esto sucede porque al comparar la fecha obtenida en el llamado a la API de la Figura 40, se denota que es menor a la fecha de guardado de los datos en la cache storage de la Figura 38. Se interpreta que los datos en cache fueron solicitados luego de la última actualización y, por lo tanto, se encuentran actualizados.

Figura 40

Fecha de actualización obtenida desde la API

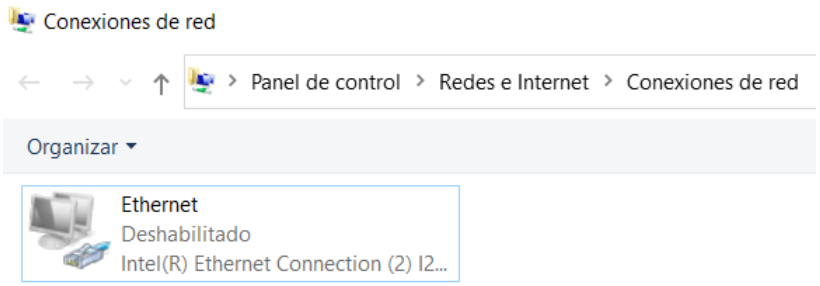


Caso de prueba 9 – Anular servicio de internet y obtener listado de clientes.

En este caso es necesario suspender el servicio de internet de manera que las solicitudes nunca lleguen al servidor. Para realizar esta tarea se debe deshabilitar el adaptador de red como se observa en la Figura 41.

Figura 41

Desconexión de internet



La carga de clientes es realizada desde el botón “Cargar clientes” en la web y se aprecia que los datos guardados en cache de la Figura 42 son presentados sin ningún impedimento.

El mensaje en rojo “Los datos pueden estar desactualizados” de la Figura 42 implica que no se pudo realizar ningún llamado a la API para verificar la última fecha de actualización de los datos (ver Figura 43).

Figura 42

Datos y cartel al estar offline

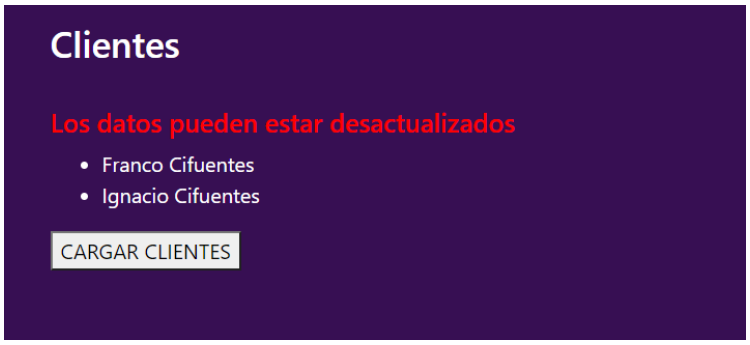
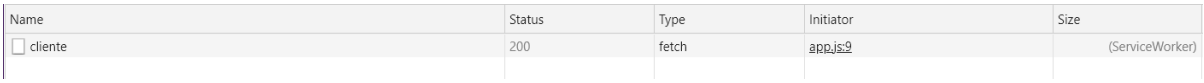


Figura 43

Evento fetch interceptado y servido desde el SW



5.4 Resumen

En este capítulo se ha trabajado con casos de prueba offline y online, versionado y el uso de modificaciones a páginas web. En todos ellos se ha visto el correcto funcionamiento y una mejora del rendimiento en dos aspectos: el ahorro de consumo de datos y la mayor velocidad de carga.

En esta implementación hay que remarcar dos aspectos relevantes. En primer lugar, las pruebas contienen una secuencialidad y esto no es un requerimiento. En segundo lugar, la implementación está dada del lado del cliente mediante un sitio web, sin embargo, la importancia del código está del lado del servidor mediante el uso de una API con acceso a la base datos.

Las pruebas dejan en evidencia cómo lograr una mejora en el rendimiento de un sitio web mediante el desarrollo de una arquitectura orientada al uso eficiente de la cache.

Conclusiones

El mundo de las PWA llegó para quedarse. La web móvil se asemeja significativamente a las aplicaciones instalables descargadas desde la tienda y ésto sucede debido a la configuración del archivo *manifest*, instalación en la pantalla inicial, trabajo en segundo plano del *service worker* y mayor velocidad de carga. La metodología aplicada en este trabajo se desarrolla sobre estas dos últimas características, mediante el diseño e implementación de una solución que contempla el manejo de versionado y el uso de fechas de actualización para mejorar el rendimiento de las PWA.

Con la implementación presentada en este trabajo final de carrera, quedan validadas las hipótesis planteadas. Mediante el prototipo y las pruebas implementadas en el capítulo previo es posible decir que el uso de SW permite implementar estrategias de almacenamiento en la cache en PWA. Adicionalmente, la implementación de versionado de código, esquemas de recuperación y llamada al servidor otorga mejoras en el rendimiento de aplicaciones web debido a que se evita descargar todo el contenido de la red en cada llamado al servidor.

Cuando se habla de versionado, su utilización se asocia a la realización del seguimiento de código en los proyectos. Sin embargo, en los casos pruebas 2, 3, 4 y 5, el versionado se empleó para dar seguimiento a los datos que eran recibidos en cada llamada al servidor. Esta funcionalidad es sumamente provechosa para el usuario debido al ahorro de datos y la obtención de una mayor velocidad de carga de la web, sin tener noción de lo que está sucediendo internamente.

En los casos de prueba 6, 7 y 8 se realiza una consulta al servidor para obtener la fecha de actualización de los datos que posteriormente van a ser consultados. Ésto puede resultar contraproducente por realizar un llamado extra, sin embargo, este llamado previo permite definir si la información debe ser servida desde la base de datos, o si la almacenada en la cache es suficiente para ser brindada al usuario.

Desde un punto de vista, se encuentra el versionado de la cache y desde otro, la fecha de actualización y la interacción con el servidor. Frecuentemente, no existe conexión a internet para realizar consultas y manejos de recursos, por lo que es importante recordar la funcionalidad offline de las PWA. Para tener en cuenta la misma, se plantearon los casos de pruebas 1 y 9. En éstos se ha observado que un mensaje al usuario es suficiente para poder brindar una navegabilidad con la ausencia de conexión a internet.

La mejora planteada está orientada a una optimización de la experiencia de usuario y al rendimiento, considerando que en la actualidad es conveniente para captar más clientes. Si

bien para el desarrollo fue necesaria la creación de una API para poder interactuar con la base de datos, el núcleo de la investigación está centrado en las líneas de código del lado del *frontend* utilizando código JavaScript. Por esta razón, sería recomendable que futuras investigaciones abordasen el desarrollo de arquitecturas de PWA del lado del *backend*.

Líneas Futuras de Investigación

El análisis del presente trabajo se ha centrado en un esquema de recuperación de datos del lado del cliente. De forma complementaria, existe la posibilidad de enfocarse en una arquitectura basada del lado del servidor llamada Server Side Rendering (SSR). Esta técnica es utilizada para representar una Single Page Application (SPA) y era la forma tradicional de servir una web. Desafortunadamente, a medida que llegaron los framework con JavaScript, se ha ido diluyendo, pero nunca desapareció. Por esta razón, se propone investigar sobre SSR teniendo en cuenta sus características más valiosas:

- Velocidad de carga: al tener las páginas renderizadas en la respuesta inicial, la aplicación carga con mayor velocidad, permitiendo a los usuarios ver la respuesta rápidamente y proporcionar una mejor experiencia.
- Rendimiento del SEO: se puede rastrear según el contenido de la página y los motores de búsqueda pueden indexarla adecuadamente debido a que se tiene una página estática.

La mejor práctica es combinar SSR y CSR tanto como sea posible para que los datos provenientes del servidor se utilicen para representar la página en el lado del servidor en una primera instancia. El objetivo es que cuando el cliente recupere la página, el service worker almacene en cache todo el contenido necesario. Posteriormente, el service worker consultará datos dinámicos en el servidor y volverá a renderizar del lado del cliente. En esencia, la utilización de SSR puede cargar rápidamente la página en primera instancia y, después de la carga inicial del sitio web, el cliente puede optar por utilizar sólo las partes que deben actualizarse para volver a renderizar la página.

Acrónimos

API ***Application Programming Interface***

Método que permite la comunicación entre dos programas de software.

CAETI ***Centro de Altos Estudios en Tecnología Informática***

Instituto dedicado a la investigación de las tecnologías informáticas de comunicación.

CSR ***Client Side Rendering***

Enfoque de renderizado de un sitio web del lado del cliente.

CSS ***Cascade Style Sheets***

Definición de estilos a ser utilizados en un documento estructurado.

DOM ***Document Object Model***

Definición de la estructura lógica, manipulación y accesibilidad de un documento.

HTML ***HyperText Markup Language***

Lenguaje de marcado para páginas.

HTTPS ***Hypertext Transfer Protocol Secure***

Transferencia segura de datos mediante la utilización del protocolo HTTP.

IDE ***Integrated Development Environment***

Aplicación que provee herramientas y servicios al programador durante la creación de software.

PWA ***Progressive Web App***

Tipo de software de aplicación que se entrega a través de la web.

SEO ***Search Engine Optimization***

Método de obtención de ranking elevado en buscadores de internet de aplicaciones web.

SSR ***Server Side Rendering***

Enfoque de renderizado de un sitio web del lado del servidor.

SPA ***Single Page Application***

Web de una sola página enfocada en dar una experiencia de usuario fluida.

SW ***Service Worker***

JavaScript ejecutado por el navegador en segundo plano.

SWIF ***Service Worker Infusion Framework***

Framework basado en services workers para mejoras de rendimiento de navegación.

URL ***Uniform Resource Locator***

Dirección de un recurso único en la web.

W3C ***World Wide Web Consortium***

Grupo de desarrolladores que recomiendan estándares para los desarrollos de aplicaciones web.

Referencias

Alterxsoft. (10 de Mayo de 2019). https://www.altexsoft.com/blog/engineering/progressive-web-apps/

Archibald, J. (28 de Septiembre de 2020). *The Offline Cookbook*. https://web.dev/offline-cookbook/

Beezer. (14 de Julio de 2020). *Beezer*. https://www.beezer.com/blog/important-pwa-statistics/

Bhilare, A., Gaikwad, Y., Kokare, V., Kumbhar, S., & Ekatpure, J. (2019). *Progressive Web App (PWA) for Organization.* Indapur: International Journal for Research in Applied Science & Engineering Technology (IJRASET).

Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T.-M. (2017). *Progressive Web Apps: The Possible Web-native Unifier for Mobile.* Kristiansand.

Chi-Tsai, Y., Tzuo-Ming, C., & Ming-Chih, C. (2019). *A Progressive Web App for Evaluating Occupation Indicator.* Department of Information Management, Shih-Chien University, Taiwan.

Clarke, R. (1999). *Fundamentals of Information Systems.*

Everts, T. (01 de Mayo de 2013). *Case Study: How a 2-Second Improvement in Page Load Time More Than Doubled Conversions*. https://blog.radware.com/applicationdelivery/applicationaccelerationoptimization/2013/05/case-study-page-load-time-conversions/

Gallagher, N. (6 de Abril de 2017). *How we built Twitter Lite*. https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html

Gambhir, A., & Raj, G. (2018). *Analysis of Cache in Service Worker and Performance Scoring of PWA.*

Google. (25 de Octubre de 2017). *Tools for Web Developers*. https://developers.google.com/web/tools/lighthouse?hl=es

Google Developers. (05 de Enero de 2019). *PWA Training - Google Developers*. https://developers.google.com/web/ilt/pwa

Ilyukha, V. (01 de Septiembre de 2020). *Jelvix*. https://jelvix.com/blog/pwa-vs-native-app-benefits-for-users-and-developers

Jiyeon Lee, H. K. (2018). *Pride and Prejudice in PWA: Abusing Native App-like Features in Web Applications.*

Kaplan, M. (27 de Abril de 2020). *Inmobi*. https://www.inmobi.com/blog/2020/04/27/mobile-app-vs-website-statistics-how-app-usage-compares-to-mobile-web

Liu, X., Ma, Y., Dong, S., Liu, Y., Xie, T., & Huang, G. (2018). *ReWAP: Reducing Redundant Transfers for Mobile Web Browsing via App-Specific Resource Packaging.*

Malavolta, I., Chinnappan, K., Jasmontas, L., Gupta, S., & Ali Karam Soltany, K. (2020). *Evaluating the Impact of Caching on the Energy Consumption.* Amsterdam.

Mozilla Developers. (13 de Agosto de 2021). *Introducción a aplicaciones web progresivas*. https://developer.mozilla.org/es/docs/Web/Progressive_web_apps/Introduction

Osmani, A. (29 de Noviembre de 2017). *Pinterest Progressive Web App Performance*. https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154

Osmani, A. (24 de Diciembre de 2017). *Tinder Progressive Web App Performance*. https://medium.com/@addyosmani/a-tinder-progressive-web-app-performance-case-study-78919d98ece0

Padmanabhan, A. (07 de Mayo de 2020). *Devopedia*. https://devopedia.org/progressive-web-applications

Pande, N., Somani, A., Prasad Samal, S., & Kakkirala, V. (2018). *Enhanced Web Application and Browsing performance through Service-worker Infusion Framework.*

Quynh, H. P. (2018). *Progressive Web App – a new trend in e-commerce.*

Reynolds, G. W., & Stairs, R. M. (2014). *Fundamentals of Information Systems.* Course Technology.

Ribeiro Neto, B., & Baeza Yates, R. (1999). *Recuperación de información moderna.*

Rodríguez, R. A., Vera, P. M., Martínez, M. R., Alderete, C., & Dogliotti, M. (2020). Aplicaciones web progresivas enfocadas en el uso y optimización de cache. El Calafate, Santa Cruz.

Rodríguez, R. A., Vera, P. M., Vallés, F., M. M., & & Giulianelli, D. A. (2014). *Analysis of current and future web standards for reducing the gap between native and web applications.* Buenos Aires.

Sayal, T., & Abhishek, J. (2018). *Impact of Progressive Web Apps on Web App Development.*

Sheppard, D. (2017). *Beginning Progressive Web App Development.* Tinley Park, Illinois, USA: Apress.

Singla, A., Chandrasekaran, B., Brighten Godfrey, P., & Maggs, B. (2014). *The Internet at the Speed of Light.*

Starbucks. (2015). *Starbucks Progressive Web App*. https://formidable.com/work/starbucks-progressive-web-app/

Thakur, P. (2018). *Evaluation and Implementation of Progressive.* Helsinki.

Vera, P. M., & Rodríguez, R. A. (2020). *Estrategias de Manejo de Cache para Aplicaciones Web Progresivas.* Congreso Nacional de Ingeniería en Informática y Sistemas de Información (CONAIISI).

Vorderer, P., & Klimmt, C. (2020). *The Mobile User's Mindset in a Permanently Online, Permanently Connected Society.*

Yberg, V. (2018). *Native-like Performance and User.* Stockholm, Sweden.

Yuliawan, D., Widyandana, D., & Nur Hidayah, R. (2020). *Development and Utilization of "NEPWA" Media in Education and Health Promotion Course Using Gagne's Model of Instructional Design on Nursing Students: Research and Development.*