ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Aplicación Web Progresiva (PWA) para la gestión de pagos de estacionamiento en superficie
(Manage Street parking payments using Progressive Web Application (PWA))

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Esteban Lanza Ortega

Octubre - 2019

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Esteban Lanza Ortega

Director del TFG: Jorge Lanza Calderón

Título: "Aplicación Web Progresiva (PWA) para la gestión de pagos de estacionamiento en superficie"

Title: "Manage Street parking payments using Progressive Web Applications (PWA) "

Presentado a examen el día: 24 de Octubre de 2019

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:

Presidente (Apellidos, Nombre): Pilar Bernardos Llorente

Secretario (Apellidos, Nombre): Jorge Lanza Calderón

Vocal (Apellidos, Nombre): Alberto Eloy García Gutiérrez

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

Me gustaría agradecerles a todos mis profesores por haberme formado a lo largo de mi etapa universitaria y especialmente a mi director del TFG, Jorge Lanza Calderón por haberme ayudado a hacer todo esto posible.

Por último, a mi familia hacia quienes solo puedo expresar mi más sincero agradecimiento por su apoyo incondicional.

Resumen

La tecnología no es algo estático, es una entidad que nace se desarrolla y en un momento dado solo le queda evolucionar o desaparecer. El contenido provisto a través de las conocidas páginas web son un claro ejemplo de ello. Nacieron para cubrir la necesidad de acceso a la información, pero las metodologías y medios de acceso a dicha información han evolucionado, del mismo modo que lo deben realizar las tecnologías de confección de los contenidos.

Actualmente, los usuarios acostumbrados a la cercanía y comodidad de las aplicaciones nativas de PC y móvil tanto en Android como en IOS, demandan la misma cercanía en el contenido servido mediante páginas web, pero éstas, hasta la fecha, son inflexibles no pudiendo ser separados de los navegadores que hacen de intermediarios entre ellas y el usuario.

Aunque las aplicaciones son más cercanas e inmediatas, distan mucho de ser perfectas, pues requieren de instalaciones previas desde las tiendas de aplicaciones y consumen recursos de los terminales que en gran cantidad de casos podemos considerar escasos.

En las últimas décadas, dado que no existía una solución óptima, ambas aproximaciones han convivido en paralelo. Sin embargo, las mejoras en rendimiento y usabilidad de las páginas web y los navegadores están comenzando a superar las deficiencias existentes. Aplicaciones híbridas y, más recientemente, las aplicaciones PWA (Progressive Web Application) se están erigiendo como la tendencia en la forma de acceder y servir contenido.

Las PWA permiten emular el concepto de aplicación nativa sustentando por las tecnologías proporcionadas por los navegadores web. Este concepto se desarrollará a lo largo de este proyecto en el que analizarán los beneficios de esta tecnología, su evolución y operativa. Mediante un caso de uso de gestión de aparcamiento en superficie soportado por una red de sensores, se busca demostrar las capacidades y viabilidad de las PWA para ofrecer servicios a los usuarios.

Abstract

Technology is not static, it is an entity which is born, grew and later only can evolve or disappear. The given content from the very well-known Web Pages concept is the best example. They were born to meet the user's needs about information access, but the methodology and the way to access that information has evolved, so the content provider technologies should also do it.

Nowadays, The users had got used to the closeness and commodities of the native apps of PC´s and mobiles in Android and IOS, they ask for this closeness also in the given content by Web Pages, but until now these pages are inflexible, they couldn´t be separated from the browsers, which are the intermediary between them and the users.

Even if the APP´s closeness and speed is bigger, they are far from being perfect, the requirements are the previous installation needed from app shops and they also consume a higher rate of the mobile resources that most times are scarce.

In the last decades, as neither of them has grown side by side as none of them had the capability to completely overturn the other one. However, the improvements in performance and usability of the web pages and browsers are starting to solve the existing deficiencies of both. Hybrid Apps and most recently the PWA's (Progressive Web Applications) are becoming the new trend to access and give content.

The PWA´s let us emulate the concept of native apps being supported by the given browser's technology. This concept will be worked on through the project, where the advances, evolution and way of work will be analyzed. Using a surface parking management case supported by a sensor network, we are looking to probe the viability and capability of the PWA in providing services to users.

Índice

Índice de figuras

Acrónimos

API	Application Programming Interface
CERN	Conseil Européen pour la Recherche Nucléaire
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
NGN	Next Generation Network
PWA	Progressive Web Application
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web

1 Introducción

Vivimos en una sociedad en eterno cambio, no solo social o generacional, sino desde un punto de vista puramente tecnológico. Las tecnologías que nos rodean están en continua evolución en aras de mejorar los mecanismos con los que proveer servicios de valor añadido a los usuarios finales. El desarrollo sufrido por la telefonía en las últimas décadas es un claro ejemplo. La Figura 1.1 muestra su evolución desde la aparición de la telefonía fija hasta llegar a los teléfonos los móviles inteligentes y más recientemente a los wearables.

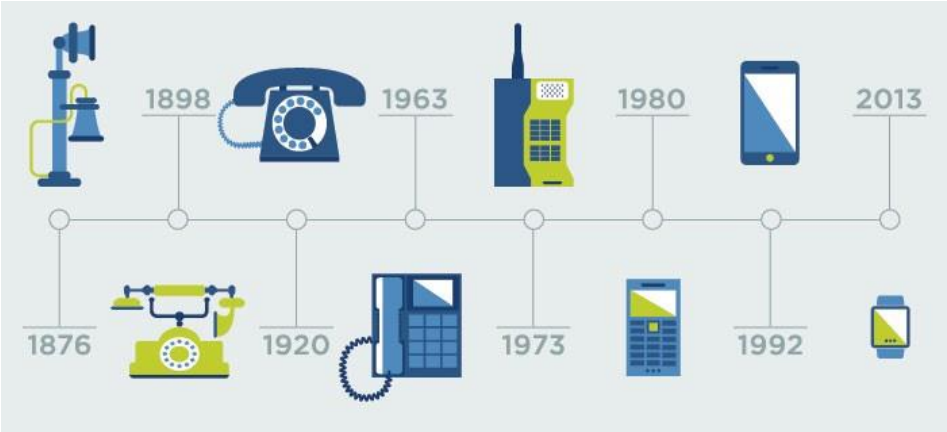


Figura 1.1 Evolución de la telefonía

La evolución tecnológica ha ido ligada a la capacidad en la miniaturización de los componentes hardware y al continuo incremento de la capacidad de procesamiento de los dispositivos. El cumplimiento de la conocida ley de Moore [1] demuestra este hecho, siendo la capacidad de procesamiento duplicada anualmente gracias a la posibilidad de introducir el doble de transistores en el mismo espacio.

Esta evolución, tan visible en el mundo de la telefonía, no ha sido exclusivamente única en este campo. Otra área de gran evolución la encontramos en el mundo de los ordenadores, el cual se ha progresado desde el primer computador de Honrad Zuse en 1936 [2], grande como una habitación, hasta los ordenadores de hoy en día, existentes en cualquier hogar.

No obstante, este desarrollo tecnológico no es únicamente debido a los desarrollos y mejoras a nivel hardware. El papel del software y su íntima relación con el hardware ha sido igualmente relevante.

Aun siendo ambos relevantes, no podemos decir que hayan seguido ambos el mismo camino. Mirando desde fuera, el mundo de la telefonía y de los ordenadores ha crecido en paralelo, ambos buscando un incremento en la cantidad de componentes posibles en un mismo espacio, logrando con ello una mejora continua de la capacidad de computo de los dispositivos. Esto, describiéndolo a grandes rasgos, ha dado lugar a la conversión de los móviles de un medio de comunicación oral a pequeños ordenadores de bolsillo. Actualmente la principal diferencia que aleja a un movil (smartphone) de un ordenador es la incapacidad de ofrecer la misma cantidad de recursos que ofrecen por limitaciones de tamaño. Eso sí, aun sin ofrecer los mismos recursos, sí que ofrece los mismos servicios

Mientras que en términos de hardware podemos decir que su crecimiento computacional ha ido de la mano, con el software la cosa ha sido muy distinta. El software ha mantenido su

incompatibilidad entre plataformas desde sus comienzos, ya no solo entre ordenadores y móviles, sino incluso entre dispositivos.

Esta diferenciación obliga a los diseñadores a adaptar todas sus implementaciones a todos los ecosistemas tecnológicos existentes. En estas circunstancias, se observó que el único punto en común entre todas estas tecnologías era el entorno web, transportado sobre HTTP y disponible en todos ellos.

Aprovechándose de ello aparecieron tecnologías como las aplicaciones hibridas o las aplicaciones PWA (PWA, Progressive Web Apps), más recientemente introducidas por Google, que intentan hacer uso de esta característica para crear un único servicio, válido independientemente de la plataforma.

1.1 Motivación y objetivos

El estado actual de continua evolución en el mundo del software y la actual tendencia actual hacia la convergencia de igual manera las NGN (NGN, Next Generation Network) [3] término que hace referencia a la evolución de redes de telecomunicación y acceso telefónico con el objetivo de la convergencia de los servicios multimedia (Voz, datos y video).

La posibilidad de trasladar esta convergencia a los entornos web ha incentivado este proyecto. Así se plantea probar que se puede crear un entorno ligero que permita obtener información en un tipo de red y trasladarlo a cualquier otra sin tener que adaptar la aplicación a todas las plataformas existentes en cada uno de los dispositivos y aun así contar con las capacidades que estos otorgan.

En base a lo anteriormente descrito, y con el objetivo principal de analizar la viabilidad del uso de aplicaciones PWA en entornos multiplataforma, se trabajarán los siguientes aspectos:

- Diseño e implementación de una plataforma de soporte a aplicaciones PWA en la que se incluirán servicios de almacenamiento, alojamiento y provisión de contenidos, etc.
- Diseño e implementación de un interfaz de usuario multiplataforma y multiecosistema.
- Diseño de un servicio que valide un caso de uso de aplicación PWA en el entorno de la Internet de las Cosas (IoT, Internet of Things).

Para llevar a cabo estos objetivos, se centrarán los trabajos en el diseño, implementación, integración y despliegue de una solución de gestión de aparcamiento de pago en superficie que haga uso de una solución de sensores de detección de vehículos estacionados.

1.2 Organización del trabajo

El presente trabajo se ha dividido en cuatro capítulos, tal como se muestra a continuación:

- En el primer capítulo, Introducción, se sitúa y expone el marco del trabajo, las motivaciones que dan lugar al mismo y objetivos perseguidos.
- En el segundo capítulo, Estado del arte, se definen las bases que sostienen los fundamentos del trabajo.
- En el tercer capítulo, Implementación PWA, se detalla el proceso de implementación práctica de la aplicación en la que se basa el proyecto.
- En el cuarto capítulo, Conclusiones y líneas futuras, se evalúan las conclusiones alcanzadas a través del desarrollo del proyecto, así como las posibilidades que ofrece de mejora.

2 Estado del arte

En este capítulo se realiza un estudio del estado del arte de los fundamentos sobre los que sustenta este trabajo, a saber, la tecnología web de desarrollo de aplicaciones conocida como PWA y las tecnologías de comunicaciones subyacentes para dar respuesta a las necesidades de las ciudades inteligentes.

2.1 Evolución en el diseño de aplicaciones

2.1.1 Contenido web

A principios de 1989, Sir Tim Berners-Lee, un científico de CERN (Conseil Européen pour la Recherche Nucléaire) ideara una metodología para el intercambio de información entre ordenadores de forma transparente [4]. Surgía entonces lo que hoy se conoce como la World Wide Web (WWW). Lo que en un principio se consideró para uso únicamente dentro del CERN, resultó ser algo con un gran potencial que se abrió a toda la comunidad.

Aunque la base de la solución planteada inicialmente se sostenía en el uso de 3 tecnologías (protocolos y lenguajes), como son HTML (HyperText Markup Language), URI (Uniform Resource Identifier) y HTTP (Hypertext Transfer Protocol), la idea subyacente, el hipertexto y los vínculos entre documentos distribuidos databa ya de muchos años antes.

La arquitectura funcional de la WWW se basa en el despliegue colaborativo de un conjunto de servidores que proveen contenido, inicialmente en texto plano, pero actualmente multimedia, y una aplicación de entorno de usuario conocida como navegador web que solicita, interpreta y presenta la información al usuario.

La evolución de la WWW ha ido ligada de la evolución en las tecnologías, lenguajes y protocolos usados tanto en la descripción de los recursos de información, como en la forma en la que estos se transmiten por la red, sin olvidar, los vinculados a la forma en la que se presentan al consumidor final. La Figura 2.1 muestra el detalle de la evolución desde el punto de vista de las tecnologías de descripción e interpretación de la información de los recursos distribuidos. El protocolo de comunicación HTTP sobre el que se sustenta la web, hasta finales de 2015 en que se implementaron mejoras notables (modo binario), no ha sufrido modificaciones mintiéndose como un protocolo de capa de aplicación basado en una estructura de texto estandarizada.

Las primeras páginas web que contenían texto plano, paulatinamente, gracias a las mejoras en los clientes de visualización, fueron incluyen contenido multimedia. La visualización de los contenidos mejoró con la incorporación de las hojas de estilo (CSS, Cascading Style Sheets) que homogenizaban la forma de describir las características de visualización. En paralelo el desarrollo de JavaScript consiguió que el contenido estático de la tradicional página web pasara a ser dinámico, logrando una interacción más fluida entre el cliente, el contenido y el proveedor del servicio. Lenguajes complementarios al HTML para la descripción de contenidos como por ejemplo los gráficos vectoriales escalables (SVG, Scalable Vector Graphics) ayudaron a adaptar los contenidos a las nuevas tecnologías de visualización y dimensiones de los dispositivos de usuario.

Son estos últimos, los dispositivos de usuario, los que están guiando la forma en la que los usuarios consumen información y por tanto las metodologías que emplean para acceder a ella, entre ellas la web. La eclosión de la telefonía móvil no fue más que el germen de una revolución aún mayor como fue la telefonía móvil inteligente. Dispositivos con capacidad de ofrecer funcionalidades más allá de la mera conversación se pusieron al alcance del usuario a mediados

de la década del 2000. Estos dispositivos, conocidos como teléfonos inteligentes o smartphones, permitían la ejecución de aplicaciones de forma local.

El mercado de las aplicaciones orientadas a teléfonos móviles inteligentes ha crecido de forma exponencial (ver Figura 2.2a). Los teléfonos móviles inteligentes viajan con el usuario por lo que abren un abanico de potenciales servicios puesto que se puede tener acceso a las necesidades, gustos, etc. del usuario en cualquier momento y lugar. La Figura 2.2b muestra el importante grado de penetración de la telefonía inteligente a nivel mundial.

Estas aplicaciones nativas, a diferencia de las soluciones web puras, son altamente dependientes de la plataforma o el sistema operativo que ejecute el teléfono inteligente. Entre sus características, se puede destacar:

- Se elaboran directamente para los dispositivos en los que se va a usar, por un lado, móvil u ordenador, y por otro lado pensando en el sistema operativo que estos emplean.
- La aplicación se aloja en la memoria local del dispositivo, haciendo que éste cargue con el peso del procesamiento.
- Mayoritariamente se descargan directamente desde las tiendas oficiales (Marketplace) de los fabricantes y/o desarrolladores del sistema operativo.
- Tienen acceso a todos los recursos hardware del dispositivo.
- Pueden emplear todas las capacidades del dispositivo de forma nativa, optimizando su funcionamiento, lo que incremente su potencia gráfica, usabilidad, velocidad, y bajo tiempo de respuesta.
- Pueden operar sin necesidad de internet.

El gran volumen de aplicaciones existente ha hecho que se desarrollen soluciones de publicación y venta de aplicaciones. Las tiendas de aplicaciones, dependientes del sistema operativo, han generado contando solo el año en curso, 39.700 millones de dólares [5].

A diferencia de las páginas web que se indexan a través de buscadores, las aplicaciones móviles suelen requerir su despliegue en las tiendas para poder instalarse en los dispositivos de los usuarios. Una página web, donde gran parte de su contenido se proporciona siguiendo estándares bien conocidos como los definidos por el W3C, puede ser accesible a través de cualquier navegador o cliente web. Como se ha indicado, para el caso de las aplicaciones móviles existe una gran diversidad de plataformas, lo que redunda en un amplio coste de producción y mantenimiento de las soluciones por parte de los desarrolladores.

Por tanto, se hace patente la necesidad de buscar una solución que encamine las aplicaciones móviles hacia la independencia del dispositivo

Pasar de un almacenamiento y ejecución local a una solución sustentada parcialmente en la distribución de funcionalidades en la red inicialmente se postula como una solución de compromiso. Las aplicaciones híbridas permiten compartir las bondades de la ejecución local nativa y la computación distribuida.

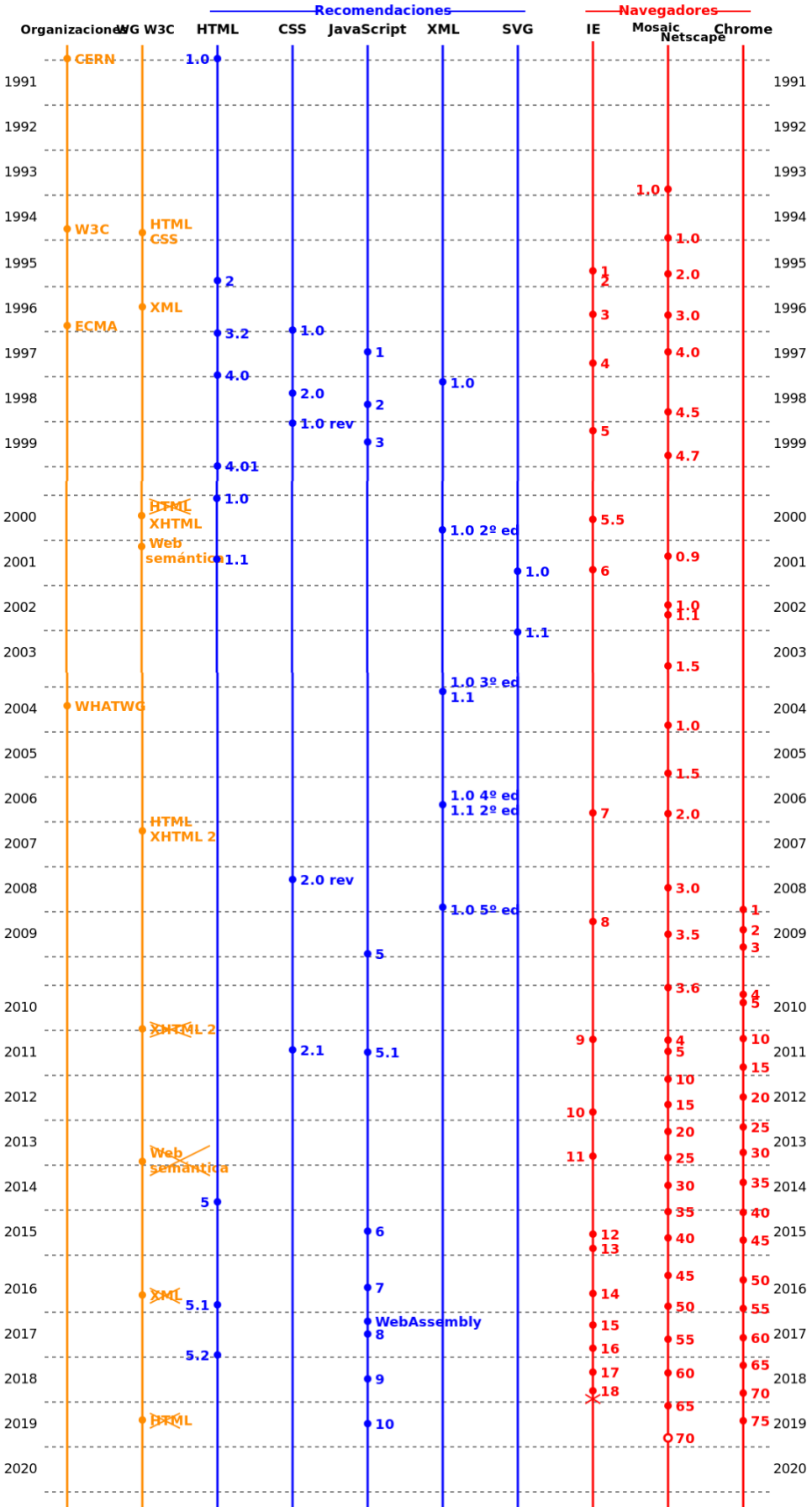
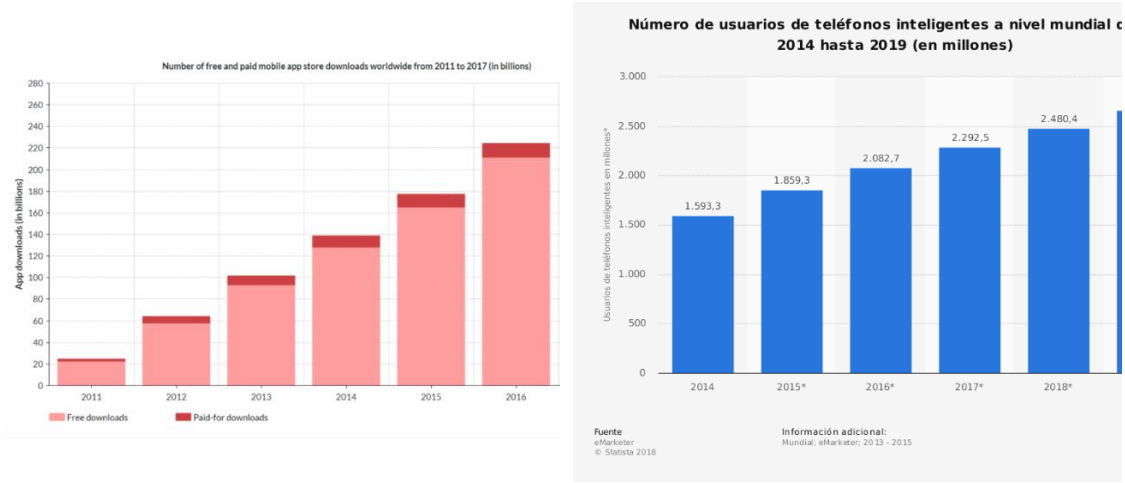


Figura 2.1 Evolución de las tecnologías web



a) Volumen de aplicaciones móviles b) Mercado de telefonía móvil inteligente

Figura 2.2 Mercado de telefonía móvil inteligente

2.1.2 Aplicaciones hibridas

Desde un punto de vista operacional, suelen estar basadas en el uso de un navegador embebido (WebView) gracias al cual presenta al usuario el contenido remoto. Este navegador se abre dentro de la propia aplicación local como un diálogo más, obteniendo el usuario una sensación de continuidad en el uso de la aplicación. Ni que decir tiene que, para mantener una óptima experiencia de uso, el diseño de las ventanas/diálogos locales debe ser coherente con lo que se muestre en el navegador embebido, y viceversa.

Las aplicaciones híbridas están escritas combinando lenguajes de programación móvil nativos y en lenguajes orientadas a la web. Por tanto, a pesar de que su estructura lógica puede diferir de la de las aplicaciones nativas, comparten con ellas el hecho de que se deben desarrollar y publicar en la correspondiente tienda de aplicaciones de forma independiente para cada plataforma móvil. Esto supone que una de las barreras apuntadas para las aplicaciones nativas, la necesidad de un desarrollo específico sigue estando latente.

En términos de rendimiento, las aplicaciones híbridas tienen un menor rendimiento puesto que la propia latencia de la red de comunicaciones implica una experiencia de uso algo inferior. En términos de funcionalidades, dado que el código nativo sigue existiendo, aplicando ciertas metodologías de programación y organización y flujo de la aplicación se puede acceder a todos los recursos disponibles.

2.2 Aplicaciones web progresivas

El último salto evolutivo en el desarrollo de aplicaciones para entornos móviles buscará el aunar por completo las bondades de la tecnología web y el acceso a recursos locales, sin necesidad de emplear plataformas de desarrollo y ejecución específicas para cada tipo de dispositivo.

En este sentido, el navegador web es el cliente universal de acceso, presentación e interacción con la información y está disponible en prácticamente cualquier plataforma. Por tanto, definir metodologías que permitan la ejecución de código con capacidades adicionales de acceso a recursos locales mediante un navegador web se considera como la línea a seguir.

En 2013 surge Electron [6] como un entorno de código abierto que permite el desarrollo de aplicaciones gráficas de escritorio usando componentes del lado del cliente y del servidor

originalmente desarrolladas para aplicaciones web. Para ello aúna la potencia de Node.js del lado del servidor y Chromium [7], versión de código abierto del navegador Chrome, como interfaz. El empaquetado de una aplicación Electron incluye por tanto un navegador y un servidor web con capacidades de acceso a servicios locales, si bien el código de la aplicación podrá interaccionar con servicios remotos. Se materializa entonces la idea de llevar la tecnología web al entorno de las aplicaciones.

Las aplicaciones basadas en Electron rápidamente ganaron gran tracción gracias en parte a la facilidad de desarrollo y a la amplia comunidad. Puesto que se basan en tecnologías web, cualquier desarrollador de páginas web sería capaz de, con no demasiado esfuerzo, realizar aplicaciones de escritorio. Su rápida adopción y potencia las llevo a ser incluidas en las tiendas de aplicaciones de los equipos de sobremesa basados en Windows y Mac.

Tras Electron el siguiente paso natural ha sido eliminar la necesidad de empaquetar el servidor y un navegador, estando principalmente el segundo incluido en los equipos clientes por defecto. Si el código del servidor, desarrollado en Node.js basado en JavaScript, se pudiera ejecutar en segundo plano desde el navegador, se daría respuesta a todas las necesidades y dificultades apuntadas hasta ahora.

En 2015, Google decide incluir estas prestaciones en su navegador [8] y con ello define un nuevo tipo de aplicaciones, a las cuales denomina Aplicaciones Web Progresivas (PWA, Progressive Web Apps). Considerando que su navegador es capaz de operar como motor de ejecución de estas aplicaciones y que este motor está disponible en cualquier dispositivo (PC y móvil) y plataforma (Windows, Mac, Linux, Android, iOS, etc.), se logra universalizar el uso de aplicaciones.

Las aplicaciones PWA son aplicaciones web, que, a diferencia de las aplicaciones nativas, son inicialmente accesibles a través de un navegador en lugar de tener que ser descargadas e instaladas en un dispositivo. Visualmente, al ejecutarse a pantalla completa, la experiencia de usuario es idéntica a la de una aplicación nativa. De hecho, tras varios usos la aplicación PWA generará un acceso directo y empleará el contenido en cache local, equivalente a una instalación, para mejorar la experiencia de uso. Es decir, su proceso de instalación es totalmente transparente para el usuario. De igual forma se puede considerar que una aplicación PWA es totalmente accesible sin conexión.

Las características clave de las aplicaciones PWA son:

- Rapidez: deben ser rápidas, con un tiempo de carga casi nulo dando así lugar a una sensación de cercanía y familiaridad. Éste es el atributo más demandado por cualquier consumidor de servicios en todos los ámbitos no solo el tecnológico.
- Fiabilidad: deben estar totalmente disponibles y aún sin conexión deben proporcionar una sesión de total y completa funcionalidad. El objetivo es hacer que las incomodas páginas de error o en blanco de los navegadores sean un recuerdo del pasado.
- Atractivo: desplegar aplicaciones con el cuidado diseño de las aplicaciones nativas para hacerlas más amigables para los potenciales usuarios.

De manera adicional, otra característica que las hacen asemejarse a las aplicaciones nativas es la posibilidad de que el usuario reciba notificaciones. Una vez descargada, una aplicación PWA podrá configurarse la recepción de notificaciones push, lo que supone un modelo asíncrono de comunicación con el usuario.

Pero las aplicaciones PWA también tienen sus limitaciones:

- Para poder hacer uso de las características de las PWA, siempre se requiere que las

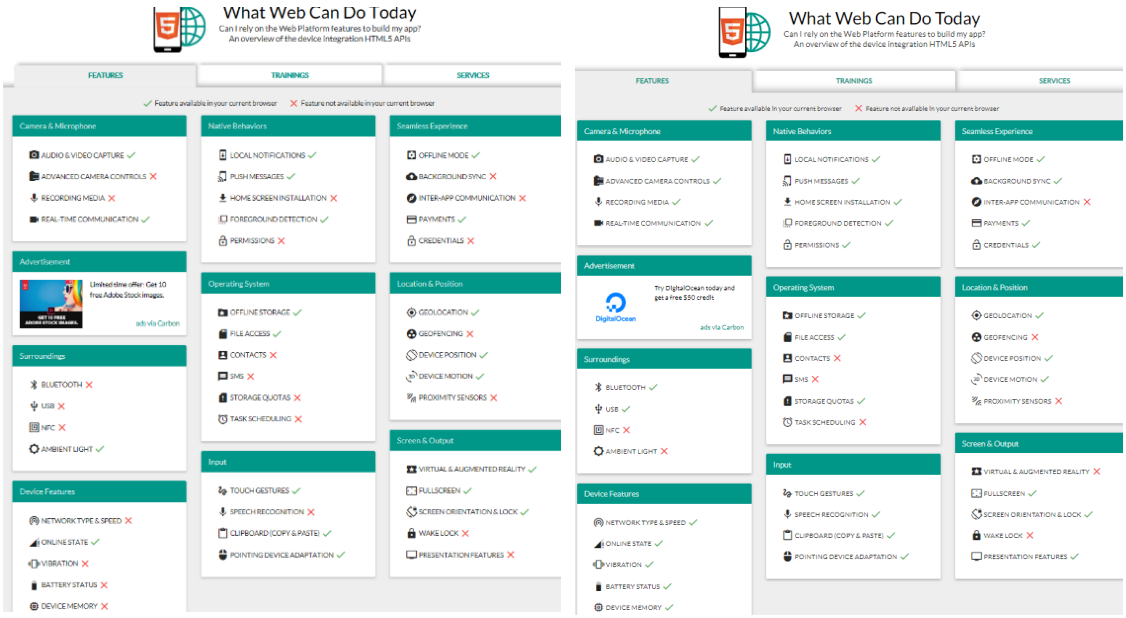
conexiones se establezcan usando TLS (TLS, Transport Layer Security), que, aunque ya prácticamente es usado por casi todos los sitios web, aún no ha sido adoptado por todos.

- Aún no todos los navegadores han implementado las PWA. Aunque los principales ya lo han hecho navegadores con gran volumen de usuarios como Safari todavía no han confirmado su adaptación.

2.2.1 Funcionalidades

Como se ha comentado las aplicaciones PWA han ido de la mano de la evolución del navegador web, quien además de ser el visualizador del contenido, es la interfaz con los recursos locales. La principal mejora con respecto a las aplicaciones híbridas es que ya no se necesita código nativo para el acceso a dichos recursos. Sin embargo, es necesario que las API (API, Application Programming Interface) disponibles para la interacción entre la página web o aplicación y los navegadores incluyan el acceso a los diferentes recursos

En whatwebcando.today [9] está disponible el listado de todas las funcionalidades locales accesibles a través del navegador. De acuerdo con ella, los navegadores pueden ofrecer un distinto grado de funcionalidad.



a) Edge b) Chrome

Figura 2.3 Comparativa entre las funcionalidades en Chrome y Edge.

Como vemos en la Figura 2.3, al ser Google el mayor impulsor de las aplicaciones progresivas, las funcionalidades que tiene habilitadas en su navegador, representado en la Figura 2.3b es bastante mayor. Edge por el contario está todavía en proceso de preparación para adaptarse correctamente a esta nueva tecnología.

2.3 Marco estructural:

El modelo óptimo para crear aplicaciones PWA es lo que se denomina una arquitectura Shell [10]. Esta aproximación busca una carga rápida (instantánea) y confiable, de forma similar a la experiencia ofrecida por las aplicaciones nativas. Este tipo de arquitectura es el más adecuado cuando la aplicación a generar no contenga cambios en la estructura o formato en la interfaz de usuario, sino en el contenido.

Para ello, define un armazón o estructura básica compuesta de los elementos comunes y mínimos que dan forma a la aplicación progresiva, es decir, la cantidad mínima de contenido HTML, CSS y JavaScript para activar la interfaz de usuario. En la mitad izquierda de la Figura 2.4 se observa esta estructura básica. El código HTML introducirá los elementos más estáticos, cabeceras, pie de página, panel de control o cualquier otro componente que se vaya a usar independiente de la información que la página maneje. Este código estará vinculado a unos estilos definidos por la plantilla CSS. Por su parte, JavaScript incluirá las reglas para que todos los elementos anteriores se interconecten y trabajen conjuntamente de una manera fluida tanto.

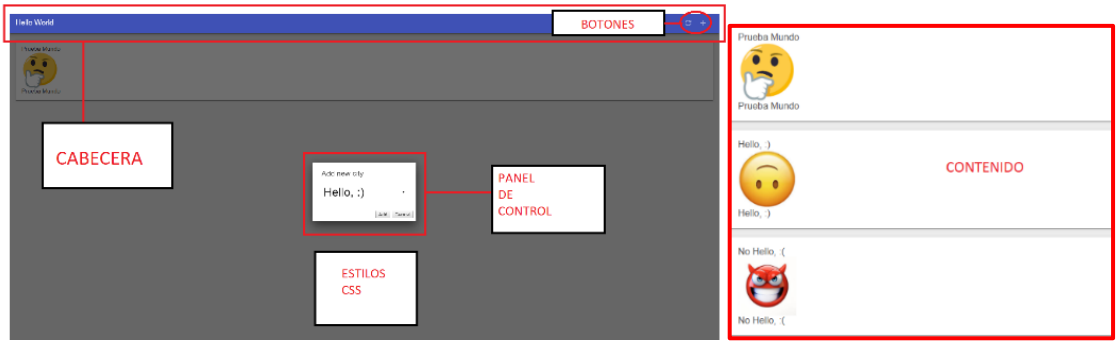


Figura 2.4 Estructura de una página de una aplicación PWA

La ventaja ofrecida por esta disposición de las páginas es que la información estructura ya está descargada y disponible sin conexión (almacenada en caché), lo que reduce los tiempos de carga y el tráfico de red generado por la misma.

Teniendo ya descargado todo el armazón de la aplicación solo quedará instalar el contenido, como el de la mitad derecha de la Figura 2.4. El solo tener que descargar el contenido reduce el número de peticiones que hay que realizar, así como la cantidad de información contenida en cada una de ellas. Esto se traduce en un menor consumo de datos por parte del usuario.

El contenido alojado y recuperado de la memoria caché es el que hace que las aplicaciones PWA se asemejen en comportamiento a las aplicaciones nativas. Se trata de un contenido temporal y, por tanto, hay que realizar una revisión periódica del mismo para adecuar la interfaz y eliminar elementos obsoletos del mismo, al tiempo que se mejora la experiencia sin conexión.

Las aplicaciones PWA deben cumplir una serie de requisitos para satisfacer las prestaciones que de ellas se esperan:

- Bajos tiempos de carga.
- Reducir el número de peticiones de información y contenido remoto.
- Realizar una gestión óptima del contenido en caché y local.
- Evitar actualizaciones globales de la interfaz de usuario y fomentar la capacidad de navegación a través de los recursos.

2.3.1 Nuevos componentes

Las aplicaciones PWA, aun manteniendo la filosofía del paradigma web, constan de elementos internos diferenciadores con respecto a una plataforma web tradicional. No obstante, mantienen los mismos lenguajes de programación web, que las diferencias de las aplicaciones nativas.

Los elementos básicos, además de la estructura de páginas web (HTML, JavaScript y CSS) que integran las aplicaciones PWA son los Service Workers y una descripción de la propia aplicación

definida en un manifiesto. Para entender correctamente que son y cuál es su función dentro de las PWA, se procede a continuación a su descripción en profundidad.

2.3.2 Manifiesto

El manifiesto [11] es un documento que define la manera en la que se presenta la aplicación al usuario final. Este documento descrito en formato JSON establece la forma en la que la aplicación se incorpora en áreas donde normalmente se ven aplicaciones nativas (por ejemplo, la pantalla de inicio de un dispositivo móvil o el escritorio de un ordenador). Además, permite establecer cómo se inicializa la aplicación y su apariencia, por ejemplo, agregando una pantalla de presentación, etc.

La inicialización de acuerdo con el manifiesto procederá como se menciona a continuación:

- Establece el nombre corto de la aplicación, y diferentes tamaños de imagen para que se puedan cuadrar con los distintos dispositivos.
- Durante el proceso de descarga de los recursos, o su restauración desde caché, establece un estado intermedio dinamizar el tiempo de carga.
- Estable una configuración predeterminada en el navegador para evitar transiciones bruscas al entrar la información.

En conclusión, el manifiesto representa los metadatos de la aplicación y proporciona la información base necesaria para que las PWA cumplan las características que prometen.

La vinculación entre las diferentes vistas de la PWA y el manifiesto se realiza desde los documentos HTML que describen las páginas web que la conforma. Como se observa en la Figura 2.5, se sigue un procedimiento similar al realizado para vincular la CSS con la página en cuestión.

```
<link rel="manifest" href="/manifest.json">
```

Figura 2.5 Etiqueta link con referencia al manifiesto

En la Figura 2.6 se muestra un ejemplo de un manifiesto para una aplicación. Seguidamente se procede a describir sus elementos, que se pueden agrupar en 3 categorías:

```
{
 "$schema": "http://json-schema.org/draft-07/schema#" 1
 "name": "HelloWorl", 2
 "short_name": "HelloWorld", 3
 "icons": [{ 4
   "src": "img/icons/icon128.png",
    "sizes": "128x128",
    "type": "image/png"
   }, {
    "src": "img/icons/icon144.png",
    "sizes": "144x144",
    "type": "image/png"
   }, {
    "src": "img/icons/icon152.png",
    "sizes": "152x152",
    "type": "image/png"
   }, {
    "src": "img/icons/icon192.png",
    "sizes": "192x192",
    "type": "image/png"
   }, {
    "src": "img/icons/icon256.png",
    "sizes": "256x256",
    "type": "image/png"
   }],
 "start_url": "/user/index.html", 5
 "display": "standalone", 6
 "background_color": "#3E4EB8", 7
 "theme_color": "#2F3BA2" 8
}
```

Figura 2.6 Elementos del manifiesto

- Elementos que facilitan la interpretación del navegador

  Esta categoría considera los componentes que facilitan el trabajo del navegador a la hora de compilar la aplicación de una manera directa. Por un lado, tenemos el elemento `schema` (1), opcional pero altamente recomendado, que incluye una URL que permite proporcionar al navegador un modo de validar el manifiesto. Y por otro, el elemento `start_url` (5) que aloja la URL de inicio de nuestra app a ser cargada por el navegador.

- Elementos relativos a los iconos de la aplicación

  En esta clasificación se incluyen los campos que afectan a nuestra manera de visualizar la aplicación en el escritorio de usuario.

  En referencia al nombre de la aplicación tenemos dos campos como son `name` (2) y `short_name` (3). El primero de ellos establece el nombre que aparecerá en el banner de instalación de la aplicación, mientras que el segundo, determina el texto que se incluye el acceso directo asociado a la aplicación y que suele ser más breve.

  Como último elemento en esta categoría hemos añadido el campo `icon` (4), el cual contiene las direcciones a los iconos de la aplicación dispuestos por tamaños.

- Elementos para la visualización de la página.

  La última categoría agrupa los elementos que afectan a la manera en que los usuarios visualizarán la aplicación.

  En este grupo hemos incluidos los elementos `display`, `background_color`, `theme_color` u `orientation`. De ellos `display` (6) proporciona al navegador la manera en que el dispositivo quiere visualizar la aplicación, es decir, como un navegador común con la barra de búsqueda (`browser`) o como una aplicación nativa ocultando la barra de búsqueda (`standalone`).

  `background_color` (7) y `theme_color` (8) se encargan de la coloración de la página en la etapa de inicialización y, como sus nombres indican, el primero afectará al fondo de la aplicación, mientras que el segundo atribuirá un color al tema de la interfaz de usuario.

  `orientation` tiene el objetivo de establecer la manera en la que la página se mostrará al usuario, vertical o apaisado con los valores de campo `portrait` y `landscape` respectivamente.

Como hemos podido ver el manifiesto tiene la capacidad de crear una experiencia de usuario similar a la de una aplicación nativa, mucho más cercana que la de una página web común.

2.3.3 Service Worker

Este componente definido en la estructura de las PWA es el que se considera el sustento de lo que las aplicaciones PWA son. El Service Worker [12] proporciona la capacidad de gestionar la memoria cache, trabajar sin conexión o realizar notificaciones push, entre otras muchas funcionalidades.

Una definición más estricta considera al service worker como una secuencia de comandos ejecutadas en un navegador en segundo plano y separadamente del servicio web desarrollado, convirtiéndolas así en un tipo de programación que se ejecuta independientemente tanto del navegador como del usuario.

Entre las características realizadas en segundo plano están como ya se ha mencionado previamente las notificaciones y la sincronización. Aunque esta última actualmente no está

disponible se espera que en poco tiempo se habilite la posibilidad de sincronizaciones periódicas [REF]. La inclusión de las notificaciones push ha permitido reducir la brecha existente con las aplicaciones nativas.

2.3.3.1 Funcionamiento

El proceso del service worker se pone en ejecución una vez se realiza el inicio de la aplicación en el navegador. A partir de ese momento, un service worker se mantiene en ejecución. Esto no quiere decir que estos tengan un tiempo de vida más largo, ya que se pueden producir modificaciones en los mismos mientras la página se mantiene abierta. Para tratar con estas diferencias temporales se pueden establecer funciones para solicitar periódicamente una nueva descarga a la última versión posible.

El proceso de inicialización de una aplicación PWA consta de las etapas que se definen en la Figura 2.7.

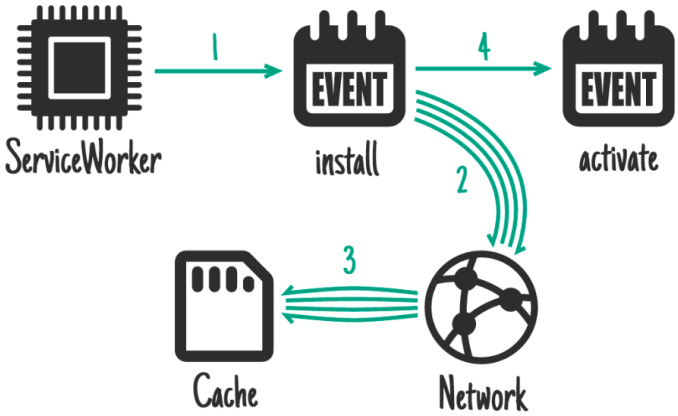


Figura 2.7 Proceso de inicialización de un service worker

Primero se inicia el proceso de descarga de los documentos estándar de una página web por el acceso a su ubicación por parte del usuario a través de un navegador. Puesto que no todos los navegadores permiten la ejecución de un service worker, parte del código JavaScript de la página de inicio incluirá la validación de la capacidad del navegador para proveer PWA. En la Figura 2.8 se incluye un ejemplo de dicho código.

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    navigator.serviceWorker.register('/user/service-worker.js').then(function(registration) {
      // Registration was successful
      console.log('ServiceWorker registration successful with scope: ', registration.scope);
    }, function(err) {
      // registration failed :(
      console.log('ServiceWorker registration failed: ', err);
    });
  });
}
```

Figura 2.8 Solicitud Service Worker

En caso de recibir una respuesta positiva, el navegador lo buscará en la ubicación solicitada y procederá a registrar e iniciar el proceso de instalación. El comando `register` detectará si el service worker existía previamente en el navegador y actuará en consecuencia.

El proceso de instalación se inicia con el registro de todos los elementos de la página en caché. Aunque esto no es obligatorio, como se ha descrito anteriormente, es una de las características más significativas de las aplicaciones PWA. En caso de originarse algún error en este registro, la

instalación fallará y no se activará. Bajo estas circunstancias habrá que esperar al próximo acceso a la página en la que se volverá a iniciar todo el proceso.

Acto seguido se continua con al proceso de activación, en el que existe la posibilidad de gestionar la memoria caché, entendiendo gestión como la capacidad de modificar la memoria existente. La tarea más comúnmente desempeñada en esta etapa es la eliminación de los restos de caché antiguos en la memoria.

Tras esto, el service worker estará funcionando sobre todas las páginas a su alcance, todas menos aquella que inicio el proceso de descarga, en la cual entrará en vigor en su próxima conexión. A partir de aquí solo existirán dos estados posibles, uno en el que se rescindirá para ahorrar recursos, y el otro que se encargará de gestionar el flujo de mensajes, guardando el resultado de las peticiones, y en el caso de que se le solicite una segunda vez la misma información y no se pueda acceder a información actualizada, devolver como respuesta a la petición la misma que almaceno como respuesta a la primera petición.

La Figura 2.9 se muestra el proceso descrito gráficamente.

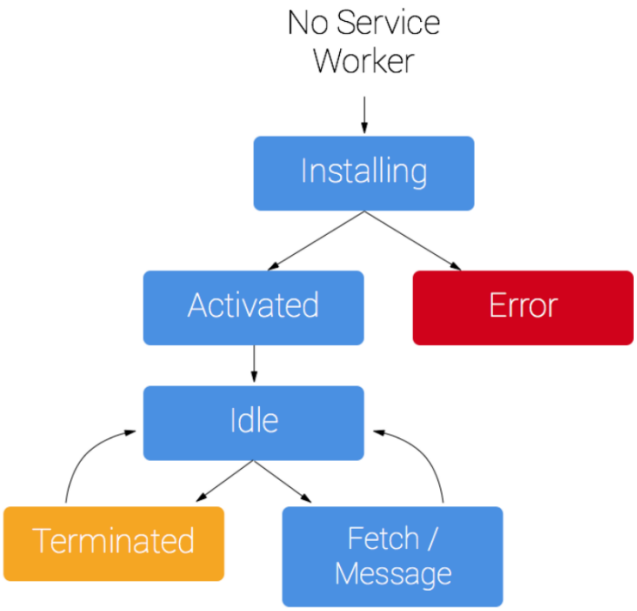


Figura 2.9 Instalación Service Worker.

2.3.3.2 Tratamiento de la memoria Caché.

Aunque es cierto que la gestión de la memoria caché no es la única funcionalidad que ofrecen, debido a su importancia hemos decidido ahondar en las distintas estrategias que una aplicación puede plantear entorno a esta función [13] .

De entre todas estas posibilidades destacan a la hora de almacenar la información tres:

- Almacenamiento en el momento de instalación: consiste en almacenar los archivos que se transmiten inicialmente y componen el esqueleto o armazón de la aplicación. Proceso visible en la Figura 2.7

- Almacenamiento ante la interacción del usuario: habilita la posibilidad de que el usuario almacene voluntariamente información para su posterior uso off-line como podrían ser artículos de noticias. Esto aparece descrito a continuación en la Figura 2.10

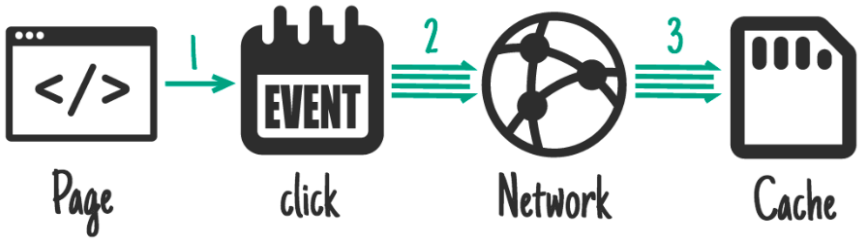


Figura 2.10 Almacenamiento por la acción del usuario.

- Almacenamiento ante las respuestas de la red: permite analizar la respuesta a las peticiones y en caso de ser estas nuevas, se almacena la respuesta en caché. Esta medida es ideal para aplicaciones con que se actualizan con gran frecuencia. Como aparece en la Figura 2.11

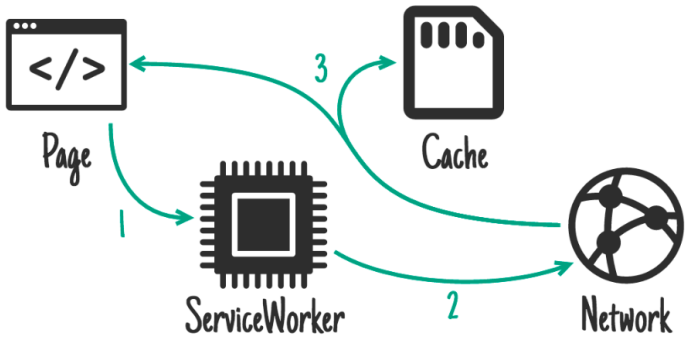


Figura 2.11 Almacenamiento ante la respuesta de la red.

Una vez abarcado el rango de las posibles formas de almacenar la información, toca tratar las distintas alternativas de devolver esta cuando el usuario lo requiera. principalmente hay cinco alternativas:

- Cache Only: la aplicación es completamente estática por lo que con poder realizar la instalación inicial no requeriría de acceso a red para acceder a nueva información y extraería los datos de la caché. Esta alternativa se corresponde con la Figura 2.12.

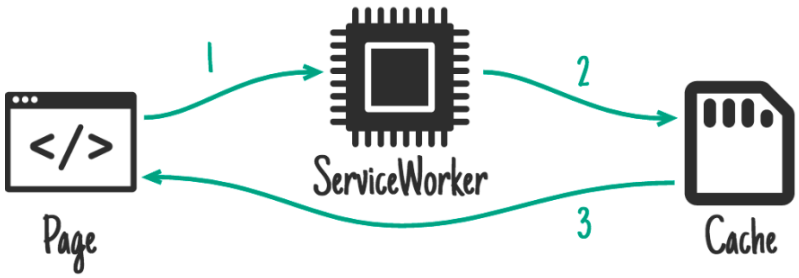


Figura 2.12 Metodología Cache Only.

- Network Only: este tipo de aproximación se usa en páginas que requieren de acceso continuo, como sería el caso de las páginas que tienen peticiones que no son del tipo GET. Hay que tener en cuenta que solo las peticiones GET pueden ser cacheadas. Se puede observar este proceso en la Figura 2.13.

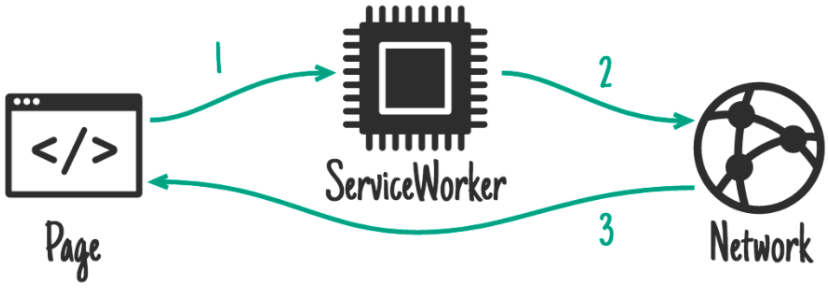


Figura 2.13 Metodología Network Only.

- Cache falling back to the network: este método está basado en los dos anteriores, pues primero intenta tratar la información de manera off-line, y en caso de no contar con la información en caché realizará la petición a través de la red para lograrla. Esto está descrito a continuación en Figura 2.14.

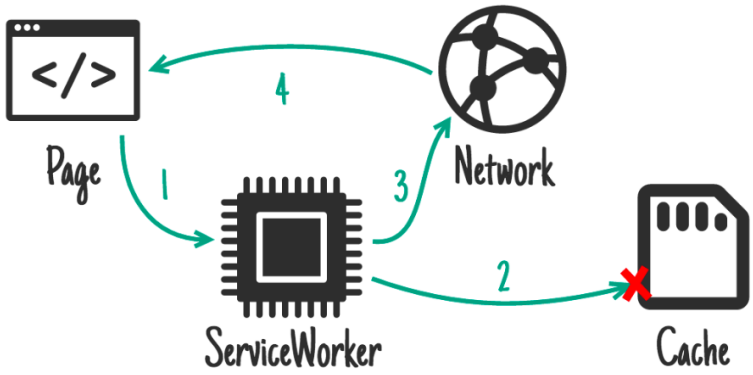


Figura 2.14 Metodología Cache falling back to the network.

- Network falling back to the Cache: alternativa totalmente opuesta a la anterior en la que primero intenta acceder a la información más actual disponible en la red, y en caso de no contar con conexión emplea los datos previos para seguir funcionando. Es una de las opciones más usadas, pero tiene la desventaja de que en los casos de contar con mala conexión presenta los retrasos inherentes a tener que realizar la petición, esperar contestación y finalmente buscar en caché. Este proceso se muestra en la Figura 2.15

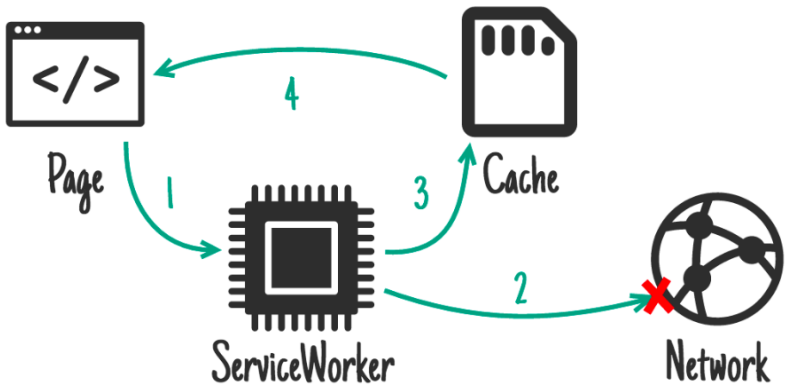


Figura 2.15 Metodología Network falling back to the cache.

- Cache then Network: esta es una alternativa intermedia entre todas las opciones previas. Se realizan ambas peticiones simultáneamente y primero mostraría la información

almacenada en caché y, en el momento de la respuesta de la red, se actualiza el contenido. En la Figura 2.16 aparece este proceso descrito.

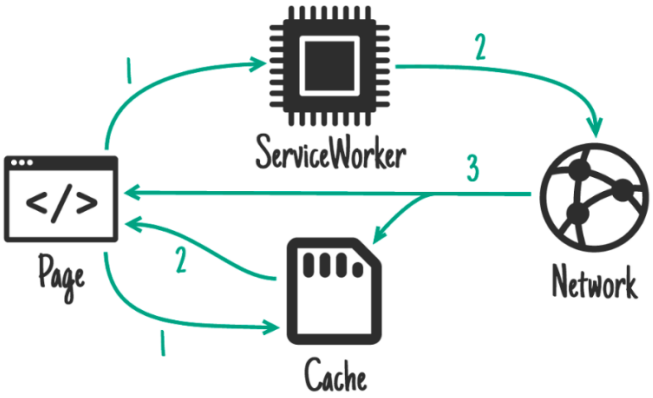


Figura 2.16 Metodología Cache then Network.

Como se puede inferir de lo anterior, no existe una arquitectura óptima y dependiendo de las circunstancias relativas a cada aplicación, se deberá elegir la mejor opción acorde a ellas.

2.3.3.3 Requerimientos adicionales

Incluso habiendo incluido todos los elementos anteriores, es posible que se tenga una aplicación PWA completamente funcional. Como requerimiento adicional se requiere que toda comunicación realizada con la aplicación se realice en un entorno seguro, es decir, transmitiendo la información sobre HTTPS.

En este sentido, no es suficiente con desplegar soluciones temporales (certificados autofirmados, etc.), sino que es necesario disponer de un certificado totalmente valido para que el service worker sea instalado y entre en funcionamiento.

2.4 Rendimiento

Todos los usuarios de algún servicio aspiran a obtener el que le haga sentirse más cómodo, es decir el que le proporcione una mejor experiencia de usuario.

Como se ha comentado con anterioridad las características de las PWA´s las hacen inclinarse por lograr páginas que parezcan nativas para acercarlas al usuario y su distintiva velocidad, que le proporciona en casos normales de servicio velocidades en cualquier situación por debajo del segundo, y más comúnmente sus retardos se miden en décimas.

Siendo esta una característica atractiva para el consumidor, observemos con datos reales como afecta al distribuidor la mejora en el rendimiento.

En el caso de Pinterest, la reducción de los tiempos de espera en un 40% le supuso un incremento de un 15% de usuarios [14]. Para Mobify, la reducción de este tiempo en 100 milisegundos le supone un aumento en sus ingresos anuales de 380.000 dólares [15]. Otro ejemplo que muestra la relevancia del rendimiento, son las cifras que mostraban cómo la BBC estimaba una pérdida de un 10% de usuarios por cada segundo de carga adicional [16].

3 Implementación PWA.

El desarrollo del presente trabajo busca la implementación de una solución de gestión de pagos y uso de aparcamiento en superficie, en la que se aunarán la colecta y publicación de datos por parte de sensores de estacionamiento y la presentación de esos datos y la interacción con la plataforma de pago mediante una aplicación PWA.

La arquitectura de alto nivel de la solución se muestra en la Figura 3.1. Por un lado, observamos una red IoT de sensores compuesta de múltiples sensores de aparcamiento, que detectarán la existencia o no de un vehículo en su proximidad. Los sensores, inicialmente dispuestos con una topología en estrella, transmitirán los datos recogidos a un nodo central o gateway que redirigirá la información hacia Internet, donde se alojarán los servidores de almacenaje y procesado de datos.

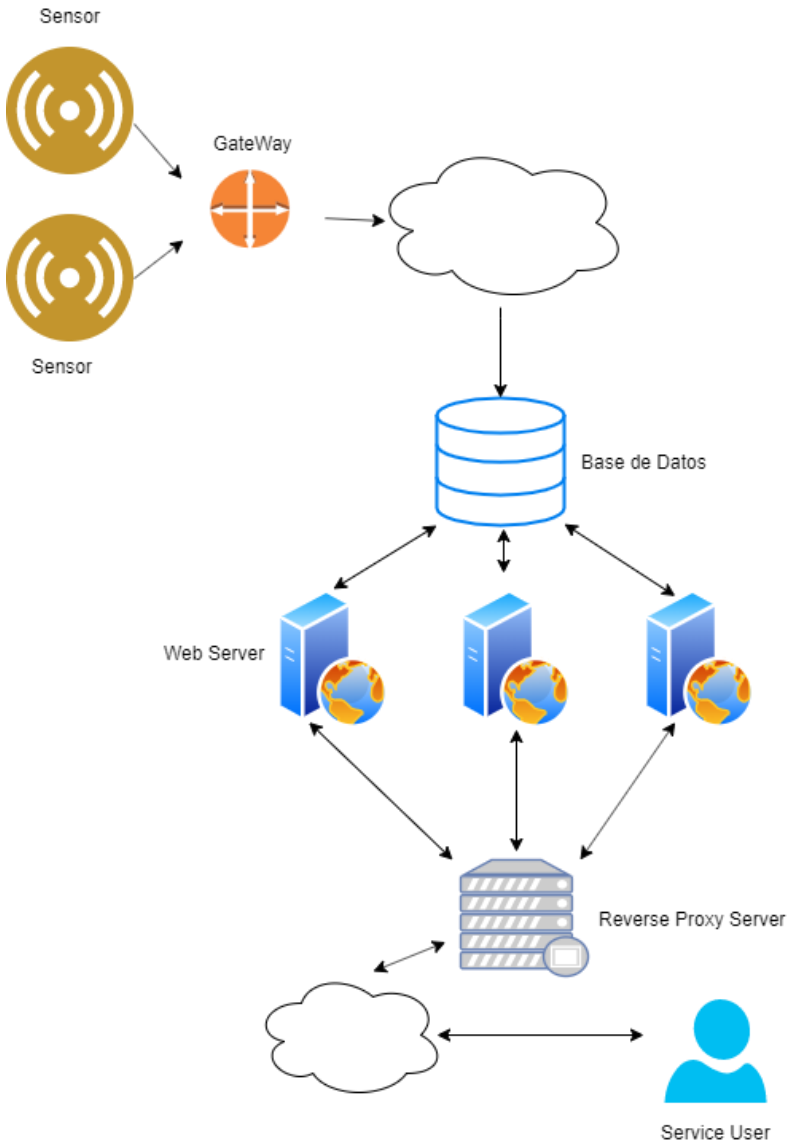


Figura 3.1 Arquitectura de alto nivel de la solución

De otro lado, se encuentra entorno web de visualización de datos en el dispositivo de usuario. Los servidores de almacenaje y procesado serán accesibles a través de interfaces web estándar.

Los servidores web intermedios entre el usuario y la información pondrán a disposición del usuario la aplicación PWA, a través de la cual los usuarios visualizarán e intercambiarán información con las bases de datos o ejecutarán acciones en función de los datos disponibles.

3.1 Plataforma de desarrollo

Para su desarrollo hemos hecho uso de las distintas alternativas que estaban a nuestra disposición.

Por un lado, hemos utilizado los lenguajes web por excelencia, HTML, CSS y JavaScript, cuyas funcionalidades citadas de izquierda a derecha son:

- HTML: Inserción de los elementos en bruto de la página.
- CSS: Creación de un diseño gráfico para los elementos introducidos por HTML.
- JavaScript: Generación de páginas web dinámicas.

De manera adicional cabe mencionar que se hace uso de PHP en un segundo plano a la hora de usar la interfaz gráfica para la gestión de bases de datos que nos proporciona phpMyAdmin [17].

Como herramientas para dar soporte a la aplicación y facilitar su montaje tenemos Node.js [18] que nos proporciona un compilador de código JavaScript del lado del servidor, también tenemos un servidor Apache [19] para proporcionar un intermediario entre la aplicación y los servicios ofrecidos y MySQL Server(SQL, Structured Query Language) [20] para almacenar nuestra base de datos.

3.2 Primeros pasos

La primera tarea realizada al comenzar a realizar el proyecto fue familiarizarse con la filosofía de desarrollo y despliegue de las aplicaciones PWA. Para ello, se define un entorno controlado de pruebas que nos permita comprobar de manera gráfica el diseño y las funcionalidades de una PWA. Este entorno básico a configurar consiste en un servidor propio que aloja el código fuentes de la aplicación PWA.

3.2.1 Instalaciones y desarrollo.

El primer paso que acometer a la hora de un desarrollo web es disponer de un servidor que sirva el contenido y realice el procesado que se tenga que realizar en remoto. Aunque lo usual es desplegar servidores web tradicionales (Apache o Nginx), algo que se hará para dar soporte a la aplicación final, en esta primera fase de aprendizaje se ha preferido emplear un servidor que se ejecuta en el propio navegador y así facilitar la ejecución de la solución básica.

El servidor Chrome Server [21] es fácilmente instalable como una extensión del navegador. Permite seleccionar un directorio y abrir un puerto en nuestra dirección local a través de la cual servir el contenido alojado en el directorio seleccionado. En la Figura 3.2 se muestra la ventana de configuración del mismo. Aunque no se requiere para esta demo también permite convertirlo en un servidor accesible desde la red local del dispositivo anfitrión.

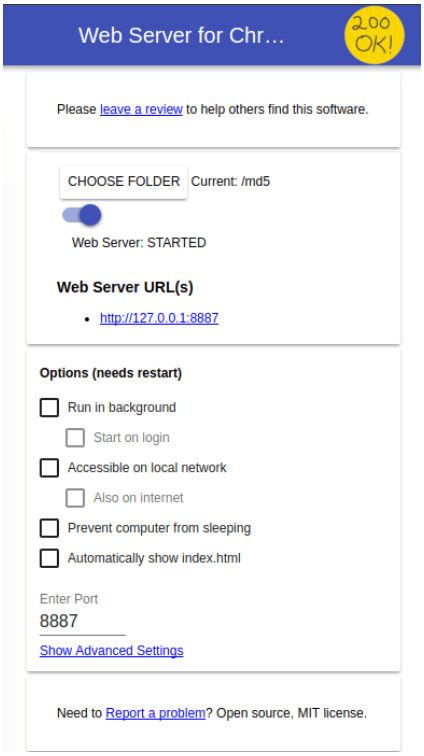


Figura 3.2 Interfaz Chrome Server

Será en el directorio seleccionado donde se alojará el código de la aplicación HelloWorld. La aplicación como tal, además de con los documentos HTML, CSS y JavaScript, cuenta con un manifiesto y el ServiceWorker, que son lo que la permiten diferenciarse de una página web común y permitirla ser una PWA.

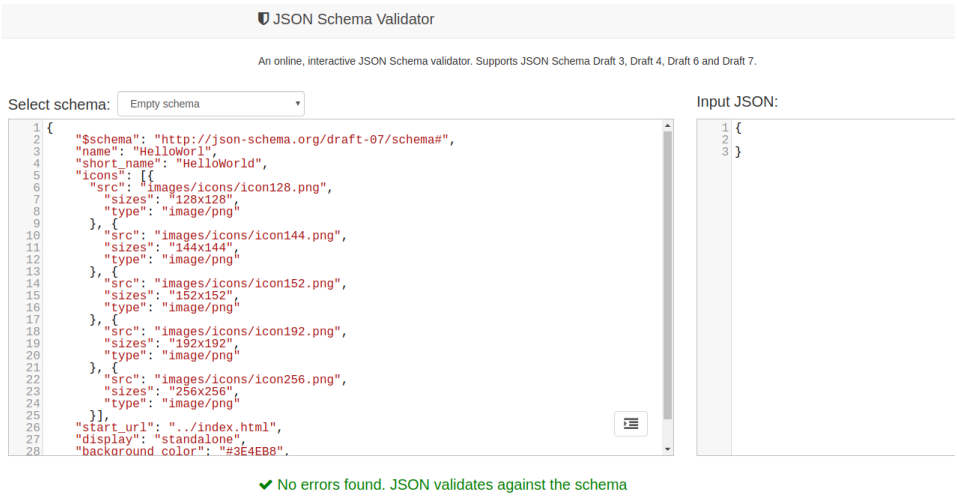


Figura 3.3 Manifiesto de la aplicación HelloWorld

En la Figura 3.3 se muestra una sección del manifiesto vinculado a la aplicación HelloWorld, cuyo formato debe ser validado sintácticamente para descartar problemas a la hora de acceder a la solución.

Antes de publicar la aplicación PWA, se debe validar que la aplicación cumple con los requerimientos para un correcto funcionamiento en términos de accesibilidad, rendimiento, etc. Al igual que en el caso anterior, el navegador dispone de una extensión denominada

Lighthouse [22], la cual es una aplicación de código abierto que permite auditar nuestras páginas de manera automatizada, haciéndola pasar por numerosas pruebas, que nos permiten a través de los resultados obtenidos mejorar las capacidades de nuestra aplicación. Entre los aspectos que se auditan se encuentran:

- Rendimiento: capacidad de la aplicación para funcionar sin errores en la ejecución.
- Accesibilidad: disponibilidad de elementos estándar de visualización, posibilidad de internacionalización, etc.
- Mejores prácticas: uso de los elementos más adecuados para servir y mostrar una página, como protocolos seguros o componentes actualizados.
- SEO: optimización del código para que la aplicación sea fácilmente indexada por los buscadores. Un ejemplo de ello sería el uso correcto de metadatos.

En la Figura 3.4 se muestra el resultado de la ejecución de la auditoría de la aplicación HelloWorld. Las puntualizaciones muestran la solvencia o capacitación en cada una de las categorías descritas anteriormente.

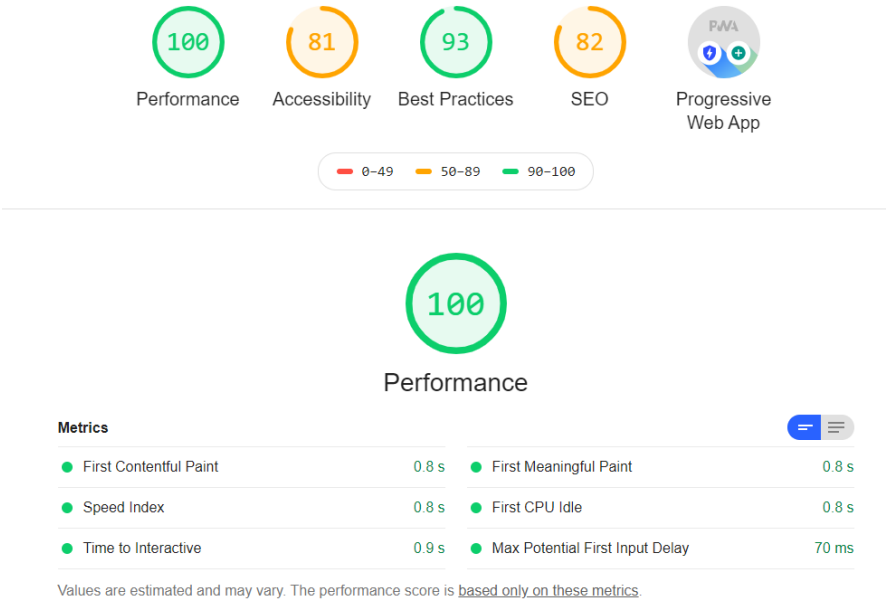


Figura 3.4 Resultado de la evaluación de la aplicación HelloWorld

Los resultados obtenidos son, en cierto modo, los esperados. Una aplicación tan sencilla y simple no incurre en retrasos y sus tiempos de carga son mínimos. En términos de accesibilidad, no se obtiene la puntación máxima puesto que no se ha implementado la internacionalización y hay limitaciones en cuanto a la semántica de la página. Aunque se ha logrado una puntación máxima en rendimiento, no se están aplicando las mejores prácticas en cuanto a la forma de servir la aplicación. En este sentido, el servidor empleado opera sobre HTTP 1.1 y el óptimo sería HTTP 2.0. Del mismo modo, el contenido de la aplicación no incluye los elementos y metadatos que permiten el posicionamiento de la aplicación en buscadores y, de ahí, que no se obtenga la puntación máxima.

3.2.2 Ejecución de la Demo

La aplicación como tal, además de con los documentos HTML, CSS y JavaScript, cuenta con un manifiesto y el service worker, que son lo que la permiten diferenciarse de una página web común y permitirla ser una PWA.

Una vez hemos depurado el código con las aplicaciones previamente mencionadas, podemos proceder a la ejecución de la misma.

Iniciamos el servidor que al ser llamado con la URL que aparece en la figura (la de Chrome server) manda a través del navegador los documentos de la app al usuario. En la etapa inicial de la petición el navegador leerá el manifiesto con la información de la página y procederá a la instalación del service worker que de administrará la memoria Cache.

Una vez se ha comprobado que el código de la aplicación PWA es apto para ser servido, se procede a la ejecución de la misma.

Iniciamos el servidor y realizamos el acceso a través del navegador a la URL asociada a la aplicación. En la etapa inicial de la petición, el navegador leerá el manifiesto con la información de la página y procederá a la instalación del service worker que administrará la memoria cache.

Una vez que la aplicación ha sido descargada por el navegador, si se cumplen con los requisitos de Chrome, se notificará al usuario de la posibilidad de instalar la aplicación en local, para lo cual se creará un acceso directo y se solicitarán los permisos necesarios para ello. Esta instalación consiste en mantener de forma off-line parte del contenido o el contenido total de las páginas web a servir. Será el módulo service worker el que se encargue de gestionar el contenido de esa caché local.

Una vez instalado y disponible el acceso directo, la PWA será indistinguible de cualquier aplicación nativa del dispositivo anfitrión. En la Figura 3.5 se muestra la ejecución de la aplicación. La ventana de usuario está estructurada con una cabecera que contiene el título de la PWA y dos botones situados en el extremo derecho que permiten refrescar la página o abrir un desplegable, el cual podemos ver en la parte central.

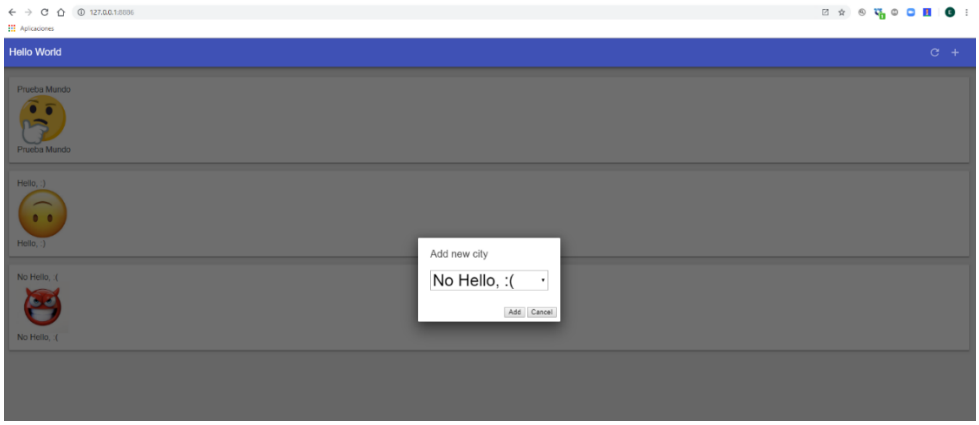


Figura 3.5 Ejecución de la aplicación HelloWorld

Dicho despegable nos ofrece la posibilidad de crear las tarjetas que vemos en el fondo de la imagen. A diferencia de lo que sería una página web tradicional, en la que la información de usuario (en este caso las tarjetas que se creen) se almacenaría en servidores remotos, en este caso al estar implementada como una PWA la gestión de la memoria y almacenaje se realiza de

forma local mediante LocalStorage API. Así una vez creadas las tarjetas, se puede disponer de disponer de ellas una y otra vez sin que tengamos que volver a generarlas al volver a entrar a la aplicación o tener que realizar solicitudes a servidores remotos.

A nivel de código, la tarea de almacenaje local y recuperación se realiza mediante un proceso en el que al abrirse la aplicación se comprueba la memoria almacenada y de existir información previa la restaura. En caso contrario crea una tarjeta nueva para no mostrar la página completamente vacía. En la Figura 3.6 se muestra un extracto de dicho código.

```
app.selectedCards = localStorage.selectedCards;
if (app.selectedCards) {
  app.selectedCards = JSON.parse(app.selectedCards);
  app.selectedCards.forEach(function(city) {
    app.updateCard(city);
  });
} else {
  app.updateCard(initialData);
  app.selectedCards = [
    {key: initialData.key, label: initialData.label}
  ];
  app.saveSelectedCards();
}
```

Figura 3.6 Extracto de código de la aplicación HelloWorld

3.2.3 Conclusiones.

La aplicación HelloWorld, a pesar de su sencillez y carencias estructurales nos ofrece una visión general del funcionamiento práctico de una PWA para poder comprenderlas sin la complejidad de una página real que en su profundidad nos dificulta la visualización de los componentes básicos.

Así mismo nos proporciona un programa base para comenzar la construcción de nuestra aplicación final de gestión de aparcamiento en superficie, cuyo diseño e implementación se describen a continuación.

3.3 Desarrollo de Parkinglot

Parkinglot es el objetivo último del proyecto, validando que las aplicaciones PWA pueden ser viables en entornos reales, fuera de la publicación de noticias y artículos como es el caso de uso más común actualmente.

El desarrollo de la aplicación busca habilitar los mecanismos que permitan a los usuarios realizar todas las tareas asociadas al pago y gestión de los servicios de aparcamiento en superficie, conocido en la ciudad de Santander (Cantabria) como OLA (Ordenanza Limitadora del Aparcamiento), sin la necesidad de hacerlo presencialmente o con descargas de aplicaciones nativas que tendrían que estar optimizadas para cada dispositivo.

3.3.1 Arquitectura

Ante la indisponibilidad de acceso a la plataforma real hemos procedido a desplegar un sistema equivalente que permita emular las condiciones de comportamiento del estado de los aparcamientos y sus sensores, y las condiciones de uso del usuario. En la Figura 3.7 se muestra la arquitectura del diseño del sistema planteado.

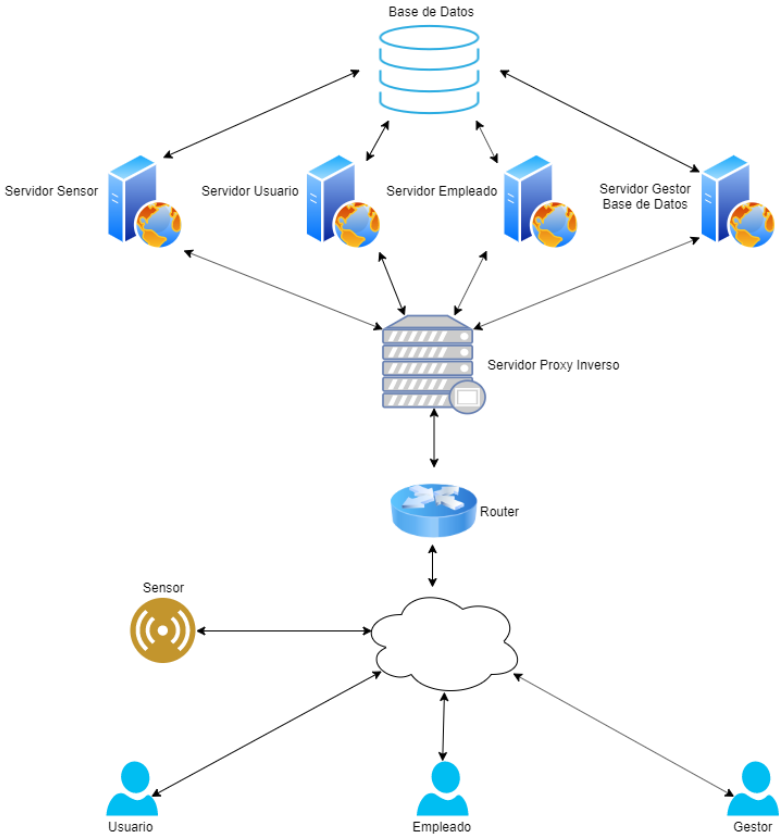


Figura 3.7 Arquitectura de la solución de gestión de aparcamiento

Entrando en el detalle de la arquitectura, se pueden distinguir, tal como se describió en la Figura 3.1, tres entornos: red de sensores de recolección de información de los puntos de aparcamiento, un entorno de soporte para el almacenaje y provisión de las páginas web de la aplicación PWA, y el entorno o dispositivo de visualización de usuario.

Aunque se podría obtenerse acceso datos reales de una red de sensores de aparcamiento en superficie, no se sería posible un entorno en que los cambios de estado se produzcan con la suficiente frecuencia como para evaluar y demostrar la aplicación diseñada. Por este motivo, hemos procedido a desarrollar un emulador de sensores basado en una solución web, que permite actualizar dinámicamente el estado de un sensor cualquiera en la base de datos asociada.

De manera adicional para facilitar el diseño e implementación se ha optado por desplegar un servidor por cada uno de los servicios que se requieren para dar respuesta a la solución. Así se dispone de entornos independientes para la emulación de sensores, para la interfaz de usuario, para la interfaz de empleado y para la gestión de la base de datos en la que se almacenarán los datos de los sensores y los resultados de la operativa de los usuarios y los empleados.

Como ya se indicó, en este caso, en lugar de emplear la extensión de servidor de Chrome empleada en la introducción, en este caso los servidores son totalmente autónomos. Se han desarrollado directamente mediante Node.js, que interpreta código JavaScript, obteniendo no solo una mayor robustez, sino también una mayor flexibilidad y facilidad de programación y desarrollo.

Será un servidor proxy inverso el que concentrará y distribuirá el tráfico según corresponda en función de la URL. Todo este desarrollo se ha realizado adaptando la configuración de un

servidor Apache para que cumpla con los requisitos. En la Figura 3.8 se aclara este concepto y se identifica quien es el usuario de cada uno de los servicios desplegados. Como podemos apreciar el usuario accede al servidor relacionado directamente con la tarea, el intercambio de información será única y exclusivamente a través de la base de datos.

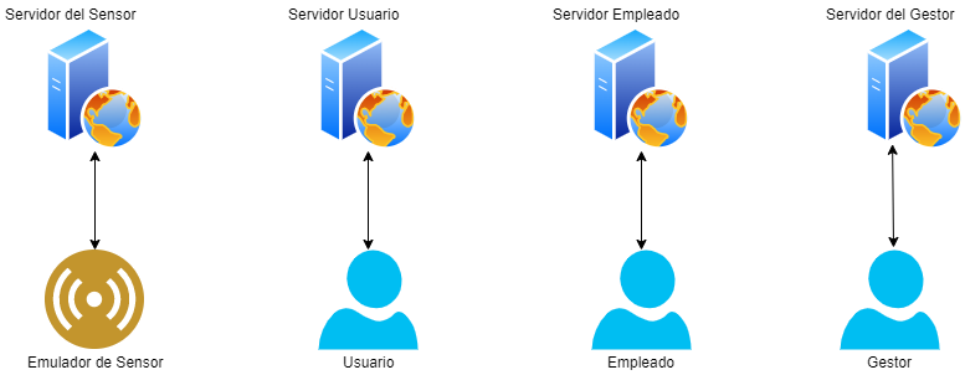


Figura 3.8 Relación destinatario del servicio-aplicación

Los usuarios mencionados pueden conectarse desde cualquier tipo de terminal, siempre y cuando este tenga la capacidad de procesamiento necesaria para soportar el uso de un navegador. En otras palabras, tablets, ordenadores y teléfonos inteligentes son indistinguibles para nuestra aplicación.

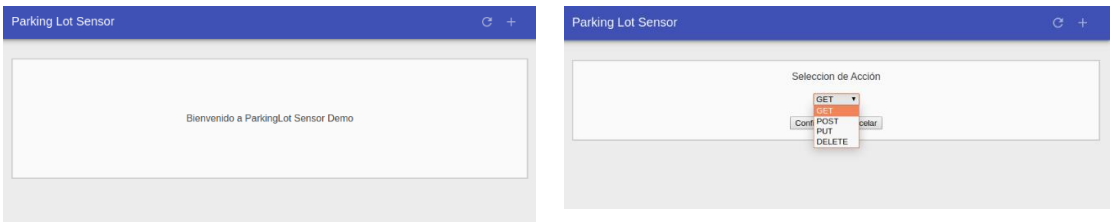
3.3.2 Implementación de los servicios

En este apartado se va a tratar la implementación y los resultados obtenidos de los distintos servicios. Todos estos servicios se han implementado sobre Node.js, como lenguaje empleando la librería Express como framework de trabajo para generar el servidor.

3.3.2.1 Servicio de emulación del sensor de aparcamiento

Como se ha comentado hemos creado un servicio de emulación de los sensores expresamente para simular una aplicación funcional. El emulador incluye los procesos necesarios para actuar sobre la base de datos que aloja la información de los sensores.

Al acceder a la aplicación del sensor se nos abrirá una aplicación con el mismo esqueleto que el usado durante la demo anterior. Este esqueleto es común para todos los servicios, aplicándosele ligeras modificaciones en algunos casos para facilitar la visualización de la información ofrecida.



a) Portada de la aplicación b) Panel de control del sensor.

Figura 3.9 Inicio de la aplicación.

En la Figura 3.9 a) se puede observar la página que nos ofrece la aplicación en el momento de acceder a esta. En la esquina superior derecha tenemos los controles de la aplicación, el botón de la izquierda reinicia la página a su estado inicial y el de la derecha despliega el panel de control.

En el panel de la Figura 3.9 b) se puede apreciar las alternativas que ofrece el emulador, puede realizar cuatro tipos de peticiones, siguiendo el paradigma RESTFul [23]:

- GET: Permite la recuperación de la información del estado de los sensores registrados, de forma individual o de todos. En la Figura 3.10 se puede apreciar un ejemplo de uso. En la parte superior se indica toda la información relativa al sensor solicitado, mientras que en la inferior se muestra un resumen del estado de todos los sensores disponibles.

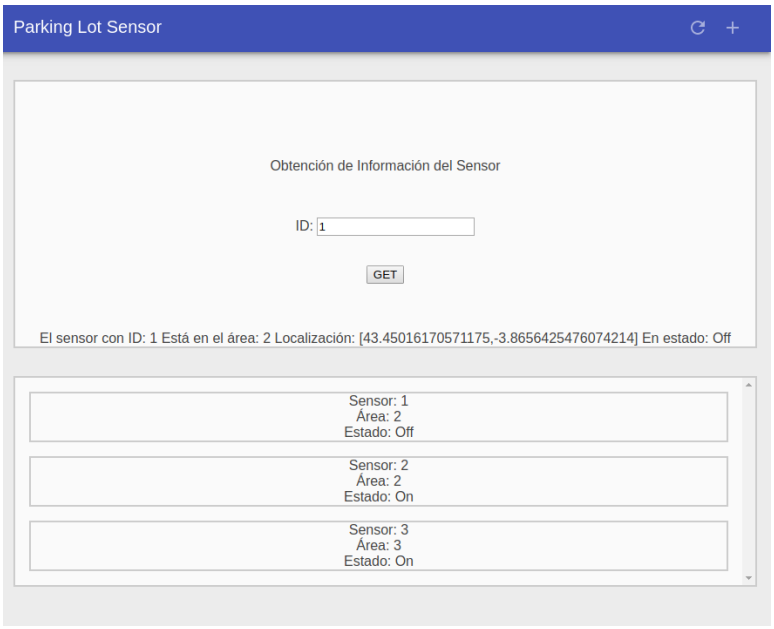


Figura 3.10 Visualización de la ventana GET del sensor

El proceso de actualización del visor de estados a nivel de programación se basa en la autoconfiguración del código HTML a través del código JavaScript. Así se generan las etiquetas necesarias para mostrar la información, y se clonan sucesivamente hasta lograr cubrir la cantidad total de elementos existentes.

- POST: Esta es la función primaria por la que se creó la figura del emulador y a través de la cual se cambia el estado de los sensores. En la Figura 3.11, vemos el funcionamiento. Introduciendo el identificador del sensor en la entrada de texto, se cambiará su estado original y la página notificará cuál es su nuevo estado.

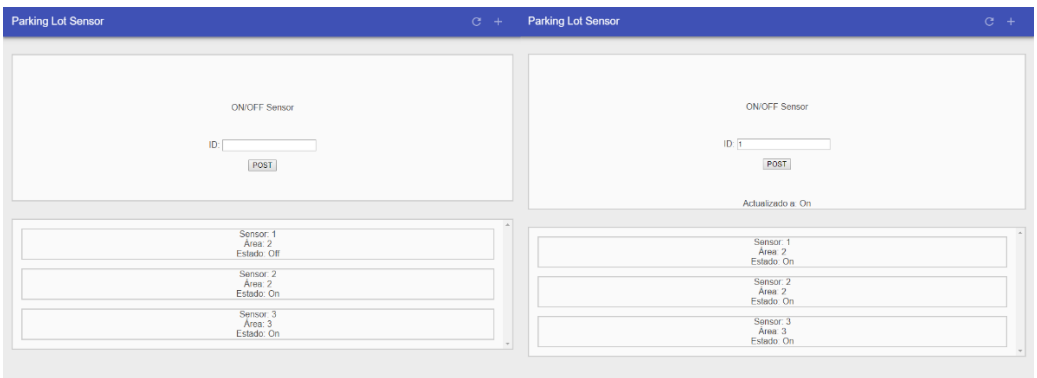


Figura 3.11 Proceso de cambio de estado

- PUT: Esta opción se utiliza a la hora de insertar un nuevo sensor a la red. Deberán introducirse todos los campos obligatorios, a saber, identificador, área de aparcamiento asociado, modelo y localización. La Figura 3.12 muestra la interfaz de usuario para este caso de uso.

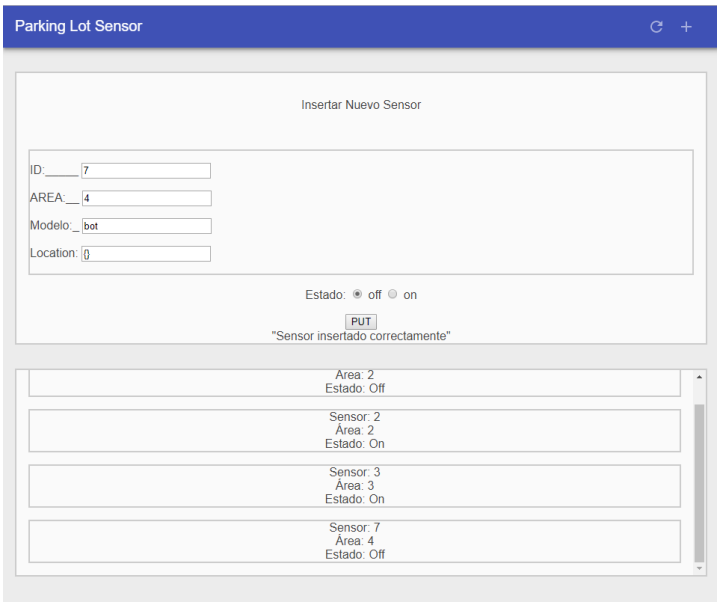


Figura 3.12 Inserción de Sensor

- DELETE: Opción mediante la cual se realiza el borrado. Como se ve en la Figura 3.13 únicamente es necesario indicar el identificador del sensor.

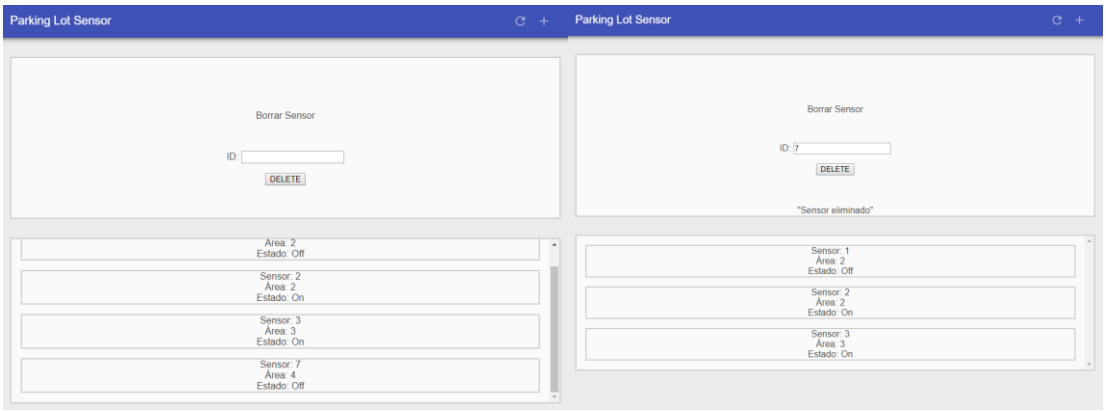


Figura 3.13 Eliminación del sensor

Con esta interfaz se logran gestionar todas las tareas necesarias para realizar la emulación de un sensor de aparcamiento.

3.3.2.2 Servicio de usuario

Este punto abarca los servicios que nuestra aplicación ofrece a los usuarios. La forma en la que se representa la aplicación es idéntica a los anteriores programas, intentando generar el menor número de cambios estéticos para dar una mayor cohesión a los servicios.

En la Figura 3.14 se muestra la ventana de operación de usuario, cuya estructura, como se ha motivado, es similar a las interfaces de los casos anteriores. Dicha figura nos muestra un mapa que refleja todas las áreas de aparcamiento y pago registradas en la base de datos como un

coche en el centro del polígono. Cada uno de estos polígonos aparecen dibujados en diferentes colores, según el porcentaje de ocupación de las áreas.

- Azul: ocupación menor al 10%
- Naranja: ocupación inferior al 70%
- Rojo: ocupación mayor al 70%

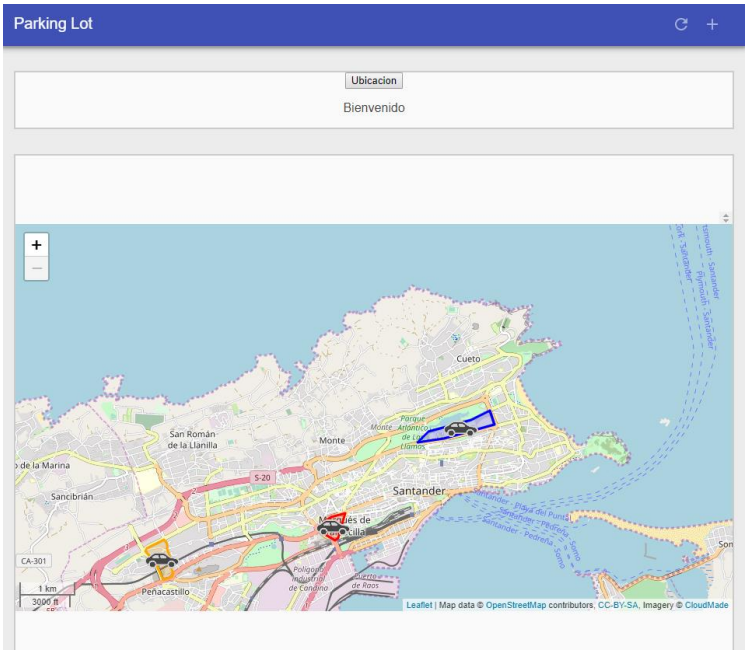


Figura 3.14 Inicio aplicación de usuario

En el supuesto de que el usuario estuviera interesado en conocer la cifra exacta de plazas y su estado, pulsando sobre la zona coloreada se abre un desplegable con los datos, como se ve en la Figura 3.15.

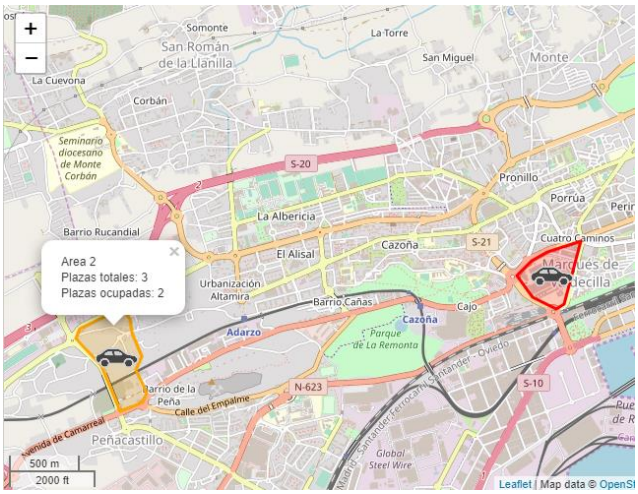


Figura 3.15 Información de estado de áreas de aparcamiento

Para facilitar la operativa del usuario, se ha planteado el uso de la geolocalización para determinar las áreas y/o plazas de aparcamiento más próximas. En la Figura 3.14 se incluye un botón de ubicación que, empleando el API de geolocalización de HTML5 es capaz de recuperar la posición en la que se encuentra el dispositivo de usuario. Esta API permite la recuperación de esta información haciendo uso del GPS, dirección IP del dispositivo o cualquier otro elemento.

La información de localización obtenida la usaremos para, combinada con otras herramientas, estima la distancia más corta entre el usuario y el área más próxima a él, mostrándosela al usuario por pantalla. En la Figura 3.16 se muestra un ejemplo, en el que tanto la representación como los cálculos se han realizado mediante:

- OpenStreetMaps [24]: Proyecto colaborativo que proporciona mapas bajo una licencia abierta. Suele usarse como capa principal en la representación de contenido de posicionamiento.
- Leaflet [25]: Herramienta de código abierto basada en JavaScript usada para crear mapas interactivos. Permite generar una capa por encima de OpenStreetMaps para lograr un resultado amigable para los usuarios.
- Turf [26]: API basada en JavaScript dedicada a hacer análisis geoespaciales avanzados para navegadores y Node.JS. En el caso de nuestra aplicación ha sido usada para realizar los cálculos necesarios para definir distancias entre puntos y áreas.

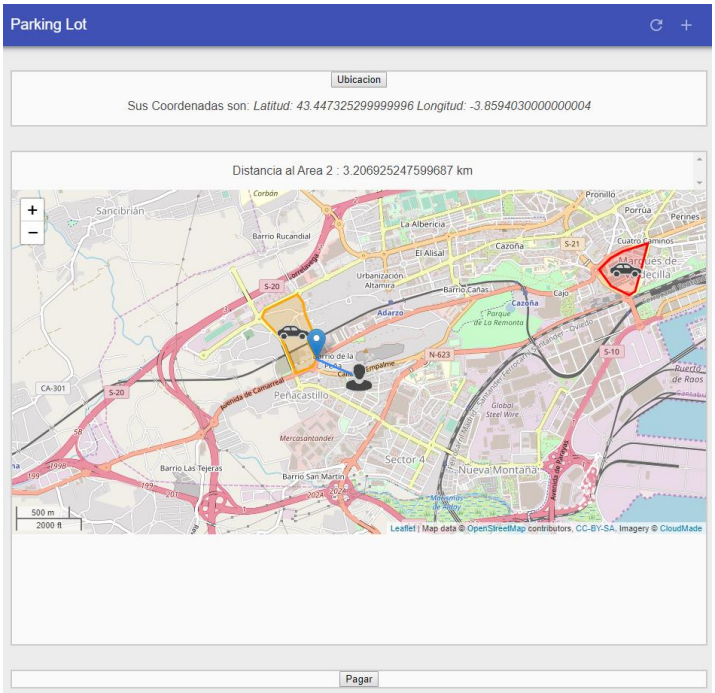


Figura 3.16 Ubicación- Distancia área más próxima

Una vez localizada la posición, estima la distancia más corta entre el usuario y el área más próxima a él, mostrándosela al usuario por pantalla. De ser toda la información correcta se procedería a iniciar el proceso de pago en la pestaña inferior.

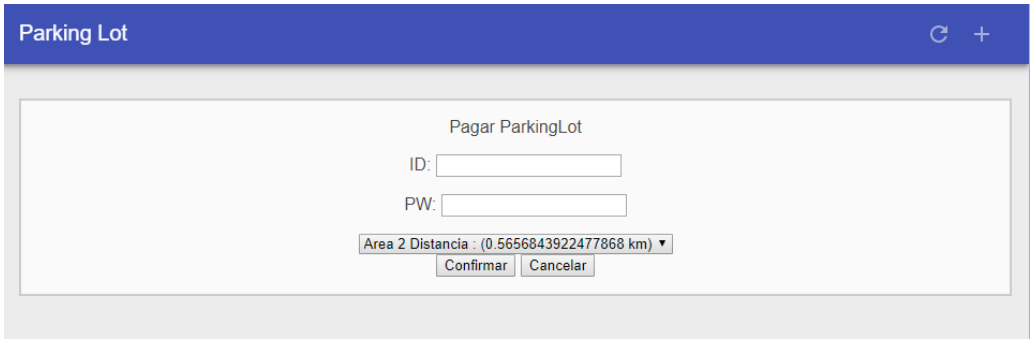


Figura 3.17 Ventana de Pago

El proceso de pago que se muestra en la Figura 3.17 consta de varios campos. Para identificar el usuario se emplea la matrícula del vehículo (se ha diseñado para que el pago vaya directamente asociado a un vehículo y no a un usuario.) y una contraseña que limitar el acceso. El último elemento es un seleccionable para escoger el área en la que se ha estacionado, por defecto se dejará preseleccionada la más próxima al usuario, pero debido a que se pueden producir inexactitudes a la hora de obtener los datos de la geolocalización siempre le damos al usuario la opción de modificar la posición detectada.

Regresando a la ventana principal, nos dirigimos al panel de control visible en la Figura 3.18.

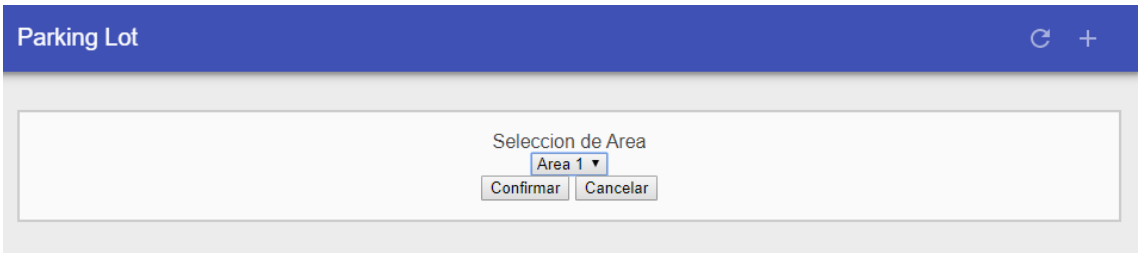


Figura 3.18 Panel de control de usuario

Este panel nos permite seleccionar un área particular entre todas las existentes, para mostrar al usuario la ubicación de esta área, los elementos de los que se componen y cuántos de ellos están disponibles, véase la Figura 3.19.

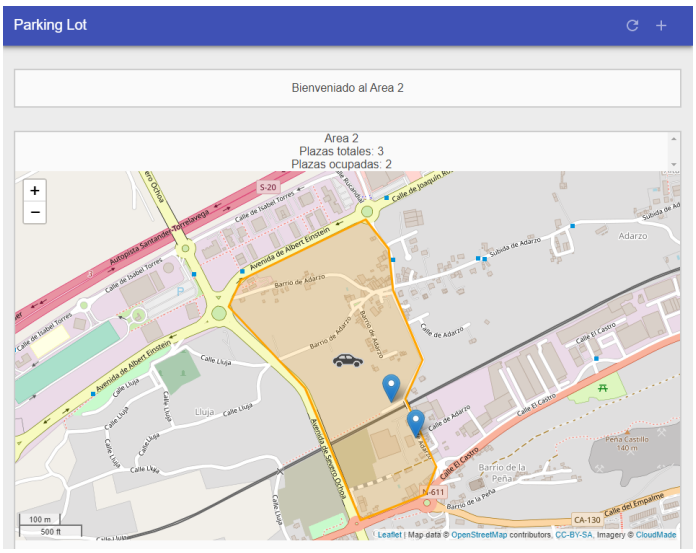


Figura 3.19 Información área 2

En esta opción vemos la posición exacta de los sensores, el identificador del área y cuantas plazas de las existentes están ocupadas, con esta información el usuario se puede hacer una idea aproximada de a qué área dirigirse para poder aparcar.

3.3.2.3 Servicio de empleado

Este servicio ha sido diseñado para cubrir las necesidades principales de cualquier empleado del servicio de la OLA, en otras palabras, es un apoyo no un sustituyente.

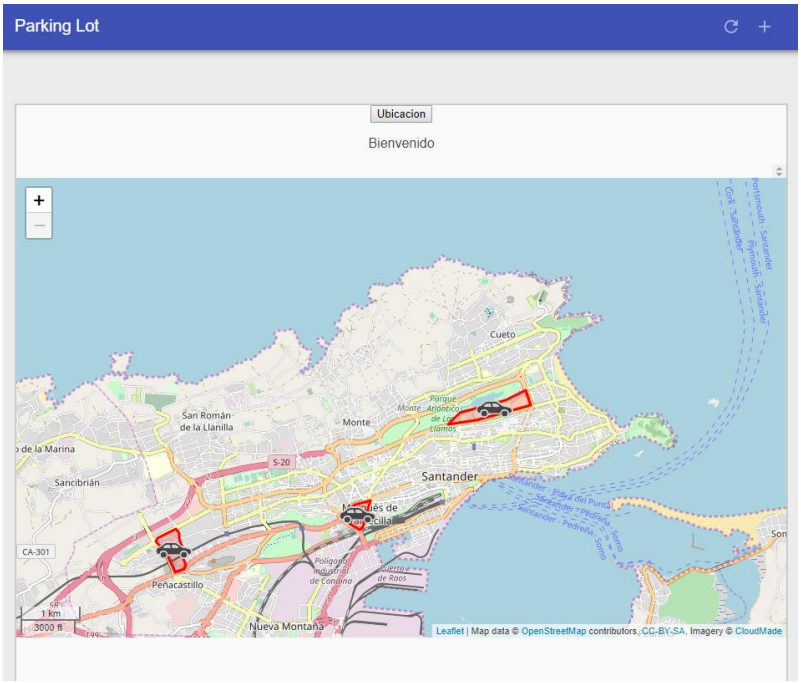


Figura 3.20 Empleado página principal

La Figura 3.20 podemos observar que tiene una estructura similar a la Figura 3.14. Como se ha comentado se trata de dar continuidad al interfaz y considerar que son herramientas similares a pesar de ofrecer un servicio distinto.

En el supuesto de que pulsemos el botón de ubicación (ver Figura 3.21), extendiéndose las capacidades de la aplicación de usuario, nos retorna gráficamente las distancias a todas las áreas, con la distancia mínima calculada en línea recta. Esta función posibilitaría al empleado u operador conocer cuál es el área más próxima. Debido a la variedad de formas que pueden tomar los polígonos, la distancia se calcula hasta el punto más cercano al área y no a su centro.

Al igual que en el caso del usuario, las áreas también se colorean de diferente manera en función de su situación. Para el empleado se mantendrán los porcentajes de corte, pero variarán las causas de estos. Para un trabajador disponibilidad de plazas no tiene tanta importancia como la cantidad de estas que, estando ocupadas, están sin pagar. Por tanto, tenemos los siguientes niveles de criticidad:

- Azul: menos de un 10% de morosidad.
- Naranja: morosidad entre el 10% y un 70%.
- Rojo: morosidad por encima del 70%.

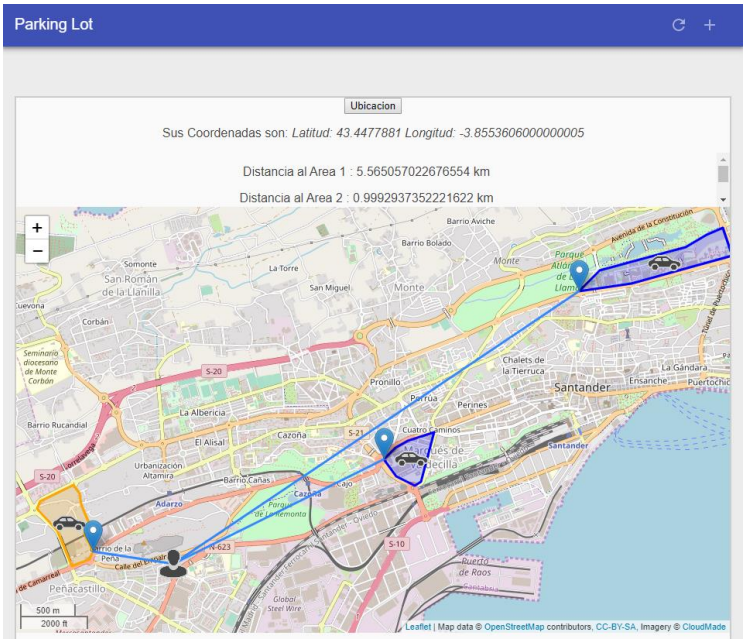


Figura 3.21 Distancia mínima a cada área

Aun siendo todas estas características atributos que facilitarían el desempeño de las tareas, la característica principal que impulsa su soporte a los trabajadores la logramos a través de una nueva API añadida solo a este servicio.

- Leaflet Routing Machine [27]: API que proporciona de manera flexible la capacidad de generar rutas estableciendo leaflet como su base.

Esta función puede ser accedida a través del panel de control, accediendo a la sección disponible para cada área, visible en la Figura 3.22.

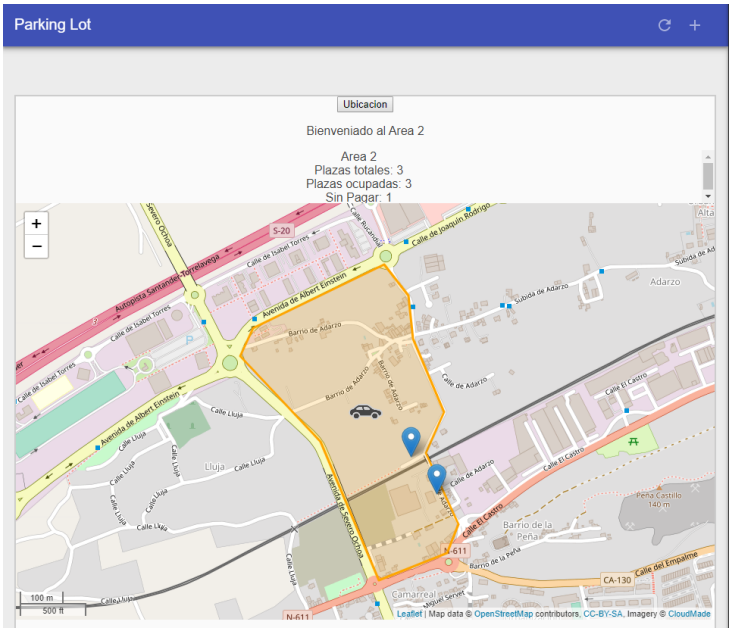


Figura 3.22 Sección del área 2

En esta parte de la aplicación, se ha añadido la opción de saber cuántas plazas están sin pagar, y el botón que activa el Routing Machine. Esta API ha sido programada para hallar la ruta más corta entre el empleado y los sensores.

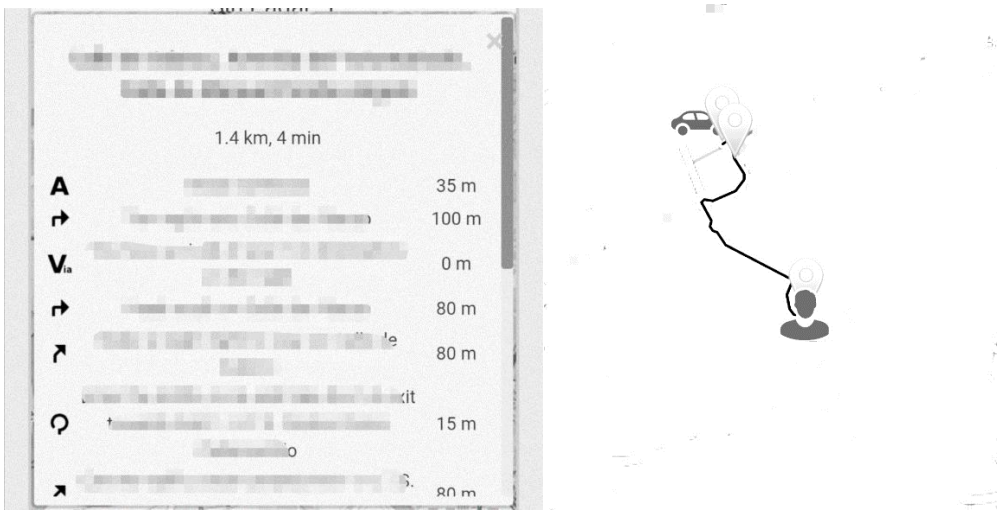


Figura 3.23 Ruta entre el empleado y los sensores

Esta función, tal como se muestra en la Figura 3.23, les proporciona a los trabajadores la ruta óptima para cumplir su objetivo. No solo es una ayuda gráfica, sino que también incluye instrucciones para prevenir la desorientación.

3.3.2.4 Auditoria de los Servicios implementados.

Haciendo uso de los servicios que nos proporciona la herramienta LightHouse hemos hecho una auditoría de la aplicación desarrollada.

A continuación, en la Figura 3.24 y la Figura 3.25 podremos ver los resultados obtenidos para las aplicaciones del sensor, usuario y empleado respectivamente. El gestor debido a ser una aplicación externa no directamente creada por nosotros no ha sido sometido a dicha evaluación.

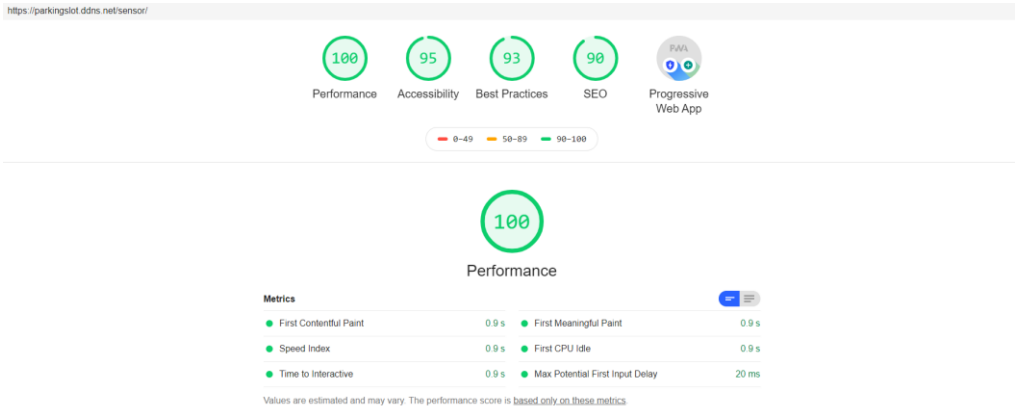


Figura 3.24 Evaluación del sensor

Observando los resultados del sensor, según la Figura 3.24, podemos extraer las siguientes conclusiones de esta evaluación:

- Perfomance 100%: La ejecución de la aplicación a nivel de código carece de errores y todas las funcionalidades que se han implementado funcionan como es debido.
- Accessibility 95%: Nuestra aplicación muestra una falta de internacionalización de los contenidos de la aplicación.
- Best Practices 86%: La plataforma desplegada no implementan protocolos de comunicación basados en HTTP2, entre otros, por lo que la evaluación total sigue siendo reducida por ello.

- SEO 82%: Puesto que se trata de una aplicación para un entono controlado no se incluyen los metadatos necesarios para ser indexada por un buscador.

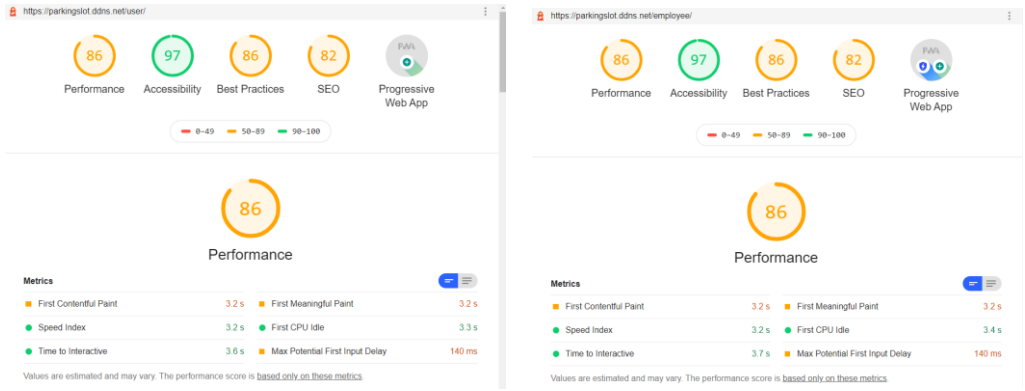


Figura 3.25 Valoración de la aplicación de usuario y empleado

Por otro lado, las aplicaciones de usuario y empleado debido a su semejanza estructural y compartir gran parte de las funciones que le dan forma, poseen resultados muy semejantes entre sí y como tal se examinarán conjuntamente. Los resultados obtenidos se muestran en la Figura 3.25:

- Perfomance 86%: Ninguna de las aplicaciones presenta fallos estructurales reales, pero la gran necesidad de información necesaria para la implementación de los mapas requiere de una gran demanda de peticiones de actualización de la información, lo que en cierta medida lastra la velocidad de la aplicación. Aun estando dentro de los rangos aceptados, no se obtiene la valoración máxima.
- Accessibility 97%: En este campo se percibe un ligero incremento debido a la integración en el código de nuevos parámetros que la fomentan.
- Best Practices 86%: La reducción en este campo sigue siendo el motivo de no usar los protocolos más actualizados, siendo el incremento de peticiones con estas características lo que reduce la puntuación comparativamente a las anteriores.
- SEO 82%: En este campo se siguen proponiendo la integración de nuevos metadatos para funcionar más eficientemente.

Aunque las aplicaciones presentan cierto margen de mejora, las características que presentan se encuentran todos en el rango más elevado de puntuaciones.

3.3.3 Gestor de información

Para la implementación del gestor de toda información que se requiere para que la aplicación opere de la forma planificada (Creación y mantenimiento de las tablas de la base de datos), se hace uso de un servicio web de control de bases de datos denominado phpMyAdmin. A través de él se ha creado y gestionado la base de datos.

Su acceso está disponible tanto en el dispositivo local como a través de la red, gracias a las redirecciones introducidas por el reverse proxy.

3.3.3.1 Estructura de la base de datos.

Debido a la importancia y protagonismo de la base de datos a lo largo del desarrollo de los servicios descritos, hemos decidido dedicarle una sección propia.

Mientras que es correcto afirmar que los servicios son totalmente independientes entre sí, la información a la que acceden es común, es decir, si uno de los usuarios de los diferentes servicios realiza algún cambio en alguno de los parámetros, esto, de manera inherente, repercutirá en las demás.

3.3.3.1.1 Esquema relacional.

A continuación, en la Figura 3.26 se muestra las relaciones entre las distintas tablas asociadas a la base de datos, por las circunstancias particulares de esta hemos decidido crear únicamente tres tablas:

- **Área**: Tabla que contiene la información necesaria para administrar correctamente las diferentes áreas. Cada área viene definida por su identificador, la secuencia de puntos (posiciones) que lo acotan, el número de sensores o plazas disponibles y cuantas están pagadas u ocupadas. En esta tabla, define una clave primaria, el identificador (`id`), que permite vincularla con los usuarios y sensores.
- **Usuario**: Tabla que contiene la información relativa a cada vehículo de usuario. Se plantea que cada vehículo pueda pertenecer a varios individuos. Incluye la información del usuario (nombre, apellidos, tarjeta de crédito y DNI). Como clave primaria se ha establecido la matrícula. La tabla incluye información del área en el que se encuentra el vehículo y su situación al corriente de pago. Puesto que un vehículo no puede estar simultáneamente en varios sitios, se ha establecido el identificador del área (id_area) como parámetro único.
- **Sensor**: Tabla que engloba toda la información relevante a los sensores (`id_sensor, área, model, location, estado`). El campo `location` contiene un documento JSON con las coordenadas en las que se encuentra el sensor. Todo sensor tendrá un identificador único y podrá estar únicamente asociado a un área.

  Es importante tener en cuenta que el estado del sensor no hace referencia al encendido o apagado, sino a la detección o no de un vehículo, ya que por norma general los sensores una vez instalados, su única desconexión sería por rotura o agotamiento de batería.

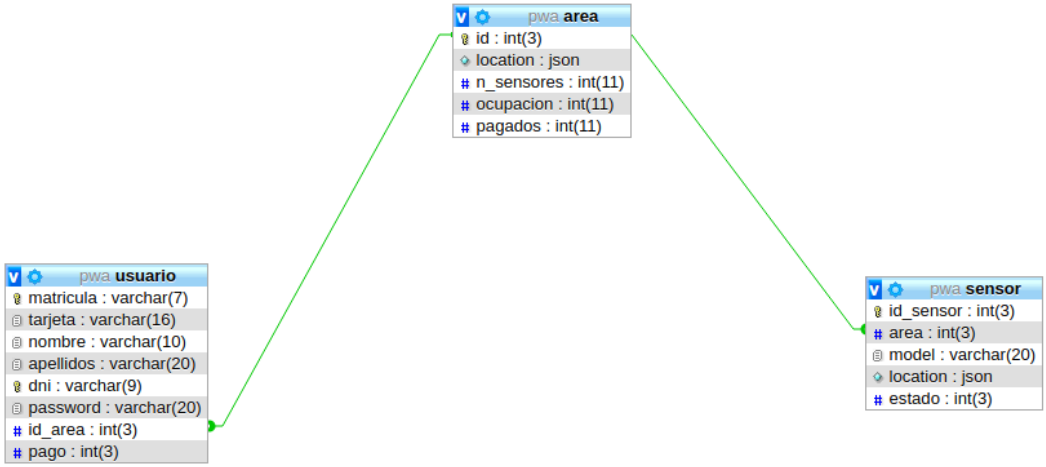


Figura 3.26 Esquema relacional de la base de datos.

Analizando los datos de la tabla en profundidad nos encontramos que se han creado dos relaciones entre las tablas. A ambas dos se les han aplicado como regla que se actualicen en cascada principalmente por un motivo, el hecho de que modifiquemos los valores de un área no

supone que esta área deje de existir; solo se está modificando su denominación y por tanto con variar el mismo valor de existir en el resto de las tablas el sistema seguiría funcionando de la misma manera.

3.3.3.1.2 Automatización de la base de datos

Definida la base de datos, se automatiza ahora la metodología para introducir y modificar los datos que en ella se incluyan. Para ello añadiremos las distintas reglas que harán posible las modificaciones de los campos de la tabla que no dependan de valores externos se actualicen automáticamente, mediante la implementación de *triggers*.

Los *triggers* que se han empleado se listan a continuación en función de los servicios ofrecidos. Analizando su nomenclatura para facilitar la visualización del momento de su activación hemos empleado las dos últimas letras para determinar cuándo y tras que comando se disparan (b=before, a=after, i=insert, u=update y d=delete).

- Inicialización de las nuevas áreas: Esta función es realizada por un solo *trigger* (`new_area_ai`) que se activa en la creación de una tabla justo tras la inserción. Su tarea es que una vez introducidos los campos de identificación y posición se establezca a cero todos los demás valores, indicando que existe ningún sensor asociado a un área antes de que esta exista.
- Establecimientos del número de sensores y de la ocupación: Estas dos tareas se realizan de manera conjunta. Todos los *triggers* que la componen son disparados por los comandos INSERT, UPDATE, y DELETE tanto antes como después de que actúen. Subdividiendo las tareas nos encontramos con dos grupos básicos, los que captan la información y los que hacen uso de ella, alterando el contenido de alguna de las tablas. En la Figura 3.27 podemos observar el proceso de borrado de un sensor ocupado y como se modifica la tabla `area` en consecuencia, reduciendo el número de sensores totales y el número de sensores ocupados.

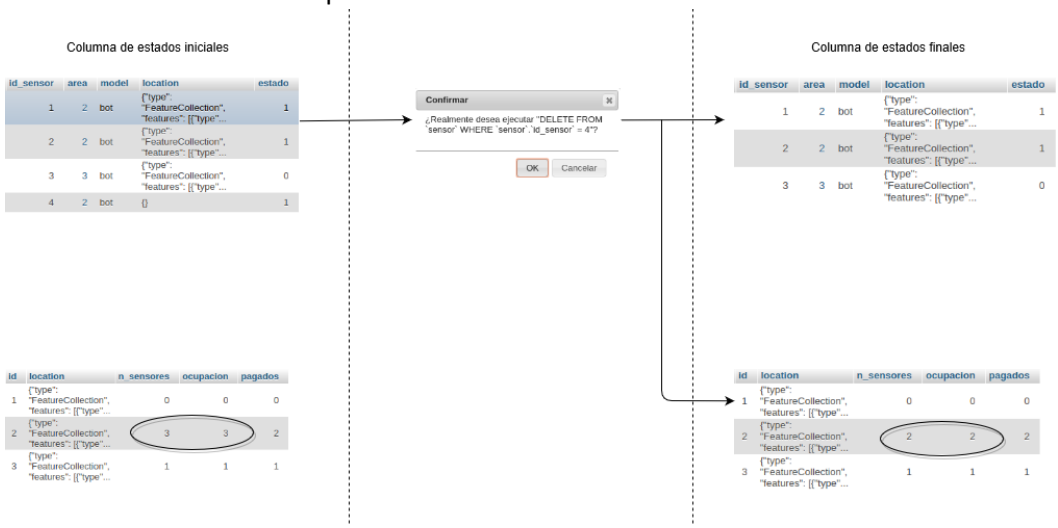


Figura 3.27 Proceso de eliminación de un sensor ocupado.

- Variación en el número de sensores pagados: Este proceso se dispara desde la tabla de usuario, donde se almacena si el vehículo ha pagado su estancia. En caso de producirse un cambio, se actualiza la tabla `area` donde disparará otros *triggers* que aplicaran el cambio para reflejar la nueva cantidad total. En total se cuenta para esta tarea con dos triggers de captación de información y uno de captación en la tabla de usuario, y dos de actuación que usan los datos previamente captados en la tabla de área.

Todos estos cambios permiten reducir la información que se debe transmitir a través de la red, reduciendo la información que debe gestionar la aplicación, el consumo de datos por parte del usuario y los tiempos de carga del sistema.

3.3.3.2 Tratamiento de la información.

Una vez ya especificados todos los elementos que componen nuestro sistema, vamos a proceder a mostrar como discurre la información dentro de la base de datos.

3.3.3.2.1 Nivel externo

De los servidores hacia afuera todas las aplicaciones se comunican de la misma forma con sus respectivos servidores.

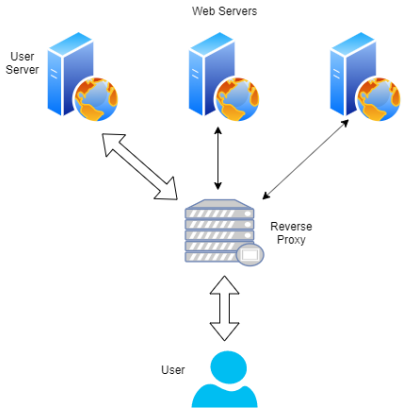


Figura 3.28 Comunicación exterior a la aplicación

Como podemos ver en la Figura 3.28, el usuario comienza la comunicación con la aplicación y es el servidor proxy el que redirige las peticiones, evitándonos el problema de tener una dirección diferente para cada servicio.

3.3.3.2.2 Comunicación Sensor - Base de Datos

La comunicación entre el Sensor y la base de datos se produce bajo cuatro casuísticas diferentes.

Figura 3.29 Obtención de datos.

La primera de ellas, mostrada en la Figura 3.29 consiste en la obtención de información de la base de datos. En este caso el servidor lanza una petición con el método GET para obtener la información referente al sensor.

Hay dos posibles situaciones en las que se hace este tipo de petición: en la primera en la parte superior de la Figura 3.29 se solicita la información relativa a un único sensor, y la segunda en la parte inferior de la misma figura es el visor de los sensores que solicita directamente información sobre todos los sensores, aunque en menor cantidad.

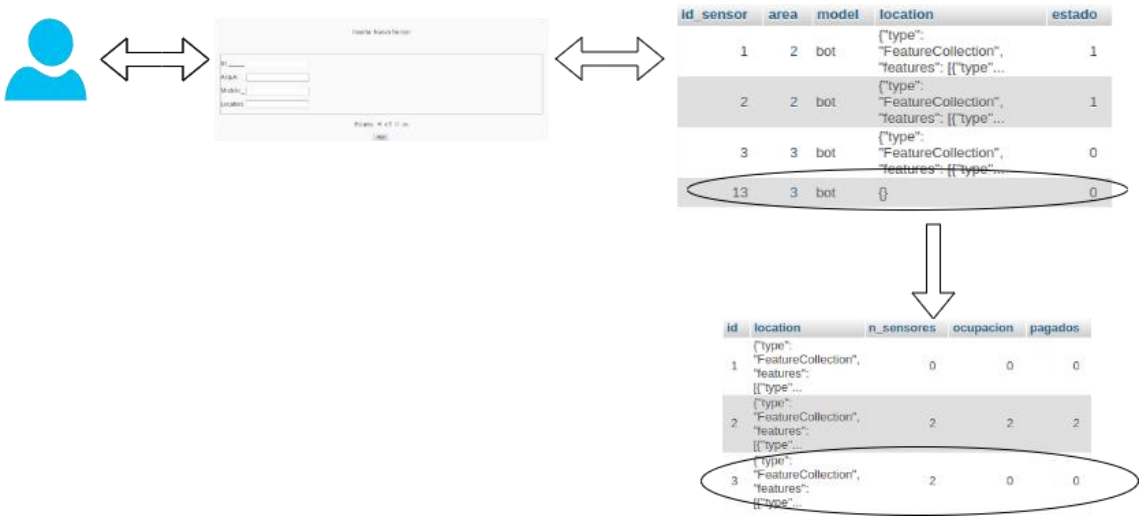


Figura 3.30 Inserción de datos

En el caso de la inserción en la Figura 3.30, el usuario introduce manualmente la información relativa al nuevo sensor. Esta información introducida en la tabla sensores, disparará los triggers que modificarán la tabla área.

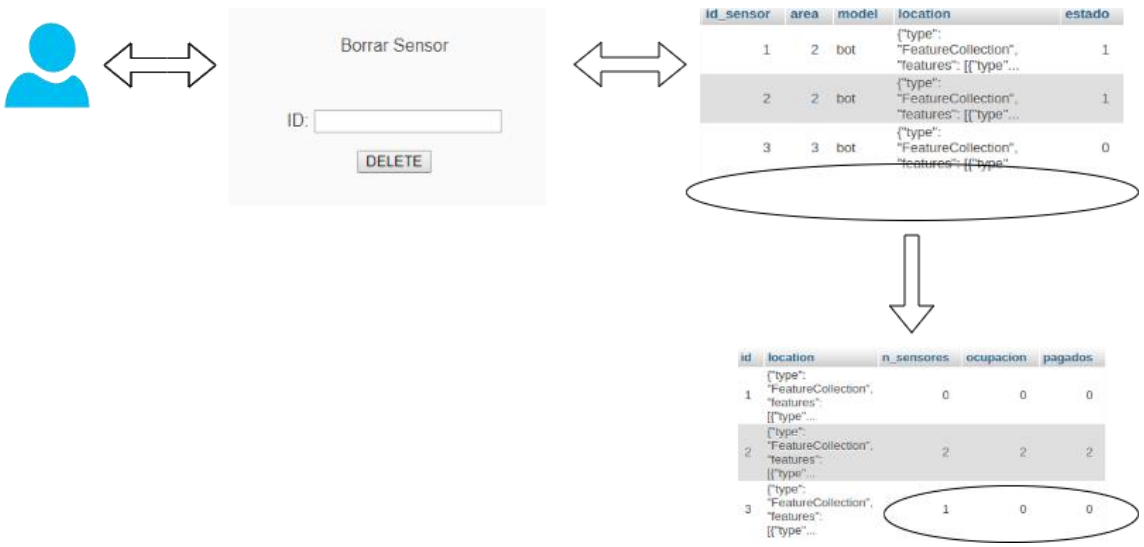


Figura 3.31 Eliminación de información.

El proceso de borrado (Figura 3.31) es idéntico al de insertado en cuanto al orden de los procesos: primero se modifica la tabla de sensor y luego los triggers ajustan los datos en la tabla area. El cambio radica en que, en esta ocasión, en lugar de añadir se sustrae un sensor, y en que solo se le requiere al usuario un valor, el de la id del sensor.

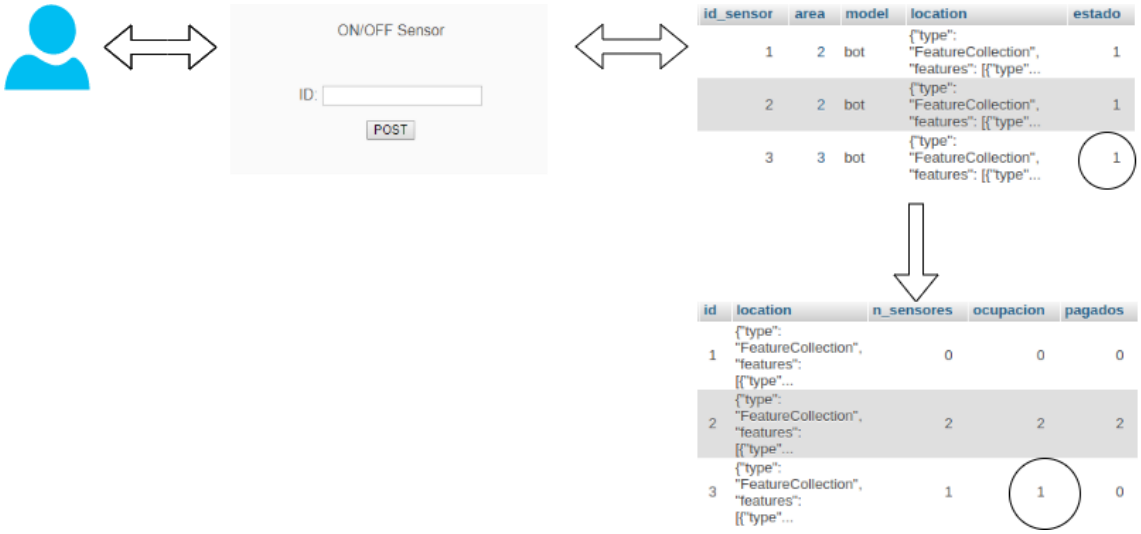


Figura 3.32 Cambio de estado del sensor.

La última de las casuísticas representada en la Figura 3.32 requiere de dos peticiones: en la primera el servidor y la base de datos se intercambian información sobre el estado actual del sensor con un GET y en la segunda fase invierte este valor para cambiarlo de detección a no detección y viceversa, siendo éste el valor que introduce mediante el método POST en la base de datos.

De manera idéntica a los casos anteriores la tabla `area` será modificada instantáneamente.

3.3.3.2.3 Comunicación de las aplicaciones de usuario y empleado con la base de datos.

Por sus semejanzas al estar ambas basadas en tareas que se apoyan en mapas, las peticiones que realizan a la base de datos son muy similares. Por tanto, las relaciones que se establecen entre estas aplicaciones y la base de datos se explicarán conjuntamente.

La primera petición que se realiza en ambas páginas nada más iniciarse la aplicación es la obtención de la información necesaria para delinear las áreas como se ve en la Figura 3.33.

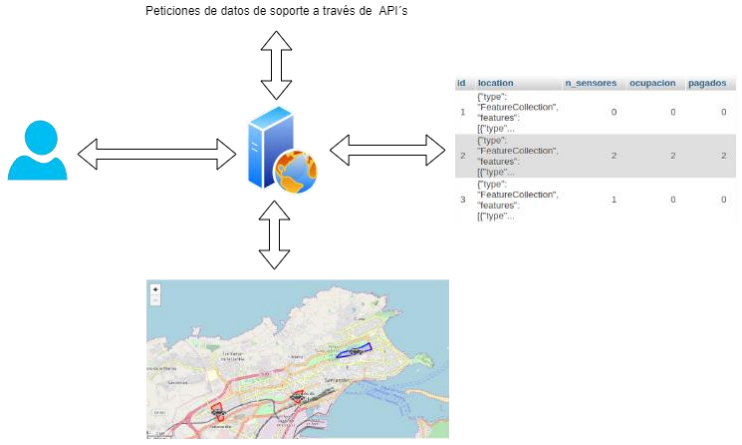


Figura 3.33 Intercambio de información para formar el mapa general

El proceso no es complejo. El mapa se crea nada más iniciarse la aplicación. Por tanto, hay que hacer una primera petición al servidor preguntando por ella. A continuación, para poder visualizar el mapa, se realiza una nueva petición a través de las API para generar un mapa sobre el que podamos trabajar y al que se le aplicaran los datos de localización recogidos de la base de datos.

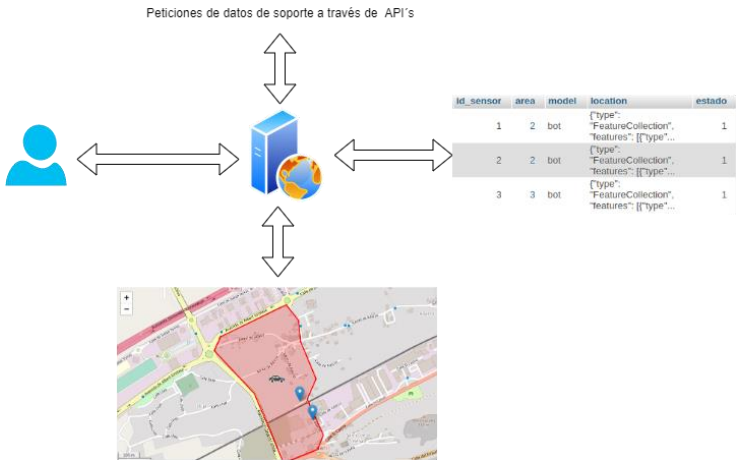


Figura 3.34 Intercambio de información para formar el mapa de área

En el caso de la Figura 3.34 se realiza el mismo proceso, solo que en esta ocasión la petición de información se la realizamos a la tabla de sensores para poder situarlos dentro del área seleccionada.

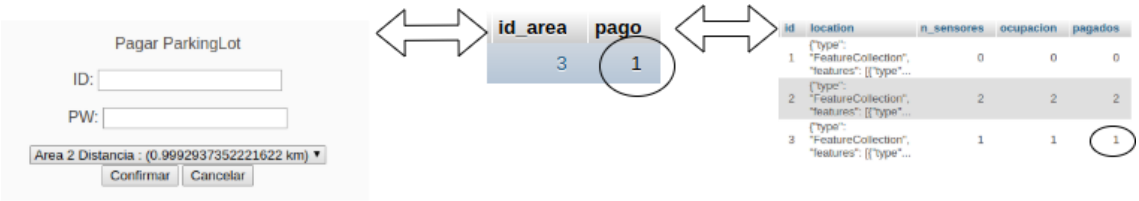


Figura 3.35 Realización de pago

El último intercambio de información con la base de datos, detallado en la Figura 3.35, nos lo encontramos única y exclusivamente en la aplicación de usuario. En esta ocasión se revisa la base de datos de usuarios para ver si coinciden con los introducidos por el usuario. De ser así se produce la actualización mediante el comando UPDATE de la misma tabla de usuarios, disparando los triggers que a su vez actualizarán los datos ubicados en la tabla area.

4 Conclusiones y líneas futuras

En este apartado vamos a proceder a realizar un análisis de las posibilidades de mejora futuras que ofrece nuestra aplicación y de los resultados logrados hasta el momento.

4.1 Conclusiones

A lo largo de todo este proyecto hemos estudiado la tecnología de aplicaciones PWA, descubriendo sus capacidades teóricas y las posibilidades que nos ofrece. Para ello tras comprender la operativa de este modelo de aplicaciones, se ha afrontado el desarrollo de una solución ejemplificativa orientada a la gestión de aparcamiento en superficie.

Como primeros pasos, hemos comenzado con la implementación de una versión de prueba, que nos ha permitido introducirnos en el entorno de las aplicaciones PWA, logrando una mayor comprensión de lo que una PWA supone. Las bases formadas durante esa primera fase nos han servido de punto de partida para desarrollar nuestra nueva aplicación con todos los servicios que esta contiene.

Tras el diseño de la arquitectura funcional de la solución se pasó a la implementación de la misma. El objetivo inicial planteaba la recolección de los datos de información de los aparcamientos a partir de sensores reales, pero en aras de facilitar el desarrollo y depuración del sistema se decidió desarrollar un servicio que emulara el comportamiento de un sensor. La operativa de este emulador sería lo más similar a la que ofrece el servicio de sensórica real, con el objetivo de facilitar una futura integración.

La información obtenida de los sensores se alojó en una base de datos con la que interactuarían las aplicaciones PWA de usuario y empleado del servicio de gestión de aparcamiento. La base de datos se diseñó teniendo en cuenta todos los campos de información que se iban a necesitar y las relaciones entre ellos.

Posteriormente se procedió a la implementación de los servicios de usuario y empleado. La interfaz planteada en ambos casos es similar puesto que muchas de las funcionalidades y necesidades de visualización son compartidas. Ambas versiones incluyen la interacción con base de datos. Su interfaz trata de dar respuesta a todas las necesidades de los usuarios, desde el punto de vista de visualización, localización, acceso a información, etc.

Se ha logrado implementar los servicios e integrar el sistema completo (emulación de sensor, bases de datos e interfaz de visualización e interacción), ofreciendo una solución PWA eficiente y funcional sobre cualquier tipo de dispositivo, independientemente de su origen. Podemos considerar este proyecto como un gran éxito tanto en el aspecto formativo como de crecimiento personal.

4.2 Líneas futuras

En este proyecto hemos desarrollado muchas funcionalidades que se podrían aplicar en un entorno real para facilitar su desempeño. Sin embargo, esto no quiere decir que este proyecto esté listo para implementarse en un entorno real. Para llegar a ese punto todavía quedan cosas que mejorar.

Entre los puntos de mejora hemos identificado se pueden incluir los siguientes:

- Integración en el sistema de una auténtica red de sensores: Este punto es fundamental ya que sin su integración ninguna gestión de aparcamientos en superficie puede ser llevada a cabo.

- Adecuación de la base de datos a entorno real: La base de datos empleada en el proyecto, aunque funcional y cumpliese con su labor a la hora de brindar soporte a nuestra aplicación, no se adecúa a todas las casuísticas posibles, por lo que se requeriría de la integración de nuevas tablas intermedias y de la modificación de ciertos campos.
- Formato de pago: Aunque nuestra aplicación cuenta con un sistema de pago, este es muy rudimentario. La aplicación necesitaría de la introducción de un nuevo sistema de pago. Entre las posibilidades existentes cabe mencionar los pagos usando NFC que, aunque todavía no sean posibles para las PWA, se les podría integrar mediante alguna aplicación nativa intermedia para hacer la experiencia de pago lo más cómoda posible para el usuario.
- Mejorar la experiencia de usuario: Nuestra aplicación cumple con su labor, pero su apariencia dista de ser suficientemente atractiva como para atraer al consumidor.
- Notificaciones Push: Las aplicaciones PWA tienen la posibilidad de, al igual que las aplicaciones nativas, enviar notificaciones al usuario. Enviar avisos con las confirmaciones de pago o del vencimiento de su periodo de estancia serían una gran mejora.
- Seguridad: Esta aplicación por los servicios que ofrece almacenaría información sensible, como direcciones, matrículas o números de tarjeta. De salir al mercado debería implementarse un sistema de seguridad acorde con la información que contiene para evitar fugas de información.

Estas son algunas de las posibles integraciones para lograr una aplicación real y, aun implementándose, al igual que cualquier tecnología, debería someterse a revisiones periódicas para que se mantenga con los nuevos avances existentes y no se quede obsoleta.

Bibliografía

[1] «intel.es,» [En línea]. Available: https://www.intel.es/content/www/es/es/silicon-innovations/moores-law-technology.html.

[2] «brandominus.com,» [En línea]. Available: https://www.brandominus.com/quien-invento-ordenador/.

[3] «redestelecom.es,» [En línea]. Available: https://www.redestelecom.es/mercado/especiales/1044666032603/ngn-quedarse.1.html.

[4] T. Berners-Lee, «w3.org,» [En línea]. Available: https://www.w3.org/History/1989/proposal.html.

[5] «applesfera.com,» [En línea]. Available: https://www.applesfera.com/app-store-1/app-store-ios-ha-generado-80-ingresos-que-google-play-que-llevamos-2019.

[6] «tss.com,» [En línea]. Available: http://www.tss.com.pe/blog/electron-un-framework-para-desarrollo-de-aplicaciones-de-escritorio-multiplataforma-primera-parte.

[7] «chromium.org,» [En línea]. Available: https://www.chromium.org/.

[8] «thinkwithgoogle.com,» [En línea]. Available: https://www.thinkwithgoogle.com/marketing-resources/experience-design/progressive-web-apps-benefit-brands/.

[9] «whatwebcando.today,» [En línea]. Available: https://whatwebcando.today/.

[10] «developpers.google,» [En línea]. Available: https://developers.google.com/web/fundamentals/architecture/app-shell.

[11] «w3.org,» [En línea]. Available: https://www.w3.org/TR/appmanifest/.

[12] «developers.google.com,» [En línea]. Available: https://developers.google.com/web/fundamentals/primers/service-workers.

[13] «developers.google.com,» [En línea]. Available: https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker.

[14] «medium.com,» [En línea]. Available: https://medium.com/pinterest-engineering/driving-user-growth-with-performance-improvements-cfc50dafadd7.

[15] «resources.mobify.com,» [En línea]. Available: https://resources.mobify.com/2016-Q2-mobile-insights-benchmark-report.html.

[16] «creativebloq.com,» [En línea]. Available: https://www.creativebloq.com/features/how-the-bbc-builds-websites-that-scale.

[17] «phpmyadmin.net,» [En línea]. Available: https://www.phpmyadmin.net/.

[18] «nodejs.org,» [En línea]. Available: https://nodejs.org/en/about/.

[19] «apache.org,» [En línea]. Available: http://www.apache.org/.

[20] «mysql.com,» [En línea]. Available: https://www.mysql.com/.

[21] «google.com,» [En línea]. Available: https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb.

[22] «google.com,» [En línea]. Available: https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=es.

[23] «notasjs.blogspot.com,» [En línea]. Available: https://notasjs.blogspot.com/2013/07/http-diferencia-entre-post-y-put.html.

[24] «openstreetmap.org,» [En línea]. Available: https://www.openstreetmap.org/about.

[25] «leafletjs.com,» [En línea]. Available: https://leafletjs.com/index.html.

[26] «turfjs.org,» [En línea]. Available: https://turfjs.org/.

[27] «liedman.net,» [En línea]. Available: http://www.liedman.net/leaflet-routing-machine/#about.