E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Aplicación web progresiva para la gestión y centralización de los trabajos de una empresa

Grado en Ingeniería Informática

Trabajo Fin de Grado

Rubén Alfredo Jiménez Echarri
José Enrique Armendáriz Iñigo
Pamplona, 9 de junio de 2020

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

AGRADECIMIENTOS

Después de un intenso período de cinco meses, hoy, al fin, escribo este apartado para dar por concluido este trabajo. Sin lugar a dudas, esto no hubiese sido posible sin mi familia que con su esfuerzo y dedicación me ayudaron a culminar mi carrera universitaria y me dieron el apoyo suficiente para no decaer cuando todo parecía complicado e imposible. En segundo lugar, me gustaría agradecer a Enrique, director de este trabajo, el esfuerzo y la ayuda que me ha proporcionado en la redacción de este documento. Sin su dedicación, la calidad de este documento hubiese sido notablemente menor. Asimismo, quisiera agradecer a todos mis amigos por brindarme el apoyo y diversión que necesite fuera de las aulas. Y por último, agradecer a mis compañeros de clase y de prácticas con los que compartí largas horas de estudio y buenos momentos durante los últimos años, sin ellos este camino habría sido mucho más duro de lo que ya fue.

A todos ellos muchísimas gracias.

RESUMEN

Este documento recoge el Trabajo de Fin de Grado del alumno Rubén Alfredo Jiménez Echarri de la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación en el grado en Ingeniería Informática realizado durante el curso 2019/2020.

La finalidad del trabajo descrito en este documento es diseñar y desarrollar una aplicación web progresiva capaz de ejecutarse en cualquier sistema operativo que posea un navegador Chrome. Además, esta aplicación implementará distintas técnicas para ser capaz de funcionar de manera normal durante los periodos de tiempo en los que no disponga de conexión a internet.

Palabras clave: Progresiva Web App, desarrollo de aplicaciones, desarrollo web, industria 4.0.

Contenido

Índice de tablas

Índice de figuras

1. Introducción

Hoy en día gran parte del uso total de internet se realiza a través del teléfono móvil. En España, concretamente, el 77,6% de las personas de entre 16 y 74 años se conectan a internet diariamente, utilizado algún tipo de dispositivo móvil fuera de la vivienda habitual o el lugar de trabajo [1]. La capacidad para poder conectarse a Internet, junto con la posibilidad de realizar diversas tareas como la grabación y edición de vídeo, realización de fotografías, reproducción de contenido multimedia, lectura de documentos, consulta y localización en mapas, gestión de eventos en agendas… en un solo dispositivo hacen de ellos un bien prácticamente necesario para el día a día. La creciente necesidad de estos dispositivos durante las últimas décadas ha propiciado que el número de fabricantes de éstos crezca y, con ello, la competencia. De ahí que los precios de estos dispositivos hoy en día, en general, son asequibles para la mayoría de la población de los países desarrollados y cada vez más para los países en vías de desarrollo y subdesarrollados.

Es por ello que, en los últimos años hemos visto un gran crecimiento en el número de aplicaciones nativas para estos dispositivos. Sin embargo, el costo de tiempo y dinero invertido para realizar los desarrollos de estas aplicaciones en todos los sistemas operativos que utilizan los usuarios a diario es bastante grande para empresas de pequeño y mediano tamaño.

En España se considera una pequeña empresa a aquellas que cuentan con una plantilla de menos de 50 trabajadores y que su volumen de negocio o su balance es igual o inferior a 10 millones de euros. Por su parte, se considera una mediana empresa a aquella con menos de 250 trabajadores y un volumen de negocio menor o igual a 50 millones de euros. Por tanto, el 99.9% de las empresas del país se consideran pequeñas o medianas empresas [2]. Si dejamos de lado a los autónomos que no poseen ningún asalariado a su cargo y que representan el 56% de estas pequeñas y medianas empresas, el 43,3% son microempresas o pequeñas empresas que poseen menos de 50 empleados y tan solo el 0,6% de estas se consideran medianas empresas con 50 o más empleados. Por lo tanto, podemos afirmar que el grueso de las empresas españolas tiene menos de 50 empleados.

Esto supone también que la facturación de estas empresas no sea desorbitada. Si nos fijamos más detenidamente en estas empresas de menos de 50 empleados, sólo el 49,9% de las empresas alcanzan los 10 millones de euros de facturación anuales y tan sólo el 17,7% alcanzan los 50 millones de euros de facturación anual. La limitación de recursos, genera que las empresas no puedan invertir el dinero que supone la creación de todas estas aplicaciones nativas dejándolas en una situación mucho menos competitiva frente a la de las grandes empresas.

Esta situación no es una situación exclusiva de las empresas españolas. Las PYMES conforman la columna vertebral de la economía de los 27 países de la Unión

Europea. Las microempresas y las pequeñas empresas son, con mucho, los tipos más comunes de PYME y representan el 98.9% de todas las empresas de la Unión Europea [3]. En oposición, las grandes empresas solo suponen el 0.2% de todas las empresas de la Unión Europea.

Esta limitación de recursos tanto financieras como de personal en la mayoría de las empresas ha generado que se haya tenido que buscar formas alternativas para poder estar presente en todos los ecosistemas tecnológicos posibles. A causa de esta necesidad, en los últimos años se han creado distintas tecnologías para poder hacer frente a este problema.

Entre ellas se encuentran las Progressive Web Apps (PWA) [4]. Estas son la evolución tecnológica de las páginas web hacia un ecosistema tecnológico más cercano al de una aplicación móvil nativa o de escritorio. Este tipo de aplicaciones web son capaces de instalarse en el sistema operativo de un teléfono móvil o de un ordenador de escritorio gracias a que se ejecutan dentro de un navegador. Además, se ven y se comportan como si fueran una aplicación móvil en los dispositivos móviles, como un programa en un ordenador o como una página web si se visualizan desde un navegador. Están diseñadas para aprovechar las características nativas de los dispositivos móviles, sin requerir que el usuario final visite una tienda de aplicaciones y son accesibles desde el navegador con una simple consulta al motor de búsquedas.

El termino PWA fue descrito en 2015 por el diseñador Frances Berriman y el ingeniero Alex Russell. Estos decidieron nombrar a este tipo de aplicaciones de este modo, debido a que, aunque estas webs en un primer momento son web normales, a medida que el usuario da permisos para agregar nuevas capacidades como poder enviarle notificaciones o estar en su pantalla de inicio, éstas *progresivamente* se convierten en aplicaciones.

Este tipo de aplicaciones web ayuda a solventar las limitaciones de recursos de la mayoría de empresas. Ahorran costes a estas al tener que desarrollar solamente una aplicación para todos los sistemas operativos populares del mercado en contra posición a las aplicaciones nativas. Además, dado que las PWA no se comparten a través de ninguna tienda de aplicaciones se consigue más control sobre la propia aplicación y la eliminación de intermediarios.

Todo esto hace que las PWA sean una tecnología interesante para las empresas de hoy en día y, en consecuencia, que la demanda de estas esté creciendo poco a poco. Este es el caso de la empresa Electricidad Ramos y Cia S.L, la cual necesita una aplicación que sea capaz de trabajar en lugares con puntos sin conexión, como son las fábricas o en exteriores, y que esté disponible en el mayor número de dispositivos posible para informatizar los partes de trabajo que realizan sus empleados. En virtud de lo cual, se describirá a lo largo de este documento el proceso de diseño y creación de una PWA real para esta empresa.

1.1. Objetivos del proyecto:

Durante la realización de este trabajo de fin de grado se intentará hacer un estudio sobre la tecnología PWA, tanto de manera teórica como práctica. Para ello, se han definido los siguientes hitos:

- Realizar un estudio teórico sobre la tecnología PWA y las tecnologías que la rodean.
- Realizar un estudio teórico de alternativas a esta tecnología.
- Desarrollar una aplicación con esta tecnología capaz de gestionar y centralizar los trabajos de una empresa.
- Realizar la implantación de esta aplicación en la empresa Electricidad Ramos y Cia. SL.

1.2. Estructura de la Memoria

Esta memoria está estructurada en ocho capítulos, según se indica a continuación:

- En el capítulo 1 se presenta una introducción de la tecnología PWA y del porqué de su uso, además de definir los objetivos y la estructura del proyecto.
- En el capítulo 2 se hace una presentación de la empresa y el trabajo que se va a realizar. Se realiza un análisis de requisitos que engloba tanto los requisitos técnicos iniciales como las tecnologías y equipos necesarios para la realización del proyecto. Por último, se presenta el estado de arte actual, tanto de los partes de trabajo, como de la tecnología PWA.
- En el capítulo 3 se muestra la planificación realizada para el proyecto, a través de una tabla de las estimaciones de tiempo en, primer lugar, y un diagrama de Gantt en segundo lugar.
- En el capítulo 4 se muestra un estudio de la tecnología PWA y de las tecnologías que la rodean.
- En el capítulo 5 se muestra el diseño de la interfaz de la PWA creada y se describe cómo se usa.
- En el capítulo 6 se profundiza más en la aplicación desarrollada y se describe cómo funciona internamente.
- En el capítulo 7 se presenta un informe de los recursos que finalmente se han destinado al proyecto y se hace una comparación con los estimados en el capítulo 3.
- Por último, en el capítulo 8 se describen las conclusiones y líneas futuras de la aplicación.

2. Visión global del proyecto

Para realizar este proyecto se hará uso de la metodología ágil Scrum. Esta metodología produce resultados parciales en cada iteración. Durante el desarrollo de la aplicación debido a: cambios en las tomas de decisiones, a mejoras o problemas hallados durante la última iteración… puede que esta visión cambie.

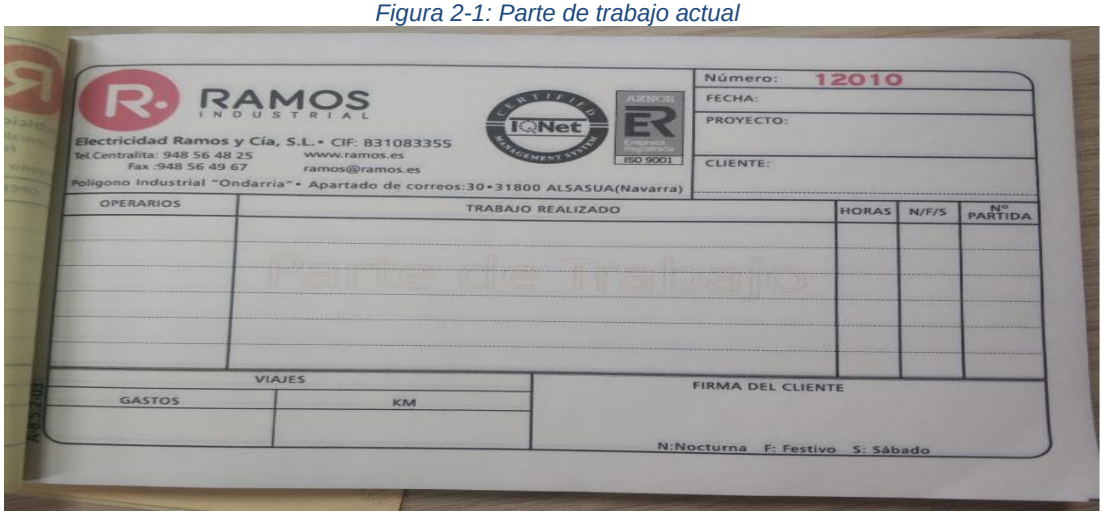
Sin embargo, para una mayor legibilidad y almacenamiento de la documentación en este capítulo se mostrará la visión global definida al inicio del proyecto de las necesidades y la solución a desarrollar. Para ello, a continuación, se realizará un análisis del problema, se presentará la solución tecnológica por la que se ha optado de entre las existentes y se hará una revisión del estado del arte en la materia.

2.1 Marco de desarrollo

El desarrollo de este Trabajo de Fin de Grado y de la aplicación se realizará en la empresa Electricidad Ramos y Cia S.L. Esta es una empresa del sector eléctrico situada en el municipio de Alsasua. La empresa trabaja dando servicio y vendiendo sus productos en un ámbito local, concretamente en la región del Corredor del Araquil (Sakana) y cuenta con más de 100 trabajadores. Aunque su actividad principal es la venta de suministros eléctricos e industriales, también posee un departamento de montaje y mantenimiento de instalaciones eléctricas. Además de, un departamento de ingeniería industrial, que se encarga de: la elaboración y redacción de proyectos, dirección de obra, tramites con organismos públicos, compañías suministradoras y consultorías. Y, por último, de un departamento de informática, en cuyo marco se hará el desarrollo de esta aplicación.

Como se ha mencionado en el párrafo anterior, una de las actividades de la empresa es el montaje y mantenimiento de instalaciones eléctricas. Al finalizar uno de estos trabajos los operarios deben rellenar un formulario denominado *parte de trabajo*.

Figura 2-1: Parte de trabajo actual



Estos formularios se rellenan a causa de que la empresa en muchas ocasiones factura una vez al mes, unificando en una sola factura todos los partes de trabajo realizados durante el mes a los clientes que presta servicios continuados. Este documento debe firmarlo siempre el cliente y queda como prueba de: las actividades realizadas por los trabajadores, el tiempo invertido en ellas, gastos que ha generado… Además es útil, también, para ayudar a la propia empresa a realizar un control de las actividades de sus empleados.

Completar estos documentos requiere bastante tiempo del trabajador, desperdicia una gran cantidad de papel y hace complicada la gestión de todos estos papeles. Es por esto que la empresa desea digitalizar estos partes de trabajo para conseguir una gestión más ágil y eficiente de estos.

La aplicación que se desarrollará en este trabajo pretende dar solución a este problema que tiene la empresa. Por este motivo, vamos a diseñar y realizar una aplicación web que sustituya el sistema anterior de partes de trabajo. Para ello esta aplicación recogerá la información necesaria para crear los partes de trabajo y distinguirá entre los partes de trabajo de averías y de órdenes de trabajo.

Se consideran averías cuando una de las máquinas o instalaciones de las que la empresa se hace cargo del mantenimiento presenta una rotura, daño o fallo que impide o perjudica el funcionamiento normal de esta. Cada una de las averías tiene un número de identificación interno que relaciona esta avería con la máquina o instalación que se ha de reparar. Esto facilita internamente comprobar el número de averías que ha tenido la maquina o la instalación en cuestión, la cantidad de tiempo y recursos que ha costado repararla…

Por el contrario se considera órdenes de trabajo (OT) a todo aquello que no sean averías. Esto incluye puestas en marcha y mantenimientos preventivos de las máquinas e instalaciones. Al contrario de las averías, todas las OT son trabajos previstos que incluso se realizan con cierta periodicidad en el caso de los mantenimientos.

En el siguiente apartado, se especificará más detalladamente los requisitos funcionales que debe cumplir la aplicación que se desarrollará en este trabajo.

2.2 Análisis de requisitos

2.2.1 Requisitos funcionales iniciales

La aplicación que se desarrollará pretende que los operarios que realizan mantenimientos y reparaciones en otras empresas puedan realizar de una forma rápida los partes de trabajo que deben firmar los clientes. Además, conseguir

centralizar y guardar todos los datos que estas generen en la base de datos interna. Para ello la aplicación debe:

- Poder distinguir entre una avería y una OT a la hora de rellenar el parte de trabajo.
- Tener campos suficientes para poder recoger toda la información que deseamos.
- Facilitar la tarea del operario lo máximo posible haciendo que todos los campos que sean posibles sean autocompletables.
- Validar los datos antes de mandarlos a la base de datos, para que no se introduzcan datos erróneos en ella.
- Tener la capacidad de crear partes de trabajo en la que se incluyan varios operarios con distintos trabajos, distintos tipos de pernoctas y dietas. Además de dar la posibilidad de indicar el desplazamiento hasta el lugar en varios coches.
- Mostrar una tabla resumen de todos los datos para que el cliente pueda visualizarlo en una única pantalla como un parte de trabajo.
- Tener una forma de firmar de manera manual en la pantalla del dispositivo y que esta se quede guardada como imagen en los servidores. Además, no debe permitir el envío de la información hasta que la firma sea hecha.
- Ser capaz de hacer todo lo anterior incluso sin conexión a internet.

Conjuntamente con todo lo anterior se deberán diseñar y crear las tablas necesarias para guardar toda esta información y añadirlas a la base de datos.

2.2.2 Tecnología y equipo

En esta sección se van a comentar las diferentes tecnologías y herramientas empleadas en el desarrollo de este proyecto; además del equipo humano detrás de él.

A modo de resumen, las tecnologías y herramientas que vamos a utilizar son las listadas a continuación:

Tecnologías:

- PWA
- LAMP
- Laravel
- Bootstrap

Herramientas:

- Visual Studio Code
- MySQL Workbech
- Git
- Ubuntu 18.04 LTS (Servidor)
- Microsoft Windows 10 (Equipo)

Seguidamente, salvo la tecnología PWA de la cual se hablará en profundidad en el capítulo 4 (Estudio de la tecnología PWA), se hará una descripción más detallada de cada una de estas herramientas y tecnologías.

LAMP:

LAMP es el acrónimo generado por el grupo de herramientas de software libre Linux, Apache, MySQL y PHP. Se ha hecho uso de todas estas tecnologías para crear la infraestructura que posibilita la realización de solicitudes dinámicas a nuestra aplicación. Su funcionamiento es simple. Linux sirve como sistema operativo base para ejecutar el servidor web Apache. Este último realiza las conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente que son gestionadas mediante el lenguaje PHP en el servidor. Por último, Mysql permite generar la base de datos que da persistencia a nuestra información y con la que podemos trabajar.

Se ha elegido usar esta infraestructura debido a que la empresa ya posee una base de datos Mysql con la que se trabajara como base, ejecutándose en un servidor propio bajo el sistema operativo Linux. Concretamente con la distribución Ubuntu 18.04. Esta distribución ya posee en sus repositorios Apache, que es el servidor web con el que se está trabajando en la intranet de la empresa. Y por último, PHP puesto que es el lenguaje con el que más familiarizado estoy de entre los que conforman normalmente esta arquitectura (PHP, Perl, Python). Es por todo esto, junto con el margen de tiempo escaso para poder aprender otras nuevas tecnologías, por la que se ha optado por esta arquitectura y no otras como podría ser MEAN.

Laravel:

Dado que se ha elegido PHP como lenguaje de programación del lado del servidor, se ha decidido usar Laravel para ayudarnos en nuestro trabajo de programación. Laravel [5] es un framework PHP gratuito y de código abierto destinado al desarrollo de aplicaciones web siguiendo el patrón modelo, vista, controlador (MVC). Entre sus características principales está el motor de plantillas Blade que nos facilita usar código PHP en las vistas para ayudarnos a escribir la menor cantidad de código repetido posible. Un sistema de enrutamiento de nuestra página web y Eloquent un ORM (Object-Relational Mapping) para manejar de forma más sencilla los procesos correspondientes al manejo de bases de datos en el proyecto.

Es por toda esta cantidad de características, junto con, una buena documentación y la facilidad para resolver problemas en virtud del número de desarrollos implementados con este framework que se ha elegido Laravel por encima de otros como Symfony, CakePHP o CodeIgniter entre otros.

Bootstrap:

Bootsrap [6] es otro framework de código abierto creado por la empresa Twitter. Orientado al desarrollo front-end, este, sirve para crear diseños de aplicaciones web gracias a la gran cantidad de elementos de diseño que posee como, por ejemplo, formularios, menús, botones, ventanas modal…

Se ha escogido este framework para crear los diseños de la aplicación dado que es una herramienta ampliamente conocida en el desarrollo de aplicaciones web, ahorra una gran cantidad de tiempo a la hora de diseñar páginas y ya se poseía experiencia previa con este framework,

Visual Studio Code:

Visual Studio Code [7] es un editor de código multiplataforma gratuito desarrollado por Microsoft. Incluye soporte para la depuración y control de versiones debido a que posee Git ya integrado.

Se ha elegido desarrollar la aplicación con este editor dado que es un editor altamente personalizable, bastante ligero, y que posee una cantidad enorme de plugins, muy útiles a la hora de desarrollar. Entre ellos se puede destacar el plugin de SFTP que se ha utilizado para editar directamente los archivos del servidor de pruebas desde el ordenador personal y de esta forma no tener que configurar nada en este último.

MySQL Workbench:

MySQL Workbech [8] es una herramienta visual gratuita para la administración, gestión y mantenimiento de bases de datos basadas en MySQL. Creada por Oracle, los mismos desarrolladores de MySQL, se trata de una herramienta visual muy completa que además es multiplataforma. Dispone de un editor de consultas y un módulo dedicado a la optimización de rendimiento entre otros.

La elección de este gestor de bases de datos es debida a que se trata de una herramienta gratuita que además, satisface todas las necesidades que requiere nuestro proyecto para gestionar la base de datos y, por último y no menos importante, porque es la herramienta que usan los compañeros del departamento para estas tareas.

Git:

Git [9] es un software de control de versiones gratuito diseñado por Linus Torvalds. Este está pensado para garantizar la eficiencia y la confiabilidad del

mantenimiento de versiones de aplicaciones con un gran número de archivos de código fuente.

El control de versiones se define como la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Para posibilitar este control existen diferentes programas software como Git.

Se ha elegido Git como nuestro software para el control de versiones por el simple hecho de que es el software más extendido, se tenía experiencia previa usándolo y es el software de control de versiones que usan los compañeros de departamento.

Por último se mostrará el equipo humano a cargo del proyecto:

Equipo del proyecto:

Alumno, encargado de realizar el proyecto:
Rubén Alfredo Jiménez Echarri
Director del TFG, encargado de la supervisión del documento:
José Enrique Armendáriz Iñigo
Tutor de la empresa; encargado de la supervisión de la aplicación:
Imanol Mosqueira Barriola

2.3. Estado del arte

El objetivo de este apartado consiste en dar información sobre el estado del arte de los partes de trabajo y las PWA, buscando información sobre soluciones actuales ya disponibles al público.

Por un lado, en cuanto a las PWA existen actualmente varias empresas que hacen uso de esta tecnología como valor añadido para satisfacer e intentar obtener un mayor número de clientes. Entre ellas se encuentran Twitter, Uber, Instagram, Financial Times, Forbes… y otras muchas. Sin embargo, para no hacer demasiado extenso este apartado nos centraremos en la PWA de Uber.

Uber es una empresa internacional que proporciona a sus clientes vehículos de transporte con conductor (VTC), a través de su web y su aplicación. Dado que opera en la mayoría de países del mundo, Uber, necesita que su servicio sea accesible a la máxima cantidad de personas posible, independientemente de la velocidad de conexión o el dispositivo desde el que estén accediendo.

Aunque Uber ya contaba con aplicaciones Android y IOS nativas. En 2017 decidió lanzar una PWA para aprovechar una plataforma con muchos años de desarrollo y altamente probada como es la web [10]. El uso de la web les ayuda a llegar a un público más amplio, incluidas personas que pueden vivir en regiones donde el acceso

a la red es lento o donde los teléfonos basados en tecnología más antigua son más comunes.

La PWA de Uber ofrece una experiencia similar al de una aplicación nativa. El núcleo de esta (la parte esencial de la aplicación que permite solicitar un viaje) ocupa tan solo 50kB de espacio frente a los 153MB de la aplicación nativa. Lo que significa que puede estar lista para su uso en tan solo 3 segundos en redes 2G convencionales (250kB/s, 300ms de latencia) algo imposible para las versiones nativas.

Figura 2-2: Instalación PWA de Uber

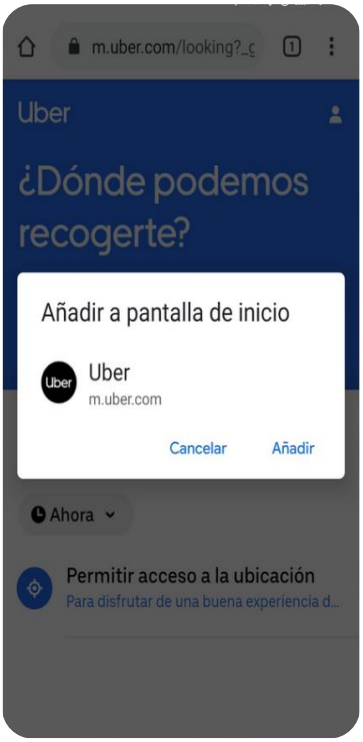


Figura 2-3: Vista conexión lenta

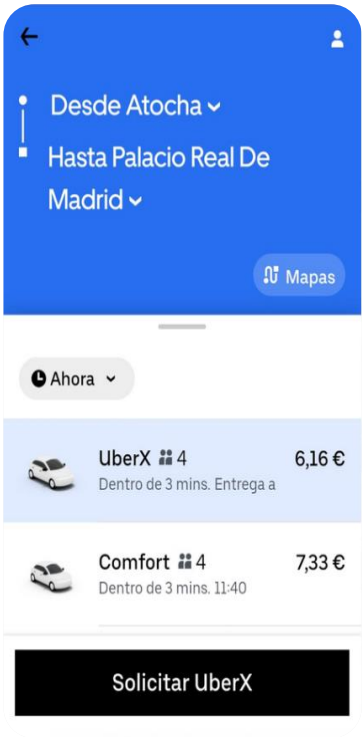
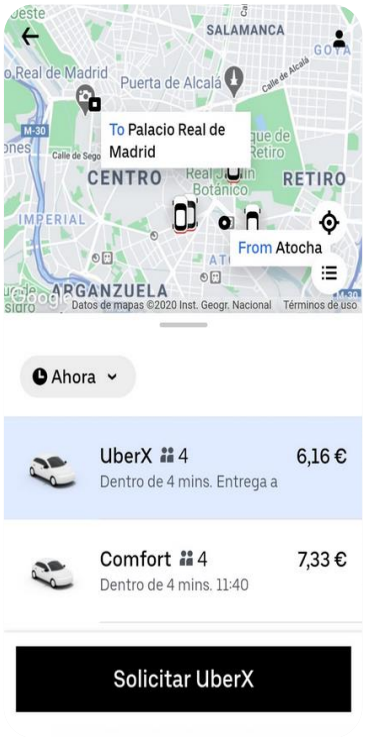


Figura 2-4: Vista conexión normal



Tras su implantación, la empresa realizo diferentes encuestas y en ellas se descubrió que los nuevos usuarios preferían usar la web, en la que no se requería ningún tipo de instalación, y posteriormente descargarse desde ella la PWA si estaban satisfechos antes que la opción de descargarse la aplicación nativa desde el primer momento [11].

Además, los usuarios que usaban la web por primera vez se convertían en clientes habituales un 10% más a nivel global que los usuarios que usaban la app por primera vez. En países en vías de desarrollo como India o Brasil este incremento llegaba hasta el 60%.

Por otro lado, las empresas utilizan los partes de trabajo para ser más productivos en sus procesos internos y, por ende, atender mejor a sus clientes. En cuanto a las soluciones que aplican otras empresas para el seguimiento de los partes de trabajo. La gran mayoría que no usan el formato en papel optan por hacer uso de aplicaciones

nativas Android. A continuación, se muestra el caso de Kizeo Forms una de las aplicaciones más populares en este ámbito.

Comenzaremos señalando que Kizeo posee una página web con la que permite diseñar los formularios con los que posibilita crear los partes de trabajo para satisfacer las necesidades de las empresas que contratan sus servicios.

Figura 2-5: Pagina web de Kizeo Forms

Una vez creado el formulario, o los formularios necesarios, para los diferentes partes de trabajo, la pagina web nos permite asociarlos a nuestro usuario. De esta forma, cuando nos registramos con la aplicación Android, estos formularios son asociados a nosotros y se nos muestra para poder completarlos. Tras ser completados, la aplicación nos permite enviarlos a nuestro correo en formato PDF.

Figura 2-6: Aplicación Kizeo Forms 1

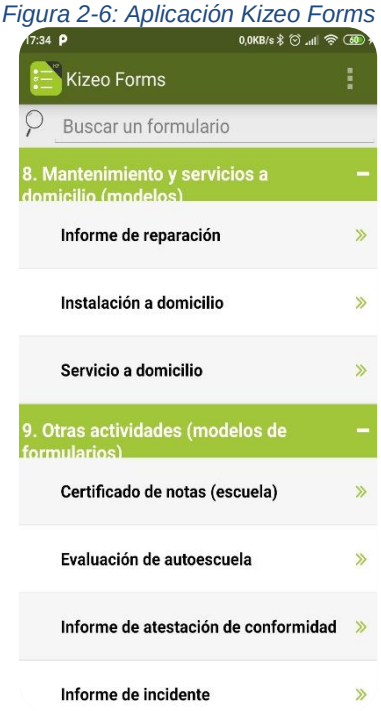


Figura 2-7: Aplicación Kizeo Forms 2

El inconveniente principal de las aplicaciones que se pueden encontrar actualmente en el mercado es que intentan abarcar el mayor número de escenarios posibles. En consecuencia, suelen estar enfocadas para autónomos u empresas con un número muy reducido de empleados. Las pequeñas, medianas y grandes empresas suelen tener unas necesidades muy concretas y este tipo de producto no suele satisfacer sus demandas. Debido a esto muchas suelen optar por crear aplicaciones o programas personalizados, como es nuestro caso.

3. Planificación

A lo largo de este capítulo se mostrará un listado con las actividades que se consideran inicialmente esenciales para la realización de la aplicación y este trabajo. Junto con el listado se mostrará las horas estimadas para completar cada uno de estos trabajos. Posteriormente, se hará un estudió de los plazos mediante un diagrama de Gantt.

3.1. Estimación de tiempos

El proyecto comienza el día 7 de enero del 2020 y finaliza el día 25 de mayo del mismo año. El desarrollo del proyecto se ha dividido en iteraciones de 2 semanas de trabajo real.

Tabla 3-1: Estimación de tiempos

Iteración	Actividad	Horas
1	Investigación y estudio teórico de la tecnología PWA	80
2	Investigación y estudio de la tecnología Laravel	20
	Pruebas prácticas PWA	20
	Estudio de alternativas a PWA	40
3	Realización del diseño de la aplicación	72
	Redacción de la Memoria del Proyecto	8
4	Programación del formulario para que funcione cuando hay conexión.	52
	Programación de una función que detecte cuando se está online y cuando no.	10
	Realización de pruebas en busca de fallos y solucionarlos	10
	Redacción de la Memoria del Proyecto	8

5	Añadir funciones de autocompletado a los formularios	42
	Realización de pruebas con la base de datos y descubrir necesidades a cubrir para la aplicación	20
	Realización de pruebas en busca de fallos y solucionarlos	10
	Redacción de la Memoria del Proyecto	8
6	Programación del Service Worker para añadirle funcionalidad offline al primer formulario.	62
	Realización de pruebas en busca de fallos y solucionarlos	10
	Redacción de la Memoria del Proyecto	8
7	Programación del Service Worker para añadirle funcionalidad offline al segundo formulario.	62
	Realización de pruebas en busca de fallos y solucionarlos	10
	Finalización de la Memoria del Proyecto	8
8	Implantación de la aplicación	8
	Realización de la presentación del proyecto	10

Todas estas tareas suman un tiempo total de 578 horas, divididas en dos partes: 128 horas durante el mes de enero y 450 desde febrero hasta mayo.

A causa de las prácticas que había hecho en el primer semestre del curso 2019-2020 y mi disponibilidad de tiempo. Durante el mes de enero estuve contratado como trabajador de la empresa Electricidad Ramos. Esto supuso que durante el primer mes pudiese dedicarle 128 horas.

Al comenzar el segundo semestre del curso 2019-2020 en febrero, firme un nuevo

convenio con la universidad para continuar el trabajo de fin de grado en la misma empresa durante un total de 400 horas. Las 50 horas restantes son las estimadas por la universidad para la realización de esta memoria.

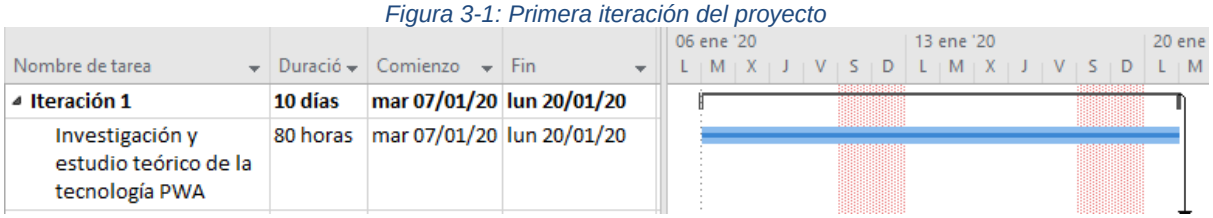
3.2. Diagrama de Gantt

Para poder ver de una forma gráfica la planificación mostrada en el apartado anterior y los plazos de cada iteración se ha elaborado un diagrama de Gantt. Además, para facilitar el análisis y la lectura de este se ha recortado cada una de las iteraciones por separado. A continuación se describirá los plazos de cada una de las iteraciones y sus objetivos.

Iteración 1:

Esta iteración comenzará junto con el proyecto el día 7 de enero y finalizará el día 20 de enero. Durante esta iteración se pretende conseguir un conocimiento teórico sobre PWA y las tecnologías que la engloban. Así como comenzar a hacer pruebas y familiarizarse con todas estas por separado.

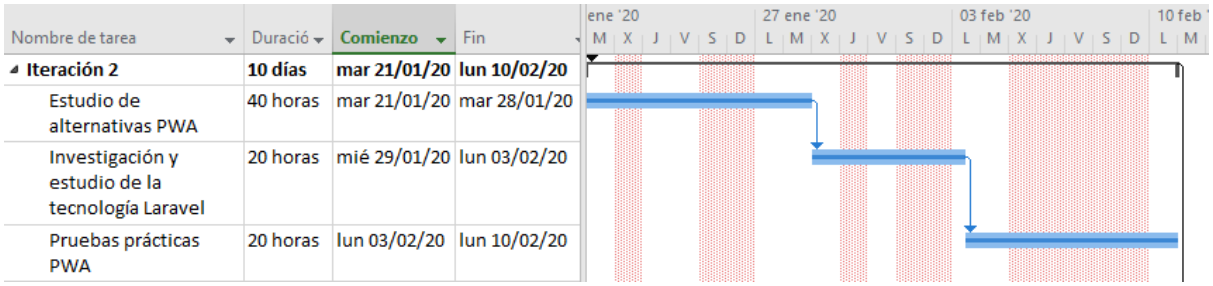
Figura 3-1: Primera iteración del proyecto



Iteración 2:

La segunda iteración dará comienzo el día 21 de enero y finalizará el día 10 de febrero. Durante esta iteración se busca descubrir si existe alguna alternativa real a las PWA, sus ventajas y desventajas para más adelante poder mostrarlas en esta memoria. Además de esto, se realizará un estudio del framework que usaremos en el lado del servidor, Laravel. Así como las pruebas que se consideren oportunas para alcanzar este conocimiento. Una vez hecho esto se realizará un pequeño desarrollo para poner en práctica todos los conocimientos adquiridos durante la primera y segunda iteración para realizar una PWA, básica que nos servirá como base para nuestro desarrollo.

Figura 3-2: Segunda iteración del proyecto



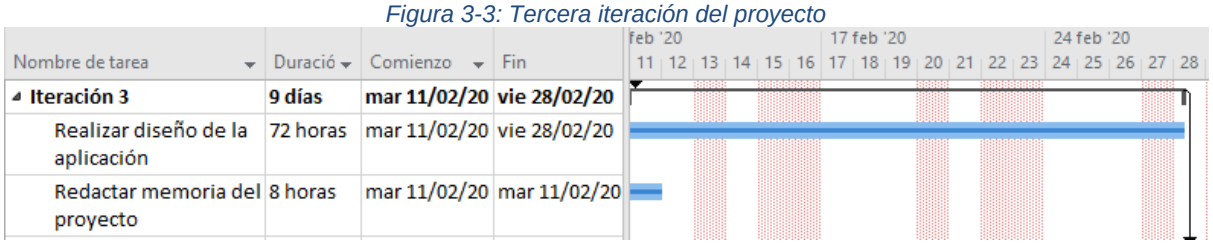
Nombre de tarea	Duració	Comienzo	Fin
Iteración 2	**10 días**	**mar 21/01/20**	**lun 10/02/20**
Estudio de alternativas PWA	40 horas	mar 21/01/20	mar 28/01/20
Investigación y estudio de la tecnología Laravel	20 horas	mié 29/01/20	lun 03/02/20
Pruebas prácticas PWA	20 horas	lun 03/02/20	lun 10/02/20

Se debe destacar que durante esta iteración se hará el cambio del contrato personal inicial con la empresa al convenio con la universidad para la realización del TFG. Es por ello que, durante la primera semana de febrero solo se trabajará los dos primeros días. En las siguientes iteraciones para respetar el convenio firmado con la universidad y la empresa se trabajarán 29 horas semanales. Para cumplir este número de horas semanales los miércoles solo se trabajará 5 horas en el proyecto y ninguna durante los jueves.

Iteración 3:

La tercera iteración se iniciará el día 11 de febrero y finalizará el día 28 de febrero. En esta iteración se persigue alcanzar una idea clara de las especificaciones que debe cumplir la aplicación y con ellas realizar un diseño que satisfaga todas ellas. Junto con esto se pretende comenzar la redacción de este trabajo, que sera a partir de ahora una tarea constante en cada una de las iteraciones. La realización de este trabajo se hará fuera de las 29 horas semanales de trabajo en la empresa. Es por esta razón que la realización de esta tarea se muestra en paralelo con todas las demás.

Figura 3-3: Tercera iteración del proyecto



Nombre de tarea	Duració	Comienzo	Fin
Iteración 3	**9 días**	**mar 11/02/20**	**vie 28/02/20**
Realizar diseño de la aplicación	72 horas	mar 11/02/20	vie 28/02/20
Redactar memoria del proyecto	8 horas	mar 11/02/20	mar 11/02/20

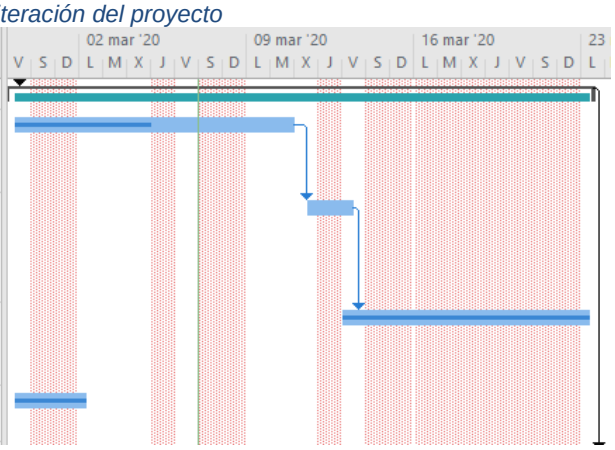
Iteración 4:

La cuarta iteración empezará el día 28 de febrero y finalizará el día 23 de marzo. Durante esta iteración se comenzará a escribir el código que dará vida a la aplicación. La principal tarea será conseguir una primera versión de la aplicación funcional pero sin datos persistentes. No se busca que la aplicación sea capaz de trabajar sin conexión pero sí que esta sea capaz de detectar cuando ha perdido la conexión y cuando la ha recuperado. Además, como es lógico, al final de la iteración se realizarán pruebas para tratar de encontrar fallos y solucionarlos. Al igual que la tarea

de redactar el trabajo, esta será también una tarea constante a lo largo de todas las iteraciones.

Figura 3-4: Cuarta iteración del proyecto

Nombre de tarea	Duració	Comienzo	Fin
Iteración 4	**9 días**	**vie 28/02/20**	**lun 23/03/20**
Programar aplicación para que funcione cuando hay conexión	52 horas	vie 28/02/20	mar 10/03/20
Programar función para detectar cuando se esta online y cuando no	10 horas	mié 11/03/20	vie 13/03/20
Realizar pruebas en busca de fallos y solucionarlos	10 horas	vie 13/03/20	lun 23/03/20
Redactar Memoria del Proyecto	8 horas	vie 28/02/20	lun 02/03/20

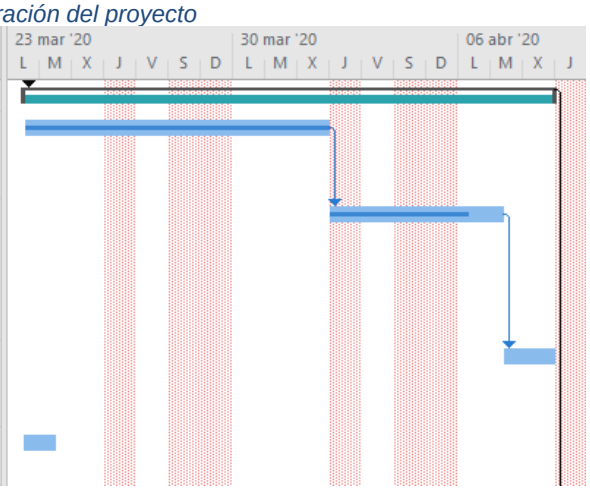


Iteración 5:

Tras terminar la iteración 4 dará comienzo, el día 23 de marzo la iteración 5 y se dará por concluida el día 8 de abril. El objetivo principal de esta iteración será realizar la conexión con la base de datos de pruebas de la empresa y realizar las primeras consultas simples. Estas primeras consultas se realizarán para poder ofrecer la información para autocompletar los campos que sean posibles. Junto con esto, se realizará un pequeño estudio personal de la base de datos para poder realizar posteriormente consultas más complejas que requerirá la parte offline, además de, para comprobar las tablas y campos que serán necesarios para la comprobación e inserción de datos.

Figura 3-5: Quinta iteración del proyecto

Nombre de tarea	Duració	Comienzo	Fin
Iteración 5	**9 días**	**lun 23/03/20**	**mié 08/04/20**
Añadir funciones de autocompletados a los formularios	42 horas	lun 23/03/20	mié 01/04/20
Realizar pruebas con la base de datos y descubrir necesidades a cubrir para la aplicación	20 horas	mié 01/04/20	mar 07/04/20
Realizar pruebas en busaca de fallos y solucionarlos	10 horas	mar 07/04/20	mié 08/04/20
Redactar memoria del proyecto	8 horas	lun 23/03/20	mar 24/03/20

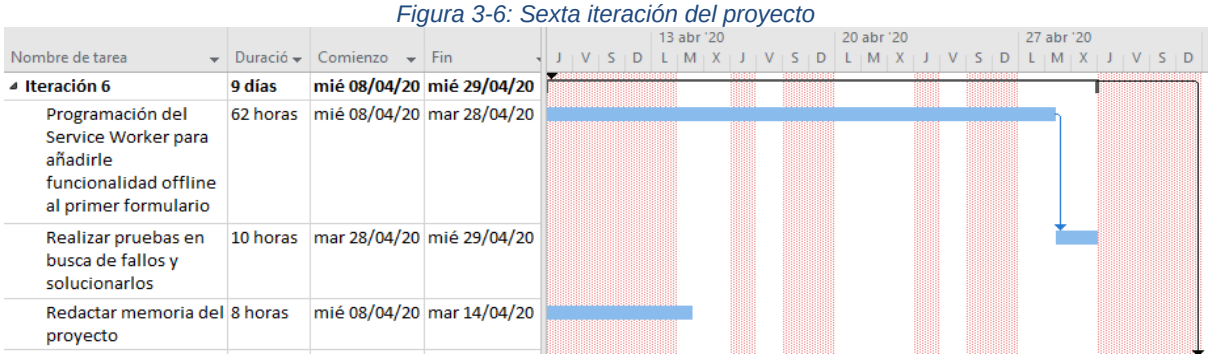


Iteración 6:

La iteración 6 comenzará el día 8 de abril y terminará el día 29 de abril. Al comienzo de esta iteración deberíamos tener una aplicación totalmente funcional que cumpla todas las especificaciones que se nombraron en el apartado 2.2.1 (Requisitos

funcionales iniciales) siempre y cuando tengamos conexión. Por este motivo durante esta iteración el objetivo principal será comenzar a darle funcionalidades offline a la aplicación. Para ello comenzaremos añadiendo la funcionalidad de crear y guardar partes de trabajo sin conexión.

Figura 3-6: Sexta iteración del proyecto

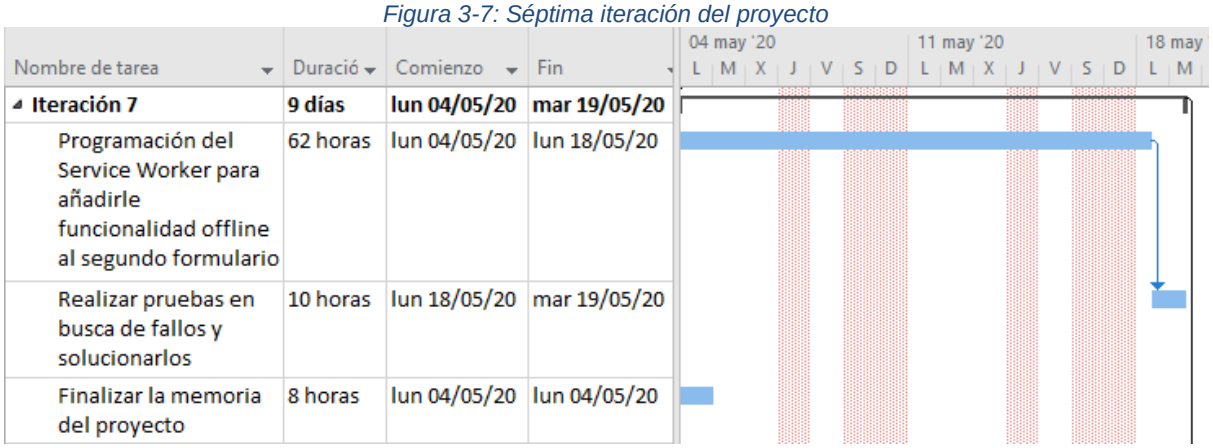


Nombre de tarea	Duració	Comienzo	Fin
Iteración 6	**9 días**	**mié 08/04/20**	**mié 29/04/20**
Programación del Service Worker para añadirle funcionalidad offline al primer formulario	62 horas	mié 08/04/20	mar 28/04/20
Realizar pruebas en busca de fallos y solucionarlos	10 horas	mar 28/04/20	mié 29/04/20
Redactar memoria del proyecto	8 horas	mié 08/04/20	mar 14/04/20

Iteración 7:

El día 4 de mayo dará comienzo la iteración 7 y se dará por finalizada el día 19 de mayo. Durante esta iteración se continuará añadiendo funcionalidades offline y esta vez se añadirá la funcionalidad de mostrar partes de trabajo y firmar partes de trabajo previamente creados sin tener conexión. La conclusión de esta iteración debería implicar también la conclusión del desarrollo de la aplicación y de este mismo trabajo.

Figura 3-7: Séptima iteración del proyecto

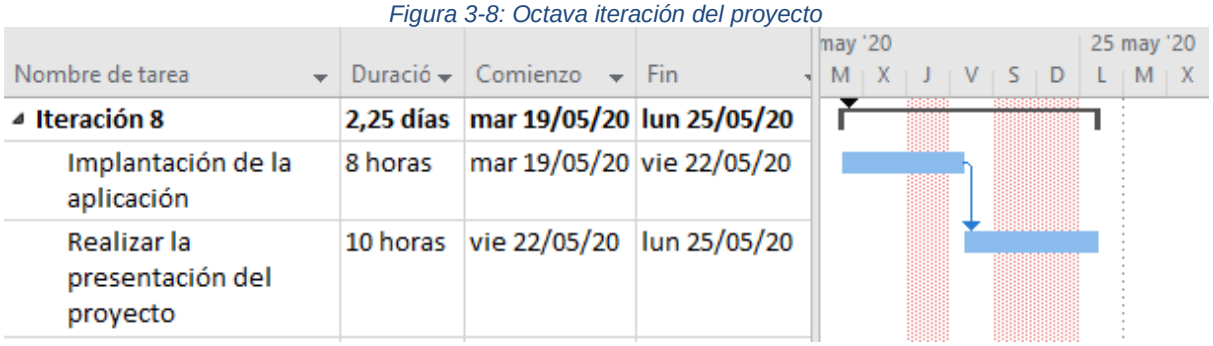


Nombre de tarea	Duració	Comienzo	Fin
Iteración 7	**9 días**	**lun 04/05/20**	**mar 19/05/20**
Programación del Service Worker para añadirle funcionalidad offline al segundo formulario	62 horas	lun 04/05/20	lun 18/05/20
Realizar pruebas en busca de fallos y solucionarlos	10 horas	lun 18/05/20	mar 19/05/20
Finalizar la memoria del proyecto	8 horas	lun 04/05/20	lun 04/05/20

Iteración 8:

Para finalizar la última iteración se iniciará el día 19 de mayo y finalizará el día 25 de mayo. En el transcurso de esta iteración, que será la de menor duración de todas, se efectuara la implantación de la aplicación en la empresa y se vigilarán los posibles problemas que esto pudiera generar. A su vez, se realizará la presentación del proyecto para la defensa del mismo.

Figura 3-8: Octava iteración del proyecto



Nombre de tarea	Duració	Comienzo	Fin
Iteración 8	**2,25 días**	**mar 19/05/20**	**lun 25/05/20**
Implantación de la aplicación	8 horas	mar 19/05/20	vie 22/05/20
Realizar la presentación del proyecto	10 horas	vie 22/05/20	lun 25/05/20

4. Estudio de la tecnología PWA

4.1. Introducción

Una aplicación web progresiva (PWA, por sus siglas en inglés) es un tipo de aplicación software que se recibe a través de la web. A diferencia de las aplicaciones nativas que dependen del sistema operativo, las aplicaciones web progresivas son ejecutadas dentro del navegador. La independencia del sistema operativo ofrece la ventaja de poder abstraerse de este y realizar el desarrollo de una única versión para todos los dispositivos.

Esta independencia del sistema operativo no impide a las aplicaciones web progresivas hacer uso del hardware del dispositivo, enviar notificaciones push o incluso la posibilidad de trabajar sin conexión.

Otra de las características de las aplicaciones web progresivas es que no requieren ser distribuidas por medio de ninguna plataforma de distribución digital como Google Play o App Store en los dispositivos móviles. Al igual que cualquier otra página web, la distribución de estas aplicaciones se realiza mediante una transferencia desde servidores, utilizando el protocolo seguro de transferencia de hipertexto (en inglés, Hypertext Transfer Protocol Secure o HTTPS).

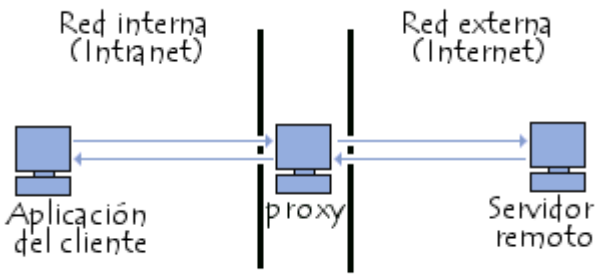
Todas estas características extra que no disponen las aplicaciones web comunes son debidas principalmente a los Service Workers.

4.2. Service Worker

Un Service Worker (SW) es una secuencia de comandos JavaScript que se ejecuta en el navegador en segundo plano en un hilo independiente a la página web [12]. Su función principal es la de interceptar y manejar solicitudes de red de la página web o el sitio web con el que está asociado. Esto posibilita que cuando el usuario cierra la página web completamente el SW pueda seguir ejecutándose en segundo plano y pueda seguir escuchando eventos sin tener que vivir dentro del navegador. De esta manera, ejecutándose en segundo plano y escuchando eventos, permite recibir notificaciones push, saber cuándo se pierde o se recupera la conexión de internet o hacer actualizaciones en segundo plano sin necesidad de preguntar nada al usuario.

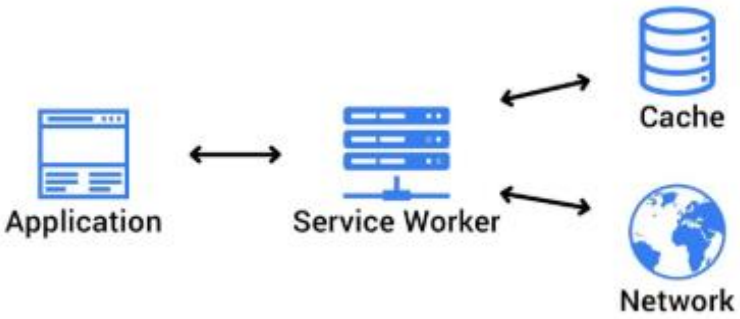
Los SW actúan esencialmente como **proxy** server asentados entre las aplicaciones web, el navegador y la red (cuando ésta es accesible). Un proxy es un servidor (programa o dispositivo), que hace de intermediario en las peticiones de recursos que realiza un cliente a otro servidor. Por ejemplo, si una hipotética máquina ***A*** solicita un recurso a ***C***, lo hará mediante una petición a ***B***, que a su vez trasladará la petición a ***C***; de esta forma ***C*** no sabrá que la petición procedió originalmente de ***A***.

Figura 4-1: Diagrama posición proxy



Esta situación estratégica de punto intermedio le permite ofrecer diversas funcionalidades. Pongamos que cambiamos de una página a otra página de nuestra web. Normalmente haríamos una petición al servidor web y este nos devolvería la página solicitada. Sin embargo, con el SW en medio no es así. Al solicitar una nueva página ahora esta solicitud va al SW, este es capaz de comprobar si previamente ha cacheado la página solicitada o no, y si lo ha hecho, puede devolvérsela al usuario sin hacer ninguna petición al servidor web y de manera más rápida. Además de eso, él SW puede manejar una base de datos local por lo que se pueden hacer búsquedas sin ni siquiera tener conexión.

Figura 4-2: Diagrama posición Service Worker



Esta habilidad del SW de actuar como proxy nos permite responder a las solicitudes del cliente más rápido que una página web normal si esta se responde desde cache y nos permite liberar al servidor de trabajo. Por contra, el uso del SW nos obliga a tener un certificado de seguridad SSL y ser enviado desde el servidor a través del protocolo **HTTPS**. Esto es debido a que se pueden realizar ataques de intermediario (en inglés, man in the middle attack) si recibimos un SW que no es el original. Gracias al protocolo HTTPS, nos aseguramos de que el SW que recibe el navegador no se ha manipulado durante su recorrido por la red.

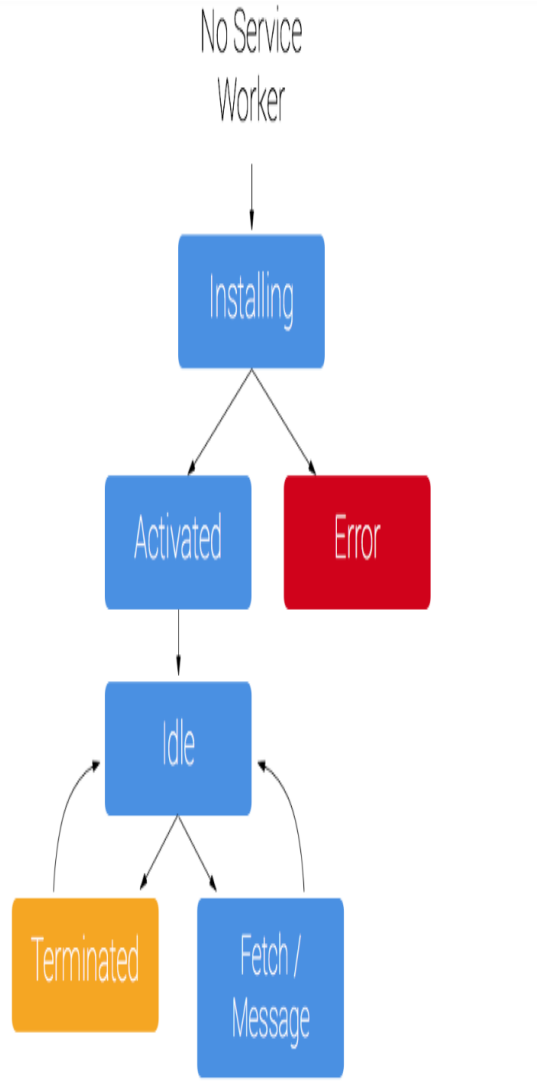
Para poder añadir un SW a una página primero hay que registrarlo mediante JavaScript. El código para poder realizarlo es el siguiente:

Figura 4-3: Registro del Service Worker

```
//Registrar el Service Worker
if('serviceWorker' in navigator){
  navigator.serviceWorker.register('sw.js')
    .then(reg => {
        console.log('Service Worker registrado');
    })
    .catch(err => console.log('No se pudo registrar el service worker', err));
}
```

Una vez hecho el registro del SW comienza el ciclo de vida de este.

Figura 4-4: Ciclo de vida del Service Worker



Instalando: Cuando registramos un SW, si no tenemos ningún SW instalado entrará en el primer estado, denominado instalando. En este estado se descargará el archivo indicado en la función *register* del servidor web y se compilará. Puede ser que falle y ahí termine todo el ciclo o se instale con éxito y pase al siguiente estado. Por lo general, durante la etapa de instalación, se almacena en caché los elementos indicados en el SW. Si el SW no tiene ningún error de sintaxis y todos los archivos se almacenan correctamente en caché, este se instalará correctamente. Si por el contrario no se puede descargar o almacenar en caché alguno de los archivos especificados, el paso de instalación fallará y el SW no se instalará.

Activando: Si la instalación ha concluido con éxito entrará en el siguiente estado, activando. Si no existía ningún SW previamente registrado para el sitio web, pasará a este estado justo después de acabar la instalación, si por el contrario ya existía un SW en la página, después de la instalación no pasará a este estado hasta que se salga del dominio y se vuelva a entrar. Esto se hace para que no haya conflictos entre la nueva y la vieja versión del SW. Durante este estado el nuevo SW eliminará las versiones de cache anteriores y la versión antigua del SW.

IDLE: Una vez terminado la activación él SW controlará todas las páginas que estén a su alcance, es decir, en su mismo directorio o en directorios hijos. Es por ello recomendable tenerlo instalado en la carpeta raíz de la web, ya que, no puede controlar archivos que estén en carpetas superiores. Mientras no reciba ninguna petición el SW se encontrará en estado de espera. Desde aquí puede saltar a cualquiera de los siguientes dos estados.

Fetch/Message: Cuando el usuario realiza una petición, se manda una petición push o cualquier otro evento que haga trabajar al SW entrará en este estado. El SW recibirá un evento que escuchará y responderá.

Terminated: Cuando el SW permanece mucho tiempo en estado idle, para liberar la carga que ejerce sobre el navegador, entra al estado llamado terminated. Sin embargo, en cuanto vuelve a recibir una petición se vuelve a despertar y a trabajar con normalidad.

4.3. Promesas

Es muy frecuente que en el SW, al trabajar en un hilo paralelo al de la aplicación principal, se realicen muchas tareas asíncronas, ya sea para guardar datos en cache, hacer peticiones web, hacer inserciones en la base de datos del navegador... Para ello el SW hace uso de los objetos promesas [13]. Una promesa es un objeto que representa la terminación o el fracaso eventual de una operación asíncrona. Para ello se le pasa dos argumentos de entrada, los dos argumentos son funciones. La primera para ejecutar en caso de que todo haya salido correctamente y la segunda para ejecutar cuando se haya producido algún error.

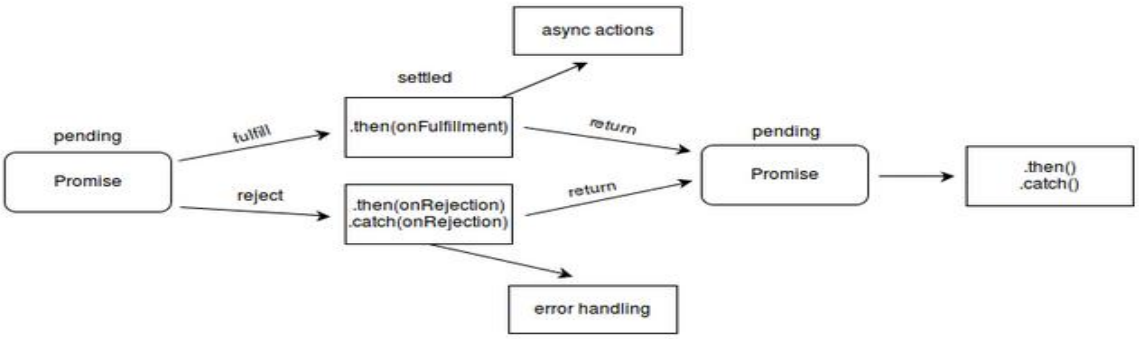
Para entender de una forma un poco más sencilla este concepto imaginemos que queremos hacer una petición de una página web para guardarla en cache. Hacemos la petición mediante una promesa y podemos continuar la ejecución del código sin volvernos a preocupar por esta promesa. Cuando la petición haya terminado si se ha devuelto la página significara que la promesa se ha cumplido y por tanto ejecutará una función de éxito en la que guardaremos en cache esta página para futuras consultas. Si por el contrario la promesa se incumple y no se trae la página porque ha ocurrido algún problema durante la petición, como por ejemplo cuando no tenemos conexión a internet. La función de error será ejecutada y en este caso, por ejemplo, crearemos otra petición para intentar volver a conseguir la página más adelante cuando tengamos conexión.

Teniendo esto en cuenta podemos observar los diferentes estados en los que se puede encontrar una promesa.

- Pendiente: Estado inicial.

- Cumplida: Estado que adquiere cuando la acción se completó satisfactoriamente
- Rechazada: Estado que adquiere cuando la acción no se completó satisfactoriamente

Figura 4-5: Estados de una promesa



Además de los estados, las promesas cumplen siempre las siguientes características:

- Una promesa sólo puede completarse con éxito o fallar una vez.
- Si una promesa se ha completado con éxito o ha fallado, y luego se agrega un callback de éxito/fallo, la promesa llamara al callback correcto, a pesar de que el evento haya sucedido antes.
- Las promesas de JavaScript se crean de la siguiente manera:

Figura 4-6: Instancia de una promesa

```
var promise = new Promise(function(funcionExito, funcionError){
    // codigo a ejecutar
    if (/*Todo ha ido bien */){
        funcionExito();
    }else{
        funcionError(Error('Algo salio mal'));
    }
});
```

Una vez creadas para poder hacer uso de ellas hay que llamar al método ***Then***. *Then* recibe los dos argumentos anteriormente mencionados, la función para cuando la promesa se completa con éxito y la función para cuando la promesa falla. Ambas son opcionales. Estas funciones son denominadas también funciones callback. Una función de callback es una función que se pasa a otra función como un argumento.

La forma de usar el método *Then* de una promesa es la mostrada en el lado derecho de este texto. El resultado de la ejecución de la promesa se asignara en este ejemplo a la variable *result* en caso de éxito o a la variable *err* en caso de error.

Figura 4-7: Ejemplo método then

```
Promise.then(function(result){
    //Función de exito
},function(err){
    //función de error
});
```

Además del método *Then*, las promesas incorporan otros métodos propios los cuales mencionaremos algunos a continuación:

Promise.all():

El método *all* sirve para juntar varias promesas que se desea se cumplan conjuntamente. Las promesas introducidas dentro de este método se ejecutan y si alguna de todas las de la lista falla, todas son rechazadas. No importa si se han introducido 10 promesas en la lista y 9 han salido correctamente, si solo una falla, todas son rechazadas.

Dentro de la lista que mandamos al método *all*, no solo se pueden introducir promesas, también se pueden introducir funciones normales e incluso variables

Figura 4-8: Ejemplo método all

```
let aEjecutar = [sumarUno(5),sumarDos(9),true, 'hola mundo']
Promise.all(aEjecutar)
    .then(function(result){
        //Función de exito
        console.log(result);
    })
    .catch(console.log('Error'));
```

Figura 4-9: Resultado del método all

```
[ 6, 11, true, 'hola mundo' ]
```

El método *all* nos devolverá el resultado de todo esto si todas se han ejecutado correctamente.

Promise.race():

El método *race* funciona de manera similar al método *all* con la diferencia que al usarlo el método *race* pone a competir todas las promesas que tenemos en la lista. La respuesta va a ser única y va a ser la primera promesa que responda. Si dos promesas responden a la vez, *race* devolverá el resultado de la promesa que más a la izquierda estuviera entre las dos.

De igual forma que *all,* si alguna de las promesas de la lista falla entrará en el catch y devolverá el error.

Promise.allSettled():

Por último el método *allSettled* recibe una lista de promesas y espera a que todas finalicen. Devuelve una promesa cuyo contenido es un array de objetos que describen el resultado de cada una de las promesas.

Para cada uno de los objetos devueltos existe un campo *status* cuyo valor es *fulfilled* si la promesa se resolvió correctamente y *rejected* si no lo hizo. Si la promesa se resolvió correctamente este campo va acompañado del valor devuelto. Si por el contrario fue rechazada el campo *status* va acompañado del campo *reason*, que explica la razón por la que fue rechazada.

4.4. Cache

Como hemos visto en este capítulo los SW nos permiten crear un proceso en segundo plano en el navegador del usuario en el que se está ejecutando. Además, este proceso se mantiene vivo cuando no hay conexión a internet o incluso cuando el usuario ha cerrado la aplicación. Sin embargo, necesitamos guardar datos para poder trabajar como una aplicación nativa cuando esta está sin conexión y no puede alcanzar los datos y ficheros del servidor. Para lograr este objetivo en este apartado veremos cómo podemos hacer para almacenar las respuestas a las peticiones HTTPS que realiza nuestra aplicación.

4.4.1. CacheStorage API

La API de almacenamiento cache nos permite gestionar objetos cache. Estos objetos son archivos en los que se almacenan tantos pares Solicitud/Respuesta como se desee. Además, la interfaz proporciona un directorio raíz donde se almacenarán todos los objetos cache a los que puede acceder un SW.

Un SW puede manejar múltiples objetos cache los cuales pueden contener múltiples pares de objetos solicitud/respuesta. Sin embargo, el SW está limitado por su dominio o por el apartado *scope* del manifiesto de nuestra aplicación, de lo cual se hablara más detenidamente en el apartado 4.5 (Manifiesto de aplicaciones web). Esto es debido a que los objetos cache no se comparten entre diferentes dominios y están completamente aislados de la memoria cache HTTP del navegador. Por ello el SW solo puede acceder a los objetos cache que hayamos creado para nuestro dominio.

Para tener acceso a estos la interfaz nos proporciona varios métodos:

CacheStorage.match(): Dado un objeto Request o una URL comprueba si existe algún objeto cache para el cual este objeto Request o la URL es una clave. El método devuelve una promesa que contiene el objeto si existe la clave o *undefined si no existe.*

CacheStorage.has(): Dado un string comprueba si existe algún objeto cache con el mismo nombre. Devuelve *true* si existe, *false* si no existe.

CacheStorage.open(): Dado un string comprueba si existe un objeto cache con ese nombre. Si existe lo abre, en caso contrario lo crea y lo abre.

CacheStorage.delete(): Elimina el objeto cache cuyo nombre coincida con el string enviado como parámetro. Si lo elimina correctamente devuelve *true*. Si no encuentra el objeto cache especificado devuelve *false.*

CacheStorage.keys(): Devuelve una promesa que contiene un array de strings con todos los nombre de los objetos cache a los que puede acceder el SW actual.

4.4.2. Cache API

La Api de cache proporciona un mecanismo para los pares de objetos Solicitud/Respuesta que se almacenan en un objeto cache [14]. Es decir, guarda para cada Solicitud HTTPS que realizamos en nuestra página, la respuesta que nos envía el servidor, ya sea una página web, una imagen, un archivo CSS o cualquier otra cosa. La API de cache se creó para permitir que los SW almacenasen en cache las solicitudes de red y que de esta forma pudiesen proporcionar respuestas adecuadas incluso sin conexión. Y aunque su especificación está dentro de la de los propios SW, la API también se puede utilizar como mecanismo de almacenamiento general fuera de estos.

Las solicitudes para conseguir los recursos necesarios no son parte de la API y pueden hacerse durante el uso normal de nuestra aplicación, como ha ocurrido siempre para las páginas web convencionales, o, gracias a los SW, pueden ser creadas también, con el único propósito de guardar esa información en cache para tenerla disponible posteriormente. Independientemente de la razón por la que hayamos hecho la solicitud, la API nos permite guardar las respuestas, pero, hay que tener en cuenta que al crear un objeto de cache estos cumplen las siguientes especificaciones:

- Los objetos de cache no se actualizan a menos que se cree una petición para ello.
- Los objetos de cache no desaparecen a menos que exista una petición para ello.
- Aunque actualicemos el SW los objetos de cache que este haya creado no desaparecen.

En general, esto quiere decir que los objetos de cache no se actualizan automáticamente. Las actualizaciones deben administrarse manualmente. Esto implica que debemos ser nosotros como desarrolladores los que decidamos que guardar en cada momento y que no guardar.

Para este propósito la API de cache nos ofrece varios métodos, todos ellos devuelven una promesa con la que posteriormente podremos trabajar:

Cache.match(Solicitud): Busca dentro del objeto cache la solicitud enviada como parámetro de entrada. Si es encontrada devuelve una promesa con la respuesta a esa solicitud.

Cache.matchAll(Solicitudes): Devuelve una promesa con un array de respuestas a todas las solicitudes que tuviese guardado el objeto cache.

Cache.add(Solicitud): Dado un objeto Request o una URL, realiza la petición al servidor y almacena la respuesta dentro del objeto.

Cache.addAll(Solicitudes): Se introduce como parámetro de entrada un array de objetos Request o URL, el método realiza todas las peticiones y almacena todas las respuestas dentro del objeto.

Cache.put(Solicitud, Respuesta): Añade el par solicitud/respuesta introducidos como parámetros de entrada al objeto cache.

Cache.delete(Solicitud): Elimina el par solicitud/respuesta que coincida con la solicitud introducida como parámetro de entrada.

Cache.keys(): Devuelve una promesa con todas las claves del objeto cache en cuestión.

4.5. IndexedDB

Gracias a las API de cache podemos guardar los resultados de las peticiones HTTP que hacemos con nuestra aplicación, sin embargo, necesitamos guardar más datos además de estos. En nuestro caso, por ejemplo, necesitamos guardar los partes de trabajo que se han generado, dado que, si estamos en un lugar sin conexión y hacemos una petición para guardarlos en la base de datos del servidor, perderemos todos los datos. Para solucionar este problema existe la API IndexedDB [15].

IndexedDB permite crear bases de datos no relacionales dentro del navegador en las cuales, una vez creadas, permite almacenar y obtener objetos indexados a partir de una “clave”. Todos los cambios realizados a la base de datos ocurren dentro de transacciones y al igual que la API cache restringía el acceso al SW a los objetos cache de nuestro dominio, IndexedDB, solo permite acceder al SW a las bases de datos de nuestro dominio.

A continuación se explicará con un poco más de detalle todas las características mencionadas anteriormente:

Utiliza pares clave-valor: Cada registro de la base de datos que crea la API consta de una clave y un valor. Los valores que almacena IndexedDB pueden ser objetos con estructuras tan complejas como deseemos, y las claves pueden ser una propiedad de esos objetos. Sin embargo, estas claves deben ser únicas. No pueden existir múltiples registros en una base de datos con la misma clave. IndexedDB permite crear índices con cualquiera de las propiedades de los objetos para realizar búsquedas, sin embargo, si no se especifica explícitamente el valor de la clave cuando se almacenan, la API genera uno automáticamente.

Es transaccional: La API está construida sobre un modelo de base de datos transaccional. Es decir, todo lo que hagamos siempre ocurre en el contexto de una transacción. Además, las transacciones tienen un periodo de vida definido. Por esta

razón, cualquier intento de utilizar una transacción que ya se ha completado generará una excepción.

Este modelo de transacciones es realmente útil, ya que, debemos tener en cuenta que un usuario podría tener ejecutándose simultáneamente nuestra aplicación en dos pestañas diferentes del navegador.

Es asíncrona: La API funciona mediante promesas, es decir, no “guardamos” o “leemos” un valor de la base de datos. En cambio, “solicitamos” una operación a la base de datos. Esta operación se realizará de forma asíncrona y se nos notificará su finalización cuando la promesa se cumpla o falle.

No utiliza SQL: Al ser una API que crea bases de datos no relacionales, esta no incluye ningún modo de realizar consultas SQL. En cambio, usa consultas sobre un índice que producen un cursor. Éste puede utilizarse para iterar sobre el conjunto de resultados.

Como se puede ver, si se revisa la documentación de la API, es una interfaz muy extensa pensada para intentar abarcar muchos usos. Además, la API es relativamente compleja de utilizar, dado que, al ser una API de bajo nivel requiere de una cantidad de código considerable para poder utilizarla correctamente. No es objeto de este proyecto realizar un estudio exhaustivo de esta API, por ello, para facilitar la realización de este proyecto se ha elegido utilizar la librería PouchDB.

PouchDB es una librería escrita en JavaScript que trabaja internamente con la API IndexedDB [16]. Al igual que IndexedDB, permite almacenar, consultar y administrar datos de la base de datos de un navegador. La gran ventaja de esta librería es que permite realizar las mismas tareas que IndexedDB pero con una cantidad de código menor agilizando el desarrollo de las aplicaciones.

4.5. Manifiesto de aplicaciones web

Además de incluir un SW, para que una web se considere una PWA y los navegadores nos dejen instalarla se debe definir un manifiesto de aplicaciones web. El manifiesto es necesario para que la aplicación web se descargue y se presente al usuario de manera similar a una aplicación nativa. Si no se especifica este archivo al navegador nunca nos dejara descargar la aplicación web. Los ficheros manifiesto son compatibles con los navegadores Chrome, Edge, Firefox, Opera y Safari aunque este último, solo parcialmente de momento.

El fichero manifiesto de las PWA es un fichero de texto JSON que proporciona información de nuestra app al navegador. Gracias a este fichero el navegador sabe cómo debe mostrar y comportarse nuestra app cuando esta se instale en un dispositivo móvil. Este fichero puede llamarse como nosotros queramos, aunque la

convención no escrita es llamarlo manifest.json y servirlo desde la raíz de nuestra aplicación web.

Un manifiesto típico debe incluir al menos el nombre de la aplicación, los iconos que la aplicación debe usar y la URL que se debe abrir cuando se inicia la aplicación. Sin embargo, hay mucho otros campos que se pueden indicar en este fichero, tales como: el color de fondo que la aplicación debe tener cuando esta se inicia por primera vez en los dispositivos, la forma en que se muestra la interfaz del navegador cuando se inicia en modo aplicación, por ejemplo, se puede ocultar la barra de dirección o incluso se puede mostrar a pantalla completa. El alcance de URLs que posee nuestra aplicación, tales como todas las de nuestro dominio web. Esta directiva le resulta útil al navegador para saber cuándo ha salido de nuestra aplicación el usuario y ha comenzado a navegar libremente, en casos en los que, por ejemplo, referenciamos otra web. Y otros muchos que se pueden consultar en la documentación del World Wide Web Consortium [17].

Figura 4-10: Ejemplo de manifiesto

```
{
  "name": "Partes de trabajo",
  "short_name": "Partes",
  "theme_color": "#292b2c",
  "background_color": "#fbfbfb",
  "display": "standalone",
  "scope": "/partes1/",
  "start_url": "/partes1/",
  "icons": [
    {
      "src": "images/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "src": "images/icons/icon-96x96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "images/icons/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "images/icons/icon-144x144.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "images/icons/icon-152x152.png",
      "sizes": "152x152",
      "type": "image/png"
    }
  ],
  "splash_pages": null
}
```

Una vez creado el manifiesto, con todas las directivas que deseamos que nuestra aplicación web cumpla e introducido dentro de los archivos de nuestro sitio web debemos, agregar una etiqueta link a todas las páginas que compongan nuestra app web, como se muestra a continuación:

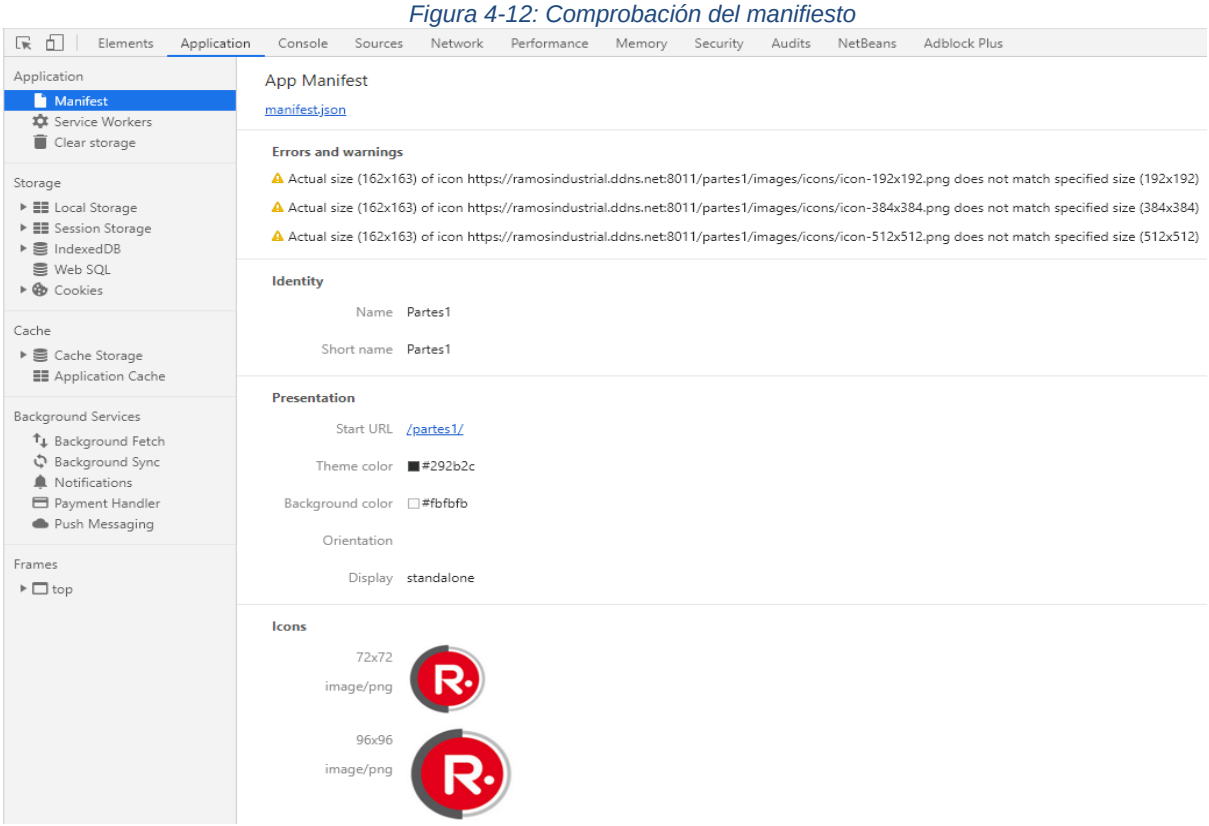
Figura 4-11: Código para importar un manifiesto

```
<!-- Meta tags obligarorios para PWA-->
    <!--Manifest -->
    <link rel="manifest" href="manifest.json" />
```

Si el manifiesto está correctamente construido y no produce ningún error será asociado a la aplicación. Para poder ver si esto ha ocurrido sin ningún problema se

puede verificar en el panel de herramientas de desarrollo dentro de la pestaña aplicación, en el apartado llamado *manifest*, el estado de este.

Figura 4-12: Comprobación del manifiesto



5. Diseño y funcionamiento de la PWA

En este capítulo se describirá la forma de uso de la aplicación de manera que cualquier usuario pueda entender su funcionamiento. Pese a que en el inicio del desarrollo la aplicación solo estaba proyectada para realizar la gestión de los partes de trabajo, durante el transcurso del desarrollo se descubrió la necesidad de incorporar la gestión de las averías también. Debido a esta situación durante este capítulo se describirá este apartado de la aplicación no mencionada hasta este instante, además del apartado de partes de trabajo.

5.1. Apartado “Partes de Trabajo”

El apartado de partes de trabajo está pensado y diseñado para facilitar a los operarios de la empresa crear, modificar, borrar y firmar partes de trabajo. Para este fin la aplicación dispone de las siguientes vistas:

Figura 5-1: Vista lista de partes

Figura 5-2: Vista inserción de datos generales

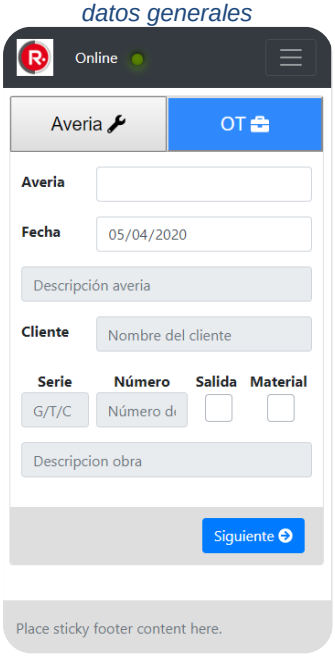


Figura 5-3: Vista inserción de datos de operarios

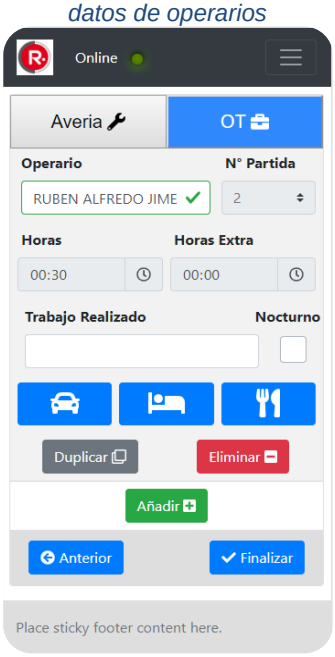


Figura 5-4: Vista firmar parte de trabajo

Es importante mencionar que al crear un parte de trabajo se crean también fichas de trabajo asociadas a este parte de trabajo. Las fichas de trabajo no son más que registros de una actividad o trabajo concreto de un trabajador. Por ello, como se mostrará más adelante, al introducir en la vista *Inserción de datos de operarios* nuevas filas de operarios y con ellos diferentes trabajos, también se crearán internamente las fichas de trabajo asociada a ese trabajo o actividad.

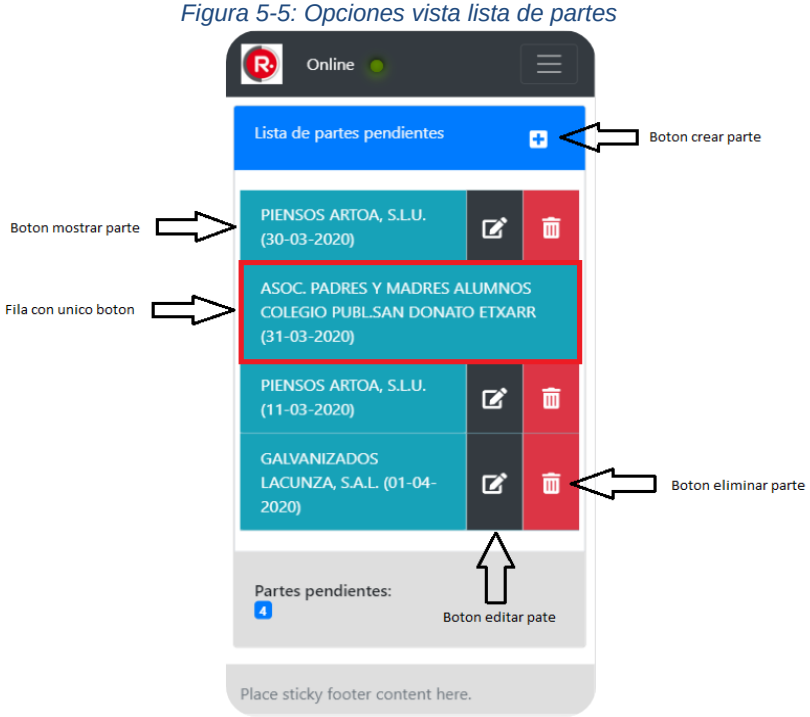
A continuación, se hará una explicación más detallada de cada una de las vistas así como del menú de la aplicación.

5.1.1. Vista lista de partes

Nada más iniciar la sesión con nuestro usuario, la aplicación nos redirigirá a esta vista. En este lugar podemos consultar todos los partes de trabajos que hayamos creado y aún no hayamos firmado. Si un parte de trabajo se firma y se envía, automáticamente el parte de trabajo se dará por terminado y desaparecerá de la lista. Es importante señalar también, que si existen partes de trabajo de días anteriores al del día actual en la lista, en estos solo aparecerá un único botón. Esto es debido a que las fichas de trabajo que generan los partes de trabajo se archivan por la administración todos los días a las 10:00 de la mañana. Para poder modificar estos partes de trabajo habría que ponerse en contacto con la administración y que estos desarchiven las fichas de trabajo asociadas a estos partes de trabajo.

Además de ver de una forma rápida los partes de trabajo que tenemos pendientes, esta vista nos permite realizar diferentes acciones gracias a los botones que contiene.

Figura 5-5: Opciones vista lista de partes

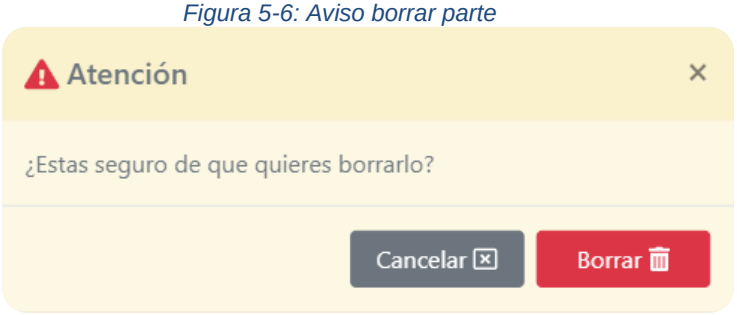


Botón crear parte: Este botón nos redirige a la vista, *inserción de datos generales*, para que podamos iniciar el proceso de crear un parte de trabajo nuevo.

Botón mostrar parte: Si hacemos clic en este botón, la aplicación nos redirigirá a la vista, *firmar parte de trabajo,* y rellenará la vista con los datos del parte de trabajo seleccionado para que podamos mostrárselo al cliente y que lo firme.

Botón eliminar parte: Al seleccionar este botón nos aparecerá el siguiente mensaje de aviso:

Figura 5-6: Aviso borrar parte



Si seleccionamos en el botón de borrar, la aplicación intentará borrar el parte de trabajo y mostrará un mensaje de éxito si se ha conseguido borrar correctamente o un mensaje de error, si no se ha podido borrar el parte de trabajo por algún motivo. En caso contrario, la aplicación simplemente no hará nada.

Botón editar parte: Al seleccionar este botón la aplicación nos redirigirá a la vista, *Inserción de datos generales*. Sin embargo, al contrario que al seleccionar el botón *crear parte de trabajo,* la vista aparecerá rellenada con los datos que previamente hayamos introducido en el parte de trabajo.

5.1.2. Vista inserción de datos generales

Esta vista nos permite introducir los datos generales que contendrá el parte de trabajo que estemos creando o editando. Como se ha mencionado anteriormente, a esta vista solamente se puede llegar si hemos seleccionado el botón de crear parte de trabajo o editar parte de trabajo. Si hemos accedido desde el botón de crear parte de trabajo la vista se mostrará con todos los campos vacíos. Si por el contrario hemos decidido modificar un parte de trabajo, se nos mostrará la vista con todos los campos ya rellenados.

La vista posee dos botones en la parte superior que nos permiten cambiar entre el formulario de datos generales para las averías y para las OT. Según cual tengamos seleccionado nos permitirá rellenar los datos generales para un formulario u otro.

Figura 5-7: Datos generales Avería

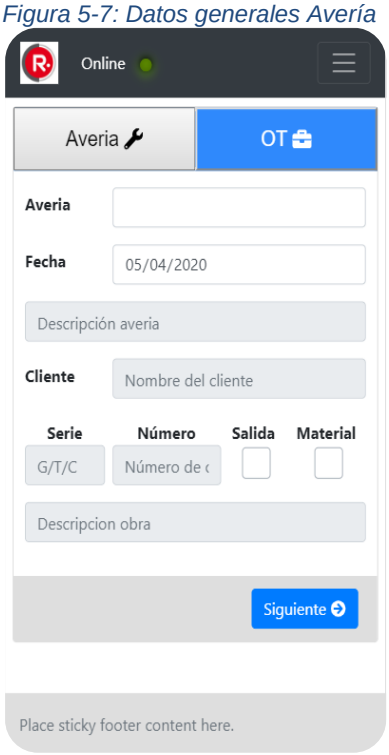
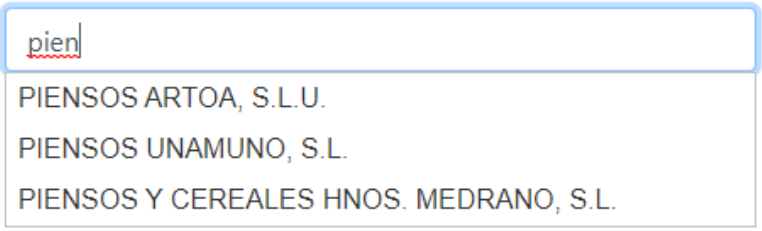


Figura 5-8: Datos generales OT

Si comenzamos a rellenar los datos del formulario, la aplicación nos mostrará sugerencias para cada campo según lo que estemos escribiendo.

Figura 5-9: Ejemplo sugerencias de campos



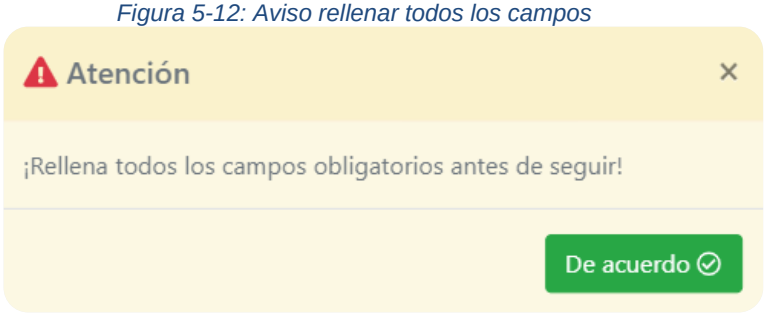
Es **obligatorio** seleccionar una de estas sugerencias, ya que de esta forma la aplicación se asegura que seleccionamos un valor que existe en la base de datos del servidor. Si no seleccionamos una de las sugerencias que la aplicación nos muestra, esta no dará por valido el campo rellenado y no nos dejara continuar con el siguiente paso.

Figura 5-10: Ejemplo campo validado

Figura 5-11: Ejemplo campo no validado

De igual forma si existe algún campo que no está rellenado e intentamos continuar con el siguiente paso, la aplicación nos mostrará un mensaje de aviso para que finalicemos de rellenar todos los campos.

Figura 5-12: Aviso rellenar todos los campos

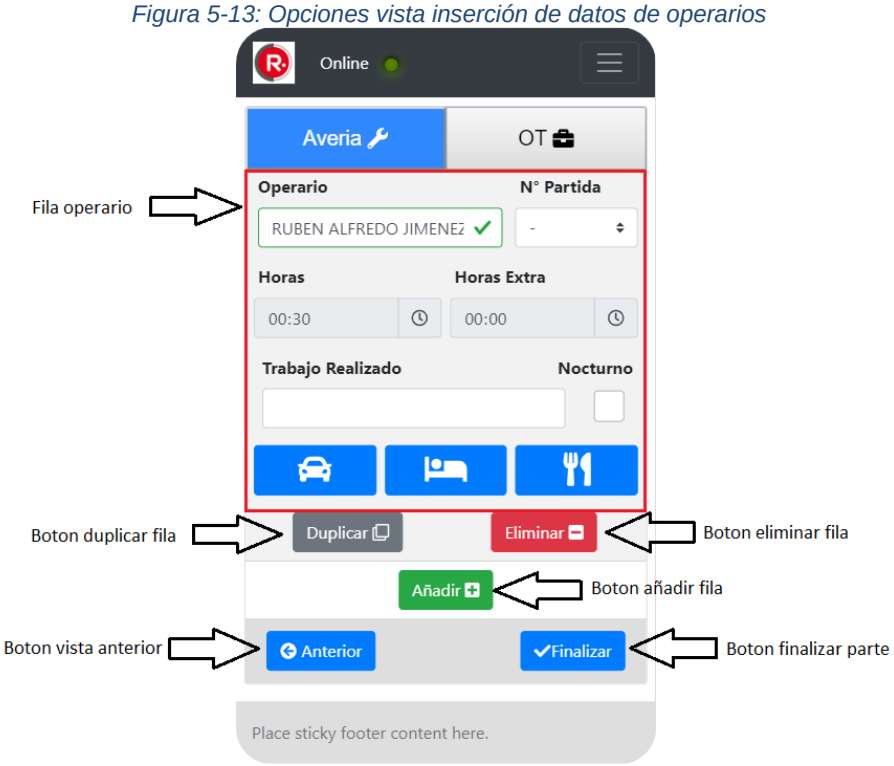


Si todos los campos se han rellenado y son validados por la aplicación, al hacer clic al botón *siguiente,* la aplicación nos permitirá continuar con el proceso de creación o edición del parte de trabajo.

5.1.3. Vista inserción de datos de operarios

Una vez rellenados y validados los datos generales del parte de trabajo y tras pulsar el botón *Siguiente* en la vista *Inserción de datos generales* se nos mostrará esta vista. Aquí podemos rellenar los datos de los operarios que han realizado alguna tarea de las que recoge el parte de trabajo. Gracias a los botones que se describen a continuación es posible añadir, eliminar o modificar tantos usuarios como se desee para cada parte de trabajo.

Figura 5-13: Opciones vista inserción de datos de operarios



En el caso en el que se esté creando un parte de trabajo nuevo, solo aparecerá una fila de operario que contendrá al operario con el que se ha iniciado la sesión. En caso contrario, si se está editando un parte de trabajo aparecerán tantas filas de

operarios como se añadieran en el momento de creación. Cada fila de operario generará internamente una ficha de trabajo nueva, además, cada una posee dos botones para realizar acciones:

Botón duplicar fila: Añade al final del todo una fila idéntica a la que se encuentra el botón.

Botón eliminar fila: Elimina la fila de operario en la que se encuentra el botón.

Además de estos botones la vista tiene 3 botones que realizan acciones independientes de las filas de operarios.

Botón añadir fila: Añade una fila nueva con todos los campos en blanco.

Botón vista anterior: Nos devuelve a la vista *Inserción de datos generales*. Por si queremos volver a modificar alguno de los datos que ya habíamos introducido.

Botón finalizar parte: Envía el formulario al servidor y comprueba si los datos de las horas ordinarias insertadas son válidos. Si alguno de los operarios se ha pasado de las horas ordinarias para ese día nos mostrará un mensaje de error indicándonos que usuario ha excedido las horas ordinarias, en caso contrario, se realizará la inserción en la base de datos del servidor. Si todo sucede de forma correcta nos devolverá un mensaje de éxito, en cambio si ha ocurrido un error nos mostrará otro mensaje de error.

Figura 5-14: Ejemplo mensaje de error

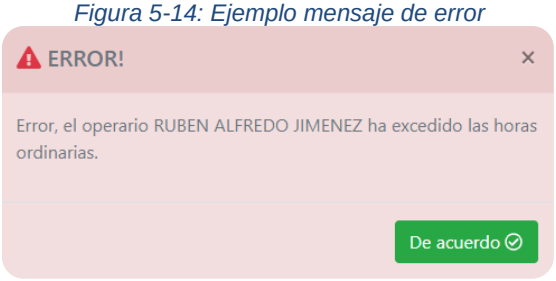
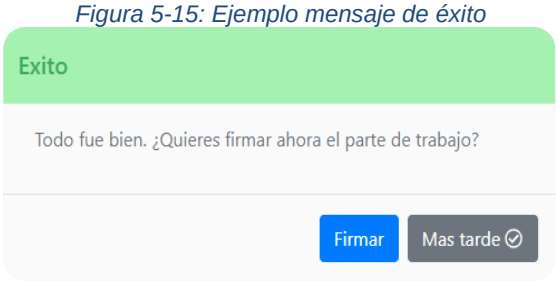


Figura 5-15: Ejemplo mensaje de éxito



Si todo ha sucedido de forma satisfactoria en el mensaje de éxito tendremos dos opciones: Pulsar el botón *Firmar* que nos redirigirá a la vista *Firmar parte de trabajo* o pulsar el botón *Más tarde* que nos redirigirá a la vista *lista de partes.*

Por último, la vista contiene 3 botones más en cada fila de operario que no sirven para realizar ninguna acción, si no, simplemente, para mostrar u ocultar campos, en principio, no obligatorios de rellenar:

Botón vehículo: El botón vehículo nos muestra los campos relacionados con el traslado al lugar del trabajo. Los campos tipo de vehículo, tipo de kilómentros y kilómetros estan relacionados, por lo que, si se selecciona un valor en alguno de ellos

los demás deben contener algún valor también. En caso contrario, la aplicación los dará como inválidos y no dejará finalizar el parte de trabajo.

Botón pernocta: El botón de pernocta nos muestra un desplegable con los tipos de pernocta. Por defecto no hay ninguno seleccionado pero se pueden seleccionar los valores *Pernocta en nacional* y *Pernocta en el extranjero*

Botón dietas: Al igual que el anterior, el botón de dietas muestra un desplegable con los tipo de dietas seleccionables. Por defecto no hay ninguno seleccionado, pero se pueden seleccionar los valores *Media dieta*, si sólo se ha comido o *Dieta completa* si además se ha tenido que cenar fuera durante el trabajo.

Figura 5-16: Botón vehículo

Figura 5-17: Botón pernocta

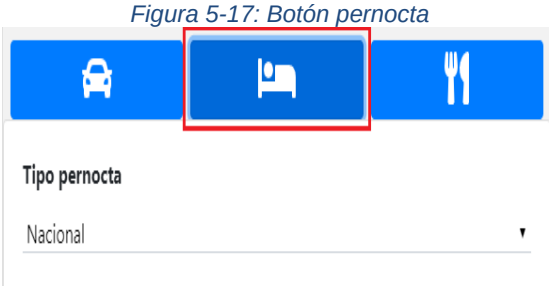


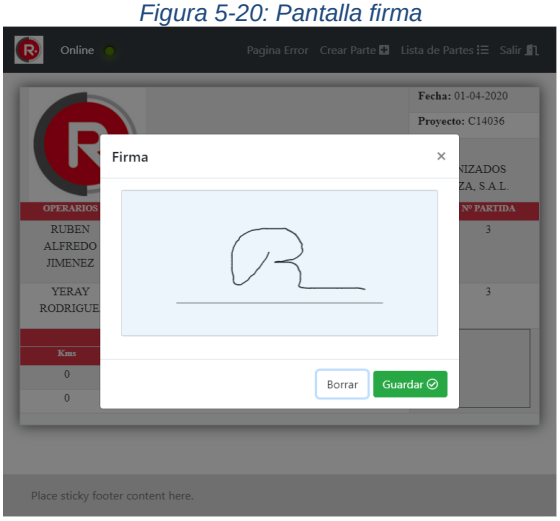
Figura 5-18: Botón dietas

5.1.4. Vista firmar parte de trabajo

Finalmente, la última de las vistas es la vista *Firmar parte de trabajo. A esta vista* se puede acceder si hemos pulsado el botón *Mostrar parte* en la vista *Lista de partes* o si hemos pulsado el botón *Firmar* en el mensaje de éxito de la vista *Inserción de datos de operarios.*

Figura 5-19: Opciones vista firmar parte de trabajo

Figura 5-20: Pantalla firma



Aquí se muestra el resumen de todos los datos que hemos introducido en el parte de trabajo en una tabla, para que de esta forma, el cliente pueda visualizarlo de una forma más amigable y pueda fírmalo. Es importante señalar que un parte de trabajo siempre estará disponible para firmar independientemente de si las fichas de trabajo se han archivado ya en administración o no. Para poder firmar el parte de trabajo debemos pulsar en el rectángulo donde debería encontrarse la firma, esto hará que una pantalla se desplegué con una cuadrícula en la que podremos introducir la firma.

Una vez dibujada la firma deberemos darle al botón *guardar,* si no pulsamos este botón y cerramos la ventana de algún otro modo, la firma no se añadirá al parte de trabajo, sin embargo, la firma se mantendrá en la venta si volvemos acceder. Además de este último botón, disponemos también del botón de *borrar,* que eliminará la firma que hayamos dibujado en la cuadrícula en el caso en el que nos hayamos equivocado.

Inmediatamente después de guardar la firma, aparecerá en el margen inferior de la pantalla el botón de *enviar* para poder enviar la firma al servidor. Una vez mandado, si la firma se guarda correctamente nos aparecerá un mensaje de éxito que tras cerrarlo nos redirigirá a la vista *Lista de partes.* En caso contrario, nos aparecerá un mensaje de error describiéndonos el problema por el que no se ha podido guardar la firma y tras cerrarlo nos dará la oportunidad de volver a intentar mandar la firma si lo deseamos. Es importante recalcar que no aparecerá el botón de firmar si no hay una firma en la tabla del parte de trabajo.

5.2. Apartado “Averías”

Al igual que el apartado de partes de trabajo sirve para gestionar los mismos, el apartado de averías está pensado y diseñado para gestionar las averías. En virtud de lo cual, este apartado de la aplicación permite la búsqueda, creación, modificación y anulación de las averías. Para este fin, la aplicación dispone de las siguientes vistas:

Figura 5-21: Vista listado de averías

Figura 5-22: Vista nueva avería

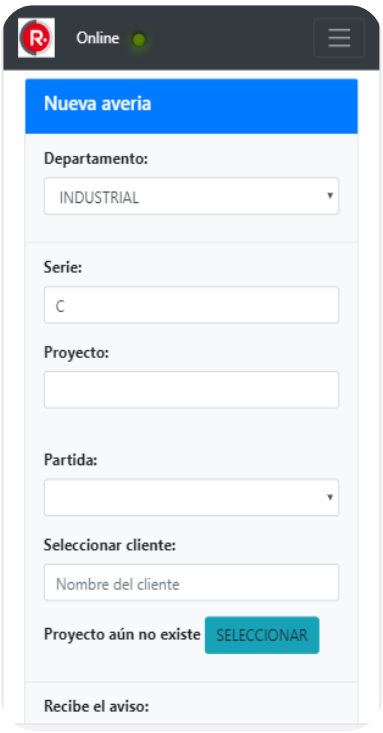
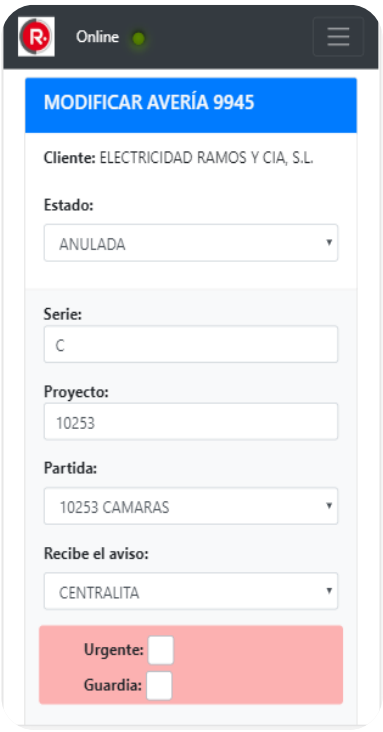


Figura 5-23: Vista modificar avería



5.2.1. Vista listado de averías

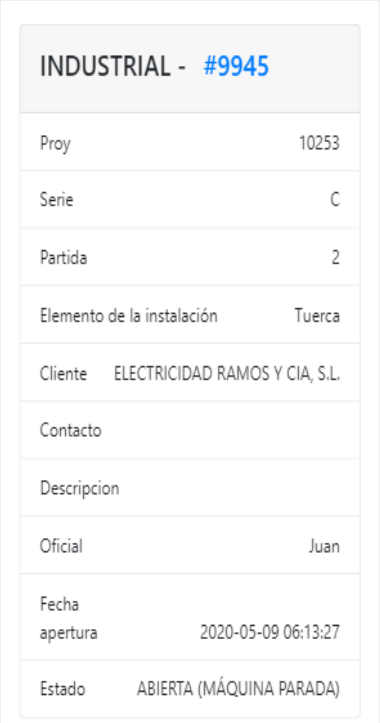
La vista *listado de averías* esta compuesta por dos partes, una primera en la parte superior para filtrar y buscar las averías que deseemos y, usa segunda, en la parte inferior para mostrar la lista de averías. Esta lista se muestra de manera diferente en pantallas de tamaño grande y tamaño pequeño. Para las pantallas de tamaño grande la información se muestra mediante una tabla y en pantallas de tamaño pequeño se muestra mediante tarjetas:

Figura 5-24: Listado averías en formato tabla

Dpt	Avería	Serie	Proyecto	Partida	Elemento de la instalación	Cliente	Contacto	Descripcion	Oficial	Fecha apertura	Ur	Estado
INDUSTRIAL	9945	C	10253	2	N/D	ELECTRICIDAD RAMOS Y CIA, S.L.			Juan	2020-05-09 06:13:27		ANULADA
ETXEA	9946	T	20014	2	N/D	MOSQUEIRA BARRIOLA IMANOL			JULEN GALARZA	2020-05-11 08:47:11		ABIERTA (MÁQUINA PARADA)

Ambas vistas incluyen la misma información y ambas disponenen de un link, para cada una de las averías disponibles, en el número de avería. Mediante este link es posible acceder a la vista *modificar avería.* En el caso de las pantallas de tamaño pequeño los links se encuentran en la cabecera de cada una de las tarjetas. Para las pantallas de mayor tamaño los links se encuentran en la columna con título "Avería". Además de estos links, la vista posee un link más en la parte superior derecha de la misma. Este link como su propio texto indica sirve para abrir una nueva avería, por lo que, al pulsar en él nos redirigirá a la vista *nueva avería*.

Figura 5-25: Listado de averías en formato tarjeta



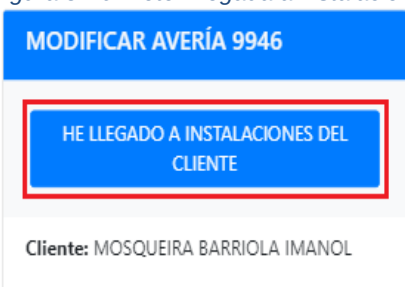
5.2.2. Vista nueva avería

Esta vista está compuesta por un único formulario que permite la introducción de los datos necesarios para crear una nueva avería. Si se observa con atención, se puede ver que estos datos están divididos en dos partes. En un primer lugar, los datos que se deben rellenar de manera obligatoria para todas las nuevas averías y en segundo lugar, los datos necesarios para crear una avería para un nuevo cliente. La única particularidad de este formulario es que al seleccionar el operario asignado a la avería podemos elegir la opción de "subcontrata". Si seleccionamos esta opción nos aparecerá un nuevo campo en el que podremos introducir la persona subcontratada.

5.2.3. Vista modificar avería

Al igual que la vista anterior, esta vista está compuesta por un formulario que, en este caso, permite modificar una avería previamente creada. La primera vez que entremos a esta vista después de crear la avería aparecerá un botón en la parte superior del formulario. Al pulsarlo, indicaremos a la aplicación que hemos llegado a la instalación del cliente y ésta actualizará la "fecha de respuesta" en la base de datos. De esta manera se podrá saber el tiempo transcurrido desde que se creó la avería hasta que se llegó al lugar para repararla.

Figura 5-26: Botón llegada a instalación



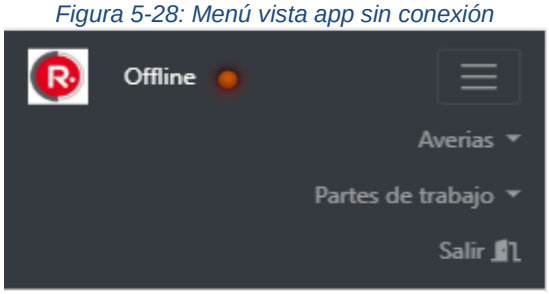
Además de modificar la información de la avería, en este formulario se puede cambiar el estado de la misma. Es desde ese apartado donde se puede “eliminar” una avería, pero, al contrario que con los partes de trabajo aquí no eliminamos la avería sino que cambiamos su estado a anulada. Igualmente, se puede cambiar a otros estados como “pendiente de material” o “cerrada” entre otros. Todos estos cambios de estados se guardan en la base de datos y pueden ser consultados en la parte inferior de la vista mediante una tabla que muestra el seguimiento de la avería.

5.3. Menú de la aplicación

Fuera de las vistas tenemos un elemento constante a lo largo de toda la aplicación que es el menú. Este está compuesto por dos desplegables, uno para el apartado de partes de trabajo y otro para el apartado de averías. Gracias al primero podemos navegar desde cualquier parte de la aplicación a la vista *lista de partes* para gestionar todos nuestros partes de trabajo que tenemos pendientes y a la vista *inserción de datos* generales para crear partes de trabajos nuevos. De igual manera, gracias al segundo, podemos acceder a la vista *listado de averías* y a la vista *nueva avería.* Además de los dos desplegables, posee también la opción de cerrar la sesión para las ocasiones en las que, por ejemplo, queramos cambiar de usuario. Por último, el menú contiene un indicador de conexión. Éste nos indica cuando tenemos conexión a internet y, en consecuencia, podemos usar todas las funcionalidades de la aplicación y cuando nos encontramos sin conexión y por tanto, solo podemos acceder a algunas de las funcionalidades.

Figura 5-27: Menú vista web con conexión

Figura 5-28: Menú vista app sin conexión



En los casos en los que no disponemos de conexión, siempre podremos acceder a la vista inserción de datos generales y a la vista inserción de datos de operarios para crear un nuevo parte de trabajo. Sin embargo, al resto de vistas de la aplicación solo se podrá acceder si previamente hemos accedido a ellas, y por tanto, las hemos guardado en cache.

Además, mientas nos encontremos en estado offline solo será posible realizar partes de trabajo para el día actual. Sí intentamos crear un parte de trabajo para cualquier otro día, la aplicación nos mostrará un mensaje de aviso, diciéndonos que no es posible la creación del mismo. De igual forma, si accedemos a una página que no hayamos visitado antes, aparecerá una página por defecto anunciándonos que no podemos acceder a ella sin conexión.

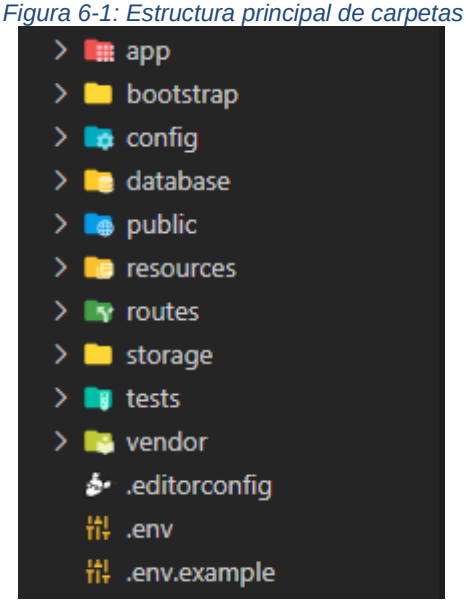
6. Funcionamiento interno de la PWA

En este capítulo se pretende explicar los detalles de la creación de la aplicación y las decisiones que se han tomado durante el transcurso del desarrollo. Para este propósito, en primer lugar se explicará de forma breve la estructura de carpetas que tiene la aplicación, posteriormente se explicará cómo se han desarrollado el sistema de inicio de sesión y los apartados de partes de trabajo y averías. Por último, se explicará algunos aspectos generales de la aplicación.

6.1. Estructura de carpetas

Antes que nada, para poder entender el funcionamiento de la aplicación, en primer lugar, hay que entender el funcionamiento de Laravel y, por tanto, la estructura de carpetas de la aplicación. Las carpetas principales que el proyecto posee son las mostradas en la imagen de la derecha. Para no hacer demasiado extenso el apartado nombraremos las más importantes y su contenido:

Figura 6-1: Estructura principal de carpetas



App: Es la carpeta principal del lado del servidor. Dentro de esta carpeta se alojan múltiples carpetas, sin embargo, las más importantes son la que contiene a los controladores (/app/http/controllers), la carpeta que contiene los modelos (app/models) y la carpeta que contiene las funciones de las que hacen uso el controlador y los modelos (app/helpers).

Public: Dentro de este directorio encontraremos los recursos de los que hará uso el usuario, es decir, archivos CSS, JavaScript, imágenes, el SW y el manifiesto de la aplicación.

Resources: Esta carpeta engloba distintos tipos de recursos como assets, archivos de idioma y la más impórtate en nuestro caso, la carpeta views. Esta carpeta contiene las vistas de la aplicación, es decir los archivos blade.php. Estos archivos combinan código HTML y PHP gracias al sistema de plantillas Blade [18] de Laravel.

Bootstrap: Permite el sistema de arranque de Laravel, es un directorio que nunca se ha tocado durante el desarrollo y que en principio no se debería tocar.

Routes: La carpeta Routes contiene un archivo muy importante llamado web.php. Este archivo será el archivo al que lleguen todas las peticiones y el que redirigirá a los controladores y funciones adecuados para cada petición.

Archivo ENV: Este archivo contiene la definición de las variables de entorno. Aquí está especificado la conexión a la base de datos, la conexión con el servidor de correo, si se está ejecutando la aplicación en modo debug… entre muchos otros.

6.2. Página de inicio de sesión

Una vez que entendemos y sabemos dónde se ubican todos los archivos de nuestra aplicación. Es momento de entender las funcionalidades que estos otorgan a la aplicación. La primera que veremos nada más entrar, es el inicio de sesión de la misma. Esto es a causa del SW, este comprueba siempre que hacemos una petición al servidor si hemos iniciado sesión o no. Si no tenemos una sesión activa en el servidor el SW nos devolverá desde cache, si no hay conexión o desde el servidor, si la hay, la página de inicio de sesión independientemente de la petición HTML que hayamos realizado.

Al rellenar los campos de nombre de usuario y contraseña y darle al botón *entrar* se hace una petición que comprueba en base de datos si existe algún usuario con ese nombre y contraseña, y se devuelve una respuesta positiva o negativa al SW. En el caso de recibir una respuesta positiva, el SW encripta los datos introducidos por el usuario y los guarda en la base de datos local para posibilitar el inicio de sesión offline en el futuro. Tras ello, el mismo SW es el que redirige a la vista de *lista de partes* al usuario. Gracias a este sistema tras iniciar sesión una vez, si en algún momento nos quedamos sin conexión podemos continuar iniciando sesión con ese usuario para acceder a la aplicación.

6.3. Apartado “Partes de Trabajo”

Como se detalla en el capítulo 2 (Visión global del proyecto), este es el principal apartado que debíamos desarrollar para hacer posible la gestión de los partes de trabajo. Para este propósito la principal tarea que debe permitir realizar la aplicación es la creación de partes de trabajo.

6.3.1. Creación de partes de trabajo

En primer lugar, cuando se inicia sesión, la aplicación descarga los datos necesarios para crear los partes de trabajo en el modo offline. Por un lado, se descargan todas las vistas necesarias para trabajar offline, que se guardan en cache, y por otro lado, se descargan todos los datos que necesita la aplicación para realizar las validaciones, tanto para los campos autocompletables, como para otros

propósitos. Estos datos al contrario que las vistas se guardan en la base de datos local como objetos.

Para realizar esto, dentro de la función *ready* de la vista de *lista de partes* se hacen varias peticiones AJAX. Estas peticiones son interceptadas por el SW, que a su vez, realiza la petición al servidor. Para finalizar, el SW introduce en la base de datos local los datos recibidos por el servidor y devuelve un objeto respuesta a la aplicación. Si esta respuesta indica que todo ha salido bien, la aplicación simplemente continua con su funcionamiento y, si por el contrario, la respuesta indica que algo ha salido mal, la aplicación muestra un aviso de error para que el usuario sea consciente.

Estas peticiones no son necesarias hacerlas desde la aplicación. El propio SW a la vez que capta la petición que se hace al servidor para mostrar la vista de *lista de partes* podría guardar esta información, sin embargo, como no sabemos cuánto tiempo puede llegar a costar esto, debido a que la información varía, se ha decidido hacer las peticiones una vez cargada la aplicación. De este modo, se agiliza el primer paso de visualización y posteriormente se muestra al usuario un spinner que bloquea la aplicación mientras se hacen estas operaciones. Esto permite a la aplicación dar una sensación de mayor fluidez y elimina la sensación de bloqueo.

Asimismo, antes de insertar ningún objeto en la base de datos local, se comprueba si ya existe un objeto exactamente igual. Esto es debido a que para insertar datos se utiliza un método llamado *put*. Este método introduce el objeto que recibe como argumento en la base de datos si no existe, pero si existe, lo actualiza. La actualización de elementos, tal y como está pensada para las bases de datos de los navegadores, es un inconveniente para nuestra aplicación. Para actualizar un objeto se *borra* el objeto anterior y se introduce el nuevo. Sin embargo al borrar no se elimina el objeto de la base de datos, sino que, se añade un campo para hacerle saber a la base de datos que ese objeto ya no es válido. Esta situación es de gran utilidad cuando hay problemas y se quiere averiguar todo lo que ha ocurrido, sin embargo, en nuestro caso, dejar objetos antiguos significa incrementar el tamaño de la aplicación. Por esta razón antes de intentar insertar ningún objeto se comprueba su existencia en la base de datos, y si existe, no se hace nada.

Una vez descargado los datos, la aplicación contiene ya en cache la vista necesaria para poder crear los partes de trabajo. Esta vista distingue entre averías y OT, para conseguir este propósito, se han creado dos formularios. Los formularios están compuestos por 2 partes cada uno: la parte de inserción de datos generales y la parte de inserción de operarios. Para evitar repetir código se han creado 3 módulos, uno para la inserción de datos generales de averías, otro para la inserción de datos generales de OT y un tercero común para los dos formularios que contiene la parte de inserción de datos de operarios. Estos módulos son unidos en un solo formulario gracias a la función *include* de Laravel. Esta función nos permite juntar dos archivos

Blade independientes. En consecuencia, los archivos Blade que contienen los formularios son muy similares y presentan la siguiente forma:

Figura 6-2: Código formulario OT

```
<form id='formularioOT' action="{{$ruta}}" method="POST"
enctype="multipart/form-data" class="">
    <div id='ot_parte1'>
        @include('layouts.modulo-OT')
        @include('layouts.footer-modulo-OT')
    </div>
    <div id='ot_parte2'>
        @include('layouts.modulo-operario',['tipo' => 'ot'])
        @include('layouts.footer-modulo-operario')
    </div>
</form>
```

Como se puede ver en la imagen anterior, en el método *include* del módulo de operario se añade un parámetro más, el parámetro *tipo*. Esto es debido a que, al repetir el mismo código, los IDs de los diferentes divs se duplican haciendo imposible trabajar con ellos con código JavaScript. Para solventar este problema cada ID tiene un parámetro *tipo* que es reemplazado, gracias a la declaración *{{ }}* de Blade, por la palabra *OT* o *Avería* para diferenciar a que div estamos apuntando.

Figura 6-3: Ejemplo de ID

```
<select id='pg-num-partida-{{$tipo}}-fila-{{$i+1}}'
```

Además de esto, al estar, la aplicación, pensada para usarse en cualquier tipo de dispositivo, los 3 módulos disponen de una vista para pantallas de tamaño grande y otra para pantallas de tamaño pequeño. Estas son mostradas dependiendo del tamaño de la pantalla gracias a las propiedades de visualización de Bootstrap [19]. Por esta razón todos los divs de los formularios comienzan con las letras *PG* o *PP* en referencia a *Pantalla Grande* o *Pantalla Pequeña*, de nuevo, para que desde el código JavaScript se pueda diferenciar entre ambos. Por último en el módulo de inserción de operarios existe también un número para saber a qué operario pertenece el div. Todo lo anterior en conjunto, finalmente, permite que cada ID de la página web sea único.

Con todo lo anteriormente hecho, los dos formularios, tanto el de averías como el de OT, son incluidos también con el método *include* en la página web que finalmente es la que realmente es llamada.

Una vez que la página está construida todas las partes son mostradas simultáneamente. Para poder ocultar y mostrar solamente las partes que queremos, existen 4 botones. Dos botones en la parte superior de la página que nos ocultan las otras 3 partes y nos muestran o bien la parte 1, inserción de datos generales, de OT,

o la parte 1 de averías. Y dos botones, *siguiente,* en la parte inferior. Estos actúan de la misma forma ocultando las partes que no queremos ver y mostrando la parte de inserción de operarios de cada respectivo formulario. Adicionalmente, cuando se pulsa alguno de los botones *siguiente*, antes de ocultar y mostrar la otra vista, también, se comprueba si se han cambiado los valores de los campos *fecha* o *número de obra*:

1. Si se ha cambiado la fecha, para cada uno de los operarios:
   1.1. Se hace una consulta a la base de datos y se comprueba si el usuario está de guardia en la nueva fecha.
      1.1.1. Si está de guardia y en la anterior fecha no estaba de guardia se añade el campo guardia.
      1.1.2. Si no está de guardia y en la anterior fecha estaba de guardia se elimina el campo guardia.
      1.1.3. En otro caso no añade ni elimina nada.
2. Si ha cambiado el número de obra, para cada uno de los operarios:
   2.1. Clonamos las opciones de la fila 1, que previamente ya se habían cambiado al seleccionar la nueva obra.
   2.2. Borramos las anteriores opciones.
   2.3. Copiamos las nuevas.

En los casos en los que no hay conexión de Internet, el SW se encarga de gestionar el error de conexión mediante una función *catch* y volver a hacer la consulta pero en este caso a la base de datos local. Evidentemente, la base de datos local no puede contener todos los datos que dispone la base de datos del servidor, en consecuencia, en el momento de la redacción de este documento la creación de partes de trabajos offline está limitada al día actual. En próximas iteraciones es muy posible que este periodo de tiempo se amplié a varios días.

Asimismo, la aplicación muestra sugerencias y valida los diferentes campos a rellenar. Para la primera tarea la aplicación hace uso de peticiones Ajax mediante el widget Autocomplete de jQuery [20]. Por un lado, Ajax nos permite que una página web que ya ha sido cargada solicite nueva información al servidor de manera asíncrona y, por otro lado, el widget Autocomplete muestra al usuario una lista de valores rellenada con los resultados y permite seleccionar uno de los valores. Estas peticiones funcionan de la siguiente manera:

1. Mediante la función Autocomplete, cada vez que el usuario teclea una letra, se captura la solicitud de búsqueda.
2. Se crea una petición asíncrona de Ajax con los datos a una URL.
3. La petición es capturada por el SW, que a su vez hace la misma petición al servidor.

3. El archivo Web.php del servidor recibe la petición y valida si esa URL es válida.
4. Si es válida, este archivo redirige la petición al controlador y función asociados a la URL.
5. El controlador llama a un modelo que realiza la petición a la base de datos y devuelve los datos en formato JSON al SW.
   5.1. Si todo lo anterior ha ido bien y la respuesta ha devuelto un estatus 200:
      5.1.1. El SW devuelve los datos a la aplicación.
   5.2. Si ha devuelto cualquier otro estatus de error (400, 404, 500…):
      5.2.1. El SW realiza una petición a la base de datos local con los mismos datos.
      5.2.2. El SW devuelve los datos a la aplicación.
6. Autocomplete muestra los resultados para que el usuario pueda seleccionarlos.

Para la segunda tarea la aplicación comprueba si se ha seleccionado alguna de las sugerencias que Autocomplete ha mostrado, si no se ha seleccionado ninguna, automáticamente invalida el campo, en caso contrario, lo valida.

Aparte de estas tareas la aplicación también permite añadir, duplicar y eliminar filas de operarios. Para este propósito al mostrar la página por primera vez, mediante JavaScript la aplicación clona la fila del primer operario, elimina todos los datos que pudiera tener insertados y lo guarda en una variable global para poder usarlo posteriormente.

Cuando el usuario selecciona el botón *añadir* se añade la fila clonada debajo de la última fila de operario ya existente. Sin embargo, antes de hacer esto se cambia en los IDs de todas las filas el número de fila, ya que, al ser una fila clonada todos los IDs hacen referencia a la primera fila.

Cuando el usuario selecciona el botón *duplicar,* al contrario que en el anterior no se coge la fila clonada que tenemos guardada, si no, la fila en la que se encontraba el botón. Una vez clonada esta fila al igual que antes, se cambian los IDs y se inserta al final.

El botón *eliminar,* elimina la fila en la que se encuentra el botón y cambia los IDs de todas las filas que hubiera posteriormente.

Finalmente, cabe destacar también, que como se ha mencionado anteriormente, los formularios poseen dos vistas, esto implica que los input de ambas vistas son diferentes. Para que no se pierda nunca información cuando se cambia de una vista a otra, todos los input PG están relacionados con los input PP mediante JavaScript

de manera que, cuando se escribe o se cambia un valor en uno de ellos automáticamente se copia en el otro.

Al pulsar en el botón *finalizar* de cualquiera de los formularios, en primer lugar se comprueba si algún campo obligatorio está vacío. Si es el caso, se señala al usuario que rellene los campos. En segundo lugar, si no existen campos vacíos se comprueba si todos los campos están validados, en caso contrario se le indica al usuario que modifique los campos no válidos.

Una vez comprobado esto, si todo es correcto, comienzan los preparativos para mandar el formulario. En este caso, se pulsan automáticamente todos los input de tipo *checkbox* y se pone como valor *off,* este valor indica en el servidor que el *checkbox* no estaba seleccionado. Esta acción se realiza, dado que, si no se seleccionan los *checkbox*, no se envía ninguna información al servidor, esto conlleva una pérdida de información fatal para nuestros intereses.

A continuación se crea una petición AJAX en la que se envía toda la información del formulario. Sí hay conexión, esta petición consulta en el servidor si alguno de los operarios se ha pasado de las horas ordinarias para el día indicado. El servidor comprueba esta información sumando, para cada usuario, las horas de todas las fichas de trabajo que haya en la base de datos para ese día, más, las horas indicadas en el nuevo parte de trabajo. Si alguno de los operarios ha sobrepasado las horas se devuelve un mensaje de error que se muestra mediante una ventana modal indicando el operario que ha sobrepasado las horas y el número de horas por las que ha excedido las horas ordinarias. De esta forma el operario puede saber cuántas horas añadir a horas extra y quitar de horas ordinarias.

Si por el contrario, ninguno de los operarios ha sobrepasado las horas, se intenta mediante una transacción: insertar en la base de datos el parte de trabajo, crear para cada fila de operarios que hubiese, una ficha de trabajo e insertar las filas que relacionan las fichas de trabajo y el parte de trabajo en otra tabla aparte. Si algo de todo esto falla se hace un rollback, es decir, se devuelve a la base de datos al estado previo que tenía antes de introducir nada. Posteriormente, se devuelve un mensaje de error para avisar al usuario. A la vez que se muestra el mensaje de error, se deseleccionan los input de tipo checkbox para que el usuario vuelva a ver el formulario como lo dejo.

En los casos en los que se produce un error al enviar la información, ya sea porque no se dispone de conexión, porque el servidor está caído o por cualquier otro motivo por el que el servidor no pueda analizar la petición, el SW se encarga de gestionar ese error nuevamente con un *catch.* Dentro de este se comprueba si el nuevo parte de trabajo es para el día actual. Si no lo es, se devuelve un mensaje de error, ya que, el SW no posee de la información necesaria para realizar las comprobaciones de las horas de los operarios para otros días. Por el contrario, si el nuevo parte de trabajo es

para el día actual, el SW realiza las comprobaciones con los datos de la base de datos local. Sí las comprobaciones no dan ningún error se guarda el parte de trabajo en la base de datos local, se crea una tarea asíncrona para cuando vuelva la conexión y como se explicará un poco más adelante se devuelve un mensaje de éxito. Esta petición asíncrona se lanzará en el momento en el que se recupere la conexión a internet y mandará una petición para guardar el parte de trabajo en la base de datos del servidor. Una vez guardado en el servidor el parte de trabajo se borrará de la base de datos local.

En el caso en el que todo sale bien, si no hay conexión se le informa al usuario que el parte de trabajo de momento solo ha podido ser guardado en la base de datos local. Si no es así y disponemos de conexión, se devuelve un mensaje de éxito al usuario en el que se le da la posibilidad de volver a la vista *lista de partes* o a la vista *firmar parte de trabajo* para firmarlo. Si se redirige a la vista de firmar el parte de trabajo, se vuelve a enviar todo el formulario. De esta forma el controlador simplemente redirige con la información enviada a la vista *firmar parte de trabajo* y no se pierde tiempo y recursos en volver a solicitar la información a la base de datos.

6.3.2. Modificación de partes de trabajo

Inmediatamente después de crear un parte de trabajo, es posible volver a modificarlo, siempre y cuando se tenga conexión y no se haya firmado o se haya pasado a administración. Para comprobar esto último, la tabla de partes de trabajo dispone de un campo llamado *estado*. Este campo tendrá el valor 0 cuando se crea el parte de trabajo y ese mismo deberá ser el valor que tenga para poder modificarlo, debido a que, cuando se listan los partes de trabajo ese es el campo que examina la aplicación para mostrar los botones de modificar y eliminar el parte de trabajo.

Al pulsar el botón modificar, este hace una petición al servidor con el ID del parte de trabajo seleccionado, el servidor busca en la base de datos el parte de trabajo, las fichas de trabajo asociadas a este y reconstruye la información para enviársela a la aplicación junto con la vista. En este caso, no es necesario distinguir entre averías y OT como era necesario al crearlo. Además, para modificar el parte de trabajo en el servidor es necesario realizar una petición distinta. A causa de todo lo anteriormente comentado, el archivo enviado es distinto. Este archivo solo contiene el formulario de averías o de OT, dependiendo de que fuera el parte de trabajo, y el formulario hace una petición al servidor para modificar el parte de trabajo en vez de para crearlo. Sin embargo, visualmente los formularios son iguales. Para lograr esto el archivo simplemente incluye uno de los formularios, los formularios a su vez incluyen los dos módulos como se muestra en la ilustración 6-2, y específica a que URL hacer la petición de la siguiente forma:

Figura 6-4: Código para incluir el formulario avería

```
<div id='form_averia'>
    @include('layouts.formulario_averia',
    ['ruta' => '/partes1/mostrar_parte_trabajo',
    'idFormulario'=> 'modificarFormularioAveria'])
</div>
```

En los formularios, al igual que se hace con los ID, como se ha explicado en el apartado anterior, se rellenan los datos si estos son entregados gracias a la declaración "*{{ }}*" del sistema de plantillas Blade de la siguiente manera:

Si existe la variable en el lado izquierdo de los símbolos "*??*" inserta su valor en el campo *value* del input, en caso contrario inserta el valor de la derecha.

Figura 6-5: Ejemplo condicional básico Blade

```
<input name="cliente" value='{{$datos["cliente"] ?? ""}}'
```

De esta forma, al haber mandado previamente los datos del parte de trabajo en la respuesta, rellenamos todos los campos necesarios. Una vez hecho esto, la aplicación se comporta exactamente igual a como se ha descrito en el apartado anterior.

6.3.3. Firma de partes de trabajo

Tras finalizar de crear o modificar un parte de trabajo o en la lista de partes de trabajo pendientes, damos la opción de cambiar a la vista *firmar parte de trabajo* para firmarlo. Esta vista está formada por una tabla en la cual se cargan los datos de manera idéntica al de los apartados anteriores. Adicionalmente, la vista también posee un botón en la parte inferior derecha que nos permite desplegar una ventana modal en la que se permite firmar.

Esta funcionalidad es posible gracias a la librería Jsignature de JavaScript [21]. Para que el usuario pueda firmar, creamos un componente Jsignature en el div que elijamos (en nuestro caso la venta modal contiene ese div) de la siguiente manera:

Figura 6-6: Código de la librería Jsignature

```
$(document).ready(function(){
    $("#signature").jSignature({lineWidth: 1,height:200, width:400});
    $("#signature").find(".jSignature").css("width", "100%")
```

Una vez hecho esto, el usuario es capaz de dibujar dentro del div que hemos indicado. Para guardar el dibujo que realiza el usuario, existe un botón *guardar*, al pulsarlo se comprueba si se ha dibujado algo:

1. Si se ha firmado:
   1.1. Se guarda la imagen, gracias a la función getData de la librería, en una variable llamada *png*.
   1.2. Se crea un elemento *img*, y se le añade al campo src los datos que contiene la variable *png*.
   1.3. Se añade un listener al elemento *img* para que vuelva a mostrar el modal si se pincha en él.
   1.4. Si ya existe una firma previa se borra de la tabla.
   1.5. Se añade el nuevo elemento a la tabla y se muestra el botón de enviar firma.
2. Si no se ha firmado:
   2.1. Se borra la firma de la tabla si está presente.
   2.2. Se borra el botón de enviar firma de la tabla si está presente.

Tras firmas el parte de trabajo y pulsar el botón *enviar firma* se manda un formulario al servidor mediante una petición Ajax con el id del parte de trabajo que se está firmando y la imagen de la firma. Posteriormente la firma es guardada en un directorio del servidor y se anota la ruta de la misma en la base de datos, en el parte de trabajo correspondiente, gracias al id del parte de trabajo, y conjuntamente, se cambia el estado del parte de trabajo a firmado para que no vuelva aparecerle al operario en la vista *lista de partes*. Si todo se realiza sin problemas la petición Ajax devuelve un mensaje de éxito y muestra un modal para hacérselo saber al operario, si no, se devuelve un mensaje de error y se muestra un modal comunicando el error.

6.4. Apartado “Averías”

Este apartado de la aplicación no estaba previsto en un primer momento. No obstante, durante el desarrollo de la aplicación se descubrió la necesidad de posibilitar la creación y gestión de las averías dentro de la aplicación para algunos usuarios en concreto. Aunque en el momento de la redacción de este documento no esté habilitado la limitación de entrada a este apartado por permisos, la idea en el futuro es hacerlo así. Es importante destacar también que este apartado no está pensado para ser capaz de funcionar de manera offline, al menos de momento. Esto significa que se hace un uso mucho menor del SW, haciendo que el funcionamiento de este se asemejé, prácticamente por completo, al funcionamiento que tendría en una página web común. Al igual que con los partes de trabajo, para poder gestionar las averías lo primero que necesitamos es una vista que nos permita crearlas.

6.4.1. Creación de averías

Al contrario que con los partes de trabajo, con las averías no necesitábamos diferenciar distintos tipos, en consecuencia, la vista de averías es un único formulario. El funcionamiento de éste es muy similar al funcionamiento de los formularios del apartado anterior. Mediante el campo partida podemos buscar una obra existente en

la base de datos y seleccionarla. Esto se hace a través de una petición Ajax, con el widget Autocomplete de jQuery, de la misma forma descrita en el apartado anterior. Al seleccionarla se lanza otra petición Ajax para buscar los números de partida asociados a esta obra y se rellena el campo partida con los números de partida devueltos por el servidor.

Del mismo modo, se hace la búsqueda de los operarios cada vez que se teclea una letra nueva para poder asignar uno a la avería.

Una vez están rellenados, al menos, todos los campos obligatorios del formulario, al pulsar el botón *abrir avería*. Se mandan al servidor los datos mediante una petición Ajax y se guardan en la base de datos. Si todo ha salido bien se devuelve un mensaje de éxito y se muestra al usuario una ventana modal de éxito. Si no es así, se manda un mensaje de error y se le muestra al usuario una ventana modal de error.

6.4.2. Listar averías

Inmediatamente después crear una avería se puede listar para ver sus datos o modificarla en la vista *lista de averías.* La vista *nueva avería* está formada por un buscador, que en esencia es un formulario, y una tabla que muestra los resultados de la búsqueda. En este caso al ser un apartado que debe gestionar un responsable de los operarios, y en el cual se puede juntar una gran cantidad de averías al mismo tiempo, se decidió que era necesario crear el buscador y no un simple listado como en la lista de partes.

Al introducir los datos por los que se quiere buscar y pulsar el botón *filtrar* se hace una petición al servidor que en este caso vuelve a mandar la misma página con los datos actualizados.

Por otra parte, la tabla contiene los datos básicos de cada una de las averías que se muestran. Estos se pueden ver en formato tabla en pantallas grande y en formato cards en dispositivos móviles. Cada una de las filas de la tabla hace referencia a una de las averías y cada una de las filas posee un link para redirigir a la vista *modificar avería.* Esto es posible gracias a que cada uno de los links envía mediante una petición post el ID de la avería. En virtud de esto, el servidor es capaz de buscar la avería en la base de datos, rellenar con estos mediante el sistema de plantilla Blade la vista y devolverla.

6.4.3. Modificar averías

La vista averías es nuevamente un formulario, que tiene un comportamiento muy similar al descrito en el apartado 6.3.2, *Modificación de partes de trabajo*. Cuando se hace una petición al servidor para modificar una avería se envía el ID de la misma, para que, de esta forma, el servidor pueda ser capaz de buscar en la base de datos la información de la avería, posteriormente rellenar la vista con los datos obtenidos mediante Blade y devolverla al usuario. Una vez que el usuario dispone de la vista en

su dispositivo, puede modificar cualquier campo del mismo y enviar una petición para actualizar esos datos en la base de datos mediante el botón *guardar cambios*. Al pulsarlo se lanza una petición con toda la información del formulario, en la que, nuevamente, contiene el id de la avería modificada. El servidor actualiza la base de datos con los nuevos datos enviados y además, envía un email a cada una de las direcciones de correo que se encuentran en la lista de destinatarios en la parte inferior de la vista. El envío de estos emails es posible gracias a la función “mail” de PHP.

Además de este botón, la vista dispone de dos más, el primero de ellos sirve para indicar que ha llegado a la instalación. Cuando se pulsa este botón, se manda otra petición al servidor, que en este caso solo actualiza la fecha de respuesta, y vuelve a mandar la misma vista con este dato actualizado. El segundo de estos botone es simplemente para agregar correos a la lista de destinatarios. Al pulsar este botón mediante JavaScript, simplemente añadimos al final de la lista el valor del campo que está seleccionado.

6.5. Aspectos generales:

Además de todas las librerías mencionadas, a lo largo de la aplicación se hace uso de varias librerías más. En primer lugar para escribir gran parte del código se ha usado jQuery [22], además de su widget Autocomplete como ya se ha mencionado. Para los iconos que se muestran a lo largo de la aplicación se ha usado la librería Fontawesome [23]. Además de eso, para los inputs de tipo time del apartado partes de trabajo se ha usado la librería TempusDominous [24]. La razón de usar está librería no fue estética, la razón real detrás de su utilización fue que a los inputs de tipo time en la aplicación de Google Chrome para Android no aceptaba la propiedad Step. En nuestro caso como queríamos limitar los minutos a la hora de seleccionar las horas trabajadas a lapsos de 15 minutos nos vimos forzados a buscar una librería como esta que nos diera esta posibilidad.

7. Informe de recursos destinados al proyecto

Ahora que se dispone de una visión global del trabajo realizado, durante este capítulo se mostrará el tiempo invertido en cada una de las tareas, se comparará con los tiempos estimados en un primer momento y se hará un análisis de las tareas y objetivos que se han cumplido satisfactoriamente, los que no y el porqué de los mismos.

Sin embargo, antes de realizar el análisis hay que entender el contexto inicial. En el momento de la creación de la tabla de estimaciones de tiempo del capítulo 3, el conocimiento que poseía tanto de JavaScript como de HTML era muy limitado y mi conocimiento de Laravel y PWA era nulo. Este desconocimiento se puede ver reflejado en las siguientes tablas comparando las horas estimadas y las reales. A continuación, para poder hacer un análisis exhaustivo y facilitar la lectura, se parte, como ya se ha mencionado, de la Tabla 3-1, presentada en el Capítulo 3 en la sección: “Estimación de tiempos”, desglosándola en varios apartados.

Durante la primera y segunda iteración se cumplieron los plazos y se dedicó el tiempo estimado a las tareas asignadas salvo por el estudio de la tecnología Laravel que resultó ser un framework que necesitó más horas de las estimadas para realizar un estudio sobre él. Esto derivó en que la cantidad de horas invertidas a investigar las posibles alternativas a la tecnología PWA para este trabajo fueran mucho menor.

Tabla 7-1: Tiempos reales 1ª y 2ª iteración

Iteración	Actividad	Horas estimadas	Horas reales
1	Investigación y estudio teórico de la tecnología PWA	80	80
2	Investigación y estudio de la tecnología Laravel	20	24
	Pruebas prácticas PWA	20	40
	Estudio de alternativas a PWA	40	16

En la tercera iteración se comenzó con el diseño de la aplicación. Esta tarea llevó mucho menos tiempo del planificado, puesto que en un primer momento solo se planteó para que contuviera dos formularios. Uno para crear los partes de trabajo y otro para firmarlos. Esto significó que se adelantaran a esta iteración tareas de la siguiente, y que además, se añadieran nuevas para agregar funcionalidades que no

se contemplaron en un primer momento como fueron el inicio de sesión con un usuario.

Tabla 7-2: Tiempos reales 3ª iteración

Iteración	Actividad	Horas estimadas	Horas reales
3	Realización del diseño de la aplicación	72	16
	Inicio de sesión online	-	8
	Programación del formulario partes de trabajo para que funcione cuando hay conexión.	-	34
	Programación de una función que detecte cuando se está online y cuando no.	-	6
	Realización de pruebas en busca de fallos y solucionarlos	-	6
	Redacción de la memoria del Proyecto	8	10

En la cuarta iteración estaba previsto que se iniciara la programación de la aplicación creando el formulario de partes de trabajo y la función que detectará y mostrará cuando la aplicación disponía de conexión y cuando no. Sin embargo, estas tareas se finalizaron en la anterior iteración, en consecuencia, se comenzó con tareas en principio previstas para la siguiente iteración. De todas las tareas de esta iteración, la que más tiempo conllevó fue la de validación de datos, a causa de la decisión tomada durante el desarrollo de no admitir la introducción de valores que no existieran en la base de datos. Adicionalmente, al ir adelantado con las previsiones que se tenían, la empresa decidió que ayudara en otros proyectos en los que se estaba trabajando.

Tabla 7-3: Tiempos reales 4ª iteración

Iteración	Actividad	Horas estimadas	Horas reales
	Programación del formulario partes de trabajo para que funcione cuando hay conexión.	52	-

4	Programación de una función que detecte cuando se está online y cuando no.	10	-
	Trabajo en otros proyectos de la empresa	-	25
	Realización de pruebas con la base de datos y descubrir necesidades a cubrir para la aplicación	20	4
	Añadir funciones de autocompletado a los formularios	42	16
	Añadir métodos para validar datos	-	29
	Realización de pruebas en busca de fallos y solucionarlos	10	8
	Redacción de la Memoria del Proyecto	8	12

Al comenzar la iteración 5 estaba una iteración completamente adelantado, por lo que, al disponer de tiempo, la empresa decidió que esta iteración la emplease en ampliar las funcionalidades de la aplicación. Las nuevas funcionalidades a añadir fueron la posibilidad de modificar los partes de trabajo ya creados y eliminarlos. Para este propósito, se necesitaba una forma de listar los partes de trabajo, en consecuencia, durante esta iteración se creó una nueva vista, la vista denominada *lista de partes*.

Tabla 7-4: Tiempos reales 5ª iteración

Iteración	Actividad	Horas estimadas	Horas reales
5	Añadir funciones de autocompletado a los formularios	42	-
	Realización de pruebas con la base de datos y descubrir necesidades a cubrir para la aplicación	20	-
	Programación vista lista de partes	-	20

	Programación funcionalidad editar partes de trabajo	-	30
	Añadir ventanas modals para los distintos avisos y spinner para cuando la aplicación está cargando	-	12
	Realización de pruebas en busca de fallos y solucionarlos	10	14
	Redacción de la Memoria del Proyecto	8	12

Durante la sexta iteración se dio comienzo a la programación del SW, sin embargo, en medio de esta, se decidió por parte de la empresa cambiar las prioridades y añadir una nueva funcionalidad para poder crear averías dentro de la aplicación. Esto provocó que se creara todo un nuevo apartado en la aplicación, pero también, que no fuese posible acabar la tarea del SW y que, por primera vez, durante el proyecto se estuviera retrasado en comparación con las previsiones que se tenía en el inicio.

Tabla 7-5: Tabla de tiempos 6 Iteración

Iteración	Actividad	Horas estimadas	Horas reales
6	Programación del Service Worker para añadirle funcionalidad offline al primer formulario.	62	32
	Añadir apartado averías a la aplicación	-	32
	Inicio de sesión offline	-	20
	Realización de pruebas en busca de fallos y solucionarlos	10	16
	Redacción de la Memoria del Proyecto	8	16

Finalmente las 2 últimas iteraciones se usaron para añadir la funcionalidad de creación de partes offline y dar inicio al estudio de la tecnología Vue para en futuras iteraciones poder añadir la funcionalidad de la firma del parte de trabajo offline.

Tabla 7-6: Tabla de tiempos 7ª y 8ª iteración

Iteración	Actividad	Horas estimadas	Horas reales
7	Programación del Service Worker para añadirle funcionalidad offline al primer formulario.	-	40
	Programación del Service Worker para añadirle funcionalidad offline al segundo formulario.	62	-
	Estudio tecnología Vue	-	14
	Realización de pruebas en busca de fallos y solucionarlos	10	14
	Finalización de la Memoria del Proyecto	8	12
8	Implantación de la aplicación	8	-
	Realización de la presentación del proyecto	10	18

En conjunto, se puede observar que las estimaciones de tiempos hechas en un primer momento fueron en general pesimistas, debido al desconocimiento inicial, y el temor de que las tareas fueran mucho más complicadas de lo que finalmente fueron. Sin embargo, el total de las horas invertidas en el proyecto fue de 636 horas frente a las 578 estimadas en un inicio. Este incremento de horas, se produjo principalmente durante las iteraciones 4, 5 y 6 con 14, 8 y 36 horas extra respectivamente.

Durante la cuarta iteración el incremento de horas se debió en primer lugar a las horas extra que se tuvo que dedicar a la memoria, que, como podemos ver en las tablas, fue una constante durante todo el resto de iteraciones. Para hacer la previsión de esta tarea me apoyé en los tiempos estimados por la universidad para la realización de la memoria (50 horas en total) que dividí posteriormente durante las iteraciones. Como se puede comprobar por las 80 horas invertidas en total, las previsiones de la

universidad tampoco fueron acertadas para este caso en particular. En segundo lugar, el incremento de horas también se debió a la tarea de la validación de datos, que fue más compleja de lo que se estimó en un inicio a causa de una decisión tomada durante el desarrollo.

A lo largo de la quinta iteración el incremento fue, de igual forma, a causa del mayor número de horas que se dedicaron a la memoria y a la tarea de búsqueda y solución de fallos. En esta última tarea no se previó en un primer momento que la necesidad de tiempo para solucionar y arreglar errores crecería a medida que el código escrito crecía también. Este incremento ocurrió principalmente porque en múltiples ocasiones solucionar un problema generaba otro.

Por último, durante la sexta iteración el incremento se produjo por el repentino cambio de prioridades para crear el nuevo apartado de averías y el esfuerzo personal por intentar finalizar con las demás tareas de la iteración a tiempo.

En conclusión, las previsiones pesimistas, debido al desconocimiento inicial sobre muchas de las tecnologías y tareas a realizar generaron que se realizaran múltiples tareas que no estaban previstas en un primer momento, provocando la creación de una aplicación mucho más grande que la que originalmente se pensó. Sin embargo, los cambios de prioridad durante el final del desarrollo provocaron que las tareas de las últimas iteraciones no se pudieran completar. Gracias al conocimiento obtenido durante el desarrollo de este trabajo, en previsiones futuras, se intentará prever tiempos un poco más cortos en general, exceptuando las tareas de búsqueda y solución de errores y documentación del proyecto.

8. Conclusiones

La tecnología PWA es un gran avance en el ámbito de la programación web, que además, como ya hemos visto a lo largo del documento, cuenta con la aprobación de los usuarios, ya que estos, consideran a esta una tecnología menos intrusiva debido a que les permite probar las aplicaciones sin tener que realizar ninguna instalación. Esto junto con los esfuerzos de compañías de la envergadura de Google, Firefox o Apple hace que su potencial sea enorme y aunque hoy en día, todavía no lo soportan todos los navegadores, haciendo que no todos los usuarios puedan disfrutar de ella, el trabajo de estas grandes compañías anteriormente mencionadas, hace pensar que el rumbo de todos los navegadores será incluir esta tecnología en el futuro.

Por otro lado, es cierto que el ahorro de tiempo, y por tanto de dinero, a la hora de desarrollar aplicaciones multiplataforma es real. Sin embargo, esto no es cierto en los casos en los que sólo se desea trabajar en una plataforma, como por ejemplo, en los casos en los que se desea realizar una web o una aplicación Android únicamente. El tiempo que ha llevado realizar esta aplicación es mayor del que hubiera llevado hacerlo para una sola de estas plataformas.

En el ámbito del desarrollo, creo que hubiera facilitado elegir un mismo lenguaje tanto en el lado del servidor como en el lado del cliente. En varias ocasiones se ha tenido que copiar una misma función en los dos lenguajes. Una en el servidor en PHP, para cuando la aplicación dispone de conexión, y otra en local, en el SW en JavaScript, para cuando esta no dispone de conexión. Como ejemplo la siguiente función que suma las horas de los operarios insertados mediante el formulario de partes de trabajo:

Figura 8-1: Función sumar horas en PHP

```php
public static function sumarHoras($arrayIds,$horas){
    $arrayIdsUnicos = array_unique($arrayIds);
    $arrayHoras=array();
    foreach ($arrayIdsUnicos as $key){
        $arrayHoras[$key]= 0;
    }
    for ($i=0; $i<count($arrayIds); $i++){
        if (!empty($arrayIdsUnicos) && in_array($arrayIds[$i],$arrayIdsUnicos)){
            for ($j=$i;$j<count($arrayIds);$j++){
                if ($arrayIds[$i]==$arrayIds[$j]){
                    $arrayHoras[$arrayIdsUnicos[0]]=$arrayHoras[$arrayIdsUnicos[0]]+$horas[$j];
                }
            }
            array_shift($arrayIdsUnicos);
        }
    }
    return $arrayHoras;
}
```

Figura 8-2: Función sumar horas en JavaScript

```javascript
function sumarHoras(arrayIds, horas){
    arrayIdsUnicos = new Set(arrayIds);
    var hashHoras = new Object();
    for (const [key, value] of arrayIdsUnicos.entries()) {
        hashHoras[parseInt(key)]=0;
    }
    for (i=0;i<arrayIds.length;i++){
        if (arrayIdsUnicos.size>0 && arrayIdsUnicos.has(arrayIds[i])){
            for (j=i;j<arrayIds.length;j++){
                if (arrayIds[i]==arrayIds[j]){
                    var it = arrayIdsUnicos.values();
                    var first = it.next();
                    hashHoras[first.value]=hashHoras[first.value]+horas[j];
                }
            }
            arrayIdsUnicos.delete(first.value);
        }
    }
    return hashHoras
}
```

Además, el hecho de elegir PHP, requisito impuesto por la empresa, implicó la utilización de Laravel como framework y la creación de las vistas dinámicas con su

sistema de plantillas Blade. Blade, evidentemente, emplea código PHP para la creación de estas vistas. Código que no puede interpretar el cliente. Por esta razón, toda la ejecución se hace en el lado del servidor y, posteriormente, se envía al cliente. En una situación normal en la que se dispone de conexión, esta es una forma adecuada de hacerlo, ya que, se evita que en el lado del cliente se pueda modificar código y en consecuencia explotar alguna vulnerabilidad. Sin embargo, en nuestro caso, era necesario que algunas de las vistas como la vista de *listas de partes* o la vista *firmar partes de trabajo* se pudiesen generar de manera local. Lo ideal probablemente para este caso sería crear las vistas con código JavaScript, código que se pudiera interpretar en el lado del cliente. Por tanto, hubiese sido interesante el estudio de alguna tecnología como Vue o Angular para la ejecución de estas vistas y la utilización de una tecnología como Node.js, basada en JavaScript, para el lado del servidor, en vez de Laravel.

Por último, como conclusiones personales, el desarrollo de este trabajo ha supuesto que me haya enfrentado al primer desarrollo real de una aplicación. Gracias a esto, he adquirido una cantidad considerable de conocimientos, primero de programación, ya que, empecé este trabajo con unos conocimientos muy básicos tanto de HTML como de JavaScript. Segundo de tecnologías como PWA y Laravel de los cuales no sabía absolutamente nada antes del comienzo de este proyecto. Y por último, de conocimientos profesionales. A lo largo de estos meses he tenido que hacer frente a todo este desconocimiento y considero, que he conseguido superarlas satisfactoriamente la mayoría de ocasiones.

8.1. Trabajo futuro

El trabajo futuro se puede dividir en dos partes. Por un lado, añadir nuevos apartados y funcionalidades a la aplicación y, por otro lado, el estudio de nuevas tecnologías para la agilización y mejora de la actual aplicación y futuros desarrollos.

En cuanto a la primera parte, el estudio de las tecnologías que permiten crear vistas dinámicas mediante JavaScript como Vue, Angular o React parece ser la mayor prioridad si se espera seguir creando apartados en los que queramos dar acceso aunque no dispongamos de conexión. La limitación de no poder ejecutar código PHP en el lado del cliente es demasiado grande para poder continuar por el camino actual. En segundo lugar, sería interesante estudiar un entorno de ejecución en JavaScript en el lado del servidor, como Node.js, para descubrir las ventajas e inconvenientes que podría darnos la utilización del mismo en lugar del actual.

Con respecto a la segunda, la aplicación tiene un amplio potencial de crecimiento y mejoras. Como ejemplo, ahora mismo ya es posible crear una avería y asignarla a un operario. Sin embargo, esta asignación es solamente a nivel de base de datos. En el futuro se puede considerar crear un sistema de notificaciones por el cual al crear una avería y asociarla a un operario, automáticamente, llegue una notificación al

operario al teléfono o a la web informándole de este suceso y, a la vez, pueda observar que se ha creado una fila nueva en la vista *listas de partes,* que contiene su nuevo parte de trabajo para la avería con la parte de datos generales rellenado. Esto agilizaría el proceso de creación de los partes de trabajo para las averías y mejoraría el sistema de llamadas actual.

Otra posible mejora podría ser la creación de un método para crear links, los cuales se puedan enviar al cliente y que de esta forma, este pueda firmar los partes de trabajo incluso cuando el operario no esté presente. Esta mejora, podría ser útil debido a que se realizan reparaciones de averías en horarios nocturnos, cuando las empresas están cerradas, para no interferir en la producción o el funcionamiento de algunas de ellas.

Además de estas mejoras, para poder considerar terminada la aplicación, en el futuro se tendrá que hacer una mejora en la encriptación de los datos de usuario para el inicio de sesión. Estos datos en estos momentos se están encriptando con una encriptación débil, como es md5, y en el futuro se tendrá que buscar un método de encriptación más robusto. Adicionalmente, también es necesario crear un sistema de permisos para la aplicación en vista de que no todos los usuarios deberán poder acceder a la sección de averías.

Referencias

[1] Instituto Nacional de Estadistica, «Encuesta sobre equipamiento y uso de tecnologías en los hogares,» 16 Octubre 2019. URL: https://www.ine.es/prensa/tich_2019.pdf. Consultado: 05 03 2020.

[2] Ministerio de industria, comercio y turismo, «Retrato de la PYME 2019,» 01 01 2019. URL: www.ipyme.org/es-ES/AreaEstadisticas/Paginas/InformesPYME.aspx. Cosultado: 05 03 2020.

[3] Ministerio de Industria, comercio y turismo, «Marco estratégico en políticia de PYME 2030,» 01 04 2019. URL: https://industria.gob.es/es-es/Servicios/MarcoEstrategicoPYME/Marco%20Estrat%C3%A9gico%20PYME.pdf. Consultado: 05 03 2020.

[4] A. Russell, «Progressive Web Apps: Escaping tabs without losing our soul,» 15 06 2015. URL: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/. Consultado: 19 03 2020.

[5] Laravel, «The PHP Framework for Web Artisans,» 21 06 2011. URL: https://laravel.com/. Consultado: 19 03 2020.

[6] Bootstrap, «The most popular HTML, CSS, and JS library,» 19 08 2011. URL: https://getbootstrap.com/. Consultado: 19 03 2020.

[7] Visual Studio Code, «Code editing redifined,» 29 04 2015. URL: https://code.visualstudio.com/. Consultado: 19 03 2020.

[8] MySQL Workbench, «MySQL Workbench,» URL: https://www.mysql.com/products/workbench/. Consultado: 19 03 2020.

[9] Git, «Local branching on the cheap,» 7 04 2005. URL: https://git-scm.com/. Consultado: 19 03 2020].

[10] Uber, «Engineering a high-performance web app for the global market,» 27 06 2017. URL: https://eng.uber.com/m-uber/. Consultado: 09 04 2020].

[11] Uber, «Building a more seamless web booking flow for uber,» 25 07 2019. URL: https://eng.uber.com/web-booking-flow/. Consultado: 09 04 2020.

[12] M. Gaunt, «Introducción a los service workers,» 12 02 2019. URL: https://developers.google.com/web/fundamentals/primers/service-workers?hl=es. Consultado: 19 03 2020.

[13] Fundación Mozilla, «Promise,» 23 03 2019. URL: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise. Consultado: 19 03 2020.

[14] World Wide Web Consortium, «Cache objects,» 24 02 2020. URL: https://w3c.github.io/ServiceWorker/#cache-objects. Consultado: 09 04 2020.

[15] World Wide Web Consortium, «Indexed database API 2.0,» 30 01 2018. URL: https://www.w3.org/TR/IndexedDB-2/. Consultado: 05 04 2020.

[16] PouchDB, «PouchDB, the JavaScript Database that Syncs!,» URL: https://pouchdb.com/. Consultado: 09 04 2020.

[17] World Wide Web Consortium, «Web app manifest,» 08 03 2020. URL: https://w3c.github.io/manifest/. Consultado: 19 03 2020.

[18] Laravel, «Blade Templates,» URL: https://laravel.com/docs/7.x/blade. Consultado: 15 04 2020.

[19] Bootstrap, «Display Property,» URL: https://getbootstrap.com/docs/4.0/utilities/display/. Consultado: 15 04 2020.

[20] Jquery, «Autocomplete Widget,» URL: https://api.jqueryui.com/autocomplete/. Consultado: 15 04 2020.

[21] Willow Systems Corp IT Team , «jSignature,» URL: https://github.com/willowsystems/jSignature. Consultado: 22 04 2020.

[22] jQuery, «jQuery wirte less, do more,» URL: https://jquery.com/. Consultado: 09 05 2020.

[23] Fontawesome, «The web's most popular icon set and toolkit.,» URL: https://fontawesome.com/. Consultado: 09 05 2020.

[24] Tempusdominus, «A robust date and time picker designed to integrate into your Bootstrap project,» URL: https://tempusdominus.github.io/bootstrap-4/Usage/. Consultado: 09 05 2020.