Gómez, Mauro Federico

Sistema de tasación online de vehículos para clientes de la organización

2021

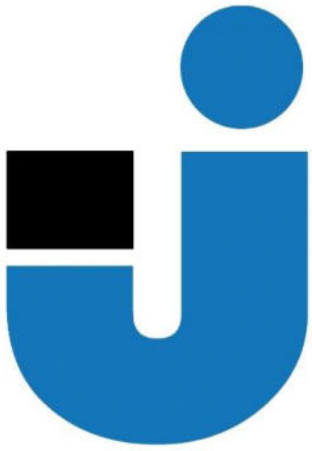
Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática



TRABAJO FINAL DE LA PRÁCTICA PROFESIONAL SUPERVISADA

Sistema de tasación online de vehículos para clientes de la organización

Estudiante:

Mauro Federico Gómez

Tutores:

Prof. Lía Lavigna

Ing. Rafael Villalba

Mg. Ing. Diego Encinas

Buenos Aires, 2021

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)
Sistema de tasación online de vehículos para clientes de la organización.
Informe Programa de Actividades

DATOS DEL ESTUDIANTE

Apellido y Nombres: Gómez, Mauro Federico

DNI: 40479907

Nº de Legajo: 20971

Correo electrónico: mauro.f.gmz@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 44

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16. (en caso de ser artículo 7 aclarar en cuál de las dos alternativas posibles se encuadra)

DOCENTE SUPERVISOR

Apellido y Nombres: Mg. Ing. Diego Encinas

Correo electrónico: dencinas@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO PARA LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Prof. Lía Lavigna

Correo electrónico: lialavigna@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Tecnom S.R.L

Dirección: Diagonal 74 Nro. 1463, B1900BZI La Plata, Buenos Aires

Teléfono: 011 5254-9236

Sector: Producto

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Ing. Rafael Villalba

Correo electrónico: rvillalba@tecnom.com.ar

FIRMA DEL COORDINADOR DE LA CARRERA

Índice

Índice de imágenes

1 Introducción

1.1 Resumen

En el presente trabajo se propone el desarrollo de una herramienta para los clientes de la empresa Tecnom S.R.L, con la que se busca facilitar la realización de tasaciones de automóviles, sumando así nuevos servicios a los ofrecidos por la compañía mencionada. Dicha organización se haya ubicada en la ciudad de La Plata y se encarga de brindar soluciones de desarrollo e implementación de software enfocadas principalmente en el mercado automotor. Los clientes de la compañía pertenecen a concesionarios de vehículos y obtienen como producto una aplicación CRM (Customer Relationship Management) para automatizar y ordenar el proceso de ventas, potenciando la gestión de las mismas.

Un CRM es una solución de gestión de las relaciones con los clientes que permite a las empresas generar más oportunidades de venta, con presupuestos actualizados en tiempo real y procesos de ventas optimizados. Los productos de Tecnom pueden integrarse con los sistemas de sus compradores, aportando así beneficios que agregan valor.

Por otro lado, Tecnom planifica brindar nuevos servicios, en conjunto con el desarrollo de aplicaciones CRM. En la presente Práctica Profesional Supervisada (PPS) se realizarán tareas centradas al desarrollo de estas nuevas utilidades, partiendo desde un objetivo propuesto desde la organización donde se realiza la misma. Se desarrollará una aplicación móvil que permita a las concesionarias de vehículos realizar tasaciones de los autos, tomar fotos de los mismos y de los documentos correspondientes, facilitando a los mecánicos el acceso a los datos necesarios de manera portable.

1.2 Objetivos

El objetivo general es desarrollar una aplicación móvil para que los mecánicos de los clientes de Tecnom puedan realizar tasaciones de los vehículos pertenecientes a su concesionaria. Dicha aplicación deberá realizarse con las tecnologías utilizadas por la empresa para llevar a cabo sus productos y su desarrollo se integrará a los servicios que se ofrecen.

A continuación, se enlistan los objetivos específicos que se pretenden alcanzar:

- Analizar las características que debe presentar el tipo de sistema a desarrollar.
- Realizar un análisis de requerimientos como primer paso para la arquitectura de la aplicación de tasación.
- Investigar sobre las tecnologías utilizadas en la empresa y cuáles son las empleadas en el desarrollo de este proyecto.
- Obtener una interfaz y experiencia amigable, es decir, que sea de fácil utilización para el usuario final, en este caso, los mecánicos pertenecientes a las concesionarias clientes de Tecnom.
- Efectuar la utilización de la aplicación de manera sencilla. La facilidad se deberá presentar tanto desde el punto de vista de la toma de fotos de los elementos (vehículos, documentación, entre otros) como desde el punto de vista del guardado y presentación de los datos en la pantalla; es decir, que no sean necesarias complejas operaciones para realizar las acciones mencionadas.
- Realizar pruebas funcionales de la aplicación desarrollada, analizar y validar los resultados.
- Integrar el desarrollo en los futuros productos y soluciones que brinda Tecnom.

1.3 Tareas a ejecutar

1- Analizar e investigar las tecnologías

Como tarea inicial, se analizará el tipo de aplicación propuesta, teniendo en cuenta: las características y utilidades que deberá tener, qué tipo de información es la que manejará y con cuáles tecnologías se desarrollará.

2- Realizar un análisis de requerimientos

En esta etapa se deberá especificar las características operacionales del software, por lo cual se indican la interfaz del software y las restricciones. Este análisis de requerimientos se realiza de manera conjunta con el grupo de trabajo, participando en la propuesta de los mismos en cuanto a los temas funcionales y técnicos.

3- Definir distintas vistas de la aplicación

Se realizarán distintas vistas que representan la interfaz de usuario, es decir, el medio que permita al usuario comunicarse con un dispositivo y las distintas interacciones con el mismo. Las mismas se presentan como prototipos de prueba sujeta a cambios.

4- Maquetación de la aplicación

Se trabajará en el front-end (parte de la aplicación que interactúa con los usuarios, es decir, la parte visible del lado del cliente) del proyecto. Para ello, se utilizarán las siguientes tecnologías:

Angular: Herramienta para la creación de vistas de aplicaciones web.

Angular Material: Son los componentes de diseño de interfaz de usuario para aplicaciones web, móviles y de escritorio hechas con Angular.

HTML: Es el lenguaje utilizado para la elaboración de páginas web.

5- Desarrollar el sistema

Una vez que se resuelve la interfaz de usuario, se desarrollará la lógica de la aplicación de tasación, la cual implica: realizar el diseño arquitectónico del sistema, donde se construirá el software siguiendo la metodología establecida por el grupo de trabajo; utilizar las tecnologías definidas para el desarrollo de cada uno de los componentes, como el front-end, la utilización de la cámara y la representación de los datos en la pantalla del usuario.

6- Realizar pruebas funcionales del desarrollo

Probar las distintas funciones del sistema de manera individual e identificar los posibles errores. Se analiza el funcionamiento en base a los resultados producidos y se verifica si son los datos esperados.

7- Efectuar test y evaluación de usabilidad en producción

Luego de haber realizado las pruebas funcionales, se ejecutará el sistema en producción. En esta etapa, se realizan las pruebas relacionadas con la utilización de la aplicación cuando esté finalizada, detectando fallos en la usabilidad, la

integración con el sistema del cliente y el análisis de la performance del software. También se evaluarán posibles futuras mejoras de la misma.

8- Llevar a cabo la integración del desarrollo en los futuros nuevos productos. Integrar el proyecto realizado en futuros nuevos productos de Tecnom. Se plantea que el resultado del desarrollo se encuentre dentro de las soluciones que ofrece la empresa a sus clientes. En esta fase, se decide cómo será el método en el que se proporciona este sistema a los usuarios finales.

1.4 Cronograma de trabajo

N° Tarea / Mes	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre
1	x	x				
2	x	x				
3		x	x			
4			x			
5			x	x	x	
6				x	x	x
7					x	x
8					x	x

2 Desarrollo

2.1 Análisis e investigación de las tecnologías

Las primeras tareas que se ejecutaron, previo al desarrollo del sistema propuesto, fueron las de analizar las características típicas y propias de los sistemas de tasación, junto al aprendizaje que conlleva y la inserción en el uso del conjunto de tecnologías que utiliza la empresa para llevar a cabo la realización de sus productos basados en software. En el primer caso, se investigó sobre las herramientas de tasación de cualquier clase existentes en la web o que forman parte de los servicios de otras empresas. Luego, se hizo hincapié en los sistemas específicos para cotizar los vehículos, de esta manera se consiguió información de referencia para la elaboración del software. En el segundo caso, se presentó en la reunión inicial con el equipo de trabajo, el stack[1] de tecnologías utilizado en Tecnom y cuáles serían empleadas a lo largo del desarrollo. Se dividió la construcción de la aplicación en distintas fases, las que pasaron de una a otra a través de una revisión y aprobación de la misma. En cada fase se usaron ciertas herramientas del stack principal.

2.1.1 Características del sistema de tasación definido

Como se mencionó anteriormente, los sistemas de tasación generales cuentan con diversas características en común. Estos poseen información acerca del elemento al que se le realiza la tasación, valor venal (importe monetario que obtendría el propietario de un bien material en un momento determinado), características del mismo, como ser: el estado en el que se encuentra, las diferentes cotizaciones que tuvo a lo largo del tiempo, evaluaciones registradas y, además, contiene fotos del objeto.

En el caso particular de los vehículos se presentan junto a las características nombradas en el párrafo anterior, lo siguiente: tipo de vehículo, tipo de combustible, kilometraje, estado del coche, fecha de primera matriculación. En cuanto a las fotos

[1] También llamado stack de soluciones o ecosistema de datos, los desarrolladores utilizan este término para asociar la lista de todos los servicios tecnológicos utilizados para construir y ejecutar una aplicación.

de los automóviles es necesario que sean de la parte delantera, trasera e interior de los mismos.

Una vez analizadas las distintas particularidades de estos tipos de sistemas se pensó, junto al equipo de trabajo, cuáles serían las mejoras alternativas para poder desarrollar una aplicación que contuviera toda esta información apuntada y accesible para los mecánicos de los clientes de Tecnom. En el análisis de la usabilidad de la aplicación también se pensó en la manera de interactuar del usuario final con la misma y qué datos tendría a disposición, ya fueran digitales, como los provistos por algún otro sistema desarrollado por Tecnom, o físicos, que se encontraban en documentos impresos. Se determinó que la mejor manera de conseguir una buena experiencia de usuario, manipulación y registro de los datos era realizando una aplicación móvil, consiguiendo de esta manera que los mecánicos (usuarios finales) obtuvieran en su bolsillo el alcance de toda la información necesaria para manejar la tasación. También se concluyó que no era necesario que se tuvieran que cargar datos propios de la tasación, ya que estos se encontraban en distintos documentos y otros sistemas. Por este motivo, fue conveniente añadir la posibilidad de tomar fotografías, tanto a los vehículos como a los documentos y también subirlos desde el dispositivo en caso de poseer los mismos en algún formato, como ser el PDF. Solo se cargarían datos que sirvieran para identificar las distintas tasaciones existentes en la aplicación generadas por el mismo usuario, tal como el número de patente o algún nombre personalizado que identificara el modelo del vehículo.

2.1.2 Definición de tareas y herramientas iniciales

Según lo planteado en la reunión inicial, se presentaron las tareas a realizar en las distintas fases del proyecto, las que se enmarcaron en distintos aspectos y funcionalidades. En la primera instancia, se puntualizó el maquetado de la aplicación y el diseño de la misma. Como se adelantó en la explicación de las tareas a ejecutar, se definieron distintas vistas para realizar la maqueta, enfocándose principalmente en el diseño de interfaz o UI Design (User Interface Design) y tomando en cuenta la experiencia de usuario o User Experience (UX). Este último término hace referencia al conjunto de factores y elementos relacionados con la

interacción de una persona con un entorno o dispositivo concreto, esto genera como resultado una percepción positiva o negativa de dicho servicio o producto. Esta etapa también contó con la supervisión de un diseñador UX/UI, quien estaba a cargo en el equipo de desarrollo de validar cada una de las vistas realizadas para el prototipo de la aplicación.

Se definieron las tareas antes mencionadas como las iniciales del proyecto, puesto que, en base a los primeros esquemas, se definió el contenido y las acciones que se pueden realizar con el sistema una vez finalizado y dado en producción. Esto generó una visión más clara del objetivo final para poder así definir los requerimientos funcionales del software. También se analizaron las tecnologías con las que cuenta Tecnom para permitir un desarrollo más eficiente en las etapas posteriores, por lo que se precisó obtener conocimientos acerca de las herramientas que utiliza la organización. Muchas de las soluciones que brinda la empresa mencionada consiste en la utilización de tecnologías de Amazon Web Services (AWS), esta tendrá su implementación en el proyecto en etapas avanzadas del mismo.

Las herramientas que se usaron para la generación de vistas fueron las siguientes:

- Framework para el frontend: Un framework es un entorno de trabajo que abarca un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular. La utilización de estas herramientas presenta muchas ventajas a la hora de programar:
    - Evitar escribir código repetitivo: La mayoría de los proyectos tienen partes comunes necesarias para el funcionamiento, por ejemplo, el acceso a la base de datos. A lo largo del presente informe se detallarán las partes del código que presentaron beneficios con esta característica.
    - Utilizar buenas prácticas: Los frameworks se basan en patrones de desarrollo, tal como el MVC (Modelo-Vista-Controlador) que ayudan a separar los datos y la lógica de negocio de la interfaz con el usuario. Esto genera un código más ordenado. Las tareas iniciales se van a enfocar en la interfaz de usuario como se explicó con anterioridad.

- Desarrollo más rápido: permite generar módulos de código integrado de manera más limpia y segura.

En la presente PPS se utilizará Angular para el desarrollo del front-end y en una fase más avanzada de la PPS se integrarán otros frameworks.

- Angular: Es un framework de diseño de aplicaciones y una plataforma de desarrollo para crear aplicaciones de una sola página eficientes y sofisticadas. La mayor parte del código estará escrito con el lenguaje de programación TypeScript que presenta las siguientes características:
  - Permite crear aplicaciones web progresivas (PWA): este concepto remite a utilizar las capacidades de la plataforma web moderna para ofrecer experiencias similares a las de una aplicación nativa[2]. La instalación es de alto rendimiento, sin conexión y sin pasos a seguir. Esta ventaja es de importancia destacar ya que, como se verá en la sección 2.2 del presente informe, se decidió que la aplicación móvil a desarrollar posea las características de las PWA. De esta forma se aprovecharán las herramientas que dispone la empresa y su integración con el proyecto.
  - Generación de Código: Angular convierte las plantillas en código altamente optimizado para las máquinas virtuales JavaScript actuales, brindando todos los beneficios del código escrito a mano con la productividad del framework.
  - División de código: las aplicaciones hechas con Angular se cargan rápidamente con el Component (subconjunto de directivas asociada a una plantilla) Router, que ofrece división automática de código para que los usuarios solo carguen el código necesario para representar la vista que solicitan.
  - Plantillas: se pueden crear rápidamente vistas de interfaz de usuario con una sintaxis de plantilla sencilla y potente. Esto fue necesario para crear distintas vistas individuales y presentarlas como un conjunto de los diferentes prototipos creados. Es necesario generar

[2] Una aplicación nativa es un programa de software que se desarrolló para su uso en una plataforma o dispositivo en particular.

estas plantillas en cada reunión para validar el diseño del sistema con el equipo y proponer mejoras.

- CLI Angular (Command Line Interface Angular): con esta herramienta de línea de comandos se puede construir rápidamente, agregar componentes y generar pruebas, para luego implementarlas instantáneamente. Esta característica agiliza la producción del código y también ayuda a encontrar errores en el mismo.

- HTML: Con este lenguaje se crea el esqueleto de la aplicación, proporcionando los principales elementos que componen cada una de las vistas. Es un lenguaje de marcación que sirve para definir el contenido de las páginas web y se compone en base a etiquetas, también llamadas marcas o tags, las cuales expresan partes de un documento al que refiere una página web, tales como: cabecera, cuerpo, párrafos, secciones, imágenes, etc. Además, esta herramienta es un prerrequisito para utilizar Angular, ya que cada plantilla de la aplicación es una sección de HTML. Esto se interpreta en que una plantilla hecha con HTML, utilizando el framework Angular, representa una vista o interfaz de usuario, en el navegador, como HTML normal, pero con mucha más funcionalidad. Al generar una aplicación en *Angular* con Angular CLI el archivo creado “app.component.html” es la plantilla predeterminada que contiene HTML como marcador de posición.
- SASS (Syntactically Awesome Style Sheets): es un lenguaje de hojas de estilos que se extiende de CSS (Cascading Style Sheets) y le agrega funcionalidades. CSS describe cómo deben ser renderizado los elementos HTML que componen la estructura de las vistas producidas. Por ende, Angular integra fácilmente las utilidades que brinda SASS, ya sea creando un proyecto nuevo con el framework mencionado e integrarlo desde el principio o migrando un proyecto con las correspondientes configuraciones establecidas. La plantilla a utilizar en el proyecto llamada Fuse Angular ya contiene integrado SASS.
- Fuse Angular: Plantilla de administración de *material design* con aplicaciones y páginas prediseñadas. Material design es una normativa de diseño enfocado en la visualización del sistema operativo Android. Tecnom

utiliza esta plantilla en el desarrollo de sus proyectos que utilizan *Angular*. Esta herramienta requiere de un nivel de entrada de conocimiento del framework mencionado para poder encontrar el camino dentro del código fuente. La versión de Fuse que se utiliza no es una plantilla de administración tradicional, sino que es una aplicación Angular escrita completamente con Typescript. La plantilla de Fuse proporcionada para empezar a realizar las vistas contenía sólo un proyecto limpio, sin los componentes que vienen por defecto, de esta forma ofreció libertad para elegir el diseño e integrar los componentes necesarios, por ejemplo, los elementos de *Angular Material*. En la misma plantilla utilizada para trabajar se configuraron e instalaron las dependencias de librerías necesarias para poder llevar a cabo el desarrollo del sistema, con las tecnologías de la empresa en etapas posteriores del proyecto.

- Angular Material: esta librería posee los componentes de *Material Design* para *Angular*. Fuse usa *Angular Material* como su biblioteca de interfaz de usuario principal. La integración de la plantilla con los componentes de esta librería es totalmente compatible, por lo que, instalar esta librería en los proyectos es totalmente sencillo, así como también emplear los componentes a utilizar, por ejemplo: botones, campos de formularios, contenedores de imágenes, menú superior o lateral, iconos, listas, barras de progreso, tablas, entre otros. Para la utilización de cada componente basta solo con importar el módulo con el código necesario hacia el proyecto, de esta manera queda listo para utilizar, pero es necesario modificar el código ya que se requieren acciones personalizadas por parte de estos componentes.

2.2 Análisis de requerimientos

Luego de haber sido presentado el *stack* de tecnologías, se procedió con la realización del análisis de requerimientos funcionales y no funcionales.

Los requerimientos funcionales son enunciados acerca de servicios que el sistema debe proveer, de cómo se debería comportar el sistema en distintos tipos

de escenarios y entornos junto con la manera de reaccionar ante entradas particulares. Dependen del tipo de software que se esté desarrollando y de los usuarios esperados del mismo, en este caso, un sistema para organizar y modificar tasaciones destinado a los mecánicos de los concesionarios clientes de Tecnom. Estos requerimientos se dividen, por un lado, en requerimientos del usuario y, por otro lado, en requerimientos del sistema. Los primeros hacen referencia a las funciones de la aplicación, sus entradas y salidas o sus excepciones. El segundo grupo varía desde requerimientos generales, que cubren lo que tiene que hacer el sistema, hasta requerimientos muy específicos, que reflejan maneras locales de trabajar o los sistemas existentes de una organización. Lo anteriormente expresado se vio reflejado en los distintos servicios y software del ecosistema de Tecnom, donde se fueron definiendo las tecnologías a utilizar, mencionadas en la sección anterior, en base a sistemas desarrollados aplicando dichas herramientas y metodologías.

Los requerimientos no funcionales son limitaciones sobre servicios o funciones que ofrece el sistema. Estos se suelen aplicar al sistema como un todo, más que a características o a servicios individuales del mismo. Además, pueden relacionarse con propiedades emergentes del sistema, como fiabilidad, tiempo de respuesta y uso de almacenamiento. Dicho en otras palabras, se mencionan todas las restricciones que presenta la aplicación en distintos aspectos de la misma, tales como: la representación de datos, el rendimiento, la seguridad, disponibilidad, etc. Estos requerimientos surgen a través de necesidades del usuario, debido a que pueden presentarse políticas de la organización o necesidad de interoperabilidad con otro software. Este último punto fue tomado en cuenta para analizar qué acción desencadena la utilización de la aplicación y qué datos pueden traerse de otros sistemas de Tecnom, así como la integración con los productos de dicha empresa. También se tuvieron en cuenta las restricciones que tiene una PWA y las aplicaciones móviles en general, ya que este fue el enfoque principal del tipo de sistema al que estaba destinado el desarrollo.

El análisis de estos tipos de requerimientos, en conjunto con el equipo de trabajo, obtuvo como resultados las siguientes características:

Requerimientos funcionales:

- El sistema debe funcionar sin presentar inconvenientes en los sistemas operativos Android e iOS, además de tener un buen rendimiento en navegadores web como Chrome o Firefox.
- ABM (Alta, Baja y Modificación) de tasaciones: Ingresar tasaciones nuevas, guardarlas y actualizarlas o eliminarlas. El mecánico tendrá a disposición dentro de la aplicación y a muy fácil acceso una lista de vehículos a los que le realizó la tasación. Esta lista deberá visualizar una foto de los vehículos, la fecha en la que se creó la tasación y su patente como atributo de identificación.
- Tendrá que tener características respectivas de las PWA, por lo que deberá poder acceder a la cámara del dispositivo del usuario o a la galería, en el caso que se desee subir una foto existente en lugar de tomar la foto en el momento de crear la tasación.
- Cuando se realiza una nueva tasación, el sistema debe utilizar un formulario para completar los datos principales del vehículo que está siendo cotizado, así como también subir fotos del mismo (parte interior y partes exteriores frontal y trasera).
- Se podrá cargar documentación que determine el estado del vehículo, tales como: permiso de circulación, ficha técnica, etc.
- Las tasaciones una vez creadas podrán ser actualizadas. Ya sea eliminando o agregando fotos a estas, cambiar datos ingresados y el valor del precio de la tasación.
- Se debe permitir la autenticación a través de Google, para poder ingresar al sistema o crear un usuario nuevo.

Requerimientos no funcionales:

- El tiempo de aprendizaje del sistema por parte de un usuario no deberá superar 1 hora.
- El tiempo de ejecución de la aplicación, cuando esta se inicia, no debe superar los 40 segundos.
- La experiencia de usuario debe ser altamente funcional, donde la interfaz gráfica sea agradable y sencilla. El texto y las indicaciones deben ayudar al

mecánico a poder manejar las distintas acciones que posee la aplicación, indicando en pocas palabras los pasos para realizar la tasación.

- El sistema debe proporcionar mensajes de error informativos y orientados al usuario final. Estos se muestran en caso de que la tasación no se haya realizado correctamente o que las fotos no se puedan subir al sistema.

Para cada uno de los requisitos funcionales definidos se tiene en cuenta el alcance del sistema y el tiempo de desarrollo del mismo. En cada requisito se tuvo en cuenta la aplicación de las metodologías de trabajo propuestas, el desarrollo por fase con entrega de un prototipo, así como también, las herramientas que iban a ser utilizadas.

2.2.1 ABM de tasaciones

El manejo de tasaciones debe realizarse a través de un CRUD (Create, Read, Update, Delete) donde se emplean las funciones básicas en bases de datos. Todas estas acciones se ejecutarán manualmente por el usuario y la aplicación se encargará de manejar estas peticiones a través de las funcionalidades, que fueron programadas para su correcto funcionamiento. El acceso a la información y la visualización de estos datos es interpretado por un framework orientado al frontend (en el desarrollo se utilizó *Angular*, tal como se ha mencionado en la sección anterior), donde se escribe el código aplicando las mejores prácticas que dicho framework aporta. Este código también se encarga de realizar las consultas a la base de datos utilizando herramientas de AWS, que se encargan de facilitar estas tareas y de configurar el backend de la aplicación. El backend está conformado por un servidor, el sistema mismo y la base de datos, donde se toman los datos, se procesa la información y se la envía al usuario. Además, se emplea la librería de *Angular Material* para crear elementos visuales de forma más rápida y agradable a la vista del usuario. La configuración del backend de la aplicación con AWS se explicará en detalle en la sección 2.5 Desarrollar el sistema.

2.2.2 Aplicación con características de PWA

Las Progressive Web Apps proponen varias ventajas, tanto para quien desarrolla el producto como para los usuarios finales del mismo. Algunos de los beneficios de los cuales se puede aprovechar para satisfacer a los usuarios son el enriquecimiento de la visibilidad, diseño, rendimiento y el compromiso.

Las aplicaciones son multiplataformas, ya que pueden ser ejecutadas sobre cualquier dispositivo y tienen la capacidad de adaptarse totalmente a la pantalla sobre la cual se visualizan. Este funcionamiento presenta dos ventajas: la primera es que la PWA está desarrollada con las tecnologías de la web, por lo que cuesta menos desarrollar una aplicación de este tipo que una app nativa ya que, si se quiere orientar hacia iOS y Android se tendría que realizar 2 desarrollos nativos distintos; la segunda ventaja se basa en la idea de que las Progressive Web Apps se ejecutan en una app universal, como ser el navegador web. Hoy en día todos los dispositivos conectados a internet poseen uno, es por este motivo que al desarrollar solamente una aplicación se racionalizan los gastos, disminuye la inversión en el producto y el tiempo de desarrollo, quitando el enfoque sobre la elección de la plataforma hacia la que se debería apuntar en caso de realizar una app nativa, junto a la necesidad de la incorporación de nuevas tecnologías.

Las PWA se distribuyen como una página web, es decir, a través de una URL (Uniform Resource Locator, dirección que se asigna a cada uno de los recursos disponibles en la red para que puedan ser localizados o identificados) llamada por un navegador. Esto presenta una gran ventaja para aplicaciones de esta índole porque no se tiene que pasar por la App Store o Google Play Store para distribuir la aplicación. Cuando se realiza el *deployment* (esto remite a todas las actividades que hacen que un sistema de software esté disponible para su uso) del producto terminado con los servicios que ofrece AWS, se puede obtener una URL con la dirección específica que se asigna a cada uno de los recursos disponibles en la red para que estos puedan ser localizados o identificados. A través de esta URL se accede a la aplicación encontrando una ventaja considerable en este punto, tal como el modo de distribución, que hace hincapié en la actualización de la PWA. De esta manera, todas las mejoras y cambios se aplican inmediatamente sin tener que

esperar o pasar por una revisión. Por otra parte, las aplicaciones nativas están hechas con código compilado, lo que significa que, para aprovechar las actualizaciones, los usuarios tienen que descargarla en sus dispositivos, que no es el caso en las PWA.

Uno de los requerimientos no funcionales definidos fue que la experiencia de usuario debe ser altamente funcional, puesto que una aplicación web progresiva es sobre todo una experiencia de usuario. Entonces, es teniendo en cuenta el usuario, que los desarrolladores se orientan hacia un panel de tecnologías resultantes de la web, de herramientas y de buenas prácticas para establecer a cambio nuevos estándares en la web tradicional. En la práctica, las PWA presentan un comportamiento en la navegación muy similar a las apps nativas cuyo objetivo principal es ofrecer interacciones fluidas al usuario final. Esto significa que, para los mecánicos, los elementos claves de la aplicación se visualizarán inmediatamente, ellos pueden acceder al contenido con tan solo un click y este se adaptará al tamaño de la pantalla del dispositivo. Este tipo de aplicaciones no necesita instalarse para ser iniciadas, por lo que ayuda al usuario a consultar el contenido de manera más sencilla, sin ninguna restricción de instalación propuesta por el sistema. En el caso de que los mecánicos quieran acceder aún más fácilmente a la app, sin tener que utilizar el navegador del dispositivo todo el tiempo, se puede instalar la PWA directamente en la pantalla de inicio del celular (este tipo de dispositivo móvil es el más adecuado para utilizar el sistema) de los usuarios de manera instantánea, sin necesidad de descargarla a través de las Tiendas. La instalación es ligera quitando cualquier restricción de memoria en los celulares al momento de instalarse la app. Cuando los mecánicos intentan acceder a una página particular del sistema de tasación, solo se cargan los elementos necesarios para visualizar a la página, lo que preserva el espacio de memoria de los dispositivos.

Una de las características mayores de una PWA es su capacidad de funcionar sin la presencia de una conexión a internet. Cuando el usuario accede por primera vez el área de almacenamiento temporal de un dispositivo (memoria caché), que guarda datos para acelerar la velocidad y el funcionamiento del móvil, almacena localmente una parte o la integridad del contenido disponible. De esta manera, en la próxima visita sin conexión a internet a la app, el mecánico podrá tener acceso al

contenido almacenado previamente y podrá navegar entre las distintas páginas que conforman el sistema. En el momento del desarrollo de la aplicación se elige qué elementos poner en caché, si todo el contenido o una parte. Como el contenido tiende a crecer y puede que se quiera agregar diversas funcionalidades en el futuro es preferible guardar en la memoria caché sólo algunas partes de la aplicación, tales como las pantallas de inicio y la lista de las tasaciones.

2.2.3 Funcionamiento y arquitectura de las PWA

Las Progressive Web Apps están desarrolladas a partir de lenguajes web del frontend, que permiten gestionar la interfaz de usuario de cada página. En el caso del presente desarrollo se utilizaron el framework *Angular*, desarrollado por Google, y los lenguajes de programación TypeScript y JavaScript. Estas tecnologías proporcionan a los programadores las herramientas para concebir experiencias de usuarios web comparables a las de las apps nativas.

Las PWA funcionan bajo la combinación de conceptos existentes: App Shell, Service Worker y el Manifest JSON.

Imagen 1. Elementos que componen las Progressive Web Apps.
Fuente: Recuperado de https://blog.goodbarber.com/es/docs/ebooks/Ebook%20PWA%20SP%20-%20PDF.pdf (2018)

La App Shell consiste en el esqueleto de la aplicación. Como se muestra en la imagen 01, se realiza un enfoque principal en la Experiencia de Usuario (UX). Contiene los elementos principales de la interfaz visual y los componentes estrictamente necesarios para el funcionamiento de la interfaz de usuario. Estos elementos están conservados localmente, mientras que los contenidos propios de cada elemento se recuperan dinámicamente a través de una API (application programming interface), que permite la comunicación entre dos sistemas o plataformas distintas para agregar diversas funcionalidades a la aplicación. En la sección 2.5 se analiza este funcionamiento y en la siguiente sección se detalla cada componente de la interfaz de usuario. La App Shell permite a partir de la segunda vista en la aplicación, una visualización más rápida de la app y también la carga de los recursos necesarios desde la memoria caché del dispositivo.

Un *Service Worker* o Trabajador de Servicio es la base técnica de las funcionalidades que distinguen las PWAs de las páginas web clásicas, ya que, aumentan el modelo de implementación web tradicional y permiten que las aplicaciones brinden una experiencia de usuario de mayor confiabilidad y rendimiento, funcionando a la par del código instalado de forma nativa. Este elemento se posiciona entre la app y el navegador y tiene la posibilidad de modificar el comportamiento de la PWA, en otras palabras, es un script (secuencia de comandos de un programa relativamente simple) que se ejecuta en el navegador web y administra el almacenamiento en caché de la app.

Los *Service Worker* funcionan como un proxy de red (servidor que actúa como intermediario entre la red e internet, que también filtra las peticiones de recursos realizadas por un cliente a otro servidor). También, interceptan todas las solicitudes HTTP (Hypertext Transfer Protocol. Protocolo de comunicación que permite las transferencias de información en la red) salientes realizadas por la app y pueden elegir cómo responderlas. Por ejemplo, pueden consultar un caché local y entregar una respuesta en el caché si alguna se encuentra disponible. Los *Service Worker* presentan un almacenamiento en caché totalmente programable y no depende de los encabezados de almacenamiento en caché especificados por el servidor.

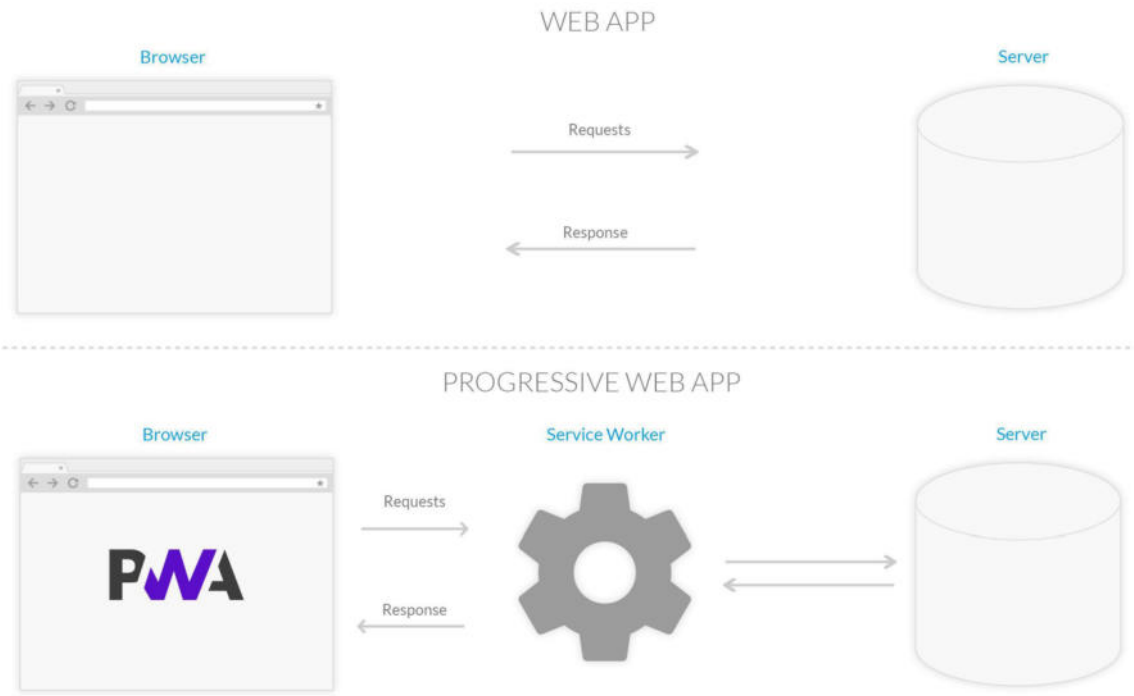


Imagen 2. Funcionamiento del Service Worker en las Progressive Web Apps
Fuente: Recuperado de *https://blog.goodbarber.com/es/docs/ebooks/Ebook%20PWA%20SP%20-%20PDF.pdf (2018)*

En la imagen 02 se puede observar la funcionalidad de un *Service Worker*. Este intercepta las consultas (request) hechas al servidor (Server) mediante el navegador (Browser) del dispositivo y será capaz de reenviar una respuesta (Response) transmitiendo el recurso recibido por el servidor o proporcionando una versión local del mismo. A diferencia de otros scripts que componen una aplicación, el *Service Worker,* se conserva incluso después de que el usuario cierre la pestaña. La próxima vez que el navegador cargue la app, el Service Worker se carga primero y puede interceptar cada solicitud de recursos para cargar la aplicación. Si el *Service Worker* se diseña para hacerlo, puede satisfacer completamente la carga total del sistema, sin la necesidad de la red. Este último punto se conseguirá en el desarrollo del sistema de tasación si se cumplen con cada una de las condiciones que se mencionan en la sección 2.7 (test y evaluación del desarrollo) y estas hacen referencia a la performance del software desarrollado.

Como se mencionó anteriormente, el sistema de tasación a desarrollar utiliza el framework *Angular* para realizar el frontend, por lo que esta herramienta puede

beneficiarse de las ventajas de los *Service Workers*. Los Trabajadores de Servicio en *Angular* están diseñados para optimizar la experiencia del usuario final al usar una aplicación a través de una conexión de red lenta o poco confiable, al tiempo que minimiza los riesgos de entregar contenido desactualizado. El comportamiento sigue el siguiente objetivo de diseño:

- El almacenamiento en caché de una aplicación es como instalar una aplicación nativa. La app se almacena en caché como una unidad y todos los archivos se actualizan juntos.
- Una aplicación en ejecución continúa ejecutándose con la misma versión de todos los archivos. No comienza a recibir repentinamente archivos en caché de una versión más reciente, que probablemente sean incompatibles.
- Cuando los usuarios actualizan la aplicación, observan la última versión completamente almacenada en caché. Las pestañas nuevas cargan el último código almacenado en caché.
- Las actualizaciones ocurren en segundo plano, relativamente rápido después de que se publican los cambios. La versión anterior de la aplicación se sirve hasta que se instala y está lista una actualización.
- El *Service Worker* conserva el ancho de banda cuando es posible. Los recursos se descargan solo si se encuentran cambios en estos.

Para admitir estos comportamientos, *el Service Worker* de *Angular* carga un archivo *manifest* desde el servidor.

Un *Manifest JSON* es un fichero de origen *JSON* que describe los recursos para almacenar en caché. Este fichero descriptivo permite dar un resultado más nativo a la aplicación, con una visualización en pantalla completa, íconos identificables o también la posibilidad de modificar la orientación de la pantalla. Cuando se implementó una actualización de la aplicación, el contenido del *manifest* cambia e informa al *Service Worker* que se debe descargar y almacenar en caché una nueva versión de la app. Este *manifest* se genera a partir de un archivo de configuración generado por CLI (Command Line Interface es el intérprete de línea de comandos de Angular que permite iniciar proyectos y la creación del esqueleto de todo tipo de elementos necesarios para el desarrollo de aplicaciones) llamado *ngsw-config.json*.

2.2.4 Acceso a los recursos del dispositivo

Una de las principales funcionalidades mencionadas que describen a las PWA son el acceso a los recursos del dispositivo donde la app está instalada. El desarrollo del sistema de tasación utiliza la cámara de fotos del teléfono móvil para realizar alguna de sus funciones más importantes, tal como la de tomar fotos de los vehículos y de los documentos necesarios.

La única manera de manipular imágenes en los dispositivos móviles era utilizando un elemento HTML, que permitía lanzar aplicaciones proporcionando imágenes obtenidas con la cámara de fotos. Las aplicaciones web o Web App no presentaban otras alternativas a esta manipulación de imágenes.

Actualmente, el “Media Capture API” permite a las Web Apps tener acceso directamente a los flujos de audio y de video del dispositivo, siendo estos legibles y manipulables desde la aplicación. De esta manera, es posible tomar una foto desde la app sin quitar el navegador o sin tener que abrir la cámara del dispositivo de forma manual, sacar la foto y luego subirla al sistema. En la imagen 03 se presenta una tabla con los distintos navegadores que son compatibles con la tecnología mencionada en diferentes sistemas operativos. Se observa que el navegador Chrome, que viene instalado por defecto en la mayoría de los teléfonos móviles, presenta un alto grado de compatibilidad con los sistemas operativos más comunes

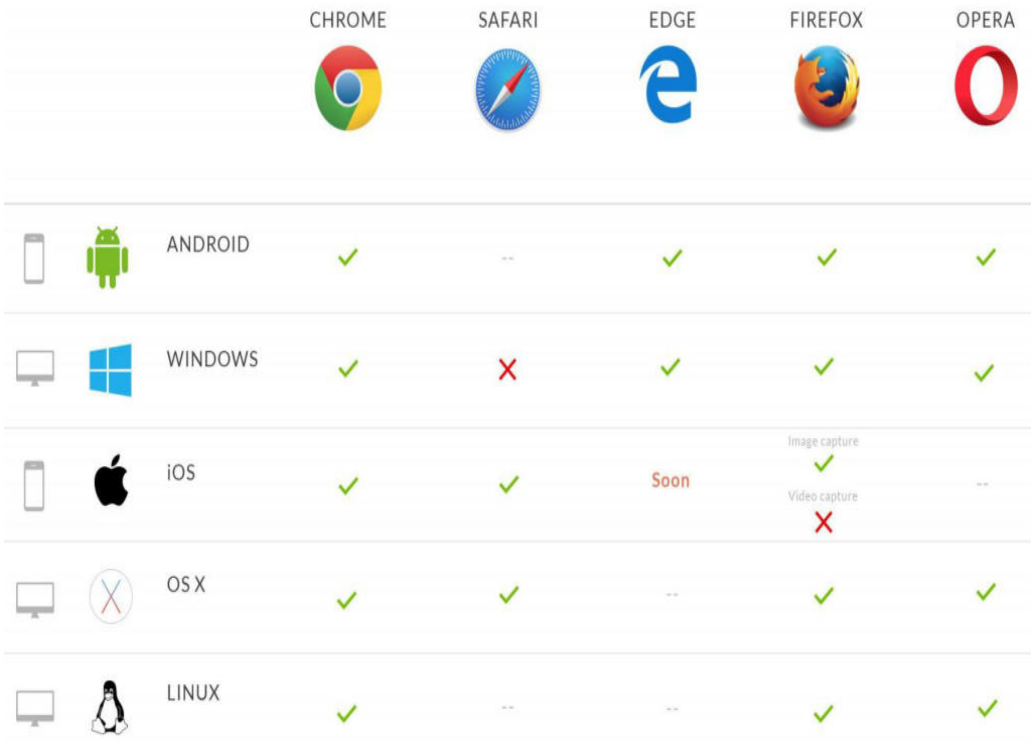


Imagen 3. Compatibilidad de los navegadores web con el acceso a la cámara de fotos y videos
Fuente: Recuperado de *https://blog.goodbarber.com/es/docs/ebooks/Ebook%20PWA%20SP%20-%20PDF.pdf (2018)*

2.3 Definición de las distintas vistas de la aplicación

En la metodología de trabajo adoptada, se debe presentar un prototipo de la aplicación una vez finalizada cada etapa. Luego de definir cada uno de los requerimientos funcionales y no funcionales, se trabajó en el diseño de un *mockup* en conjunto con un diseñador UX, que forma parte del equipo de desarrollo y quien se encarga de validar el diseño una vez terminado para empezar con la *maquetación*. Un *mockup* es un modelo o un prototipo que se utiliza para exhibir o probar un diseño. Gracias al empleo de este modelado de sistema, un diseñador puede analizar y mostrar cómo avanza su trabajo. De este modo, si es necesario realizar cambios y dichas modificaciones se establecen antes de la presentación de la versión final del producto.

En ocasiones, el *mockup* es una maqueta realizada a escala, sin ningún tipo de funcionalidad; en otros, el *mockup* presenta herramientas y opciones que permiten su uso y su evaluación en funcionamiento. La idea principal es presentar

la primera versión del diseño en el entorno apropiado, de este modo, quien encargó el trabajo tiene la posibilidad de apreciar cómo será el resultado antes de que se proceda al cierre de las tareas del diseñador.

Un *mockup*, en desarrollo de sistemas, debe permitirle al grupo de trabajo obtener una idea bastante clara del resultado final del producto, por lo tanto, requiere un enfoque de diseño web, además de experiencia de diseño de interfaces para otros softwares con funcionalidades similares o ya realizados. Uno de los puntos fundamentales a la hora de crear un *mockup* son las herramientas, donde eligiendo las adecuadas se puede conseguir mejores resultados en menor tiempo. En muchos productos realizados en la empresa Tecnom se utiliza *Angular Material* y una plantilla de diseño basada en *Material Design,* llamada Fuse Angular.

2.3.1 Fuse Angular

Fuse Angular se trata de una plantilla de administración potente y profesional para aplicaciones web, CRM (Customer Relationship Management), paneles de administración, entre otras. En Tecnom se suele utilizar esta plantilla para desarrollar productos de CRM para concesionarias, cuyo enfoque propone gestionar la interacción de una empresa con sus clientes actuales y potenciales, donde se recopilan datos de una variedad de canales de comunicación diferentes. Entonces, es necesario que se presenten los datos en una interfaz agradable para manejar la experiencia del usuario en estos sistemas. Es este último punto donde Fuse Angular ofrece sus utilidades para desarrollar interfaces de usuario más fácilmente. De esta manera, se procedió a realizar el *mockup* del sistema de tasación con esta plantilla para agilizar el desarrollo del mismo.

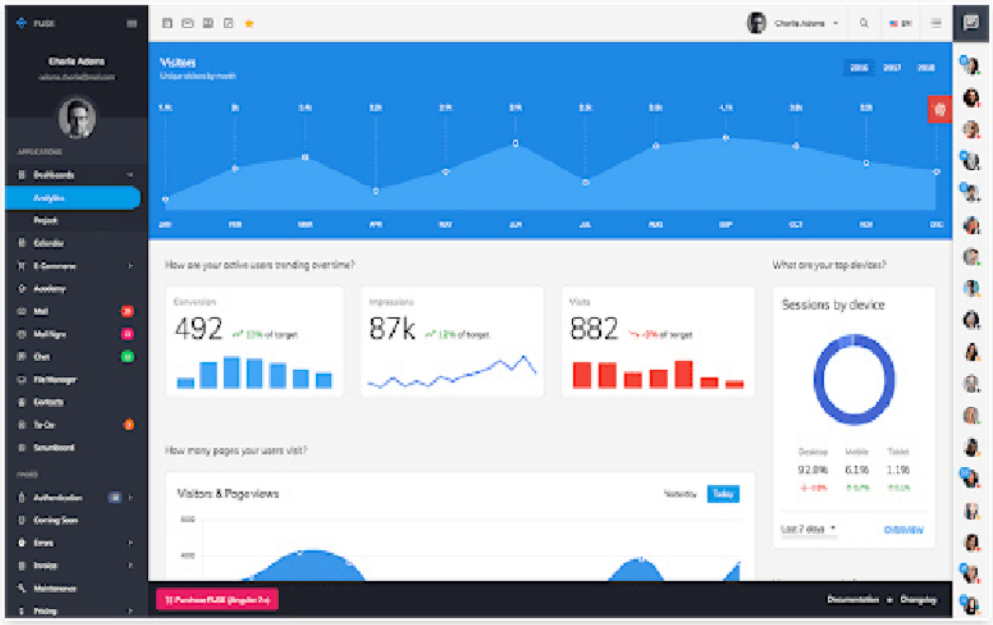


Imagen 4. Visión general de las utilidades de Fuse Angular
Fuente: Recuperado de http://fusetheme.com/admin-templates/angular/ (2020)

Fuse es un gran motor de arranque, que contiene aplicaciones y páginas incluidas, lo cual facilita el punto de partida de cada nuevo proyecto. El código fuente está perfectamente estructurado, por lo que la curva de aprendizaje de *Angular* se disminuye y permite obtener conocimientos de buenas prácticas en el enfoque del diseño. Las aplicaciones que trae esta plantilla incluidas no se utilizarán, en su lugar, Tecnom proporciona una plantilla de Fuse en blanco, a la que se pueden aplicar las funcionalidades de la herramienta que sean necesarias. Se integra fácilmente con *Angular Material*, además, de estar construido con el framework de frontend *Angular*, también se pueden usar todos los elementos de la librería mencionada de manera sencilla y así conseguir la elaboración de distintas vistas y *mockups* en menor cantidad de tiempo. En la imagen 04 se presenta una visión general de Fuse con todas sus utilidades instaladas. Se eliminan todos los componentes que no son necesarios para el desarrollo del sistema de tasación, tal como se muestra en la imagen 05 y se proporciona una plantilla en blanco para trabajar en dicho sistema.

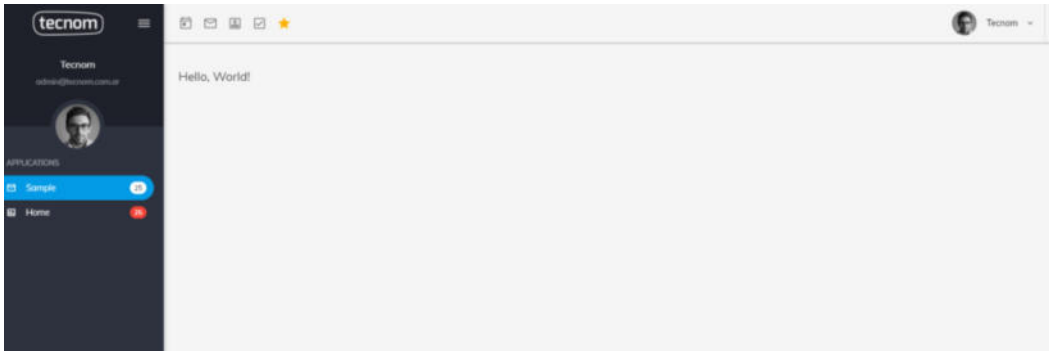


Imagen 5. Plantilla de Fuse Angular proporcionada por Tecnom.
Fuente: Elaboración propia, basada en la práctica.

La plantilla también presenta otras características, que varían en las versiones de Fuse Angular y que contienen diferentes estructuras y casos de uso. En el sistema de tasación se utilizó la versión estable 8.1.2, que maneja estas características generales:

- Está basada en SCSS (hoja de estilo que contiene al lenguaje Sass), es decir, que todos los estilos están escritos en Sass, permitiendo una fácil personalización.
- El manejo del color con esta plantilla es más práctico y se tiene una gestión avanzada de este, con la posibilidad de añadir temas con cierta cantidad de colores que se muestran en diferentes elementos de la aplicación.
- Fácil desarrollo e implementación a través del lenguaje de programación TypeScript y Angular CLI.
- Contiene una estructura flexible, con la posibilidad de cambiarla según las rutas, que hacen referencia a las distintas páginas de la aplicación y el flujo que se tiene para navegar entre ellas dentro del sistema.

2.3.2 Mockup del sistema

Una vez que se tienen las herramientas mencionadas anteriormente instaladas se realiza el *mockup* dentro de la plantilla de Fuse Angular. En una página en blanco se detallan los principales componentes de la aplicación, así como el flujo que hay entre el redireccionamiento de páginas cuando sucede un evento dentro del sistema (se presiona algún botón, se cargan fotos, errores en pantalla).

En primer lugar, se diseña la vista del *home* de la aplicación. Esto hace referencia a la página de inicio, siendo la entrada principal, la recepción y la primera impresión que van a tener los mecánicos cuando ingresen a la página por primera vez y cada vez que ejecute la app. La estructura se diseñó de tal manera que sea simple de usar, por lo que tiene que incluir la menor cantidad de opciones para recorrer todas las funcionalidades de la aplicación. Debe contener botones y secciones con un tamaño de letra que no afecte la visión de la persona ya que, por ejemplo, si el tamaño de la fuente es demasiado pequeño puede imposibilitar la lectura y si es grande llegaría a causar incomodidades en el uso de la app; además de que puede afectar a los demás componentes de la interfaz gráfica si es que la pantalla del dispositivo es pequeña.

El *home* consiste en dos botones y un título con el nombre de la aplicación. Este se visualiza luego de que el usuario se registre o inicie sesión en el sistema, esto se explicará en detalle en la sección 2.5 donde se hace hincapié al desarrollo del sistema, ya que elaborar un *login* (herramienta de autenticación e ingreso al sistema) corresponde a la utilización de herramientas de AWS, sumado al desarrollo de funcionalidades establecidas y configuraciones realizadas para manejar las dependencias de estas tecnologías. Los botones del *home* corresponden a la redirección a las páginas “lista de tasaciones” y “crear tasación”, donde la primera muestra las tasaciones presentes en el sistema con algunos detalles visibles que ayudan a la identificación de alguna tasación realizada y la segunda indica los pasos para crear una nueva tasación con todas las acciones necesarias.

En la imagen 06 se observa, en primer lugar, la estructura del *home* de la aplicación, donde se encuentran dos opciones las cuales permiten listar las tasaciones o crear una nueva tasación. En segundo lugar, se encuentra presente una estructura simple de la lista de tasaciones a la que se puede acceder presionando la opción “Tasaciones” en el home de la app. La lista presenta cada tasación realizada por el mecánico, donde se visualiza una imagen del automóvil al que se le realizó la tasación, en el desarrollo se incluirá una de las fotos que se le tomó al vehículo. Los detalles se muestran en el centro de cada ítem del listado y se decidió que la información que contenga esta sección sea mínima y suficiente para no dificultar la presentación de la misma. La información que se muestra

consiste en un título descriptivo de la tasación que ayude a identificarla (nombre colocado de forma manual, modelo del vehículo o su patente). Por último, se agrega un botón “Ver”, donde al presionarlo se redirecciona a otra página que permite obtener más características y detalles de la tasación correspondiente, por ejemplo: la cotización, la patente del vehículo, las distintas fotos de este y sus respectivos documentos. Es posible también editar los valores de la cotización o eliminar y agregar fotos en la página que visualiza los detalles específicos.

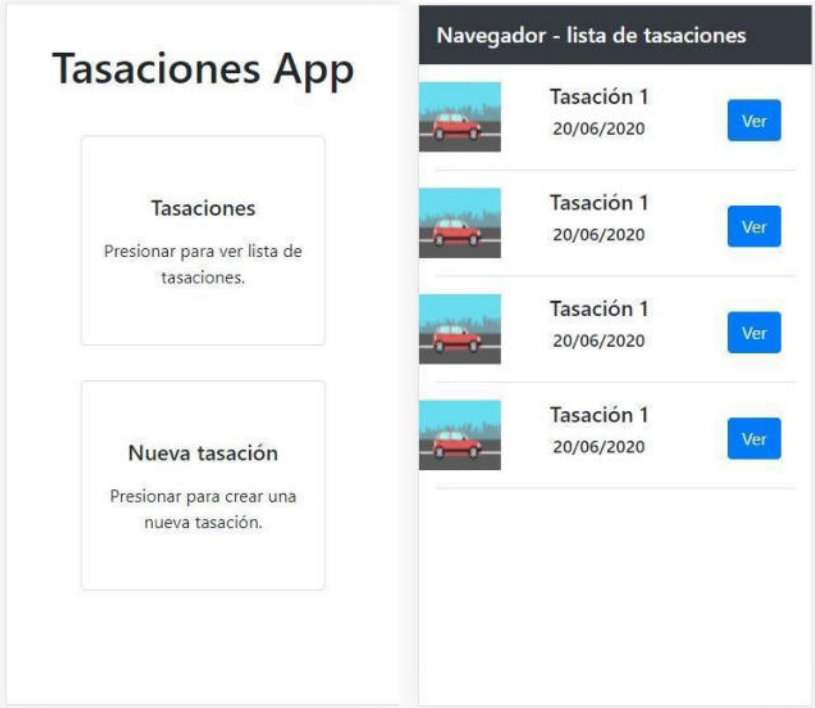


Imagen 6. Diseño de la vista del home principal (izquierda) y de la lista de tasaciones (derecha).
Fuente: Elaboración propia, basada en la práctica.

En la imagen 07 se puede observar la estructura de la página “Nueva tasación”. Para ingresar en esta página hay que presionar el botón “Nueva Tasación” dentro del *home* principal, una vez dentro de la sección se procede a crear la tasación. En primer lugar, se tendrá un formulario dentro de un contenedor (elemento de HTML que contiene a otros elementos) que se completará manualmente con distintos valores en los respectivos campos: la patente o modelo del automóvil (a modo de identificador de la tasación en la lista de tasaciones) y el valor de la cotización que estimó el mecánico al tasar el vehículo. Luego, se

presiona el botón “siguiente” y se abrirá otro contenedor para permitirle al mecánico tomar fotos del automóvil y, de la misma forma, también se redirecciona dentro de la misma página a un nuevo contenedor para tomar fotos de los documentos o subirlos desde el dispositivo móvil en el que se está manejando la aplicación. En el último contenedor habrá un botón “fin” que indica que no hay que seguir completando datos para crear la tasación. Debajo de toda esta sección se encuentra un botón “Crear Tasación” que carga esta nueva tasación dentro de la base de datos del sistema. Solo es posible presionar este elemento si hay algún dato escrito dentro del formulario del primer contenedor, de lo contrario, dicho botón se encontrará bloqueado.

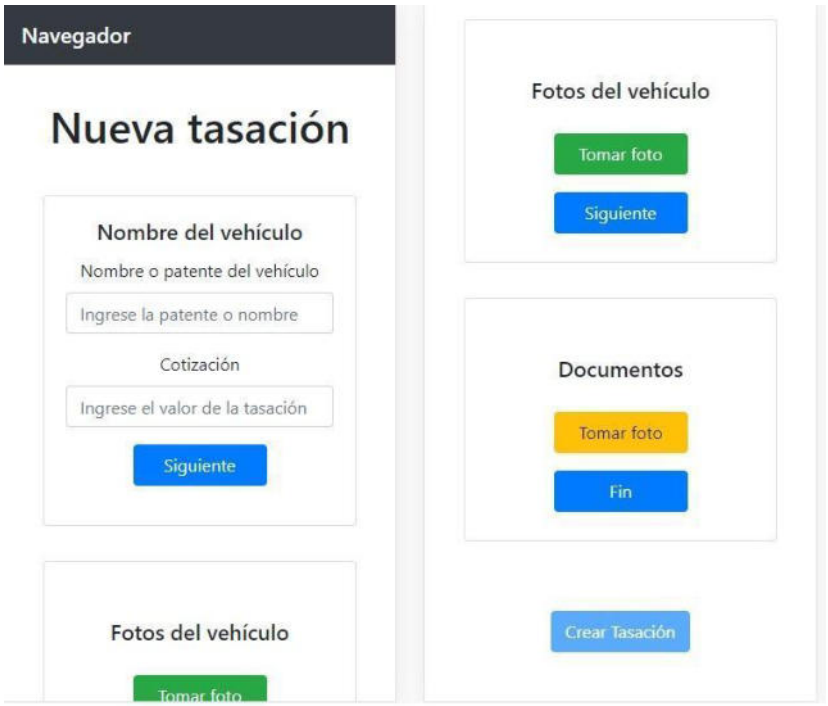


Imagen 7. Diseño de la vista “Nueva Tasación” con las indicaciones para ingresar datos.
Fuente: Elaboración propia, basada en la práctica.

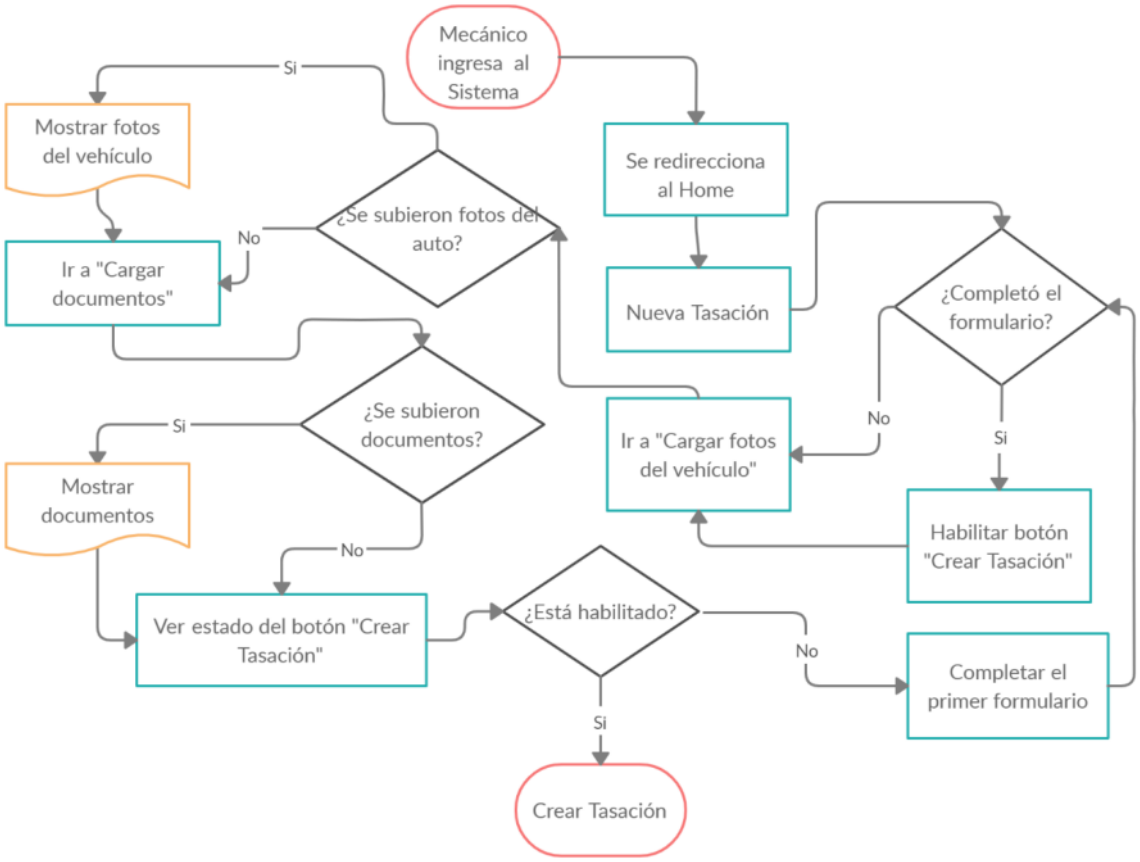


Imagen 8. Diagrama de flujo de la creación de una tasación en la aplicación por el mecánico.
Fuente: Elaboración propia, basada en la práctica.

Para poder comprender mejor cómo se realiza el proceso de creación de una nueva tasación se elaboró un diagrama de flujo con la representación gráfica de los procesos, tal como se muestra en la imagen 08. Las terminales se indican en el gráfico con “Mecánico ingresa al sistema” y “Crear tasación” que señalan el inicio y fin de las tareas dentro de una operación.

En la imagen 08 se puede observar que el proceso clave para que se pueda crear la tasación es la habilitación del botón “Crear Tasación”. Este elemento se encuentra bloqueado hasta que se escriban datos en el formulario del primer contenedor (nombre de la tasación, modelo del vehículo o patente y su cotización). Es necesario que la aplicación pida estos datos al mecánico, ya que pueden generarse problemas para identificar la tasación de algún vehículo en particular al momento de querer acceder a sus detalles. Si no se consigue habilitar el botón, por

no proporcionar los datos requeridos, es posible volver al *home* principal y en el caso de haber subido fotos, estas no se guardarán. Si el botón se encuentra habilitado, al presionarlo se observa en pantalla un mensaje que confirma la creación de la tasación y el sistema redirecciona automáticamente la página actual donde está el usuario al *home* principal.

Todas las vistas descritas del *mockup* están solamente realizadas con HTML y CSS, pero implementar su estructura utilizando Fuse Angular se vuelve más sencillo por las ventajas mencionadas en la descripción de esta herramienta. En la siguiente sección se muestra cómo cada una de las vistas desarrolladas se implementan con Fuse Angular y los componentes de *Angular Material*, para generar un diseño final de las vistas de la aplicación.

2.4 Maquetación de la aplicación

Cuando se terminó el *mockup* del sistema, con la realización de todas las estructuras de las vistas, este se valida con el equipo de trabajo, particularmente con el diseñador UX. Si el *mockup* cumple correctamente con las funcionalidades definidas, este es aceptado para continuar con la *maquetación* de la aplicación. Se lleva a cabo la implementación de las vistas en la plantilla en blanco de Fuse Angular proporcionada por Tecnom.

Cuando se abre la plantilla en un IDE (Entorno de Desarrollo Integrado) tal como *Visual Studio*, se puede observar la estructura general del proyecto como indican los archivos y carpetas en la figura “a” de la imagen 09. Las carpetas presentadas son las siguientes:

- *e2e*: esta carpeta, denominada “*end to end*”, engloba una serie de archivos cuya función es la realización de test automáticos, como si un usuario interactuara con la app. Se ejecuta con el comando “*ng e2e*”.
- *node_modules*: es la carpeta que contiene todas las dependencias del proyecto.
- *src*: es el directorio en donde se trabajan con los módulos. Además, es el más importante, ya que contiene todo el código. En la estructura de la figura “b” de la imagen 09 se presentan los siguientes archivos y carpetas:

- *@fuse*: contiene todos los componentes y funcionalidades de Fuse Angular.
- Carpeta *app*: se ubica toda la implementación de los componentes principales, junto a su template HTML y archivos de estilos CSS.
- Carpeta *assets*: contiene todos los assets (hojas de estilos, archivo JavaScript e imágenes) y archivos adicionales para hacer que el proyecto funcione.
- Carpeta *environments*: aquí se encuentran las configuraciones y variables de entorno para poner el proyecto tanto en desarrollo como en producción.
- Carpeta *graphql*: se generan los ficheros cuyo tipo de archivo pertenece a GraphQL (lenguaje de consulta y manipulación de datos para APIs)
- Archivo *index.html*: es el archivo de la página principal del proyecto.
- Archivo *main.ts*: es el archivo TypeScript inicial del proyecto donde se encuentran todas las configuraciones globales del proyecto.

- Archivo con extensión de tipo *gitignore*: son las carpetas o archivos que debe ignorar *git* cuando se suban cambios al repositorio.
- Archivo *angular.json*: contiene la configuración de *Angular*. Incluye las rutas de la aplicación y versiones de las dependencias.
- Archivo *package.json*: es la configuración de la aplicación. Contiene el nombre de la app, las dependencias necesarias para su correcta ejecución y configuración de versiones de estas.
- Archivo *tslint.json*: se utiliza para que el código sea sostenible y se mantenga.

La carpeta *src* contiene otra carpeta denominada *app* y esta a su vez contiene los archivos que se visualizan en la figura “c” de la imagen 09. Estos corresponden a los componentes generados en *Angular*, por lo tanto, cada aplicación realizada con el framework mencionado tiene al menos un componente. Por defecto, existe un componente raíz que conecta una jerarquía de componentes con el modelo de objeto de documento de página (DOM). Cada componente define una clase que contiene datos y lógica de la aplicación y está asociado con una plantilla HTML que

define una vista que se muestra en un entorno particular, como el navegador en el dispositivo móvil del mecánico.

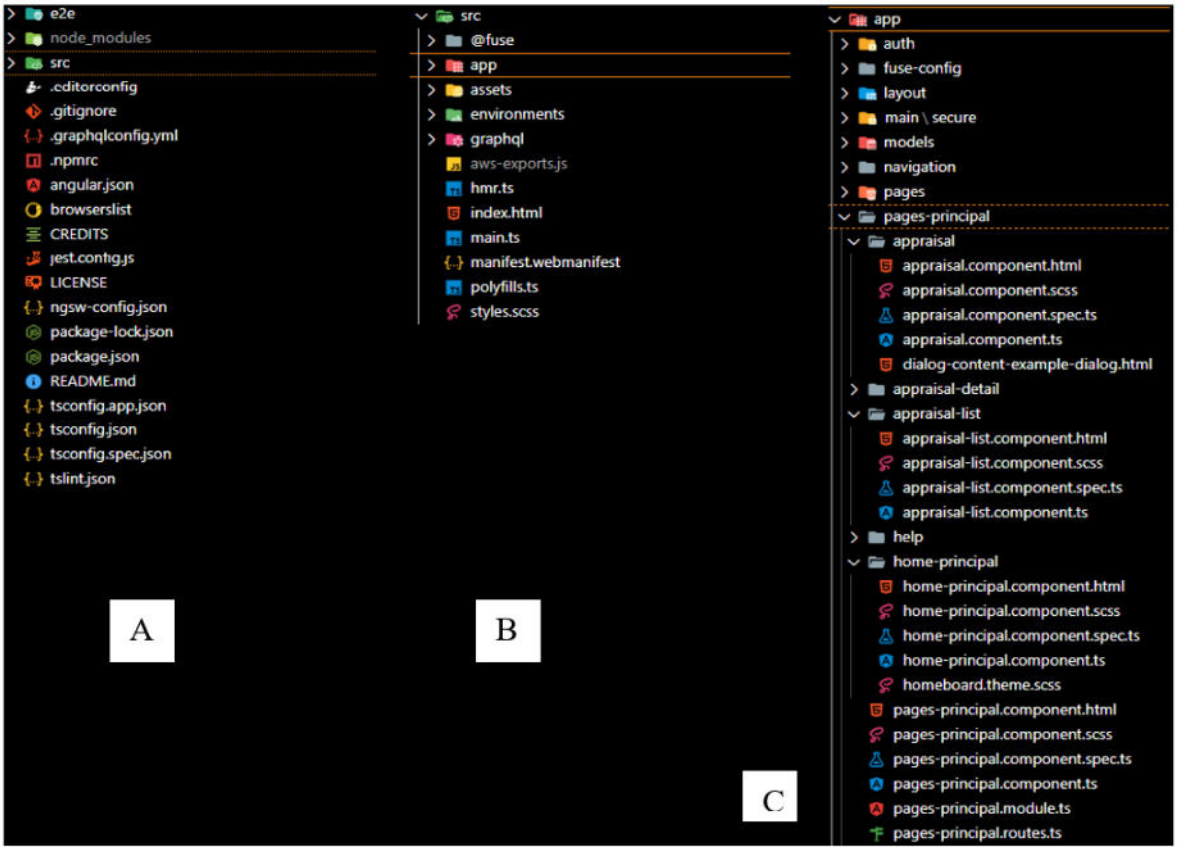


Imagen 9. A) Estructura general del código fuente del sistema. B) Contenido de la carpeta src. C) Contenido de la carpeta app dentro de src.
Fuente: Elaboración propia, basada en la práctica.

Los módulos de *Angular* (también denominado *NgModules*) declaran un contexto de compilación para un conjunto de componentes que está dedicado a un dominio de aplicación, un flujo de trabajo o un conjunto de capacidades estrechamente relacionado. Un *NgModule* puede asociar componentes con código relacionado, como servicios, para formar unidades funcionales. Los *NgModules* pueden importar la funcionalidad de otros *NgModules* y permitir que su propia funcionalidad sea exportada y utilizada por otros módulos. Por ejemplo, en el frontend para el sistema de tasación es indispensable tratar con rutas para redireccionar las páginas cuando el mecánico esté navegando en la app y para resolver este punto, se puede usar el servicio de enrutador importando el *RouterNgModule*. Este servicio importado se encarga de agregar directivas y

proveedores para la navegación en la aplicación entre las vistas definidas de la misma.

Se generaron los siguientes módulos y carpetas:

- Módulo *auth*: se generó un módulo de autenticación del usuario, que estará vacío en el proceso de maquetación ya que será trabajado con herramientas de backend y servicios de AWS.
- Módulo *layout*: aquí se encuentran todas las posiciones para las posibles vistas que ofrece Fuse Angular. Solo se utilizará la aplicación en modo vertical.
- Carpeta *models*: contiene clases simples que se usan para almacenar datos y funciones. Una clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Una de las buenas prácticas de *Angular* consiste en crear la carpeta *models* para aislar las estructuras de datos que se está utilizando del código del componente.
- Módulo *pages-principal*: Contiene todos los componentes que se utilizarán en el diseño de las vistas de cada página del sistema de tasación. Se observa que cada componente dispone de cuatro archivos de los siguientes tipos: html, scss, ts, spect.ts. El primer tipo de archivo contiene la página con todos los elementos de HTML; el segundo, los estilos SASS; el tercero corresponde al código que se ejecutará cuando se cargue la página implementando las funcionalidades dinámicas que se hayan programado y, el último tipo de archivo, se refiere a las pruebas (*tests*) que se realizaron en conjunto con el código total. Cada componente es creado ejecutando el siguiente comando (*component-name* se reemplaza por el nombre del componente a crear en la consola de comandos del IDE o del sistema operativo):

  ```
  ng generate component [component-name]
  ```

En el módulo *pages-principal* se crearon 3 componentes:

- *Home-principal*: En este componente se añade todo el código relacionado a la vista del *home* principal, tal como: la estructura HTML desarrollada en el *mockup*; los estilos SASS, que importan otros estilos provenientes de Fuse;

por último, contendrá el código TypeScript con los métodos para navegar en la página junto a un archivo de *test*.

- *Appraisal*: contiene archivos similares al *home-principal*, solo que el código TypeScript contiene los métodos que interactúan con la API y el servidor del sistema, para crear una tasación.
- *Appraisla.list*: contiene archivos similares al *home-principal*, con la diferencia de que el código TypeScript contiene los métodos necesarios para listar las tasaciones de la base de datos del sistema haciendo consultas a la API.

El módulo *pages-principal* tiene ciertas características de los componentes, como el tipo de archivos que lo componen, pero a su vez contiene todos los *imports* del sistema (librerías de *Angular Material*, modelos, servicios de *Angular*, entre otros) en un archivo denominado *pages-principal.module.ts*. También, contiene el enrutamiento necesario para la redirección de páginas a través de la configuración de las rutas en el archivo *pages-principal.routes.ts*.

Se trabaja, en primer lugar, con la vista del *home* principal, debido a que es la primera página que el usuario encuentra al abrir la aplicación. Además, se facilita el trabajo con las funciones del framework *Angular* al definir las rutas de paginación desde el *home*. Este se refiere a la forma de trabajar del framework mencionado con la definición de módulos y la relación entre los componentes, que contiene cada módulo para compartir información entre ellos o definir rutas para acceder a los mismos. En segundo lugar, se trabaja con la vista definida para crear tasaciones, la cual implementa características de las PWA. Por último, se integran las páginas de la lista de tasaciones y la vista con información general correspondiente a los detalles de la tasación seleccionada en la lista mencionada. También, se aplican los elementos de *Angular Material* necesarios para mejorar el diseño de la app. Es preciso indicar que en la maquetación se resuelve el frontend del sistema y se implementan algunas funcionalidades que trabajan con datos de prueba. Cuando se termina el maquetado y la validación con el equipo de trabajo es confirmada, se procede a añadir código perteneciente al backend y una base de datos para que se trabaje con datos reales que proporcionan los usuarios. En esta etapa de *maquetación*, el diseño está sujeto a cambios en el aspecto visual.

El *home* principal está compuesto por los siguientes elementos y componentes de *Angular Material*:

- *mat-icon*: es un componente de *Angular Material* que facilita el uso de íconos basados en vectores o SVG (Scalable Vector Graphics, es un formato de gráficos vectoriales bidimensionales, tanto estáticos como animados) en la aplicación.
- *fxLayout*: es una directiva que se utiliza para definir el diseño de los elementos HTML, es decir, decide el flujo de elementos secundarios dentro de un contenedor *flexbox* (modelo de diseño que organiza automáticamente los elementos dentro de un contenedor, dependiendo del tamaño de la pantalla o del dispositivo).
- *fxLayoutAlign*: es una directiva que define la alineación del elemento secundario dentro del contenedor principal de *flexbox*.
- *fxFlex*: esta directiva especifica el cambio de tamaño del elemento anfitrión dentro de un flujo de contenedor de caja flexible como *flexbox*.
- *fusePerfectScrollbar*: es una directiva *Angular* para la biblioteca *Perfect Scrollbar* (barra de desplazamiento), que acepta un valor booleano (*true* o *false*) opcional que puede controlar la barra de desplazamiento. Si se proporciona como valor *false*, *Perfect Scrollbar* se destruirá o no se iniciará.

El archivo HTML de la lista de tasaciones está compuesto por los siguientes elementos:

- *mat-spinner*: es una etiqueta de *Angular Material* que indica el progreso de una acción a través de un gráfico circular.
- *mat-card*: es un contenedor de distintos elementos, como texto, imágenes y botones. En la aplicación el contenido de cada *mat-card* pertenece a una tasación del mecánico, donde el usuario ve una foto del vehículo relacionado a la misma junto a un texto que muestra información resumida de la tasación.

En la imagen 10 se observa el resultado de la *maquetación* del *home* principal y la lista de tasaciones. Fuse Angular proporciona elementos que ayudan al usuario a interactuar con la aplicación de manera más sencilla. Por ejemplo, aporta distintos tipos de barras de navegación, separadores de contenido y colores definidos para que se tenga una paleta de colores que mantengan un tema, es decir, colorizar las

distintas vistas de manera que no haya contraste entre ellas y que ayude a relacionar elementos con funcionalidades parecidas, como los botones. Se añadió también un menú lateral (*Sidenav*) para poder acceder a todas las páginas sin importar en qué página el usuario se encuentre presente. El *Sidenav* cuenta con acceso al *home* principal, lista de tasaciones, creación de tasación y finalización de sesión en la aplicación, para salir de esta.

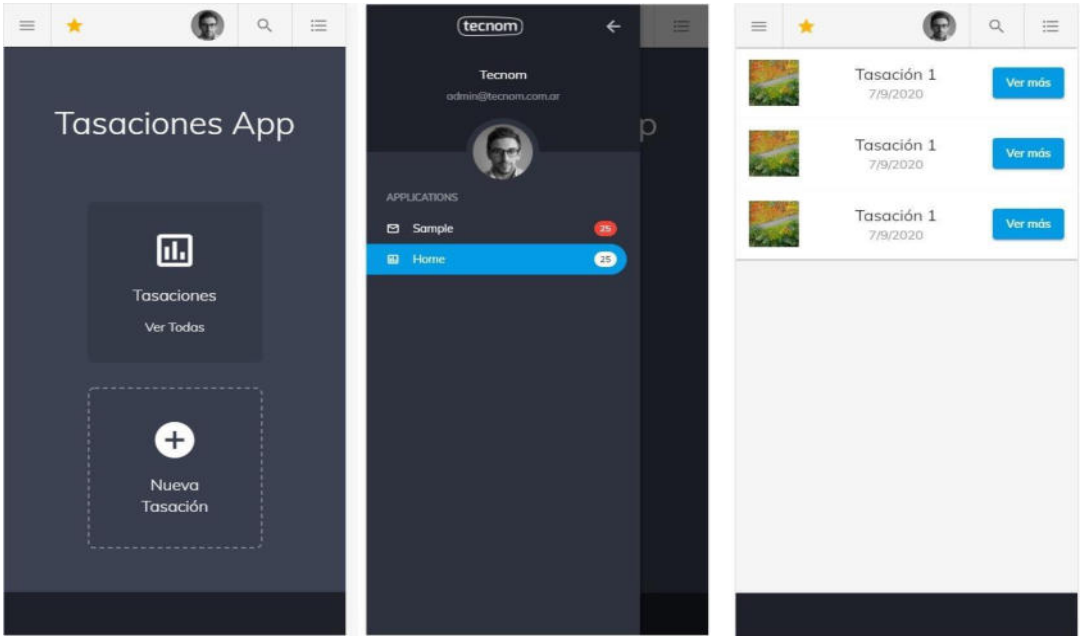


Imagen 10. Implementación de los componentes de Angular Material y Fuse Angular en el mockup realizado en la definición de vistas de la app. Se presenta en primer lugar, el home principal, en segundo lugar, el menú lateral y, por último, la lista de tasaciones.
Fuente: Elaboración propia, basada en la práctica.

La página donde se crea la tasación contiene elementos de *Angular Material* e implementa algunas etiquetas con funcionalidades, que permiten incorporar características de las PWA, tal como el acceso a la cámara de fotos. En esta página se encuentran presentes los siguientes elementos:

- *mat-accordion*: directiva de *Angular Material* que se emplea para diseñar un contenedor tipo *accordion* (contenido plegable) expandible de elementos HTML.
- *mat-expansion-panel*: proporciona una vista con el resumen de detalles y es expandible. Se utiliza para separar el formulario, donde se completan los

datos generales del vehículo, de las secciones en las cuales se cargan las fotos a la tasación que se encuentra en proceso de creación. La información de estas secciones se puede expandir y contraer, por lo que ayuda a que no se muestren todos los pasos de la creación de la tasación para tener una visión simplificada de los elementos en pantalla y completar los datos necesarios haciendo click en la sección correspondiente.

- *form*: es una etiqueta HTML que se emplea para recopilar la entrada de datos por parte del usuario a través de campos de formulario. Todos los datos entrantes se envían a un servidor al pulsar un botón del tipo *submit*.
- *input*: este elemento sirve para capturar desde el sistema archivos locales de un dispositivo para que el usuario pueda subir un archivo a un servidor *online*. El *input* se ha dotado de posibilidades que se adaptan a las necesidades del usuario final en un dispositivo móvil permitiendo capturar una foto con la cámara. Esto último es posible gracias al atributo *capture,* el cual permite activar de forma directa la cámara del dispositivo especificando el tipo de archivo que se necesita capturar. Se setea el valor "*camera*", ya que, el archivo proviene de un elemento desde la cámara de fotos. En la siguiente línea de código HTML se pueden ver los valores nombrados seteados en el sistema:

```
<input type="file" accept="image/*" capture="camera"
class="inputget">
```

El código anterior pertenece al archivo *appraisal.html,* que se encuentra dentro del componente *Appraisal* del código fuente del sistema.

- *mat-grid-list*: es una vista de lista bidimensional, que organiza las celdas en un diseño basado en cuadrículas. Con este elemento se pueden mostrar las fotos subidas del vehículo y los documentos en una cuadrícula de tamaño 3x3.
- *mat-dialog-content*: es un servicio que posibilita abrir cuadros de diálogo *modales* (requieren que el usuario responda antes de continuar con el programa) con estilos y animaciones de *Material Design*. En la aplicación se lo emplea para tener una vista más aumentada de las fotos subidas al pulsar

sobre ellas. Además, se puede eliminar la foto seleccionada con este cuadro de diálogo.

- *snackBar*: este componente informa a los usuarios sobre un proceso que la aplicación ha realizado o realizará. Aparecen, temporalmente, en la parte inferior de la pantalla. Por lo tanto, no interrumpen la experiencia del usuario y no requiere que el mecánico pulse un botón para desaparecer (esto es opcional). En el sistema de tasación aparece cuando se crea una tasación exitosamente.

En la imagen 11 se observa el resultado de la *maquetación* de la sección *nueva tasación* y allí se puede observar una vista que se obtiene al pulsar el botón "Nueva Tasación" en la sección del *home* principal. Lo primero que se muestra es un cuadro con un formulario que pide detalles al usuario sobre el auto al que se le realiza la tasación, el mismo está diseñado utilizando las directivas de *Angular Material mat-accordion* y *mat-expansion-panel* que se mencionaron anteriormente. La primera directiva separó la creación de la tasación en tres pasos distinguibles, siendo posible alternar los mismos en cualquier momento del proceso de creación, estos pasos son los siguientes: primer paso, "Datos del vehículo", en el cual se deberá completar con el nombre que se quiera identificar a la tasación en la base de datos para luego reconocerla en la lista de tasaciones que se indicó en la figura 10; "Fotos del vehículo". El segundo paso que contiene un botón para tomar fotos del vehículo otorgando una vista previa de las imágenes que serán subidas y, por último, "Documentos", donde se puede subir fotos de la documentación a través de un botón idéntico al diseño del paso anterior, pero con otro color. La segunda directiva incluye todos los elementos del espacio contenedor de cada uno de los pasos siendo posible ocultarlos o visualizarlos, según convenga cuando se va avanzando en el proceso de creación de la tasación. Además, facilita la visualización de la información en la pantalla, ya que si se quiere volver a un paso anterior o saltar alguno el usuario no tendrá que navegar mucho por la pantalla.

La vista de la imagen 11, perteneciente a la creación de la tasación, incluye la utilización de la directiva *form* para los campos del formulario donde el usuario tendrá que ingresar los datos. La información colocada en esos campos es otorgada como valor de los atributos definidos en el código, que define la lógica del

componente, y el mismo se encuentra dentro de la carpeta *appraisal* en la dirección *src/app/pages-principal* del proyecto en el archivo *appraisal.component.ts*. Cuando se termina de completar el formulario y se pulsa el botón “Agregar Tasación”, se utiliza la directiva *ngModel* para asignar finalmente los valores ingresados en los campos a los atributos del archivo *appraisal.component.ts*. *NgModel* crea una instancia de control del formulario (FormControl) a partir de un modelo de dominio y se la vincula a un elemento de control de formulario.

Cuando se sube una fotografía tomada desde el celular, se la muestra debajo del botón para obtener las imágenes dentro del contenedor del *mat-expansion-panel* correspondiente (fotos del vehículo o de los documentos). La directiva utilizada para presentar todas las fotos al usuario es *mat-grid-list*, la cual divide el contenedor en una determinada cantidad de columnas y se establece una medida vertical que se ajuste al tamaño de cada columna, es decir, que tengan lados iguales. Para que el diseño no afecte a la experiencia de usuario ni a la interfaz se decidió que haya tres columnas adaptables a cualquier tipo de dispositivo, cumpliendo las características del diseño *responsive*.

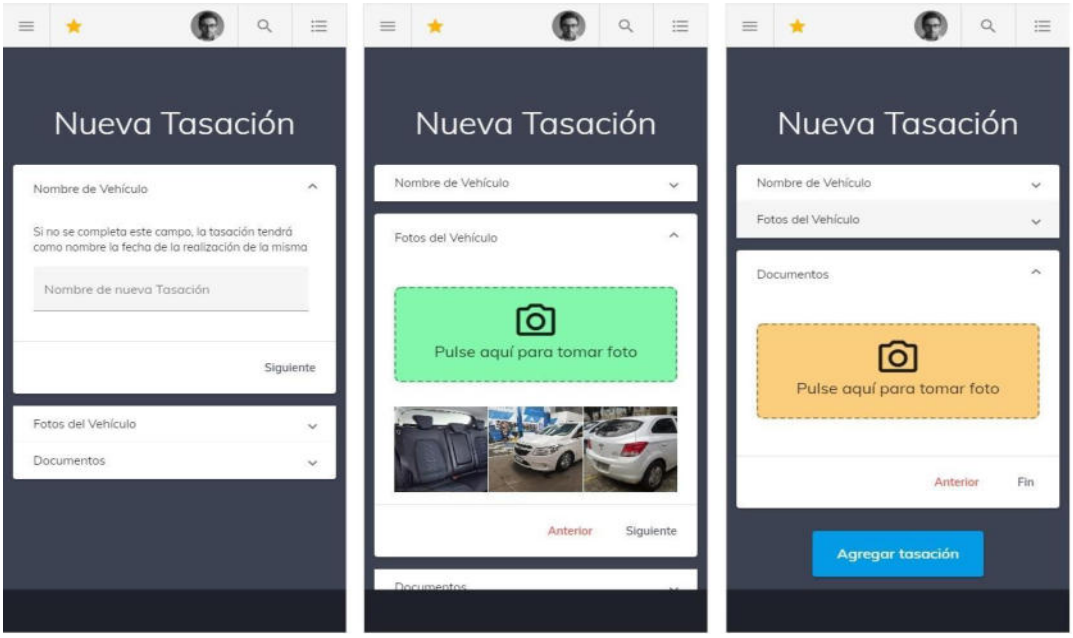


Imagen 11. Implementación de los componentes de Angular Material y Fuse Angular en el mockup realizado en la definición de vistas de la app. Se presenta la vista de la creación de tasaciones y todos los pasos para concretar la misma.
Fuente: Elaboración propia, basada en la práctica.

2.5 Servicios de AWS

En el desarrollo del backend del sistema se implementaron las funcionalidades que se describieron en el análisis de requerimientos y también las propias del frontend. La empresa utiliza tecnologías de Amazon Web Services (AWS), que es el proveedor líder mundial en tecnología Cloud, cuya plataforma ofrece más de 200 servicios integrales de centros de datos a nivel global. Tecnom utiliza dicha tecnología para reducir los costos, aumentar su agilidad e innovar de forma más rápida, en los distintos proyectos que realiza hace uso de varios de los servicios de AWS en conjunto para el desarrollo de los sistemas.

AWS cuenta con una gran cantidad de servicios y de características incluidas en ellos que ofrecen desde tecnologías de infraestructura, como cómputo, almacenamiento y bases de datos hasta tecnologías emergentes, como aprendizaje automático e inteligencia artificial, lagos de datos y análisis e Internet de las Cosas[3] (IoT). Esto trae beneficios cuando se quieren llevar aplicaciones a la nube siendo realizadas de forma más rápida, fácil y rentable. A su vez, existe una amplia variedad de bases de datos que están diseñadas especialmente para diferentes tipos de aplicaciones, de modo que se puede elegir la herramienta adecuada para el trabajo a fin de obtener el mejor costo y rendimiento. Los CRM y demás soluciones que ofrece Tecnom para mejorar las ventas en los concesionarios son desarrollados con la tecnología de AWS, ya sea para facilitar la implementación de los sistemas como para el diseño de la base de datos y el despliegue o puesta en producción de las aplicaciones.

En la presente PPS tuvo lugar el aporte de los beneficios que obtuvo la utilización de las herramientas que brinda AWS con los siguientes servicios: IAM (Identity and Access Management), Cognito, AWS Amplify, DynamoDB, AWS AppSync y S3 (Amazon Simple Storage Service). Al momento de comenzar a utilizar cada servicio se tiene que crear una cuenta en Amazon para poder obtener acceso a las herramientas, pero en lugar de crear una nueva cuenta se generó un usuario a partir de una cuenta existente, de un empleado de la empresa, a través de IAM.

[3] La internet de las cosas (en inglés, Internet of Things, abreviado IoT) es un concepto que se refiere a una interconexión digital de objetos cotidianos con internet.

Para incorporar el inicio de sesión en la aplicación se utilizó Cognito, que permitió el control de acceso a través de un proveedor de identidad social como Google o Amazon. Con AWS Amplify se facilitó la configuración del backend del sistema que se desarrolló en la presente PPS, permitiendo conectar el frontend a través de las bibliotecas de Amplify. Incorporando el uso de DynamoDB se facilitaron las tareas relacionadas a la creación de la base de datos, ya que se encuentra completamente administrada, es duradera y cuenta con copia de seguridad, restauración y seguridad integrada. La API del sistema se desarrolló con GraphQL, que es un lenguaje de consulta y manipulación de datos para APIs, cuyo entorno de ejecución sirve para realizar consultas con datos existentes y se facilita el desarrollo de la API con el servicio AWS AppSync. Los objetos como las fotos, que son subidas a la base de datos de la aplicación por parte de usuario, necesitan ser almacenados en alguna ubicación que permita almacenar un determinado volumen de datos, entonces por este motivo se agregó el servicio de S3 que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento.

2.5.1 IAM

AWS IAM es un servicio web que ayuda a controlar de forma segura el acceso a los recursos de AWS y es conveniente utilizarlo para controlar quién está autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos. Cuando se crea una cuenta de AWS por primera vez, se comienza con una única identidad de inicio de sesión que tiene acceso completo a todos los servicios y recursos de la cuenta. Esta identidad recibe el nombre de AWS de la cuenta de *usuario raíz* y se obtiene acceso a ella iniciando sesión con la dirección de correo electrónico y la contraseña que se utilizó para crear la cuenta.

IAM presenta las siguientes características:

- Acceso compartido a la cuenta de AWS: se puede conceder permiso a otras personas para administrar y utilizar los recursos de la cuenta de AWS sin tener que compartir la contraseña o clave de acceso. Este punto fue de suma importancia para el desarrollo de la práctica, ya que a través de la cuenta de AWS de un desarrollador del miembro del equipo se obtuvo acceso a los servicios que se incorporaron en la aplicación y todos los detalles de los

mismos. La cuenta propietaria o *usuario raíz* creó un usuario IAM que sería utilizado para el acceso, este usuario estaba compuesto por las siguientes credenciales: nombre de usuario, contraseña, clave de acceso (Key Access ID), clave de acceso secreta y un enlace de inicio de sesión de la consola.

- Permisos detallados: es posible conceder diferentes permisos a distintas personas para una cantidad determinada de recursos. Por ejemplo, se puede permitir que algunos usuarios tengan acceso de solo lectura a algunos *buckets* (contenedor de objetos en la nube como las fotografías en el caso de la aplicación desarrollada) de S3 para garantizar que los objetos no sean alterados u obtenidos por otros medios a través del enlace del objeto.
- Integración con muchos servicios de AWS: las herramientas que brinda AWS utilizadas en la práctica pueden integrarse fácilmente con IAM.
- Consistencia final: IAM ofrece una alta disponibilidad, ya que replica datos entre varios servidores ubicados en centros de datos de Amazon de todo el mundo. Si una solicitud para cambiar algunos datos se realiza correctamente, el cambio se confirma y se almacena de forma segura. Sin embargo, el cambio se debe replicar en IAM, lo que puede llevar algún tiempo. Estos cambios incluyen la creación o actualización de usuarios, grupos, roles o políticas.

En la imagen 12 se puede observar el panel de administración y control de IAM, donde el usuario raíz puede generar nuevos usuarios IAM y asignarles roles. La vista pertenece a un usuario de IAM (mgomez como usuario creado para realizar las tareas en la presente PPS), que es una entidad que se crea en AWS. El usuario de IAM representa a la persona o servicio que utiliza el usuario de IAM para interactuar con AWS. El uso principal de los usuarios de IAM consiste en proporcionar a las personas la posibilidad de iniciar sesión en la Consola de administración de AWS para realizar tareas interactivas y solicitudes programáticas a los servicios de AWS mediante o la CLI (Command Line Interface). Al crear un usuario de IAM se le concede los permisos convirtiéndole en miembro de un grupo que tiene asociadas políticas de permisos adecuados o directamente adjuntándole políticas a dicho usuario. Se observa también en la imagen 12 que en el menú de la izquierda del panel se encuentra la administración del acceso, donde pueden

configurarse los Roles de IAM, siendo estos una identidad con políticas de permisos que determinan lo que la identidad puede hacer y lo que no en AWS. Sin embargo, un rol no tiene ninguna credencial (contraseña o clave de acceso). Por ello, en lugar de asociarse exclusivamente a una persona, la intención es que cualquier usuario pueda asumir un rol para disponer temporalmente de diferentes permisos para una tarea específica.

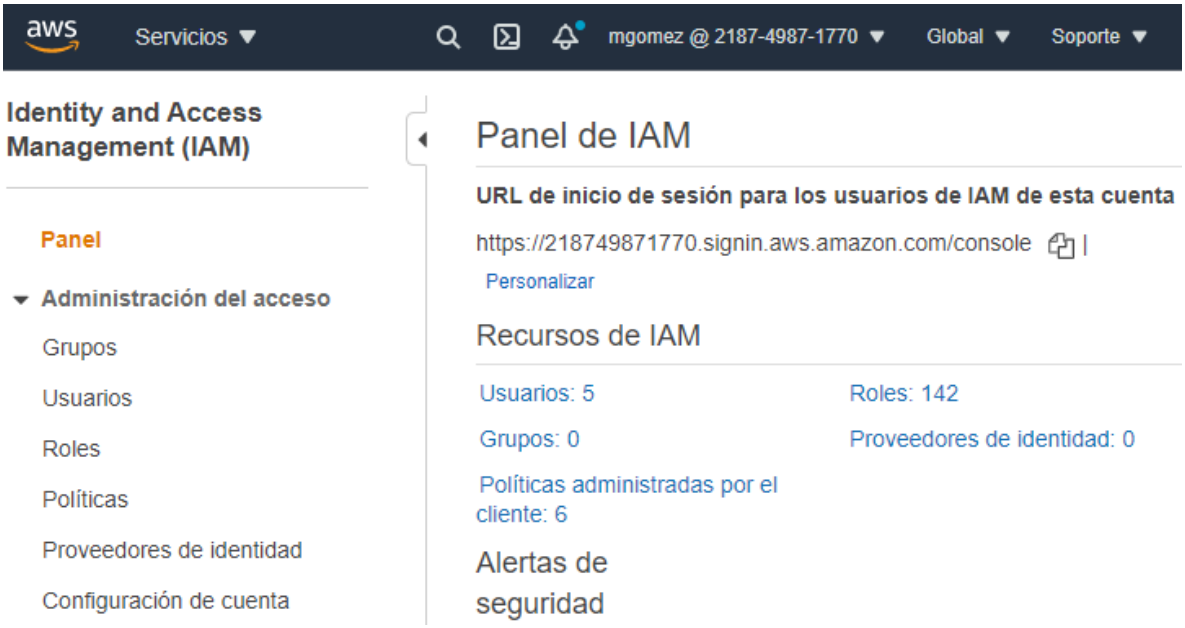


Imagen 12. Panel principal de IAM donde se pueden controlar los distintos usuarios generados por una cuenta de usuario raíz con los permisos y roles correspondientes.
Fuente: Elaboración propia, basada en la práctica.

2.5.2 Cognito

Amazon Cognito ofrece grupos de usuarios y de identidades. Por una parte, los grupos de usuarios son directorios de usuarios que proporcionan a los clientes de las aplicaciones opciones para registrarse e iniciar sesión. Por otra parte, los grupos de identidades proporcionan las credenciales de AWS para conceder a los usuarios acceso a otros servicios de AWS. Este servicio permite, de manera rápida y sencilla, incorporar a las aplicaciones web y móviles funcionalidades, como el control de acceso, la inscripción y el inicio de sesión de los usuarios, también cuenta

con escalado para una gran cantidad de usuarios y admite el inicio de sesión mediante proveedores de identidad social, como Facebook, Google y Amazon.

Una de las características de Amazon Cognito es el control del acceso a los recursos de AWS desde su aplicación, donde se pueden definir los roles y asignar usuarios a diferentes roles para que el sistema que utilice este servicio únicamente pueda acceder a los recursos autorizados para cada usuario. Otra característica importante es que la integración con la aplicación es sencilla gracias a una interfaz de usuario integrada y se configuran los proveedores de identidades federadas (soluciones para abordar la gestión de identidad en los sistemas de información) de manera simple.

Los usuarios de la aplicación desarrollada en la presente PPS pueden iniciar sesión directamente a través de un grupo de usuarios el cual administra la sobrecarga que conlleva el tratamiento de los tokens que se devuelven desde el inicio de sesión de redes sociales mediante Facebook, Google, Amazon y Apple. Después de una autenticación correcta, Amazon Cognito devuelve tokens de grupos de usuarios a la aplicación, que se pueden utilizar para conceder a los usuarios acceso a los recursos del lado del servidor que le pertenecen a Amazon API Gateway. La gestión y administración de los tokens de grupos de usuarios para la aplicación se realiza del lado del cliente por medio de los SDK de Amazon Cognito. Una vez que el usuario haya iniciado sesión correctamente, Amazon Cognito crea una sesión y devuelve un token de *ID* (identificación), acceso y actualización para el usuario autenticado.

Mediante la integración de Amazon Cognito con el código cliente o frontend se conecta la aplicación a la funcionalidad de AWS de backend, que asiste en los flujos de trabajo de autenticación y autorización. La aplicación utilizará la API de Amazon Cognito para crear nuevos usuarios en el grupo de usuarios, recuperar tokens de grupos de usuarios y obtener credenciales temporales del grupo de identidades. Cuando se integró Amazon Cognito con el frontend del sistema desarrollado se utilizó uno de los servicios de AWS, que permite realizar la unificación de las características mencionadas en el inicio de sesión con el backend y se trata del framework AWS Amplify. La incorporación de AWS Amplify ofreció servicios y bibliotecas para el desarrollo utilizando Amazon Cognito como principal

proveedor de autenticación, ya que es un servicio sólido de directorio de usuarios que gestiona el registro de usuarios, la recuperación de cuentas y otras operaciones.

La aplicación desarrollada permite que los mecánicos tengan un inicio de sesión social (*OAuth*). OAuth 2.0 es el framework de autorización común que utilizan las aplicaciones web y móviles para obtener acceso a la información del usuario de manera limitada. Para que *OAuth* sea compatible con AWS Amplify se utiliza *Cognito User Pools* (Un *User Pool* es un directorio de usuario) y se admite la federación con proveedores sociales, que crearán automáticamente un usuario correspondiente en *User Pool* después de iniciar sesión. Los tokens *OIDC*[4] están disponibles en la aplicación después de que esta haya completado el acceso. Para el inicio de sesión se añadió a la aplicación de tasaciones la utilización de Google para realizar el ingreso al sistema por parte de los usuarios, ya que Amazon Cognito se integra con Google para ofrecer una autenticación federada a los usuarios de aplicaciones móviles. En la imagen 13 se puede observar la pantalla de inicio de sesión al sistema, donde hay botones para ingresar con Google y con Facebook. Como se mencionó antes ya se encuentra implementado el ingreso a la aplicación a través de Google registrándose con una cuenta de gmail, en cuanto a Facebook se espera añadirlo en una futura implementación.

[4] OpenID Connect (OIDC) es una capa de autenticación o identidad sobre el protocolo OAuth 2.0 que permite a los clientes informáticos verificar la identidad de un usuario final en función de la autenticación realizada por un servidor de autorización, así como obtener información de perfil básica.

Imagen 13. Inicio de sesión en la aplicación de tasaciones realizada en la presente PPS utilizando Google como método de autenticación.
Fuente: Elaboración propia, basada en la práctica.

2.5.3 AWS Amplify

AWS Amplify ofrece servicios y bibliotecas con las cuales se pueden crear aplicaciones, que se integren con entornos de backend compuestos por servicios de AWS. Con Amplify se pueden configurar backends de aplicaciones y conectarlas de manera rápida, implementar aplicaciones web estáticas con tan solo unos clics y administrar el contenido de estas fácilmente fuera de la consola de AWS. En la práctica realizada se utilizó este framework enfocándose principalmente en la integración del backend con el frontend para luego administrarlo desde la consola, donde se puede ver el estado del despliegue y la puesta en producción. En la aplicación de tasaciones se usaron frameworks y herramientas que Amplify admite, tales como JavaScript y Angular, pero también se pueden usar otras herramientas como React, Vue, Next.js, así como plataformas móviles, incluidas Android, iOS, React Native, Ionic o Flutter.

Las principales características de AWS Amplify son las siguientes:

- Configuración rápida de backends: utilizando los flujos de trabajo intuitivos de la CLI y la interfaz de usuario de administración de Amplify se pueden configurar backends de AWS escalables con autenticación, almacenamiento de datos y otros casos de uso comunes.
- Conectar frontends sin problemas: se utilizan las bibliotecas de Amplify en la aplicación para conectarse a recursos de AWS nuevos y existentes con tan solo algunas líneas de código.
- Implementación con pocos clics: se utiliza la consola de Amplify para alojar aplicaciones web de una sola página (siendo en la práctica la aplicación de tasaciones con el frontend desarrollado en Angular) con un flujo de trabajo basado en GIT. Para conseguir la implementación, tan solo se debe conectar el repositorio de la aplicación, donde se propuso que el mismo pertenezca a Github, que es una plataforma de alojamiento de código para el control de versiones y la colaboración que utiliza la empresa con sus diversos proyectos.
- Administrar contenido fácilmente: se utiliza la interfaz de usuario de administración de Amplify para conceder acceso administrativo a quienes no son desarrolladores para administrar usuarios y contenido de aplicaciones sin una cuenta de AWS.

En la imagen 14 se pueden observar las etapas de la utilización de AWS Amplify en el desarrollo de la aplicación de tasaciones, la cual consta de tres pasos: la configuración del backend, conectar la aplicación e integrar los componentes UI.

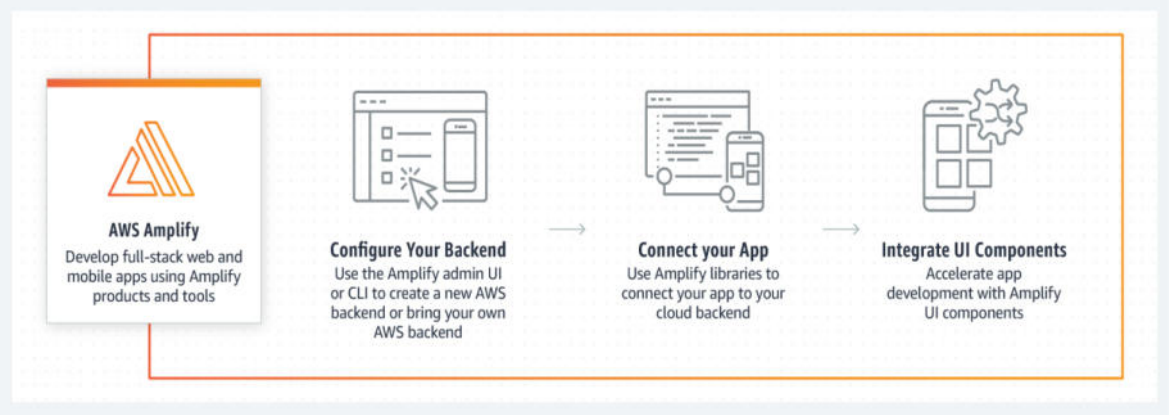


Imagen 14. Etapas en el desarrollo de una aplicación utilizando AWS Amplify.
Fuente: Recuperado de https://aws.amazon.com/es/amplify/ (2021)

1- Configuración del backend

Se utiliza la interfaz de usuario o la CLI de administración de Amplify para crear un nuevo backend de AWS o traer un backend propio de AWS, en el desarrollo realizado en la práctica y, en el caso de la plantilla empleada inicialmente, esta viene con una configuración del backend establecida para Amplify. Desde la raíz del proyecto se ejecuta lo siguiente:

```
amplify init
```

De esta manera queda configurado el backend de la aplicación con Amplify y se crean nuevos elementos:

- Un directorio de nivel superior llamado *amplify,* que almacena la definición de backend.
- Se crea un archivo llamado *aws-exports.js* en el directorio *src,* que contiene toda la configuración de los servicios que crea con Amplify.
- Se modifica el archivo *.gitignore* agregando algunos archivos generados a la lista de ignorados.

2- Conectar la aplicación

Se utilizan las bibliotecas de Amplify para conectar la aplicación al backend en la nube. El entorno de backend generado se puede administrar por un usuario administrador y es un contenedor para todas las capacidades de la nube agregada en la aplicación, como API, autenticación y almacenamiento.

3- Instalar los componentes UI

Se pueden ingresar componentes de la interfaz de usuario de Amplify para acelerar el desarrollo de la aplicación, pero en la práctica se toman las interfaces definidas en la plantilla.

En la imagen 15 se observan las etapas en las que se entregan y alojan las aplicaciones web estáticas como el frontend del sistema de tasación a través de los productos y herramientas de Amplify. El proceso cuenta con los siguientes pasos:

1- Conectar el repositorio

Se conecta el código fuente desde un repositorio de Git o cargando archivos en la consola de Amplify. En el desarrollo se emplea el repositorio de Github creado para el código fuente de la aplicación de tasaciones.

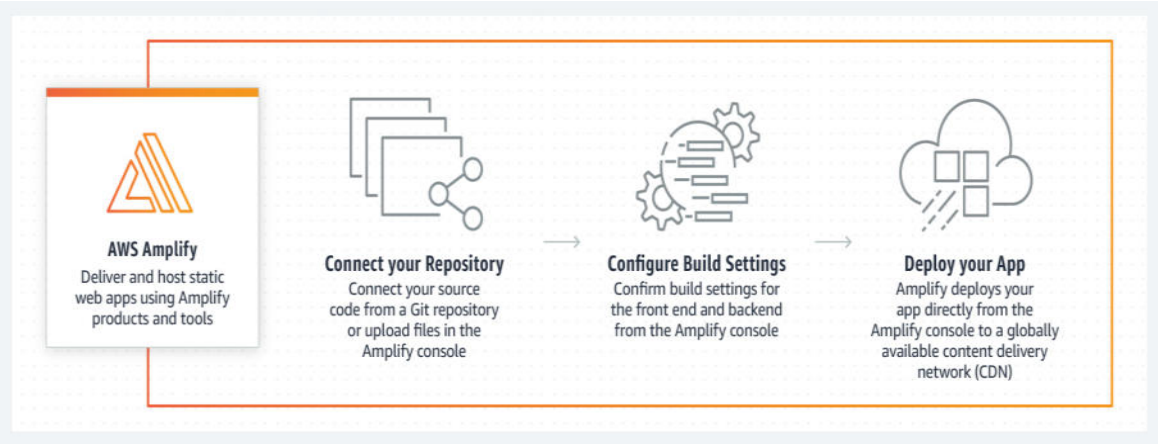


Imagen 15. Etapas en la puesta en producción de una aplicación utilizando AWS Amplify.
Fuente: Recuperado de https://aws.amazon.com/es/amplify/ (2021)

2- Configurar las características de la compilación

Se realiza la configuración de la compilación para el frontend y el backend desde la consola de Amplify, donde en el panel de administración se pueden ver los detalles y el estado de compilación del entorno del frontend y el backend junto a las ramas utilizadas en *git* para el desarrollo.

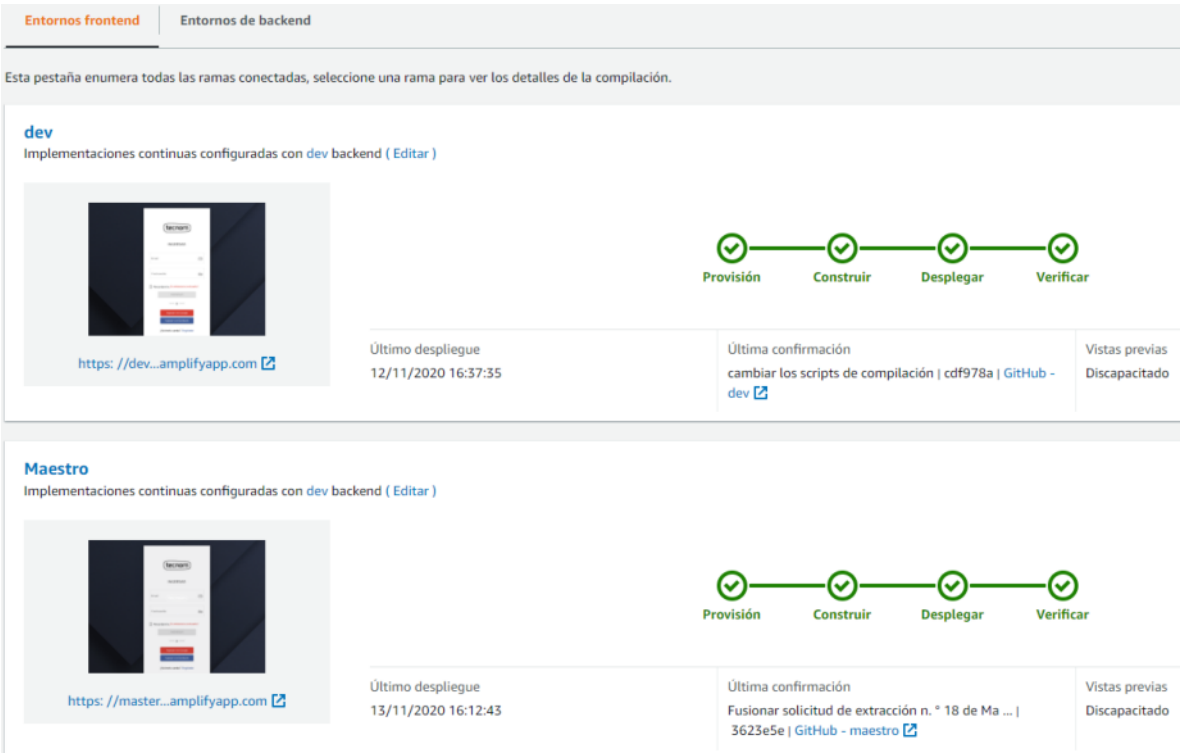


Imagen 16. Inicio de sesión en la aplicación de tasaciones realizada en la presente PPS utilizando Google como método de autenticación.
Fuente: Elaboración propia, basada en la práctica.

3- Deploy de la aplicación

En esta etapa Amplify implementa la aplicación directamente desde su propia consola en una red de distribución de contenido disponible a nivel mundial. En la imagen 16 se muestra la configuración y el estado de la compilación de la aplicación de tasaciones en las distintas ramas (*master* como la rama principal y *dev* como la rama en la que se desarrollan las mejoras y se hacen las pruebas) una vez que la app se encuentra desplegada o puesta en producción.

2.5.4 DynamoDB

Amazon DynamoDB es una base de datos de *clave-valor* y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos completamente administrada, duradera, multiactiva y de varias regiones que cuenta con copia de seguridad, restauración y seguridad integradas,

así como almacenamiento de caché en memoria para aplicaciones a escala de internet. Esta base de datos posee los siguientes beneficios:

- Rendimiento a gran escala: DynamoDB admite algunas de las aplicaciones de escala más grandes del mundo, proporcionando tiempo de respuesta de gran velocidad, por lo que se pueden crear aplicaciones con capacidad de almacenamiento y procesamiento prácticamente ilimitada. Las tablas globales de DynamoDB replican los datos en varias regiones de AWS para darle acceso rápido y local a los datos para las aplicaciones distribuidas globalmente.
- No es necesario administrar servidores: con DynamoDB no hay servidores que aprovisionar, parchear o administrar y no hay software que instalar, mantener o utilizar. Esta base de datos proporcionada como servicio de AWS aumenta o reduce automáticamente las tablas para ajustar la capacidad y mantener el rendimiento. La disponibilidad y la tolerancia a errores están integradas, por lo que no es necesario tener en cuenta estas capacidades a la hora de diseñar aplicaciones. Es importante hacer hincapié en este beneficio, ya que la empresa prioriza la creación de aplicaciones *serverless* (también llamadas aplicaciones con arquitectura sin servidor) y esto permite crear y ejecutar sistemas sin tener que administrar infraestructura. Aunque se menciona que la aplicación tenga una arquitectura sin servidor, la aplicación continúa ejecutándose en servidores, pero AWS se encarga de toda la administración de los mismos. DynamoDB proporciona los modos de capacidad bajo demanda y de capacidad por carga de trabajo o el pago de los recursos que se encuentran consumiendo.
- Admite las transacciones ACID[5] para que se puedan crear aplicaciones de vital importancia para el negocio a escala. DynamoDB cifra todos los datos de forma predeterminada y proporciona un control de acceso e identidad detallado en todas las tablas.

[5] El modelo ACID explica que en las transacciones (conjunto de operaciones) que se deben realizar en una base de datos de manera conjunta tendrán que realizarse todas las operaciones o ninguna cumpliendo las reglas que marca el acrónimo ACID (Atomicity, Consistency, Isolation, Durability) en los datos.

2.5.5 AWS AppSync

AWS AppSync es un servicio completamente administrado que facilita el desarrollo de API de GraphQL ya que se encarga de conectar de manera segura los orígenes de datos como AWS DynamoDB. Este servicio presenta las siguientes características:

- Utilización de GraphQL, un lenguaje de datos que permite a aplicaciones cliente conseguir, modificar y suscribirse a datos de servidores. En una consulta de GraphQL, el cliente especifica la manera en la que los datos deben estructurarse cuando el servidor los devuelve, esto permite que la aplicación cliente o el frontend de la misma consulte únicamente los datos que necesita y en el formato requerido.
- Acceso a datos simple y seguro: se respalda el funcionamiento de las aplicaciones con los datos correctos, provenientes de uno o más orígenes de datos con una solicitud de red simple mediante GraphQL. AWS AppSync facilita la protección de los datos de la aplicación mediante múltiples modos de autenticación concurrentes, además de permitir definir la seguridad, la memoria en caché y el control de acceso detallado en el nivel de definición de datos directamente desde el esquema creado en GraphQL.
- Capacidades integradas en tiempo real y sin conexión: con las suscripciones administradas de GraphQL, AWS AppSync puede enviar actualizaciones de datos en tiempo real a través de Websockets[6] a una gran cantidad de clientes. Para aplicaciones móviles y web, AppSync también ofrece acceso a datos locales cuando los dispositivos están sin conexión y sincronización de datos con resolución de conflictos personalizable, cuando vuelven a estar en línea.
- No hay que administrar servidores: se ofrece una configuración de API completamente administrada de GraphQL y mantenimiento con una infraestructura sin servidor de alta disponibilidad integrada. La API de

[6] Websocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP (orientados a la conexión). Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

GraphQL se crea en cuestión de segundos a través de Amplify CLI, método que fue utilizado en la presente práctica para crear las consultas en la base de datos con GraphQL, pero también se puede crear esta API con otros servicios como AWS CLI o CloudFormation.

Cuando se crean los modelos de datos que se utilizan en la aplicación de tasaciones se debe definir la API en base a este modelo. El primer paso para crear una API de AppSync consiste en diseñar el *esquema*, el cual está compuesto por los tipos de datos *objeto, input, interfaz, unión, enum* y *escalar* organizados como un gráfico. Los archivos de esquema son archivos de texto normalmente denominados *schema.graphql*, estos pueden ser creados y enviados a AWS AppSync usando la CLI o navegando a la consola y añadiendo en la página *Schema* el código necesario, tal como se muestra en la imagen 17.

Cada API de GraphQL se define mediante un único esquema de GraphQL. El sistema *Type* de GraphQL describe las funcionalidades del servidor y se utiliza para determinar si una consulta es válida. Un sistema *Type* del servidor se denomina esquema de dicho servidor y está compuesto por un conjunto de tipos de objeto, escalares, tipos de entrada, interfaces, enums y uniones. Además, define la forma de los datos que fluyen por la API y también las operaciones que se pueden realizar. GraphQL es un protocolo con establecimiento inflexible y todas las operaciones de datos se validan en función de dicho esquema. En la imagen 17 se puede ver que el *type* está definido por los modelos de datos de la aplicación de tasación, tal como *Appraisal* (tasación), *CarPhoto* (fotos del vehículo) y *DocPhoto* (Imágenes de los documentos del vehículo), los cuales contienen distintos campos que representan el valor y el tipo de datos de cada *type* definidos en la base de datos *NoSQL*[7] que utiliza el método simple de *clave/valor*[8] para almacenar datos. Se tienen en cuenta las relaciones de los modelos en la definición del esquema, puesto que *Appraisal* contiene fotos de vehículos y documentos y debe relacionar

[7] Las bases de datos NoSQL están diseñadas específicamente para modelos de datos específicos y tienen esquemas flexibles para crear aplicaciones modernas. Este tipo de bases de datos son ampliamente reconocidas porque son fáciles de desarrollar, por su funcionalidad y el rendimiento a escala.

[8] Se refiere a un método simple de almacenamiento de datos como una matriz asociativa o estructura de datos. En cada par *clave/valor*, la clave se representa mediante una cadena arbitraria, como un nombre de archivo y el valor puede ser cualquier tipo de datos, como una imagen, archivo o documento.

esta lista de datos con los modelos de *CarPhoto* y *DocPhoto* a través de un modelo *Connection,* que contenga los ítems de los objetos que se subieron a la base de datos. Al momento de crear el esquema, se generan un par de campos definidos por defecto que indican la fecha de cuándo se creó el dato en la base de datos (definido con la clave *createdAt*) y la fecha de la última actualización que tuvo (definido con la clave *updatedAt*). La fecha está representada por el tipo de escalar *AWSDateTime*, que se refiere a una cadena de fecha y hora *ISO 8601* ampliada, un estándar internacional que cubre el intercambio de datos relacionados con la fecha y la hora.

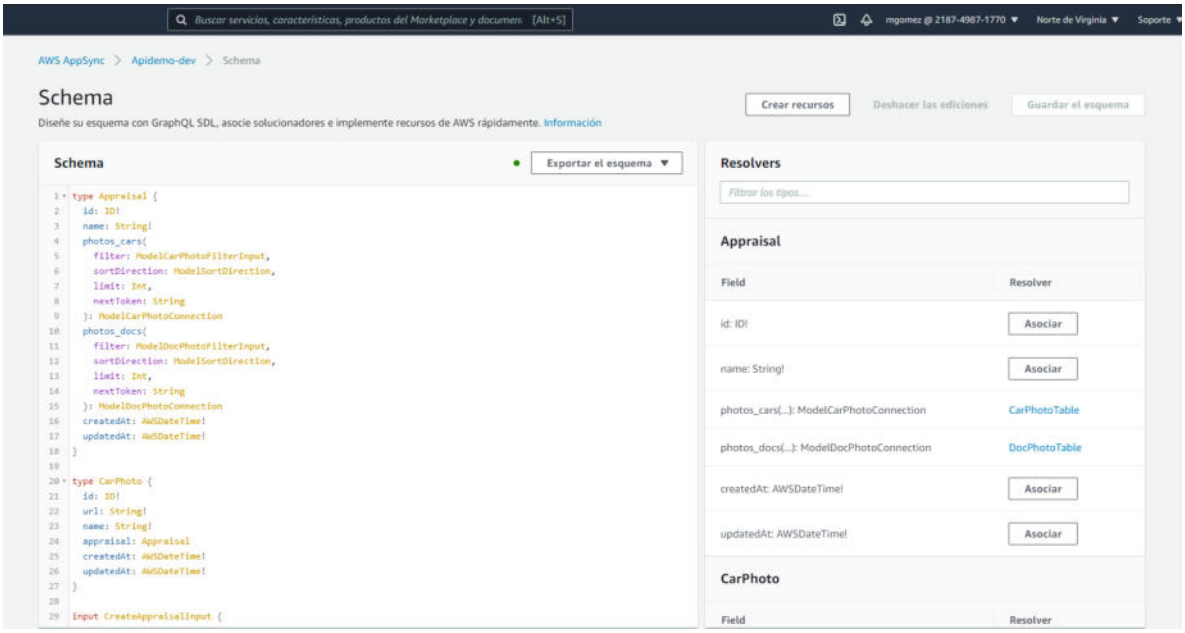


Imagen 17. Esquema definido con la API de GraphQL visualizado en el panel de la sección Schema del servicio AppSync de AWS.
Fuente: Elaboración propia, basada en la práctica.

En la imagen 18 se definen los orígenes de datos, que son recursos en la cuenta de AWS con los que las API de GraphQL pueden interactuar. Una API de AWS AppSync se puede configurar para interactuar con varios orígenes de datos, permitiendo agregarlos en una sola ubicación como los proporcionados por las tablas de DynamoDB desde una definición del esquema. AppSync puede crear automáticamente versiones de los datos para cada solicitud cuando varios clientes

intentan actualizarlos a la vez. El control de versiones se utiliza para mantener múltiples variantes de los datos con fines de detección y resolución de conflictos.

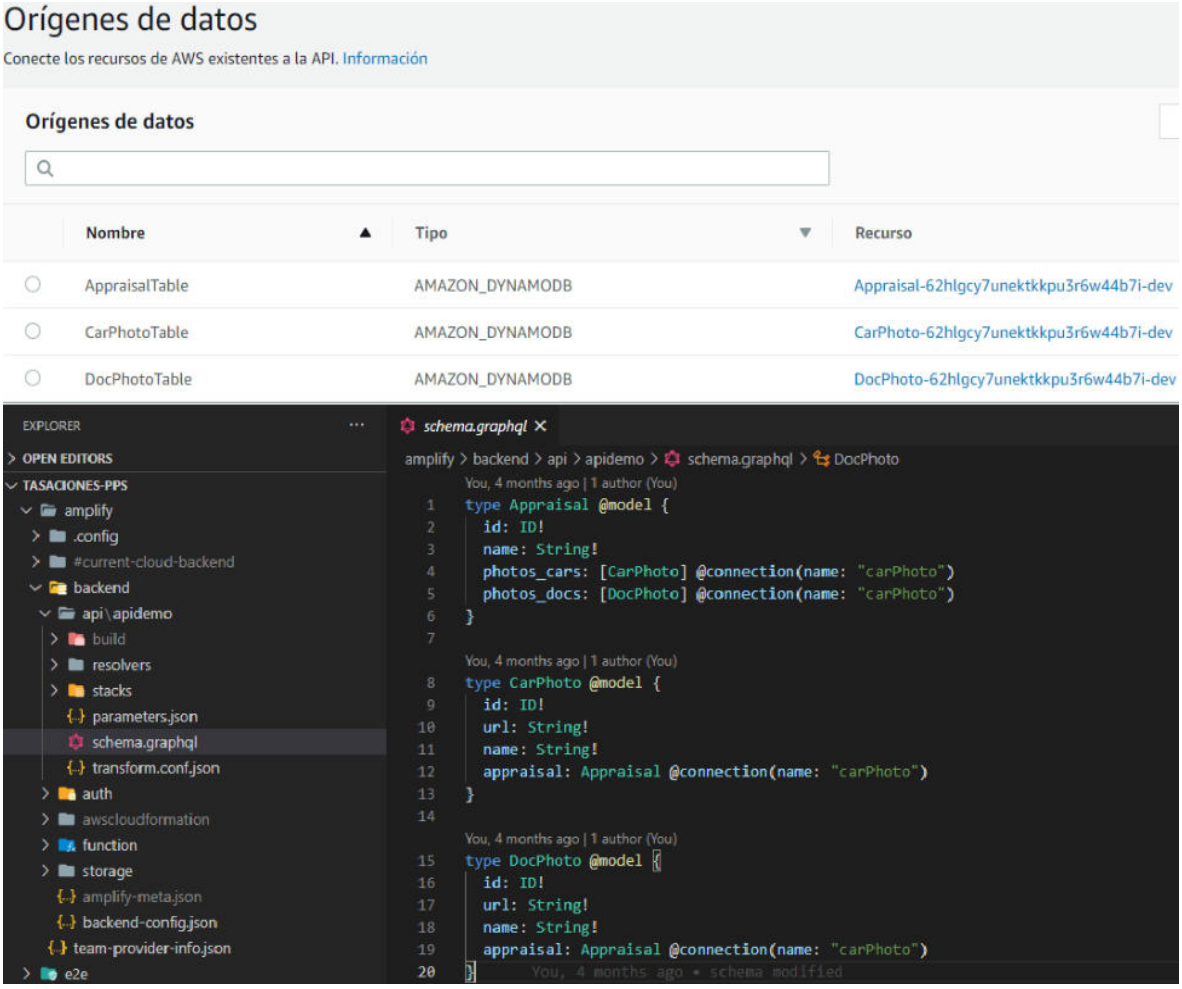


Imagen 18. Orígenes de los datos mostrados como tablas de DynamoDB y su definición en el código fuente de la aplicación.
Fuente: Elaboración propia, basada en la práctica.

2.5.6 Amazon S3

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento. Se pueden almacenar y proteger cualquier volumen de datos para los más variados fines, como usarlos en lagos de datos, sitios web,

aplicaciones móviles, procesos de copia de seguridad y restauración, operaciones de archivado, aplicaciones empresariales, dispositivos IoT y análisis de big data. Amazon S3 proporciona características de administración fáciles de utilizar, que permiten organizar los datos y configurar sofisticados controles de acceso con el objetivo de satisfacer los requisitos empresariales, organizativos y de conformidad. Este servicio de AWS proporciona los siguientes beneficios:

- Rendimiento, escalabilidad, disponibilidad y durabilidad líderes en el sector: esto significa que los datos están disponibles cuando se necesitan y protegidos frente a errores y amenazas. Amazon S3 también ofrece una consistencia sólida de lectura tras escritura de forma automática, sin coste alguno ni cambios en el rendimiento o la disponibilidad.
- Grandes capacidades de seguridad, conformidad y auditoría: los datos almacenados en Amazon S3 se encuentran protegidos del acceso no autorizado a través de características de cifrado y herramientas de administración de acceso. S3 es el único servicio de almacenamiento de objetos que permite bloquear el acceso público a todos los objetos de la aplicación a nivel de bucket[9], tales como las fotos de los autos y documentos. Administración de datos y fácil acceso a los controles: S3 ofrece capacidades sólidas para administrar el acceso, el costo, la replicación y la protección de datos. Además, los puntos de acceso de S3 facilitan la administración del acceso a los datos con permisos específicos para las aplicaciones utilizando un conjunto de datos compartidos. También, la replicación de S3 administra la replicación de datos dentro de la región o de otras regiones y las operaciones por lote ayudan a administrar los cambios a gran escala entre grandes cantidades de objetos.

2.6 Codificación de la aplicación

En esta sección se mencionan las principales características del desarrollo de la aplicación de tasaciones en cuanto a la escritura de código, tales como: la

[9] Un bucket de Amazon S3 es un recurso de almacenamiento en la nube pública disponible en el S3 de AWS, son similares a las carpetas de archivos, almacenan objetos, que consisten en datos y sus metadatos descriptivos.

funcionalidad y lógica de los componentes, las clases que se desarrollaron, las dependencias y librerías empleadas, la forma de efectuar una consulta a la base de datos y las respuestas obtenidas en las peticiones con la utilización de los servicios desarrollados con Angular.

2.6.1 Componentes principales

La aplicación cuenta con diversos componentes realizados en el frontend de la misma, de los que se mencionaron y detallaron los elementos de diseño en la sección 2.4. “Maquetación de la aplicación”. Estos componentes, además del código empleado para su visualización en la pantalla del usuario, cuentan con una lógica que se siguió en la programación de cada uno. El lenguaje de programación utilizado fue TypeScript siguiendo el paradigma *orientado a objetos* y se utilizaron los siguientes componentes:

Tasación

Definida como *appraisal.component.ts*, cuenta con distintos atributos que se encargan de obtener el valor de los elementos creados al momento de originar una nueva tasación. Cuando se genera una instancia de la clase de este componente, el usuario puede presionar el botón “tomar foto” y, en caso de presionarlo, se toma la URL del objeto que en ese momento se guarda en memoria. Luego se crea una instancia de solicitud *http* que será utilizada para guardar la URL en la base de datos. La foto es un objeto, por lo tanto, se utiliza el servicio de S3 para llamar a un método de la clase *Storage* de este servicio el cual guarda el archivo en un *bucket* de S3 al que el usuario tiene acceso y se devuelve al usuario un mensaje en caso de que hubiese un fallo en la subida del objeto al *bucket* de la aplicación.

En la imagen 19 se observa una porción de código perteneciente a los métodos que crean la tasación en este componente. El método *sabeAppraisal* hace uso de un servicio que llama a la API para cargar la solicitud a la misma y así generar los elementos de la tasación en la base de datos. Se utilizan funciones asíncronas como *async* y *await* cuya finalidad es simplificar el comportamiento del uso síncrono

de *promesas* y realizar algún comportamiento específico en un grupo de *Promises*[10]. Cuando la función *async* devuelve un valor, *Promise* se resolverá con el valor devuelto, en cambio, si se produce una excepción o algún valor, *Promise* se rechazará con el valor generado. La expresión *await* pausa la ejecución de la función asíncrona y espera la resolución de la *Promise* pasada y, a continuación, reanuda la ejecución de la función *async* y devuelve el valor resuelto. El método *addPhotosToAppraisal* definido utiliza las funciones asíncronas mencionadas para obtener las fotos subidas al *storage* y consultar por su URL, consiguiendo el valor del enlace de la foto que se subirá a la base de datos, la cual solo podrá ser visto por el usuario creador de la tasación con las fotos que él mismo subió.

```
176       You, 4 months ago • added create appraisal with API service;
177   async saveAppraisal(appraisal: Appraisal) {
178     await this._api.createAppraisal(appraisal).then(event => {
179       this.appraisalID = event.id;
180       console.log('Appraisal created!');
181       this.addPhotosToAppraisal();
182     })
183       .catch(e => {
184         console.log('error creating appraisal...', e);
185       });
186   }
187
188   public async addPhotosToAppraisal() {
189     for (let photo of this.photosArrayCar) {
190       const signedURL = await Storage.get(photo.name);
191       this.photoCarCreate = {
192         carPhotoAppraisalId: this.appraisalID,
193         url: signedURL,
194         name: photo.name
195       }
196       console.log(this.photoCarCreate);
197       this._apiPhoto.createCarPhoto(this.photoCarCreate);
198     }
199     for (let photo of this.photosArrayDoc) {
200       const signedURL = await Storage.get(photo.name);
201       this.photoDocCreate = {
202         docPhotoAppraisalId: this.appraisalID,
203         url: signedURL,
204         name: photo.name
205       }
206       this._apiPhoto.createDocPhoto(this.photoDocCreate);
207     }
208   }
```

Imagen 19. Métodos de la clase Tasación encargados de crear una tasación y guardar las fotos en la base de datos y en el bucket de S3.
Fuente: Elaboración propia, basada en la práctica.

[10] Las promises se utilizan para la realización de tareas asíncronas, como por ejemplo la obtención de respuesta a una petición HTTP. Una promise puede tener 4 estados: pendiente, cumplida, rechazada, arreglada.

Detalle de tasación

En la aplicación se la define como *appraisal-detail.component.ts* y se compone de elementos que conforman la vista y otros que implementan cierta lógica en la misma. Por ejemplo, se puede visualizar en la pantalla del usuario la tasación que se seleccionó junto a sus detalles, los cuales se pueden editar. La edición de la tasación se presenta en el cambio de valores en los datos pertenecientes a la descripción del vehículo y en la eliminación o agregado de fotos, tanto en la sección de fotos del vehículo como en la de fotos de los documentos.

La funcionalidad que implementa la edición de los elementos es la misma que se utiliza en el componente de creación de tasaciones, con la diferencia que se agregan métodos que permiten eliminar objetos de la base de datos y el *bucket*. Para no repetir el código de implementación y consultas a la API, se crean servicios que son consultados por este componente y el de creación de tasaciones, de esta forma se reutiliza el código definido y solo se establecen las consultas con una instancia de la API del servicio requerido. Las características de los servicios definidos se explicarán en la siguiente sección.

Una de las librerías utilizadas para poder visualizar la fecha en la pantalla del usuario con un formato más adecuado y entendible fue *Momentjs*, donde implementando un método de cambio de formato, tal como *moment().format(data)* dentro del código, es posible obtener un cambio en los resultados devueltos por la base de datos con el formato provisto por *AWSDateTime*.

Lista de tasaciones

Definida como *appraisal-list.component.ts*, cuenta con un arreglo de las tasaciones realizadas por el usuario en el sistema. Cuando este entra a su lista de tasaciones desde la aplicación, se ejecutan las consultas a los servicios definidos en la API, tal como se ve en el método *ngOnOnit()* del código en la imagen 20. Luego se recorren todas las tasaciones realizadas por el usuario que están en la base de datos y se las guarda en un arreglo, para que sea recorrido y posteriormente mostrado en la pantalla del usuario. También se utiliza la librería de *MomentJs* para el cambio de formato de visualización de la fecha en la que se realizó cada tasación presente en la lista de tasaciones.

```
27    constructor(
28      private router: Router,
29      private _api: AppraisalService
30    ) { }
31
32    async ngOnInit() {
33
34      /* fetch appraisals when app loads */
35      this.showSpinner = true;
36      await this._api.getAppraisals().then(event => {
37        this.appraisals = event.items;
38      });
39
40      for (let element of this.appraisals) {
41        this.appraisalID = element.id;
42        await this.getAppraisal(this.appraisalID)
43      }
44      this.showSpinner = false;
45      console.log(this.allAppraisals);
46
47    }
48
49    async getAppraisal(id) {
50      var moment = require('moment');
51      moment.locale('es');
52      await this._api.getAppraisalsById(id).then(data => {
53        data.createdAt = moment(data.createdAt).format('L');
54        this.allAppraisals.push(data);
55      })
56    }
```

Imagen 20. Código que se ejecuta cuando el componente perteneciente a la lista de tasaciones es invocado, donde automáticamente se realizan las consultas necesarias a la base de datos.
Fuente: Elaboración propia, basada en la práctica.

Menú principal

Definida como *home-principal.component.ts*, cuenta mayormente con las características visuales y animaciones de la plantilla de Fuse Angular. Como funcionalidad implementa los métodos de la clase *Router* propia de Angular que sirve para navegar entre las distintas partes de la aplicación, en este caso la vista de la lista de tasaciones del usuario y la de creación de una nueva tasación.

2.6.2 API Service

En Angular, los servicios pertenecen a una categoría amplia que abarca cualquier valor, función o característica que necesite una aplicación. Asimismo, un servicio es típicamente una clase con un propósito limitado y bien definido. Angular distingue los componentes de los servicios para aumentar la modularidad y la reutilización. Al separar la funcionalidad relacionada con la vista de un componente de otros tipos de procesamiento, se puede hacer que las clases de los componentes sean más eficientes y simples de manejar o entender. Además, un componente puede delegar ciertas tareas a los servicios, como obtener datos del servidor, validar la entrada del usuario o iniciar sesión directamente en la consola. Amplify y AppSync facilitan las consultas a las bases de datos y las operaciones realizadas con las respuestas a través de un archivo denominado *API.service.ts* el cual toma el esquema de GraphQL e implementa todos los métodos necesarios para realizar peticiones.

El API Service contiene todas las consultas a las bases de datos y la forma en la que los datos son regresados, de esta manera se tiene conocimiento de cómo pedir una petición en la que se utilice este servicio para obtener algún dato en particular y visualizarlo en la pantalla del usuario. Para obtener una tasación se realiza una función asíncrona que devuelve una *promise* con los datos solicitados en una *query*, la cual establece el formato de la respuesta y la API de GraphQL se encarga de realizar las operaciones para traer los datos a la aplicación.

Se crearon dos servicios para aplicar la reutilización de código y a su vez no generar código duplicado en el desarrollo de cada componente, además, estos servicios obtienen una instancia del API Service. El primer servicio creado fue *appraisal.service*, el cual obtiene las tasaciones del usuario en la base de datos, tanto la lista completa o alguna tasación en particular obtenida por su identificador. Otras tareas que se realizan en *appraisal.service* son las de actualizar los datos de la tasación como el valor de la patente del vehículo y, también, se encarga de crear las fotos del vehículo y los documentos para cargarlos en la base de datos del sistema. El segundo servicio llamado *photo.service* se encarga de realizar las operaciones que eliminan las fotos del vehículo junto a sus documentos en la pantalla donde se visualizan los detalles de la tasación seleccionada por el usuario. Además, en *photo.service* se crean las nuevas fotos del vehículo al utilizar la

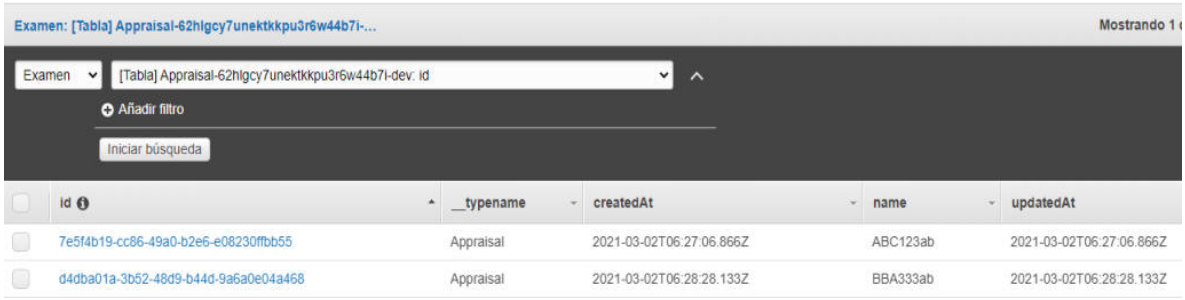
funcionalidad de agregado de fotos desde la tasación ya creada y no desde la pantalla de creación de nueva tasación.

2.7 Pruebas funcionales del desarrollo

Luego de que la aplicación de tasaciones se encuentre desarrollada y cumpliendo con los requerimientos funcionales establecidos, se prueban las funcionalidades de dicha app teniendo en cuenta elementos como la experiencia de usuario, la visualización de los componentes y que la cámara pueda utilizarse. Las funciones para iniciar sesión y la visualización de la pantalla de inicio están establecidas de manera correcta siendo testeadas a través de un usuario generado con Amazon Cognito. El usuario puede ingresar a la aplicación a través de su cuenta de correo de Google y con una contraseña proporcionada con Cognito, ya que se estableció como método de autenticación a Google, aunque se pueden generar otros métodos con Facebook o una cuenta de Amazon. Los elementos visuales que interactúan en este proceso con el usuario son los mismos que se definieron en las vistas de la aplicación, solo se modificaron pequeñas cuestiones estéticas como colores y tamaño de texto para mejorar la experiencia de usuario. Los tiempos de ingreso a la aplicación y su repercusión en la performance serán analizados en la sección 2.8. “Test y evaluación de usabilidad en producción “.

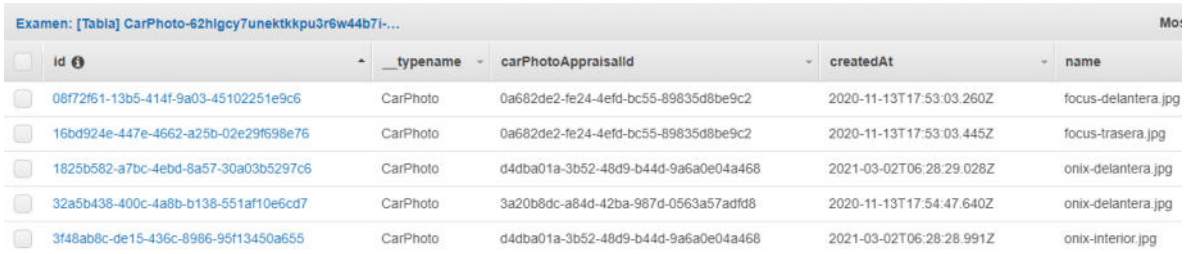
Una vez que el usuario se encuentra conectado dentro de la aplicación puede empezar a operar con las opciones del menú principal, donde la creación de la tasación cumple con las mismas funcionalidades explicadas en el diseño de las distintas vistas. Cuando se crea la tasación, esta se carga en la base de datos y como se muestra en la imagen 21 se puede observar la tabla presente en DynamoDB con la tasación creada, donde el usuario subió dos tasaciones indicando el número de patente y las fechas de creación y actualización de las tasaciones correspondientes. En la imagen 22 se muestra la tabla donde se guardan las fotos que son tomadas del vehículo y en la imagen 23 se observa la tabla donde se almacenan las fotos de los documentos de los vehículos. A pesar de que la lista de elementos de las tablas mencionadas de la base de datos se encuentre sin referenciar a qué tasación en particular pertenecen, es en la tabla *Appraisal*

(tasación) donde cada tasación tiene referenciada a cada elemento que la compone de otras tablas como las fotos.



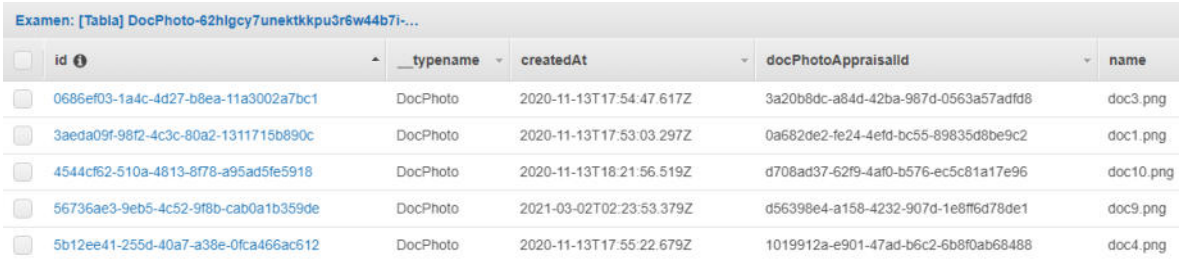
id	__typename	createdAt	name	updatedAt
7e5f4b19-cc86-49a0-b2e6-e08230ffbb55	Appraisal	2021-03-02T06:27:06.866Z	ABC123ab	2021-03-02T06:27:06.866Z
d4dba01a-3b52-48d9-b44d-9a6a0e04a468	Appraisal	2021-03-02T06:28:28.133Z	BBA333ab	2021-03-02T06:28:28.133Z

Imagen 21. Tasaciones creadas en la tabla "Appraisal" visualizadas a través del panel de administración de DynamoDB.
Fuente: Elaboración propia, basada en la práctica.



id	__typename	carPhotoAppraisalId	createdAt	name
08f72f61-13b5-414f-9a03-45102251e9c6	CarPhoto	0a682de2-fe24-4efd-bc55-89835d8be9c2	2020-11-13T17:53:03.260Z	focus-delantera.jpg
16bd924e-447e-4662-a25b-02e29f698e76	CarPhoto	0a682de2-fe24-4efd-bc55-89835d8be9c2	2020-11-13T17:53:03.445Z	focus-trasera.jpg
1825b582-a7bc-4ebd-8a57-30a03b5297c6	CarPhoto	d4dba01a-3b52-48d9-b44d-9a6a0e04a468	2021-03-02T06:28:29.028Z	onix-delantera.jpg
32a5b438-400c-4a8b-b138-551af10e6cd7	CarPhoto	3a20b8dc-a84d-42ba-987d-0563a57adfd8	2020-11-13T17:54:47.640Z	onix-delantera.jpg
3f48ab8c-de15-436c-8986-95f13450a655	CarPhoto	d4dba01a-3b52-48d9-b44d-9a6a0e04a468	2021-03-02T06:28:28.991Z	onix-interior.jpg

Imagen 22. Fotos de vehículos creadas en la tabla "CarPhoto" visualizadas a través del panel de administración de DynamoDB.
Fuente: Elaboración propia, basada en la práctica.



id	__typename	createdAt	docPhotoAppraisalId	name
0686ef03-1a4c-4d27-b8ea-11a3002a7bc1	DocPhoto	2020-11-13T17:54:47.617Z	3a20b8dc-a84d-42ba-987d-0563a57adfd8	doc3.png
3aeda09f-98f2-4c3c-80a2-1311715b890c	DocPhoto	2020-11-13T17:53:03.297Z	0a682de2-fe24-4efd-bc55-89835d8be9c2	doc1.png
4544cf62-510a-4813-8f78-a95ad5fe5918	DocPhoto	2020-11-13T18:21:56.519Z	d708ad37-62f9-4af0-b576-ec5c81a17e96	doc10.png
56736ae3-9eb5-4c52-9f8b-cab0a1b359de	DocPhoto	2021-03-02T02:23:53.379Z	d56398e4-a158-4232-907d-1e8ff6d78de1	doc9.png
5b12ee41-255d-40a7-a38e-0fca466ac612	DocPhoto	2020-11-13T17:55:22.679Z	1019912a-e901-47ad-b6c2-6b8f0ab68488	doc4.png

Imagen 23. Fotos de documentos creadas en la tabla "DocPhoto" visualizadas a través del panel de administración de DynamoDB.
Fuente: Elaboración propia, basada en la práctica.

En la imagen 24 se muestran los elementos que contiene el *bucket* creado con el servicio S3 de AWS para almacenar las fotos que fueron subidas en la aplicación de tasaciones al momento de crear la tasación. El formato de subida

admite tanto jpg como png para las imágenes, ya que así se estableció en el desarrollo al momento de indicar la utilización de S3 en el código para subir archivos. Este *bucket* se configuró con acceso público en el testeo de la funcionalidad para verificar que las fotos fueran subidas correctamente y se puedan visualizar dentro de la aplicación desarrollada, sin embargo, se requiere que se continúe trabajando en el futuro con las políticas de acceso al *bucket*.

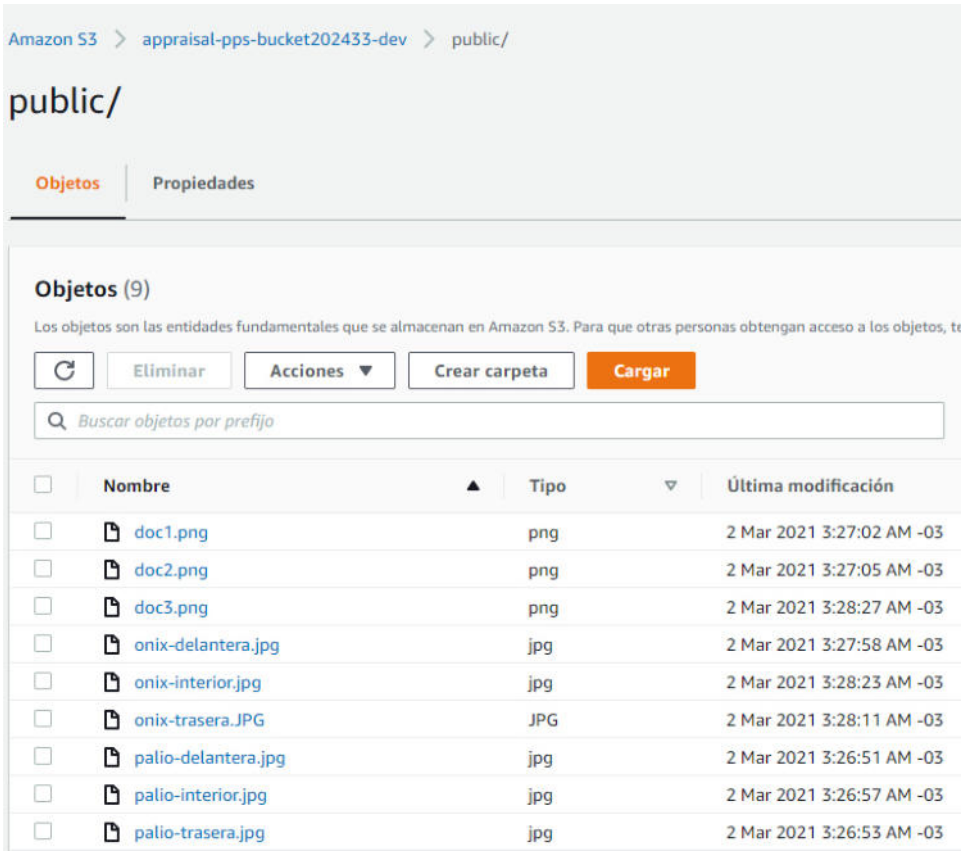


Imagen 24. Bucket creado automáticamente en S3 donde se suben fotos en la creación de una tasación o cuando se agregan nuevas fotos a una tasación existente.
Fuente: Elaboración propia, basada en la práctica.

Una vez que el usuario crea la tasación, se regresa a la pantalla principal y si se presiona el botón “ver tasaciones” se presentará al usuario una lista con las tasaciones que haya creado con la opción de ver detalles de cada una. En la imagen 25 se muestra en forma de lista cómo se ven las dos tasaciones creadas presentes en la base de datos.

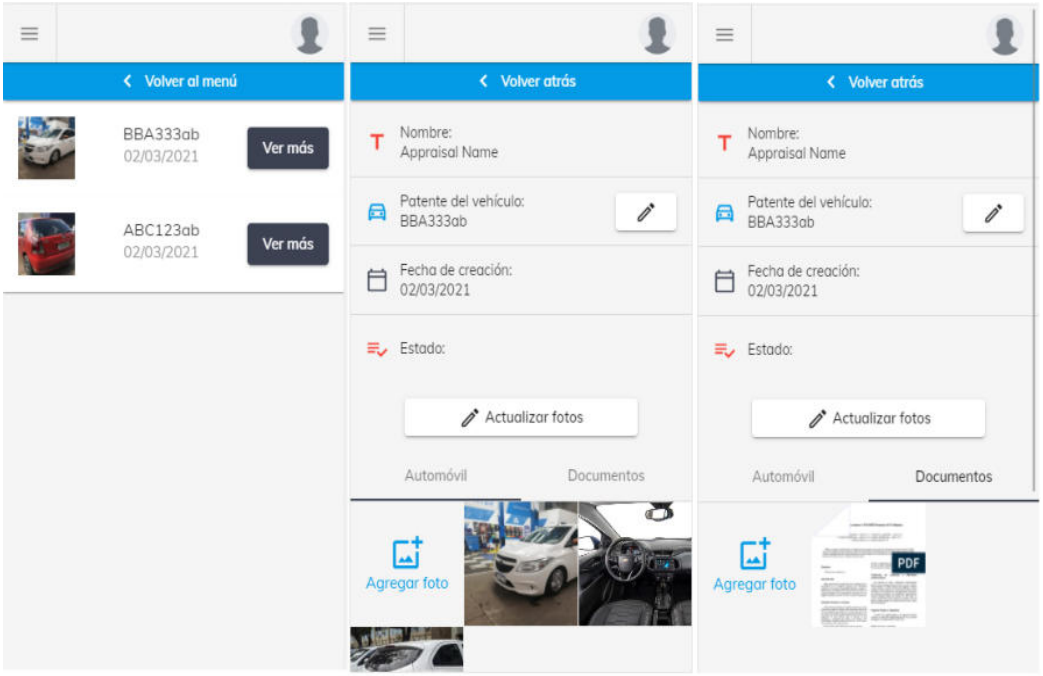


Imagen 25. Visualización en la pantalla del dispositivo de la lista de tasaciones del usuario y los detalles que se obtienen al ver alguna tasación presente en la lista.
Fuente: Elaboración propia, basada en la práctica.

Al pulsar el botón “ver más” en alguna de las tasaciones se visualizan los detalles de la tasación seleccionada, donde se muestra la patente del vehículo y se separan las fotos del vehículo de los documentos, además, se encuentran dos botones que permiten editar características de la tasación. Se puede ver la funcionalidad de los dos botones en la imagen 26, correspondiendo el primero a la actualización de las fotos y el segundo a la edición de la patente. Cuando se actualizan las fotos, en cada una de las imágenes se presenta un cuadro denominado *checkbox*, donde al seleccionar una imagen se rellena el cuadro con una tilde indicando que la foto fue seleccionada. Si se encuentran una o más fotos seleccionadas se habilita el botón de color rojo “Eliminar”, en el caso de no querer eliminar fotos se vuelve a presionar el botón “Actualizar fotos”, también es posible agregar nuevas fotos pulsando el recuadro “Agregar foto”. En la edición de la patente se muestra un cuadro de diálogo con un campo de texto que mantiene el dato de la patente con la posibilidad de reescribirla, luego al pulsar “Editar” dentro del mismo cuadro, se subirán los cambios a la base de datos. La imagen 27 muestra

cómo al presionar una foto esta se amplía y se visualiza con mayor resolución para ver la foto en su totalidad.

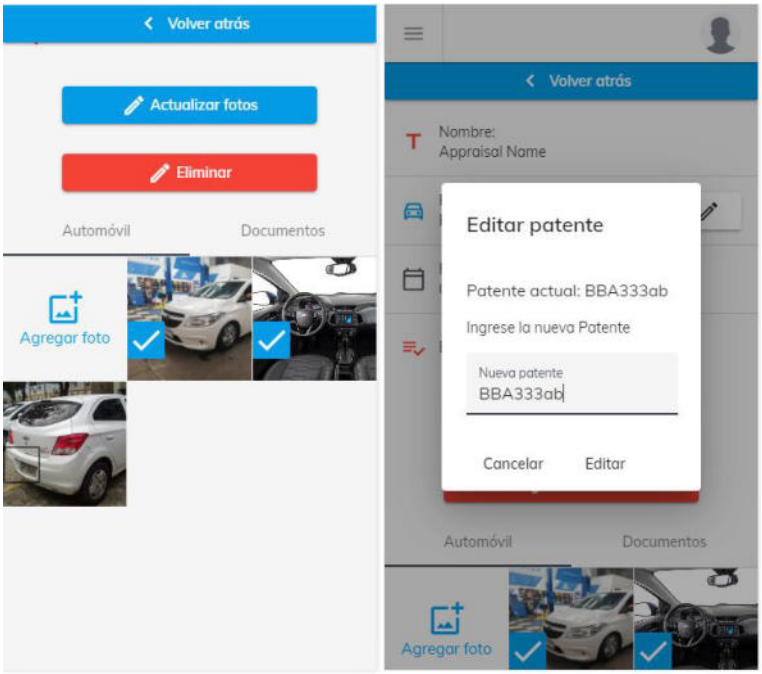


Imagen 26. Edición de una tasación en particular por parte del usuario, donde puede seleccionar distintas fotos para eliminarlas, agregar nuevas fotos y editar el nombre de la patente del vehículo.
Fuente: Elaboración propia, basada en la práctica.

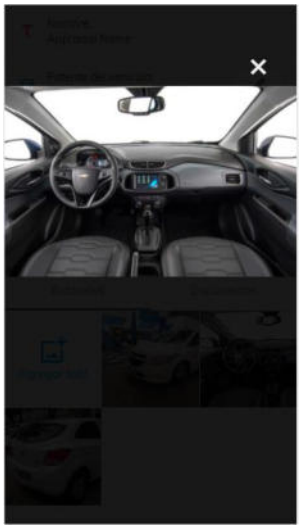


Imagen 27. Vista ampliada de una foto del vehículo subida en la tasación que se obtiene al pulsar por encima de alguna de las fotos presentes.
Fuente: Elaboración propia, basada en la práctica.

2.8 Test y evaluación de usabilidad en producción

Una vez que la aplicación ya se encuentre desarrollada, en producción y cumpliendo los requerimientos funciones establecidos, se evalúa el rendimiento que tiene en base a diferentes parámetros que se pueden medir en las PWA. Estos parámetros consisten en la performance, accesibilidad, mejores prácticas, SEO[11] (Search Engine Optimization) e instalación, que pueden ser medidos utilizando Lighthouse, una herramienta automatizada de código abierto para mejorar el rendimiento, la calidad y la corrección de la app. Al auditar la aplicación, Lighthouse ejecuta una serie de pruebas en la página y luego genera un informe sobre el rendimiento de la misma.

Cuando se realizó el análisis del rendimiento por primera vez, se obtuvieron los resultados que se muestran en la imagen 28, la cual muestra que la aplicación tiene una carga en el tiempo de ejecución y no cuenta con los requisitos de performance necesarios para que la PWA pueda ser instalada en el dispositivo. El índice de velocidad y el tiempo para interactuar de la página de ingreso e inicio es alto por lo que influye en la performance. El índice de velocidad muestra la rapidez con la que se completa visiblemente el contenido de una página y el tiempo para interactuar es la cantidad de tiempo que tarda la página en volverse completamente interactiva. Lighthouse indica que la carga de la página no es lo suficientemente rápida en las redes móviles, se carga demasiado lenta y no es interactiva en al menos diez segundos, además el archivo *service worker* necesario para la funcionalidad de la PWA no responde de forma positiva cuando está fuera de línea. Sin embargo, se realizaron las siguientes correcciones para mejorar la performance de la aplicación de tasaciones e incrementar los valores del test realizado por Lighthouse:

- Se eliminaron archivos de JavaScript que no se utilizaron y se mejoró el código de algunos archivos con el fin de simplificarlos y que sean más óptimos.

[11] El SEO se preocupa por la relevancia. Se asegura de que la web está optimizada para que el motor de búsqueda entienda lo principal, que es el contenido de la misma. Se incluye dentro de la relevancia del sitio a la optimización de keywords, el tiempo de carga, experiencias del usuario, optimización del código y formato de las URLs.

- Se eliminó el código CSS no utilizado
- Se cambiaron las imágenes de los logos de la aplicación por otras de mejor resolución.
- Se tuvieron en cuenta los contrastes generados por los colores en la página de inicio de sesión para no generar molestias al usuario en cuento a lo visual, también se corrigieron tamaños adecuados de botones y fuente de la letra para mejorar la accesibilidad.
- Se corrigió el acceso a la URL en una definición de *start_url* propio del *manifest.json* para que se pueda reconocer el Service Worker.

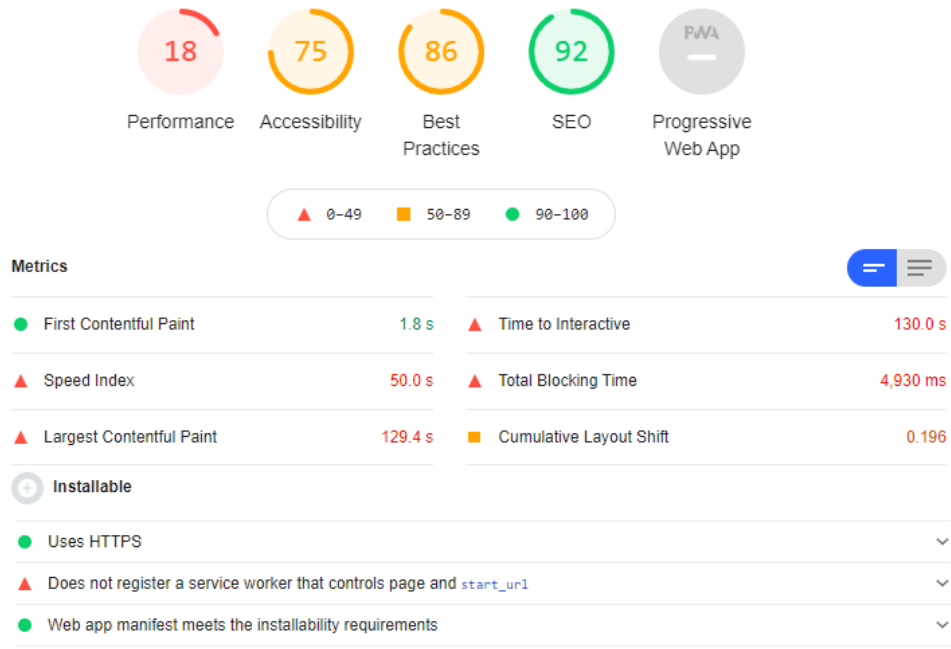


Imagen 28. Resultados obtenidos al realizar un análisis de rendimiento por primera vez con Ligthhouse, donde se presentan aspectos que se pueden mejorar para incrementar el rendimiento de la aplicación desarrollada.
Fuente: Elaboración propia, basada en la práctica.

En la imagen 29, se observan los resultados de la evaluación del rendimiento y las demás características de la PWA luego de realizar las correcciones mencionadas, en los cuales se notó mejoría en los valores obtenidos. En primer lugar, se alcanzaron valores totalmente favorables en la accesibilidad, mejores prácticas y el SEO de la aplicación, esto se debe a que se aprobaron gran parte de

las auditorías que aborda Lighthouse en estas variables. En segundo lugar, el valor de la performance no cambió debido a que hay tiempos que no son los adecuados, pero sí se muestran cambios en los tiempos de ejecución siendo estos más rápidos. Se puede mejorar la performance optimizando funcionalidades en el inicio de sesión y del almacenamiento en caché del dispositivo, pero no forma parte del alcance de los requerimientos no funcionales establecidos, por lo que puede tenerse en cuenta para una posible futura mejora de la aplicación. Por último, se muestra que la app puede ser instalable y se validan los aspectos de una aplicación web progresiva, puesto que el archivo *manifest.json* y el *Service Worker* cumplen con los requisitos de la instalación.

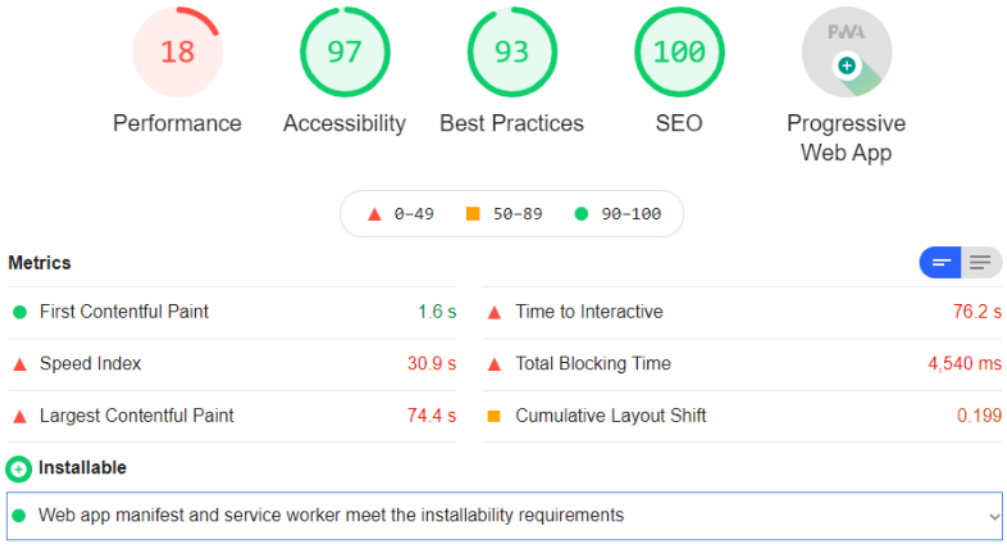


Imagen 29. Resultados obtenidos con Ligthhouse cuando se realizan las actividades recomendadas para mejorar los tiempos de ejecución, generando así, una PWA que cumpla con las condiciones de instalación en el dispositivo móvil.
Fuente: Elaboración propia, basada en la práctica.

3 Conclusiones

En la presente práctica profesional supervisada se tuvo como objetivo principal desarrollar un sistema que pudiera guardar información de las tasaciones de automóviles realizadas por los mecánicos. Es decir, que permitiera al usuario tener la facilidad de acceder, a través de una aplicación móvil a los datos de una tasación en particular, tales como: nombre, patente y fotos del vehículo junto a su documentación específica. Este objetivo fue alcanzado en su totalidad partiendo desde el análisis de las características propias de un sistema de este tipo. Además, se planteó el desarrollo y la funcionalidad del mismo abarcando el stack de tecnologías utilizada por la empresa Tecnom.

Para comprender la lógica que implementa un sistema de tasación en general y uno de cotización de vehículos en particular, se realizó una búsqueda previa en la web sobre estos tipos de aplicaciones. A la vez, se plantearon qué datos eran necesarios o no en la información que se pretendía guardar. De esta forma, se realizó un análisis de requerimientos funcionales y no funcionales comprendiendo el alcance del software a desarrollar.

Una vez que se analizaron las características que se debían alcanzar en el marco de los requerimientos definidos, se eligieron las principales herramientas con las que se trabajó el proyecto. Al poseer un stack que comprende diversas tecnologías de frontend (Angular, FUSE, Angular Material, PWA, Amplify) se facilitó el cumplimiento de uno de los objetivos específicos planteados, el cual hizo hincapié en la interfaz de la aplicación. El uso de una plantilla (FUSE) para crear la interfaz visual, optimizó los tiempos de codificación, asimismo, permitió un desarrollo y asociación a otras herramientas más eficientes.

Debido a que la utilización de la aplicación se enfocó en la experiencia del usuario, se consiguió un sistema sencillo de manejar y comprender, ya que se siguió el diseño que ofrecía la plantilla, además de otras herramientas que la empresa desarrolló anteriormente. No se necesitaron realizar muchos pasos ni acciones que pudieran causar confusión en su uso en un dispositivo móvil, para esto se definieron las distintas pantallas que conformaron el software. Es decir, para tomar fotos de los vehículos y guardarlos en una tasación, tanto nueva como antigua, tan solo

basta con presionar pocos botones que guían al usuario en el proceso que necesita realizar (crear, eliminar o modificar tasaciones).

En cuanto a los resultados obtenidos, estos fueron los esperados respecto a la lógica planteada, ya que se usaron correctamente los recursos del teléfono siguiendo la implementación que ofrecen las PWA. En términos de la eficiencia se encontraron desperfectos en los tiempos de ejecución en el inicio de la aplicación al realizar pruebas funcionales de la misma. Estos fueron corregidos utilizando una herramienta llamada Lighthouse, la cual ayudó a encontrar los errores que ralentizaban los procesos iniciales y analizó la performance del sistema para ser considerada una PWA eficiente.

3.1 Líneas futuras

El proyecto realizado cumple con los objetivos específicos planteados, sin embargo, existen características que se pueden mejorar e incluso agregar. A continuación, se enuncian algunos puntos que podrían trabajarse en un futuro.

En primer lugar, el usuario necesita una pantalla que le ofrezca ayuda en el uso de la aplicación si es que lo precisa, también es necesario que pueda tener información sobre a quién dirigirse en el caso de que se presente algún problema en la utilización. De esta manera, se podría trabajar en la vista y la lógica de una vista denominada “ayuda”, común en muchas aplicaciones.

En segundo lugar, se podría mejorar la carga de documentación de los vehículos en una tasación, ya que se encontraron limitaciones al cargar archivos de tipo documento y visualizarlos usando tecnologías de AWS como S3, que no formaba parte del alcance general del proyecto, ni del stack de tecnologías definido en un principio. También se puede mejorar la implementación de S3 teniendo en cuenta el acceso y las características de seguridad que ofrece esta herramienta.

Por último, es preciso recalcar el último objetivo específico planteado, que indica la integración del desarrollo en las soluciones que brinda la empresa. Debido a las ventajas que aportan las PWA, el sistema de tasación obtenido puede ayudar a seguir su lógica de desarrollo en la creación de futuras aplicaciones móviles que

sigan este paradigma. Además, se puede mejorar el software producido en cuanto a la eficiencia haciendo hincapié en la ejecución del mismo al cargar la pantalla de ingreso.

3.2 Reflexión sobre la Práctica Profesional Supervisada como espacio de formación

El haber realizado el proyecto propuesto en la presente Práctica Profesional Supervisada me ha ayudado a integrarme en el ambiente laboral por primera vez. Los puntos a destacar en el desarrollo de la misma fue el acople que se obtuvo con las metodologías de trabajo que se llevan a cabo en la empresa y también con los sistemas que utilizan para desarrollar sus productos. Además, fue de gran ayuda el seguimiento por parte del equipo de trabajo seleccionado para la supervisión, ya que mejoró la curva de aprendizaje de las herramientas necesarias para la práctica. Por otra parte, utilizar el stack de la organización me facilitó la adquisición de muchos conocimientos acerca del desarrollo web, tanto del frontend como del backend, acercándome a las tecnologías que se emplean en la actualidad para la elaboración de software.

Las distintas materias cursadas a lo largo de la carrera sirvieron para tener una mirada cercana al ambiente de trabajo y la PPS termina de englobar aquellos conocimientos técnicos y teóricos aprendidos. Fue de utilidad el aprendizaje obtenido en los cursos relacionados a la creación y al desarrollo de software, desde la definición de objetivos y requerimientos, además de la codificación e implementación de los sistemas.

Por último, es necesario destacar que la PPS es un espacio demasiado importante no solo por el aprendizaje, sino que también para la formación profesional como ingeniero. Además de la elección de realizar la práctica en una empresa, también se tuvo la posibilidad de realizar un trabajo de investigación que ayuda a obtener un perfil de ingeniero con enfoque en la investigación académica. Por lo tanto, la carrera de ingeniería en informática ofrece muchas posibilidades de crecimiento en distintos ámbitos del propio sector.

4 Referencias bibliográficas

- Amazon Cognito Developer Guide. Recuperado de https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-dg.pdf
- Amazon DynamoDB: Servicio de base de datos NoSQL rápido y flexible para cualquier escala. Recuperado de https://aws.amazon.com/es/dynamodb/
- Amazon S3. Almacenamiento de objetos creado para almacenar y recuperar cualquier volumen de datos desde cualquier ubicación. Recuperado de https://aws.amazon.com/es/s3/
- Autenticación con un grupo de usuarios. Recuperado de https://docs.aws.amazon.com/es_es/cognito/latest/developerguide/authentication.html
- AWS AppSync: Acelere el desarrollo de la aplicación con las API escalables de GraphQL. Recuperado de https://aws.amazon.com/es/appsync/
- AWS AppSync: Guía para desarrolladores de AWS AppSync. Recuperado de https://docs.aws.amazon.com/es_es/appsync/latest/devguide/appsync-dg.pdf
- Connect API and database to the app. Recuperado de https://docs.amplify.aws/start/getting-started/data-model/q/integration/angular
- Creación de aplicaciones con arquitectura sin servidor. Recuperado de https://aws.amazon.com/es/lambda/serverless-architectures-learn-more/
- File Access levels. Recuperado de https://docs.amplify.aws/lib/storage/configureaccess/q/platform/js
- Introducción a AWS AppSync. Recuperado de https://aws.amazon.com/es/appsync/
- Introduction to services and dependency injection. Recuperado de https://angular.io/guide/architecture-services
- Lima, E. (2019, Agosto). Building progressive web apps with the Amplify Framework and AWS AppSync. *AWS blogs*. Recuperado de https://aws.amazon.com/es/blogs/mobile/building-progressive-web-apps-with-the-amplify-framework-and-aws-appsync/
- Momentjs Documentation. Recuperado de https://momentjs.com/docs/
- OpenID (2021). Recuperado de https://en.wikipedia.org/wiki/OpenID_Connect

- Pressman, R. (2010). *Ingeniería del Software: Un Enfoque Práctico*. México: McGraw Hill 7ed.
- ¿Qué es NoSQL? Bases de datos no relaciones con excelente rendimiento y modelos de datos flexibles. Recuperado de https://aws.amazon.com/es/nosql/
- Ramalingam, S. K. (2019, Febrero). Angular 7 upload file to Amazon S3 Bucket. *Medium*. Recuperado de https://medium.com/ramsatt/angular-7-upload-file-to-amazon-s3-bucket-ba27022bad54
- Sans, G. (2019, Julio). How to create a highly scalable serverless GraphQL data-driven. *Medium*. Recuperado de https://gerard-sans.medium.com/how-to-create-a-highly-scalable-serverless-graphql-data-driven-app-in-minutes-57aca6264855
- Social sign-in (OAuth). Recuperado de https://docs.amplify.aws/lib/auth/social/q/platform/js
- Sommerville, I. (2005). *Ingeniería del Software*. España: Pearson 7ed.
- Visualización de fotos en un bucket de Amazon S3 desde un navegador. Recuperado de https://docs.aws.amazon.com/es_es/sdk-for-javascript/v2/developer-guide/s3-example-photos-view.html