UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

"APLICACIÓN ANDROID PARA LA EDICIÓN Y FIRMA DIGITAL AVANZADA DE DOCUMENTOS"

TRABAJO FIN DE GRADO

Febrero - 2021

AUTOR: Carlos Daniel López Pastor

DIRECTOR/ES: Miguel Onofre Martínez Rach

César Fernández Peris

AGRADECIMIENTOS

Agradecer a mi familia el apoyo incondicional que he tenido de su parte durante todos estos años.

Agradecer a Miguel Onofre Martínez Rach y César Fernández Peris por guiarme en el desarrollo de este proyecto. También a todos los profesores con los que he coincidido a lo largo de la carrera universitaria en la Universidad Miguel Hernández.

RESUMEN

Este informe recoge la documentación del proyecto que se presenta como TFG (Trabajo se Fin de Grado) del Grado en Ingeniería de Tecnologías de Telecomunicación de la Universidad Miguel Hernández de Elche.

El proyecto surge como propuesta del profesor Miguel Onofre Martínez Rach durante las clases de su asignatura Aplicaciones Móviles Multimedia del curso 2019-2020.

Se propuso el desarrollo de una aplicación móvil que ayude a los usuarios a generar documentos utilizando información de su entorno. La idea no es redactar, sino utilizar unos documentos preestablecidos que tienen una serie de campos a rellenar. La información del tipo fecha, hora, localización, etc. podrán recogerse de manera automática y se facilitará la recogida de otro tipo de datos como imágenes y firma para agregarlas al documento.

Para que los usuarios de la aplicación puedan tener a su disposición un gran número de documentos sin necesidad de almacenarlos en el dispositivo, se incluye en el proyecto el diseño de un servidor web que almacene los documentos y se comunique con la aplicación Android. De este modo la aplicación móvil se encargará de visualizar el documento y de ofrecer una interfaz al usuario para completar el documento obteniendo estos documentos del servidor. El proyecto, por lo tanto, se puede dividir en la aplicación Android y el servidor web, los cuales trabajan conjuntamente y se han diseñado en paralelo.

Está enfocado en su aplicación en sectores como el inmobiliario, transporte, administración, facturación, etc. En estos sectores se necesita constantemente generar documentos como contratos, acuerdos o permisos que recogen la información y firma de los clientes. Para ofrecer a los usuarios garantía sobre la validez legal de los documentos generados, se utilizará una firma digital avanzada con confirmación por email.

A pesar de esto, no se pretende desarrollar un producto comercial, sino un prototipo funcional que implemente las funcionalidades requeridas en la propuesta de proyecto. Implementar la aplicación móvil y el servidor serán los objetivos del proyecto junto con el estudio de las herramientas de desarrollo utilizadas.

Palabras clave

La siguiente lista contiene palabras descriptivas que podrán utilizarse para catalogar el proyecto.

- Smartphone
- Aplicación móvil
- Android
- Laravel
- API Rest
- OAuth2
- Geocodificación
- HTML
- PHP

ÍNDICE

Índice de ilustraciones

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Imagínese que fuera de su ubicación habitual de trabajo necesita rellenar una serie de documentos con información sobre datos de terceros junto con su autorización en forma de firma. Estos documentos podrían necesitar también información sobre la dirección actual, fecha y hora o podría requerir de alguna imagen o los datos de contacto del usuario.

Esta idea es la base del presente proyecto, que tiene como pretensión poder facilitar la recogida de datos en cualquier lugar y generar documentos con ellos de manera instantánea con posibilidad de almacenarlos o compartirlos a través de una aplicación móvil.

1.1 Objetivos

Este proyecto, en el que se desarrolla una aplicación móvil que dé solución al escenario planteado, surge como propuesta de TFG por parte del tutor. En esta propuesta se especificaba una serie de requisitos que debía cumplir la aplicación, por lo que el objetivo principal del proyecto sería diseñar dicha aplicación. A pesar de esto, no es objetivo de este trabajo desarrollar una aplicación para su comercialización, aunque se ha pretendido diseñar una aplicación con aspecto profesional y es posible que se continúe el proyecto más adelante. No será objetivo del trabajo la publicación de la aplicación en el portal de Google u otros portales de aplicaciones.

A parte del objetivo principal de diseño de la aplicación, se han perseguido una serie de objetivos académicos y personales. Se ponen a prueba los conocimientos adquiridos a lo largo del grado universitario y el estudio independiente de las diferentes herramientas y servicios.

Lista de objetivos:

- Diseño de una aplicación móvil que cumpla los requisitos propuestos por el tutor, funcional, robusta y con un aspecto profesional. Esta aplicación pretende facilitar a los usuarios la recopilación de datos para generar posteriormente un documento en el que se reflejen todos los datos con el formato adecuado junto con la ubicación, fecha y firma y que le permita dar una confirmación de esa firma. La descripción detallada de la aplicación y sus requisitos se detallan en los siguientes capítulos.
- Diseño de un servidor web que aumente las capacidades y funcionalidades de la aplicación. Deberá almacenar los documentos y permitir el acceso de la aplicación para descargarlos.

- Poner en práctica los conocimientos de desarrollo en Android, mejorar la eficiencia de trabajo con el entorno de desarrollo Android y obtención de nuevas destrezas.
- Iniciación en el entorno de diseño web de Laravel, poniendo en práctica los conocimientos de PHP.
- Utilización de conocimientos adquiridos sobre otras tecnologías, protocolos y lenguajes que complementan la aplicación móvil.
- Afrontar un proyecto completo hasta completarlo y resolver todas las etapas de su desarrollo desde su análisis hasta su conclusión, gestionando cambios y resolviendo incidencias.

1.2 Contexto de desarrollo

Los dispositivos móviles y tabletas son una buena plataforma sobre la que implementar este tipo de proyectos, ya que reúnen una serie de características que favorecen a que los desarrolladores puedan crear nuevas herramientas. Algunas de estas características aplicadas al planteamiento inicial de este proyecto son:

- Capacidad de procesamiento y almacenamiento: Un dispositivo móvil posee potencia suficiente para procesar documentos o memoria para almacenarlos. Los desarrolladores tienen más recursos para utilizar en las aplicaciones.
- Movilidad: Una de las principales características de estos dispositivos. Sin perder capacidad es capaz de llevar a cabo tareas de gestión e incluso redacción o dibujo avanzado en el caso de las tabletas.
- Conexión a internet: Característica indispensable en estos dispositivos ya sea utilizando una tarifa de datos móviles o a través de las diferentes versiones WIFI[1]. Esto hace que los desarrolladores sean capaces de implementar aplicaciones móviles muy potentes haciendo uso de servidores que reciben la mayor parte de la carga computacional y de almacenamiento. Aplicaciones para gestión de cuentas bancarias, redes sociales, buscadores y un gran número de aplicaciones de cualquier categoría utilizan este recurso.
- Entrada de datos: Los dispositivos móviles y tabletas son capaces de recoger distintos tipos de datos gracias a las distintas tecnologías que incorpora. Antenas para conexión y localización, sensores que recogen la actividad física, cámara, micrófono o pantalla táctil.
- Mercado de los dispositivos móviles: Los dispositivos móviles ha tenido un crecimiento de la demanda a nivel mundial en las últimas décadas. Esto ha

[1] WIFI es una tecnología que permite la interconexión inalámbrica de dispositivos electrónicos. Existen distintas versiones incluidas en la familia de estándares IEEE 802.11. Trabaja en las bandas de 2,4Ghz o 5GHz según la versión.

propiciado un ecosistema favorable para cualquier desarrollador que tiene a su disposición un gran número de herramientas y servicios.

Los smartphones se están convirtiendo en parte fundamental de la vida diaria de muchas personas en todo el mundo. El número de usuarios ya ha alcanzado los 3.000 millones y solo se espera que esta cifra aumente en los próximos años. China, India y Estados Unidos son los países con mayor número de usuarios. La Ilustración 1 muestra la evolución en el número de usuarios de los últimos años en un gráfico publicado en el portal Statista [1].

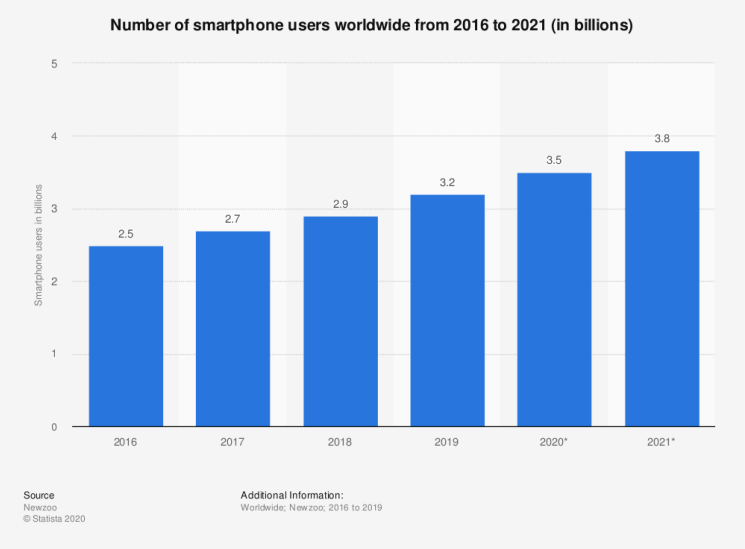

ILUSTRACIÓN 1. EVOLUCIÓN DEL NÚMERO DE USUARIOS DE SMARTPHONE

Existe un gran número de fabricantes y modelos en el mercado de smartphones. Según StatCounter [2], las principales marcas que se comercializan en España son Samsung, Xiaomi, Apple y Huawei, el resto de las marcas supondrían un 11,4% del mercado en el que se encuentran LG y BQ.

En principio la variedad de dispositivos podría ser un inconveniente a la hora de desarrollar aplicaciones, pero no es el caso ya que prácticamente la totalidad de los dispositivos se basan en un par de sistemas operativos que acaparan el mercado. Estos son Android y IOS. Mientras que IOS es el sistema operativo de los dispositivos móviles y tabletas de Apple, Android está instalado en la mayoría de los dispositivos móviles del resto de fabricante alcanzando un 70% de la cuota de mercado. Esto beneficia a los desarrolladores que disponen de un entorno de desarrollo estable y compatible con la mayoría de los dispositivos en el mercado. En la Ilustración 2 se muestra una comparativa de los principales sistemas operativos para dispositivos móviles. [3]

Android, compañía adquirida en 2005 por Google, ofrece el sistema operativo móvil basado en el Kernel de Linux[2]. Es el más extendido, debido principalmente a su sencillez, compatibilidad con un gran número de dispositivos móviles diferentes y distribución como SO libre. Google Play Store es el Servicio de distribución digital que permite a los usuarios de dispositivos móviles Android descargar aplicaciones.

[2] El Kernel de Linux es la interfaz fundamental entre el hardware de una computadora y el software. Se encuentra dentro del sistema operativo y controla todas las funciones principales del hardware. Gestiona memoria, procesos, periféricos, seguridad y llamadas al sistema. Se trata de un software libre y de código abierto similar al núcleo de Unix.

Según algunos portales web, la cantidad de aplicaciones Android en Google Play Store es superior a 3 millones de las cuales un 96,2% son aplicaciones gratuitas. Se agregan 3739 aplicaciones a Play Store todos los días y se estima que la cantidad de descargas por día es de 250 millones. Google Play Store se presenta como una plataforma para que los desarrolladores de aplicaciones de Android suban sus proyectos y obtengan un mayor alcance de los clientes. [4], [5]

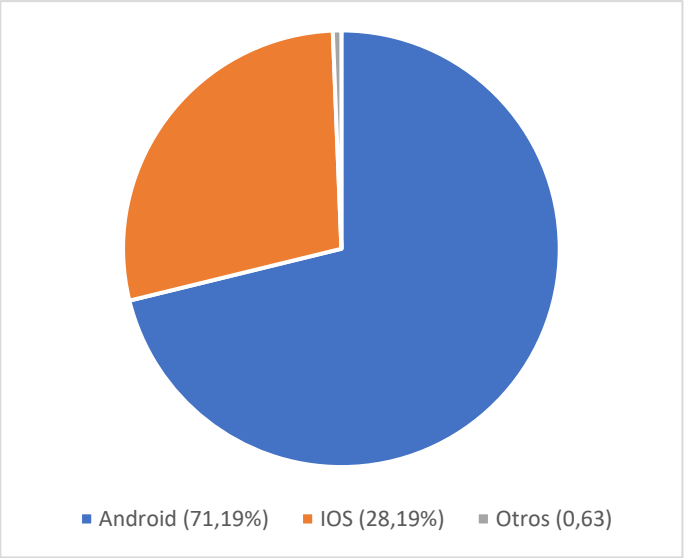

ILUSTRACIÓN 2. COMPARATIVA DE SISTEMAS OPERATIVOS MÓVILES

1.3 Descripción de la aplicación

En este proyecto se ha diseñado una aplicación móvil para el sistema operativo Android junto con un servidor HTTP[3] que trabaja con la aplicación.

La aplicación Android servirá de interfaz para que los usuarios puedan utilizar una plataforma de edición de documentos. Dispone de varias pantallas por las que se podrá navegar para utilizar las diferentes funcionalidades. La aplicación permitirá realizar una búsqueda de los documentos, editarlos una vez seleccionados, previsualizar su contenido y, por último, permite exportarlos en formato PDF para almacenarlos en la memoria del dispositivo o compartirlos.

Para la edición de los documentos y el resto de las funcionalidades se pone a disposición de la interfaz de usuario todos los recursos que posee el dispositivo móvil. La recogida de datos se efectuará a través de las entradas del dispositivo como el teclado, pantalla táctil, antenas y sensores o cámara y a través de los diferentes servicios que ofrecen las plataformas como Google. Con esto se pretende facilitar y automatizar la edición de los documentos con los diferentes tipos de datos: Fecha y hora, localización, imagen, firma, datos de contacto, etc.

La implementación del servidor web incrementa las funcionalidades de la aplicación Android. Sin este servidor, el dispositivo debería reservar parte de su memoria para almacenar los documentos en una lista estática que podría requerir actualizaciones

[3] "HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor" (MDN Web Docs) [6]

cada cierto tiempo. Por lo tanto, la aplicación dispondrá de una lista actualizada de documentos tan extensa como permita el servidor, de la que puede extraer información adicional, y con el mínimo gasto de almacenamiento en el dispositivo. Es habitual, por parte de los usuarios, que la mayor parte del almacenamiento esté dedicado a contenido multimedia como imágenes, música o aplicaciones de entretenimiento. El modelo cliente-servidor en dispositivos móviles permite desarrollar aplicaciones muy potentes y minimiza los recursos del dispositivo utilizados, ya que tanto almacenamiento como procesamiento pueden tener lugar en el lado del servidor.

Para el diseño de las dos partes, servidor web y aplicación Android, se ha seguido fundamentalmente la documentación de ambas plataformas en sus portales oficiales para desarrolladores. [7], [8]

En la Ilustración 3 se muestra un esquema que representa la relación entre la aplicación y el servidor. [9]

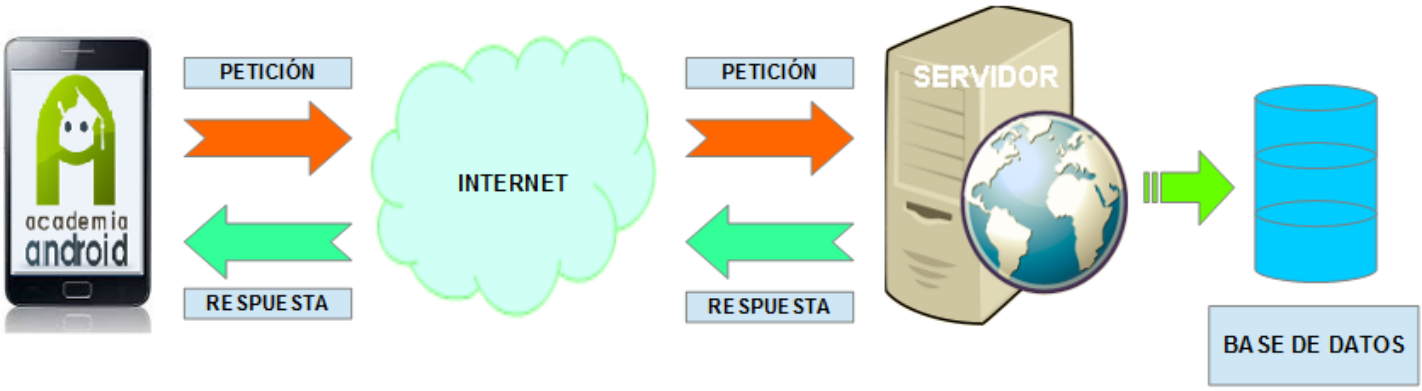

ILUSTRACIÓN 3. MODELO CLIENTE-SERVIDOR

Los documentos gestionados por la aplicación pueden considerarse plantillas que se deben rellenar con los datos que se recogen en la aplicación. Poseen una estructura especifica de la aplicación que se ha diseñado para poder ser gestionados tanto por el servidor como por la aplicación. El texto y formato se almacena en un archivo que será almacenado en el servidor y contendrá los campos a rellenar identificados mediante etiquetas. Los documentos se referencian en una entidad[4] almacenada en la base de datos del servidor. La entidad posee información para acceder al texto, contiene la información acerca de los campos del documento y otros datos como fecha de

[4] Cada una de las filas en la tabla de la base de datos representa una entidad. IBM lo define como: "Objeto exclusivo único en el mundo real que se está controlando. Algunos ejemplos de entidad son una sola persona, un solo producto o una sola organización." (IBM Knowledge Center) [10]

creación y edición o el usuario que subió el documento. La estructura y formato de los documentos se describe y justifica en el capítulo de diseño.

Mientras que el texto se utiliza exclusivamente para previsualizar y exportar los documentos, la entidad solicitada a la base de datos será utilizada por la aplicación para gestionar el documento. La información de los campos se utiliza para generar dinámicamente la interfaz de edición o para rellenar los campos del texto en la previsualización.

Los documentos generados se deben poder firmar a través de algún método seguro y que garantice su validez jurídica. Para ello se implementará el concepto de firma digital avanzada recogido en el Reglamento (UE) Nº 910/2014[5]. Una vez se confirmen los datos introducidos en el documento y recogida la firma a través de la firma biométrica (o manuscrita), se asociará la firma con los datos del cliente y se mandará un email de confirmación con el que se finalizará el proceso de validación de la firma digital avanzada.

Uno de los aspectos que no se ha concretado es el ámbito de trabajo del proyecto. Se ha aclarado anteriormente que este proyecto no se plantea como un producto comercial, aunque se puede aplicar a ciertos sectores y podría ser un futuro objetivo su comercialización. En el ámbito privado del usuario, puede ser muy útil también, nos puede evitar tener que rellenar manualmente algunos documentos administrativos. El servidor podría tener diferentes características en función del producto final que se quiera desarrollar. Puede ser un servidor privado para uso exclusivo de la plantilla de una empresa o puede ser un servidor público a modo de banco de documentos y basado en subscripciones. A pesar de todo, no se ha tenido en cuenta este aspecto, justificado en el capítulo de objetivos. Los requisitos del servidor y de la aplicación se detallan más adelante.

[5] "En agosto de 2014 el Parlamento Europeo aprobó un nuevo marco regulatorio para los servicios de identificación y confianza de las transacciones electrónicas en el mercado interior: el Reglamento (UE) Nº 910/2014, que finalmente entró completamente en vigor el 1 de julio de 2016, derogando la directiva anterior." [11]

Capítulo 2. HERRAMIENTAS DE DESARROLLO

2.1 Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. La primera versión se publicó en diciembre de 2014 de forma gratuita a través de la Licencia Apache 2.0. Ha sido diseñado específicamente para el desarrollo de Android y está disponible para Microsoft Windows, macOS y GNU/Linux. En agosto de 2020 se actualizó a la versión 4.1. [12]

ILUSTRACIÓN 3. LOGOTIPO DE ANDROID STUDIO

Las herramientas que ofrece Android Studio facilitan el desarrollo de aplicaciones. Cuenta con un editor de diseño visual, editor de código optimizado para la plataforma, soporte para emulación y sistema de compilación flexible, entre otras características. Los dos principales lenguajes de programación son Kotlin y Java.

El sistema de compilación de Android compila recursos y código fuente de la App y los empaqueta en APK (Android Application Package) que puedes probar, implementar, firmar y distribuir. Está basado en Gradle[6], y agrega varias funciones que son específicas de la compilación de Apps para Android. A pesar de que el complemento para Android se suele utilizar a través del IDE[7], el sistema Gradle se pueden ejecutar independientemente de Android Studio y actualizar por separado. [13]

El SDK (Software Development Kit) de Android incorpora los paquetes necesarios para desarrollar las aplicaciones en las diferentes versiones del sistema operativo. Para utilizar las funciones de la plataforma el SDK debe tener instalado el nivel de API[8] correspondiente. Las API se clasifican por niveles, siendo las de mayor nivel las que implementan las funciones de las versiones más recientes.

Utilizar el nivel de API más alto implica tener acceso a las funciones más modernas de los dispositivos Android, mientras que las API de nivel bajo utilizan funciones más básicas pero compatibles con versiones superiores. El problema es que no todos los dispositivos se actualizan a la última versión haciendo que muchos dispositivos dispongan de distintas versiones de Android. Por este motivo, se debe conocer la

[6] Gradle es una herramienta de compilación que se centra en la automatización de compilación y el soporte para el desarrollo en varios lenguajes. [14]
[7] Entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE).
[8] Interfaz de programación de aplicaciones, en inglés application programming interface. Interfaz con la que un software es capaz de utilizar funciones de otro software o servicio.

distribución de las distintas versiones y hacer un estudio previo del alcance que se espera de la aplicación junto con las funciones Android que debe implementar.

En la Tabla 1 se muestra las últimas versiones de Android y el nivel de API correspondiente [15], mientras que en la Ilustración 5 se muestra la distribución de las versiones Android [16].

TABLA 1. VERSIONES ANDROID

Nombre Interno	Versión	Nivel de API
Android 11	11	30
Android 10	10	29
Pie	9	28
Oreo	8.0.0, 8.1.0	26, 27
Nougat	7.0, 7.1	24, 25
Marshmallow	6	23
Lollipop	5.0, 5.1	21, 22
KitKat	4.4 – 4.4.4	19, 20
Jelly Bean	4.3.x	18

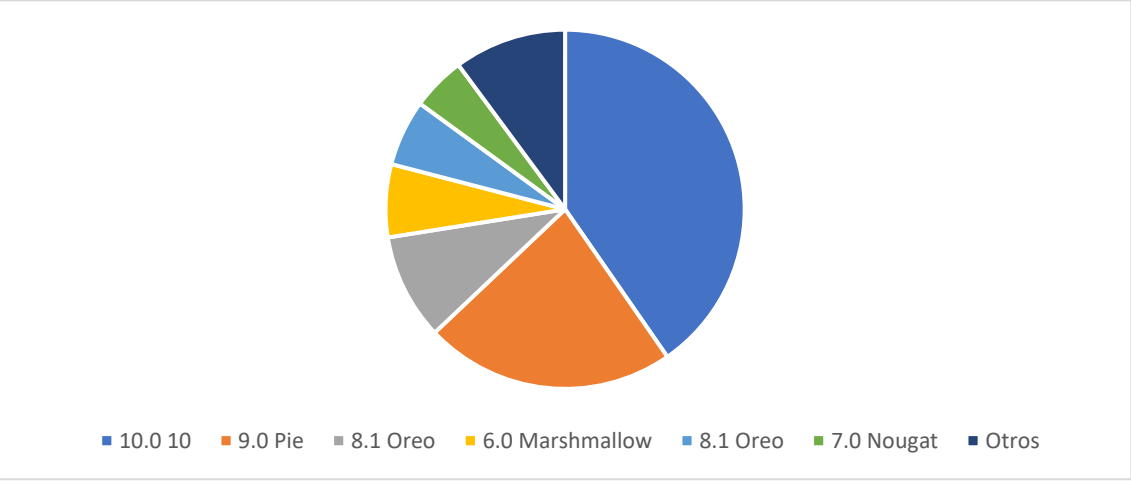

ILUSTRACIÓN 5. DISTRIBUCIÓN DE VERSIONES ANDROID

2.2 Laravel

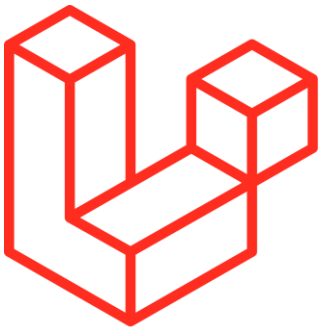

ILUSTRACIÓN 4. LOGOTIPO DE LARAVEL

Laravel es uno de los entornos de desarrollo en PHP para web más utilizados. Desarrollado por Taylor Otwell, lanzó su primera versión beta en junio de 2011. Ofrece utilidades potentes a los desarrolladores, que permiten agilizar el desarrollo de las aplicaciones web. Algunas de sus características son la calidad del código y sintaxis elegante, la facilidad de mantenimiento y escalabilidad, lo que permite realizar proyectos de forma sencilla. El marco de Laravel es un software de código abierto con licencia MIT[9] que permite utilizarlo tanto para desarrollar software libre como para ser software no libre. [17], [22]

Laravel tiene una estructura modular y extensible, y posee un poderoso conjunto de librerías que permite desarrollar servicios y APIs de gran rendimiento. Proporciona una serie de funcionalidades que facilitan el desarrollo web como el control de enrutamiento, autenticación y autorización de usuarios, gestión de sesiones o instalaciones de correo electrónico. Su sistema de abstracción de base de datos ORM[10] llamado Eloquent permite trabajar con la información de la base de datos como si fueran objetos y Blade, su motor de plantillas, compila las vistas en código PHP simple y las almacena en caché. Todos estos recursos se pueden gestionar a través de Artisan, un potente sistema de comandos de consola. [18], [19]

Actualmente, Laravel admite cuatro bases de datos: MySQL 5.6+, PostgreSQL 9.4+, SQLite 3.8.8+ y SQL Server 2017+. Para trabajar con Laravel es necesario tener una versión actualizada de PHP instalada en el servidor, la última actualización Laravel 8.x requiere PHP 7.3 o una versión superior. Laravel utiliza Composer, el sistema de administración de dependencias PHP, para la gestión de paquetes. [20], [21]

2.3 XAMPP

XAMPP es uno de los entornos más populares de desarrollo web. Es una distribución de Apache completamente gratuita y fácil de instalar que incluye varias herramientas de software libre como los sistemas de gestión de bases de datos MySQL y MariaDB, y los intérpretes para lenguajes de script PHP y Perl. [23]

[9] La licencia MIT es una de tantas licencias de software que ha empleado el Instituto Tecnológico de Massachusetts(MIT, Massachusetts Institute of Technology).
[10] El mapeo relacional de objetos (ORM) es un mecanismo para vincular clases de un lenguaje de programación orientado a objetos a tablas de un sistema de administración de bases de datos relacionales.

Se utilizará principalmente para instalar el servidor web Apache, la base de datos MySQL y la versión PHP necesarios para implementar el proyecto Laravel. La facilidad de instalación permite centrar el desarrollo del proyecto en el comportamiento del servidor a través de Laravel, aunque ofrece herramientas interesantes como phpMyAdmin, utilizada para gestionar la base de datos del proyecto.

A pesar de las ventajas que ofrece XAMPP, no se recomienda su uso para servidores públicos ya que presenta limitaciones y fallos de seguridad en pro de facilitar su instalación y gestión, por lo que este proyecto solo contempla utilizar este entorno para pruebas y en un futuro se estudiará la instalación de un nuevo servidor si se quiere lanzar la aplicación al mercado.

2.4 Lenguajes de programación.

Para el desarrollo del proyecto se han utilizado una serie de lenguajes y formatos de datos para implementar el comportamiento de la aplicación y para definir la estructura de alguno de sus componentes. A continuación, se hace un repaso de estos lenguajes y de su papel dentro del proyecto

2.4.1 Java

Para la implementación de la aplicación móvil se ha utilizado Java. Es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Java es un lenguaje de programación orientado a objetos utilizado para el desarrollo de aplicaciones. Su intención es permitir que los desarrolladores ejecutar en cualquier dispositivo sus aplicaciones. El código se compila una única vez y necesita una máquina virtual de Java para su ejecución. Java proporciona una gran biblioteca estándar y herramientas para que los programas puedan ser distribuidos.

Java es el idioma oficial para el desarrollo de Android junto a Kotlin. Gran parte de Android está escrito en Java y sus API están diseñadas para ser llamadas principalmente desde este lenguaje. Es posible desarrollar aplicaciones C y C ++ utilizando el kit de desarrollo nativo de Android (NDK), sin embargo, no es algo que Google promueva.

2.4.2 PHP

PHP (Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web dinámicas con acceso a información almacenada en una base de datos. Se trata de un lenguaje pensado para ejecutarse en el lado del servidor, que dispondrá de un intérprete PHP para ejecutará el código. Es, por lo tanto, un lenguaje interpretado, multiplataforma y orientado a objetos. [24]

2.4.3 SQL

SQL (por sus siglas en inglés Structured Query Language) es un lenguaje diseñado para administrar y recuperar información de sistemas de gestión de bases de datos relacionales. Permite efectuar consultas para recuperar, de forma sencilla, información de bases de datos, así como realizar cambios en ellas. SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos.

Aunque Laravel incluya herramientas de gestión de base de datos, será necesario conocer este lenguaje por si es necesario manipular las tablas manualmente.

2.4.4 HTML

HTML, siglas en inglés de HyperText Markup Language, es un lenguaje de marcado para creación páginas web. Es un estándar a cargo del World Wide Web Consortium (W3C). A través de HTML se define la estructura y contenido de una página web.

Se utilizará este lenguaje para definir la estructura de los documentos que utilizará la aplicación, ya que permite implementar una serie de funcionalidades necesarias utilizando las herramientas de Android y Laravel.

2.4.5 JSON

JSON (acrónimo de JavaScript Object Notation) es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript. Es una alternativa a XML y se utilizará para el intercambio de información entre el servidor y la aplicación. Definirá la estructura de las respuestas del servidor y los recursos almacenados en la base de datos.

2.4.6 XML

XML es el acrónimo de Extensible Markup Language. Mientras que HTML fue diseñado para presentación de los datos, XML es específico de datos y se suele utilizar para definir objetos.

Para la mayoría de los objetos y diseños que necesitaremos definir en Android Studio, se utiliza el lenguaje XML. Android proporciona un vocabulario XML simple que coincide con las clases y subclases de las vistas, como las que se usan para widgets y diseños. También es posible utilizar el editor de Android Studio para crear el diseño XML mediante una interfaz de arrastrar y soltar. En Android, la mayoría de los recursos se codifican en XML, mientras que su comportamiento se define en Java o Kotlin.

Capítulo 3. ANÁLISIS

Capítulo fundamental para entender la aplicación y de suma importancia para el diseño del proyecto, ya que sobre estos requerimientos se efectúa el diseño de la aplicación.

Aunque el análisis sea una etapa previa al diseño, conocer las herramientas de diseño y metodología ayuda a analizar la aplicación y evitar futuros errores de diseño graves.

3.1 Catálogo de requisitos

En este punto se recoge el comportamiento que se espera de la aplicación en forma de listado. El listado incluye los requisitos que el tutor del TFG describió en su propuesta y una serie de requisitos que incluí.

Se introducirá en primer lugar los requisitos del servidor web y posteriormente la aplicación para entender mejor la relación cliente-servidor.

3.1.1 Requisitos del servidor

- **Almacenamiento de documentos**: El servidor deberá almacenar todos los archivos relacionados con los documentos: Texto, imágenes, campos.
- **Base de datos**: Los documentos tienen que estar referenciados en una base de datos que permita realizar búsquedas y gestionar los documentos.
- **Crear documento**: El servidor incorporará una ruta web que muestre un formulario mediante el que poder subir los archivos al servidor, que se seleccionarán a través del explorador de archivos de nuestro pc o dispositivo móvil. En un futuro se pretende diseñar un editor de texto para redactar el documento directamente en el portal web.
- **Búsqueda de documentos**: El servidor debe poder devolver un listado de los documentos sin necesidad de incluir el texto, pero indicando ciertos elementos como el título, fecha de última modificación y campos.
- **Descarga de documentos**: Desde el servidor se deben poder descargar toda la información referente a los documentos como texto e imágenes.
- **Autenticación**: No se puede acceder a las funciones del servidor si no se ha autenticado el cliente que accede a él. En este caso el cliente puede ser la aplicación Android o el navegador web a través del que el usuario accede.
- **Gestión de usuarios**: El servidor debe ofrecer una gestión de usuarios que incluya las siguientes funciones: registro, inicio de sesión y cierre de sesión.

- **Validación de firma**: Una vez firmado el documento, la aplicación podrá subir la información al servidor para generar un enlace de confirmación que se enviará por email. El enlace de confirmación redirigirá a una ruta del servidor que registrará la confirmación dando validez a la firma. Esta implementación ofrecerá una firma de categoría firma digital avanzada a la aplicación.

3.1.2 Requisitos de la aplicación

Gestión de usuarios

- **Registro**: La aplicación tiene una pantalla desde la que poder registrarse en el servidor y poder acceder a los documentos
- **Inicio de sesión**: También tendrá una pantalla de inicio de sesión en el caso de que el usuario ya se haya registrado.
- **Cierre de sesión**: Se podrá cerrar la sesión una vez el usuario se encuentre en una sesión activa.
- **Mantener la sesión**: La aplicación mantendrá la sesión en el caso de que se cierre la aplicación a no ser que se haya cerrado la sesión previamente.

Exploración de documentos

- **Búsqueda de documentos**: La aplicación debe mostrar un listado de los documentos seleccionables en la pantalla, donde cada elemento contiene información de interés: título, fecha de modificación y código de identificación.
- **Carga dinámica**: Las solicitudes al servidor no devuelven un listado muy grande para no incrementar los tiempos de carga, pero la aplicación debe acceder al servidor para mostrar más documentos a medida que nos desplazamos por los elementos de la lista.
- **Búsqueda por nombre**: Posibilidad de realizar la búsqueda por nombre.

Entrada de datos

- **Interfaz**: La aplicación mostrará una interfaz accesible, en la que introducir todos los datos que necesite el documento y acompañada de información adicional sobre los campos. La interfaz se genera de forma dinámica en función de los campos y el tipo de datos.
- **Texto**: Introducir texto a través del teclado.
- **Fecha y hora**: Introducir fecha y hora a través de un calendario y un reloj con la fecha y hora actual por defecto.

- **Localización**: Acceder a la localización del dispositivo para ofrecer sugerencias sobre la dirección en la que se encuentra el usuario. También se puede introducir con el teclado.
- **Firma**: Mostrar una pantalla que recoja la firma de una persona. La persona a la que se le recoja la firma tendrá que escribirla en la pantalla con el dedo o un lápiz táctil.

Exportar y compartir

- **Guardar en PDF**: Acceder a la memoria del dispositivo para guardar el documento resultante en PDF.
- **Guardar en JPEG**: Guardar el documento en formato JPEG.
- **Enviar por email**: Adjuntar el documento en PDF a un email.

Flujo de la aplicación

- **Inicio**: Una vez se ha iniciado la aplicación, iniciará sesión en caso de que no se haya cerrado la sesión con anterioridad. Si no existe ninguna sesión se solicitará el usuario y la contraseña o el registro de un nuevo usuario.
- **Pantalla principal**: Desde esta pantalla se accede a la selección de formularios. No se le ha dado otro uso, pero se utilizará en futuras versiones de la aplicación.
- **Pantalla de selección**: En esta pantalla se efectuará la exploración de documentos. Se podrá seleccionar un documento solo con tocarlo en la lista y ver su contenido pulsando el icono de previsualización. Una vez seleccionado se introducirán los datos en otra pantalla.
- **Pantalla de edición**: En esta pantalla se editarán los documentos. Mostrará la interfaz que recoge los datos.
- **Previsualización**: Se mostrará el documento seleccionado en la pantalla. Si se han introducido los datos, el documento los reflejará. También dará la opción de firmar el documento si todos los campos se han rellenado.
- **Barra lateral**: La aplicación dispondrá de una barra lateral con la que poder navegar por las distintas pantallas.

Otros requisitos

- **Ayuda**: Se accederá a una sección diferente de la ayuda en función de en qué pantalla se haya seleccionado, mostrando información relacionada.
- **Ajustes**: La aplicación dispone de una pantalla de ajustes.
- **Tema de la aplicación**: La aplicación dispone de dos temas distintos que cambiaran el estilo de la aplicación.

4.2 Estructura del documento

Se debe diseñar una estructura para los documentos que permita su gestión y edición de forma sencilla por parte de la aplicación. La aplicación distinguirá dos elementos del documento que formarán su estructura, el texto y los campos. Cada elemento se utiliza por separado en la aplicación, el texto se utiliza en la previsualización y la exportación, mientras que los campos se utilizan a nivel interno en la aplicación para generar la interfaz de introducción de datos y modificar el texto del documento.

De este modo, los campos deben indicar la siguiente información:

- Posición en el texto o identificador. Para conocer la posición en el texto del campo a rellenar se necesitará un identificador.
- Tipo de datos.
- Información sobre el campo o ayuda.
- Datos recogidos por la aplicación. Estos datos son los que se verán reflejados en el texto.

Por otro lado, el texto del documento debe tener una estructura que le permita ser manipulado y renderizado por la aplicación sin que se requiera una carga computacional elevada. El texto debe identificar los lugares en los que se introduce la información con alguna etiqueta y esta debe ser encontrada y manipulada fácilmente por la aplicación.

La solución que se propone en el proyecto es la utilización de HTML para definir el texto y su formato. Nos da la posibilidad de incluir estilos, fácil de manipular si se conoce el lenguaje, nos da la posibilidad de crear nuevas etiquetas y existen muchos métodos para manipular y renderizar archivos de este tipo en Android.

A pesar de esto, existen inconvenientes a la hora de utilizar HTML. Los usuarios no tienen por qué conocer el lenguaje y puede dificultar su redacción y subida al servidor. Se puede solucionar a través de alguna herramienta de conversión de archivos o implementando un editor de texto en el propio portal web del servidor, pero el formato Word que utiliza la mayor parte de los usuarios de editores de texto no es compatible con la aplicación por el momento.

Capítulo 4. DISEÑO

Una vez finalizada la etapa de análisis, podemos hacer una primera estructuración del proyecto distinguiendo cada uno de los elementos de mayor abstracción[11] que lo componen y estableciendo sus relaciones. Existen tres elementos fundamentales:

- **Documento**: Es una entidad independiente que es utilizada por el servidor y la aplicación.
- **Servidor**: El servidor y su base de datos debe diseñarse para poder trabajar con los documentos. El servidor depende de la estructura del documento, pero es independiente de la estructura de la aplicación. A pesar de que no dependa de la aplicación, el servidor debe diseñar una API completa para que la aplicación pueda comunicarse.
- **Aplicación Android**: El diseño de la aplicación depende de la estructura del documento, las diferentes pantallas de la aplicación deben poder gestionar los documentos. También depende del servidor, en el diseño de la API se establece cómo se acceder al servidor y que respuestas deberá gestionar la aplicación.

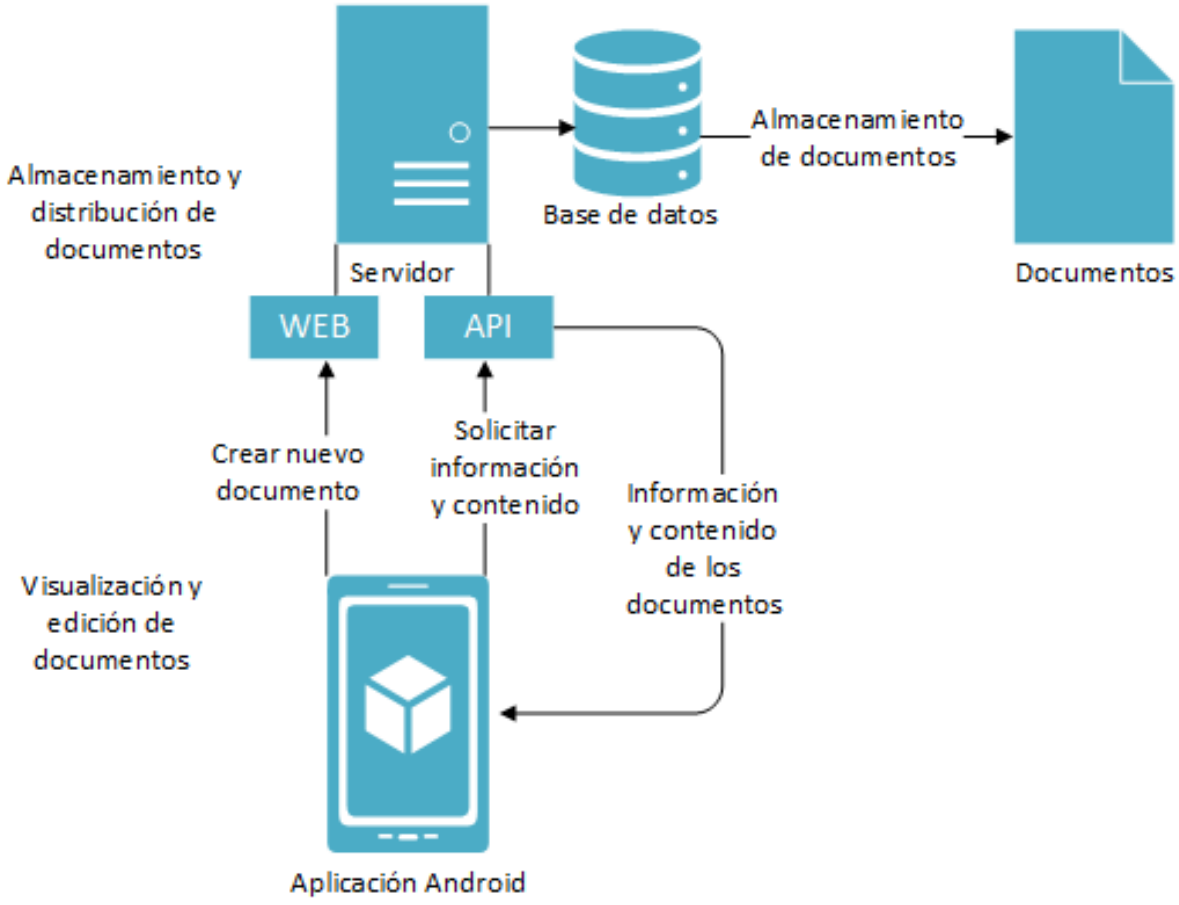

ILUSTRACIÓN 7. ESQUEMA DEL PROYECTO

[11] Una capa de abstracción (o nivel de abstracción) es una forma de ocultar los detalles de implementación de ciertas funcionalidades.

4.1 Documento

Como se ha ido introduciendo, el documento está compuesto de dos partes: texto y campos. El diseño de la estructura de los documentos depende de cómo la aplicación trabaja internamente. Se está trabajando con clases y métodos de la librería de Android para visualizar y exportar los documentos. En concreto, la clase utilizada trabaja con archivos de tipo HTML y es la solución más sencilla para implementar. Además, el formato HTML funciona a base de etiquetas y es muy útil para referenciar los campos a lo largo de todo el texto.

4.1.1 Texto del documento

Se trata de un archivo HTML que se almacena en algún directorio del servidor y en el que se redacta el texto del documento, se referencias las imágenes, los estilos, los formatos y se estructura en párrafos, listas, tablas, etc. Sigue la misma estructura que cualquier archivo HTML y las imágenes y estilos pueden almacenarse en archivos y directorios distintos, siempre y cuando estén bien referenciados en el texto y el directorio sea accesible por la aplicación.

Los campos se referencian en el texto a partir de las etiquetas características del lenguaje. Existen etiquetas propias del lenguaje, pero las etiquetas que referenciarán los campos se definirán a la hora de redactar el documento. Estas etiquetas se definen con un identificador único para cada campo y, aunque no tengan ninguna función estructural, son reconocidas por la aplicación móvil que las sustituirá por los datos correspondientes.

A continuación, se muestra cómo se referencia un campo en el archivo HTML:

```
<html>
<head> […] </head>
<body>
[…]<Identificador>_____________</Identificador>[…]
</body>
</html>
```

Se distinguen las etiquetas propias de HTML como <head> o <body>. Estas etiquetas se procesan en la aplicación Android para previsualizar de manera correcta el documento en la pantalla, mientras que la etiqueta que hemos definido nosotros como identificador no tiene ningún efecto sobre el contenido. Dentro del campo se debe especificar los símbolos o la cadena de texto que se desea que tenga como valor por defecto.

4.1.2 Campos

La aplicación necesita una forma de identificar los campos y las etiquetas en el archivo HTML y necesita también toda la información referente a los campos para generar la interfaz dinámicamente y con ella recoger los datos.

Un objeto JSON es idóneo para representar esta información y comunicarla fácilmente a través del servidor. En este caso será un Array JSON, una cadena que recoja todos los campos del documento en formato JSON. Para ello hay que enumerar la información de la que se componen:

- **Identificador**: El identificador coincide con el nombre que se le ha dado a la etiqueta HTML que contiene el campo en cuestión. De este modo la aplicación podrá encontrar y sustituir el texto.
- **Tipo**: Lo utiliza la aplicación a la hora de crear la interfaz de introducción de datos. Los tipos de datos que se pueden utilizar son los que están implementados en la aplicación.
- **Información o ayuda**: Un pequeño texto a modo de comentario que ayude al usuario. Esta información aparecerá en la interfaz de la aplicación.
- **Datos**: Almacena la información que ha recogido el usuario a través de la interfaz de la aplicación. Será el texto que aparezca en el documento.

Una vez recopilada la información que necesita un campo, la estructura del Array JSON sería la siguiente:

```
[{    "type": "text",
      "id": "Identificador1",
      "hint": "Nombre de persona",
      "data": ""},
 {    "type": "time",
      "id": "Identificador2",
      "hint": "Hora de llegada",
      "data": ""},[…]]
```

4.2 Servidor

Se ha utilizado la herramienta software XAMPP para la implementación del servidor, en una máquina virtual Debian GNU/Linux 10 de la universidad. Esto nos permitirá utilizar la máquina virtual como servidor público y probar la aplicación móvil desde cualquier lugar.

4.2.1 Base de datos

El servidor debe almacenar los documentos y referenciarlos en una base de datos para poder gestionarlos y realizar búsquedas eficaces. El método más habitual para diseñar bases de datos que trabajan con información de gran tamaño, como pueden ser archivos de texto o imágenes, es referenciar estos archivos en lugar de almacenarlos en los propios atributos de una tabla. El atributo que representa el archivo almacenará únicamente la dirección, ya sea con una URL[12] o con la ruta que lo identifica en el sistema de archivos del servidor.

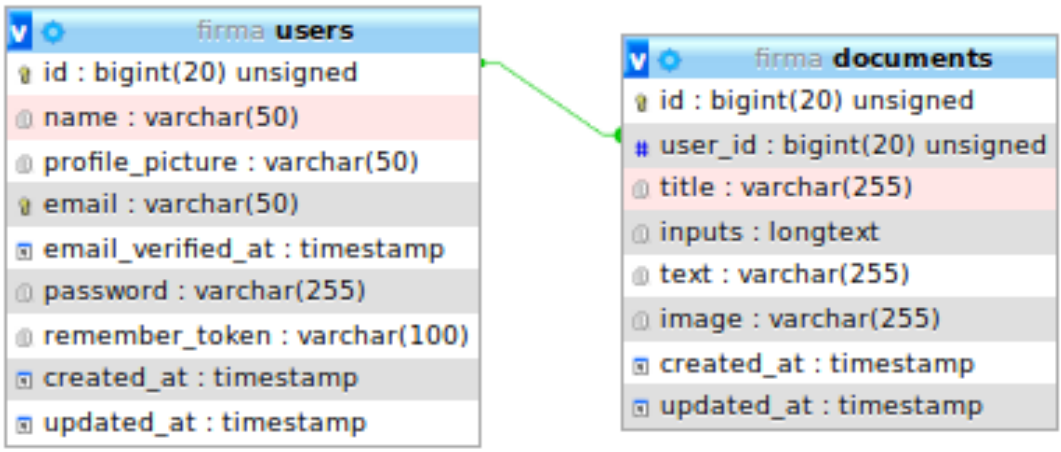

ILUSTRACIÓN 8. DIAGRAMA ENTIDAD RELACIÓN

La base de datos tiene un diseño sencillo, ya que la aplicación solo necesita acceder a la información de usuario y a los documentos. En la Ilustración 8 se muestra el diagrama entidad relación de la base de datos implementada en el servidor. Se ha generado a través de la aplicación phpMyAdmin. En esta ilustración se muestran las dos tablas principales que forman la base de datos del servidor, aunque existen más tablas que se generan al implementar las funciones de autenticación de Laravel.

Documents: La tabla documents es la que almacena los documentos que se utilizarán en la aplicación móvil. La clave primaria es un identificador de tipo entero con 20 dígitos al igual que en la tabla users, sus atributos son:

- **Id**: Entero de 20 dígitos que identifica el documento. Único para cada documento.

[12] URL son las siglas en inglés de Uniform Resource Locator, en español Localizador Uniforme de Recursos. El URL es la dirección específica que se asigna a cada uno de los recursos disponibles en la red con el fin de ser localizados o identificados

- **User_id**: Se trata de una clave foránea que relaciona la tabla documents con la tabla users. Esta clave establece una relación de uno a muchos en la que un usuario puede disponer de varios documentos.
- **Title**: Se trata de una cadena de texto de 255 caracteres en la que se almacena el título del texto. Será utilizado por la aplicación para mostrar el listado de documentos para que el usuario lo pueda reconocer.
- **Inputs**: En este atributo se almacena un texto en el que se recogen los campos del documento en formato JSON tal y como se ha especificado en el diseño de documentos.
- **Text**: En lugar de guardar el texto entero, se referencia en un atributo tipo cadena de texto mediante la ruta en el sistema de directorios del servidor. El servidor se encarga de acceder al archivo si se solicita desde la aplicación móvil.
- **Image**: La aplicación podrá incluir una imagen en el listado de búsqueda para que el usuario pueda reconocer el documento más fácilmente. Se trata de una cadena de texto que almacena la ruta del archivo en el servidor.
- **Created_at, updated_at**: Almacenan la fecha de creación y modificación del documento respectivamente. Tienen un formato de fecha y hora y los modifica el servidor automáticamente gracias a las funcionalidades de Laravel.

TABLA 2. EJEMPLO DE ENTIDAD EN LA TABLA DOCUMENTS

Id	User_id	Title	Text	Image	Inputs	Created_at	Updated_at
1	1	Documento de ejemplo	text1	image1	[{"type": "date", "id": "date", "hint": "fecha", "data": ""}, {"type": "time", "id": "time", "hint": "hora", "data": “”}]	2020-09-01 07:34:09	2020-09-01 07:34:09

En el ejemplo observamos como no aparece ninguna ruta en los atributos text e image. Esto es porque, dado que las imágenes y los archivos de texto se van a almacenar en un mismo directorio, el servidor ya conoce la ruta. Tampoco se indica la extensión del archivo, solo es necesario el nombre del archivo.

Users: Tabla que almacena la información de los usuarios para gestionar la autenticación y el acceso al servidor. Se ha generado a través de los comandos en Laravel junto con más tablas que utiliza internamente para gestionar la autenticación.

- **Id**: Identificador del usuario tipo entero de 20 dígitos.
- **Name**: Nombre de usuario que se introducirá al registrarse mediante la aplicación o el portal web. Se puede utilizar para iniciar sesión.
- **Email**: El correo electrónico del usuario que, al igual que el nombre, se introduce en el registro de usuario. Se puede utilizar para iniciar sesión.

- **Email_verified_at**: Esta en formato fecha y hora y representa el momento en el que se verificó el email. Esta función no está implementada en el servidor que no enviará ningún email.
- **Password**: Almacena la contraseña encriptada. La contraseña no se guardará encriptada si se crea un usuario desde la aplicación de gestión de base de datos, debe registrarse a través de la aplicación o portal web.
- **Remember_token**: Se ha generado al crear la tabla users utilizando los comandos Laravel al igual que el atributo email_verified. Es una cadena de texto que almacena información sobre una sesión abierta anterior a través de la API.
- **Created_at, updated_at**: Al igual que en la tabla documents, se almacena el momento en el que el usuario se registró o modificó sus datos.

TABLA 3. EJEMPLO DE ENTIDAD EN LA TABLA USERS

Id	Name	Email	Email_verified_at	Password	Remember_token	Created_at	Updated_at
1	Carlos	carlos@gmail.com	Null	$2y$10$uEwIWkq.VVOXIxd4banZ2.g6kCocqaJdRaqDwlDoBOKkiOOUraxri	Null	2020-09-01 07:34:09	2020-09-01 07:34:09

4.2.1.1 Llamadas a la base de datos

Las llamadas o peticiones a la base de datos se hacen a través de sentencias SQL. Estas sentencias son invocadas por el servidor cuando se accede a él a través de la API. El servidor tendrá un controlador que gestione estas llamadas y que devolverá el resultado al dispositivo móvil. Están enfocadas en ofrecer los datos necesarios para que la aplicación pueda funcionar como se requiere. A continuación, se enumeran las sentencias SQL que se utilizarán para acceder a la información de la base de datos:

```
SELECT * FROM documents ORDER BY id ASC LIMIT 20;
```

Se utiliza para obtener la información de los 20 primeros documentos en la base de datos y así poder generar el listado de selección en la aplicación Android.

```
SELECT * FROM documents WHERE id > i ORDER BY id ASC LIMIT 20;
```

Una vez se haya alcanzado el final de la lista de selección en la aplicación Android, esta utilizará la API del servidor para descargar la información de los 20 siguientes documentos. El valor "i" representa el último documento de la lista.

```
SELECT * FROM documents WHERE title LIKE '%title%' ORDER BY id
ASC LIMIT 20;
```

Con esta sentencia podemos hacer una búsqueda de los documentos que contengan la palabra “title” en su título con un máximo de 20 resultados.

```
SELECT * FROM documents WHERE id > i AND title LIKE ‘%title%’
ORDER BY id ASC LIMIT 20;
```

Una vez se a alcanzado el límite de la lista de búsqueda por título, se solicitarán los 20 siguientes resultados. La variable “i” representa el id del último documento de la lista.

Las llamadas y búsquedas a la tabla users las ejecuta el proyecto Laravel a través de sus clases y métodos. No tenemos que preocuparnos de diseñarlas a no ser que se quiera modificar su comportamiento.

4.2.2 API

La interfaz de programación de aplicaciones (API) es un conjunto de métodos que implementa una aplicación software o un servidor web, a través de los cuales otras aplicaciones pueden comunicarse con él y acceder a sus recursos. Podemos encontrar este tipo de interfaces en un gran número de herramientas software y servicios sobre las que se implementan todo tipo de aplicaciones, como Google, Twitter, Amazon, Telegram, etc.

4.2.2.1 API REST

Una API REST es un tipo de arquitectura de desarrollo web en la que se define una interfaz sobre el protocolo HTTP, a través de la que acceder a los recursos del servidor. REST son las siglas de Representational State Transfer y al sistema que implementa esta arquitectura se le llama RESTful si cumple la arquitectura. [25], [26]

API REST define una interfaz uniforme, todos los clientes deberían acceder a la misma interfaz. Las llamadas a la API se implementan como peticiones HTTP en las que la URL representa el recurso, el método (HTTP Verbs[13]) representa la operación y el código de estado HTTP representa el resultado. La API devuelve una representación del recurso a través de lenguajes como HTML, XML, JSON, etc. La representación del recurso que le llega al cliente será suficiente para poder modificar o borrar el recurso en una nueva solicitud.

[13] “HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado.” [27]

Se trata de un sistema sin estado al igual que el protocolo sobre el que se implementa. Todo lo necesario para generar una respuesta se encuentra en la petición actual. Las respuestas pueden ser cacheadas para ser contestadas más rápidamente. El sistema puede estar construido por varias capas de servicio y esto debe ser transparente para los clientes, únicamente necesitan conocer las llamadas a la API.

API REST define una semántica específica a través de los métodos HTTP para implementar CRUD[14].

- GET: Obtener un recurso.
- PUT: Actualizar datos de un recurso.
- POST: Crear un nuevo recurso.
- DELETE: Borrar el recurso.
- PATCH: Para actualizar ciertos datos

4.2.2.2 OAuth

Para implementar la autenticación en el servidor se ha utilizado el protocolo OAuth2.0. Open Authorization (OAuth) es un estándar abierto propuesto por Blaine Cook y Chris Messina, que permite autorización segura de un sitio web, API o aplicaciones informáticas de modo estándar y simple. OAuth se lanzó como estándar abierto en 2010 definido en RFC[15] 5849. [30]

OAuth permite a un usuario interactuar con datos protegidos y publicarlos. Proporciona a los usuarios un acceso a sus datos al mismo tiempo que protege las credenciales de su cuenta. Este mecanismo es utilizado por compañías como Google, Facebook, Microsoft, Twitter y Github para permitir a los usuarios compartir información sobre sus cuentas con aplicaciones de terceros o sitios web. [28]

Los tokens de portador son el tipo predominante de token de acceso que se usa con OAuth 2.0. Un token de portador es una cadena opaca sin ningún significado para los clientes que la utilizan. Los servidores emitirán tokens que son una cadena corta de caracteres hexadecimales tal y como se muestra en la Ilustración 9. [29], [26]

[14] CRUD es el acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete). El concepto CRUD está estrechamente vinculado a la gestión de datos digitales.
[15] "El Request for Comments (RFC) es un documento numérico en el que se describen y definen protocolos, conceptos, métodos y programas de Internet. La gestión de los RFC se realiza a través de IETF (Internet Engineering Task Force)." [31]

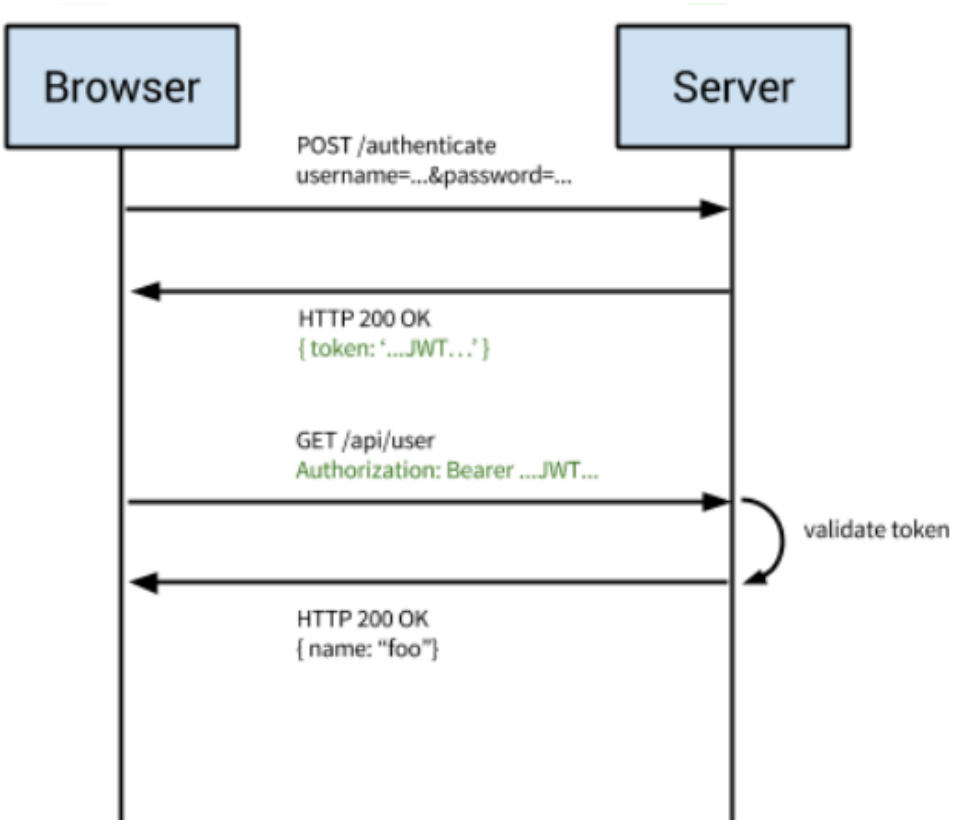

ILUSTRACIÓN 9. DIAGRAMA DE FUNCIONAMIENTO DE OAUTH2

4.2.2.3 Diseño de API

Los recursos que la aplicación Android solicitará al servidor a través de la API son principalmente son los documentos y los usuarios. Los recursos se identifican a través de una URL y se solicita una acción a través de los HTTP Verbs, del mismo modo la autenticación y cierre de sesión se realizan a través de una URL. El servidor devolverá la respuesta con un código que determina el resultado de la petición y los datos del recurso solicitado en formato JSON para ser procesado fácilmente por la aplicación móvil.

A pesar de haberse diseñado la API sobre una arquitectura REST, no se han implementado todas las recomendaciones de un servidor RESTful sobre el uso de los métodos HTTP. Esto es debido a que la API solo está diseñada para solicitar información de los documentos y no para crear o editar los recursos del servidor. Por ejemplo, no se utiliza el método POST para crear un recurso, en su lugar se utilizará para realizar búsquedas avanzadas. En el caso de crear un documento se accederá a través del portal web.

Por último, todas las llamas al servidor deben pertenecer a un usuario autenticado por medio del protocolo OAuth2, a excepción de las rutas de registro e inicio de sesión. Esto se hace a través del Bearer token que identifica la sesión al ser incluido en el campo de cabecera HTTP Authorization. El token será gestionado por la aplicación que lo incluirá en cada llamada al servidor o lo recibirá una vez se registre o inicie sesión.

Login:

Para iniciar sesión se debe acceder a la siguiente URL:

```
http://servidor/firma/public/api/login
```

Se utilizará el método POST y los parámetros que componen la llamada serán el email y la password. Si el usuario es verificado por el servidor, en la respuesta se incluirá el Bearer token junto con la información del usuario.

```
POST /firma/public/api/login HTTP/1.1
[…]Content-Type: application/json
Content-Length: 60

{"email":"carlos@gmail.com","password":"123456"}
```

La respuesta del servidor puede ser afirmativa o por el contrario puede expresar algún error en la solicitud. En caso de ser correcta la solicitud, el servidor devuelve el código 200 junto con el Bearer token que identifica la sesión del usuario en un objeto JSON:

```
HTTP/1.1 200 OK
[…]Content-Type: application/json

{"success":{"token":"oeYKq5[…]8_xdf"}}
```

Si la solicitud no se puede validar o no está autorizada devuelve el código de respuesta 401 que representa un fallo de autorización:

```
HTTP/1.1 401 Unauthorized
[…]{"error":"Unauthorised"}
```

Register:

Para crear un nuevo usuario tenemos que hacer una petición a la siguiente URL:

```
http://servidor/firma/public/api/register
```

Al igual que en la solicitud de inicio de sesión, esta petición al servidor debe incluir los parámetros necesarios a través del método POST. En este caso, además del email y password, se incluirá el nombre de usuario y la confirmación de la contraseña.

```
POST /firma/public/api/register HTTP/1.1

[…]

{"name":"carlos","email":"carlos@gmail.com","password":"123456",
"c_password": "123456"}
```

Si la solicitud se verifica correctamente, el servidor crea el usuario y devuelve el código 200, el Bearer token y el nombre de usuario.

```
HTTP/1.1 200 OK

[…]{"success":{"token":"oeYKq5[…]8_xdf","name":"carlos"}}
```

El código 401 indicará el fallo de autorización y el servidor no registrará el usuario. Devolverá un mensaje indicando cual ha sido el error.

```
HTTP/1.1 401 Unauthorized

[…]{"error":{"c_password":["The c password and password must
match."]}}
```

Documents:

Las llamadas al servidor se podrán hacer mediante los métodos GET o POST en función de las necesidades de la búsqueda. Además, se necesita utilizar el Bearer token que previamente habremos recibido al iniciar sesión. La URL a la que se accede es:

```
http://servidor/firma/public/api/documents
```

Búsqueda básica:

```
GET /firma/public/api/documents/20 HTTP/1.1

[…]Authorization: Bearer oeYKq5[…]8_xdf
```

Esta llamada a la API devuelve un listado de máximo 20 resultados, con la información de los documentos almacenados en la base de datos. El numero entero que se añade al final de la URL representa el último documento recibido en la aplicación, por lo que el servidor devolverá los 20 siguientes recursos. Se reciben los 20 primeros si no se especifica ningún número o utilizando el número 0.

Búsqueda por título:

```
POST /firma/public/api/documents HTTP/1.1

[…]Authorization: Bearer oeYKq5[…]8_xdf
```

```
{"id":"0","title":"doc"}
```

El servidor devolverá una lista de 20 documentos que incluyan la cadena de texto solicitada en su título. La cadena y el último recurso solicitado se incluyen en los parámetros del método POST.

Ambas solicitudes a la API comparten el formato de respuesta.

```
HTTP/1.1 200 OK

[…]

{"success":true,"data":[{"id":1,"title":"Documento de prueba
1","inputs":[{"type":"text", "id":"name1","hint":"nombre de
pueba","data":""}],"text":"3","image": "1600569001",
"created_at":"01/09/2020","updated_at": "01/09/2020"},[…]

{"id":20,"title":"Documento de prueba 20","inputs":[{"type":
"name","id":"name1","hint":"nombre de pueba","data":""}],
"text":"3","image":"1600569001","created_at": "01/09/2020",
"updated_at":"01/09/2020"}],

"message":"Products retrieved successfully."}
```

La respuesta incluye la información que se incluirá en la lista de búsqueda junto con los campos del documento que se utilizarán para generar la interfaz de introducción de datos. Es posible que con documentos con un gran número de campos sea preferible utilizar una nueva llamada a la API que devuelva únicamente los campos del documento solicitado.

Text:

Cuando se quiera previsualizar el documento, la aplicación utilizará la referencia al texto del recurso de la base de datos y lo descargará a través de la siguiente URL:

```
http://servidor/firma/public/api/text/i
```

Se accede a los recursos a través de un identificador en la URL que coincide con el nombre de fichero del texto en el servidor y con el atributo text de la tabla documents.

Petición:

```
GET /firma/public/api/text/1 HTTP/1.1

[…]Authorization: Bearer oeYKq5[…]8_xdf
```

Respuesta:

```
HTTP/1.1 200 OK

[…]Content-Type: text/html;charset=UTF-8
```

```
<HTML><HEAD>[…]</HEAD><BODY>[…]
<name1>_____</name1>[…]</BODY></HTML>
```

La respuesta ya no es un JSON sino un archivo tipo HTML en el que se define la estructura del texto y su contenido.

Details:

Se utiliza para recibir información del usuario.

```
GET /firma/public/api/details HTTP/1.1

[…]Authorization: Bearer oeYKq5[…]8_xdf
```

Respuesta:

```
HTTP/1.1 200 OK

[…]{"success":{"id":3,"name":"carlos","email":"carlos@gmail.com"
,"email_verified_at":null,"created_at":"2020-11-
18T02:10:28.000000Z","updated_at":"2020-11-
18T02:10:28.000000Z"}}
```

Errores:

Las respuestas pueden devolver un error en diferentes situaciones, acompañado del código que las identifica.

Cuando intentamos acceder a la API a través de un método HTTP equivocado.

```
HTTP/1.0 405 Method Not Allowed

[…]{"message": "The GET method is not supported for this route.
Supported methods: POST.", "exception": "[…]", "file": "[…]",
"line": […], "trace": [[…]]}
```

Cuando la URL no es correcta.

```
HTTP/1.0 404 Not Found

[…]{"message": "", "exception": "[…]", "file": "[…]", "line":
[…], "trace": [[…]]}
```

Cuando no son correctas las credenciales de inicio de sesión o registro o cuando accedemos con un Bearer token equivocado.

```
HTTP/1.1 401 Unauthorized […] {"message":"Unauthenticated."}
```

Cuando ocurre un fallo en el servidor. Este error nos ayudará a solucionar los errores del servidor consultando el mensaje y la traza.

```
HTTP/1.1 500 Internal Server Error[…]
```

4.2.3 Web

Se necesita diseñar un entorno web para subir los documentos al servidor. Accederemos, a través del navegador web del PC o del dispositivo móvil, a la ventana de inicio del servidor desde la que podremos generar nuevos recursos para utilizarlos posteriormente en la aplicación Android.

La ventana tendrá un formulario en el que se introducirán los datos necesarios para generar un nuevo documento. Como se muestra en la Ilustración 10, el formulario está compuesto por los campos título, inputs, image y document. Los campos son validados por el servidor una vez se ha enviado. Si el usuario no se ha autenticado, al enviar el formulario no será aceptado y se le redirigirá a la ventana de inicio de sesión y registro. Una vez se verifiquen los datos introducidos en el formulario, el servidor almacenará los dos archivos que se han seleccionado en sus respectivas carpetas dentro del sistema de archivos del servidor. Por último, se utilizarán los nombres de los archivos para generar un nuevo recurso en la base de datos. En la Ilustración 11 se muestra un diagrama representativo de cómo funciona el servidor web.

Los campos del formulario son:

- **Title**: Es una cadena de texto que contiene el título del documento.
- **Inputs**: Se establecen los campos que tendrá el documento en este atributo en formato JSON.
- **Image**: Cuando se seleccione este campo se abrirá una ventana de explorador de archivos del dispositivo que accede a la web para seleccionar la imagen de cabecera que se desea subir al servidor.
- **Document**: Del mismo modo que con la imagen, el archivo HTML se seleccionará desde los archivos del dispositivo que se conecte al servidor.

Únicamente es necesario introducir el siguiente fragmento de código HTML en el archivo de la vista principal ***welcome.blade.php*** del servidor, creada automáticamente por Laravel. No se ha modificado el estilo de la vista ni se ya que no es el objetivo del proyecto diseñar gráficamente el portal web sino implementar una funcionalidad necesaria para que la aplicación Android pueda funcionar correctamente.

```
<div class="title2 m-b-md"> Crear formulario </div>

<form method="post" id="formDocument"
action="{{route('document.add')}}" enctype="multipart/form-
data">

<div style="height: 20px;">Tittle</div> <input type="text"
name="title" id="title"/>

<div style="height: 20px;">Inputs</div> <input type="text"
name="inputs" id="inputs"/>
```

```
<div style="height: 20px;">Image</div> <input type="file"
name="image" id="image"/>

<div style="height: 20px;">Document</div> <input type="file"
name="document" id="document"/>

<div style="height: 5px;"></div> <input type="submit"/>

</form>
```

ILUSTRACIÓN 5. VISTA WEB PRINCIPAL

Se deben gestionar dos rutas web por parte del servidor. La ruta principal que muestra la vista de la Ilustración 10 y la ruta que gestionará la solicitud del formulario de creación de documentos. El formulario enviará los datos a través del método POST a la siguiente ruta:

```
http://servidor/firma/public/fileUpload
```

Si se verifica la solicitud, el servidor devolverá el recurso creado en un objeto JSON:

```
{"success":true,"data":{"id":32,"title":"Documento 32",
"inputs":[{"type": "name","id":"name1","hint":"nombre de
pueba","data":""}],
"text":1607492426,"image":1607492426,"created_at":"09/09/20
20", "updated_at":"09/09/2020"},"message":"Document created
successfully."}
```

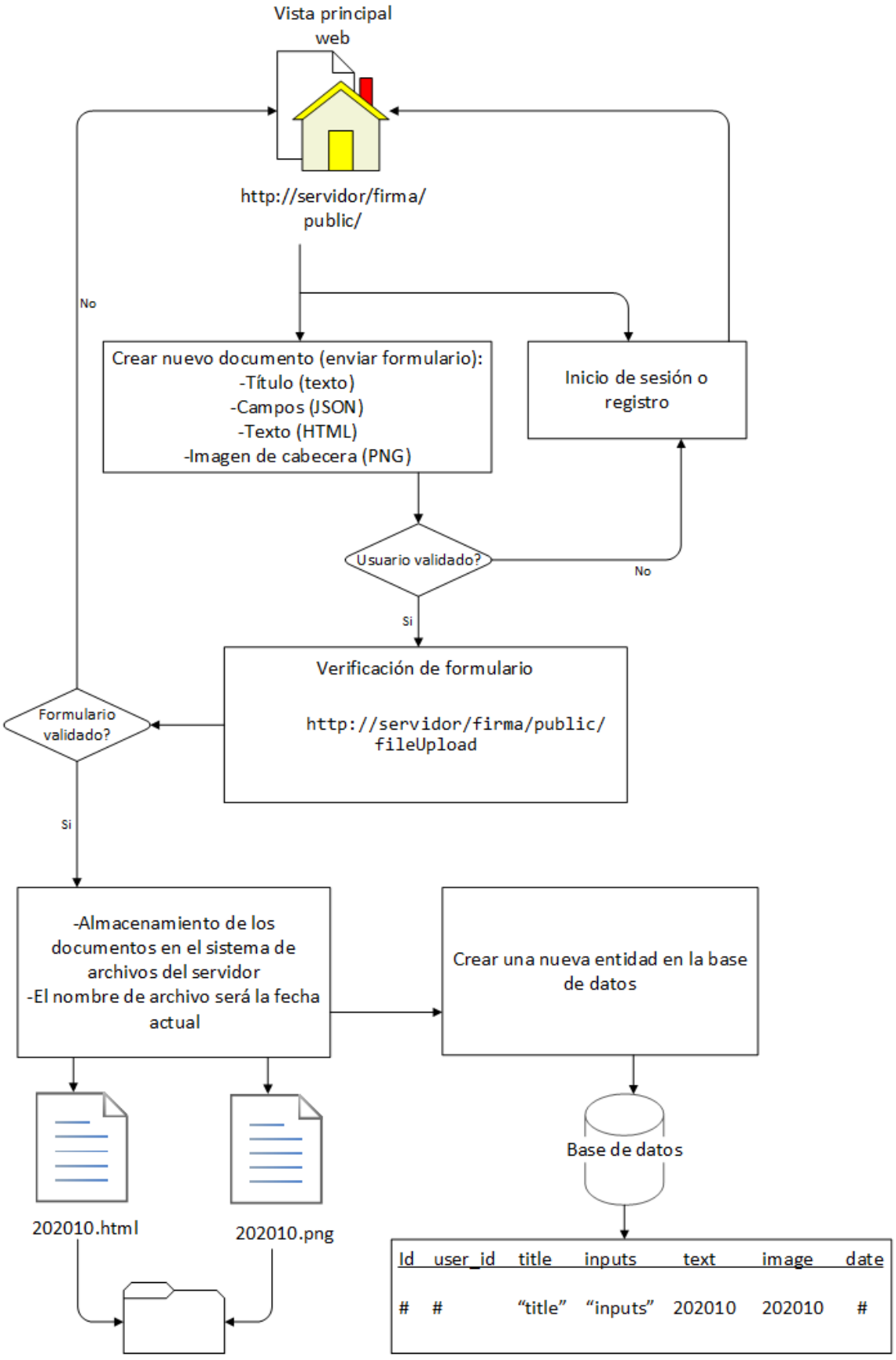

ILUSTRACIÓN 11. DIAGRAMA DE FLUJO DE LA WEB

4.2.3 Verificación de firma

La aplicación pretende ofrecer un sistema de firma seguro y que tenga validez legal y admisible como prueba ante cualquier tribunal. Para ello se pretende implementar el concepto de firma digital avanzada que se define en el eIDAS (el Reglamento UE Nº 910/2014).

Junto con la definición de firma digital avanzada se definen otras dos categorías de firma que se diferencian en el grado de confianza que tiene el método de firma. Estos métodos de firma por orden de menor confianza a mayor son: firma digital, firma digital avanzada y firma digital cualificada.

En el caso de la firma digital no es adecuada para el caso de uso de la aplicación ya que, aunque tiene una configuración sencilla, no se asegura la validez legal ya que no permite identificar al usuario de forma inequívoca, sino que consiste en marcar una casilla o introducir un código PIN. La firma digital cualificada tampoco es útil debido a que, a pesar de ser la que mayor confianza tiene, su implementación es más específica y no se adapta al caso de uso del proyecto.

La firma digital avanzada es la que mejor se adapta al caso de uso de la aplicación y no necesita un certificado cualificado de firma electrónica (DNIe) o de un dispositivo seguro de creación de firma cualificado. Los requisitos de la firma avanzada se recogen en el artículo 26 del eIDAS: [32]

> a) estar vinculada al firmante de manera única;
>
> b) permitir la identificación del firmante;
>
> c) haber sido creada utilizando datos de creación de la firma electrónica que el firmante puede utilizar, con un alto nivel de confianza, bajo su control exclusivo, y
>
> d) estar vinculada con los datos firmados por la misma de modo tal que cualquier modificación ulterior de los mismos sea detectable.

Para cumplir con los requisitos, la aplicación recogerá la firma y la información relacionada con el cliente combinando datos biométricos (firma manuscrita) e información del usuario. Una vez hecho esto, se generará un token a través de los datos del cliente y se enviará un correo de confirmación con un enlace a la ruta de validación del servidor. Cuando el cliente acceda a la ruta utilizando el token que se ha generado específicamente para él quedará validada la firma. Ejemplo de enlace:

```
http://servidor/firma/public/validate/8jD0f3j[...]jn0J8J3n
```

4.3 Aplicación Android

El diseño de la aplicación establecerá un sistema de pantallas junto con las diferentes funcionalidades que tiene cada una, así como los diferentes menús que facilitarán al usuario la navegación a través de la aplicación. En el capítulo de implementación se detallará todos los métodos y clases utilizadas para desarrollar las funcionalidades de la aplicación.

En la Ilustración 12 se muestra un diagrama con el flujo principal de la aplicación y los procesos que las diferentes pantallas realizan. En este diagrama vemos como se relacionan las distintas pantallas y como acceden a los recursos tanto del servidor como del dispositivo móvil para ofrecer las funcionalidades que se requieren.

4.3.1 Diseño de pantallas

A continuación, se hará una exposición de los diseños de las pantallas y las funcionalidades que contiene. Las pantallas y los diferentes objetos que se muestran en ellas se han diseñado a través del entorno de trabajo Android Studio, que facilita esta tarea gracias a su interfaz y herramientas. Las pantallas se diseñan sobre archivos XML que serán gestionados internamente por el sistema Android. El comportamiento de los objetos definidos en los archivos XML se codifica a través de las clases definidas en lenguaje Java.

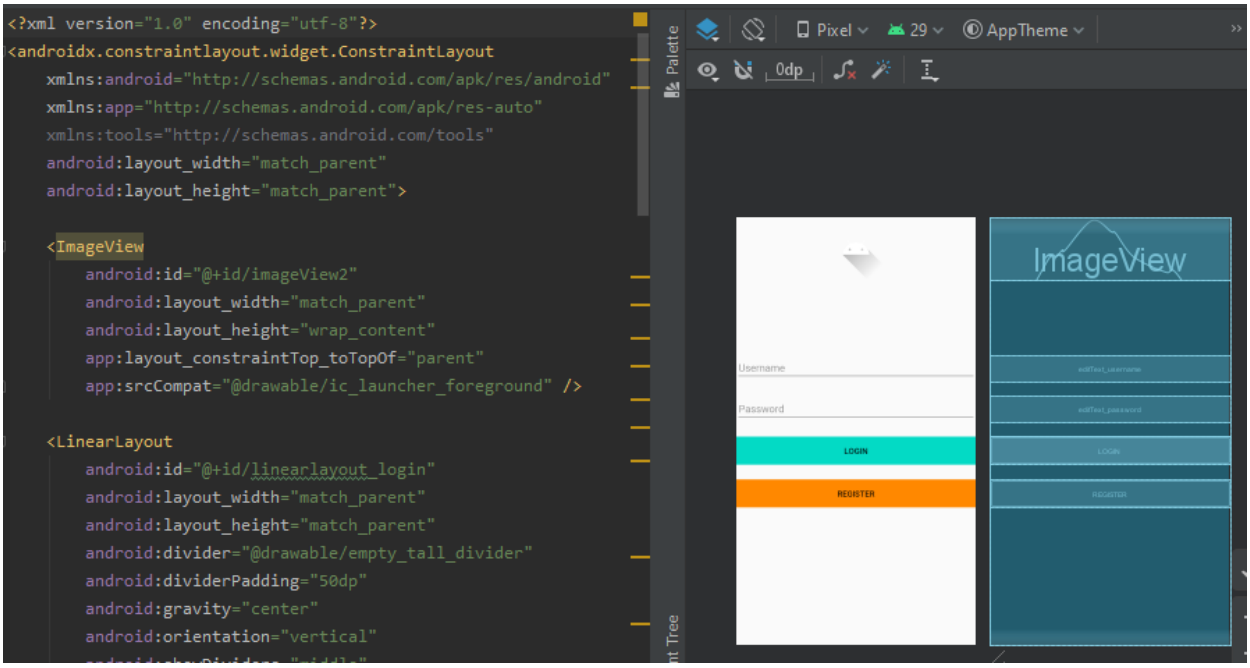

ILUSTRACIÓN 6. CAPTURA DE PANTALLA DEL EDITOR DE DISEÑO DE ANDROID STUDIO

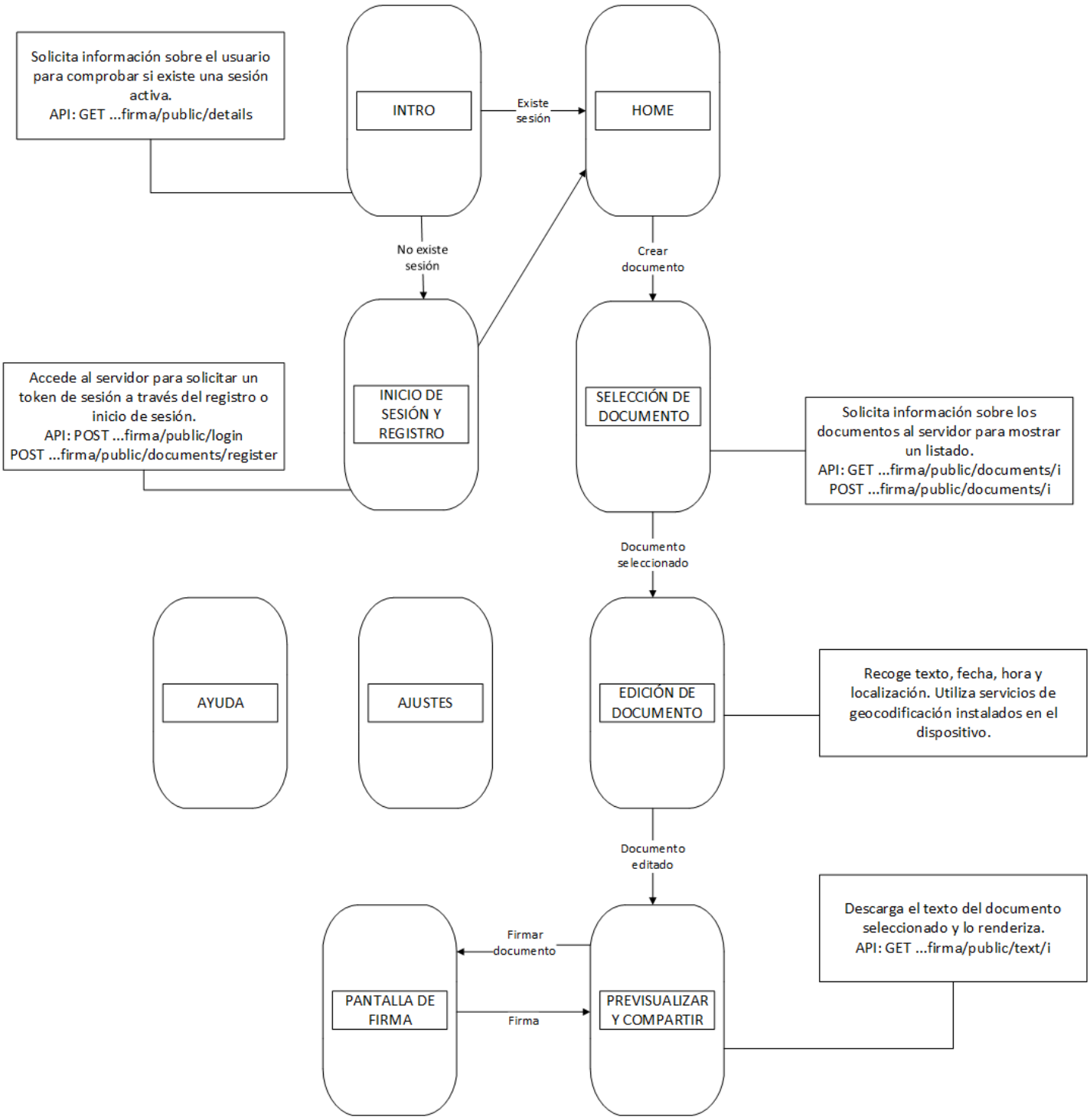

ILUSTRACIÓN 7. DIAGRAMA DE PANTALLAS DE LA APLICACIÓN

Pantalla de inicio:

Únicamente sirve de introducción y para comprobar si se encuentra una sesión abierta con anterioridad. Se comprueba si existe algún Bearer token almacenado y si sigue siendo válido. En caso de no existir sesión se accede a la pantalla de inicio de sesión, si existe sesión navegamos a la pantalla home.

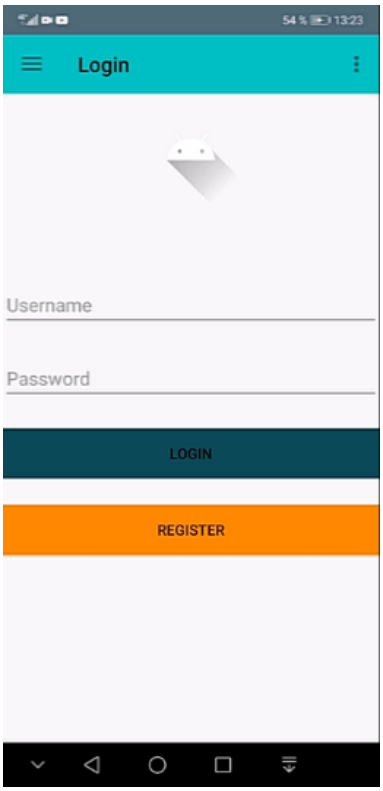

ILUSTRACIÓN 8. PANTALLA DE INICIO DE SESIÓN

Login y registro:

Se trata de dos formularios en los que se introduce la información de inicio de sesión o de registro. En el caso de que la información de usuario sea incorrecta, se mostrará un mensaje de error. Para iniciar sesión la aplicación accederá a la API del servidor a través de las dos rutas:

```
POST firma/public/api/login

POST firma/public/api/register
```

Home:

La pantalla home no tiene ninguna funcionalidad que aparezca en los requisitos de la aplicación, únicamente se ha diseñado para hacer de pantalla principal desde la que se inicia la creación de documentos.

El botón que aparece en la pantalla con el símbolo "+" es el que tenemos que presionar para crear un nuevo documento. Este botón es uno de los objetos de la librería de Android que nos facilita la tarea de diseño de la interfaz. Cambiará de icono en función de la tarea que desempeñe en cada pantalla y se ocultará si no es necesario.

ILUSTRACIÓN 9. PANTALLA HOME

Pantalla de selección:

En esta pantalla se hace uso de las clases de la librería de Android para generar la lista de documentos. Es necesario diseñar tanto la pantalla como los ítems que aparecerán en la lista haciendo uso del editor de Android Studio para definir su estructura a través de un archivo XML.

Los ítems que aparecen en la lista contienen información sobre el título, la fecha de publicación y el código identificador. Además, se mostrará un botón de previsualización en cada ítem de la lista. La pantalla dispone de un buscador en la parte superior en que se podrá introducir una cadena de texto que se utilizará para hacer las búsquedas por título en el servidor.

La aplicación hará las peticiones de documentos a través de la API utilizando las rutas correspondientes, tal y como se ha diseñado la API. Cuando se llega al final de la lista la aplicación cargará automáticamente nuevos resultados.

```
GET firma/public/api/documents/i

POST firma/public/api/documents/i
```

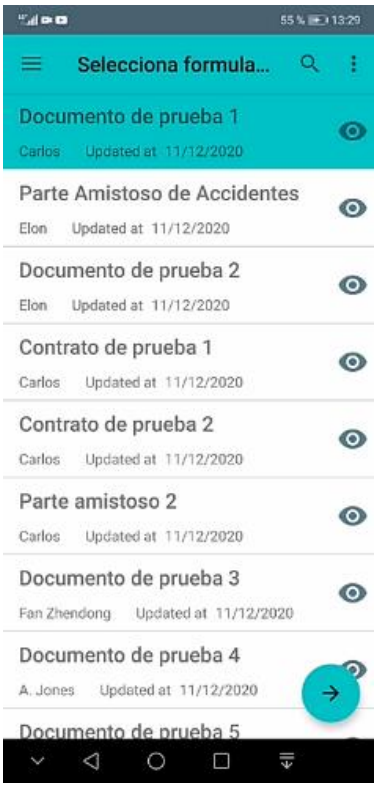

ILUSTRACIÓN 10. PANTALLA DE SELECCIÓN

Pantalla de edición:

En esta pantalla se muestran los campos que aparecen en el documento en una interfaz dinámica que nos permite introducir los datos necesarios. En función de los campos del documento aparecerán las casillas en su respectivo orden y con un tipo de datos distintos. Pueden ser de texto, fecha, tiempo, dirección.

La aplicación accederá a recursos del móvil para facilitar y automatizar la introducción de los datos. La fecha y hora actual aparecerán por defecto en los campos de fecha y hora. En el campo de dirección el dispositivo

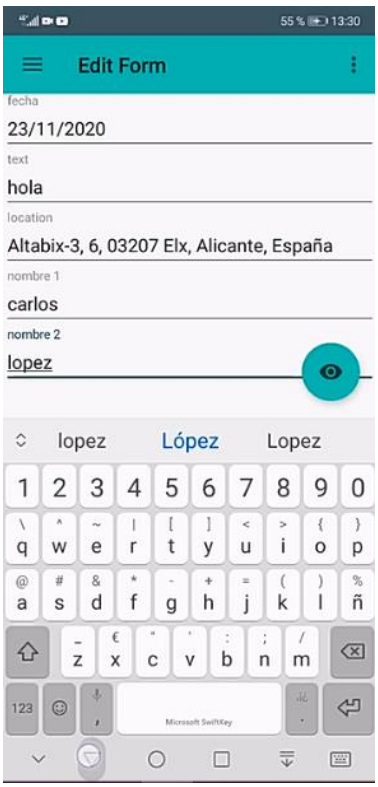

ILUSTRACIÓN 11. PANTALLA DE EDICIÓN

accederá a nuestra localización para mostrar una ayuda con las direcciones más próximas.

Previsualización:

En este caso la pantalla renderiza documentos HTML respetando el formato y la estructura. Para ello deberá acceder al servidor y solicitar el archivo correspondiente.

```
GET firma/public/api/text/i
```

Si se han introducido todos los campos de manera correcta y se necesita una firma para finalizar el documento, la pantalla de previsualización nos mostrará un botón desde el que acceder a la pantalla de firma. Si se ha completado el documento, se mostrará un botón para compartir el documento o almacenarlo en la memoria del dispositivo.

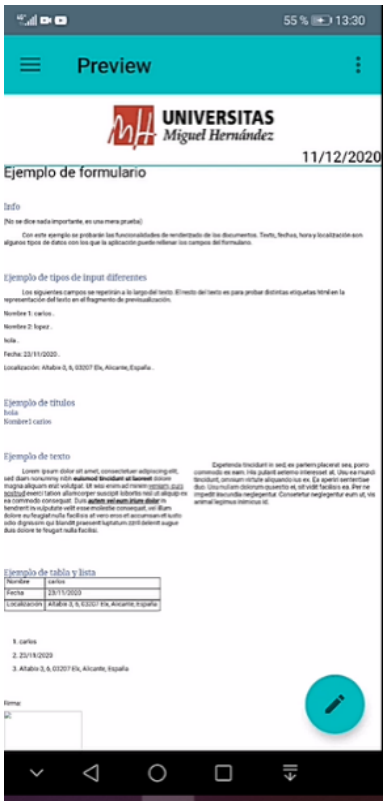

ILUSTRACIÓN 12. PANTALLA DE PREVISUALIZACIÓN

Pantalla de firma:

Esta pantalla recogerá la firma de una persona que escriba sobre la pantalla táctil del dispositivo móvil ya sea con su dedo o con un lápiz. A parte de para recoger la firma, puede servir para hacer pequeños dibujos si lo necesitase el documento. Una vez se confirma la firma, se volverá a la pantalla de previsualización.

ILUSTRACIÓN 13. PANTALLA DE FIRMA

Pantalla de ayuda:

Esta pantalla muestra la ayuda de la aplicación y se puede acceder a ella desde cualquier pantalla. Se trata de una pantalla deslizable en la que cada página muestra información sobre una parte de la aplicación. Se mostrará una página distinta de la ayuda en función de la pantalla desde la que se accede.

Pantalla de ajustes:

Mostrará una serie de información de interés como la URL de la API o el token utilizado para acceder al servidor. Además, se podrá revisar el estado del permiso de localización y activarlo si es necesario. Por último, se podrá cambiar el tema de la aplicación pulsando sobre el icono correspondiente.

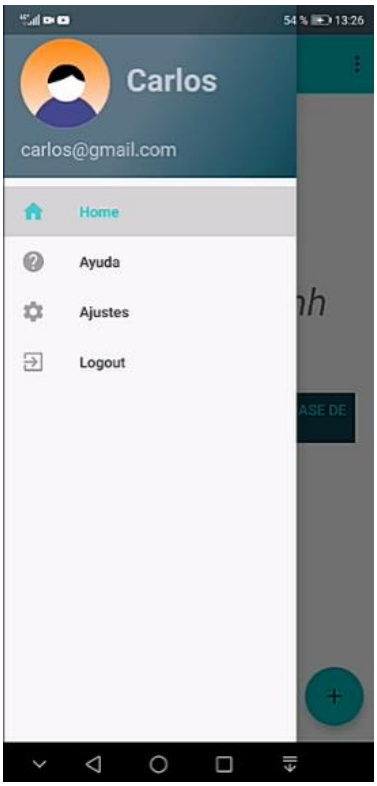

ILUSTRACIÓN 14. MENÚ LATERAL DE NAVEGACIÓN

Menú lateral desplegable:

Uno de los requisitos de la aplicación era que se pudiera navegar a través de ella a través de un menú lateral como en la mayoría de las aplicaciones comerciales. El uso de esta barra lateral facilita la navegación y es fácil de implementar ya que Android también nos lo facilita a través de su librería. Los ítems del menú desplegable nos servirán para navegar a las pantallas que no pertenecen al flujo principal de la aplicación. El diseño se define como cualquier otro elemento visual de la aplicación a través de un archivo XML.

4.3.1 Estilos y temas

Del mismo modo que para el diseño y distribución de los objetos de la interfaz, los estilos y temas se puede definir en un archivo XML en Android Studio. Estos estilos permitirán definir otro tipo de aspectos del diseño como pueden ser el color, tamaño y tipo de fuente, iconos, etc. El archivo de estilos es un recurso como lo pueden ser el diseño de pantallas, de navegación o imágenes y por lo tanto deberá incluirse en **\app\src\main\res\values\styles.xml**.

Utilizando estos estilos se han definido dos temas para la aplicación con la intención de mejorar la aplicación y añadir una nueva opción a la pantalla de ajustes. En la Ilustración 21 se muestra una comparativa de cada uno de los temas. Los elementos del tema son principalmente los iconos y los colores de la aplicación y del texto. Un tema tiene un estilo oscuro y el otro un estilo más claro. Se ha utilizado como base los temas de la librería de Android y se han extendido para incorporar nuevos elementos.

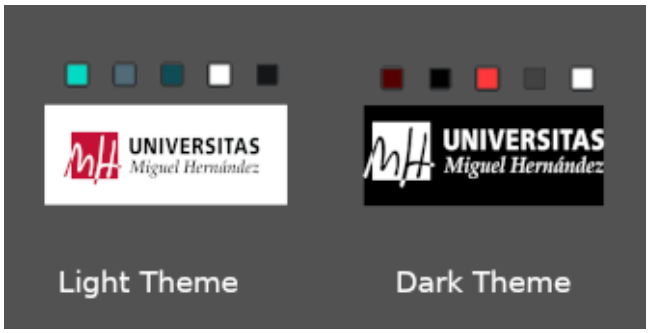

ILUSTRACIÓN 15. TEMAS DE LA APLICACIÓN

En la Ilustración 22 se muestran unos ejemplos de cómo la aplicación se ve en función del tema utilizado.

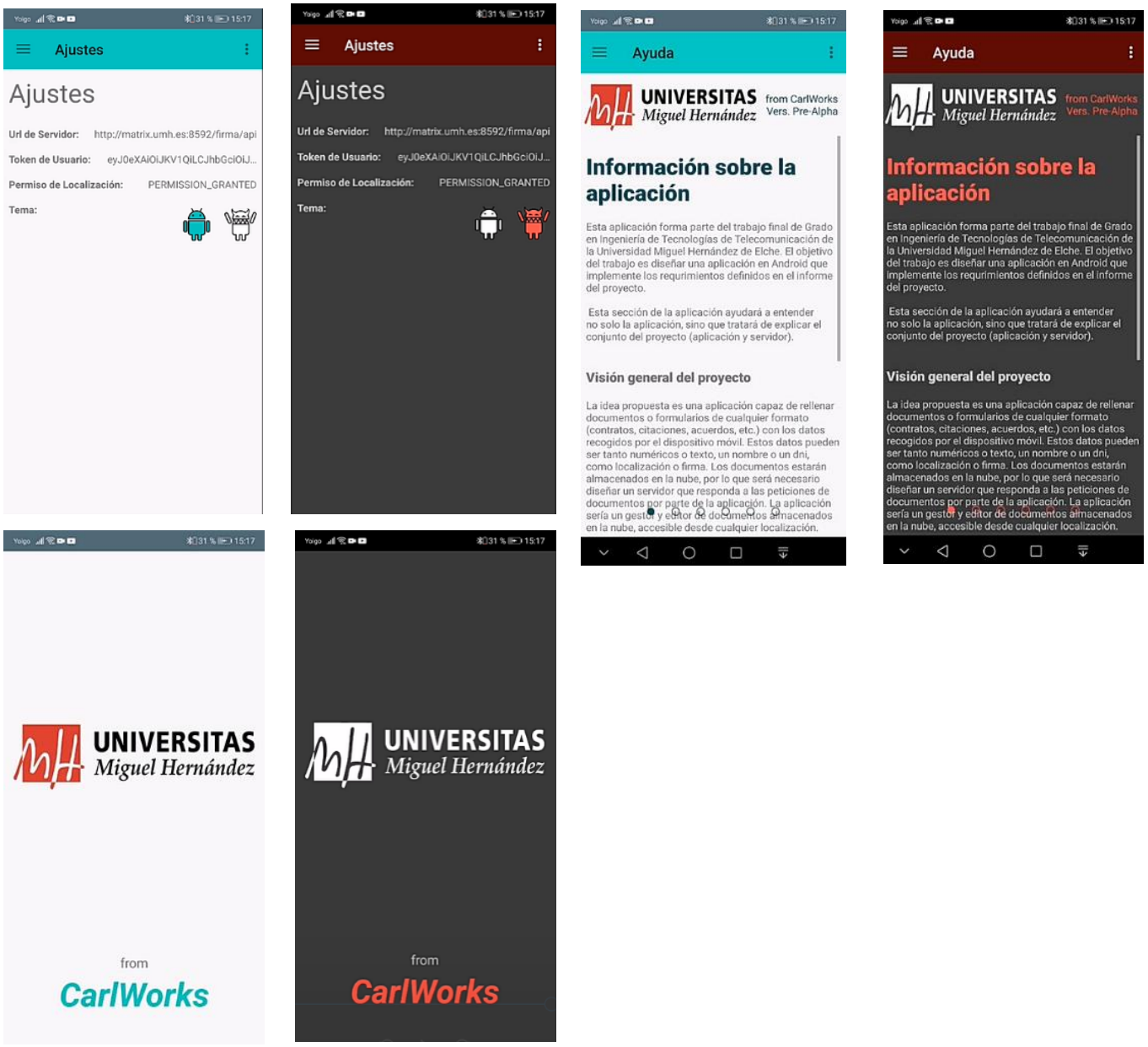

ILUSTRACIÓN 16. COMPARATIVA DE LOS TEMAS EN LAS VENTANAS DE AJUSTES, INTRODUCCIÓN Y AYUDA

5. IMPLEMENTACIÓN

5.1 Aplicación Android

La aplicación Android se ha desarrollado utilizando la herramienta software Android Studio. Se ha implementado sobre API de nivel 29 (Android 10) y es compatible desde API nivel 22 (Lollipop) hasta las versiones más recientes. Se ha codificado en el lenguaje Java.

5.1.1 Creación del proyecto y estructura

Es muy sencillo crear un proyecto en Android Studio siguiendo la ayuda del programa. Existen modelos de Activity[16] desde los que poder comenzar a implementar nuestra aplicación y sus pantallas, y uno de esos modelos es el óptimo para cumplir con los requisitos de la aplicación y el diseño. Este modelo se llama "Navigation Drawer Activity". Una vez elijamos el modelo de proyecto sobre el que nos queremos basar, elegiremos una serie de parámetros como nombre de la aplicación y versión Android con la que se quiere trabajar.

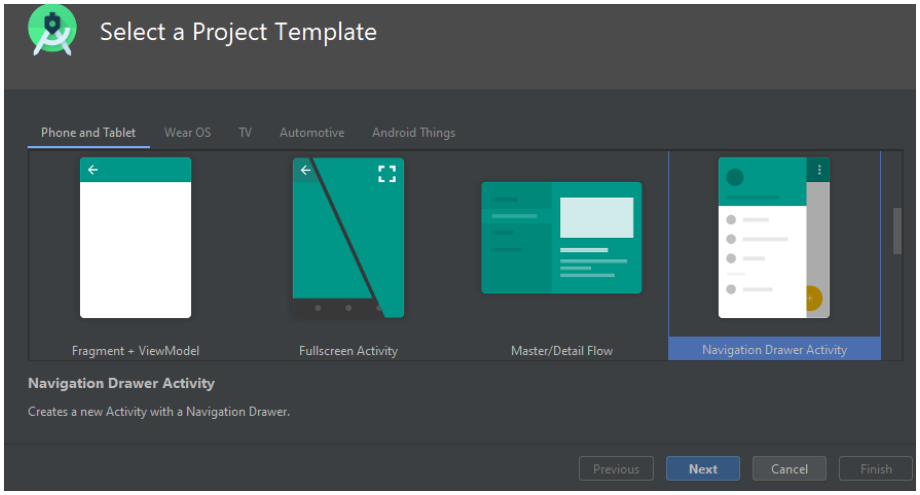

ILUSTRACIÓN 23. CREACIÓN DE PROYECTO EN ANDROID STUDIO

Al iniciar el proyecto basado en "Navigation Drawing Activity", se generan una serie de archivos que serán la base de la estructura del proyecto. La activity principal, el archivo Manifest y los recursos de la aplicación se generarán automáticamente y podremos completarlos e incluir modificaciones posteriormente. La aplicación está compuesta

[16] "La clase Activity se encarga de crear una ventana para usted en la que puede colocar su interfaz de usuario. Si bien las actividades a menudo se presentan al usuario como ventanas de pantalla completa, también se pueden usar de otras formas: como ventanas flotantes, modo de ventanas múltiples o incrustadas en otras ventanas. […] La clase Activity es una parte importante del ciclo de vida general de una aplicación" [33]

por una única activity que contendrá las distintas pantallas. Cada pantalla tendrá una funcionalidad diferente y serán independientes. La activity gestionará las pantallas y la información que devuelven.

5.1.2 Android Manifest

Según la documentación de Android Developers: "Todos los proyectos de apps deben tener un archivo ***AndroidManifest.xml*** (con ese mismo nombre) en la raíz de la fuente del proyecto. El archivo de manifiesto describe información esencial de tu aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play." [34]

Los elementos que se deben declarar son el nombre de paquete de la aplicación, los componentes que utiliza la aplicación (actividades, servicios, receptores de emisiones y proveedores de contenido), los permisos de la aplicación y funciones de hardware y software específicas.

El archivo manifest de la aplicación incluye los siguientes apartados:

Nombre del paquete:

```
<manifest […] package="com.carlosdlp.appammforms">
```

Permisos:

```
<uses-permission android:name="android.permission.INTERNET" />

<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />

<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Los permisos deben declararse para acceder a los recursos del dispositivo móvil y sistema operativo. Estos permisos deberán aceptarse por el usuario por medio de una solicitud de la propia aplicación o a través de los ajustes del sistema.

Necesitaremos permisos para acceder a internet, para acceder a la localización del dispositivo y para acceder al almacenamiento.

Servicios:

```
<application […]
  <service android:name=".AddressManager" />
```

La aplicación implementa un servicio para traducción de localización a dirección. Se deben declarar los servicios en el manifest.

Otros elementos en manifest:

Podemos declarar también otros elementos como el icono que tendrá la aplicación en el menú de aplicaciones del dispositivo haciendo referencia una imagen de los recursos del proyecto.

```
<application
    […]
    android:icon="@mipmap/ic_launcher"
```

El manifest se autocompleta si utilizamos la interfaz de Android Studio, al crear el proyecto se declaran la activity principal y proveedor. Se puede editar el manifest para agregarle algún tipo de restricción a la activity, en este caso se desea que nunca se pueda utilizar la aplicación con el dispositivo apaisado.

```
<application
    […]
    <activity
        android:name=".MainActivity" […]
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

La aplicación únicamente dispone de una activity que gestiona toda la aplicación y que se lanzará al iniciar la aplicación.

5.1.3 Archivo de compilación y librerías

El sistema de compilación de Android compila recursos y código fuente de la app para generar el archivo APK. Android Studio usa Gradle, un paquete de herramientas de compilación avanzadas, para automatizar y administrar el proceso de compilación, y al mismo tiempo definir configuraciones de compilación personalizadas y flexibles.

Será necesario manipular uno de los archivos de compilación Gradle para incluir una serie de componentes que se utilizarán en la aplicación y que no pertenecen a las librerías por defecto de Android o pertenecen a librerías externas. Para ello incluiremos las dependencias en el formato que se utiliza en todos los proyectos Gradle. Accedemos al archivo ***\app\build.gradle***(Module: app) para incluir las dependencias que no se han incluido automáticamente al generar el proyecto con la ayuda de la interfaz de Android Studio:

```
dependencies {
    […]
    implementation 'com.android.volley:volley:1.1.1'
    implementation 'com.google.code.gson:gson:2.8.6'
    implementation 'org.jsoup:jsoup:1.11.3'
    implementation "com.google.android.gms:play-services-
location:17.0.0"
}
```

Se necesita la librería Volley de Android para gestionar peticiones HTTP, Gson de Google para gestión de objetos JSON, la librería Jsoup para manipular contenido HTML y las librerías de Google para el acceso a localización. En los próximos apartados se detalla la utilización de estas librerías.

5.1.4 Activity y Fragments

La aplicación estará compuesta principalmente de dos elementos fundamentales que deben quedar claros para comprender el desarrollo y estructura de la aplicación. Estos dos elementos con la activity y los fragments, ambos son clases de la librería de

Android que proporcionan las funcionalidades necesarias para el desarrollo de aplicaciones Android y la gestión de la interfaz de usuario.

La activity es el elemento fundamental para definir la pantalla de la aplicación, conteniendo todas las vistas de la interfaz de usuario con las que interactúa. Años atrás las activitys definían una única pantalla, por lo que eran necesarias varias activities para implementar una aplicación multipantalla. Con la llegada de las tabletas se introdujo un nuevo concepto a partir de la versión Android 3.0 que permitía dividir la pantalla en fragments y ofrecer varios paneles con una interfaz y funciones independientes del resto de paneles, tal y como se muestra en la Ilustración 25 se muestra el concepto de fragment. Android permite actualmente diseñar una aplicación con una sola activity y varios fragments que irán intercambiándose para dar lugar a las diferentes pantallas. [35]

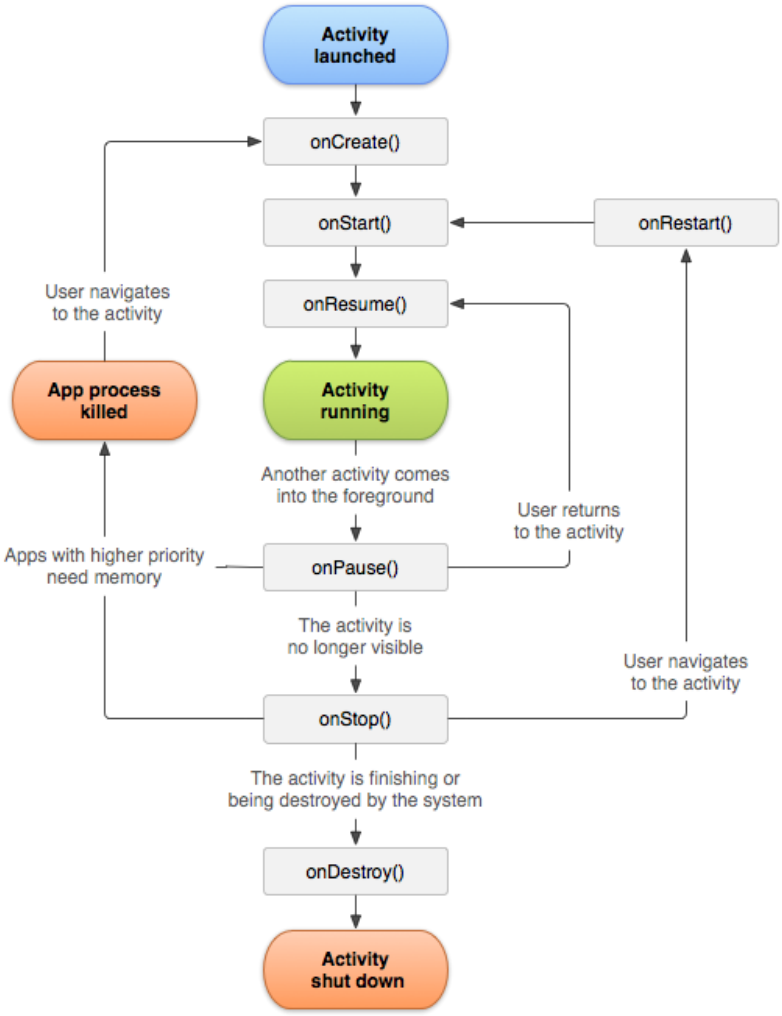

ILUSTRACIÓN 17. CICLO DE VIDA DE UNA ACTIVITY. [33]

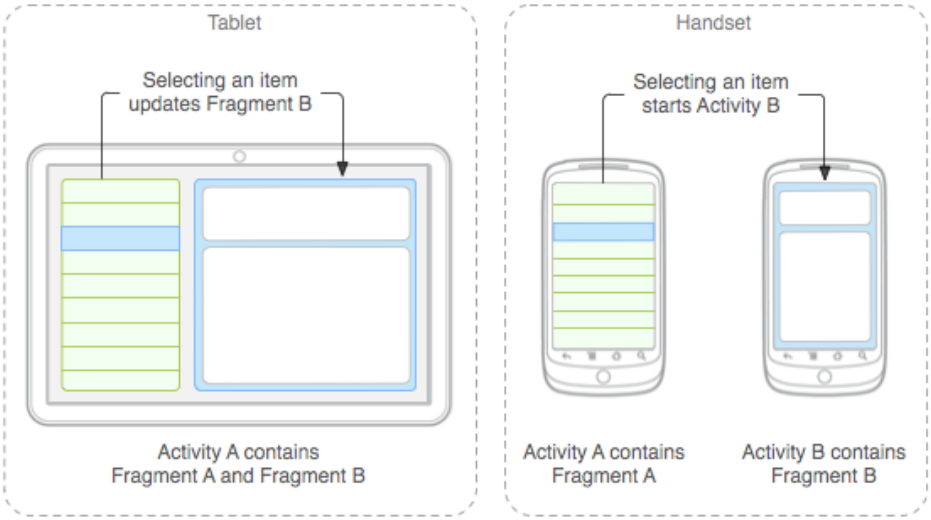

ILUSTRACIÓN 25. ESQUEMA DE CÓMO SE COMBINAN LOS FRAGMENTOS EN UNA ACTIVIDAD. [35]

El sistema Android gestiona las diferentes activities en su pila de ejecución. Cuando el estado de la activity cambia en la pila de ejecución, se invocan una serie de métodos que debemos definir en su código de comportamiento en ***\app\src\main\java\com\carlosdlp\appammforms\MainActivity.java***. Se ejecutará el método onCreate() cuando se inicie la activity y onPause() cuando el usuario detiene la interacción. [33]

Dado que la aplicación únicamente estará compuesta por una única Activity, podemos considerar el ciclo de vida de la Activity como el ciclo de vida de la propia aplicación, que se debe tener en cuenta a la hora de programar la aplicación. Los Fragments también tienen un ciclo de vida que depende del ciclo de vida de la Activity.

5.1.5 Navegación entre pantallas

Para navegar entre las pantallas de la aplicación se hace uso de la clase NavHostFragment del paquete **androidx.navigation.fragment.NavHostFragment**. Proporciona un área dentro de la activity para que se produzca la navegación autónoma entre los diferentes fragments de la aplicación. Se define dentro del diseño gráfico de la activity, en el archivo **\app\src\main\res\layout\activity_main.xml**.

```
<fragment
android:id="@+id/nav_host_fragment"
android:name="androidx.navigation.fragment.NavHostFragment"[…]
app:navGraph="@navigation/mobile_navigation" />
```

NavHostFragment utiliza un objeto NavGraph que definiremos en uno de los recursos XML de la aplicación, en el que se define el esquema de navegación de la aplicación. En este esquema se definen los fragments y sus transiciones que tendrán lugar en la aplicación.

NavGraph es una colección de nodos NavDestination, donde cada uno de los nodos identifica uno de los fragment de la aplicación. Estas dos clases pertenecen al paquete **androidx.navigation** y se deben incluir en el la ruta ***\app\src\main\res\navigation\mobile_navigation.xml.***

```
<navigation […]
    android:id="@+id/mobile_navigation"
    app:startDestination="@+id/nav_intro" […]
```

```
    <fragment
      android:id="@+id/nav_home"
      android:name="com.carlosdlp.appammforms.ui.home.
      HomeFragment"
      android:label="@string/menu_home"
      tools:layout="@layout/fragment_home">
    />
    <fragment […]
      tools:layout="@layout/fragment_preview" />
    <fragment […]
      tools:layout="@layout/fragment_editform"/>
    <fragment […]
      tools:layout="@layout/fragment_login"/>
    <fragment […] />
    <fragment […] /> […]
</navigation>
```

Por último, para poder gestionar toda la navegación se necesita hacer uso de la clase NavController del paquete **androidx.navigation.NavController**. Todo objeto NavHostFragment tiene este controlador que nos permite manejar las transiciones entre destinos. Las siguientes sentencias en Java se deben incluir dentro del controlador de la activity.

```
[…]//Obtener el controlador

NavController navController = Navigation.findNavController(this,
R.id.nav_host_fragment);

[…]//Obtener el destino (pantalla) actual

navController.getCurrentDestination()

[…]//Navegar al fragment de inicio de sesión

navController.navigate(R.id.nav_login);
```

5.1.6 Menú lateral desplegable

Podemos implementar esta funcionalidad con la clase NavigationView del paquete de Google **com.google.android.material.navigation.NavigationView**. Se debe incluir en el diseño XML de la activity, aunque al generar el proyecto a través del modelo "Navigation Drawer Activity" debe haberse incluido automáticamente. El objeto está compuesto por una cabecera y un menú que contendrá los destinos a los que

queremos navegar. La cabecera y el menú se definen en su propio archivo de diseño XML en el directorio de recursos.

```
<com.google.android.material.navigation.NavigationView
    android:id="@+id/nav_view"
    […]
    app:headerLayout="@layout/nav_header_main"
    app:menu="@menu/activity_main_drawer" />
```

Una vez diseñado el menú desplegable y todos sus componentes necesitaremos asociarlo con el controlador de navegación a través de la siguiente sentencia del controlador de la activity:

```
NavigationUI.setupWithNavController(navigationView,
navController);
```

5.1.7 Conectar con el servidor a través de Android Volley

Volley es una biblioteca HTTP que facilita y agiliza el uso de redes en apps para Android. Ofrece varias conexiones de red simultáneas, almacenamiento de respuestas en caché y en disco, cancelación de solicitudes, personalización sencilla y compatibilidad con strings sin procesar, imágenes y JSON. A pesar de esto, Volley no es adecuado para operaciones de transmisión o descarga grandes, ya que almacena todas las respuestas en la memoria durante el análisis.

Para utilizar el paquete **com.android.volley** tenemos que incluirlo en el archivo de compilación de Gradle. Una vez incluida la biblioteca volley, se crea una clase singleton[17] para gestionar la cola de solicitudes HTTP. La clase RequestQueue de la librería volley representa la cola de peticiones y se utiliza por la clase singleton a través de estos métodos:

```
[…]//Crear la cola de solicitudes
requestQueue =
Volley.newRequestQueue(context.getApplicationContext());
```

[17] Una clase singleton es un patrón de diseño en Java que impide que una clase pueda instanciarse más de una vez. De este modo tendremos una única cola de peticiones a la que accederemos a través de esta clase singleton.

```
[…]//Añadir una solicitud a la cola de solicitudes
getRequestQueue().add(req);
```

Para añadir una solicitud, el método utilizado recibe como parámetro un objeto Request. De la clase Request heredan una serie de clases que facilitan la solicitud a ciertos recursos del tipo imagen (ImageRequest), cadena de texto (StringRequest) o JSON (JsonRequest) y se encuentran en el paquete **com.android.volley.toolbox**.

Las solicitudes utilizan los datos característicos del protocolo HTTP como la URL, cabecera, método y parámetros. Además, necesitan las clases Response.Listener<t> y Response.ErrorListener que funcionarán como callback[18] y se ejecutarán cuando se reciba la respuesta del servidor.

Ejemplo de solicitud de imagen:

```
public class MyImageRequest extends ImageRequest {[…]
    public MyImageRequest(String url, Response.Listener<Bitmap>
  listener, int maxWidth, int maxHeight, ImageView.ScaleType
  scaleType, Bitmap.Config decodeConfig, Response.ErrorListener
  errorListener) {[…]}
    public void setHeaders(Map<String, String> headers) {[…]}
    public Map<String, String> getHeaders() {[…]}
}
```

Ejemplo de solicitud de objeto JSON:

```
public class ObjectRequest<T> extends JsonRequest<T>{[…]
    public ObjectRequest(int method, String url, Map<String,
  Object> requestParams, Class pClass, Response.Listener<T>
  responseListener, Response.ErrorListener errorListener) {[…]}
    public String getUrl() {[…]}
    public void setHeaders(Map<String, String> headers) {[…]}
    public Map<String, String> getHeaders() {[…]}[…]
  }
```

[18] Un Callback (llamada de vuelta) es una función que se ejecutará después de haber sucedido algún evento. Esto permite desarrollar capas de abstracción de código para implementar una comunicación asincrona con el servidor.

Los distintos fragments pueden generar solicitudes y procesar las respuestas para mostrar los resultados en la interfaz de usuario. Los fragments que utilicen las clases Volley deberán implementar una interfaz en la que se definen los callbacks de la respuesta del servidor web. Los datos de las respuestas son los argumentos de los métodos de la interfaz.

```
public interface VolleyListener {
    public void onResponseJSONObject(JSONObject response, String
requestName);
    public void onErrorResponse(VolleyError error, String
message, String requestName);
    public void onResponseImage(Bitmap response);
    public void onResponseString(String response);
}
```

Para utilizar las solicitudes se crea un objeto del tipo correspondiente y se añade un nuevo Response.Listener<T>. En el método onResponse, que se ejecutará cuando se recibe respuesta del servidor, se llama a uno de los métodos definidos en la interfaz para gestionar la información. Por último, se añade la solicitud a la cola de solicitudes.

```
MyImageRequest imageRequest = new MyImageRequest(url,
         new Response.Listener<Bitmap>() {
            public void onResponse(Bitmap response) {
                mVolleyListener.onResponseImage(response);
         }},
         0, 0, null, null, […]);
imageRequest.setHeaders(httpHeaders);
VolleySingleton.getInstance(mContext).addToRequestQueue(imageReq
uest, "my image request");
```

5.1.8 Obtener localización y dirección para editar el documento

En el fragment de edición de formulario se generan dinámicamente los campos en los que introducirá el usuario los datos para el documento. Estos campos pueden tener un tipo de dato asociado que facilita su recogida o pueden no tener ningún tipo específico y utilizar el teclado del dispositivo. En el caso del tipo de dato dirección, se pretende acceder a la localización del dispositivo y ofrecer la dirección en la que se encuentra.

En el método onCreateView() del fragment de edición se comprobarán los campos del documento y se generará la interfaz de usuario dinámica. Cuando se necesita un campo de dirección se utiliza la clase NewAutoEditText. Se trata de una vista de texto editable que muestra sugerencias mientras el usuario está escribiendo. El menú desplegable se completará con las direcciones posibles o más cercanas a la localización del dispositivo móvil. Esta clase utiliza un adaptador ArrayAdapter<String> que gestionará el menú para incluir las direcciones en formato cadena de texto.

```
final NewAutoEditText actv = new NewAutoEditText(context);
adapter = new ArrayAdapter<String>(context,
android.R.layout.simple_list_item_1, addressStrings);
actv.setAdapter(adapter);
```

Una vez hemos encontrado las direcciones cercanas a la localización del dispositivo las incluimos en el adapter de este modo:

```
public void onReceiveAddress(ArrayList<String> addressArray){
    addressAdapter.clear();
    addressAdapter.addAll(addressArray);
    adapter.notifyDataSetChanged();}
```

Para obtener la locación se hará uso de las clases **com.google.android.gms.location** y **com.google.android.gms.tasks**. Se accederá a la última localización que ha registrado el dispositivo, que en la práctica es obtener la dirección actual del dispositivo a no ser que nos encontremos en un punto sin cobertura. Se utiliza un listener para gestionar el resultado en formato Location:

```
Task<Location> locationTask =
        fusedLocationProviderClient.getLastLocation();
locationTask.addOnSuccessListener(new
OnSuccessListener<Location>() {
    public void onSuccess(Location location) {
        fetchAddressFromLocation(location);
    }
});
```

5.1.8.1 Geocodificación

Cuando se disponga de la localización, es el momento de utilizar la técnica de geocodificación para traducirla a una dirección. La codificación geográfica es el proceso de convertir direcciones (como "1600 Amphitheatre Parkway, Mountain View, CA") en coordenadas geográficas (como latitud 37.423021 y longitud -122.083739), que suele ser utilizado para colocar marcadores en un mapa. La codificación geográfica inversa es el proceso de convertir coordenadas geográficas en una dirección legible por humanos. Las plataformas como Google, OpenStreetMap o TomTom implementan este tipo de servicios. A la mayoría de los servicios de geocodificación se accede a través de internet, pero existen algunos sin conexión o incluso una implementación propia de geolocalización, aunque sería muy limitada. [36], [41]

Para implementar la función de geocodificación inversa se ha definido un servicio[19] que debe aparecer reflejado en el archivo ***AndroidManifest.xml***. En el servicio se utilizará la biblioteca de Android del paquete **android.location.Geocoder** para traducir la localización. La clase Geocoder requiere un servicio de backend[20] que no está incluido en el marco principal de Android. Los métodos de consulta de Geocoder devolverán una lista vacía si no hay un servicio de backend en el dispositivo. [37], [39]

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
List<Address> addressList =
geocoder.getFromLocation(location.getLatitude(),
location.getLongitude(), maxResults); […]
ArrayList<String> stringList = new ArrayList<>();
for(int i = 0; i < maxResults; i++) {
  Address address = addressList.get(i);
  stringList.add(address.getAddressLine(0));
}
Bundle bundle = new Bundle();
bundle.putStringArrayList("com.carlosdlp.appammforms.RESULT_DATA
_KEY", arrayList);
resultReceiver.send(resultCode, bundle);
```

[19] Un servicio es un componente de la aplicación Android que ejecuta sin interactuar con el usuario y proporciona funcionalidad para que la utilicen otras aplicaciones. Si el servicio va a realizar operaciones de uso intensivo de CPU, como reproducción de MP3, debería generar su propio hilo en el que realizar ese trabajo.

[20] Un backend es un proceso o aplicación que se ejecuta en segundo plano para resolver las peticiones de otra aplicación y es transparente tanto al usuario como al desarrollador.

Una vez se obtienen las direcciones, se expresan en formato lista de cadenas de texto en lugar de lista de direcciones y se utiliza un objeto ResultReceiver para acceder al método onReceiveAddress(ArrayList<String>) del fragment de edición.

Es posible que esta implementación tenga diferentes resultados dependiendo del dispositivo móvil en el que se ejecute, debido a que no todos disponen de mismos servicios instalados y es posible que utilicen otro servicio de backend. En la mayoría de los dispositivos este servicio será proporcionado por los servicios de Google Play del dispositivo. Por norma general, estaremos utilizando la API de Google Maps incluida en los servicios de Google, aunque se podría haber utilizado directamente la API de Google para un mejor rendimiento y control a pesar de ser un servicio de pago. Al utilizar la clase de Android estamos utilizando la API con la clave del dispositivo móvil y no la de desarrollador, lo que nos permite utilizar el servicio de forma gratuita, pero tendrá una limitación de solicitudes, de tal modo que se monitoreará activamente para detectar abusos (por ejemplo, solicitudes por segundo). La implementación no debería tener ningún problema ni limitación con el caso de uso de esta aplicación. [38], [40]

5.1.9 Fragment para firma electrónica

Para implementar la funcionalidad de firmar o dibujar en la aplicación como método de recogida de datos se deben gestionar los eventos de toque en la pantalla. Para ello se ha extendido la clase View para definir el área que escuchará el evento y dibujará el trazado de la firma.

En primer lugar, definimos el estilo que tendrá el trazo con la clase Paint de **Android.graphics**. El siguiente paso es sobre escribir el método onTouchEvent() de la clase View para recoger el movimiento del dedo sobre la pantalla y almacenarlo en un objeto Path. Se debe implementar también un método que elimine el dibujo sustituyendo el Canvas de la vista por uno en blanco. Por último, definimos el método onDraw() para dibujar el objeto Path.

```
public class SignatureView extends View {
    public SignatureView(Context context, AttributeSet attr,
SignatureFragment fragment) {
        path = new Path();
        brush = new Paint();
        brush.setColor(Color.BLACK); […]
    }
    public boolean onTouchEvent(MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                path.moveTo(event.getX(), event.getY());
                return true;
```

```
                case MotionEvent.ACTION_MOVE:
                    path.lineTo(event.getX(), event.getY());
                    break;
            }
            postInvalidate();
            return true;
        }
        protected void onDraw(Canvas canvas) {
            canvas.drawPath(path, brush);
        }
        public void clear(){[…]
            canvas.drawRect(0, 0, 0, 0, clearPaint);
            postInvalidate();
         }
    }
```

5.1.10 Previsualización

Para implementar esta funcionalidad se ha utilizado la clase WebView de la librería **android.webkit.WebView**. Permiten mostrar contenido web, pero carecen de algunas de las características de los navegadores convencionales. Utilizar esta clase ha condicionado de cierta forma el diseño de los documentos y su formato HTML. A pesar de esto, las ventajas que WebView ofrece son la renderización de los documentos y la exportación a PDF. Para renderizar el contenido del documento únicamente necesitamos utilizar esta sentencia:

```
 webview.loadDataWithBaseURL(null, doc.html(), "text/html",
"utf-8", null);
```

Para obtener el contenido del documento se genera una petición al servidor a través de la API. La respuesta devuelve el archivo HTML que contiene la estructura y el texto del documento y se gestionará a través de la biblioteca de **Java org.jsoup** para trabajar con HTML. Proporciona una API muy conveniente para manipular HTML5 y selectores CSS.

```
Document doc = Jsoup.parse(this.text);[…]
Elements inputtype = doc.select(id);
if (!data.equals("")) inputtype.html(data);[…]
webview.loadDataWithBaseURL(null, doc.html(), "text/html", "utf-
8", null);[…]
```

El contenido del documento se pasa como argumento en formato cadena de texto para crear el objeto jsoup. Para sustituir los campos del documento utilizamos la clase Elements de la librería jsoup. Por último, si queremos añadir la firma o alguna imagen utilizamos la siguiente sentencia.

```
doc.html().replace("IMAGE_PLACEHOLDER", signature);
```

La firma se envía al fragment de previsualización desde el fragment de firma a través de un Bundle en formato matriz de bytes y se codifica en Base64.

```
byte[] byteArray = getArguments().getByteArray("signature");
String imgageBase64 = Base64.encodeToString(byteArray,
Base64.DEFAULT);
signature = "" + imgageBase64;
```

5.1.11 Exportar y compartir

La clase WebView facilita la tarea de exportar a PDF el contenido que se está mostrando. Para ello crea una clase PrintDocumentAdapter que convierte el contenido del WebView en un flujo de PDF. Junto con esta clase se debe definir una serie de atributos en la clase PrintAtributes. Ambas clases se encuentran en la librería **Android.print**.

Se ha creado una nueva clase que implemente los métodos necesarios del adaptador y utilice los atributos de impresión. Además, se debe incluir la ruta en la que se guardará el archivo por lo que debemos incluir los permisos de almacenamiento en el manifest. Esta clase de debe incluir en la ruta ***\app\src\main\java\android\print\PdfPrint.java*** para que funcione correctamente. La implementación de esta clase llamada PdfPrint se ha obtenido de un artículo donde se documenta y explica su diseño. Con estas líneas de código se implementa la exportación a PDF. [42]

```
String jobName = getString(R.string.app_name) + " Document";
PrintAttributes attributes = new PrintAttributes.Builder()
        .setMediaSize(PrintAttributes.MediaSize.ISO_A4)
        .setResolution(new PrintAttributes.Resolution("pdf",
         "pdf", 600, 600)).[…].build();
String dirpath = context.getExternalFilesDir(
```

```
            Environment.DIRECTORY_DOCUMENTS).getAbsolutePath();
    String filename = new impleDateFormat("yyyyMMddHHmm'.pdf'").
            format(new Date());
    PdfPrint pdfPrint = new PdfPrint(attributes, context,this,
            dirpath + "/" + File.separator + filename, option);
    pdfPrint.print(webview.createPrintDocumentAdapter(jobName));
```

Otra funcionalidad que incluye la aplicación es la posibilidad de compartir el documento PDF exportado a través del correo electrónico. Para ello se utilizará la clase Intent con la información de la ruta en la que se encuentra almacenado y el cuerpo del mensaje. Un intent es una llamada a una activity o a un servicio tanto de la aplicación como otra aplicación instalada en el dispositivo. El intent busca aplicaciones que ofrezcan el servicio de mensajería. Existen algunas aplicaciones que ofrecen la posibilidad de incluir contenido HTML por lo que el documento se puede incluir en el cuerpo del mensaje.

5.1.12 Estilos y temas

En primer lugar, se deben haber diseñado los temas en el archivo correspondiente. En este caso se han extendido dos temas de la librería de Android para cambiar los colores principales y añadir otros elementos como el logo de la universidad que aparece en algunas partes de la aplicación. Theme.AppCompat.Light.DarkActionBar Y ThemeOverlay.AppCompat.Dark.ActionBar.

Los elementos de la interfaz que dependan de los estilos, como puede ser el color de la cabecera del menú desplegable, se les asigna un valor a través de un atributo definido en ***\res\values\attrs.xml***. El valor del atributo cambiará en función de qué tema se haya elegido utilizando AppCompatActivity.setTheme().

Ejemplo de cómo se define la imagen del logo de la universidad en la pantalla de introducción:

\res\values\styles.xml

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="umhLogo">@drawable/logo_umh </item>
</style>

<style name="DarkAppTheme"
parent="ThemeOverlay.AppCompat.Dark.ActionBar">
    <item name="umhLogo">@drawable/logo_umh_reversed</item>
</style>
```

\res\values\attrs.xml

```
<attr name="umhLogo" format="reference"/>
```

\res\layout\fragment_intro.xml

```
<ImageView
    app:srcCompat="?attr/umhLogo" />
```

5.1.13 Otras implementaciones

A continuación, se muestran otras clases e implementaciones utilizadas para el desarrollo de la aplicación:

-FAB: Se trata del botón redondo que aparece en la interfaz de usuario y con el que navegaremos entre los fragments de la aplicación a través del flujo principal. Puede cambiar de icono y modificar su visibilidad en función del fragment que nos encontremos. Servirá tanto para crear los documentos como para previsualizarlos, firmarlos o compartirlos. Pertenece a la librería **com.google.android.material.floatingactionbutton**.

-Barras de progreso: Se utilizan para notificar al usuario a través de la interfaz que la aplicación está realizando una operación en segundo plano. Se implementa a través de la clase **android.widget.ProgressBar**.

-Pantalla de ayuda: Se ha implementado sobre un objeto ViewPager2 incluido en el diseño del fragment de ayuda. Esta clase permite desplazarse entre fragments con un deslizamiento lateral sobre la pantalla. Los fragments únicamente cargarán un diseño predefinido con la información de ayuda a través de una modificación de la clase FragmentStateAdapter.

-Lista de documentos: Para implementar esta lista se ha utilizado la clase **androidx.recyclerview.widget.RecyclerView**. Se trata de una clase que permite generar listas dinámicas para un gran conjunto de datos. Estas listas ahorrar recursos ya que mantiene en memoria únicamente los ítems que se encuentran en posición de ser visualizadas. Junto con esta clase se deben utilizar las clases Adapter y ViewHolder extendiéndolas para ofrecer funcionalidades como la cargar automáticamente nuevos ítems desde el servidor.

Para la funcionalidad de carga automática se ha utilizado el método onBindViewHolder() de la clase ViewHolder que se llama cuando el contenido del ítem se carga dinámicamente en la lista. Cuando esto ocurre en el último elemento de la lista, que es de un tipo distinto del resto de los ítems, se lanza una solicitud al servidor.

-**Ventanas de dialogo**: Para ofrecer este recurso ya sea a la hora de establecer la fecha y hora en algún documento o para selección la opción de compartir se ha empleado la clase DialogFragment. Además, para crear los DialogFragment para introducir fecha y hora se ha utilizado la clase DatePickerDialog y TimePickerDialog.

5.2 Servidor con Laravel

Una vez instalado debidamente los paquetes de XAMPP y Composer en el servidor, se instalará Laravel. Para su instalación se ha seguido la documentación oficial del framework y algunos artículos web que pueden resultar de gran ayuda. A continuación, se muestra un ejemplo de los pasos y comandos que se deben utilizar para su configuración. [43], [44]

-Para descargar el instalador de Laravel a través de la consola de comandos:

```
composer global require laravel/installer
```

-Instalamos Laravel en la carpeta ***htdocs/*** de XAMPP a la vez que se genera el proyecto. Se utiliza el siguiente comando posicionándonos en la carpeta mencionada:

```
composer create-project --prefer-dist laravel/laravel tfgfirma
```

-Será necesario especificar los parámetros de la base de datos editando el archivo ***.env*** del directorio raíz del proyecto Laravel.

-Podemos evitar que los clientes accedan a la ruta http://servidor/firma haciendo uso del archivo de configuración de Apache ***.htaccess*** en el directorio ***htdocs/***. También se deben modificar los permisos de algunos directorios del proyecto laravel como ***storage/***.

-Para comprobar que toda la instalación ha sido correcta podemos acceder desde el navegador web a la ruta localhost/firma/public o http://servidor/firma/public

5.2.1 Autenticación con Laravel Passport

Para implementar la autenticación de la API utilizando el estándar OAuth2, Laravel ofrece un paquete que lo facilita. Con Laravel Passport se gestionan los accesos al servidor a través del Bearer Token como se muestra en el diseño. Passport está construido sobre el servidor League OAuth2 mantenido por Andy Millington y Simon Hamp. [45]

Para instalar este paquete en primer lugar solicitamos el paquete con el siguiente comando en la ruta raíz del proyecto.

```
composer require laravel/Passport
```

Para poder utilizar Passport en versiones inferiores a 5.4, como cualquier otro servicio, debemos añadirlo manualmente en el archivo de configuración ***config/app.php***.

El proveedor de servicios de Passport registra su propio directorio de migración de base de datos. Las migraciones de Passport crearán las tablas que su aplicación necesita para almacenar clientes y tokens de acceso.

```
php artisan migrate
```

Una vez ejecutado el comando la base de datos tendrá nuevas tablas que se utilizar internamente para la gestión del acceso, la única tabla que se utilizará en la aplicación Android será la tabla de usuarios. Las tablas que crea este comando son: failed_jobs, migrations, oauth_access_tokens, oauth_auth_codes, oauth_clients, oauth_personal_access_clients, oauth_refresh_tokens, users.

Se generan las claves de cifrado necesarias para generar tokens de acceso.

```
php artisan passport: install
```

Es el momento de configurar el proyecto para que trabaje con Passport. Agregamos **Laravel\Passport\HasApiTokens** al modelo de usuario ***App\User.php*** para Proporcionar algunos métodos auxiliares.

```
<?php
namespace App;[…]
use Laravel\Passport\HasApiTokens;
class User extends Authenticatable{
      use HasApiTokens, Notifiable;[…]}
```

Mediante Passport::routes() en el método de arranque de ***app/Providers/AuthServiceProvider.php*** se registrarán las rutas necesarias para emitir tokens de acceso y revocar tokens de acceso, clientes y tokens de acceso personal.

```
<?php
namespace App\Providers;
```

```
[…]use Laravel\Passport\Passport;
class AuthServiceProvider extends ServiceProvider{
   public function boot()    {[…]Passport::routes();}
}
```

Por último, en el archivo de configuración ***config/auth.php*** se debe establecer el driver de la API. Esto le indicará a su aplicación que use Passport Token para autenticar solicitudes entrantes a la API.

```
<?php
return [ […]
      'guards' => [[…]
          'api' => [
              'driver' => 'passport',
              'provider' => 'users',],
      ],[…]]
```

5.2.2 Implementación API

Para implementar la API se utilizarán principalmente el directorio ***app/HTTP/Controllers/API*** para definir el comportamiento del servidor y se modificará el archivo ***routes/api.php*** para establecer las diferentes rutas que representan cada uno de los recursos del servidor o de las acciones que solicita el cliente.

El archivo de rutas queda definido del siguiente modo:

```
<?php
[…]use Illuminate\Support\Facades\Route;
Route::post('login', 'API\UserController@login');
Route::post('register', 'API\UserController@register');
Route::group(['middleware' => 'auth:api'], function(){
      Route::resource('documents', 'API\DocumentController');
      Route::resource('image', 'API\ImageController');
```

```
        Route::resource('text', 'API\TextController');
        Route::post('details', 'API\UserController@details');
    });
```

Se incluyen las rutas de inicio de sesión y de registro a través del método Route::post(). En los argumentos se establece el nombre de la ruta y el controlador que definirá su comportamiento. Se hace referencia al controlador a través de la ruta del archivo dentro de la carpeta ***app/HTTP/Controllers/*** antes mencionada y se le asignará un método de dicho controlador por medio de "@". Se debe especificar el método del controlador siempre que la ruta se defina a través de un único método HTTP. De este modo si accedemos a la ruta http://servidor/firma/public/login, el servidor ejecutará el método login() del archivo ***app/HTTP/Controllers/API/UserController.php***.

El resto de las rutas necesitan autenticación para poder acceder. Esto se realiza a través del middleware que proporciona un mecanismo conveniente para filtrar las solicitudes HTTP que ingresan a su aplicación. El grupo de rutas estarán sujetas a autenticación por medio de Passport.

5.2.3 Crear tablas en la base de datos

Para crear nuevos recursos definimos un archivo de migración para dicho recurso.

```
php artisan make:migration create_documents_table
```

En el archivo que se ha generado en ***database/migrations/*** se define la estructura de la tabla documents:

```
<?php[…]
class CreateDocumentsTable extends Migration{
    public function up() {
      Schema::create('documents', function (Blueprint $table){
          $table->id();
          $table->string('title');
          $table->longtext('inputs');
          $table->string('text');
          $table->string('image');
```

```
            $table->timestamps();
        });}[…]}
```

Para ejecutar la migración se ejecuta el comando:

```
php artisan migrate
```

Laravel proporciona una implementación para trabajar con su base de datos. Cada tabla de la base de datos tiene un "Modelo" correspondiente que se utiliza para interactuar con esa tabla. Los modelos le permiten consultar datos en sus tablas, así como insertar nuevos registros en la tabla. Todos los modelos extienden la clase **Illuminate\Database\Eloquent\Model**.

El archivo ***app/Documents.php*** define el Modelo que gestionará la tabla que registra los documentos. Por convención, el nombre plural de la clase se utilizará como nombre de la tabla a menos que se especifique explícitamente otro nombre.

```
<?php[…]
class Document extends Model{
      protected $fillable = ['title','inputs','text','image'];
}
```

5.2.4 Controladores

Son los archivos que definirán el comportamiento que tendrá el servidor al ejecutar cada una de las rutas de la API por parte de la aplicación Android. Como se puede comprobar, el archivo de rutas registra un controlador para usuarios, documentos, textos e imágenes. Los controladores extienden la clase Controller.

5.2.4.1 UserController

El controlador de usuarios ***app/Http/Controllers/API/UserController.php*** gestiona las peticiones de inicio de sesión, registro y detalles de la API. La función login verifica al usuario a través de la clase Auth para luego generar el Token de acceso. Finalmente, construye la respuesta a través de un objeto Json con el Token y el código de respuesta 200.

```
public function login(){
      if(Auth::attempt(['email' => request('email'), 'password'
      => request('password')])){
            $user = Auth::user();
            $success['token'] =  $user->createToken('MyApp')->
            accessToken;
            return response()->json(['success' => $success],
            200);
      }[…]}
```

Para el registro debemos verificar que los parámetros de usuario cumplen una serie de requisitos básicos para evitar campos en blanco, que la contraseña y su confirmación sean diferentes o un formato email no válido. Una vez verificada la solicitud de registro, se encripta la contraseña y se crea un nuevo usuario a través del Modelo de usuario. Genera una respuesta con el nombre de usuario, el token y el código 200, al igual que en la función de inicio de sesión.

```
public function register(Request $request){
      $validator = Validator::make($request->all(), [ 'name' =>
      'required', 'email' => 'required|email', 'password' =>
      'required', 'c_password' => required|same:password',]);[…]
      $input = $request->all();
      $input['password'] = bcrypt($input['password']);
      $user = User::create($input);
      $success['token'] = $user->createToken('MyApp')->
      accessToken;
      $success['name'] =  $user->name;
      return response()->json(['success'=>$success], 200);
}
```

Por último, para solicitar los detalles del usuario se utiliza la siguiente función:

```
public function details() {
```

```
        $user = Auth::user();
        return response()->json(['success' => $user], 200); }
```

Los errores de autorización se envían junto con el código 401 si existe algún error en el registro o el login a través de la siguiente línea. No es necesario utilizar esta línea con la función details() ya que esta ruta se encuentra dentro del grupo de rutas que necesitan autorización en el archivo ***routes/api.php***:

```
return response()->json(['error'=>'Unauthorised'], 401);
```

5.2.4.2 DocumentController

El controlador encargado de gestionar los accesos a la tabla documents de la base de datos. Para las rutas, que en el archivo ***routes/api.php*** se definen como recursos a través de Route::resource(), existen unos métodos específicos que se ejecutan para cada uno de los HTTP Verbs.

- GET […]firma/public/documents -> index()
  Utilizado para obtener un listado de los recursos.
- GET […]firma/public/documents/id -> show($id)
  Utilizado para obtener recursos según su identificador.
- POST […]firma/public/documents -> store(Request $request)
  Genera nuevos recursos con la información de los parámetros de solicitud.

En el controlador de la ruta document se utilizarán las funciones ya definidas para implementar las llamadas a la API. A través de la función index() solicitamos los 20 primeros documentos que se mostrarán la lista de selección de la aplicación.

```
public function index()  {
    $documents = Document::where('id', '>', 0)->take(20)->get();
    return $this->
sendResponse(DocumentResource::collection($documents),
'Products retrieved successfully.');
  }
```

Mediante la función show($id) solicitamos los siguientes 20 documentos que se mostrarán en la lista al deslizar hacia abajo.

```
  public function show($id)  {
    $documents = Document::where('id','>',$id)->take(20)->get();
    return $this->
sendResponse(DocumentResource::collection($documents), 'Products
retrieved successfully.');
  }
```

Con la función store(Request $request) gestionaremos la búsqueda de documentos por nombre a pesar de que el método esté diseñado para crear nuevos recursos según la arquitectura RESTfull.

```
public function store(Request $request)  {
      $documents = Document::where('title', 'LIKE', "%$request-
      >title%")->where('id','>', $request->id)->take(20)->get();
      return $this->
      sendResponse(DocumentResource::collection($documents),
      'search');
}
```

Podemos comprobar cómo en la respuesta a través del método sendResponse se utiliza la clase DocumentResource para presentar los resultados. Se trata de una capa de transformación que se encuentra entre los modelos Eloquent y las respuestas JSON que recibe el usuario. La clase se define en ***app/Http/Resources/Document.php*** y extiende la clase **Illuminate\Http\Resources\Json\JsonResource**.

5.2.4.2 Controlador de imágenes y texto

Se trata de un controlador muy sencillo que ofrecerá los archivos de imagen y HTML que utilizará la aplicación Android para previsualizar el documento. Dado que el nombre de los archivos es un entero, se puede acceder a través del método show($id).

```
class TextController extends Controller{
  public function show($id){
```

```
    return response()->
download(public_path('documents/'.$id.'.html'), 'text');}}


class ImageController extends Controller{
  public function show($id){
    return response()->
download(public_path('images/'.$id.'.jpg'), 'image');}}
```

5.2.5 Portal Web

Se ofrecerá una vista de creación de documentos para acceder desde un navegador web. En primer lugar, se debe crear una vista web a través del motor de plantillas Blade. Se ha utilizado la vista de inicio ***resources/views/welcome.blade.php*** que trae por defecto Laravel en el directorio **resources/views**. Se ha modificado para que contenga el formulario HTML que se ha diseñado en el capítulo anterior.

La ruta que gestionará la creación del documento se debe definir en el archivo de rutas del portal web en ***routes/web.php***. Este archivo gestiona los accesos al servidor por medio de la interfaz web. Para solicitar la creación de un documento se necesita autenticación por lo que se establece el middleware en la ruta.

```
Route::post('fileUpload', [
      'as'=>'document.add',
      'uses'=>'HomeController@fileUpload'
])->middleware('auth');
```

La ruta invoca uno de los métodos del controlador que será necesario definir. El controlador se define en el archivo ***app/Http/Controllers/HomeController.php***. El método crea un nuevo recurso en la base de datos y almacena tanto el texto como la imagen en el sistema de directorios del servidor. El nombre de los archivos de imagen y texto se generará en función de la fecha de creación, por lo que puede ser representado por un numero entero en la base de datos y podrá ser utilizado como índice en la ruta de la API.

En primer lugar, se validan los datos introducidos en el servidor para que ningún campo se envíe en blanco y que los archivos tengan el formato adecuado. Una vez se

ha verificado la solicitud, el servidor nombra los documentos con la marca de tiempo y los almacena en la carpeta publica ***public/documents*** o ***public/images***. Después de almacenar los archivos, se crea un nuevo recurso a través del Modelo Document. Haciendo uso del recurso Json para este modelo, enviamos una respuesta para indicar que ha sido satisfactoria la creación del documento en la base de datos.

```
public function fileUpload(Request $request){[…]

      $this->validate($request, ['title' => 'required', 'inputs'
      => 'required', 'image' => 'required|mimes:jpg', 'document'
      => 'required|mimes:html']);[…]

      $document = $request->file('document');

      $documentname = time();

      $document->move(public_path('/documents'),
$name.'.'.$document->getClientOriginalExtension());[…]

      $doc_resource = Document::create(['title' => $request->
input('title'), 'inputs' => $request->input('inputs'),'text' =>
$documentname, 'image' => $imagename]);

      return $this->sendResponse(
new DocumentResource($doc_resource), 'Product created
successfully.'); }
```

En el código de ejemplo solo se muestra cómo se almacena el texto del documento y no la imagen. Para almacenar la imagen se procede del mismo modo que con el texto.

6. Resultados y discusión

La aplicación cumple con los requisitos establecidos al comienzo del informe, aunque, como se ve en el siguiente capítulo, la aplicación necesita mejorar en algunos apartados. A pesar de que se han realizado distintas pruebas para comprobar el funcionamiento de la aplicación, no se han realizado pruebas de rendimiento en diferentes dispositivos para conocer sus limitaciones. Con las pruebas que se han realizado, tanto dentro de una red WIFI como en el exterior con la tarifa de datos del dispositivo móvil, demuestran que la aplicación funciona correctamente y se ha logrado cumplir con el diseño y la implementación del proyecto.

A parte de las mejoras que se deben realizar para completar la aplicación, existe el riesgo de que la implementación de ayuda de locación no funcione en todos los dispositivos. Esto es debido a que la funcionalidad de geocodificiación necesita un servicio de backend que ofrezca traducción entre localización y dirección. Si el usuario no dispone de este servicio, no podrá hacer uso de esta funcionalidad. Podría incluirse en la aplicación un aviso o mensaje informando sobre este aspecto, aunque la mayoría de los dispositivos Android disponen de los servicios de Google Play instalados en los que se encuentra este servicio. A parte de esta cuestión, se deberán tener en cuenta otro tipo de factores a la hora de utilizar esta funcionalidad como tener conexión a internet o habilitar el uso de la localización del dispositivo.

6.1 Demostración de la aplicación

Para mostrar el funcionamiento de la aplicación se ha escogido un caso de uso concreto. Queremos rellenar un parte amistoso de accidentes a través de la interfaz de la aplicación.

En primer lugar, necesitamos diseñar el documento en HTML. El documento tendrá la imagen del parte amistoso de fondo y sobre ella se colocarán los campos del documento. Es una de las posibles formas de redactar el documento, con el inconveniente de la limitación de resolución, demostrando que es un método de redacción versátil que nos permitirá redactar desde contratos hasta facturas:

```
<html>
<head></head>
<body>
<div style="position:absolute;left:0px;top:0px">
<img
src="http://servidor/firma/public/images/parte_amistoso.png"
width="820" height="1050"></div>
<div style="position:absolute;left:25px;top:49px"
class="cls_004"><span class="cls_004"><date></date></span></div>
```

```
<div style="position:absolute;left:152px;top:49px"
class="cls_004"><span class="cls_004"><time></time></span></div>
<div style="position:absolute;left:235px;top:49px"
class="cls_004"><span class="cls_004"><pais></pais></span></div>
<div style="position:absolute;left:310px;top:35px;
width:155px;height:25px;overflow:hidden"
class="cls_004">        &#1
60 <location></location></div>
<div style="position:absolute;left:70px;top:163px"
class="cls_004"><name1></name1></div>
<div
style="position:absolute;left:575px;top:163px;width:220px;height
:15px;overflow:hidden" class="cls_004"><span
class="cls_004"><name2></name2></span></div>
<div
style="position:absolute;left:70px;top:191px;width:220px;height:
15px;overflow:hidden" class="cls_004"><span
class="cls_004"><location1></location1></span></div>
<div
style="position:absolute;left:575px;top:191px;width:220px;height
:15px;overflow:hidden" class="cls_004"><span
class="cls_004"><location2></location2></span></div>
<div style="position:absolute;left:86px;top:207px"
class="cls_004"><span class="cls_004"><cp1></cp1></span></div>
<div style="position:absolute;left:588px;top:207px"
class="cls_004"><span class="cls_004"><cp2></cp2></span></div>
<div style="position:absolute;left:81px;top:223px"
class="cls_004"><span class="cls_004"><tel1></tel1></span></div>
<div style="position:absolute;left:584px;top:223px"
class="cls_004"><span class="cls_004"><tel2></tel2></span></div>
<div style="position:absolute;left:250px;top:730px"><img
src="SKETCH_PLACEHOLDER"width="350" height="230"></img></div>
<div style="position:absolute;left:230px;top:923px"><img
src="SIGNATURE_PLACEHOLDER"width="200" height="200"></img></div>
<div style="position:absolute;left:408px;top:923px"><img
src="SIGNATURE_PLACEHOLDER"width="200" height="200"></img></div>
</body>
</html>
```

Una vez redactado el documento se sube al servidor a través del formulario de la web. En el formulario deberán especificarse los campos del siguiente modo y opcionalmente se podrá añadir una imagen de cabecera:

```
[{"type": "date", "id": "date", "hint": "fecha", "data": ""},
{"type": "time", "id": "time", "hint": "hora", "data": ""},
{"type": "text", "id": "pais", "hint": "país", "data": ""},
{"type": "location", "id": "location", "hint": "location",
"data": ""},
{"type": "text", "id": "name1", "hint": "nombre 1", "data": ""},
{"type": "text", "id": "name2", "hint": "nombre 2", "data": ""},
{"type": "location", "id": "location1", "hint": "location1",
"data": ""},
```

```
{"type": "location", "id": "location2", "hint": "location2",
"data": ""},
{"type": "text", "id": "cp1", "hint": "código postal 1", "data":
""},
{"type": "text", "id": "cp2", "hint": "código postal 2", "data":
""},
{"type": "text", "id": "tel1", "hint": "tel 1", "data": ""},
{"type": "text", "id": "tel2", "hint": "tel 2", "data": ""}]
```

Este documento podrá seleccionarse desde la pantalla de selección de la aplicación después de que la aplicación haya solicitado los documentos al servidor. Una vez seleccionado se accederá a la ventana de edición. En la ventana de edición comprobamos el funcionamiento de la ayuda de localización que se ha utilizado desde uno de los edificios del campus de Elche.

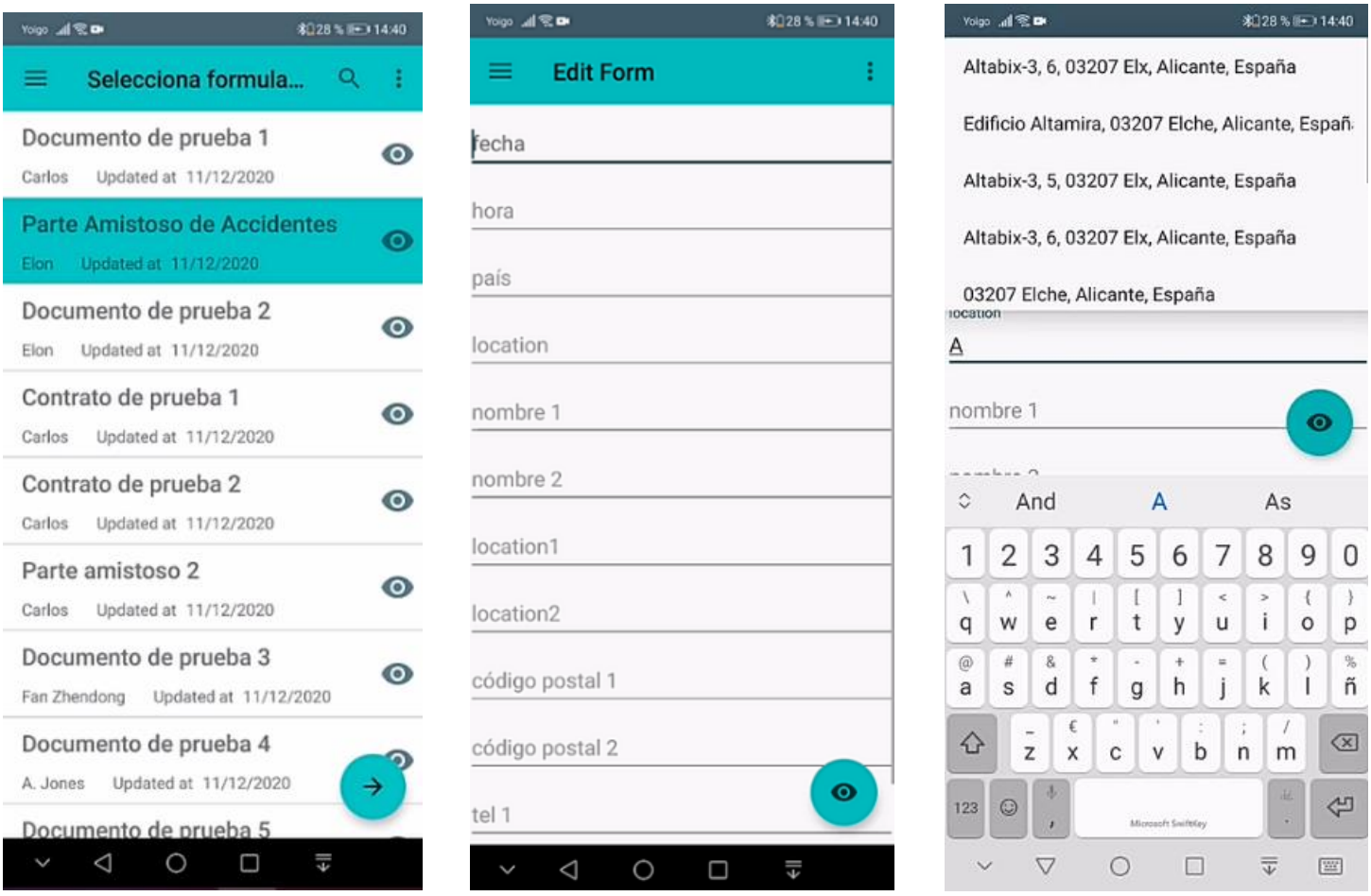

ILUSTRACIÓN 18. DEMOSTRACIÓN DE LA APLICACIÓN 1

Una vez se hayan completado los campos podemos renderizar el texto que subimos anteriormente con los campos rellenados. Una vez firmado el documento quedará completado y listo para ser exportado o compartido.

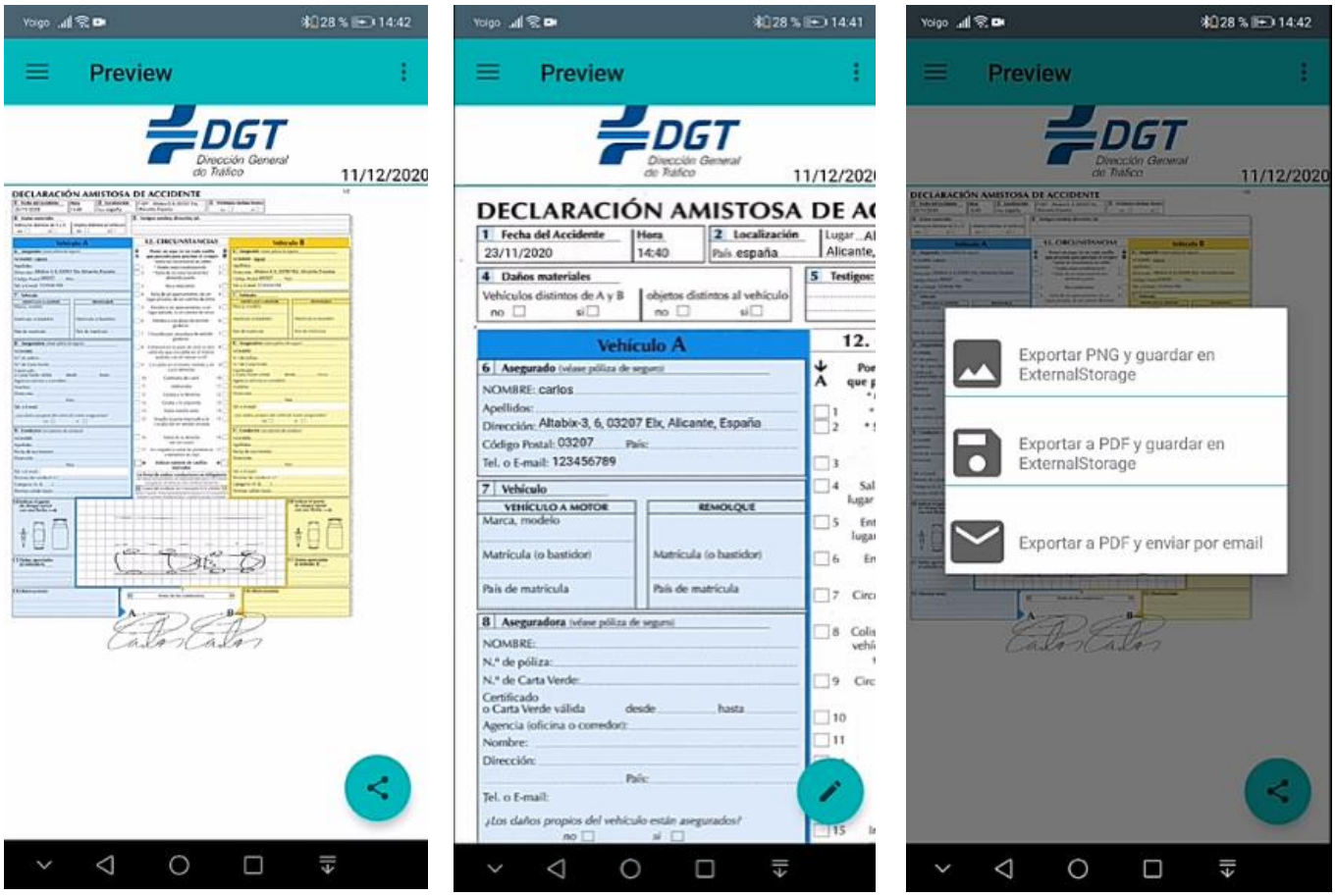

ILUSTRACIÓN 21. DEMOSTRACIÓN DE LA APLICACIÓN 2

7. Propuestas de mejora

El proyecto en general puede mejorarse en la mayoría de sus apartados tanto en la implementación de nuevas funcionalidades como en la depuración del código utilizado. Estas mejoras no forman parte de los objetivos del proyecto, que es capaz de ofrecer una interfaz móvil de edición de documentos que se comunica con un servidor a través de autenticación.

A pesar de esto, existe un gran número de aspectos de la aplicación que deben mejorarse para completarla y mejoras que ofrezcan un mayor recorrido al proyecto como aplicación comercial. Destaca la sencillez del diseño del servidor y de los recursos que gestiona. No es en realidad un mal diseño si cumple los requisitos establecidos, pero necesita implementar una serie de mejoras. La mayoría de las mejoras que se mencionan se pueden implementar de manera sencilla utilizando las herramientas de desarrollo del proyecto.

- Modificar el diseño e implementación del servidor ya que el modo en el que se almacenan los campos puede tener limitaciones impidiendo que los documentos tengan un gran número de campos. En lugar de almacenarlos en uno de los atributos de la tabla documents, pueden registrarse en un documento JSON a parte y referenciarse del mismo modo que con el texto y la imagen. Además, se deberá modificar la API para incluir una ruta específica para los campos, que devuelva únicamente los campos del documento seleccionado.

- Mejorar en el apartado de seguridad del servidor. El servidor debe actualizarse a HTTPS[21]. Además, debe establecer un directorio más seguro para almacenar los documentos que suben los usuarios al servidor. En definitiva, se debe establecer un periodo de estudio del diseño e implementación del servidor para encontrar fallos de seguridad y resolverlos.

- Configurar el servidor una caché de puerta de enlace (o proxy inverso)[22]. Con esto se ganará velocidad y se reducirá la carga del servidor. También se deberá gestionar la caché del dispositivo móvil utilizando la librería Volley que implementa estas funcionalidades.

[21] HTTPS es una forma segura del protocolo HTTP, en la que la comunicación HTTP se envuelve en la capa cifrad Transport Layer Security (TLS). [46]
[22] La caché en el lado del servidor almacena los recursos del servidor para reducir los tiempos de respuestas de futuras solicitudes al recurso hasta que sea modificado. [47]

- Implementar nuevas rutas para la API que ofrezcan funcionalidades como cierre de sesión, cambio de contraseña o solicitar información de otros usuarios.

- Implementar un sistema de gestión de documentos ya rellenados, así como la lógica de generación de token de validación a través de la información del cliente. Por el momento, cuando la aplicación finaliza el proceso de recopilación de datos y firma, se envía un email con una copia en PDF y un enlace de confirmación. Este enlace no se ha generado a través de ningún proceso por lo que no daría validez a la firma digital, pero si ayuda a tener una idea de cómo el cliente percibe el proceso de validación.

- Realizar un proceso de testeo de la funcionalidad de geocodificación inversa utilizada en la aplicación móvil. Al utilizar la clase que ofrece Android los resultados pueden variar dependiendo del dispositivo tal y como se ha explicado anteriormente en este proyecto. Se deberá recurrir a otro servicio de geocodificación si la aplicación no satisface la demanda de los usuarios.

- Mejorar el diseño de pantallas para ofrecer interfaz más completa. La pantalla Home está incompleta y las pantallas de selección y edición de documentos podrían ofrecer más detalles. Sobre todo, mejorar la pantalla de edición ya que si trabajamos con documentos con muchos campos el usuario puede encontrar incomoda la interfaz. Podría implementarse un sistema de páginas en las que distribuir los campos del documento.

- Los métodos para compartir el documento final pueden no ser suficientes o estar mal enfocados. No existe ningún método para volver a subir el resultado en el servidor y cuando se comparte el documento por medio de correo electrónico se podría automatizar los parámetros de la cabecera. En principio, para determinar los métodos para compartir primero se debe analizar qué clientes va enfocada esta aplicación.

- Definir los clientes hacia los que va dirigido la aplicación. Es una de las tareas más importantes que se debe hacer, ya que muchos de los métodos de la aplicación se deberán modificar o implementar en función del uso que se le quiera dar:
    - En el caso de diseñar la aplicación para su uso cotidiano la aplicación no necesitaría grandes cambios.
    - Si por el contrario se pretende diseñar como aplicación de uso profesional, deberán mejorarse las funcionalidades del servidor para que los usuarios tengan una mayor gestión de los recursos del servidor y

poder compartir los documentos. Podría dar soporte a algunas empresas que quieran gestionar sus documentos a través de la aplicación.

8. Conclusiones

El desarrollo de este proyecto ha sido un trabajo que ha puesto a prueba los conocimientos adquiridos a lo largo del grado. Además, he adquirido nuevos conocimientos para el desarrollo web y de aplicaciones durante el proceso de diseño e implementación de la aplicación.

A pesar de que a la aplicación le faltan mejoras para considerarse una aplicación completa, me satisface el resultado final no solo por haber adquirido conocimientos y destrezas como desarrollador, sino por haber cumplido con los objetivos y requisitos del proyecto.

Se ha realizado un estudio previo de la herramienta de diseño web Laravel, recomendada por parte del tutor del trabajo, aunque todavía me queda mucho por aprender de esta herramienta. También se ha realizado un trabajo de recopilación de información sobre distintos ámbitos como la validez de la firma digital o los distintos servicios de geocodificación, los servicios de Google y más concretamente GoogleMaps, sus limitaciones y sistemas de subscripción que ofrecen.

Los objetivos del proyecto los considero cumplidos ya que se ha conseguido desarrollar una aplicación funcional, pero todavía hace falta un proceso de testeo, depuración y rediseñar algunas partes de la aplicación si fuera necesario.

Referencias

[1] Statista, Number of smartphone users from 2016 to 2021.

URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] StatCounter, Mobile Vendor Market Share in Spain - November 2020.

URL: https://gs.statcounter.com/vendor-market-share/mobile/spain

[3] StatCounter, Mobile Vendor Market Share in Spain - November 2020.

URL: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[4] AppBrain (December 4, 2020), Android app download statistics on Google Play.

URL: https://www.appbrain.com/stats/android-app-downloads

[5] Avinash Sharma (September 7, 2020), Top Google Play Store Statistics 2019-2020 You Must Know.

URL: https://appinventiv.com/blog/google-play-store-statistics

[6] MDN Web Docs (August 7, 2020), Generalidades del protocolo HTTP.

URL: https://developer.mozilla.org/es/docs/Web/HTTP/Overview

[7] Android Developer

URL: https://developer.android.com/

[8] Laravel - The PHP Framework for Web Artisans

URL: https://laravel.com/

[9] Academia Android (August 25, 2015), Aplicaciones cliente-servidor y redes de telefonía móvil

URL: https://academiaandroid.com/aplicaciones-cliente-servidor-y-redes-de-telefonia-movil/

[10] IBM Knowledge Center (February 19, 2019), Conceptos clave: entidad, atributo y tipo de entidad.

URL: https://www.ibm.com/support/knowledgecenter/es/SSWSR9_11.6.0/com.ibm.mdmhs.overview.doc/entityconcepts.html

[11] Leyre Soto (May 04, 2020), El Blog de Signaturit - Firma electrónica avanzada, simple o cualificada, ¿sabes distinguirlas?

URL: https://blog.signaturit.com/es/firma-electronica-simple-vs-avanzada

[12] Android Developer (November 25, 2020), Notas de la versión de Android Studio.

URL: https://developer.android.com/studio/releases/index.html

[13] Android Developer (Nobemver 03, 2020), Configura tu compilación.

URL: https://developer.android.com/studio/build

[14] Gradle

URL: https://gradle.org/

[15] Android Source (December 01, 2020), Nombres internos, etiquetas y números de compilación.

URL: https://source.android.com/setup/start/build-numbers

[16] StatCounter, Mobile & Tablet Android Version Market Share Worldwide - November 2020.

URL: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[17] Maks Surguy (July 27, 2013), History of Laravel PHP framework, Eloquence emerging

URL: https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/

[18] Laravel, Eloquent: Getting Started.

URL: https://laravel.com/docs/8.x/eloquent

[19] Laravel, Blade Templates.

URL: https://laravel.com/docs/8.x/blade

[20] Laravel, Installation.

URL: https://laravel.com/docs/8.x/installation

[21] Laravel, Database: Getting Started.

URL: https://laravel.com/docs/8.x/database

[22] Graham Campbell (Jan 25, 2020), Laravel-GitHub/LICENSE.

URL: https://github.com/laravel/framework/blob/8.x/LICENSE.md

[23] Apache Friends

URL: https://www.apachefriends.org/es/index.html

[24] PHP, ¿Qué es PHP?

URL: https://www.php.net/manual/es/intro-whatis.php

[25] Héctor Patricio (May 6, 2019), Diseño y desarrollo de una API RESTful desde cero - La importancia de diseñar tu API

URL: https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[26] Creación de una API.

URL: https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html

[27] MDN Web Docs (March 23, 2019), Métodos de petición HTTP.

URL: https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[28] OAuth, OAuth 2.0.

URL: https://oauth.net/2/

[29] Oauth, RFC 6750: Uso del token de portador de OAuth 2.0.

URL: https://oauth.net/2/bearer-tokens/

[30] Roger A. Grimes and Josh Fruhlinger (September 20, 2019), What is OAuth? How the open authorization framework works

URL: https://www.csoonline.com/article/3216404/what-is-oauth-how-the-open-authorization-framework-works.html

[31] NFON AG, Request for Comments (RFC).

URL: https://www.nfon.com/es/servicio/base-de-conocimiento/base-de-conocimiento-destacar/request-for-comments-rfc

[32] REGLAMENTO (UE) No 910/2014 DEL PARLAMENTO EUROPEO Y DEL CONSEJO

de 23 de julio de 2014

URL: https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32014R0910

[33] Android Developer (September 30, 2020), Activity.

URL: https://developer.android.com/reference/android/app/Activity

[34] Android Developers (October 28, 2020), Descripción general del manifiesto de una app.

URL: https://developer.android.com/guide/topics/manifest/manifest-intro

[35] Android Developers (December 27, 2019), Fragmentos

URL: https://developer.android.com/guide/components/fragments

[36] Stack Overflow, Android reverse geocoding requires either an Internet connection or a backend data provider

URL: https://stackoverflow.com/questions/30775953/android-reverse-geocoding-requires-either-an-internet-connection-or-a-backend-da

[37] Stack Overflow, Android Geocoder quota limits.

URL: https://stackoverflow.com/questions/8218764/android-geocoder-quota-limits

[38] Stack Overflow, Android Geocoder behaves different on some devices.

URL: https://stackoverflow.com/questions/47070264/android-geocoder-behaves-different-on-some-devices

[39] Android Developers (September 09, 2020), Geocoder.

URL: https://developer.android.com/reference/android/location/Geocoder

[40] Google Developers (December 08, 2020), Google Maps Platform FAQ.

URL: https://developers.google.com/maps/faq

[41] Google Developers (Nobember 19, 2020), Geocoding

URL: https://developers.google.com/maps/documentation/geocoding/overview

[42] Annalytics (April 06, 2017), Save (Print) a PDF from an Android WebView

URL: http://www.annalytics.co.uk/android/pdf/2017/04/06/Save-PDF-From-An-Android-WebView/

[43] Urjit Rajgor (December 29, 2017), Create REST API in Laravel with authentication using Passport.

URL: https://medium.com/techcompose/create-rest-api-in-laravel-with-authentication-using-passport-133a1678a876

[44] Hamza Ali (Juny 15, 2018), Laravel Passport – Create REST API With Authentication.

URL: https://tutsforweb.com/laravel-passport-create-rest-api-with-authentication/#Create_Product_CRUD

[45] Laravel, Laravel Passport

URL: https://laravel.com/docs/8.x/passport

[46] Google Developers, Proteger sitios web con el protocolo HTTPS

URL: https://developers.google.com/search/docs/advanced/security/https?hl=es

[47] Apache, Guía de proxy inverso.

URL: https://httpd.apache.org/docs/trunk/es/howto/reverse_proxy.html