UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA POLITÉCNICA SUPERIOR DE ALCOY

TRABAJO FIN DE GRADO

Aplicación Android para la gestión de comunidades de vecinos

Autor: Ángel Calles Díaz

Tutor: Javier Esparza Peidro

ALCOY, JUNIO DE 2020

Aplicación Android para la gestión de comunidades de vecinos

Resumen

El siguiente trabajo final de grado consiste en una aplicación móvil, concretamente destinado para la plataforma Android, con la intención de simplificar y facilitar las tareas propias de la gestión de comunidades de vecinos.

Principalmente, la aplicación pretende mantener comunicación entre todos los integrantes de la comunidad y lograr gestionar aspectos como las incidencias, reparaciones, los pagos comunitarios e incluso mantener constancia de todas las reuniones que se vayan a producir.

Para el desarrollo de la aplicación “ComuniTools” se ha empleado Android Studio, que es el entorno de desarrollo (IDE) integrado oficial de la plataforma Android. Además, se ha utilizado la plataforma Firebase de Google, tanto para gestionar la base de datos como para mejorar la comunicación con los usuarios de la aplicación.

Palabras clave

app, android, dispositivos móviles, smartphones, java, xml, gestión, vecinos, administración, comunicación, comodidad.

Abstract

The next final degree work consists of a mobile application, specifically designed for the Android platform, with the intention of simplifying and facilitating the tasks inherent in the management of neighborhood communities.

Mainly, the application aims to maintain communication between all members of the community and manage aspects such as incidents, repairs, community payments and even keep a record of all meetings to be held.

For the development of the application "ComuniTools" Android Studio has been used, which is the official integrated development environment (IDE) of the Android platform. In addition, Google's Firebase platform has been used, both to manage the database and to improve communication with the application's users.

Keywords

app, android, mobile devices, smartphones, java, xml, management, neighbors, administration, communication, comfort.

Índice de contenidos

1. Introducción.

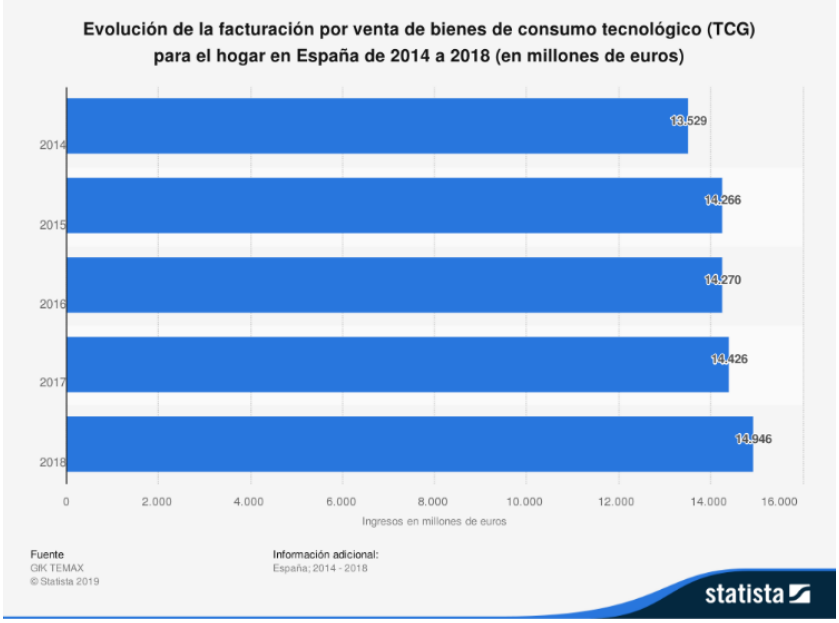
1.1. Motivación.

La gestión de fincas juega un papel fundamental para lograr una buena marcha de una comunidad de vecinos. Por ello, gestionarla de una forma correcta implica un buen estado, tanto a nivel material, como legislativo e incluso humano, ya que se generará un ambiente positivo entre los vecinos, aumentando su bienestar dentro de la finca.

Además, si nos encontramos ante una comunidad mal gestionada se reducen las posibilidades de entrada de nuevos propietarios e incluso la propiedad puede bajar considerablemente su valor, potenciando una gran debilidad para toda la finca.

Un gestor de fincas debe realizar todas las tareas de forma correcta, se trata de un papel imprescindible y muy exigente, generando en él la necesidad de una herramienta para optimizar todas sus acciones y tener constancia en todo momento de hasta el más mínimo detalle, por ello se presenta la aplicación móvil “ComuniTools”.

Debido al crecimiento del consumo de tecnología a nivel mundial, es una buena oportunidad crear dicha herramienta adaptada a los smartphones. En la siguiente imagen se muestra un gráfico que demuestra el incremento mencionado anteriormente.



Extraída de statista.com

1.2. Estudio del mercado.

Para realizar un estudio del mercado donde nuestra aplicación va a entrar es imprescindible conocer varios factores clave que pueden llevar al éxito la herramienta “ComuniTools”. Una forma muy efectiva para conocer el sector es emplear las 5 fuerzas de Porter, ya que nos brindará información muy relevante para competir con aplicaciones similares e incluso superarles. En la siguiente imagen 3 se muestran éstas 5 fuerzas que vamos a aplicar a nuestro caso.

Extraída de [5fuerzasdeporter.com]

- **Poder de negociación de los clientes.**

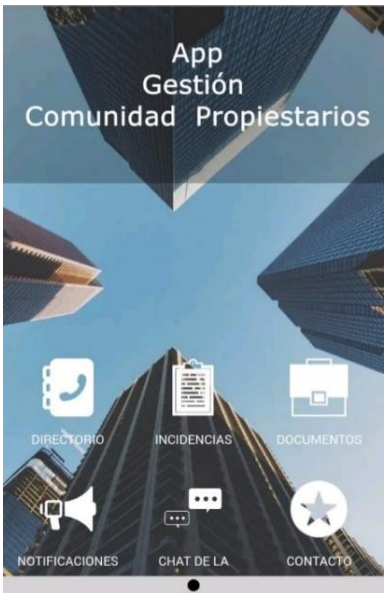
Primeramente, es conveniente considerar cuál es el público hacia el que nos dirigimos, ya que hoy en día poseen un gran poder sobre las empresas a la hora de llegar a acuerdos en los precios de los productos. Gracias al gráfico de la imagen que se muestra a continuación hemos podido observar que el perfil de usuario medio es de personas que oscilan desde los 25 hasta los 44 años, por lo que nos orientamos a clientes que en su mayoría poseen una responsabilidad en la comunidad de vecinos, ya sea como propietario e incluso como presidente o secretario. Es primordial conocer cómo la aplicación va a llegar a más comunidades de vecinos, y como métodos principales de expansión actualmente se encuentran las recomendaciones entre amigos y familiares y la aparición en los principales buscadores.

Extraída de [ignaciosantiago.com]

- **Rivalidad entre las empresas.**

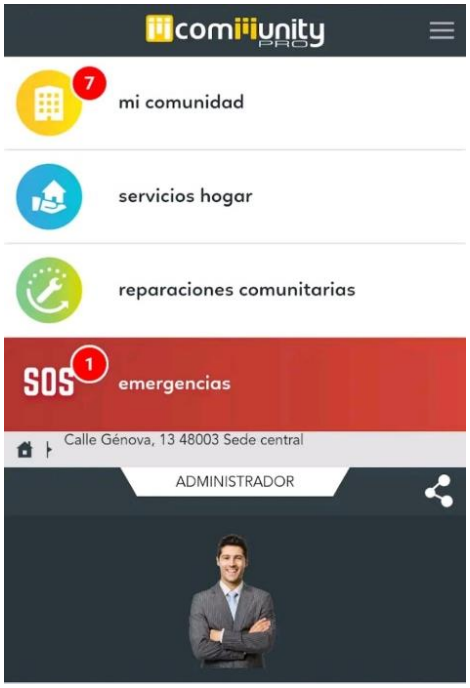
Una vez conocemos el perfil de usuario que va a emplear nuestra herramienta, es imprescindible descubrir con qué otras aplicaciones similares vamos a competir. Tras realizar búsquedas variadas en la Play Store de Google, podemos destacar la presencia de 5 aplicaciones destacadas dedicadas a la gestión de comunidades de vecinos:

✓ **Comunidad de vecinos del grupo FmoFreelancer.**



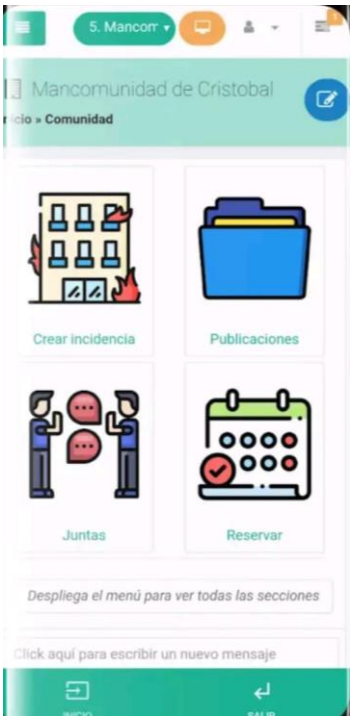
✓ **Fynkus administración de fincas del grupo Fintech Driver.**

✓ **Mi Comunidad del grupo Community Pro.**



✓ **Plusvecinos App del grupo Possible Dev Team.**

✓ **Colindar App del grupo Colindar.**



Tras realizar el testeo de todas estas, por lo general, destacan por lo intuitivas que resultan y la sencillez que ofrecen a la hora de emplearla, pero hay que destacar los graves errores que generan Mi Comunidad y Plusvecinos, ya que ambas presentan problemas en el acceso de los usuarios, hecho que pueden llevar a los clientes a dejar de utilizarlas directamente. Los errores que se muestran en otras aplicaciones se pueden tomar como referencia para evitar futuros problemas en el desarrollo de nuestra aplicación.

- **Amenaza de los nuevos competidores.**

Es de vital importancia tener en cuenta la posible entrada de nuevos competidores, ya que el nicho de las aplicaciones de gestión no está muy explotado y podría dar pie a empresas desarrolladoras de aplicaciones para realizar un proyecto con características similares. En España hay una gran cantidad de empresas cuya labor se centra en el desarrollo de aplicaciones en Android, por lo que en cualquier momento pueden introducirse en nuestro mercado.

- **Productos sustitutivos.**

En el sector en el que nos movemos se producen cambios constantes, ya que de forma periódica entran en el mercado nuevas tecnologías disruptivas a las cuáles las empresas deben adaptarse para no quedarse obsoletos. Todo esto podría dar pie a nuevas herramientas que tendrían la oportunidad de sustituir una aplicación como “ComuniTools”. Por tanto, durante toda la fase de desarrollo del proyecto debemos estar atentos a la última hora de posibles actualizaciones y cambios en el mercado.

- **Poder de negociación de los proveedores.**

En el caso del desarrollo de la aplicación, los proveedores se encontrarán en los servicios externos que permitan el correcto funcionamiento, como por ejemplo el servicio de gestión de la base de datos, que mantendrá en la nube toda la información. Por tanto, se debe llegar a un acuerdo con esta empresa en caso de que se requiera ampliar la capacidad o mejorar en cuanto a ventajas se refiere.

1.3. Objetivos del proyecto.

El objetivo principal que se busca en este Trabajo de Fin de Grado es el desarrollo de una aplicación móvil en Android para la gestión de comunidades de vecinos, donde todos ellos podrán estar comunicados y al corriente de todos los sucesos que ocurran en el edificio, pudiendo incluso insertar incidencias, mantener constancia de las reuniones que se vayan a celebrar y realizar preguntas a la dirección de la comunidad.

Se persigue que la aplicación sea fácil de manejar, con una interfaz intuitiva y limpia de cara al usuario, con el fin de que todos los vecinos puedan aprovechar todas las herramientas disponibles y no experimenten errores y experiencias negativas utilizando las funciones.

Por último, otro de los objetivos presentes es llevar a cabo un proyecto software, teniendo en cuenta todos los detalles que implica cada etapa, estudiando en todo momento el tiempo y los recursos de los que disponemos, así como la puesta en marcha de todas las habilidades adquiridas durante el estudio del grado.

1.4. Estructura del resto de la memoria.

Durante el transcurso de este documento se tratarán todos los temas imprescindibles acerca de la aplicación:

- **Apartado 2. Análisis y definición del problema**, es decir, la búsqueda de una solución para cumplir con los requisitos previamente establecidos.

- **Apartado 3.** Las **tecnologías** y recursos que se han empleado para llevar a cabo el desarrollo software (entorno de desarrollo, lenguaje de programación, versiones…), además de las capas del proyecto (cliente y servidor).

- **Apartado 4.** El **tour por la aplicación** que muestra los resultados obtenidos en el desarrollo del proyecto.

- **Apartado 5.** Las **conclusiones** obtenidas tras realizar el proyecto y las **mejoras** de cara al futuro.

2. Análisis y definición del problema.

Análisis de requisitos.

El propósito del TFG es aportar una solución práctica para la correcta gestión de las comunidades de vecinos, con el fin de mejorar la comunicación entre los propietarios y, por lo tanto, de incrementar su bienestar personal.

Además, para obtener los requisitos de la aplicación, se ha contactado con administradores de fincas cercanos que conocen más detalladamente la función de una comunidad, por lo que, en base a la entrevista con ellos, se han listado funcionalidades esenciales para que se pueda abordar con más conocimiento el proyecto.

En este caso, la aplicación constará de las funciones principales de un gestor de fincas, con el fin de mantener un orden entre todo el vecindario y controlar todos los gastos y recursos disponibles. La herramienta tratará de mejorar el ambiente entre los propietarios y, además, facilitar las funciones necesarias.

Es importante conocer el alcance del producto software que se va a desarrollar, y es que se trata de una aplicación orientada a un público muy general, por lo que es evidente que se debe diseñar de tal forma que resulte sencilla e intuitiva al usuario final.

Al tratarse de una comunidad de vecinos se deben tener en cuenta los privilegios de los usuarios ya que está compuesta por:

- **Presidente.** Podrá gestionar todos los ítems relacionados con la comunidad.

- **Secretaría.** En algunas comunidades, el presidente dispone de un secretario que le puede ayudar a realizar gestiones a un nivel inferior de privilegio.

- **Usuario normal.** Los propietarios sin cargo no podrán modificar aspectos generales de la base de datos del sistema, pero sí visualizar en tiempo real los cambios realizados por los gestores.

A la hora de llevar a cabo un proyecto software se deben conocer los principales requisitos funcionales de la aplicación, por lo que a continuación se van a enumerar y desarrollar individualmente para encontrar la solución idónea para cada caso:

- **Registro.**

  El usuario debe poder registrarse en la plataforma de forma sencilla, por lo que rellenará un formulario donde introducirá sus datos y será dado de alta en la base de datos del sistema.

- **Inicio de sesión.**

  Se busca la máxima comodidad del usuario, por lo que este iniciará sesión de una forma intuitiva, introduciendo usuario y contraseña, además de tener la opción de recordar sus credenciales y que la aplicación inicie la sesión automáticamente.

- **Comunidad.**

  Los usuarios podrán visualizar a los integrantes de la comunidad que se hayan dado de alta en la aplicación. Para listarlos, cada usuario guardará un atributo en la base de datos que indicará el código de la comunidad a la que pertenece, y se comprobará cada vez que el usuario quiera acceder a la sección, ya que los ítems pueden variar en caso de que se dé un nuevo registro en la comunidad.

- **Contactos.**

El usuario podrá consultar los contactos de interés para todos los propietarios de la comunidad en tiempo real. Además, tanto presidencia como secretaría podrá añadir nuevos ítems a la lista.

- **Pagos.**

Cualquiera de los propietarios debe poder acceder a toda la información acerca de los gastos comunitarios, además de conocer todos los detalles de los pagos para realizar sus ingresos correspondientes en la cuenta bancaria de la comunidad.

- **Incidencias.**

En una comunidad de vecinos pueden producirse problemas que afecten directamente a todos los propietarios, averías e incidencias que puedan perjudicar al bienestar general. La herramienta será capaz de almacenar todo tipo de incidencias, permitiendo a los propietarios añadir todo aquello que consideren importante.

Además, si la solución a la incidencia tiene un coste, se podrá indicar el precio de la reparación, por lo que esta sección estará enlazada con el apartado de los pagos.

- **Reuniones.**

Es de vital importancia realizar juntas de propietarios para aclarar los aspectos más importantes de la comunidad. Para ello, la aplicación cubrirá la creación de una sección que sea capaz de informar a todos los propietarios de la existencia de una reunión, ya sea ordinaria (aprobación de presupuestos, cuentas de la Comunidad, solución de problemas…) o extraordinaria, con todos los datos necesarios para acudir.

El usuario podrá visualizar las reuniones comunitarias que se vayan a celebrar, consultando la fecha, hora y lugar del encuentro. También se podrán conocer detalles como los temas que se tratarán y si la asistencia es obligatoria o no.

- **Anuncios.**

Los vecinos deben conocer información relevante para la comunidad, por ello la elaboración de un tablón de anuncios permitirá exponer últimas actualizaciones. Solamente los usuarios con privilegios podrán añadir nuevos ítems.

- **Documentos.**

Una comunidad de vecinos debe tener en regla todos los documentos que la conviertan en legal y así lograr registrarla sin ningún tipo de problema. Además, se establecerán los cargos de la junta de propietarios, dejando constancia en acta en todo momento para lograr justificar todas las acciones que se han llevado a cabo.

Es la sección de la aplicación donde se adjuntarán documentos relevantes para la comunidad de vecinos, facilitando al usuario el acceso a los mismos.

- **Ajustes.**

Los miembros de la comunidad podrán realizar modificaciones relacionadas con su perfil de usuario, como modificar su foto de perfil e incluso el nombre de usuario que aparecerá en el listado de la comunidad.

3. Diseño y solución del problema.

3.1. Arquitectura de la aplicación.

Cliente/servidor.

Es el modelo que se va a seguir de cara a construir nuestro sistema de información, aprovechando todos los recursos disponibles a nivel de sistema para realizar un tratamiento de los datos e información.

En este tipo de arquitectura el usuario tendrá que disponer de la disponibilidad de los datos, es decir, en cualquier momento debe ser capaz de obtener la información que requiera y de procesarla en función de sus intereses.

Es conveniente conocer los principales detalles de la arquitectura, por lo que se ha realizado un listado con las características más importantes:

- Los clientes y los servidores son objetos que se encuentran separados desde el punto de vista lógico, es decir, se comunican a través de una red para realizar las tareas necesarias de forma conjunta.

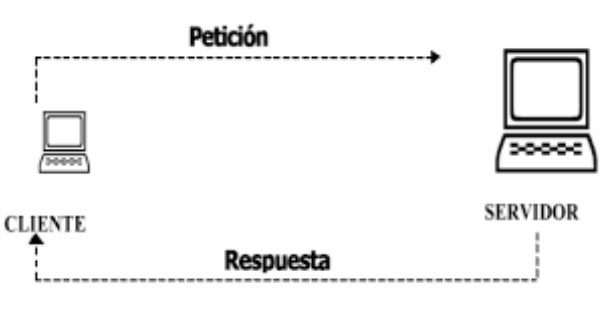
- El cliente realiza las peticiones de los servicios, esperando a la respuesta del servidor, que cuando recibe la petición la procesa y devuelve una respuesta en función de la solicitud. Todas estas peticiones se basan en el paso de mensajes.

- Un servidor puede proporcionar los servicios que contiene a más de un cliente de forma simultánea.

- El servidor debe estar disponible, quedando a la espera de mensajes de petición de clientes.

- El servidor tiene que poder atender a solicitudes simultáneas, por lo que debe garantizar la integridad de los recursos que se comparten, al igual que trabajar de forma concurrente.

Más adelante, en el punto 3.3. y 3.4. se detalla cómo se ha adaptado la aplicación al modelo cliente/servidor.



Extraída de www.siul02.si.ehu.es

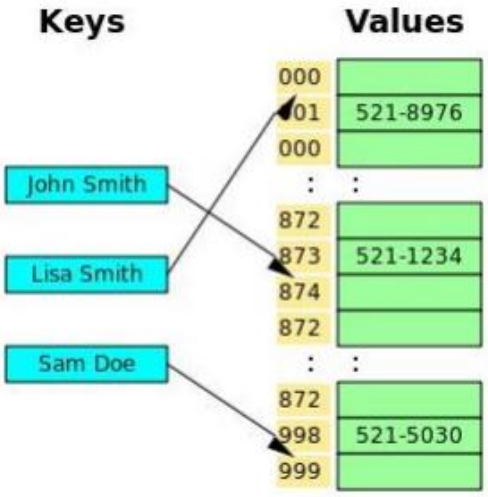
Tratamiento de datos.

Los datos que almacena la aplicación se guardarán en una base de datos alojada en la plataforma de Firebase, es decir, de forma online. Se ha decidido emplear esta herramienta debido a que la aplicación trata de comunicar al vecindario, por lo que se requiere de una conexión a internet. Además, el servicio gratuito que ofrece Firebase garantiza el correcto funcionamiento de la aplicación y el mantenimiento de la base de datos.

Cuando hablamos de una base de datos de Firebase, estamos ante un esquema no relacional o NoSQL, es decir, aquellas bases de datos que no tienen un esquema exacto de lo que se va a almacenar (no hay relación entre un conjunto de datos y otro), permitiendo así la versatilidad que puede ofrecer el sistema, ya que se pueden hacer una gran cantidad de cambios sin la necesidad de alterar las estructuras. También, el NoSQL nos permite asumir grandes volúmenes de datos y permitiría el crecimiento horizontal, es decir, son escalables y se podrían instalar más nodos para ampliar la capacidad sin interrumpir la usabilidad de la base de datos.

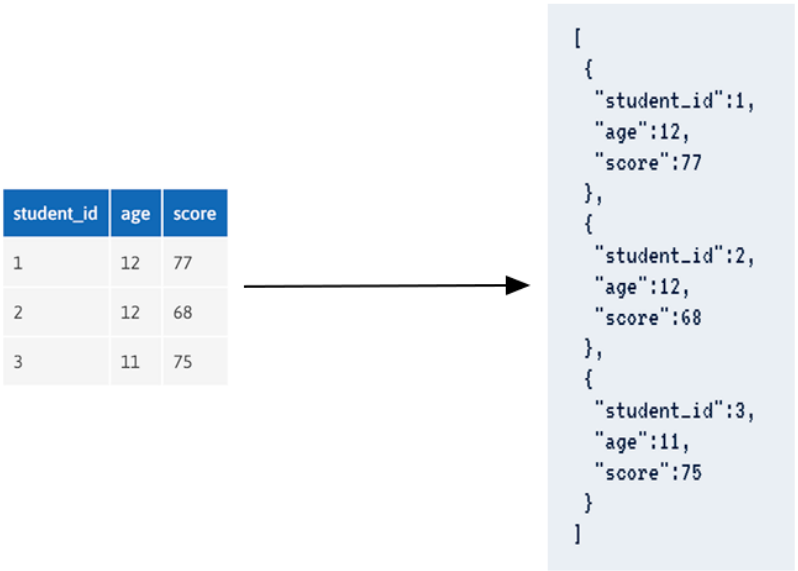
Existen varios tipos de esquemas NoSQL:

- **Clave-valor.** Cada elemento de la base de datos se identifica por una clave única, por lo que se hace más sencillo el proceso de recuperación de la información que almacena.



Extraída de www.acens.com

- **Documentales.** Se almacena la información a modo de documento, empleando un árbol JSON. Se pueden realizar búsquedas clave-valor y también consultas avanzadas acerca del contenido del documento, ofreciendo una gran versatilidad.



student_id	age	score
1	12	77
2	12	68
3	11	75

```
[
 {
  "student_id":1,
  "age":12,
  "score":77
 },
 {
  "student_id":2,
  "age":12,
  "score":68
 },
 {
  "student_id":3,
  "age":11,
  "score":75
 }
]
```

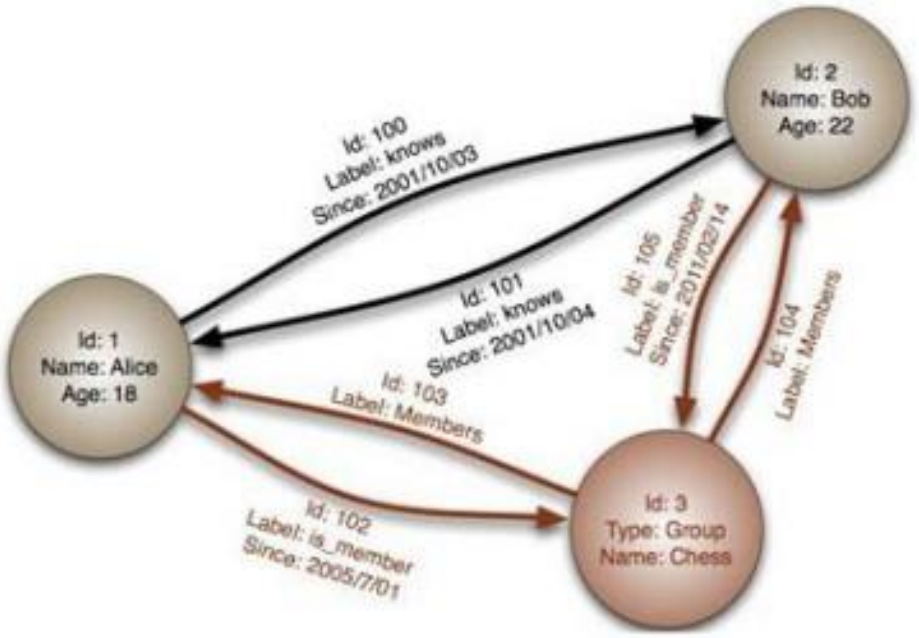
Extraída de www.aukera.es

- **Columnares.** Los datos se almacenan en columnas en lugar de filas, por lo que se logra leer y escribir datos de una forma más eficiente, ya que los valores de una columna están juntos (de forma unidimensional) y se almacena en orden de registro. Se invierte el sistema orientado a filas.

Número	1.	2.	3.
Apellido	Skywalker	Kenobi	Organa
Nombre	Luke	Obi-wan	Leia
Clave	3FN-Z768	7TR-K345	8NN-R266

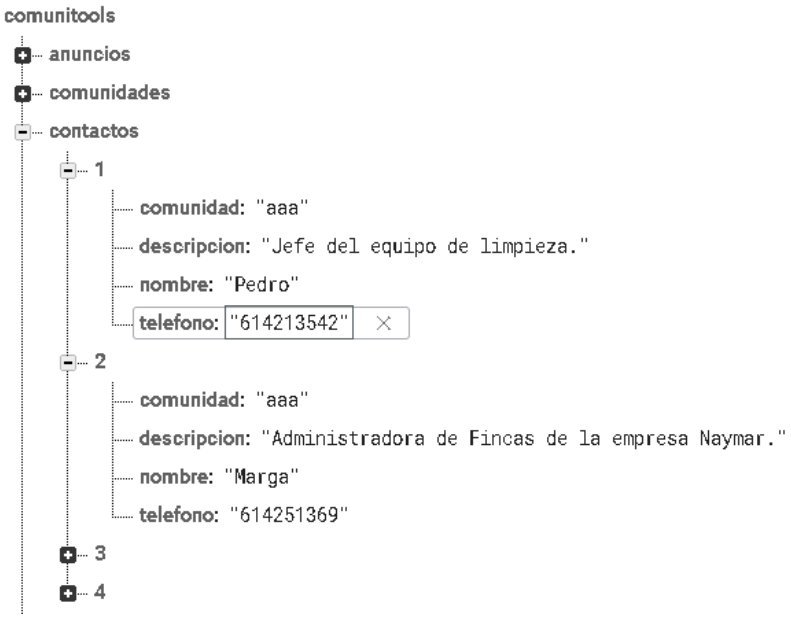
Extraída de www.ionos.es

- **Grafo.** La información se almacena en diferentes nodos de un grafo, relacionándose con las aristas de todos ellos. Es interesante mencionar que este tipo de estructura de base de datos ofrece una mayor eficiencia que un modelo relacional, ya que la estructura debe estar normalizada en su totalidad, es decir, cada tabla tiene una columna y cada relación dos.



Extraída de es.wikipedia.org

En el caso de la aplicación “ComuniTools” se va a emplear la base de datos de documentos, ya que se busca la versatilidad y Firebase ofrece el almacenamiento con la estructura de un árbol JSON y se puede aprovechar a nivel estructural.



3.2. Tecnologías utilizadas.

Sistema Operativo: Android.

Es un sistema operativo principalmente orientado a dispositivos móviles, basado en Linux, multiplataforma y gratuito. Está desarrollado por la empresa Android, la cual fue comprada por Google en el año 2005, que fue la empresa encargada de llevar este sistema operativo al éxito.

Debido a que su núcleo está basado en Linux, el sistema ofrece seguridad al usuario, ya que este no tiene los suficientes permisos para permitir el acceso de amenazas al dispositivo.

La principal razón del éxito de Android se debe a que es gratuito y su código abierto, por lo que cualquiera puede acceder a la descarga del mismo. Gracias a este hecho, se ha conseguido mejorar el sistema de una forma notable, permitiendo ciertas libertades a los desarrolladores y así contener una gran cantidad y variedad de aplicaciones en su tienda, Google Play, que cuenta con más de un millón de aplicaciones donde la mayoría son gratuitas para los usuarios, por lo que aumenta todavía más el atractivo hacia este sistema.

Además, Android ofrece una amplia capa de personalización al usuario, pudiendo éste modificar la apariencia e incluso funcionalidades en sus dispositivos.

La evolución de Android.

El famoso sistema operativo debe su éxito a la mejora constante en su rendimiento y funcionalidades, y esto le ha llevado más de 10 años, por lo que se deben conocer todas las novedades que se han ido incorporando.

Extraída de www.todoandroid360.com

Inicialmente, Android fue fundado en 2003 por Nick Sears, Chris White y Andy Rubin, con la misión de desarrollar software para dispositivos móviles. Dos años más tarde, Google adquirió algunas pequeñas empresas con la intención de combinar su éxito en internet con las futuras tecnologías, y Android fue una de ellas, ya que el mercado de la telefonía móvil mostraba tener un gran potencial de cara al futuro.

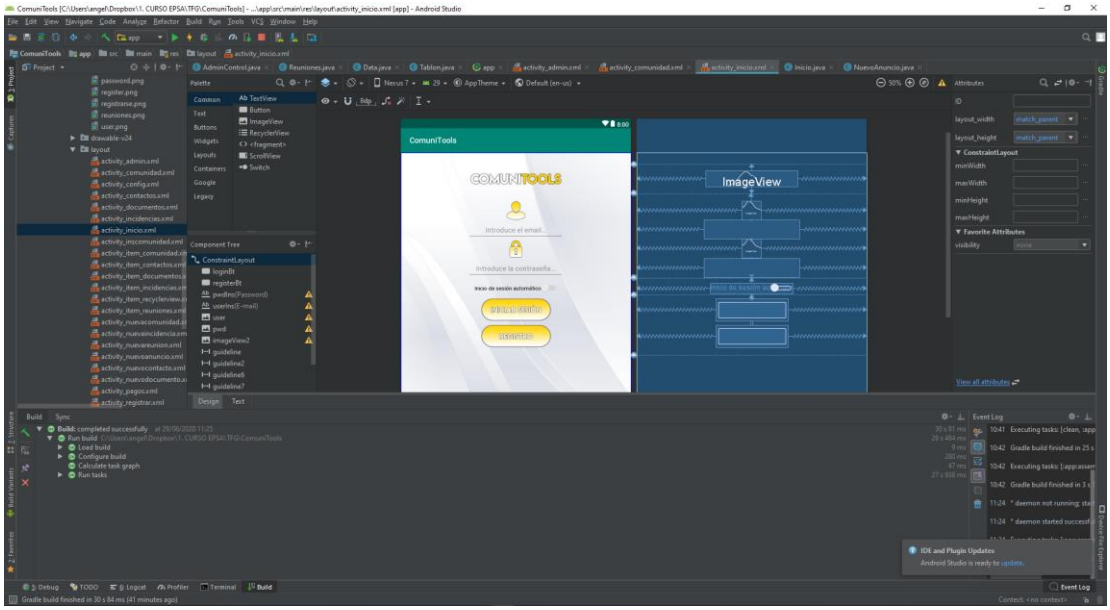
En 2007 llegaría el paso más importante para Android, ya que fue incluido en el proyecto de Google llamado Open Handset Alliance (OHA), que era una agrupación de 35 organizaciones enfocadas al mercado móvil (fabricantes de dispositivos, de hardware y de software).

El primer dispositivo que corrió el sistema operativo Android fue el HTC Dream en el año 2008 y posteriormente en un HTC G1 con pantalla táctil, con teclado y funcionalidades inalámbricas, por lo que se podía acceder a internet gracias al navegador web (compatible incluso con XHTML) que llevaban estos dispositivos integrados, además de incorporar un servicio email. Sin duda la gran característica era la presencia de Android Market (el actual Google Play), que ofrecía una gran variedad de aplicaciones para aprovechar al máximo el dispositivo.

A raíz de su lanzamiento, Android no hizo más que crecer, lanzando diferentes versiones que, gracias a la mejora constante, le han convertido en uno de los sistemas operativos móviles más utilizados del mundo.

Entorno de desarrollo: Android Studio.

A la hora de llevar a cabo el desarrollo de una aplicación debemos elegir un IDE (entorno de desarrollo integrado) que se adapte a nuestras necesidades y nos resulte más cómodo.



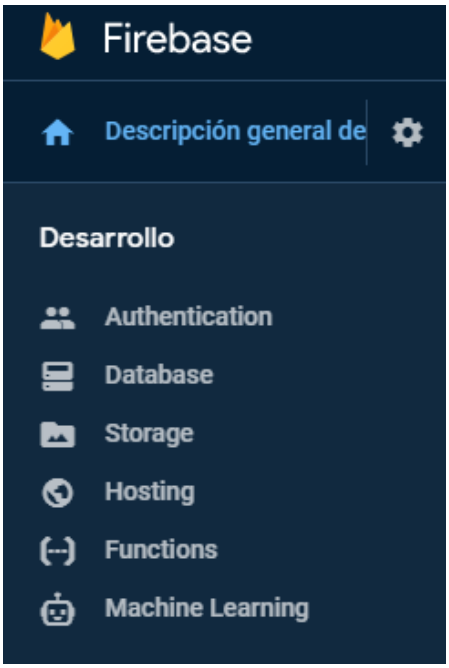
En este caso, se ha escogido la opción de Android Studio, ya que es el IDE oficial de Android que incorpora un editor de código potente y una gran cantidad de funciones que ayudarán al desarrollador a conseguir un producto más consistente. Algunas de estas funciones son:

- **Emulador de dispositivos Android.**

  Android Emulator es una herramienta que permite simular dispositivos Android en tu ordenador. Se pueden emular todo tipo de dispositivos, variando su resolución de pantalla, la versión de Android e incluso diferentes niveles de API. Además, cuenta con una conexión a internet (gracias al puente que se realiza automáticamente con la conexión del ordenador) para poder realizar pruebas con la aplicación que se está desarrollando. En este caso, se empleará esta característica para comprobar el estado de Firebase y poder realizar las pruebas pertinentes.

- **Compatibilidad con Google Firebase.**

Google Firebase es una plataforma online de Google, que permite a los desarrolladores facilitar tareas a la hora de crear sus aplicaciones, además de aumentar la productividad de éstas y así lograr grandes éxitos.



- **Herramientas para maximizar la compatibilidad de las aplicaciones en una gran cantidad de dispositivos Android.**

Existen varias funcionalidades que permiten adaptar tanto el diseño como el nivel de software a una gran cantidad de dispositivos, para así maximizar la compatibilidad:

- ✓ **ConstraintLayout.** Crear un diseño reactivo para la adaptación a diferentes tamaños de pantalla. Permite especificar la posición de todos los elementos según las relaciones con otras vistas del diseño.

- ✓ **Mapas de bits alternativos.** Para definir diferentes densidades de píxeles, se pueden importar recursos en función del tamaño de cada pantalla (MDPI, HDPI, XHDPI, XXHDPI, XXXHDPI).

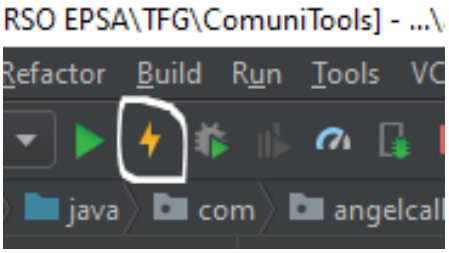
- ✓ **APKs para diferentes niveles de API.** En caso de que la aplicación se instale desde un terminal más antiguo, podría acceder a la descarga de la misma gracias a esta función. La única limitación se encuentra cuando en niveles tan inferiores de API todavía no estaban implementadas ciertas funciones que impiden lograr el cometido de la aplicación.

- ✓ **Gestión de idiomas.** Gracias a la modificación del fichero “strings.xml” se puede establecer un diccionario en función del idioma que el usuario tenga configurado en su terminal.

- ✓ **Diferentes plataformas.** La herramienta permite compilar la aplicación para una gran cantidad y tipos de dispositivos, como por ejemplo Smart TV, aunque en este caso se implementa solamente para smartphones y tablets.

- **Entorno que permite realizar pequeños cambios en el código y ver los resultados reflejados al momento.**

  Gracias a este tipo de funciones podremos ver los cambios de forma instantánea en el emulador o en el dispositivo en el cual se esté volcando la aplicación, ganando tiempo a la hora de encontrar soluciones a los problemas que vayan apareciendo.



Java y Android SDK.

Para el desarrollo de la aplicación se ha empleado el lenguaje de programación Java y el kit de desarrollo de software (SDK) de Android.

Para conocer un poco más Java, se han buscado las características principales del lenguaje y las ventajas que puede aportar a la hora de desarrollar software:

- **Lenguaje simple.**

No es un lenguaje complejo, es más, la curva de aprendizaje es corta gracias a la sencillez de sus funciones.

- **Seguridad.**

La compilación que ofrece permite al desarrollador crear un código alejado de fallos de seguridad. Un ejemplo serían los métodos que emplea para que no se permitan los accesos a memoria ilegales.

- **Multiplataforma.**

Java permite desarrollar software para una gran cantidad de dispositivos, incluso servidores y sistemas operativos. Esto se debe a la existencia de una implementación de una máquina virtual de Java compatible con prácticamente cualquier sistema operativo. Además, es un lenguaje interpretado y compilado, por lo que se puede ejecutar en cualquier lugar sin dar problemas.

- **Robustez.**

Java evita cualquier tipo de problema para asegurar la integridad y gestiona la memoria de forma automática prácticamente en su totalidad gracias a la máquina virtual que posee, permitiendo así que la memoria no se corrompa y que no haya conflictos entre datos de otras aplicaciones o del sistema.

- **Programación orientada a objetos.**

Gracias a su orientación a objetos, ha sido posible acercar a la programación el pensamiento humano, ya que resulta más sencillo desarrollar aplicaciones modulares y se pueden implementar patrones de diseño.

```
private String descripcionContacto;
private String nombreContacto;
private String telefonoContacto;

public static ArrayList<ItemContactos> listItemsCon = new ArrayList<>();

public Data(String descripcionContacto, String nombreContacto, String telefonoContacto) {
    this.descripcionContacto = descripcionContacto;
    this.nombreContacto = nombreContacto;
    this.telefonoContacto = telefonoContacto;
}

// Getters & Setters.

public String getDescripcionContacto() {
    return descripcionContacto;
}

public void setDescripcionContacto(String descripcionContacto) {
    this.descripcionContacto = descripcionContacto;
}

public String getNombreContacto() {
    return nombreContacto;
}

public void setNombreContacto(String nombreContacto) {
    this.nombreContacto = nombreContacto;
}

public String getTelefonoContacto() {
    return telefonoContacto;
}

public void setTelefonoContacto(String telefonoContacto) {
    this.telefonoContacto = telefonoContacto;
}
```

Además, es conveniente hablar del Android SDK, ya que proporciona librerías de desarrollo que permiten al desarrollador tanto construir el código como realizar pruebas y depuraciones del mismo. Está formado por:

- **Librerías.**
- **Depurador de código.**
- **Emulador.**
- **Documentación acerca de la API.**
- **Código fuente a modo de ejemplo.**
- **Tutoriales de Android.**

Base de datos.

Al tratarse de una aplicación que requiere de comunicación entre usuarios, es conveniente que la base de datos se encuentre en la nube, por lo que en la sección de la arquitectura de la aplicación (Punto 3.1) se desarrollará la estructura de la misma.

Se han estudiado algunas alternativas para mantener la base de datos y la subida de ficheros a la nube:

- ✓ **Parse Server.**

Las ventajas de utilizar esta plataforma son:

- **Almacenamiento de datos.**

  Se utiliza MongoDB (sistema gestor de base de datos No-SQL) para almacenar datos y para almacenar ficheros Amazon S3.

  El servidor ofrece facilidades al usuario respecto a la base de datos, como por ejemplo mantener copias de seguridad, permitir la restauración de datos e incluso la indexación y refactorización del rendimiento.

- **Consultas en vivo.**

- **Despliegue sencillo.**

- ✓ **Google Firebase.**

Las principales características que se han extraído del servidor son:

- **Accesibilidad.**

  Los datos pueden ser accesibles desde cualquier lugar y en cualquier momento, y, además, el desarrollador podrá almacenar los datos fácilmente debido a la integración del formato JSON.

- **Sincronización de datos en tiempo real.**

  Es una de las características más atractivas, ya que al mantener una sincronización en tiempo real se mejorará la interactividad de la aplicación, además de mejorar el rendimiento a la hora de modificar variables.

- **Seguridad de los datos.**

  Se trata de una plataforma robusta y que garantiza la seguridad de toda la información que se almacena en la misma, permitiendo incluso conexiones con servicios web externos como las cuentas de Google y Facebook.

La elección final ha sido Firebase, ya que, además de ser una plataforma enfocada a las aplicaciones en tiempo real, tiene una API más potente y mejor conectada con el entorno de programación de Android, permitiendo al desarrollador un mayor manejo y explotación de los recursos.

- **Registro e inicio de sesión.**

  Es imprescindible que el usuario tenga facilidad a la hora de registrarse en la plataforma, al igual que iniciar sesión e incluso crear una comunidad, por lo que se desarrollarán formularios sencillos e intuitivos.

  Se empleará el sistema de autentificación que implementa Google Firebase, debido a que la plataforma se encargará de alojar todas las credenciales de forma efectiva. Se almacenan todos los datos de los usuarios en tiempo real, por lo que el rendimiento mejorará notablemente.

3.3. Servidor.

Firebase.

Sus principales funcionalidades que se van a emplear en el desarrollo del proyecto, todas disponibles en la nube, son:

- ✓ **Realtime Database.** Base de datos NoSQL que almacena los datos y los sincroniza en tiempo real, incluso los mantiene disponibles sin conexión.

Para importar a la aplicación el repositorio de la base de datos, se debe añadir el siguiente código al gradle:

```
apply plugin: 'com.google.gms.google-services'

dependencies {

    implementation 'com.google.firebase:firebase-database:16.0.1'

}
```

Para realizar las consultas se ha utilizado una estructura de código semejante a la siguiente:

```
import com.google.firebase.database.*;

public class ConsultaBBDD extends AppCompatActivity {

    DatabaseReference mDatabase;

    protected void onCreate (Bundle savedInstanceState) {

        mDatabase = FirebaseDatabase.getInstance().getReference();

        mDatabase.child(xxx).addValueEventListener(new ValueEventListener() {

            @Override
            public void onDataChange (@NonNull DataSnapshot dataSnapshot) {

                // Operaciones de lectura y tratamiento de datos.

            }

            @Override
            public void onCancelled (@NonNull DatabaseError databaseError) {


            }

        });

    }

}
```

Es conveniente hablar de las principales dificultades que se han presentado durante la implementación de la base de datos, por lo que se ha elaborado un pequeño listado con pequeños detalles:

- **Conectar datos con la aplicación.**

  Ha resultado complejo en un principio obtener la información y mantenerla dentro de la aplicación, de cara a realizar el tratamiento de datos y comunicar valores entre clases.

Finalmente, se ha creado una clase “Data.java” que mantendrá variables estáticas para mantener la información necesaria para funciones dentro de la aplicación, es decir, se accederá a las variables creando un objeto en la clase que lo requiera.

Para la información que se desea mantener en todo momento, incluso cuando la aplicación no está en ejecución, se han utilizado las SharedPreferences, que guarda valores en la memoria del dispositivo, pudiendo así guardar credenciales importantes.

- **Inserción y actualización de datos.**

A la hora de introducir y actualizar valores dentro de la base de datos es importante utilizar correctamente las referencias y la API que nos ofrece Firebase, ya que en ciertas ocasiones se pueden experimentar errores donde se eliminan campos o se sobreescriben sin la posibilidad de volver atrás.

Se ha logrado estructurar el código de forma que queden claros todos los accesos a la base de datos de la nube, además de una estructura sencilla cada vez que se añadan nuevos elementos, es decir, que queden bien almacenados y con identificadores sencillos.

- **Recorrer la base de datos.**

Firebase tiene ciertos límites a la hora de realizar consultas, por lo que es importante considerar mejoras de eficiencia a la hora de recorrer las tablas y los campos de las mismas. Se han dado problemas relacionados con bucles que no terminaban y podrían haber llegado a superar la tasa de consultas, pero se ha logrado encontrar un orden a la hora de obtener información de la base de datos.

Gracias a las posibilidades que ofrece Firebase, se han utilizado comandos que permiten al desarrollador realizar comprobaciones previas sobre la base de datos para no experimentar grandes problemas. Por ejemplo, el uso de “getChildrenCount()” ayuda a conocer el tamaño de la tabla antes de abordarla, con el fin de no sobrepasar los límites y que la aplicación deje de funcionar.

- ✓ **Authentication.** Sistema de inicio de sesión, manteniendo una lista de credenciales.

  Documento de repositorios para la aplicación:

```
apply plugin: 'com.google.gms.google-services'

dependencies {

    implementation 'com.google.firebase:firebase-auth:16.0.1'

}
```

Clase Java donde se va a iniciar sesión:

```
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.AuthResult;

public class IniciarSesion extends AppCompatActivity {

    private EditText userIns, pwdIns;
    private FirebaseAuth mAuth;

    protected void onCreate(Bundle savedInstanceState) {

        mAuth = FirebaseAuth.getInstance();

        Button iniciarSesion = (Button) findViewById(R.id.iniciarSesion);

        iniciarSesion.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                startSignIn();

            }

        });

    }
```

```
private void startSignIn() {

    final String user = userIns.getText().toString();
    String pwd = pwdIns.getText().toString();


    if (user.length() == 0 || pwd.length() == 0) {

        // Operaciones si el usuario no inserta las credenciales.

    } else {

        mAuth.signInWithEmailAndPassword(user, pwd)
                .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {

                        if (!task.isSuccessful()) {

                            // Operaciones si el inicio de sesión no es correcto.

                        } else {

                            // Operaciones que se ejecutan cuando el inicio de sesión es correcto.

                        }

                    }
                });

    }

}
```

El principal reto de la función ha sido lograr enlazar el registro e inicio de sesión con la base de datos, y el único problema que se ha podido producir se ha encontrado en el orden de la ejecución del código.

Al principio de desarrollar la función de registro, si se experimentaban problemas con la parte de Firebase Authentication y no se registraba correctamente el usuario, el Firebase Database sí que llevaba a cabo su función, por lo que se añadía a la base de datos información respectiva a un usuario que realmente no existía dentro de la aplicación.

Para resolver este problema, se ha organizado el código de tal forma que cuando la tarea de Authentication ha finalizado correctamente, se ejecuta el apartado de adición de información a la base de datos.

- ✓ **Storage.** Sistema de almacenamiento de ficheros.

  Documento de repositorios para la aplicación:

```
apply plugin: 'com.google.gms.google-services'

dependencies {

    implementation 'com.google.firebase:firebase-storage:16.0.1'

}
```

Sección del código Java donde se va a subir el fichero:

```
FirebaseStorage mStorageRef = FirebaseStorage.getInstance();

StorageReference mStorage = mStorageRef.getReference();

filePath = mStorage.child("docs").child(dataD.getPathPdf().getLastPathSegment());

filePath.putFile(dataD.getPathPdf()).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {

        Toast.makeText(NuevoDocumento.this, "Fichero subido correctamente", Toast.LENGTH_SHORT).show();

    }

});
```

La tarea más importante dentro de esta sección ha sido enlazar el sistema de almacenamiento con la base de datos. El principal problema se ha encontrado a la hora de recoger las rutas de los ficheros, pero la función "getLastPathSegment()" ha ayudado notablemente a llegar a una solución.

Cuando se sube un fichero, se enlaza al campo correspondiente de la base de datos, introduciendo su ruta de acceso para poder listarla en cualquier momento. En este caso, se ha resuelto obteniendo el enlace referencia del fichero dentro del almacenamiento una vez ha sido subido con éxito (para evitar errores se ha ordenado el código) y enviándolo a la base de datos.

✓ **Notificaciones push.**

Es imprescindible que el usuario en todo momento reciba notificaciones de nuevas modificaciones para estar al tanto de todas las reuniones y los anuncios de la comunidad.

En este caso, tras realizar un estudio de las posibilidades que nos ofrece Firebase Cloud Messaging (FCM), se ha podido desarrollar una idea para que los usuarios reciban mensajes de forma selectiva, es decir, por clasificación según la comunidad a la cual pertenecen.

Para llevar a cabo el desarrollo de esta función se han seguido una serie de pasos:

- **Implementación del SDK de FCM.**

  Se añaden los repositorios correspondientes al manifiesto de la aplicación y al documento que incluye las librerías empleadas.

- **Creación del servicio de recepción de mensajes.**

Para lograr la escucha para recibir nuevas notificaciones, se ha desarrollado un servicio dentro de la aplicación llamado “MyFirebaseMessagingService.java”, donde se trata con el método “onMessageReceived” de cara a tratar los datos que la aplicación recibe y adaptarlos al dispositivo para que el usuario pueda visualizar fácilmente la notificación.

- **Desarrollo del sistema de canales de envío.**

Los usuarios de una comunidad se registrarán automáticamente al canal de recepción (código de su comunidad) para recibir las notificaciones en cualquier momento. Cuando el usuario inicie sesión, se ejecutará la siguiente línea de código que hará posible que se dé de alta a las notificaciones:

```
FirebaseMessaging.getInstance().subscribeToTopic(comunidad);
```

Además, cuando se cierra sesión, el usuario dejará el mismo canal de comunicación para no recibir mensajes en su dispositivo. Esto es posible gracias a la siguiente línea de código:

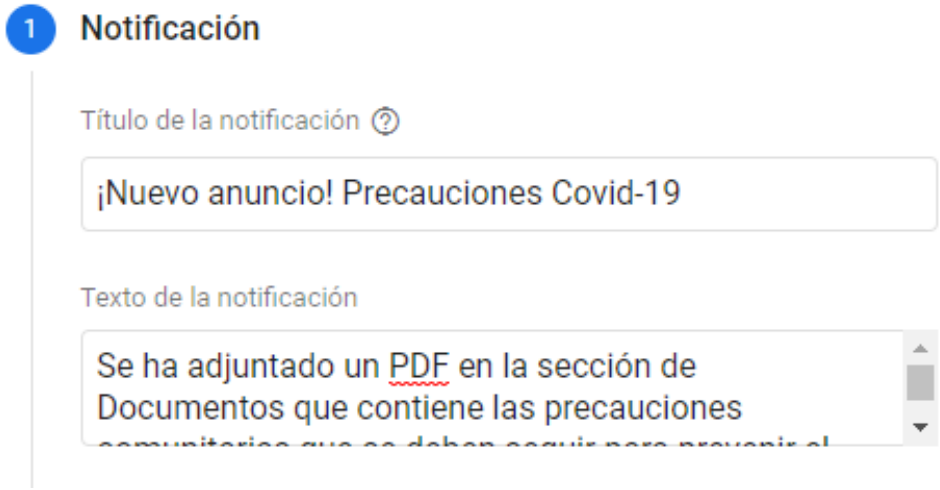
```
FirebaseMessaging.getInstance().unsubscribeFromTopic(comunidad);
```

- **Envío de mensajes.**

El inconveniente que se ha encontrado en esta sección se debe a que la API de Firebase Cloud Messaging permite solamente, desde la aplicación Java, crear peticiones de envío de mensajes y no enviarlos directamente, por lo que sería necesario crear las notificaciones finales desde el panel de control de Firebase.

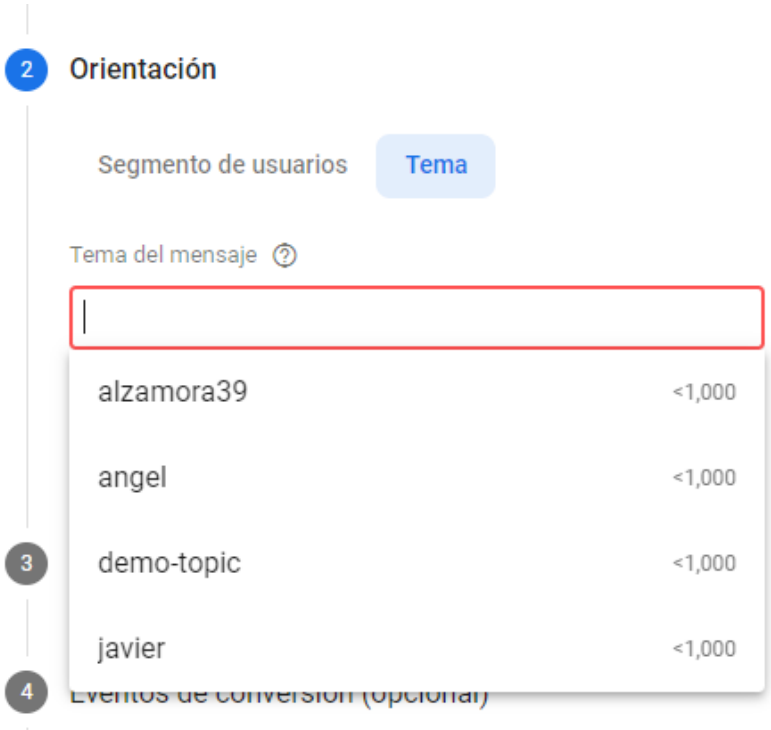
A continuación, se muestra una demostración de la recepción de mensajes con un ejemplo en el que aparecen Javier y Ángel (usuarios de la comunidad Javier) y Manuel (usuario de la comunidad Alzamora 39):

✓ Creación de la notificación desde el Panel de Control de Firebase:



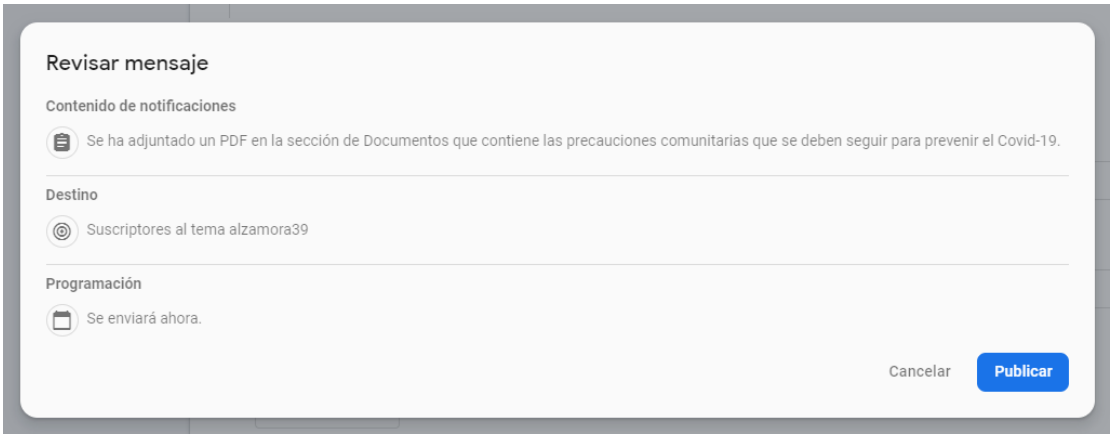
Datos de la notificación.

✓ Selección del canal de comunicación (comunidad) por el que circulará el mensaje:

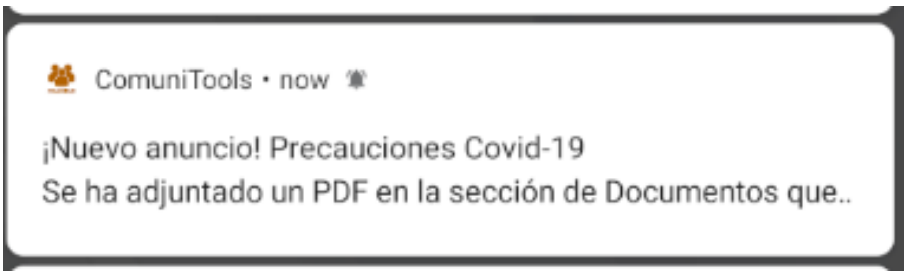


Se ha escogido “alzamora39”, comunidad a la que pertenece Manuel.

✓ Revisión y publicación del mensaje:



✓ Recepción final del mensaje:



Manuel recibe la notificación, mientras que Javier y Ángel no, ya que están suscritos a un canal de comunicación diferente.

3.4. Cliente.

Es la capa en la que se ha tenido que analizar con más delicadeza el diseño a la hora de generar una interfaz intuitiva y sencilla para el usuario, pudiendo éste acceder a todas las funcionalidades de forma sencilla y sin experimentar errores en el tratamiento de datos.

Se ha abordado en función de diferentes campos y estructuras de componentes:

- **Listar elementos.**

  Para listar los elementos será necesario mantenerlos ordenados y que aparezcan correctamente clasificados por comunidad. Se dispone de varias opciones para cumplir con esta funcionalidad, por lo que se ha analizado qué opción es la mejor para la aplicación:

- **ListView.**

Es una vista que agrupa varios elementos de un array y los muestra en una lista por la que el usuario se puede desplazar.

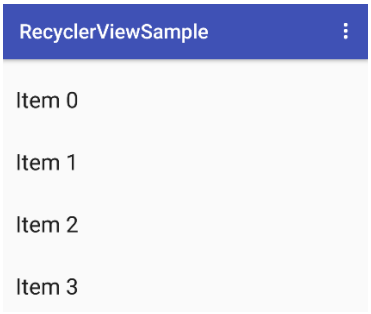
Los elementos se introducen de forma automática gracias a un adaptador que extrae todos los datos que se van a mostrar, es decir, el adaptador encargado de conectar la interfaz de usuario con la fuente de datos.

Existen varios tipos de adaptador que proporciona Android: ArrayAdapter, BaseAdapter, CursorAdapter, SimpleCursorAdapter, SpinnerAdapter y WrapperListAdapter.

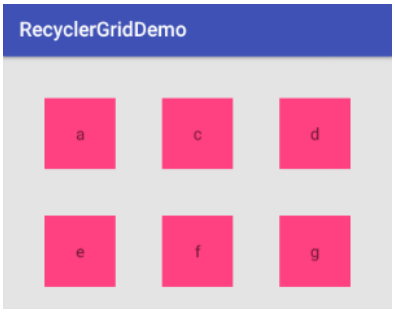
- **RecyclerView.**

Es una mejora de ListView, ya que se pueden reutilizar las celdas mientras el usuario se desplaza por los elementos, se pueden animar acciones de la lista y también se desacopla la lista del contenedor. En aspectos de diseño, el RecyclerView es más personalizable, brindando más poder al desarrollador.

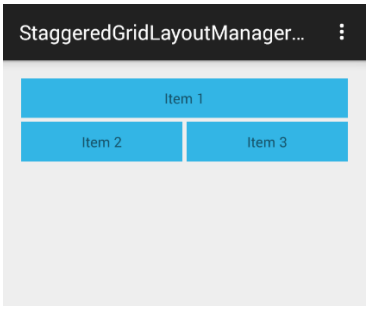
Además, el RecyclerView puede contener tanto una lista normal (LinearLayoutManager), como un grid (GridLayoutManager) e incluso una lista con distribución personalizada (StaggeredLayoutManager).



LinearLayoutManager

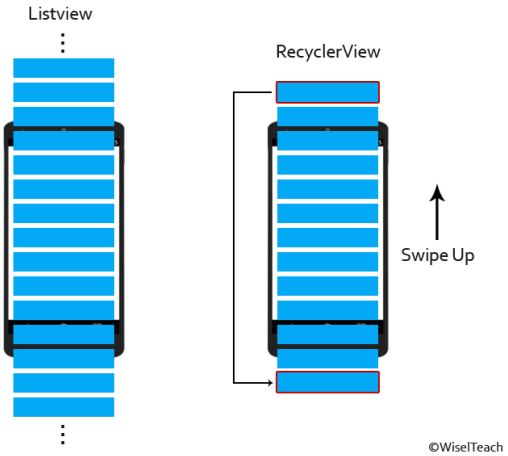


GridLayoutManager



StaggeredLayoutManager

Elegiremos esta opción, ya que permite llevar un control más flexible sobre los datos que se van a mostrar y es más eficiente debido a que el tamaño de la lista se calcula previamente e incluso se almacena en caché.



Extraída de www.medium.com

- **Clases y actividades.**

El código se ha estructurado de tal forma que quede bien organizado y claro para poder comunicar correctamente las actividades.

A continuación, se muestra un listado de las comunicaciones que se producen entre actividades:

- La clase Inicio está conectada a las clases:
  - Registrar. Se accede mediante el botón de registro en caso de que el usuario decida crear una cuenta.
  - MenuPpal. El menú aparece cuando un usuario perteneciente a una comunidad inicia sesión. Además, cuando el usuario cierra sesión, es redirigido a la pantalla de inicio.
  - InsComunidad. Cuando el usuario que inicia sesión no pertenece todavía a ninguna comunidad es redirigido a la sección donde se puede insertar el código de la comunidad.

- La clase InsComunidad está conectada a:
  - MenuPpal. Cuando el código de comunidad es correcto, el usuario es redirigido al menú principal.
  - NuevaComunidad. Si el usuario desea crear una nueva comunidad, es redirigido al panel de creación.

- La clase MenuPpal está conectada a todas las clases que contienen las funcionalidades de la aplicación:
    - Comunidad.
    - Contactos.
    - Documentos.
    - Reuniones.
    - Incidencias.
    - Anuncios.
    - Pagos.
    - Configuración.
- Las clases de las funcionalidades están conectadas con sus respectivas actividades de inserción de nuevos elementos:
    - Incidencias con NuevaIncidencia.
    - Contactos con NuevoContacto.
    - Anuncios con NuevoAnuncio.
    - Reuniones con NuevaReunion.

  Cabe recalcar que a estas funciones solamente pueden acceder usuarios con privilegios de presidencia o secretaría. Esta función se ha desarrollado consultando en el inicio de sesión del usuario el campo de permiso que contiene. Si el valor del permiso es 0, el usuario será normal; si el valor es 1, el usuario será el presidente de la comunidad; y si el valor es 2, el usuario es el secretario. El listado de permisos y las acciones que pueden realizar se encuentran al final del Punto 4.

Todo el contenido de las actividades mencionadas se trata en el Punto 4 de la memoria.

Además, es importante mencionar que para reflejar todos los datos en el RecyclerView ha sido necesario crear adaptadores y la clase de ítems de cada apartado.

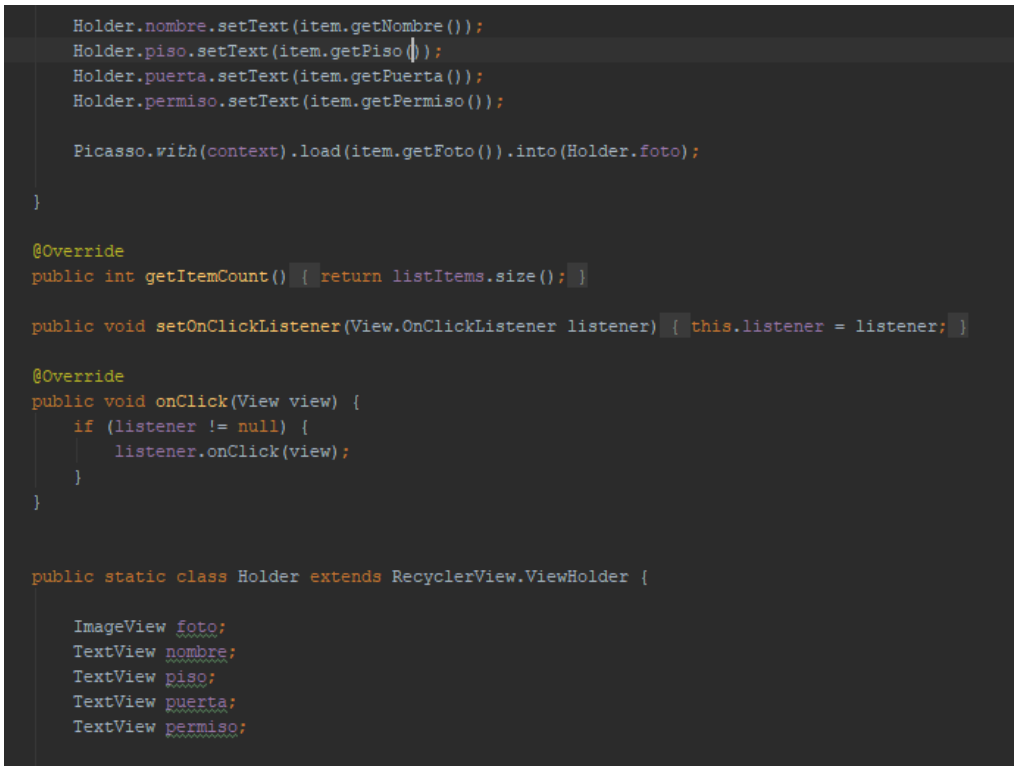
La clase de ítems de cada apartado contiene el paso de información desde la Firebase Database hasta el adaptador.

```
public class ItemComunidad extends AppCompatActivity {

    private String foto;
    private String nombre;
    private String piso;
    private String puerta;
    private String permiso;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_item_comunidad);

    }

    public ItemComunidad(String foto, String nombre, String piso, String puerta, String permiso) {
        this.foto = foto;
        this.nombre = nombre;
        this.piso = piso;
        this.puerta = puerta;
        this.permiso = permiso;
    }

    public String getFoto() { return foto; }

    public String getNombre() { return nombre; }

    public String getPiso() { return piso; }
```

Las clases existentes son:

- ItemComunidad.
- ItemContactos.
- ItemDocumentos.
- ItemIncidencias.
- ItemReuniones.
- ItemAnuncios.

Un adaptador traduce toda la información y estructura del elemento obtenida previamente en la clase de ítems (se crea un objeto de la clase) para acomodarlos a la lista y que se puedan consultar de forma correcta. Son necesarios para acceder a funciones especiales (eliminar ítems y más funcionalidades) desde la clase principal donde se listan los elementos.



```
    Holder.nombre.setText(item.getNombre());
    Holder.piso.setText(item.getPiso());
    Holder.puerta.setText(item.getPuerta());
    Holder.permiso.setText(item.getPermiso());

    Picasso.with(context).load(item.getFoto()).into(Holder.foto);

}

@Override
public int getItemCount() { return listItems.size(); }

public void setOnClickListener(View.OnClickListener listener) { this.listener = listener; }

@Override
public void onClick(View view) {
    if (listener != null) {
        listener.onClick(view);
    }
}


public static class Holder extends RecyclerView.ViewHolder {

    ImageView foto;
    TextView nombre;
    TextView piso;
    TextView puerta;
    TextView permiso;
```

Los adaptadores existentes son:

- ContactosAdapter.
- ComunidadAdapter.
- DocumentosAdapter.
- IncidenciasAdapter.
- AnunciosAdapter.
- ReunionAdapter.

Las principales dificultades que se han producido a la hora de desarrollar esta sección han sido las siguientes:

- **Inserción de los elementos.**

  Para insertar los elementos en la lista, previamente se debía de obtener correctamente la información. Debido a problemas con el código, inicialmente no lograba obtener todos los datos necesarios, por lo que se reestructuró el código para que se completase correctamente la recolección de información antes de pasarla al adaptador, ya que sino no se actualizaría y el usuario no podría visualizar cambios al momento.

- **Recorrido de la lista.**

  Cuando se logró obtener toda la lista completa de elementos, se dieron problemas a la hora de tratar la información en aspectos de eliminación debido a los índices e identificadores.

  Para solucionar el problema, se utilizó la función que ofrece la clase extendida “RecyclerView.Adapter” y permite obtener la posición de cada elemento llamada “getChildAdapterPosition(view)”.

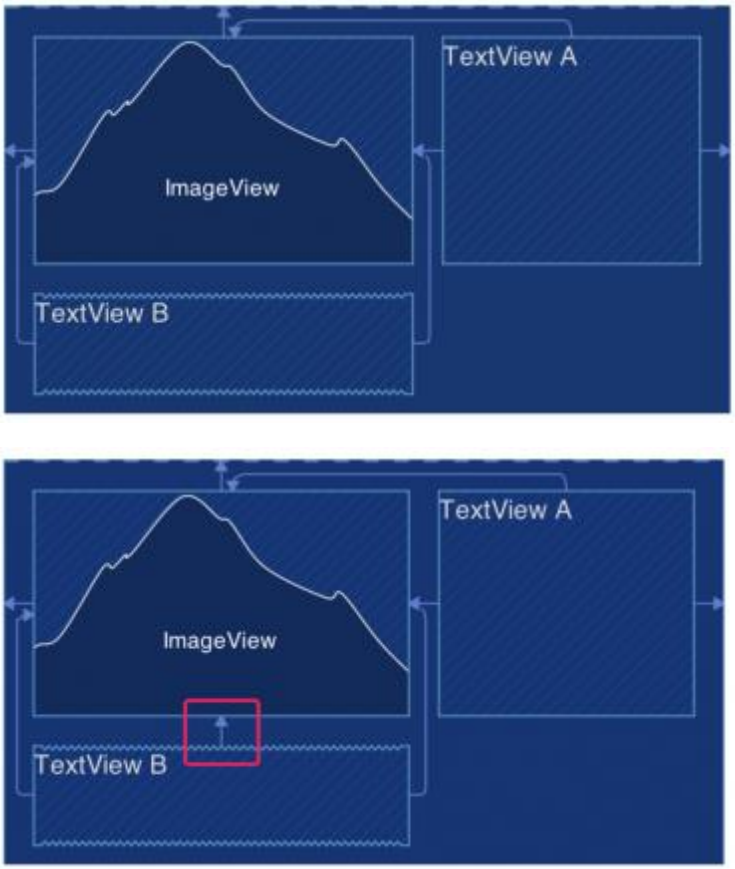
- **Diseño de botones y formularios.**

De cara a diseñar los botones, se ha utilizado la herramienta Photoshop, ya que ofrece grandes posibilidades de personalización.

En todo momento se ha buscado crear un diseño sencillo y básico, con colores que no resalten y que sean tranquilos para navegar.

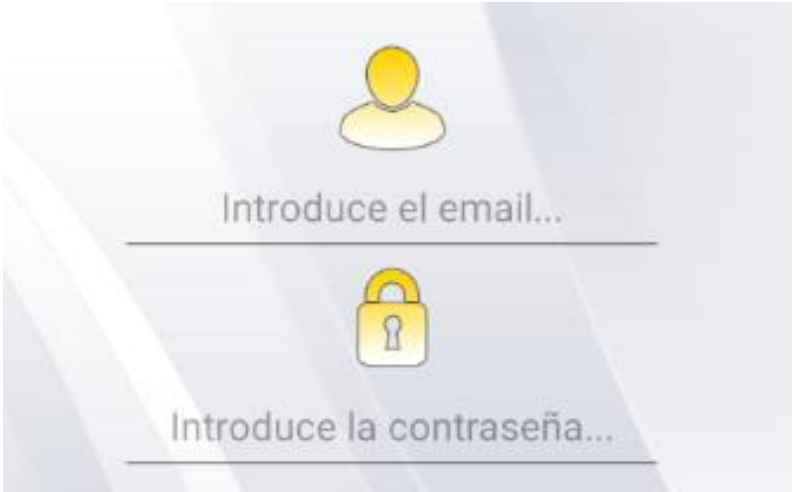
Además, se ha tenido en cuenta la adaptación a la pantalla de los dispositivos móviles, por lo que se ha establecido un tamaño de imagen medio-alto para que el usuario no experimente ningún tipo de problema visual.

Toda la aplicación ha sido introducida en un ConstraintLayout, ya que permite al desarrollador simplificar las interfaces en cuanto a anidamiento se refiere. Se han establecido relaciones entre todos los elementos que conforman la vista para que así estemos ante un diseño más flexible.



Extraída de elandroidelibre.elespanol.com

A la hora de elaborar los formularios se ha tenido en cuenta la sencillez que requieren, por lo que se han utilizado EditText que indican previamente la información que debe ser introducida en cada campo.



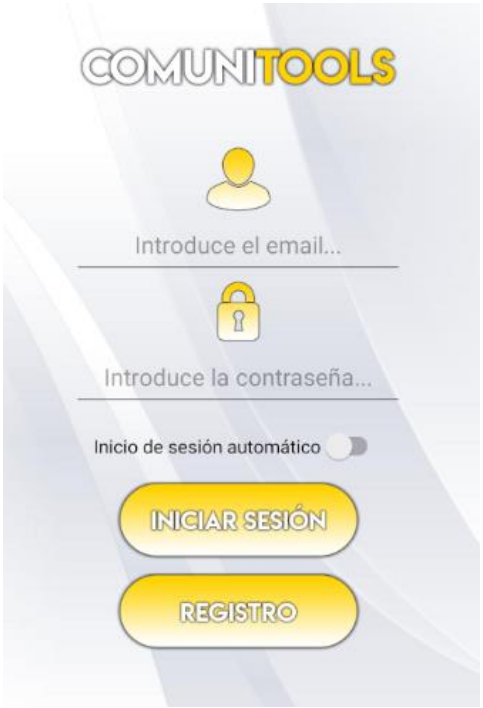
4. Tour por la aplicación.

4.1. Inicio de sesión.

Se trata de la pantalla principal, donde el usuario de la comunidad podrá introducir directamente sus credenciales de acceso para iniciar sesión. Para ello, se ha creado un formulario básico, formado por:

- ✓ **Usuario.** Se ha introducido un EditText básico para recoger el dato.
- ✓ **Contraseña.** Se ha implementado el campo con un EditText modificado para que aparezca cifrada la contraseña.
- ✓ **Inicio de sesión automático.** Gracias a esta función, el usuario accederá automáticamente la sesión sin la necesidad de introducir sus datos de acceso cada vez que ejecute la aplicación. La función ha sido implementada con un switch, que comprueba si está activado o no a la hora de iniciar la sesión, guardando un token en los datos de la aplicación que almacena el dispositivo, siendo este modificado en caso de un posterior cierre de sesión.
- ✓ **Botón de inicio de sesión.**
- ✓ **Botón de registro.**

Si el usuario que inicia sesión no está dado de alta en ninguna comunidad, será redirigido a una pantalla donde podrá insertar el código de la comunidad correspondiente. En caso contrario, accederá directamente al menú principal.



4.2. Registro de nuevo usuario.

Se ha creado un formulario de registro para el usuario, formado por:

- ✓ **Email.** Campo de introducción de texto en formato email, para que pueda realizarse correctamente la validación.
- ✓ **Usuario.** Campo de texto normal, que va a reflejar el nombre del usuario al resto de la comunidad.
- ✓ **Piso.** Datos del piso del vecino.
- ✓ **Puerta.** Número de puerta del vecino.
- ✓ **Contraseña.** Campo de introducción de texto en formato de contraseña, que será la credencial para el posterior inicio de sesión.
- ✓ **Botón de confirmación.**
- ✓ **Botón de inicio de sesión.**

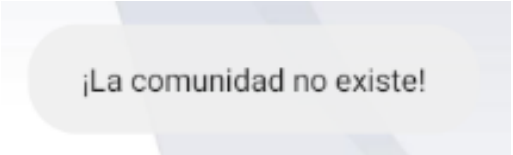
Tras confirmar el formulario, el usuario será registrado tanto en el sistema de autentificación de Firebase, como en la base de datos de la misma plataforma. Una vez se realiza esta acción, el usuario es redirigido a la pantalla de inicio de sesión para que pueda introducir sus credenciales.

4.3. Darse de alta en comunidad.

Tras el registro, el usuario recién dado de alta en la aplicación no pertenece a ninguna comunidad de vecinos, por lo que cuando inicia la sesión es redirigido a esta pantalla, donde tendrá dos posibilidades:

- ✓ **Rellenar el campo del código de comunidad.**
- ✓ **Acceder al panel de creación de una comunidad.**

En el proceso de confirmación, el algoritmo programado realiza una comprobación de comunidad, y en caso de que esta no exista aparece un mensaje de aviso para el usuario.



En el caso de que la comunidad exista, se actualiza el campo de comunidad de los atributos del usuario que mantiene la base de datos y el usuario estará dado de alta correctamente y podrá acceder al menú principal de la comunidad.

4.4. Crear comunidad.

Cualquier usuario registrado podrá crear una comunidad, accediendo a esta función desde la pantalla anterior de inserción de comunidad. Para crearla, el usuario debe completar el formulario formado por:

- ✓ **Nombre.**
- ✓ **Código.**
- ✓ **Saldo.**
- ✓ **Banco. (Opcional)**
- ✓ **Número de cuenta bancaria. (Opcional)**

Además, cuando el usuario crea la comunidad, automáticamente se actualizan en la base de datos los campos de:

- ✓ **Comunidad.**
- ✓ **Permiso de presidente.**
- ✓ **Banco y número de cuenta. (Si son introducidos)**

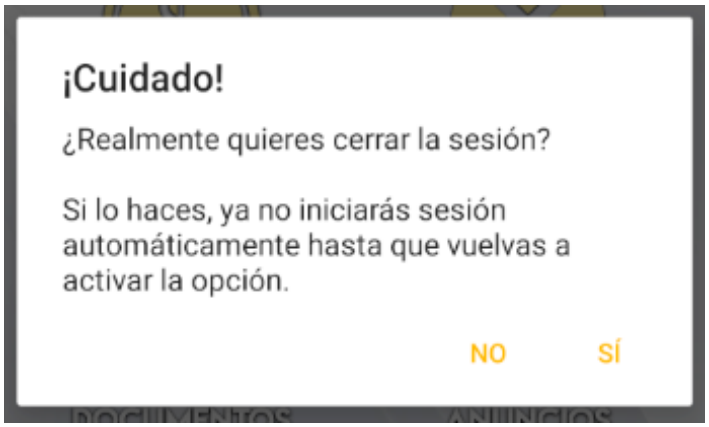
4.5. Menú principal.

Una vez el usuario inicia sesión y está dado de alta en una comunidad, accede al panel de funciones, que está formado por:

- ✓ **Comunidad.**
- ✓ **Contactos.**
- ✓ **Pagos.**
- ✓ **Incidencias.**
- ✓ **Documentos.**
- ✓ **Anuncios.**
- ✓ **Reuniones.**
- ✓ **Ajustes.**
- ✓ **Cerrar sesión.**

El usuario podrá acceder a cualquier funcionalidad, aunque dentro de cada una no podrán realizar las mismas acciones los usuarios con privilegios de presidencia o secretaría que los usuarios normales.

Además, existe la posibilidad de cerrar sesión, apareciendo una ventana emergente que pide la confirmación de la acción:



4.6. Ajustes.

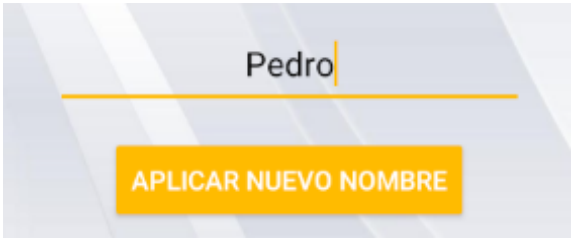
Cualquier usuario puede acceder a estas funciones para modificar aspectos relacionados con su perfil en la comunidad como:

- ✓ **Foto de perfil.**

  El usuario podrá previsualizar la imagen que tiene de perfil, y si selecciona una nueva tendrá la opción de ver cómo quedaría esta antes de cambiarla. Una vez la modifica, se sube al almacenamiento en la nube y la base de datos modifica el campo correspondiente a la imagen.

- ✓ **Nombre de usuario.**

  Se modifica el campo del usuario en la base de datos con el nuevo nombre.

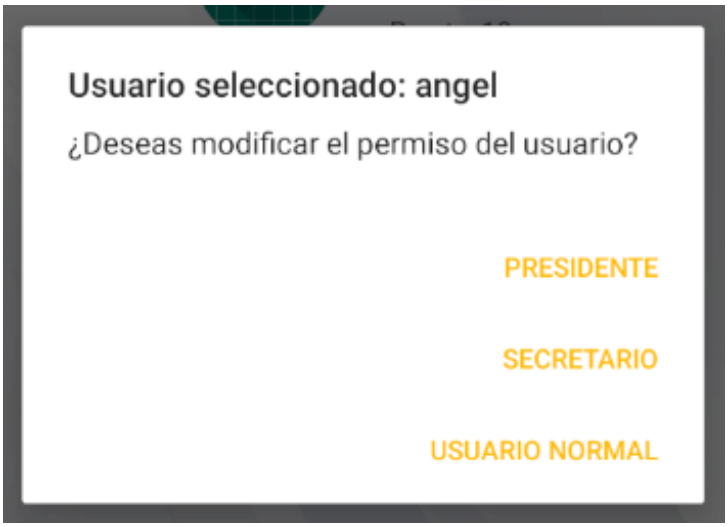


4.7. Comunidad.

Se listan todos los integrantes de la comunidad a la que pertenece el usuario que accede a esta sección, pudiendo comprobar quiénes son el presidente, el secretario y resto de propietarios.

Además, los usuarios con la insignia de presidente tendrán disponible la opción de modificar el permiso del resto de usuarios, pudiendo así rotar la presidencia y secretaría de forma clara e intuitiva. Esta acción se puede aplicar a cada

usuario pulsando en su nombre, apareciendo así una ventana emergente que permite seleccionar el permiso deseado.



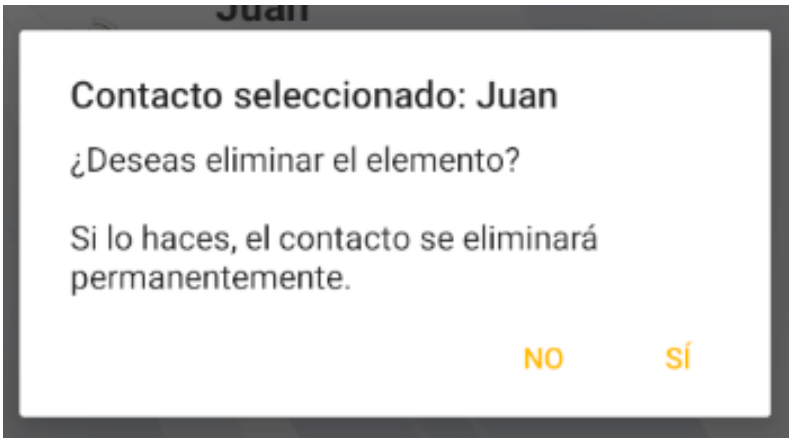
4.8. Contactos.

En esta sección los usuarios podrán ver una lista de contactos de interés relacionados con la comunidad.

Además, tanto presidencia como secretaría, podrán añadir nuevos contactos, rellenando un formulario con los siguientes campos:

- ✓ **Nombre.**
- ✓ **Descripción.**
- ✓ **Teléfono.**

A modo de gestión, los usuarios con privilegios administrativos podrán modificar los datos del contacto, además de poder eliminar el elemento.



4.9. Pagos.

En esta pestaña los usuarios podrán consultar el saldo comunitario, ya que es una información que todos pueden conocer. Además, podrán consultar los gastos debido a las incidencias y otros factores externos.

Cuando el coste de las incidencias se modifica, automáticamente se actualiza el valor del saldo actual, pudiendo comprobar en todo momento las modificaciones.

En el Punto 5 se mencionan las mejoras que se van a llevar a cabo de cara al futuro para que cobre aún más importancia la funcionalidad.

4.10. Incidencias.

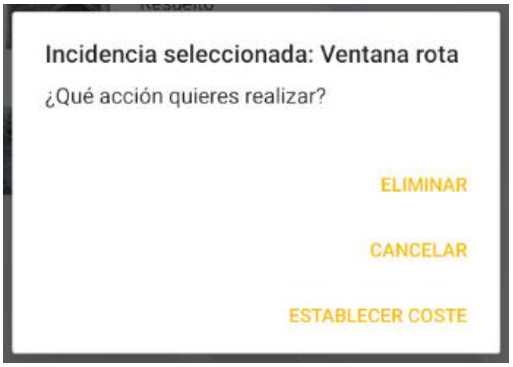
Esta funcionalidad permite a cualquier usuario de la comunidad revisar todas las incidencias que se han producido, además de tener disponible la función de añadir. Gracias a este formulario, la presidencia será consciente de los posibles problemas que haya que solucionar. El formulario contiene:

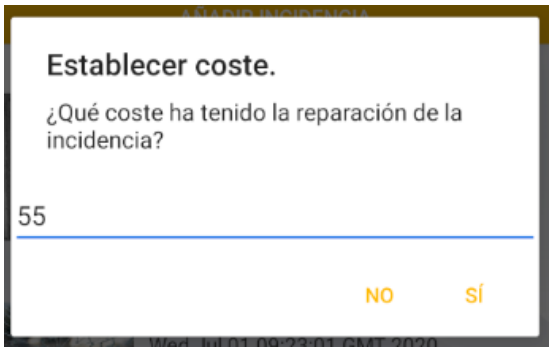
- ✓ **Título.**
- ✓ **Descripción.**

- ✓ **Imagen.**
- ✓ **Fecha.**
- ✓ **Estado.**
- ✓ **Coste.**

El usuario visualizará en la pantalla todas las incidencias con todos sus detalles:

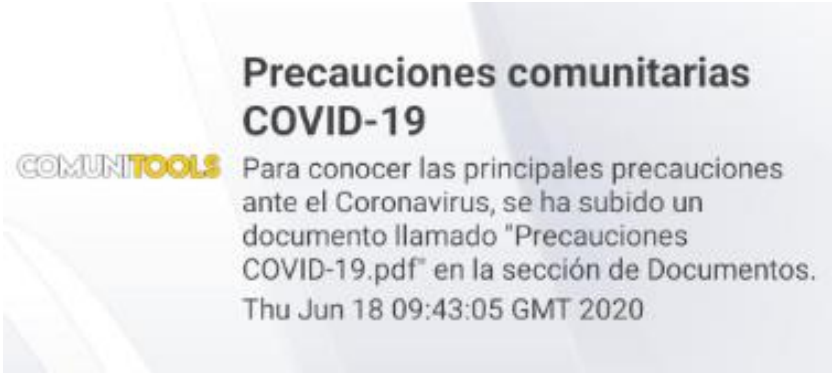
El presidente o secretario tendrán disponible la opción de establecer un coste de reparación, y este será añadido al historial de incidencias además de restar la cantidad establecida al saldo existente. En caso de que se modifique nuevamente el coste de la reparación, el anterior será sumado al saldo en ese preciso instante y posteriormente se le restará el valor del nuevo coste.



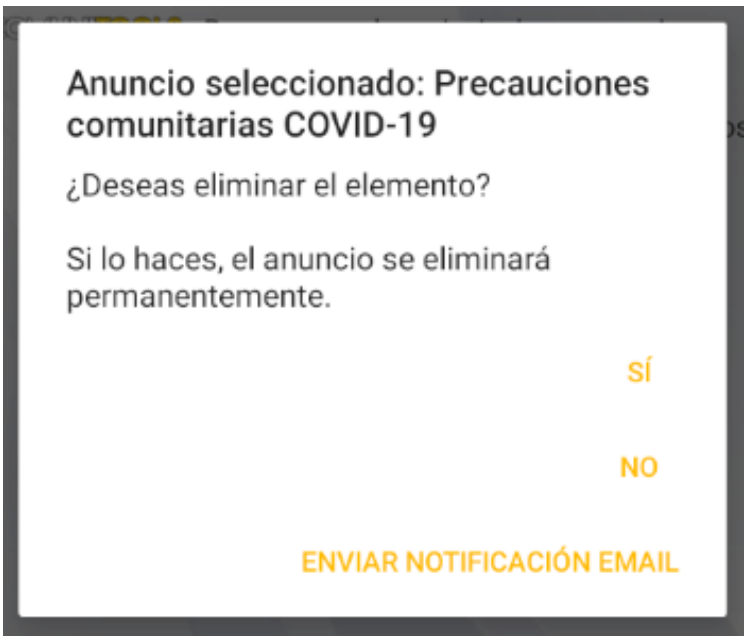


4.11. Anuncios.

Esta función está disponible para la presidencia y la secretaría, permitiendo así enviar cualquier notificación acerca de la comunidad o de algún tema interesante para el vecindario.

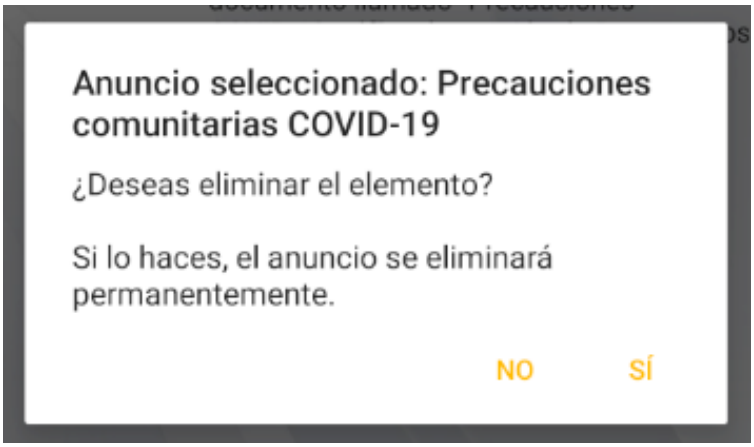


Es una funcionalidad destinada a informar, por lo que se ha añadido una función que permite a presidencia y secretaría enviar un correo electrónico a todos los usuarios de la comunidad mencionando la existencia de un nuevo anuncio. Los servicios email podrían detectar un comportamiento de spam, por lo que la aplicación recopilará todos los datos necesarios para el correo electrónico (título, descripción y destinatarios) y redirigirá al usuario a su cliente de email (aplicación de Gmail). Además, los usuarios con privilegios de administración pueden eliminar los anuncios pulsando sobre el ítem que desean quitar de la base de datos.

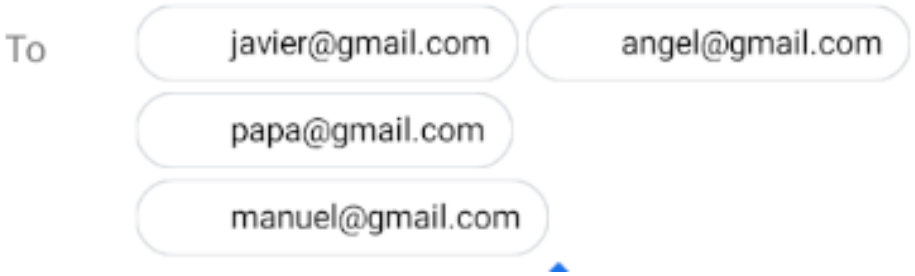


A la hora de añadir un nuevo anuncio, se envían los siguientes datos:

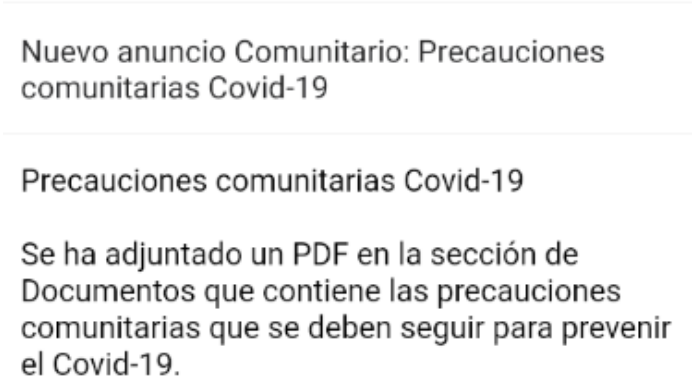
- ✓ **Título.**
- ✓ **Descripción.**
- ✓ **Fecha.**



Si se accede a la funcionalidad “Enviar notificación email” la aplicación prepararía automáticamente un correo electrónico con la siguiente estructura:



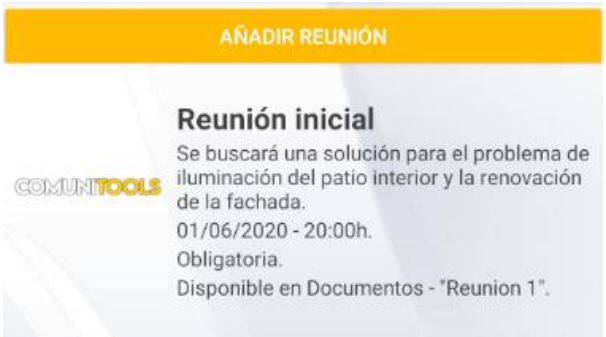
Se genera la lista de los emails de los integrantes.



Como título del correo electrónico se establece “Nuevo anuncio Comunitario” seguido del título del anuncio. En el cuerpo se introduce la descripción detallada.

4.12. Reuniones.

Un apartado primordial, ya que gracias a él los usuarios podrán consultar en cualquier momento las reuniones que se vayan a celebrar, además de poder introducir el evento en el calendario integrado del dispositivo.

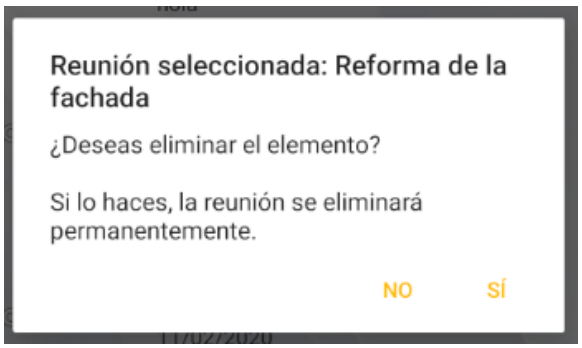


Vista desde el dispositivo del presidente.

Solamente podrán añadir nuevas reuniones presidencia y secretaría, rellenando los siguientes campos:

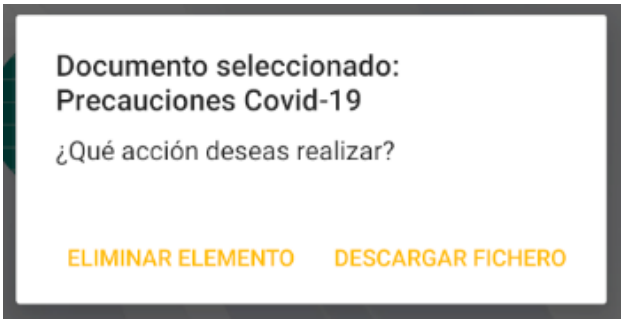
- ✓ **Título.**
- ✓ **Acta.**
- ✓ **Descripción.**
- ✓ **Fecha y hora.**
- ✓ **Asistencia.**

Tanto la presidencia como la secretaría pueden eliminar los elementos correspondientes a las reuniones pulsando el nombre de la reunión a eliminar y posteriormente confirmando en la ventana emergente.



4.13. Documentos.

En esta sección, el usuario podrá consultar todos los documentos adjuntos de la comunidad, por lo que debe ser capaz de descargar los ficheros sin mayor problema.



"Eliminar elemento" estará solamente disponible para los usuarios con privilegios

Tanto presidencia como secretaría podrán subir nuevos documentos, indicando título y descripción, además de abrir el documento deseado abriendo automáticamente el gestor de archivos integrado del dispositivo.

Si se selecciona la opción de descargar fichero la aplicación reenviará al usuario al navegador web para que pueda visualizarlo o descargarlo en su dispositivo.

**** Resumen de los permisos de los usuarios.**

	Presidencia	Secretaría	Usuario normal
Modificar permisos de usuarios	Sí	No	No
Añadir y eliminar contactos	Sí	Sí	No
Añadir incidencias	Sí	Sí	Sí
Eliminar incidencias	Sí	Sí	No
Modificar datos bancarios	Sí	Sí	No
Añadir y eliminar documentos	Sí	Sí	No
Añadir y eliminar anuncios	Sí	Sí	No
Añadir y eliminar reuniones	Sí	Sí	No
Modificar foto de perfil del propio usuario	Sí	Sí	Sí

5. Conclusiones y trabajos futuros.

Gracias al desarrollo de la aplicación para la gestión de comunidades ComuniTools se pueden sacar varias conclusiones:

- ✓ **Producto eficaz e intuitivo.**

La aplicación debe cumplir con las principales funcionalidades para que el usuario sienta que la herramienta realmente le aporta valor y nota como aumenta su productividad, por lo que no es necesario desarrollar grandes funcionalidades a nivel de complejidad, sino las correctas de una forma orientadas a la comodidad del usuario.

Además, a nivel de diseño es necesario que la aplicación de una imagen seria y legible, ya que el usuario debe poder navegar por los menús conociendo en todo momento lo que está haciendo.

- ✓ **Internet y comunicación.**

Sin duda internet se ha convertido en un pilar fundamental para las aplicaciones móviles, ya que brinda a los usuarios seguridad y efectividad en cualquier momento.

En el caso de ComuniTools ha sido imprescindible disponer de internet, ya que sin este no se podrían comunicar los vecinos de la comunidad. Y es que la comunicación para gestionar cualquier agrupación u organización es imprescindible, por lo que el valor que se aporta creando una herramienta así brinda grandes probabilidades de éxito para el desarrollador.

- ✓ **Testing.**

Para un desarrollador es imprescindible contar con la opinión de personas cercanas sin experiencia en el campo del desarrollo software y que podrían ser usuarios de la aplicación en la que está trabajando, ya que pueden aportar ideas orientadas al bienestar de ellos mismos dentro de la aplicación, es decir, su objetivo es que puedan manejarse por los menús de forma independiente y sin la necesidad de una ayuda externa.

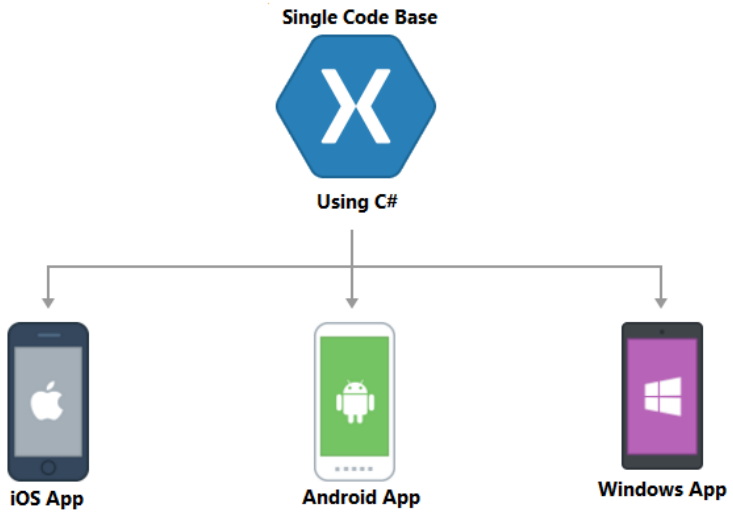
También es imprescindible realizar pruebas con diferentes dispositivos de cara a posibles errores dependiendo de la versión del sistema operativo e incluso la resolución de la pantalla.

De cara a posibles mejoras de la herramienta en un futuro, se han considerado algunas formas de pulir las funcionalidades con un equipo más extenso de trabajo:

- **Versión para Apple.**

A la hora de mantener a toda una comunidad en la aplicación, es conveniente maximizar la compatibilidad de la herramienta para que cada usuario pueda accederla desde prácticamente cualquier dispositivo móvil, por lo que programar una versión de la aplicación para Apple ayudaría a conseguir los objetivos de la comunidad.

Una gran idea para lograr este cometido sería programar la aplicación con el entorno de desarrollo Visual Studio y la plataforma de código abierto Xamarin, ya que esta permite compilar aplicaciones compatibles con iOS, Android e incluso Windows, brindando grandes posibilidades al desarrollador para convertir la aplicación en una herramienta multiplataforma.



- **Google Calendar.**

  A día de hoy, los consumidores de smartphones disponen de numerosas herramientas para mejorar su productividad en su día a día, y una de las más utilizadas es el calendario integrado en el terminal, por lo que añadir una función que posibilite a los usuarios agregar eventos comunitarios al calendario puede mejorar la calidad de la aplicación y aumentar la confianza de los vecinos.

  Esto sería posible utilizando el Google Calendar API, ya que permite al desarrollador acceder a herramientas que permiten conectar variables de la aplicación con el Calendario del dispositivo. Se utiliza una variable del tipo Event, a la cual se le van añadiendo parámetros como la localización, descripción, fecha, hora y muchos más detalles. En nuestro caso, se insertarían los detalles de la reunión de forma automática, pasando todos los datos al evento.

```
Event event = new Event()
    .setSummary("Google I/O 2015")
    .setLocation("800 Howard St., San Francisco, CA 94103")
    .setDescription("A chance to hear more about Google's developer products.");

DateTime startDateTime = new DateTime("2015-05-28T09:00:00-07:00");
EventDateTime start = new EventDateTime()
    .setDateTime(startDateTime)
    .setTimeZone("America/Los_Angeles");
event.setStart(start);
```

Extraída de www.developers.google.com/calendar/

- **Mejora de las notificaciones push.**

  Sin duda es una mejora imprescindible de cara al futuro, ya que incrementaría el interés del usuario por la aplicación de forma notable. Se debe tratar evitar que el usuario deba estar atento a otra aplicación externa (correo electrónico) para ser notificado de todas las novedades comunitarias.

  Se ha realizado un estudio para conocer el campo de desarrollo que se debe abordar para cumplir con la función.

Extraída de www.firebase.google.com

Firebase tiene dos opciones para enviar mensajes a dispositivos:

✓ **Firebase Console.**

Desde el panel de control de la página web de Firebase se pueden enviar notificaciones a dispositivos específicos, grupos de usuarios y temas a los que un sector de usuarios está suscrito.

✓ **Firebase Cloud Messaging API.**

La API puede enviar mensajes tras facilitar la clave de servidor, por lo que esta acción se realiza desde un servidor de aplicaciones. El mensaje se envía en formato JSON, pudiendo éste contener diferentes campos:

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Portugal vs. Denmark",
      "body":"great match!"
    },
    "data" : {
      "Nick" : "Mario",
      "Room" : "PortugalVSDenmark"
    }
  }
}
```

Extraída de www.firebase.google.com

Las peticiones de envío de mensajes deberían seguir el protocolo HTTP, dirigiendo a estas al extremo:

https://fcm.googleapis.com/fcm/send.

Otra alternativa sería mediante el uso de la sintaxis XMPP, conectándose la aplicación a los extremos:

fcm-xmpp.googleapis.com:5235 (Producción)
fcm-xmpp.googleapis.com:5236 (Testing)

- **Gestión de pagos mensuales del usuario.**

  El usuario podrá revisar sus cuentas y la cantidad de dinero que debe ingresar en la cuenta del banco de la comunidad. Esto permitirá a la presidencia y secretaría administrar la morosidad que pueda darse dentro de la comunidad.

- **Generación de informes económicos.**

  Una mejora que se desea realizar es la creación de una funcionalidad que genere un informe con el historial de pagos de la comunidad, ya que el usuario podrá consultar en cualquier momento todos los movimientos económicos que se han producido.

  Para abordar esta mejora, se utilizará el paso de variables de cara a generar un documento en formato PDF.

  El Android SDK ofrece opciones de cara a imprimir información en documentos gracias al PrintManager, que contiene un adaptador de impresión que incluye todo el proceso de generación del documento. Además, ofrece la posibilidad de personalizar el documento con colores y demás propiedades.

Otra opción disponible es mediante el uso de la función File (FileOutputStream) combinada con las posibilidades que brinda la clase PdfWriter, donde se pueden ir añadiendo datos al documento y descargarlo posteriormente.

Extraída de www.academiaandroid.com

Además, es imprescindible conocer la experiencia de los usuarios comunes de la aplicación, ya que serán los principales focos de cara a nuevas ideas y mejoras de las funcionalidades que se presentan, por lo que habrá que acercarse al cliente y que expongan los principales problemas que se puedan dar.

6. Bibliografía.

¿Qué es Android? (s. f.). Android. Recuperado 10 de marzo de 2020, de https://www.android.com/intl/es_es/what-is-android/

Sitio oficial para desarrolladores de apps para Android. (s. f.). Android Developers. Recuperado 12 de marzo de 2020, de https://developer.android.com/

Firebase. (s. f.). Google Firebase. Recuperado 14 de marzo de 2020, de https://firebase.google.com/

Curso de Desarrollo de Apps Móviles. (s. f.). Google Actívate. Recuperado 9 de marzo de 2020, de https://learndigital.withgoogle.com/activate/course/apps

Curso de Android con Firebase. (s. f.). CódigoFacilito. Recuperado 13 de marzo de 2020, de https://codigofacilito.com/cursos/android-firebase

Android: Listas dinámicas usando RecyclerView. (s. f.). Programacionymas. Recuperado 23 de marzo de 2020, de https://programacionymas.com/blog/listas-dinamicas-android-usando-recycler-view-card-view

Hathibelagal, A. (2018, 24 enero*). Comenzando Con Cloud Firestore para Android.* Code Envato Tuts+. Recuperado 20 de marzo de 2020, de https://code.tutsplus.com/es/tutorials/getting-started-with-cloud-firestore-for-android--cms-30382

Ciencia de la Computación e IA. (2012). *Introducción a Android [Libro electrónico].*
http://www.jtech.ua.es/dadm/restringido/android/sesion01-apuntes.pdf

Bosonit. (2019, 18 diciembre). *Tipos de bases de datos NoSQL 2: Clave-valor, documentales y columnares.* Recuperado 20 de marzo de 2020, de https://bosonit.com/tipos-de-bases-de-datos-nosql-2-clave-valor-documentales-y-columnares/

Acens. (2014*). Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar.* https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

Jessica Clark. (2020, 17 marzo). *Parse vs. Firebase. Back4App Blog.* Recuperado 14 de marzo de 2020, de https://blog.back4app.com/parse-vs-firebase/

Ramírez, I. (2019, 21 marzo). *Historia y evolución de Android. Xataka Android.* Recuperado 01 de abril de 2020, de https://www.xatakandroid.com/sistema-operativo/historia-y-evolucion-de-android-como-un-sistema-operativo-para-camaras-digitales-acabo-conquistando-los-moviles

Ayuntamiento de Málaga. Área de Participación Ciudadana, Inmigración y Cooperación al Desarrollo. (s. f.). *Guía para la convivencia en las comunidades de vecinos y vecinas.* *https://participa.malaga.eu/portal/menu/portada/documentos/Guxa_de_Comunidades.pdf*

Riquelme Leiva, M. (2015, junio). *Las 5 Fuerzas de Porter – Clave para el Éxito de la Empresa Las 5 Fuerzas de Porter. 5FuerzasDePorter.* Recuperado 2 de abril de 2020, de https://www.5fuerzasdeporter.com/

Ignacio Santiago. (2018, 30 octubre). *Informe del Estado de las Apps en España [INFOGRAFÍA].* Recuperado 2 de abril de 2020, de https://ignaciosantiago.com/informe-del-estado-de-las-apps-en-espana/