GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO DE UNA RED COLABORATIVA DE AUDITORÍA DE CAMPAÑAS DE PUBLICIDAD EXTERIOR

Autor: Miguel Enrile Fernández de Arévalo

Director: Alberto Saquero Rodríguez

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño de una red colaborativa de auditoría de campañas de publicidad exterior

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2018/19 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Miguel Enrile Fernández de Arévalo Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Alberto Saquero Rodríguez Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D._____

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: _____, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a …….. de ……... de …….

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO DE UNA RED COLABORATIVA DE AUDITORÍA DE CAMPAÑAS DE PUBLICIDAD EXTERIOR

Autor: Miguel Enrile Fernández de Arévalo

Director: Alberto Saquero Rodríguez

Madrid

Agradecimientos

A mis padres y a mi familia por el apoyo y la confianza a lo largo de estos años.

DISEÑO DE UNA RED COLABORATIVA DE AUDITORÍA DE CAMPAÑAS DE PUBLICIDAD EXTERIOR

Autor: Enrile Fernández de Arévalo, Miguel.
Director: Saquero Rodríguez, Alberto.
Entidad Colaboradora: dePoste

RESUMEN DEL PROYECTO

Desarrollo de una aplicación para el sistema operativo Android para diseñar una red colaborativa de auditoría de campañas de publicidad exterior. El resultado final cumple con las especificaciones para poder ser publicado en el Marketplace de Android ("Playstore").

Palabras clave: Android, App, Auditoría, publicidad exterior

1. Introducción

La empresa dePoste[1] , ubicada en Madrid, se dedica principalmente a la venta de espacios y colocación de anuncios de publicidad exterior. Tratan de dar este servicio de la manera más innovadora y cómoda para el usuario posible, como se puede ver en su página web, dónde en 3 sencillos pasos se puede iniciar una campaña de publicidad exterior.

La aplicación desarrollada permite al usuario realizar auditorías de publicidad exterior de una manera sencilla e interactiva, de manera que para el usuario sea atractiva la idea de llevar a cabo auditorías de anuncios.

2. Definición del Proyecto

2.1 La aplicación deberá ser desarrollada para Android, cumpliendo esta todos los requisitos para que pueda subirse al Marketplace de Android "Play Store".
2.2 Debe ser una aplicación *Friendly* y muy fácil de utilizar (Ej, Acceso mediante correo electrónico y contraseña o mediante huella dactilar).
2.3 Las variables de costes, penalizaciones, tiempo, importe mínimo para transferencias al auditor y demás campos deberán ser parametrizables.

Las categorías básicas que deberá tener la aplicación son las siguientes:

- Localizador de soportes: En la cual salga un mapa con los soportes que necesitan una auditoría (se obtienen, con los datos del propio soporte y el valor de realizar dicha auditoria. Adicionalmente, valorar la posibilidad de que se puedan realizar filtros para que el auditor pueda afinar su búsqueda.

  Adicionalmente, dar la oportunidad al auditor de reservar una auditoria por un tiempo limitado, en caso de que no cumpla con el servicio, se le tendrá que aplicar una penalización para próximas reservas de auditoria.

- Alertas recientes: El auditor podrá visualizar las alertas que le han saltado al realizar una trayectoria.
- Mis auditorias: El auditor podrá visualizar las auditorias que ha realizado, el estado de las mismas (pendiente, aprobada, etc.) y un resumen final con los totales.
- Mi cartera: Resumen de todas las acciones que ha llevado a cabo, numero de auditorías, cuantía total, cuantía pagada, etc.)
- Mi cuenta: Datos personales del auditor. correo electrónico, Nombre, Apellidos, Fecha de nacimiento, Sexo, Numero de cuenta bancaria, Cambio de contraseña, etc.

3. Descripción del sistema

La aplicación tiene una pantalla de bienvenida, dónde el usuario se puede autenticar. Desde esa pantalla se puede ir a la de registro, en caso de que el usuario no tenga cuenta, o al menú principal, en caso de que el usuario introduzca un email y contraseña válidos.

Desde el menú principal, podrá acceder a todas categorías definidas anteriormente e interactuar con cada una de ellas de la manera oportuna.

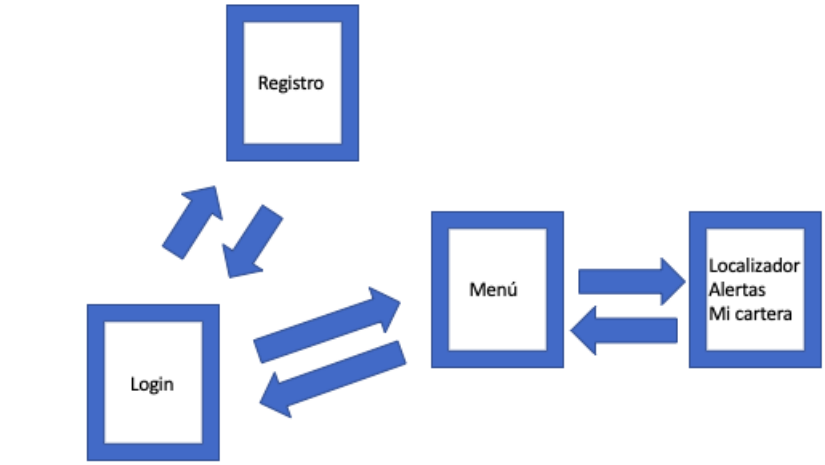


Ilustración 1 – Diagrama de flujo de la aplicación

4. Resultados

- Se ha conseguido el comportamiento deseado para la aplicación. A continuación, se incluyen algunas pantallas de la misma.

Ilustraciones 2,3,4 –(De izquierda a derecha) Login, menú principal y localizador de soportes

A través de la aplicación el usuario es capaz de realizar los servicios que son requeridos por la misma de manera sencilla e interactiva. La app toma los datos de la base de datos de dePoste, por lo que la información que muestra es modificable desde la web sin entrar en código Java.

5. Conclusiones

Con esta aplicación se consigue aportar valor tanto para el usuario de la aplicación (que es recompensado económicamente) como para el anunciante (cuyos anuncios son verificados por terceras personas). El Sistema se ha podido diseñar sobre Android y puede ser usada desde cualquier teléfono Android con cámara.

6. Referencias

[1] dePoste; Página web y contacto. https://www.dePoste.com/

DESIGN OF A COLLABORATIVE AUDITING NETWORK FOR EXTERNAL ADVERTISING CAMPAIGNS

Author: Enrile Fernández de Arévalo, Miguel.

Supervisor: Saquero Rodríguez, Alberto.

Collaborating Entity: dePoste

ABSTRACT

Development of an application for the Android operating system to design a collaborative network of auditing outdoor advertising campaigns. The result meets the specifications to be published in the Android Marketplace ("Playstore").

Key words: Android, App, Audit, External Advertising

1. Introduction

The company dePoste [1], located in Madrid, is mainly dedicated to the sale of spaces and placement of outdoor advertising ads. They try to give this service in the most innovative and comfortable way for the possible user, as you can see on their website, where in 3 simple steps you can start an outdoor advertising campaign.

The developed application allows the user to carry out external advertising audits in a simple and interactive way, so that the idea of carrying out ad audits is attractive for the user.

2. Definition of the project

2.1 The application must be developed for Android, fulfilling all the requirements so you can get on the Android Marketplace "Play Store".

2.2 It must be a Friendly application and very easy to use (Eg, Access via email and password or by fingerprint).

2.3 The variables of costs, penalties, time, minimum amount for transfers to the auditor and other fields must be parametrizable.

The basic categories that the application should have are the following:

• Locator of supports: In which a map with the supports that need an audit is displayed (obtained with the data of the support itself and the value of carrying out said audit.) Additionally, assess the possibility that filters can be made so that the Auditor can refine your search.

Additionally, give the auditor the opportunity to reserve an audit for a limited time, in case he does not comply with the service, he should apply a penalty for upcoming audit reserves.

• Recent alerts: The auditor will be able to visualize the alerts that he has received while he was not on the app.

• My audits: The auditor can visualize the audits he has done, the status of them (pending, approved, etc.) and a final summary with the totals.

• My portfolio: Summary of all the actions carried out, number of audits, total amount, amount paid, etc.)

• My account: Personal data of the auditor. email, Name, Surname, Date of birth, Sex, Bank account number, Change of password, etc.

3. Description of the system

The application has a welcome screen, where the user can log in. From this screen you can go to the registry, in case the user does not have an account, or to the main menu, in case the user enters a valid email and password. From the main menu, you can access all previously defined categories and interact with each of them in a timely manner.

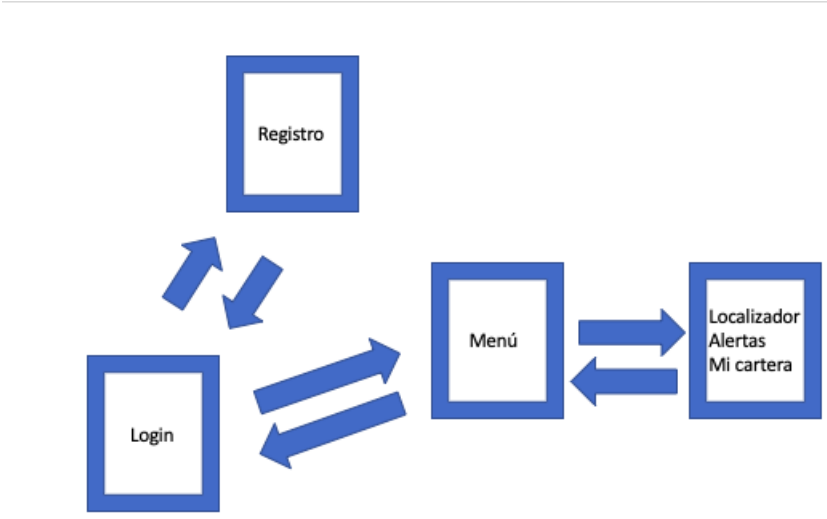
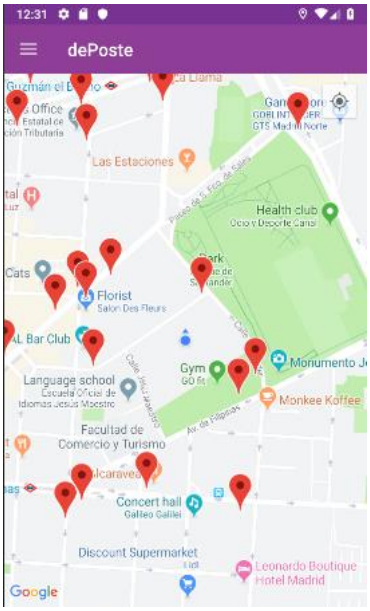


Image 3 Diagram of the app

4. Results

The desired behavior for the application has been achieved, then some screenshots of it are included.



Through the application the user can perform the services that are required by it in a simple and interactive way. The app takes the data from the dePoste database, so the information it shows is modifiable from the web without entering Java code.

5. Conclusions

With this application, it is possible to provide value for both the user of the application (who is rewarded financially) and the advertiser (whose ads are verified by third parties). The system has been designed on Android and can be used from any Android phone with camera.

6. Bibliografy

[1] dePoste; Página web y contacto. https://www.dePoste.com/

Índice de la memoria

Índice de figuras

Capítulo 1. INTRODUCCIÓN

La empresa dePoste, ubicada en Madrid, se dedica principalmente a la venta de espacios y colocación de anuncios de publicidad exterior. Tratan de dar este servicio de la manera más innovadora y cómoda para el usuario posible, como se puede ver en su página web, dónde en 3 sencillos pasos se puede iniciar una campaña de publicidad exterior.

Acorde con esta innovación previamente mencionada surgió una idea que podría revolucionar el mercado de la publicidad exterior tal y como lo conocemos. A día de hoy, al lanzar una campaña de publicidad, la empresa anunciante (la que contrata a los servicios de dePoste o cualquier otra empresa similar) debe confiar en cierto modo en la misma, ya que la empresa que busca anunciarse nunca podrá comprobar si los anuncios efectivamente están colocados o se ven como deberían. Esto, con fácil solución en los anuncios en la web (simplemente hay que ir a la web que hemos contratado y contrastar que el anuncio está ubicado dónde hemos contratado) se complica bastante en el mundo físico. Imaginemos una gran campaña con 500 anuncios repartidos por Madrid, la empresa anunciante no puede ir uno por uno comprobando que cada anuncio particular está instalado como se espera. Para ahorrar esta tediosa labor de comprobación, las propias empresas de publicidad llevan a cabo lo que se conoce como auditorías.

Actualmente, las auditorías (una auditoría de un anuncio consiste en verificar que el anuncio cumple con las expectativas en torno a su colocación, visibilidad, calidad…) en el campo de la publicidad exterior son realizadas por el propio personal interno que instala el anuncio, esto tiene algunos problemas que se exponen a continuación.

Ocurre no pocas veces que el anuncio no puede llegar a su público por diversos motivos; por ejemplo, un árbol que tapa la valla publicitaria, o un quiosco tapa las marquesinas de las paradas de autobús de tal manera que sólo se ve desde un ángulo muy

concreto (de tal manera que revisando la auditoría interna parece que el anuncio está bien instalado, aunque no sea así), lo cual desde luego no es lo que busca.

En un primer lugar queda el hecho de que se debe confiar en la empresa que coloca los anuncios (al contratar 300 anuncios deben dar 300 auditorías válidas), y, además, la propia empresa debe confiar en su empleado. Es posible que la persona que tiene que físicamente instalar el anuncio no realice un estudio de la visibilidad del mismo a unos niveles que el cliente final consideraría óptimos, y simplemente realice la auditoría de manera parcial (Ej:el cliente quedará satisfecho porque le parecerá que su anuncio en la marquesina de la parada bus lo verá cada persona que pase en coche por esa calle, cuando realmente por la colocación del quiosco es muy complicado de ver).

A continuación, se incluye un caso real encontrado el 2019-01-26 en la C/Raimundo Fdez Villaverde, en Madrid. Supongamos que somos el anunciante y contratamos el cartel del jugador de tenis.

Imagen 1 – Anuncio de ejemplo a la vista

Es posible que, al llevar a cabo la auditoría interna, por la propia persona que ha colocado el cartel, recibamos una imagen parecida a esta, y el cliente quede satisfecho con el resultado.

Desde luego, no nos esperemos que el cartel en realidad se ve así:

Imagen 2 – Anuncio de ejemplo oculto

(Aclaración: Los carteles en los postes de ese tipo son para anuncios de carácter público y se utilizan para anunciar temas culturales, eventos deportivos…, un anuncio privado no podría ubicarse en esa localización. Con este ejemplo se ha querido exponer el problema, aunque este caso en concreto no afecta a ninguna empresa privada, sino al ayuntamiento)

El cartel es completamente invisible desde cualquier punto, queda totalmente tapado. No se ve ni desde la acera ni desde la calzada por dónde circulan los coches, es decir, tener un anuncio ahí es completamente inútil. Es por ello que la aplicación puede suponer una auténtica ventaja competitiva frente a otras empresas que no aporten este servicio, los auditores externos expondrán el problema y la empresa anunciante podrá reclamar soluciones al respecto.

Para solucionar el problema anteriormente descrito en dePoste se les ocurrió la idea de diseñar una red colaborativa de auditoría de campañas de publicidad exterior a través de una aplicación para el móvil.

De esta manera, a cambio de una pequeña remuneración económica, serían los propios usuarios de la aplicación los que realizarían las auditorías (con la imparcialidad que ello conlleva), los usuarios obtienen la remuneración en base a factores que no dependen de si efectivamente el anuncio cumple las expectativas, si no a que muestren fielmente cómo se ve el anuncio.

La aplicación que se pretende desarrollar permitirá al usuario realizar lo mencionado anteriormente de una manera sencilla e interactiva, de manera que para el usuario sea atractiva la idea de llevar a cabo auditorías de anuncios.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para el desarrollo de la aplicación se han usado cuatro tecnologías principalmente.

Android Studio [1] – Entorno de desarrollo más usado para la creación de aplicaciones, incluye diversas funcionalidades como el manejo automático de máquinas virtuales, la ejecución sencilla de las aplicaciones creadas sobre terminales físicos, pre visualización de diseños de pantallas, etc. Debido a estas ventajas prácticamente todas las aplicaciones de Android se crean sobre esta plataforma.

Postman [2] – Herramienta para simplificar el desarrollo e implementación de APIs. Con esta herramienta se pueden realizar peticiones http/https de todo tipo, fue usada el desarrollo para depurar la app.

Volley [3] – Librería que implementa la comunicación https con la API, se ha convertido en un estándar a la hora de realizar comunicaciones a través de java.

Maps SDK for Android – Permite introducir mapas y modificarlos para el desarrollo de aplicaciones.

GPS [15] – "Global Positioning System" es un sistema de navegación por radio que permite a los usuarios conocer su posición y su velocidad. Es vital para la aplicación ya que ésta depende de la ubicación del usuario para la interacción con las auditorías.

Bases de datos GIS[16] – Es un sistema administrador de bases de datos que maneja datos espaciales. Estos datos espaciales son puntos (Latitud y Longitud), líneas (Unión de dos puntos) o polígonos (Unión de varias líneas). Debido a la información geo posicional, tienen una complejidad añadida, estas Bases de Datos tienen un lenguaje de consulta especial, llamado SSQL (Spatial SQL).

JSON[17] – Es un formato ligero de intercambio de datos. En la aplicación fue usado para la comunicación entre la BD y la aplicación. Al realizar una petición, la respuesta del servidor es un JSON. Es beneficioso porque al ser un formato en texto plano es sencillo de leer para los humanos, mientras que para las máquinas es sencillo generarlo e interpretarlo.

HTTPs – Protocolo de comunicaciones que permite la transferencia de información a través de la web. Es la versión segura del protocolo http.

Sistemas de autenticación por contraseña – Hay tres tipos de autenticación principales, a través de algo conocido (password), a través de algo poseído (Tarjetas de coordenadas) o a través de características físicas (reconocimiento por voz). En la aplicación se optará por autenticar a través de algo conocido, el usuario y la contraseña.

KML[19] – es un lenguaje de marcado basado en XML usado para representar datos geográficos en tres dimensiones.

Capítulo 3. ESTADO DE LA CUESTIÓN

La solución que se suele emplear al tipo de problema propuesto, a día de hoy, es a través del concepto de "economía colaborativa", definida como sigue:

"La economía colaborativa la conforman aquellos modelos de producción, consumo o financiación que se basan en la intermediación entre la oferta y la demanda generada en relaciones entre particulares, empresas o de particular a profesional, a través de plataformas digitales que no prestan el servicio subyacente , esto genera un aprovechamiento eficiente y sostenible de los bienes y recursos ya existentes e infrautilizados y permite utilizar, compartir, intercambiar o invertir los recursos o bienes, exista o no una contraprestación entre los usuarios"[4].

A día de hoy existen multitud de estos servicios que se dedican a distintos sectores como [5]:

1. Transporte colaborativo: Compartir tu viaje en coche. Ejemplos: BlaBla Car o Uber.
2. Alojamiento colaborativo: Compartir una habitación de tu casa o el apartamento completo cuando no está habitado. Ejemplos: Airbnb o HomeAway.
3. Comercio colaborativo: Compra venta de segunda mano. Ejemplos: Ebay, Wallapop o Chicfy.
4. Espacios colaborativos, en diversos aspectos como compartir maletero con Shipeer, compartir trastero con LetMeSpace o compartir espacio de trabajo y experiencias en los "coworking".

5. Otros tipos, donde tenemos ejemplos como Comprea (hacer la compra y recibirla en casa, el shopper gana un dinero extra y el comprador comodidad) o Compartoplato (compartir tu comida).

Analizando estos servicios mencionados anteriormente, vemos que la mayoría basan sus servicios en una aplicación para el móvil. Las claves que se pueden sacar de todas ellas son la fácil interacción con la interfaz (*user-friendly*), la rapidez de uso y la sencillez. Ninguna requiere complejos procesos para empezar a utilizarla, y en general desde que se descarga la app hasta que se puede usar no pasan más de 5 minutos.

A continuación, se investiga el estado del arte del desarrollo de aplicaciones para móvil. La siguiente imagen enseña a fecha de enero de 2019 la cuota de mercado para cada sistema operativo de móvil. Podemos ver que entre Android e IOS se reparten prácticamente el 98% del mercado.

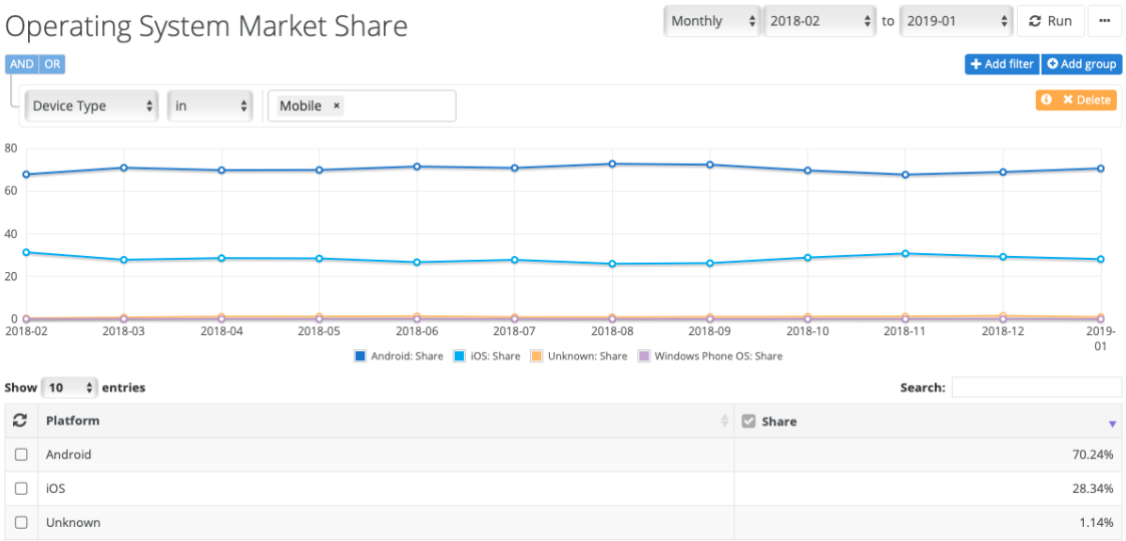


Imagen 3 – Uso de sistemas operativos [6]

"En países como España las diferencias son más significativas, donde Android tiene más del 90% de la cuota de mercado" [7].

El criterio de selección para decidir para qué plataforma se desarrollará la aplicación será el número potencial de usuarios que ésta nos pueda aportar. Mirando la estadística [8] anterior, tanto a nivel mundial como en España en particular, nos decidimos por desarrollar la aplicación para Google Play, Android.

La idea se va a llevar a cabo a través de una aplicación para Sistema Android, la cual servirá de plataforma para poner en contacto a los usuarios (o auditores externos) con los anuncios que requieren de una auditoría.

La solución que se va a desarrollar entraría en la categoría de las apps de economía colaborativa. A día de hoy, "El principal pilar de las aplicaciones de economía colaborativa es el valor de la confianza" [¿], según afirmaba Jaime Rodríguez, responsable de BlaBlacar España. Precisamente este es el pilar en el que se basa dePoste a la hora de llevar a cabo la aplicación, la confianza que pueden aportar a las auditorías de usuarios externos a la aplicación.

Estas aplicaciones consiguen aportar una solución escalable. En el caso concreto de dePoste, es más asequible usar este nuevo paradigma para realizar auditorías externas. Los auditores a cambio recibirán una recompensa variable. La empresa anunciante tiene la ventaja de poder acceder a un servicio externo de auditoría a un coste variable. Es por tanto una situación en la que todas las partes ganan.

dePoste intentará por tanto llevar a cabo su idea de construir un sistema de auditorías, a través de un modelo de economía colaborativa. Uno de los principales puntos a la hora de desarrollar la app, es que sea lo más *user-friendly* posible, ello es un punto fundamental según los expertos:

"Una de las claves del éxito de la economía colaborativa es que el foco se centra en las personas. Muchos de los proyectos de economía colaborativa han visto la luz para hacer frente a los problemas sociales. Unos modelos económicos disruptivos que se han convertido en un fenómeno imparable y que ha irrumpido a manos del cambio generacional para hacer frente a los modelos tradicionales." [8]

Este concepto de economía colaborativa encaja en este proyecto, ya que las acciones que se requieren de los usuarios son sencillas y rápidas de hacer, sin necesidad de una formación previa al respecto. Además, se quiere aprovechar el *momentum* de este tipo de economías en el mercado español, ya que "más de la mitad de los españoles usan apps de economía colaborativa" [9], [10].

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

El proyecto se va a llevar a cabo debido a que se estima que existe un gran nicho de mercado tanto para usuarios (hay varias aplicaciones similares como glovo que tienen una buena fuente de "riders", se entiende que, si hay gente dispuesta a trabajar en estas apps, también lo habrá para trabajar para dePoste realizando auditorías).

Se espera además que el proyecto completo arroje beneficios desde el primer año de su puesta en funcionamiento. (Para más información acerca de la cuantía de pagos a usuarios, métodos de pago, estimaciones… consultar el apartado "Estudio Económico"). El precio de la auditoría externa es muy barato en comparación con el de la contratación del anuncio (con factores de 100 la aplicación sería rentable), por lo que se entiende que los anunciantes querrán esa seguridad extra que le aporta el hecho de que las auditorías sean realizadas por personas externas a la empresa que instala la publicidad.

La aplicación forma por tanto un ecosistema dónde hay dos actores fundamentales, los anunciantes (que introducen dinero en el sistema y buscan auditorías a cambio) y los usuarios (proceso inverso). Por los motivos argumentados previamente esperamos que a ambas partes le interese participar en la aplicación, llevando a cabo un negocio rentable para dePoste.

4.2 OBJETIVOS

En este proyecto los objetivos fueron definidos conjuntamente entre el alumno y la empresa dePoste. El objetivo principal es la construcción de una aplicación móvil de auditoría, funcional. Se va a dividir este objetivo principal en cinco grupos de objetivos intermedios.

Estudio económico de costes y beneficios
Información del auditor
Evidencias
Realización de auditoría
Aplicación

4.2.1 INFORMACIÓN DEL AUDITOR

El auditor, después de descargarse la aplicación, deberá crearse una cuenta con los siguientes datos mínimos:

- Correo electrónico (será su ID único)
- DNI
- Nombre
- Apellidos
- Teléfono móvil
- Género
- Fecha de nacimiento

Dando el consentimiento de la finalidad para la que se va a utilizar su información y que funciones de su dispositivo móvil se van a utilizar (entre otros, será necesario utilizar su conexión WIFI y GPS para tener localizado al auditor y que le puedan salir alertas si tiene algún soporte cerca que pueda auditar y para la propia evidencia es necesario tener GPS con una precisión aceptable, permitir usar su cámara de fotos, etc.…).

Es importante que tanto el almacenamiento como el tratamiento de la información debe cumplir con la directiva GDPR[18]. Siguiendo con la directiva, se ha de dejar muy claro ciertos aspectos; como quién es el responsable de la gestión de la información, el plazo durante el que se conservarán los datos…

Además, con este reglamento el consentimiento cobra especial importancia, con el nuevo reglamento es imprescindible que se produzca una declaración del interesado, o una acción positiva, que muestre su conformidad. Es decir, si antes se podía establecer que el usuario al utilizar la aplicación acepta implícitamente el uso de sus datos por parte de la empresa, con esta directiva ya no. El consentimiento debe ser explícito.

Esta directiva también establece las medidas de seguridad que deben tomar las empresas para proteger los datos. Por ejemplo, se deben establecer accesos seguros a las bases de datos de las empresas, establecer procedimientos de copias de seguridad y por supuesto protegerse contra fugas de información e infecciones de malware…

Adicionalmente, cada auditor tendrá un detalle de los soportes que ha auditado, con el estado de verificación, valor de la auditoria y demás datos relevantes para el auditor.

Todos los datos personales del usuario son recogidos al registrarse en la app, en la pantalla de Registro. En principio el usuario sólo cuenta con una contraseña de texto común, aunque para futuras versiones se planteó poder llegar a usar otro tipo de contraseña como la huella digital, o incluso llevar a cabo una autenticación de dos factores.

4.2.2 EVIDENCIAS

Cuando el auditor añade una evidencia, debe cumplir los siguientes requisitos:

- GPS activo y precisión alta.
- Orientación
- Fecha
- Galería de fotos (una o más)

- Puntuación de la instalación de la publicidad.
- Semejanza entre la creatividad y la instalación.
- Puntuación de la visibilidad
- ¿Tiene soportes al lado?
- Notas adicionales (opcional)

La aplicación mediante la API, le pasará los parámetros al servidor para crear nuevas evidencias, así como leer las evidencias que tiene cada usuario para mostrárselas.

Los posibles estados de las evidencias serán: Reservada, Realizada, Verificada, Cancelada.

4.2.3 REALIZACIÓN DE AUDITORÍAS

Para poder empezar a realizar la auditoria de un soporte, se comprobará previamente que el auditor está cerca del soporte a auditar (distancia parametrizable según soporte).

Existe un campo en la auditoria que es **structure_mobility_id**, siendo el tipo de soporte que es [fijo, móvil o zona aproximada], en caso de que sea fijo sí que tiene que cumplir que el usuario esté cerca del soporte para tomar las evidencias, pero en los otros casos no hace falta. Una vez que empiece la auditoría, el soporte quedará bloqueado para que los otros auditores no puedan realizar la auditoria. Para ello se crea la evidencia en *status* reservada y cuando termine se pasa a estado realizada.

Un auditor podrá reservar una auditoria, aunque no se encuentre en el lugar donde está ubicado el soporte (con el fin de que dos auditores no se desplacen a un lugar alejado de su zona y cuando estén allí ya esté el soporte auditado). Se reservará durante X tiempo (parametrizable). En caso de que no cumpla con la reserva en el tiempo establecido, no podrá reservar la auditoria de ese soporte (aunque podrá desplazarse y realizar la auditoria sin

reservar). En caso de que haga muchas reservas (parametrizable) sin realizar, se le aplicará una penalización.

4.2.4 APLICACIÓN

La aplicación deberá ser desarrollada para Android, cumpliendo esta todos los requisitos para que pueda subirse al Marketplace de Android "Play Store".

Debe ser una aplicación Friendly y muy fácil de utilizar (Ej, Acceso mediante correo electrónico y contraseña o mediante huella dactilar).

Las variables de costes, penalizaciones, tiempo, importe mínimo para transferencias al auditor y demás campos deberán ser parametrizables.

Las categorías básicas que deberá tener la aplicación son las siguientes:

- Localizador de soportes: En la cual salga un mapa con los soportes que necesitan una auditoria (se obtienen, con los datos del propio soporte y el valor de realizar dicha auditoria. Adicionalmente, valorar la posibilidad de que se puedan realizar filtros para que el auditor pueda afinar su búsqueda.

  Adicionalmente, dar la oportunidad al auditor de reservar una auditoria por un tiempo limitado, en caso de que no cumpla con el servicio, se le tendrá que aplicar una penalización para próximas reservas de auditoria.

- Alertas recientes: El auditor podrá visualizar las alertas que le han saltado al realizar una trayectoria.
- Mis auditorias: El auditor podrá visualizar las auditorias que ha realizado, el estado de las mismas (pendiente, aprobada, etc.) y un resumen final con los totales.
- Mi cartera: Resumen de todas las acciones que ha llevado a cabo, numero de auditorías, cuantía total, cuantía pagada, etc.)

- Mi cuenta: Datos personales del auditor. correo electrónico, Nombre, Apellidos, Fecha de nacimiento, Sexo, Numero de cuenta bancaria, Cambio de contraseña, etc.

4.3 METODOLOGÍA

Para realizar la aplicación se optó por ir realizando sesiones programadas con el director del TFG para la aclaración de dudas, el planteamiento de objetivos a corto plazo y a través de un seguimiento semanal.

4.4 PLANIFICACIÓN

Para la estimación económica ver "Estudio económico".

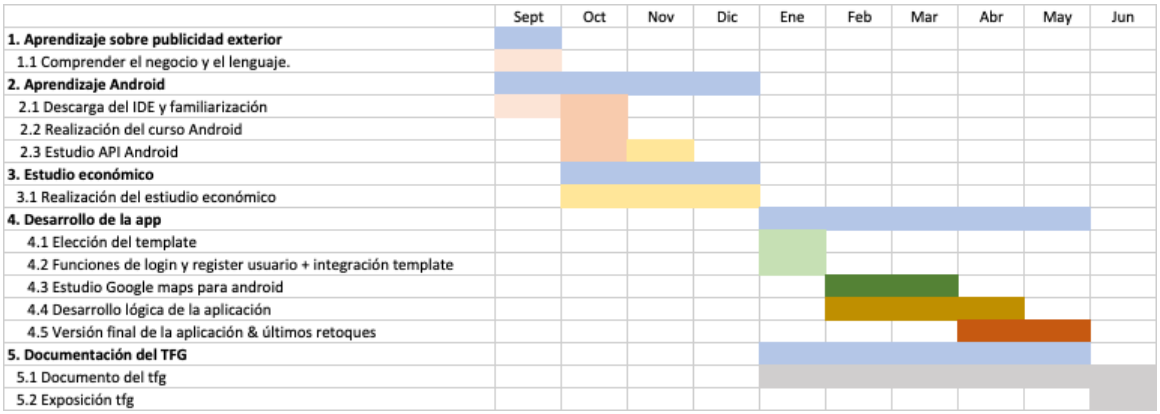


Figura 1 –Planificación

Se puede observar que los primeros meses eran dedicados a la formación en Android, esto es debido a que para llevar a cabo una aplicación son necesarios una gran cantidad

de conocimientos "base", sobre los cuales se desarrolla la aplicación. Antes de Navidad también se llevó a cabo el estudio económico, confirmando la viabilidad económica de la puesta en marcha del proyecto.

En enero se empezó con la elección de los estilos de la aplicación (para ello se hizo uso de plantillas de estilo, que sirvieron como base para el diseño visual) y con las primeras pantallas. Hasta finales de mayo se trabajó en el core de la aplicación, intentando implementar la máxima funcionalidad posible. Paralelamente se fue documentando. Para junio se dejó únicamente la finalización de la documentación y posibles retoques de la aplicación para la empresa, que no quedarán reflejados en la documentación del TFG por la naturaleza iterativa del desarrollo de las aplicaciones, aunque no podrán contener nada de importancia sobre el funcionamiento de la aplicación.

4.5 ESTUDIO ECONÓMICO

El primer punto del que depende el estudio económico es el pago que se le realiza a los usuarios que realizan auditorías para la aplicación. Los pagos son variables debido a que hay anuncios por los que dePoste está dispuesto a pagar más que otros (por ejemplo, no supone el mismo esfuerzo auditar un anuncio en el centro de la ciudad en una marquesina que una valla publicitaria en Alcobendas).

Para asignar, entonces, el pago a realizar para cada auditoría parece ideal una función ponderada por coeficientes.

La compensación será variable y se basará en la dificultad para obtener la evidencia, en términos generales, esta dificultad se puede caracterizar por los siguientes factores:

1. Tipo de soporte

   Debido a que la dificultad de la auditoría puede variar en función del objeto a fotografiar/valorar.

   1.1 mupis, opis, marquesinas (dificultad baja)
   1.2 relojes y barómetros (dificultad media-baja)
   1.3 lonas en fachadas (media)
   1.4 vallas publicitarias (alta)
   1.5 medios de transporte públicos en movimiento (muy alta)

   Los coeficientes van asociados a cada dificultad y serán

   Baja – 0

   Media- baja -0,15

   Media - 0,3

Alta - 0,7

Muy alta – 1

Este coeficiente será alfa(α) en la fórmula final.

2. Ubicación de la imagen

Debido a que la distancia a un núcleo urbano puede ser un factor determinante a la hora de decidirse a realizar la auditoría.

2.1 baja (0-3km) -- 0 (se encuentra en el mismo núcleo urbano o alrededores)
2.2 media (3-10KM) –0,2
2.3 Alta (>10KM) –0,5

El motivo por el que he pensado que es mejor medir la distancia en función a núcleos urbanos y no a dónde se ubiquen directamente (carretera, camino, calle normal…), es que es posible que en los pueblos (zonas poco pobladas entre ciudades) salgan precios exageradamente altos cuando realmente a sus habitantes es posible que no les cueste tanto llegar a la auditoría a realizar.

Además, para incentivar a los auditores en zonas de menor densidad de población, dónde se entiende que será más complicado encontrar un auditor dispuesto a realizar el trabajo, se podría sumar a los valores anteriores una parte (por ejemplo 0,2). En principio habría que observar los resultados sin implementar esta funcionalidad ya que en una ciudad con menos población se entiende que habrá menos anuncios auditables, por lo que en principio el

ecosistema podría subsistir sin necesidad de aumentar el precio incentivo de los auditores. En cualquier caso, aquí queda planteado.

Este será el coeficiente beta(β).

3. Calidad, fidelidad del usuario

   Sistema de fidelidad para retener usuarios y fomentar las buenas auditorías, un usuario cobra más cuanto mejor y más auditorías haga.

   Coeficiente Z (0-0,2).

   El coeficiente Z lleva a su vez cierta complejidad implícita, ya que habrá que tener en cuenta la regularidad con la que se realizan auditorías. Un método, no muy complejo, puede ser simplemente que si en la última semana ha realizado una auditoría satisfactoria, se otorga el bonus entero, si no, 0.

4. Urgencia

   En caso de que exista urgencia por una auditoría en concreto, se puede incrementar su recompensa

   Coeficiente δ (0-1)

5. Cuantía mínima básica

   Lo mínimo por lo que una persona está dispuesta a sacar el móvil y realizar una auditoría, podemos estimar entre (0,5 – 2€).

De esta manera, el coste por una auditoría sería una combinación de los parámetros anteriores, resultando algo tal que así:

Auditoría (€) => $A = x + \alpha * x + \beta * x + Z * x + \delta * x$

Si es exclusivista, todo por 0. Esto es debido a que los propietarios de los soportes no reciben una compensación por este trabajo, ya que son los encargados de proveer un servicio de calidad y están obligados a la entrega de las evidencias sin ninguna contraprestación económica.

Así, obtenemos unos límites inferiores y superiores parametrizables, que en el caso de ejemplo son:

X=0,5€	X=2€
Límite inferior: 0,5€	Límite inferior: 2€
Límite superior: 1,85€	Límite superior: 7,4€

Para prevenir un "exceso de éxito" y que todos los soportes sean instantáneamente auditados o lo contrario, se debe variar el parámetro "cuantía mínima básica". En principio, con esto debería bastar para variar el incentivo que se le aporta al auditor.

Con esto ya tendríamos todo lo necesario para poder asignar precios a los diferentes anuncios auditables, y, además, al estar totalmente parametrizado el dinero que se paga por cada anuncio auditado, se podrá modificar en función a las diferentes necesidades,

El siguiente punto a valorar en el estudio económico sería el método para realizar los pagos a los auditores, nos enfocaremos en buscar la plataforma de micropagos que mejor nos convenga, y por supuesto, que sea automatizable.

Existen bastante opciones en el mercado, Paypal, Stripe,MangoPay…

Veo como una buena opción usar el Paypal como método de pago, la descripción para desarrolladores de Paypal se da en el siguiente enlace:

https://developer.paypal.com/docs/payouts/#payout-models

El motivo por el que me inclino por PayPal es su simplicidad, es de suponer que prácticamente todos los usuarios que estén dispuestos a probar una nueva iniciativa tecnológica como esta tendrán cuenta de PayPal. Por lo que la facilidad del pago puede atraer a nuevos auditores. Además, PayPal aporta bastante seguridad al usuario, que no tiene que introducir sus datos bancarios, datos que la gente suele ser bastante escéptica de introducir en una app que no conocen demasiado (esto al inicio de la app, claro). Posteriormente se podría implementar el pago por transferencia bancaria común, en otra versión de la aplicación.

El funcionamiento del sistema sería el siguiente: Un anunciante paga el importe de la auditoría a través de Paypal. Una vez realizado el pago, a los usuarios se les da la opción de recoger evidencias de este anunciante. Al realizar la auditoría, el saldo que tiene cada usuario en la app aumenta. Cuando deseen pueden solicitar que se les ingrese el dinero a través de una transferencia bancaria.

En una segunda versión se podría ofrecer el pago a través de criptomonedas, que llevan asociadas unas comisiones mucho menores. El principal motivo por el cual no se ha incluido en la primera versión es que al investigar sobre el tema se decidió que sería bastante

beneficioso para la experiencia de usuario que los métodos de pago que se pudiera elegir fueran algo más "tradiciones". Sin embargo, en un futuro se podría plantear añadir opciones de pago relacionadas con criptomonedas. También se podrían usar Smart Contracts, de manera que, si un anunciante paga una auditoría, en el momento que el auditor realice la auditoría se le ingrese el dinero sin posibilidad de reversión. Además favorece la internacionalización de la aplicación.

Esta opción se deja planteada, con información más detallada de cómo llevarla a cabo en el siguiente link[15].

4.5.1 COSTE-BENEFICIO TOTAL DEL SISTEMA:

En dePoste, entienden este nuevo servicio como un servicio añadido, es decir, sobre el precio que le costaría al anunciante poner el anuncio, se le suma lo que vaya a costar auditar ese anuncio por parte de usuarios de la app.

Es por ello que los costes de ofrecer el servicio para dePoste serán escalables y estarán relacionados con las licencias necesarias para operar este tipo de negocios de economía colaborativa y con el coste de los servidores necesarios para almacenar los datos de la aplicación, la API, las evidencias, un historial desde el inicio.

- Proyecto:
  - Horas (horas de trabajo * coste/hora): 16000 €

    Se ha estimado que el desarrollo llevaría por una persona especializada 40 horas semanales durante dos meses (50€/h).
  - Equipos informáticos/servidores: 50 €/mes (600/año).

    Coste de servidores y mantenimiento.

- Primer Año de Operación:
  - Ingresos: 3000€
  - Compensación Auditores: 1000€
  - Comisiones PayPal: 150€

    5% de 3000
  - Amortización del proyecto (suponemos 5 años): 3200 €
  - TOTAL: 1250€

- Segundo Año de Operación:
  - Ingresos: 10000€
  - Compensación Auditores: 3500 €
  - Comisiones PayPal: 500 €
  - Amortización del proyecto (suponemos 5 años): 3200 €
  - TOTAL: 5400 €

Hay muchos servicios de servidores y almacenamiento, presuponiendo que la app empezará desde los cero usuarios, con las ofertas más baratas de Internet nos bastará. (Tan sólo tendrá que gestionar peticiones que se realicen a la API). Podemos estimar que unos 50€ al mes en las etapas iniciales de mantenimiento es un coste razonable. En caso de que sea necesario ampliar, se supone que ya tendremos una base de usuarios bastante grande, por lo que se ampliará sin pensarlo.

dePoste está planteando una revolución de la publicidad, dónde por primera vez el anunciante tendrá una opción que le permita que sus anuncios sean auditados por personas ajenas a la empresa que instala los anuncios. Esto aporta un gran nivel de transparencia, y un factor diferenciador respecto a la competencia.

dePoste espera conseguir con esta aplicación un aumento de la repercusión del mundo exterior en las planificaciones publicitarias. Debido a la transparencia aportada por la aplicación, quizás incrementen sus inversiones.

Finalmente, a la última parte que necesita la aplicación para funcionar serían usuarios dispuestos a auditar. Es previsible que tengamos los usuarios necesarios, ya que aplicaciones que ofrecen micro pagos (por ejemplo, repartidor de glovo) ofrecen menos dinero por un trabajo de características similares (sencillo, rápido, desplazarse a lugares para realizar tareas). En este aspecto es previsible que dePoste tenga los auditores necesarios para llevar a cabo el trabajo, ya que se puede comprobar que existe una fuerza laboral numerosa, dispuesta a realizar este tipo de servicios y a recibir una compensación acorde a la carga de trabajo

Capítulo 5. INTERFAZ Y EXPERIENCIA DE USUARIO

[11]Cada sistema operativo tiene su propia seña de identidad, que lo diferencia del resto a nivel visual. Es por ello que sabemos distinguir muy rápidamente entre Android e Ios, sin necesidad de observar el hardware que ejecuta el SO.

Conseguir una buena experiencia de usuario es complejo, pero vital para que una aplicación tenga éxito. Si se crea algo que no es atractivo visualmente, será muchísimo más complicado que sea adoptado por los usuarios, aunque el producto en sí cumpla con las funciones esperadas.

En este sentido, podemos identificar tres características principales de las interfaces de usuario que aumentarán la posibilidad de que tengamos una aplicación de éxito.

- Simplicidad

  Es importante que las aplicaciones sean sencillas a nivel visual, ya que esto aumenta su usabilidad por usuarios medios de Android. Cada pantalla y cada botón deben tener muy definida sus funciones, y limitar el número de acciones posibles al mínimo posible. Esto favorecerá que la primera vez que se descargue la aplicación el usuario sepa usarla directamente, lo cual favorecerá la adopción.

- Consistencia

  Las aplicaciones deben ser consistentes a nivel visual con el sistema operativo en uso. Respetar lo que el propio SO nos ofrece para crear las apps tales como botones, estilos de navegación… Un ejemplo de esto se da al cambiar de Windows a Mac o viceversa, aunque ambos SO son muy sencillos de usar, el

cambio crea algunas inconsistencias. En algunos casos personas menos afines a la tecnología pueden verse en serios problemas a la hora de entender el funcionamiento básico de la app. Esto puede causar que la dejen de usar, cosa que no se busca en la app.

- Navegación intuitiva

  El usuario debe saber navegar por la aplicación con los conocimientos ya adquiridos del propio uso de su terminal, es decir, se deben respetar los flujos naturales del propio SO. Va muy hilado con la consistencia y la simplicidad de la aplicación.

Finalmente, se debe hablar de la forma en la que se sostiene el móvil, ya que cambia la interfaz totalmente. Los móviles se encuentran la mayoría del tiempo en vertical, mientras que las tabletas se suelen poner más en horizontal. La orientación horizontal ayuda a aprovechar más la pantalla, mientras que la vertical favorece la usabilidad, ya que en móviles se podrán usar con una mano.

En el caso de esta aplicación, se ha optado por un diseño que cumpla los tres patrones anteriores con una orientación vertical que no cambia al poner el terminal en horizontal por dos razones; la primera es que para usar la aplicación es necesario conexión continua a internet, y esto no suele ser común en tabletas, que suelen usar más wifi que conexiones tipo 3G/4G. Además, la aplicación está pensada para ser usada en la calle, mientras se va andando buscando el soporte a auditar. Se prevé que en este caso sea muy incómodo tener que llevar el móvil el horizontal.

A la hora de obtener la imagen, mientras la cámara está abierta, se podrá capturar la imagen tanto en vertical como en horizontal.

Vamos a ver la UI que se ha implementado y a justificar porqué cumple con los patrones, por ejemplo, en la pantalla de *login*.

Imagen 4 – Pantalla de *login*

Se puede ver en la imagen que la pantalla de login cumple con lo especificado anteriormente, tiene las mínimas funciones necesarios con la mayor claridad posible. Además, es muy sencillo prever que ocurrirá al pulsar cada botón.
Ejemplo de una mala interfaz sería por ejemplo poner la pantalla de registro unida a la de *login*, esto haría que la pantalla pasase de tener 2 campos de texto con la misma funcionalidad (logearse) a muchos más con distinta funcionalidad, habría unos campos para logearse y otros para registrarse, pudiendo resultar en confusiones.

Para el diseño visual (campos de textos, botones, imagen de fondo…) se ha adaptado una plantilla que fue proporcionada por la empresa, de manera que cumpliese con

sus estándares visuales. La plantilla contenía estilos de letra, colores, imágenes, botones, campos de texto…

Imagen 5 – Localizador de soportes

Este es un caso de una pantalla menos común, pero que también cumple con los patrones. Dado que su objetivo es el de poder seleccionar soportes únicamente, no tiene sentido que nada más aparezca en ella. En este caso el usuario, por consistencia con el SO y con el objetivo de la app, sabrá que si quiere abrir el menú tendrá que pulsar las tres líneas superiores izquierdas de la pantalla, y que los soportes auditables son los puntos morados en el mapa (que posiblemente sean cambiados por algún icono de dePoste en la versión final). El usuario sabe esto sin ninguna explicación,

ya que la app cumple con los principios de experiencia de usuario UX/UI explicados anteriormente.

Inicialmente todos los soportes aparecen en color morado, cumpliendo con los estándares de la aplicación. En caso de que el soporte esté reservado, este se encuentra en amarillo.

5.1 CASO DE USO ESPERADO

En esta sección se va explicar el ejemplo de un caso de uso de un usuario que se acaba de descargar la aplicación.

En primer lugar, llegaría a la pantalla de login. Al no tener cuenta, pulsaría el botón de registro. Una vez allí introduciría los datos de registro y pulsaría en "registrarse", que lo llevaría directamente a la pantalla de login de nuevo.

Al entrar con la cuenta recién creada, accedería al mapa, dónde le saldrán los soportes que tiene opción a auditar y que se encuentran en la parte del mapa cargado debido a la gran cantidad de soportes, por temas de velocidad, no se cargan todos los soportes de la BD al abrir el mapa, si no únicamente los que pueden aparecer en la pantalla, si el usuario mueve el mapa los soportes se irán cargando según aparezcan por pantalla. (Desarrollado en la sección 6.9).

Si el usuario quiere auditar, pero se encuentra lejos del soporte, la app le pedirá que se acerque al mismo, y no le permitirá acceder a la pantalla de auditoría. En este caso el usuario tiene la opción de reservar el soporte para poder auditarlo en los próximos X minutos, de esta manera el soporte desaparecerá del mapa para el resto de usuarios, y se ahorran problemas del tipo de que dos usuarios llegan a la vez al mismo soporte y uno se queda sin auditarlo.

La pantalla descrita anteriormente es así:

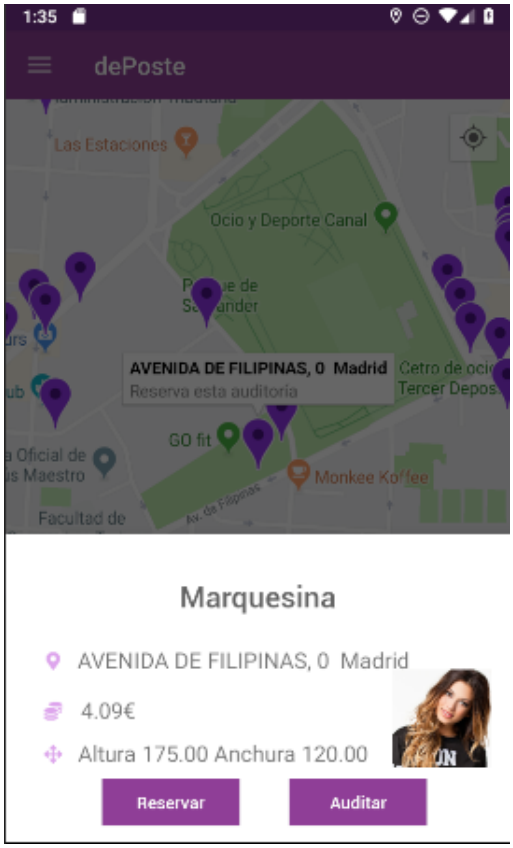


Imagen 6 – Detalles de la auditoría

En caso de que estemos cerca del soporte, tendrá la opción de auditar, accediendo al cuestionario que debe rellenar y dándole la opción (aunque es necesario para realizar la auditoría) de sacar fotos. En caso de que no se encuentre cerca del soporte, al usuario le saldrá un mensaje en la pantalla avisándole de que debe acercarse más al soporte para poder auditarlo. Este comportamiento es debido a que al pulsar el botón auditar se busca que el usuario lo audite en ese mismo momento, si el soporte se encuentra lejos, no podrá hacerlo. Si quiere reservar el soporte podrá pulsar en "Reservar" y de esa manera sólo le aparecerá en su mapa.

Tras realizar la auditoría, en caso de que esté todo correcto se enviará y el usuario volverá automáticamente a la pantalla del mapa.

Imagen 7 – Menú principal

El menú general incluye las funciones básicas de la app, aparte del localizador, que es el core de la funcionalidad, se incluye un registro de las evidencias realizadas en el pasado, una cartera dónde se especifica cuánto saldo tiene en su cuenta y una pantalla de “mi cuenta” para modificar sus datos personales.

Capítulo 6. DISEÑO DEL BACKEND

El diseño del backend es en la mayoría de aplicaciones la parte más compleja, debido a que funciones que pueden parecer aparentemente sencillas se pueden complicar de maneras que no preveíamos al comenzar a desarrollar.

6.1 COMUNICACIÓN CON LA API

La parte más compleja de este trabajo fue la comunicación con la API de dePoste, debido a que son desarrollos internos de la compañía por lo que es necesario documentarse previamente. La API proporciona acceso a la base de datos de dePoste, imprescindible para el funcionamiento de la app. Los datos sobre los que se sustenta la aplicación pueden cambiar constantemente, por lo que hay que realizar una llamada cada vez que se muestre información de la misma en la app (no es suficiente con llamar al abrir la app y luego guardar esa información para las siguientes ejecuciones. Por ejemplo, en caso de que un usuario reserve un soporte, a otro usuario que abra la aplicación después le debe desaparecer el marcador).

Los métodos de la API se fueron desarrollando iterativamente a medida que se iba desarrollando la app, ya que en un principio no estaba claro qué información de la BD sería necesaria y cuál no. Al final la versión inicial de la aplicación cuenta con catorce métodos, que serán resumidos a continuación.

register → Sirve para registrar un nuevo usuario en la app, recibe sus datos y nos devuelve OK si se ha registrado correctamente.

login_check→ Comprueba usuario y contraseña. Devuelve OK y un token imprescindible para el resto de métodos, esto es principalmente por motivos de seguridad. Sin ese token personalizado no se puede ejecutar ningún método de la api, salvo los dos anteriores.

get_user→ Devuelve los datos de un Usuario

get_audits_pending→ Devuelve las auditorías pendientes que puede realizar un usuario

create_evidence→ Crea una evidencia en la BD

check_audit_user_execute→ Comprueba parámetros del móvil del usuario previos a la auditoría(principalmente, que se encuentra cerca del soporte que va a auditar), si no se encuentra cerca no le permite empezar con la auditoría.

check_audit_user_alert→ Devuelve las auditorías que podría realizar el usuario.

Delete_evidence_blocked--> Elimina en la BD la evidencia en caso de que el usuario cancele totalmente el proceso (ej, se salga de la app).

Update_user→ Sirve para que un usuario cambie sus datos en la app si algo está mal

Get_user_wallet→ Devuelve la cantidad de dinero que tiene este usuario ganada gracias a las auditorías.

Get_evidences_user→ Devuelve las evidencias que ya ha realizado un usuario, útil para llevar un registro

Get_evidence/delete_evidence→ Necesarias para depurar durante el proceso de desarrollo.

Como se puede ver, estos métodos nos aíslan del manejo de su BD y aportan seguridad. Nos aportan la información necesaria que mostraremos de una manera fácil para el usuario en la aplicación.

Estos métodos son llamados en diferentes pantallas a lo largo de la app, para seguir los paradigmas de programación y por limpieza se ha decidido crear un archivo DAO, el cual se llamará desde cualquier punto de la app para ejecutar las llamadas a la API necesarias. Este archivo en la app se llama ApiDePosteDao.java

Un método ejemplo de esta API sería:

```
public static void UpdateUser ( String token,  OnTaskCompleteListener listener,
UserComplete usuario){
    HttpsConnectionHandler.updateUser(token,listener,usuario);
}
```

Esta API será implementada a bajo nivel en el archivo HttpsConnectionHandler, como se puede ver en la llamada estática al método. Este archivo se ha creado entonces para servir de pasarela entre la implementación a bajo nivel y la aplicación en sí, además nos aporta un lugar donde poder modificar los datos que enviamos y recibimos. Aunque en esta app la adaptación (*parseo*) de las repuestas se hace directamente en el propio código de la aplicación, se podría haber hecho en esta clase y devolver directamente objetos (ideal en caso de que el mismo método sea llamado en varias partes de la aplicación).

Este método recibe un UserComplete(usuario con todos los datos que se piden en la pantalla de registro, su *token* (String único para cada usuario) y un *listener* que nos permitirá esperar a que la comunicación con la BD acabe de una manera asíncrona).

Vemos como llama de manera estática a *HttpsConnectionHandler*, que se ocupará de la implementación del método.

Este archivo es el último punto entre la app y dePoste, es dónde se encuentra el código que lleva a cabo la comunicación con la API como tal. Es el más complicado, y ha sido rehecho varias veces por temas de optimización de las comunicaciones. Por suerte, la librería de Volley[3] es una herramienta que simplifica esta tarea.

Volley permite realizar las comunicaciones Https por internet de una manera mucho más sencilla y eficiente que nosotros mismos. Además, utiliza procedimiento estándar que limitan los desarrollos propios a realizar.

La comunicación con la API de dePoste tiene varios requisitos.

Debe ser por HTTPS

Debe ser por POST

Despúes será el servidor que recibe la petición el encargado de recuperar la información de la BD y elaborar una respuesta.

Volley ya tiene esto implementado de una manera rápida y consistente, por lo que se prefirió usar esta librería frente a implementar nosotros de cero las comunicaciones.

Volley aísla al desarrollador de la construcción del paquete, sólo es necesario conocer la información que se debe introducir y en qué campos del paquete debe ir, para luego introducirlo usando los métodos que la librería proporciona.

A continuación, se incluirá el código del método updateUser y se explicará, el resto de métodos de la Api, aunque con sus particularidades, siguen la misma filosofía de implementación.

```
public static void updateUser(final String token, final OnTaskCompleteListener listener,
final UserComplete usuario){

        if (HttpsConnectionHandler.haveNetworkConnection() == false) {
         listener.onComplete("[{\"code\":999,\"message\":\"Error: No internet
Connection\"},[]]");
        } else {

            final String key = "Sjn5vUPQOD5TbolCArT63n56oUUdBxSM";

            try {

                //USO DE VOLLEY
                //Se crea una cola, se crea un request, se pone el request en cola.
```

```
                RequestQueue queue =
Volley.newRequestQueue(ApplicationContextProvider.getContext());

                String url = "https://demo.dePoste.com/es/api/update_user";

                Log.d("Peticion", url);
                //Request a string response from the provided URL
                StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                        new Response.Listener<String>() {
                            @Override
                            public void onResponse(String response) {

                                try {


                                    listener.onComplete(response);

                                } catch (Exception ex) {

                                    Log.d("Error Volley", "Error al llamar a onComplete");
                                    ex.printStackTrace();

                                }


                            }
                        }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {


                        try {
                            //En caso de que el servidor no responda nada, salta la
excepción y devolvemos error desconocido.
                            System.out.println(new String(error.networkResponse.data));
                            listener.onComplete(new String(error.networkResponse.data,
"UTF-8"));
                        } catch (Exception e) {

                            e.printStackTrace();
                            listener.onComplete(new String("+{" + "code" + ":" + "999" +
"}"));
                        }

                    }
                }) {

                    @Override
                    public Map<String, String> getHeaders() throws AuthFailureError {
                        Map<String, String> headers = new HashMap<>();

                        headers.put("Authorization", "Bearer " + token);
                        return headers;
                    }

                    @Override
                    protected Map<String, String> getParams() throws AuthFailureError {
```

```
                        Map<String, String> params = new HashMap<String, String>();

                        params.put("key", key);
                        params.put("company_fiscalidcode",
usuario.getCompany_fiscalidcode());
                        params.put("user_name", usuario.getUser_user_name());
                        params.put("user_surname", usuario.getUser_surname());
                        params.put("user_mobile", usuario.getUser_mobile());
                        params.put("user_email", usuario.getUserMail());
                        params.put("user_birthday", usuario.getUser_birthday());
                        params.put("user_gender", usuario.getUser_gender());
                        params.put("user_password[first]", usuario.getUser_password());
                        params.put("user_password[second]",
usuario.getUser_password_repeated());


                        return params;
                    }
                };


                //Add the request to the RequestQueue
                queue.add(stringRequest);


            } catch (Exception ex) {
                ex.printStackTrace();

            }

        }
    }
```

Vamos a explicar paso a paso el código anterior, el resto de métodos son parecidos con algunas variantes, aunque el core se encuentra aquí:

Lo primero e imprescindible en las comunicaciones es comprobar que efectivamente se tiene conexión a internet, ello se hace aquí:

```
        if (HttpsConnectionHandler.haveNetworkConnection() == false) {
         listener.onComplete("[{\"code\":999,\"message\":\"Error: No internet
Connection\"},[]]");
```

La implementación de esta función no será explicada, pero lo que hace es comprobar si el teléfono tiene una conexión válida. En caso de que se cumple que no tiene conexión, se activa el listener con el mensaje "No hay conexión a internet". De esta manera en caso de que no haya conexión se le informa rápidamente al usuario, no es necesario que Volley falle, lo cual lleva más tiempo. Esto le aporta consistencia a la aplicación.

Volley funciona mediante un sistema de colas de peticiones, donde éstas se incluyen a la cola y volley las va ejecutando linealmente. A continuación se crea la cola.

```
        //USO DE VOLLEY
                //Se crea una cola, se crea un request, se pone el request en cola.

                RequestQueue queue =
Volley.newRequestQueue(ApplicationContextProvider.getContext());
```

El siguiente paso es crear la petición (*request*).. Adjuntándole todos los datos necesarios como el método (POST/GET) ,la url , y además se le especifica qué hacer en caso de respuesta correcta e incorrecta. En esta implementación:

```
String url = "https://demo.dePoste.com/es/api/update_user";

                Log.d("Peticion", url);
                //Request a string response from the provided URL
                StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                        new Response.Listener<String>() {
                            @Override
                            public void onResponse(String response) {

                                try {
                                    listener.onComplete(response);
                                } catch (Exception ex) {

                                    Log.d("Error Volley", "Error al llamar a onComplete");
                                    ex.printStackTrace();

                                }
```

Haciendo override al método onResponse, hacemos que cuando consigamos una respuesta correcta la reenviemos para la aplicación a través del método con OnComplete. El funcionamiento es similar para los errores ("códigos no 200OK"), donde también se envía el error para la aplicación (" Que puede ser por ejemplo token no encontrado").

Además, podemos escribir los *headers* y los parámetros que queramos haciendo *overrides*,

```
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    Map<String, String> headers = new HashMap<>();

    headers.put("Authorization", "Bearer " + token);
    return headers;
}
```

Aquí se incluye el *override* de las cabeceras.

Finalmente, se añade la petición a la cola:

```
//Add the request to the RequestQueue
queue.add(stringRequest);
```

Y con ello ya tendríamos la petición HTTPS en funcionamiento.

En este contexto se entiende mejor la necesidad de la aplicación Postman para el desarrollo las comunicaciones, debido a que un error aquí es difícilmente depurable debido a que pueden estar fallando muchas cosas: los procesos del controlador, las comunicaciones, o el código de la app.

Para ayudar en la depuración de la aplicación se usa Postman.

Es muy útil ya que con ella nos aseguramos de que la API funciona correctamente, en cuyo caso podemos aislar el error. Y además en caso de error ver la respuesta del servidor de una manera muy user-friendly.

Para el método update user se usaría así:

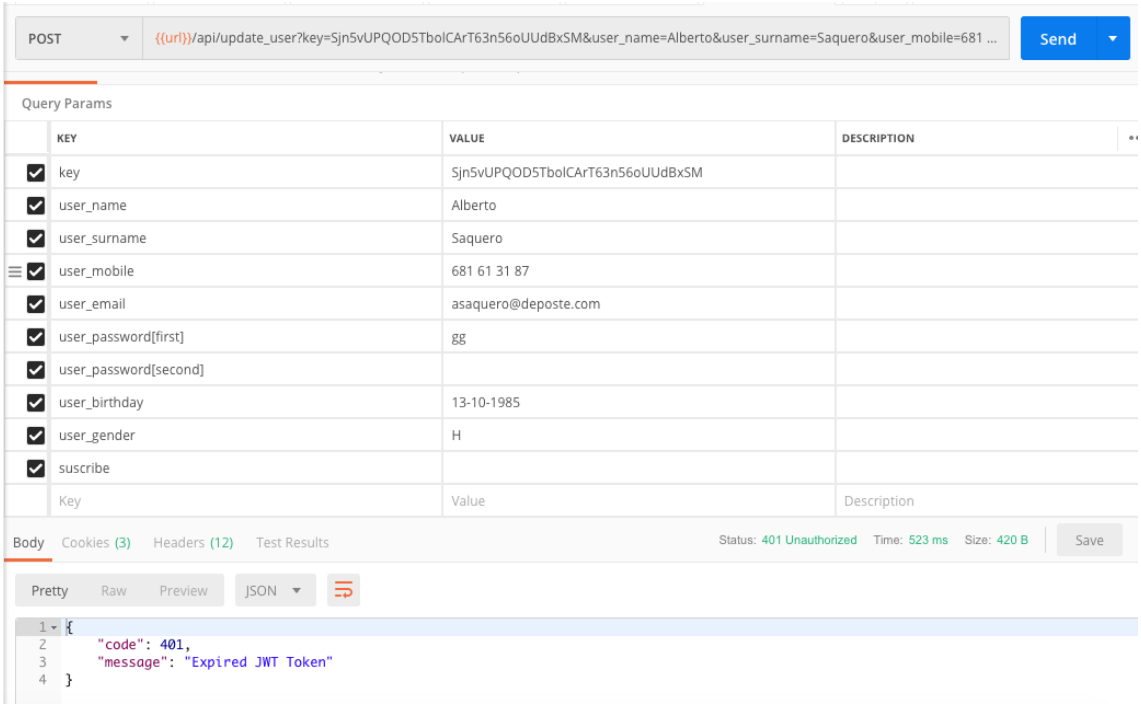


Imagen 8 – Ejemplo Postman

Además, como la aplicación funciona a través de cuenta con usuarios, esto permite a dePoste modificar métodos de la API o crearlos de 0, al desarrollador poder ver ejemplos de cómo se utilizan y qué responden los métodos (se pueden crear cuentas compartidas en Postman, de manera que varias personas pueden acceder a las llamadas que estén guardadas).

Se usó la versión de Postman gratuita, que permite hasta mil llamadas al mes, suficiente para equipos pequeños y para este proyecto.

La API responde con mensajes JSON; por ejemplo:


```
[
    {
        "code": 0,
        "message": "SUCCESS"
    },
    {
        "id": 9,
        "name": "Alberto",
        "surname": "Saquero",
        "email": "asaquero@deposte.com",
        "mobile": "681 61 31 87",
        "gender": "H",
        "birthday": {
            "date": "1985-10-13 00:00:00.000000",
            "timezone_type": 3,
            "timezone": "Europe/Madrid"
        },
        "image": "/assets/images/users/1_1523892308.jpeg"
    }
]
```


Imagen 9 – Ejemplo JSON

Esto lo recibimos como un string de caracteres. Es imprescindible para el desarrollo de la app que consigamos cada campo en una variable diferente, para ello, nos ayudaremos de las clases JSONArray y JSONObject.

6.2 LÓGICA DE LA API

En este apartado se van a explicar los métodos a los que se deben llamar para realizar una evidencia.

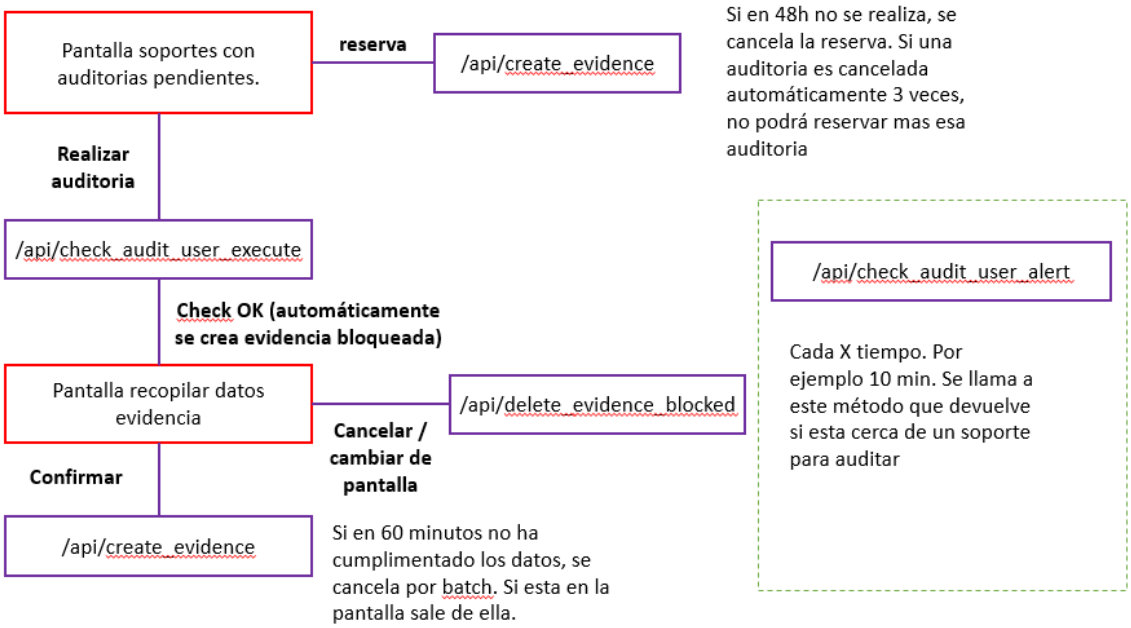


Imagen 10 – Lógica de la API

Este diagrama se inicia en "Pantalla de soportes con auditorías pendientes". Si el usuario pulsa el botón "reserva", se llama a *create_evidence*. Con ello se consigue que sólo este usuario vea el soporte en el mapa, evitando que varios lleguen a la vez, pero sólo uno pueda auditar. El tiempo de cancelación de la reserva es parametrizable desde la

BD, inicialmente, 48h. Además, hay un límite de auditorías que puede reservar, inicialmente cinco.

En cambio, si decide realizar la auditoría, se comprueba su localización. Si es correcta, se llega a la Pantalla para recopilar los datos.

Finalmente, aquí se le da la opción de confirmar (enviar la evidencia) o cancelar (delete_evidence_blocked). La cancelación no se realiza sólo si el usuario pulsa en el botón, sino que debe ser cancelada en caso de que cierre la aplicación, cambie de pantalla, se apague el móvil…

Para ello, tendremos que sobrescribir el método que convenga del ciclo de vida de la actividad…

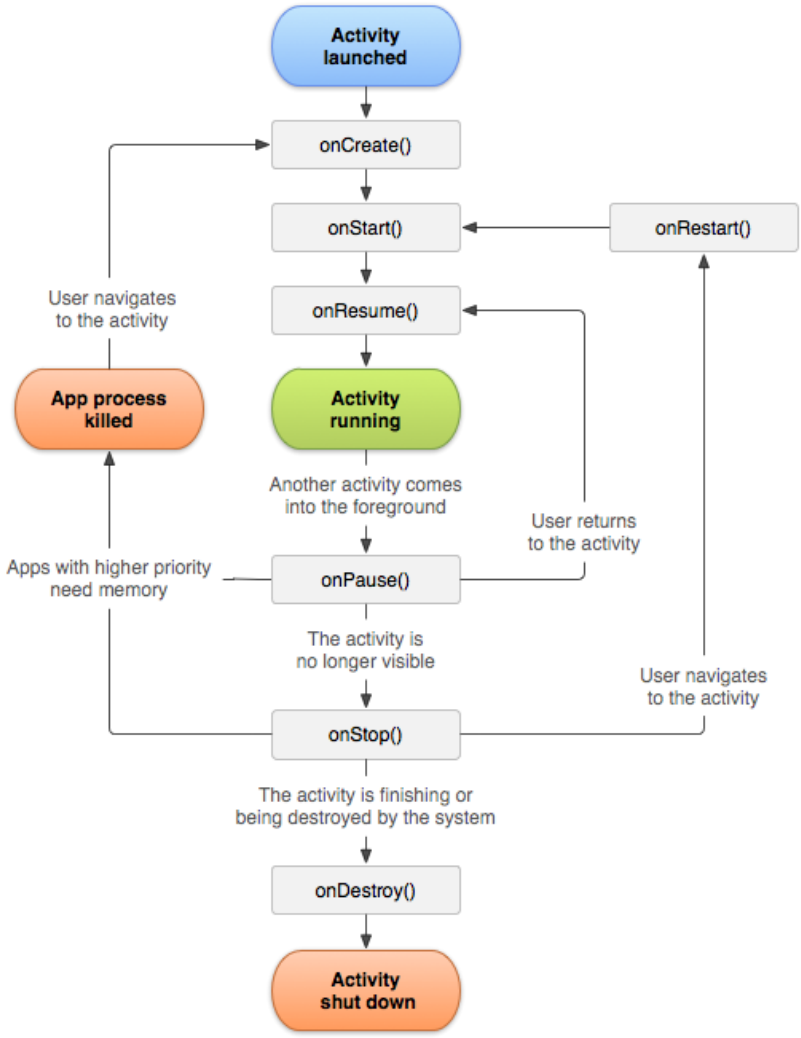


Imagen 11 – Ciclo de vida de las aplicaciones

En este caso el método onPause() también deberá llamar a delete_evidence_blocked.

6.3 DEPOSTELOCALIZADORDESOPORTES

Esta pantalla recoge la idea principal de la app, los usuarios deben ser capaces de seleccionar los soportes a auditar y obtener evidencias de ellos.

El resultado final de esta pantalla ha sido:

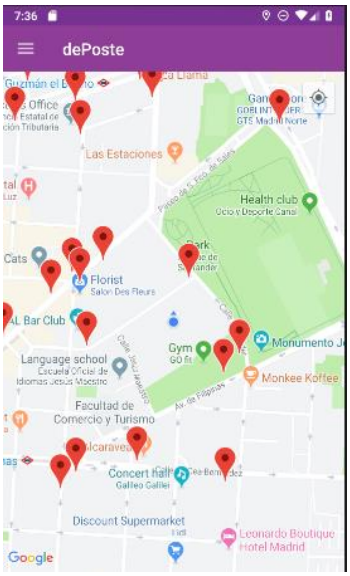


Imagen 12 - DeposteLocalizadorDeSoportes

Hay unas medidas básicas de seguridad que deben implementarse para asegurar la calidad de las evidencias, la primera es que el usuario de la app debe estar cerca (distancia parametrizable) del soporte que va a auditar. No tiene sentido que alguien intente sacar una evidencia de un soporte que esté a 10km. La segunda es que no se puedan obtener imágenes de la galería, es decir, la imagen debe ser tomada en el momento de realizar la evidencia. Con esto se protege que algún usuario obtenga una imagen del cartel y cada vez que se repita el mismo no saque fotos nuevas, si no que envíe la misma.

Por otro lado, debido a que la BD contiene demasiados soportes para poder ser enviados todos (tardaría demasiado y llenaría la memoria), sólo se deben cargar los soportes que se

puedan ver en el mapa. En consecuencia, el zoom mínimo que se podrá realizar en el mapa debe estar controlado.

Finalmente, cuando se pulse en un marcador, aparecerá información relativa a este y se dará la opción de reservar o auditar.

La forma de implementar un Mapa de Google en Android es simplemente extendiendo de la clase *SupportMapFragment* .

```
@Override
public void onMapReady(final GoogleMap googleMap) {}
```

En el interior de este método podremos usar la funcionalidad que google proporciona para sus mapas.

Por ejemplo, para seleccionar el mínimo zoom que el usuario podrá usar en este mapa:

```
googleMap.setMinZoomPreference(15);
```

A continuación, conseguiremos la ubicación del usuario para centrar el mapa sobre él.

```
MyLocation.LocationResult locationResult = new MyLocation.LocationResult() {
    @Override
    public void gotLocation(Location location) {
        //Lógica de la aplicación
};

MyLocation myLocation = new MyLocation();
myLocation.getLocation(ApplicationContextProvider.getContext(), locationResult);
```

Una vez tengamos la localización del usuario, para llamar a *getPendingAudits* (método que devuelve la ubicación de los soportes que podría auditar el usuario) necesitaremos los límites de su pantalla, que son básicamente dos coordenadas (con las que se pueden sacar las otras dos):

```
LatLngBounds bounds = googleMap.getProjection().getVisibleRegion().latLngBounds;
```

Además, en caso de que el usuario mueva el mapa, tendremos que volver llamar a *getPendingAudits* para cargar los soportes de la nueva zona del mapa que se muestra.

Para esto, implementamos un *listener*:

```
googleMap.setOnCameraIdleListener(new GoogleMap.OnCameraIdleListener() {
```

A continuación, hay que introducir un número no predeterminado de marcadores, esto se hace iterativamente con un bucle de orden lineal.

Para obtener cada marcador, lo implementamos de la siguiente manera:

```
for (int i = 0; i < audits.length(); i++) {
JSONObject pendingAudits = audits.getJSONObject(i);
structure_geom[i] = pendingAudits.getString("structure_kml");
```

Con esto se consigue la ubicación (en formato kml) de cada marcador, hay que adaptar la información del kml para conseguir un objeto *LatLng* con el que introducir un marcador en el mapa.

Esto no está bien explicado en la api de Android:

```
String kmlFile = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
        "<kml xmlns=\"http://www.opengis.net/kml/2.2\"> <Placemark>\n" +
        structure_geom[i] +
        " </Placemark> </kml>";

InputStream stream = new ByteArrayInputStream(kmlFile.getBytes(Charset.forName("UTF-8")));
```

Kml es un tipo de archivo xml.

```
KmlLayer layer = new KmlLayer(googleMap, stream, ApplicationContextProvider.getContext());
```

Es necesario conseguir un objeto tipo *layer*, ya que con él conseguiremos pasar del *String* que nos devuelve la Api a un objeto tipo *LatLng*. Como los Kml *Layer* necesitan para ser

creados un *InputStream* que contenga un archivo kml, lo que se ha hecho es crear un archivo por coordenada en el mapa (*String kmlFile*) crear con ello un *InputStream*. Finalmente creamos un layer.

```
for (KmlPlacemark placemark : layer.getPlacemarks()) {
    //Vamos a intentar averiguar qué tenemos

    Point ubicacion = (Point) placemark.getGeometry();
    Log.d("Ubicacion_check", ubicacion.toString());

    //De aquí sacamos un Latlng object y con ello podemos añadir markers

    Marker marker = googleMap.addMarker(new
MarkerOptions().position(ubicacion.getGeometryObject())
            .title(address[i])
            .snippet("Reserva esta auditoría"));
    //Con el tag le paso el objeto que representa este marker
    marker.setTag(temp);
```

A continuación, de cada *Layer* sacamos los *placemark* que contenga (en este caso 1 por layer, ya que el archivo lo hemos creado nosotros). Al objeto *placemark* usamos su método HEREDADO de la clase padre *getGeometry*(), que nos devuelve un objeto de tipo Point.

Finalmente, con ese punto creamos el marcador en la posición correspondiente.

A continuación, queremos que al pulsar un marcador nos aparezca un menú inferior con las opciones, para ello tendremos que implementar un *Listener* que esté a la escucha de cuándo un marcador es pulsado:

```
googleMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
    @Override
    public boolean onMarkerClick(Marker marker) {
        DePostePendingAudits audit = (DePostePendingAudits) marker.getTag();

        DePosteBottomSheet bottomSheet = new DePosteBottomSheet();


        Bundle bundle = new Bundle();
        bundle.putSerializable("DePostePendingAudits", audit);
        bundle.putString("token", token);
        bottomSheet.setArguments(bundle);


        bottomSheet.show(getFragmentManager(), "BottomSheet");
        return false;
```

```
    }
});
```

Y creamos una "*bottomSheet*", que contendrá la información que se requiera sobre cada marcador. Lo más destacable es que *bottomSheet* extiende de la clase "*BottomSheetDialogFragment*".

Finalmente, dentro de *bottomSheet*, al pulsar el botón de auditar se llama a *ApiDePosteDao.checkAuditUserExecute* para comprobar que el usuario se encuentra lo suficientemente cerca del soporte. En caso de que así sea, se le permite auditar.

El *bottomsheet* ha quedado implementado de la siguiente manera:

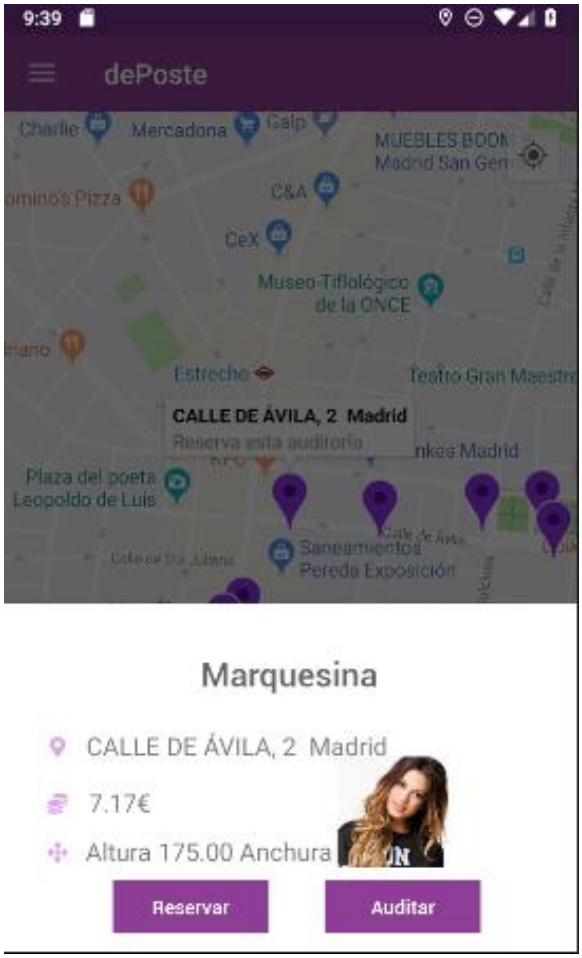


Imagen 13 – Detalles evidencias

El bottomsheet incluye la información necesaria para que el usuario encuentre el anuncio del que tiene que obtener la evidencia. El tipo de soporte (Marquesina), la ubicación (CALLE DE ÁVILA, 2, Madrid), la cantidad de dinero que recibirá el auditor y la altura y anchura del soporte. Además de una imagen (denominada la creatividad) que es el anuncio que en teoría debería estar instalado, y que el usuario tiene que buscar.

A continuación, se incluye un diagrama de flujo que representa la lógica seguida desde que el usuario llega a la pantalla del localizador de soportes hasta que realiza un soporte de manera satisfactoria.

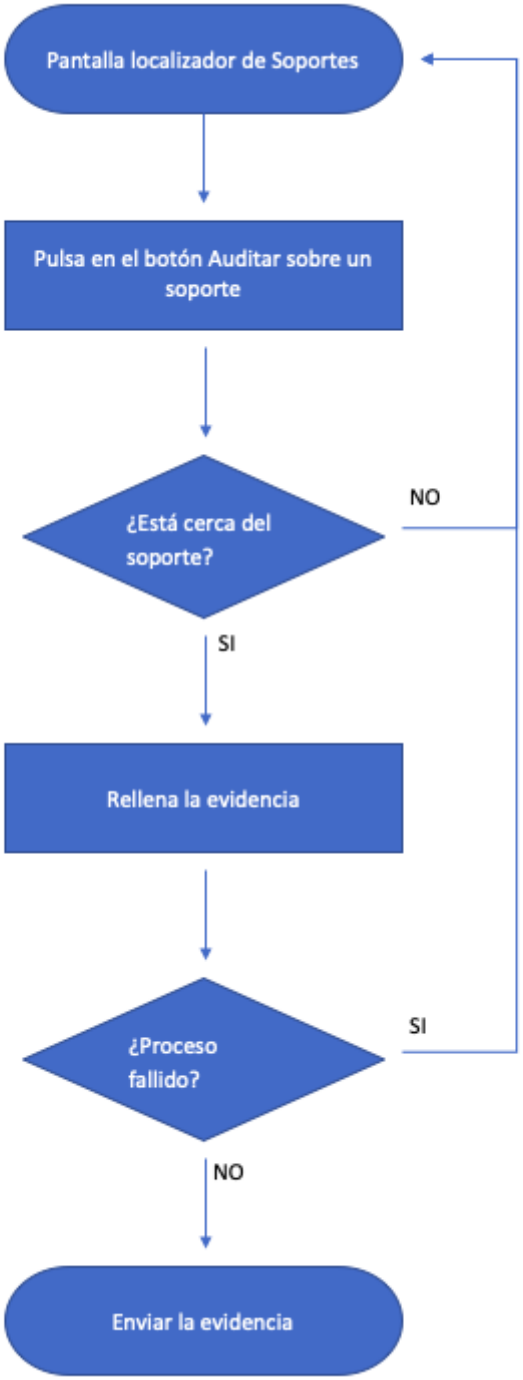


Figura 2 – Diagrama de flujo realización de auditoría

6.4 NOTIFICACIONES BACKGROUND

En la aplicación se quiere implantar un sistema de notificaciones que avise al usuario cada cierto tiempo si tiene soportes cercanos que puede auditar.

Para ello, el sistema que se va ha implementado consta de llamadas a la Api periódicas que preguntan a la BD si hay algún soporte cerca que se pueda auditar. Debe funcionar incluso si la aplicación se encuentra cerrada.

Para implementarlo se ha optado por usar una clase "servicio" de Android, que es "es un componente de una aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario"[13].

Esto es debido a que el funcionamiento en segundo plano es el comportamiento que se necesita, ya que cuando el usuario cierra la app, aún se podrá mostrar notificaciones.

La clase en cuestión es:

```
public class DePosteNotificationService extends IntentService {
```

Se ha optado por extender de IntentService debido a que "IntentService, y no es más que un tipo particular de servicio Android que se preocupará por nosotros de la creación y gestión del nuevo hilo de ejecución y de detenerse a sí mismo una vez concluida su tarea asociada"[14].

Esto simplifica bastante la implementación de cara a desarrollo, lo único que hay que hacer para crear esta clase es sobreescribir su método onHandleIntent, que es el que es llamado cuando ejecutamos la clase.

```
@Override
protected void onHandleIntent(Intent intent) {
handleActionFoo(param1, param2);
}
```

Lo que hacemos en esta función es llamar a una propia función personalizada, de esta manera, ya tenemos total libertad sobre las tareas realizables en segundo plano.

La idea es que la siguiente función se quede en bucle para siempre, recordando al usuario cuando hay un soporte cerca, y no haciendo nada cuándo no. Para ello, primero tratamos de conseguir la localización del usuario. Si esta no se consigue, no se podrá llamar a la Api para preguntarle si hay soportes cercanos.

Mientras que se está usando la app, se entiende que siempre habrá localización, ya que el usuario acepta al abrir la misma que la aplicación use los servicios de GPS. Sin embargo, este método cuenta con la particularidad de que se ejecuta en el background, incluso con la aplicación cerrada. Esto pueda llevar a situaciones en las que el usuario por ejemplo ponga el modo avión, pero los procesos de la aplicación sigan en segundo plano, en tal caso no se puede llamar a la API.

Para solucionar este problema, simplemente, antes de crear la notificación se comprueba si se puede obtener la localización. En caso de que sí, se comprueba si hay soportes cerca. Únicamente en caso de que sí los haya, se muestra un aviso al usuario.

```
ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);
executor.scheduleAtFixedRate(helloRunnable,0,20,TimeUnit.SECONDS);
```

Para ejecutarlo cada cierto tiempo se usa la clase anterior, que permite ejecutar un Objeto *Runnable* cada cierto periodo de tiempo.

Para crear una notificación, se usa la clase NotificationCompat, creando un Objeto builder…

```
NotificationCompat.Builder builder = new
NotificationCompat.Builder(ApplicationContextProvider.getContext(), "dePoste")
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("C/ del soporte")
        .setContentText("Soporte auditable!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setAutoCancel(false);
```

Finalmente, una vez está creada, para mostrarla:

```
NotificationManagerCompat notificationManager =
NotificationManagerCompat.from(ApplicationContextProvider.getContext());

// notificationId is a unique int for each notification that you must define
notificationManager.notify(i, builder.build());
i = i + 1;
```

Es importante destacar las últimas dos líneas. En el método notify, la variable i (int) es tan sólo un identificador único para la notificación. En este caso al ir variando el id, saldrían todas las notificaciones que se creasen en la pantalla apiladas; en cambio, si se usa el mismo id, las notificaciones se irán sobrescribiendo.

Cada vez que se ejecute el servicio se deberá obtener nuevamente la ubicación, ya que esta es posible que haya cambiado. La ubicación es inicialmente conseguida en

```
@Override
public int onStartCommand( Intent intent, int flags, int startId) {

    if(intent!=null) {
        token = intent.getStringExtra("token");
    }

    MyLocation.LocationResult locationResult = new MyLocation.LocationResult() {
        @Override
        public void gotLocation(Location location) {

            Log.d("Localización service", "Obtenida" +location.toString());
```

```
            setLocation(location);
        }
    };

    MyLocation myLocation = new MyLocation();
    myLocation.getLocation(ApplicationContextProvider.getContext(), locationResult);

    Log.d("Token Service:",token);
    return START_REDELIVER_INTENT;
}
```

En este método, se consigue la localización y se guarda en una variable de la propia clase (setLocation).

Entonces, deberemos asegurarnos de que cada vez que se vaya a intentar mostrar una notificación, se actualice la ubicación. Esto se consigue añadiendo

```
int ej = onStartCommand(null,0,0);
```

antes de llamar a la API.

Hay pocas aplicaciones que pueden permitirse el lujo de apilar notificaciones, las aplicaciones que suelen tener más permisividad en este tema son las redes sociales. Esto hace que pueda mostrar toda la información al usuario. Sin embargo, no se considera que dePoste deba tener este comportamiento.

El sistema de notificaciones avisa periódicamente si y solo si hay soportes cerca, pero estas notificaciones no desaparecen con el tiempo. Por tanto, no tiene sentido que se apilen notificaciones sobre soportes dónde el usuario estuvo hace tiempo y que ya se ha alejado.

Por otro lado, en caso de que esté un tiempo sin usar el teléfono, tampoco se busca que al abrir el mismo tenga muchas notificaciones indicándole lo mismo, esto es muy intrusivo y puede llevar a la desinstalación.

6.5 RECOGIDA DE EVIDENCIAS (CÁMARA)

La recogida de evidencias es la parte fundamental de esta aplicación. Para ello, el proceso de realizar una evidencia empieza cuando el usuario pulsa el botón "auditar" sobre un soporte.

El primer paso sería, en tal caso, ver una información más detallada del soporte. En esta pantalla se mostrará una imagen del anuncio que el usuario debe buscar para obtener la evidencia. Aunque sea un requerimiento que el usuario se encuentre cerca del soporte para poder auditar, es posible que en la misma zona haya varios anuncios (o incluso algunos soportes publicitarios tienen distintos anuncios en ambas caras). Para ello, con la imagen que teóricamente está anunciada, le será más fácil encontrar el soporte al usuario.

Imagen 14 – Pantalla previo auditoría

(Finalmente, esta pantalla no formó parte de la versión final de la aplicación).

Una vez que el usuario ha encontrado el soporte, puede rellenar el cuestionario que le servirá a dePoste como evidencia. Todos los campos son necesarios, y se ha optado por empezar obteniendo una imagen de un soporte. Posteriormente, mediante un sistema de estrellas, se valoran unas preguntar del 1 al 10.

Imagen 15 – Recopilación de evidencias

Las preguntas son:

¿Está la publicidad bien fijada al soporte?

Con esto se pretende medir la instalación de la publicidad en el soporte. Una mala nota vendría motivada por que la misma esté rota, despegada, mal encuadrada…

¿La publicidad que ve es la misma que la imagen?

En estos casos pocas estrellas vendrían motivadas o por error en la publicidad (hay otro anuncio instalado) o por error de impresión (tonos de colores no acordes con la imagen dada).

¿Tiene buena visibilidad?

Juzga la visibilidad que tiene respecto a la que se esperaría de un soporte de tales características. Pej, no se espera que una marquesina esté tapado por un árbol, o que una valla publicitaria no se vea desde la carretera…

¿Tiene otros soportes cerca?

Juzga la cantidad de anuncios en una zona

En caso de que haya algún problema, se podrá introducir información adicional rellenando un campo de texto en la aplicación. Aquí el usuario podrá escribir lo que vea conveniente.

Finalmente, cuando esté todo relleno se puede enviar la evidencia, que quedará almacenada en los servidores de dePoste. Posteriormente se aumentará el saldo del usuario en su cuenta.

El campo más destacable de esta pantalla es el de obtener imagen, en una primera instancia de esta pantalla, sólo aparece un hueco para imágenes. En caso de que el usuario quiera introducir más imágenes, tendremos que ir ofreciendo más vistas previas según va creando las imágenes. Para ello, usaremos los llamados “layouts programáticos”. Cuando el usuario pulse en obtener imagen, crearemos otro “ImageView” dinámicamente, de tal manera que se puedan mostrar todas las imágenes que cree el usuario simultáneamente.

Imagen 16 – Obtención de imágenes

El resultado es que las imágenes se van apilando en la pantalla, en principio ilimitadas, aunque es fácilmente parametrizable con un contador.

```
try {
    ImageView imageView=null;
    if(i==0) {
         imageView = findViewById(R.id.imagen_obtenida);
         //Cuenta el número de imágenes
         i=i+1;
    }else if(i>0 &&i<6) {

        LinearLayout linearLayout = findViewById(R.id.dynamic_layout);

        imageView = new ImageView(this);

        LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT);

        layoutParams.setMargins(0, 20, 0, 0);

        linearLayout.addView(imageView,layoutParams);
        i=i+1;
```

```
    }else{
        Toast.makeText(ApplicationContextProvider.getContext(), "Ha alcanzado el máximo
número de imágenes", Toast.LENGTH_LONG).show();

    }
```

Se implementa el contador "i" por dos motivos, el primero es que inicialmente (cuando i es igual a 0) ya tenemos un ImageView estático que tendremos que modificar.

Para los siguientes (hasta 5 como máximo) se van creando "huecos" para las imágenes dinámicamente, cada vez que el usuario obtiene de manera satisfactoria una imagen (no mostrado en el código anterior) se crea un Layout con las mismas características que el estático, que es colocado justo debajo del anterior y centrado con respecto al centro.

Finalmente, para evitar la sobrecarga de imágenes, se establece un límite parametrizable, a partir del cual al obtener una imagen nos saldrá un aviso diciendo que se ha llegado al límite.

6.6 MIS EVIDENCIAS

El objetivo de esta pantalla es que el usuario pueda ver las evidencias que ha realizado en el pasado recopiladas. Le aporta información al usuario. También podrá ver las auditorías que tiene reservadas para realizarlas más tarde.

Imagen 17 – Mis evidencias

Se ha optado con un diseño visual a modo de "cartas" dónde cada una es una evidencia diferente, se le aporta la dirección y la imagen como información básica. Luego, pulsando en el botón "ver" podrá acceder a más detalles sobre esa evidencia en concreto.

En caso de que la evidencia esté reservada, se toma una imagen proporcionada por dePoste por defecto. De esta manera en un futuro se podrá incluir la creatividad de la imagen

reservada. En caso de que la evidencia esté realizada, se mostrará la primera imagen que subió el usuario.

Además, para que el usuario pueda ver si valoración pasada de esa evidencia se incluye el botón detalles, el cual lleva a la siguiente pantalla…

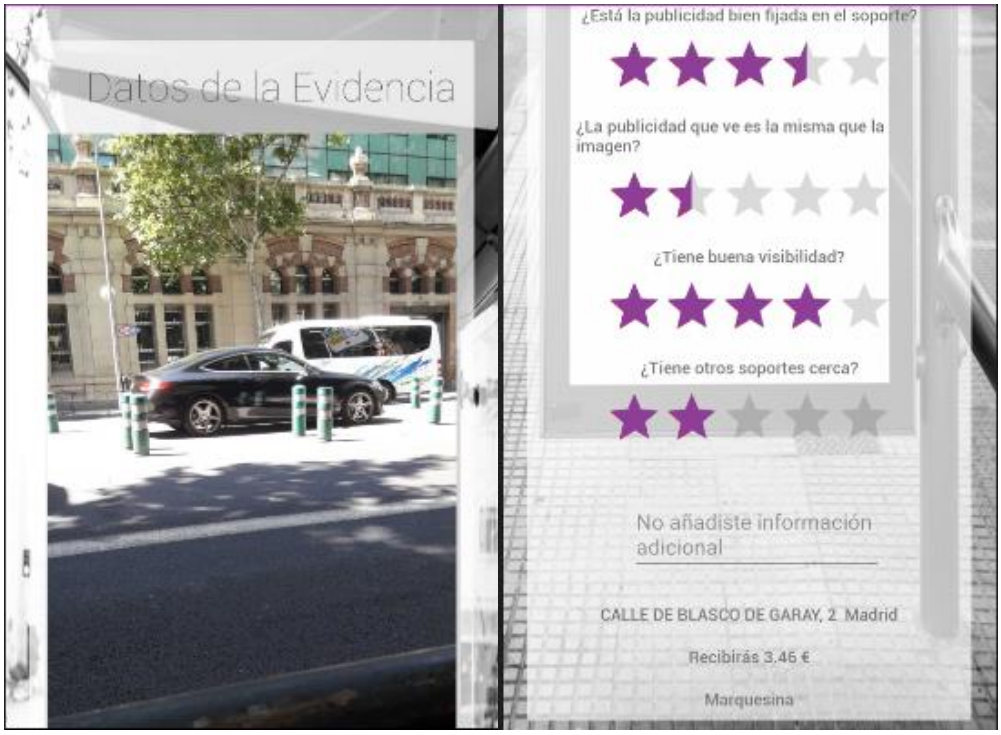


Imagen 18 – Cuestionario evidencias

Primero se enseñan la/s imágenes que subió el usuario y después la puntuación que puso en el cuestionario y los datos de esa auditoría.

6.7 MI CARTERA

Esta pantalla es muy simple, se ha optado en una primera versión por mostrar el saldo pendiente, y un botón que avisará a dePoste para que realice el pago al usuario.

Imagen 19 – Mi cartera

6.8 MI CUENTA

En esta pantalla se pretende que el usuario pueda ver los datos que introdujo en el momento de registrarse, y en caso de que alguno sea incorrecto pueda cambiarlo de manera dinámica.

Simplemente pulsa en el campo que decida, lo cambia y pulsa el botón de actualizar.

Imagen 20 – Mi cuenta

6.9 OPTIMIZACIONES DE RENDIMIENTO

Para el correcto funcionamiento de la aplicación fue necesario llevar a cabo una serie de optimizaciones en el mapa. Lo ideal hubiera sido que al abrir el localizador de soportes se cargase el mapa de España entera, con todos los soportes que hay en ella, y el usuario pudiera navegar por donde quisiese.

En un principio se intentó optar por esta solución, aunque acabó descartada por los siguientes motivos:

1. Demasiada información: Al pedirle todos los soportes, los búffers de la BD se llenan y el sistema cae. Por tanto no es viable que cada vez que se entre en el mapa se pida una petición que puede tardar más de un minuto, o hacer caer el sistema.
2. Capacidad del teléfono: Una vez que llega la información al teléfono, el procedimiento para introducir los soportes en el mapa es lineal. Se coge un punto y se pone su soporte. Para pocos soportes (menos de 500) el tiempo es inapreciable, para más empieza a ser un tiempo no ignorable. Además, en teléfonos con poco procesador disponible esto puede ser un problema incluso antes.
3. Desperdicio de recursos: Finalmente, no parece lógico cargar todos los recursos si el usuario no los va a usar.

La solución que se llevó a cabo pasa por averiguar los límites de la pantalla de mapa que ve el usuario, enviar esas coordenadas y cargar sólo lo que se encuentre en esa pantalla. Esta solución exige además que exista un límite de *unzoom* en el mapa.

A medida que el usuario explora y se mueve en el mapa, se van cargando nuevos soportes.

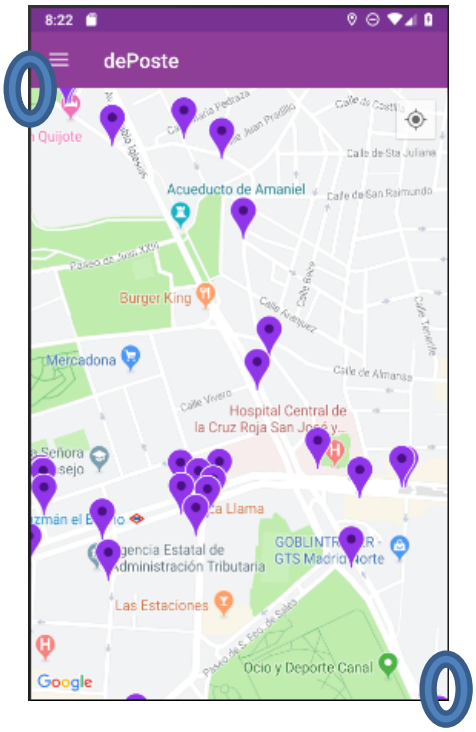


Imagen 21 – Localizador de soportes… optimización del rendimiento

Para llevar a cabo lo anterior, cada vez que el mapa se queda quieto, se realiza una nueva petición a la base de datos enviando dos puntos (latitud y longitud) con los que la BD nos responde con los soportes que se encuentran dentro del rectángulo. De esta manera se consigue una mayor escalabilidad de la aplicación y una mejor experiencia de usuario.

Para ver esta pantalla se han cargado unos 20 soportes, en vez de los miles y miles que contiene la BD completa.

Capítulo 7. ANÁLISIS DE RESULTADOS

El final de este proyecto de fin de grado es una aplicación para la empresa dePoste que se puede considerar una aplicación que cumple con la funcionalidad esperada en los resultados finales.

Tal como está planteada, es susceptible de ser subida a Google Play, además, cumple con los requisitos iniciales planteados.

La aplicación es *user-friendly*, sencilla de entender y el flujo a través de la misma es bastante natural. Está optimizada para utilizar la cantidad de recursos necesaria. Cumple con los objetivos definidos en la sección 6.2

Estudio económico de costes y beneficios
Información del auditor
Evidencias
Realización de auditoría
Aplicación

El estudio económico es adjuntado en este documento.

La información del auditor se consigue cuando se registra un usuario, y la puede cambiar cuando desee mediante la pantalla “mi cuenta”.

La realización de la auditoría se hace manera sencilla e intuitiva a través del Smartphone, quedando almacenadas en las BD de dePoste, con lo que luego los anunciantes podrán acceder a ellas.

En las evidencias se almacena toda la información relevante sobre la instalación del soporte, se rellana un cuestionario y se envía un mínimo de una imagen (y un máximo de cinco).

La aplicación cuenta con las categorías que fueron especificados en los objetivos, con las funciones esperables a esas pantallas.

Resumiendo, se puede afirmar que, como se previó en el análisis del estado del arte de las aplicaciones de economía colaborativa y en la justificación de la motivación, una aplicación en el sistema operativo Android permite que el usuario le aporte valor al anunciante en el contexto de la auditoría de anuncios.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

La aplicación cumple su función y es una más dentro de lo que se considera como "economía colaborativa". Es previsible que tenga un buen acogimiento debido al margen de beneficios que le aporta al usuario. Además, aporta valor para los anunciantes a un precio bajo.

Sin embargo, no se puede decir que esta aplicación sea su "versión final" aunque probablemente si esté preparada para ser considerada una versión inicial. Debido al constante cambio e innovación que requieren estas ideas, es posible que una vez que se lleven a cabo pruebas con usuarios reales, la app pueda cambiar.

Trabajos futuros planteados son: La introducción de SmartContracts en la aplicación y el uso de criptomonedas, esto la haría más segura a la par que favorecería su internacionalización.

Mejoras en los algoritmos para conseguir mejor funcionalidad en diversas partes de la aplicación. Por ejemplo, en el localizador de soportes, actualmente cuando un usuario pulsa sobre un soporte, sólo se le indica dónde se encuentra este, no una ruta dibujada en el mapa hacia él. Esto podría ser una mejora que favorecería la experiencia de usuario.

Posibilidad de borrar imágenes en auditar activity, actualmente se pueden incluir un número parametrizable de imágenes, pero no se pueden borrar.

Multiplataforma: Desarrollarlo tanto para Android como para IOS.

Capítulo 9. BIBLIOGRAFÍA

[1] Download Android Studio and SDK tools. (2019). Retrieved from https://developer.android.com/studio

[2] Postman | API Development Environment. (2019). Retrieved from https://www.getpostman.com/

[3] Volley overview | Android Developers. (2019). Retrieved from https://developer.android.com/training/volley/

[4] Adigital: "El 'carsharing' no es economía colaborativa." Retrieved from https://www.elmundo.es/economia/2017/03/08/58bfac73468aeb48508b45ce.html

[5] Economía colaborativa - Definición, qué es y concepto | Economipedia. (2019). Retrieved from https://economipedia.com/definiciones/economia-colaborativa.html

[6] Market share for mobile, browsers, operating systems and search engines | NetMarketShare. (2019). Retrieved from https://www.netmarketshare.com/

[7] Corcobado, M. (2019). Más allá de Airbnb o Wallapop: las apps de economía colaborativa que no conoces. Retrieved from https://elpais.com/tecnologia/2017/06/21/actualidad/1498053063_148486.html

[8] El secreto del éxito de la economía colaborativa - Diario Responsable. (2019). Retrieved from https://diarioresponsable.com/noticias/24814-el-secreto-del-exito-de-la-economia-colaborativa

[9] colaborativa, M. (2019). Más de la mitad de los españoles ya usa apps de economía colaborativa | Reason Why. Retrieved from https://www.reasonwhy.es/actualidad/digital/mas-de-la-mitad-de-los-espanoles-ya-usa-apps-de-economia-colaborativa-2018-02-07

[10] Diseño interfaz. (2019). Retrieved from https://ihcbyclaualvarado.blogspot.com/2017/05/diseno-interfaz.html

[11] Material Design UI Android Template App. (2019). Retrieved from https://codecanyon.net/item/material-design-ui-android-template-app/9858746?s_rank=8

[12] Servicios | Android Developers. (2019). Retrieved from https://developer.android.com/guide/components/services?hl=es-419

[13] Tareas en segundo plano en Android (II): IntentService | sgoliver.net. (2019). Retrieved from http://www.sgoliver.net/blog/tareas-en-segundo-plano-en-android-ii-intentservice/

[14] Blockchain Wallet API: Bitcoin Wallet API - Blockchain. (2019). Retrieved from https://www.blockchain.com/es/api/blockchain_wallet_api

[15] TomTom Support. (2019). Retrieved from http://es.support.tomtom.com/app/answers/detail/a_id/3254/~/acerca-de-gps

[16] Base de datos espacial. (2019). Retrieved from https://es.wikipedia.org/wiki/Base_de_datos_espacial

[17] JSON. (2019). Retrieved from https://www.json.org/

[18] BOE.es - Documento DOUE-L-2016-80807. (2019). Retrieved from https://www.boe.es/buscar/doc.php?id=DOUE-L-2016-80807

[19] Tutorial de KML | Keyhole Markup Language | Google Developers. (2019). Retrieved from https://developers.google.com/kml/documentation/kml_tut?hl=es-419

ANEXO A

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

I.C.A.I.

PROYECTOS FIN DE GRADO

Titulación y optatividad: Grado en Ingeniería de Telecomunicaciones

Alumno 1º Apellido: Enrile

2º Apellido: Fernández de Arévalo

Nombre: Miguel

Teléfono de contacto: 649383168

e-mail: miguel.enrile1@gmail.com

Título del Proyecto Fin de Grado: DISEÑO DE UNA RED COLABORATIVA DE AUDITORÍA DE CAMPAÑAS DE PUBLICIDAD EXTERIOR

Director (nombre y dos apellidos): Alberto Saquero Rodríguez
Teléfono de contacto: 681 61 31 87

e-mail: asaquero@deposte.com

Breve descripción del proyecto (5 o 6 líneas)

El objetivo del Proyecto es diseñar una red colaborativa de auditoría de publicidad exterior. Para ello será necesario desarrollar una aplicación móvil en Android para que los usuarios/auditores puedan tomar fotografías de emplazamientos de publicidad exterior como evidencias del trabajo ejecutado. Estas evidencias incluirán información de la geoposición donde fueron tomadas, la orientación de la cámara, la fecha y hora y cualquier otro campo necesario para la trazabilidad. Además, se almacenarán en una base de datos propia de la compañía y se escribirán en la Blockchain de Ethereum como doble verificación de la prueba. El auditor recibirá una compensación económica por cada evidencia recogida. Por ello, es necesario incluir un estudio económico de costes y beneficios de esta red.

El documento final del proyecto será subido al Repositorio Institucional de Comillas con acceso público. El alumno podrá solicitar un nivel restringido de acceso (incluido el "cerrado" o "confidencial") que podrá concederse, excepcionalmente, si está plenamente justificado.

Se solicita que sea confidencial por motivos de la empresa

The final report of the Project will be uploaded to the Comillas Institutional Repository with public access. The student will be able to ask for a restricted access (even "closed" or "confidential") which will be exceptionally accepted if it is fully justified.

Aceptación del Director.

En Madrid, a 17 de septiembre de 2018.

Alberto Saquero Rodríguez