Aplicación Android con integración *blockchain* en respuesta al acoso en espacios públicos

Autora:
Patricia Barrios Palacios

Director:
Samer Hassan Collado
Codirectora:
Silvia Díaz Molina
Colaborador:
David Llop Vila

Trabajo de fin de grado
Doble Grado en Ingeniería Informática - Matemáticas
Universidad Complutense de Madrid

Departamento de Ingeniería del Software e Inteligencia Artificial (ISIA)
Facultad de Informática

Curso 2018-2019

UNIVERSIDAD COMPLUTENSE DE MADRID

Resumen

A día de hoy, la sociedad moderna está empezando a tomar conciencia de los problemas relacionados con la violencia de género, apareciendo múltiples movimientos feministas frente a dicho problema. Asimismo, viendo cómo los teléfonos móviles se han convertido en una herramienta indispensable en nuestro día a día, sería práctico aplicar los avances tecnológicos de los que disponemos como respuesta a estas necesidades. De aquí surgió la idea de este proyecto que pretende dar una herramienta a la **ciudadanía** para poder ayudarse entre sí sin necesidad de intermediarios.

Dicho esto, el proyecto consiste en el desarrollo de una aplicación **Android** en respuesta a los **acosos** producidos en los espacios públicos que permite enviar alertas de emergencia entre dispositivos así como el almacenamiento y consulta de las mismas y la publicación de quejas ante anuncios cuyo contenido sea misógino. Particularmente los datos generados y compartidos quedarán almacenados en ***Ethereum***, basada en el modelo ***blockchain***, con el fin de mantener el pseudoanonimato de los usuarios, tener una versión consistente de los datos en todo momento y asegurarlos mediante el uso de **criptografía**.

Cabe destacar que en paralelo a la implementación, se ha llevado una evaluación continua tanto a nivel funcional como interactivo mediante la realización de entrevistas, gracias al tiempo dedicado por voluntarios.

Palabras clave: *Blockchain*, Android, Colaboración ciudadana, Ethereum, Acoso, Criptografía.

UNIVERSIDAD COMPLUTENSE DE MADRID

Abstract

Currently, modern society is starting to become aware of issues related to gender-based violence and many feminist movements have appeared. Likewise, cell phones have become an essential tool for us, so it would be useful to apply available technological improvements to social demands. The idea of this project emerged from here, and is to provide **citizens** a tool to enable them to assist each other without intermediaries.

That said, the goal of this project is to develop an **Android** application, in response to **harassment** in public spaces , which allows users to send emergency alerts as well as store and search them and post complaints about advertisements with misogynous content. In particular, every piece of information generated or shared by users will be stored into ***Ethereum***, founded on ***blockchain***, in order to ensure pseudo-anonymity, have an updated consistent version of the data and guarantee its security, using **cryptography**.

Along with the development, continuous evaluation has been made functionally as well as interactively through interviews, thanks to volunteers.

Keywords: *Blockchain*, Android, Cooperation of citizens, Ethereum, Harassment, Cryptography.

Agradecimientos

Tomándome la licencia de agradecer no solo por el desarrollo de este proyecto sino por el recorrido en la universidad en su conjunto.

En primer lugar, dar las gracias a Silvia Díaz, antropóloga feminista y a Samer Hassan, doctor en Informática y profesor de la facultad. Directora y director del proyecto que nos han motivado y ayudado desde el primer momento en la toma de decisiones y desarrollo y al mismo tiempo nos han dado una enorme libertad para poder enfocarlo a nuestro gusto. Destacando su labor por inculcarnos que el desarrollo de cualquier tecnología debe estar al servicio de la sociedad y no al revés.

Agradecer también a todo el equipo de *P2P Models* que no han dudado en ofrecer su ayuda a lo largo del desarrollo. Especialmente a Elena Martínez, investigadora y experta en diseño UX/UI por el *feedback* sobre la interfaz y a David Llop, Antonio Tenorio y Jordi Burguet, investigadores en la facultad en tecnologías descentralizadas, por su apoyo con la parte más técnica de *Blockchain*.

Y por supuesto, a todos aquellos que han estado siempre ahí a nivel personal. Gracias a mis padres, quienes han vivido más de cerca los altibajos y han mostrado un apoyo incondicional; a Lucía, Pablo, Javier y Diana, capaces de sacarle la parte positiva a cualquier situación, la haya o no; y a compañeros de la universidad porque hemos compartido muchas experiencias, quejas y alegrías juntos que nos han unido; con una mención especial a David, Elena, Beatriz e Iván.

Habéis sido indispensables en esta etapa y no hay páginas suficientes para agradeceros vuestro tiempo y esfuerzo invertidos.

Índice general

Índice de figuras

Capítulo 1

Introducción

> Technology is the campfire around which we tell our stories.
>
> *Laurie Anderson*

1.1. Motivación

Previo al desarrollo del proyecto se presenta su contexto analizando la situación de España frente al acoso callejero y la violencia de género.

Comenzando con un estudio de la situación a nivel social, las redes sociales han tenido una enorme influencia en la visibilidad y actuación en torno a este problema, sobre todo tras el escándalo de Hollywood en 2017, donde cientos de actrices denunciaban los abusos de poder de la industria y animaban al resto de mujeres a romper el silencio mediante la iniciativa *#MeToo*. Fue especialmente popular en Twitter, donde alcanzó cerca de 200,000 denuncias sociales y vino seguida de otras como *#womenwhoroar* también a nivel internacional, *#balancetonporc* en Francia o más recientemente aquí en España con *#yosítecreo*, *#cuéntalo* y *#somosuna* que aun apareciendo en distintos contextos tienen un objetivo común que es mostrar apoyo y unidad.

Acompañando a los movimientos de las redes sociales han surgido diversas iniciativas como los puntos violeta. Estos están gestionados por activistas que velan por la creación de espacios seguros y se organizan en torno a eventos de mucha afluencia, donde las aglomeraciones son un pretexto para que se produzcan todo tipo de agresiones. El año pasado en Madrid se instalaron 60 puestos, frente a los 15 de 2017 y se prevé que la cifra aumente en 2019.

Teniendo en cuenta la magnitud del problema y considerando su relevancia, procedemos a investigar qué medidas tanto a nivel nacional como europeo se están tomando para intentar reducir estas cifras:

- Por una parte, *Unsafe in the city* desplegó **Free to be**, una plataforma web de mapeo en cinco ciudades, entre las cuales se encuentra Madrid, donde las usuarias pueden contar sus testimonios para identificar y compartir espacios públicos que las hacen sentir incómodas o por el contrario, seguras.

- Continuando en la búsqueda, el ayuntamiento de Madrid ha abierto la plataforma Decide Madrid que da voz a propuestas de todo colectivo, llevándose a cabo si consiguen el suficiente apoyo y pasan la votación. Entre ellas, la creación de espacios de igualdad o el desarrollo de un programa de perspectiva de género y educación sexual para jóvenes.

  Deteniéndonos a pensar en estos dos proyectos encontramos una oportunidad de acción, ya que la idea de emplear la colaboración ciudadana para mantenernos informados resultó muy atractiva y podría hacerse más accesible ampliándose al resto de ciudades.

- A nivel internacional, la ONU ya ha puesto en marcha varias iniciativas bajo el nombre de *UN Women's Global flagship iniciative*, asentadas en cuatro fases :

  · Generación de evidencias para concienciar a la población del problema.

  · Desarrollo y aplicación de leyes y políticas en favor de la causa.

  · Inversión en seguridad en espacios públicos.

  · Tranformación de las normas sociales, educando y promoviendo la igualdad de género.

  Actualmente y hasta 2021, se planea tener especial impacto en el empoderamiento político y económico, mediante campañas como *Empower Women* [11] o *iKNOW Politics* [24], que se apoyan en la creación de talleres y espacios *online* para ayudar a impulsar las carreras profesionales de mujeres en política.

- Actuando simultáneamente, la Comisión Europea presentó en 2016 un compromiso estratégico [14] en favor de la igualdad de género que aún está vigente y pretende reducir y eliminar la brecha de género.

Finalmente, se concluye este informe introductorio con el apoyo de los datos proporcionados por el INE [23], haciendo un breve análisis de la evolución de las denuncias presentadas por violencia de género desde 2013. Aunque las estadísticas no tienen una categoría concreta con este nombre, se ha tomado el conjunto que engloba todo acto de violencia física o psicológica incluyendo delitos contra la libertad e indemnidad sexual, amenazas o coacciones.

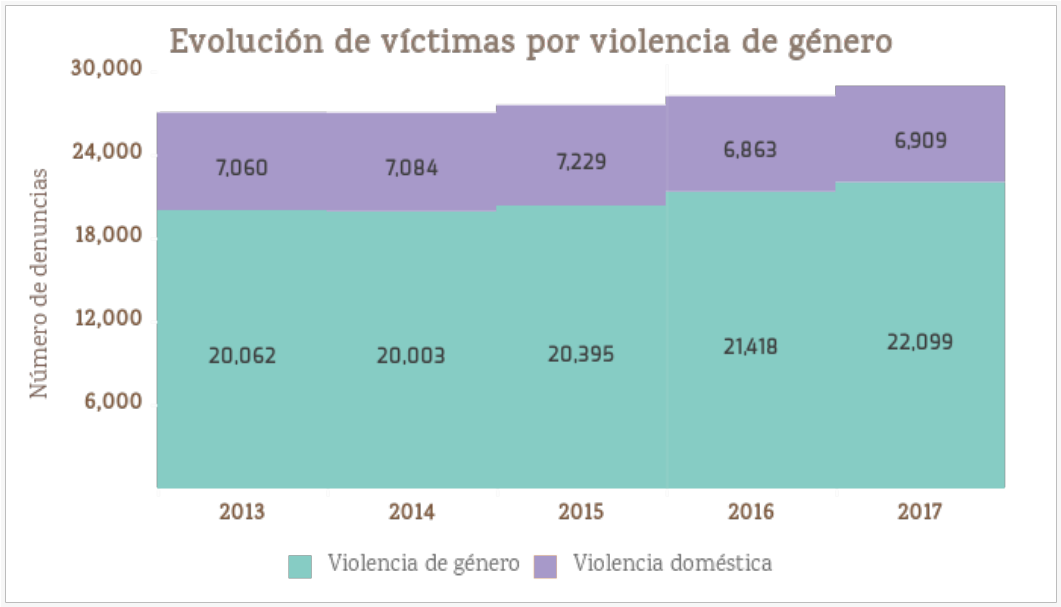


FIGURA 1.1: Gráfico de denuncias por violencia de género

Sin embargo, pese a que los resultados extraídos del gráfico 1.1 ya son suficientemente alarmantes, posicionándose en torno a 70 denuncias diarias con una tasa de variación del 7 %, cabe destacar que solo se ven recogidos los casos denunciados y además, éstos no muestran indicadores para medir las distintas formas de violencia, a excepción de la violencia doméstica que ocupa alrededor del 25 %. Por lo que puede suponerse que el problema es aún mayor de lo que reflejan las cifras.

1.2. Objetivos

A raíz de lo expuesto anteriormente, la finalidad de nuestro proyecto es posibilitar el empoderamiento a la ciudadanía en respuesta al acoso en espacios públicos. Para conseguirlo hemos decidido desarrollar una aplicación Android que se basa en la colaboración ciudadana sobre la que se han fijado los siguientes objetivos:

- Desarrollar una aplicación nativa Android enfocada a la finalidad descrita.

  · Diseñar una interfaz intuitiva y clara en función del feedback recibido, satisfaciendo sus necesidades.

  · Permitir lanzar avisos de emergencia a otros usuarios que estén cerca geográficamente y personas de confianza a través de grupos de *Telegram*.

  · Permitir denuncias colectivas de publicidad misógina.

  · Tener un registro conjunto de los avisos y carteles en cuestión y mostrárselo al usuario.

- Comprender las tecnologías *blockchain* y usar *Ethereum* en sustitución a una base de datos para almacenar de forma descentralizada la información generada por la aplicación. A su vez, abstraer esta integración para que la persona usuaria no tenga que tener conocimientos previos en la materia para utilizarla.

1.3. Estructura y planificación

Este proyecto es desarrollado por dos personas a lo largo del curso 2018/2019, donde se han diferenciado dos bloques de trabajo, acorde a la duración de los cuatrimestres académicos.

- Durante el primer bloque se abarca la fase de investigación y definición de objetivos, siempre en función de las necesidades identificadas en las entrevistas.
- Una vez fijados los objetivos y la interfaz, se dedica la mayor parte del tiempo a la implementación del proyecto y la evaluación del mismo. Aunque el diseño siempre ha estado sujeto a cambios por el trabajo en paralelo de evaluación y desarrollo.

El reparto de tareas ha sido equitativo entre las dos integrantes. Por una parte, durante el primer bloque se ha llevado a cabo un trabajo conjunto de investigación y modelado, poniendo en común los resultados obtenidos de las entrevistas y alcanzando un diseño final, como se verá en el capítulo 3. Por otra parte, el desarrollo de la aplicación tiene dos partes claramente diferenciables:

- El muestreo de las alertas y carteles mediante un mapa y un histórico, sincronizados entre sí, así como el almacenamiento manual de los mismos en la *blockchain*.

  Este documento detalla los pasos llevados a cabo para completar este primer apartado, incluyendo un apartado adicional sobre el envío de alertas a través de *Telegram*.

- La gestión de las notificaciones y datos enviados entre dispositivos, incluyendo el almacenamiento de las alertas automatizado en la *blockchain* y los datos generados localmente.

  Para conocer los detalles de este apartado, se sugiere consultar la memoria de Elena Pérez Tirador.

Estructura de la memoria

Esta memoria contiene la información necesaria para la comprensión del desarrollo del proyecto. Está organizada por capítulos cuyo contenido es el siguiente:

- **Capítulo 1 - Introducción.** Es el capítulo actual y pretende ofrecer una primera contextualización para mostrar la motivación del mismo.
- **Capítulo 2 - Estado del arte.** Recoge la comprensión y explicación en profundidad de cómo funcionan y cuál es la utilidad de las tecnologías descentralizadas dentro del marco del proyecto.
- **Capítulo 3 - Diseño de la aplicación.** Engloba las fases del diseño guiado por objetivos. Realizando una primera fase de investigación sobre las últimas tecnologías o productos relacionados con nuestro proyecto y una comparación de las posibles ramas de acción que teníamos fijadas como objetivo. Se definen entonces los elementos funcionales y el flujo de la navegación y se itera sobre el diseño generado hasta alcanzar una interfaz intuitiva.
- **Capítulos 4 - Implementación de la aplicación.** Comienza con una introducción y breve descripción de las herramientas usadas desde una perspectiva aplicada, para después profundizar en los detalles de elección e implementación de los elementos funcionales de Android, así como la integración de Ethereum.
- **Capítulo 5 - Resultados y trabajo futuro.** Tras completar la implementación de la aplicación, se presentan una serie de conclusiones extraídas de las evaluaciones con los usuarios y cuáles serían las posibles ramas de evolución a futuro del proyecto.
- ***Apéndices.*** En esta sección queda reflejado el trabajo auxiliar realizado que aunque es prescindible para la comprensión del desarrollo del proyecto, ha formado parte del mismo.

Capítulo 2

Estado del arte

En los sistemas descentralizados, los nodos gozan de la libertad de no necesitar una entidad intermedia para poder comunicarse entre sí. En nuestro caso, esta transmisión de información se realizará a través de *Ethereum*.

En este capıtulo se estudiarán en profundidad los pilares criptográficos sobre los que se construye *Ethereum* y por qué se considera una plataforma segura.

2.1. Conceptos criptográficos

En primer lugar, es necesario dar una definición clara de qué se entiende cuando se habla de función hash y cuáles son sus características para considerarse criptográfica, según describe el libro [2].

Definición 1 *Una **función hash** es una función que cumple las siguientes propiedades:*

1. *Recibe una entrada de longitud arbitraria.*
2. *Produce una salida de tamaño fijo.*
3. *Es eficiente, es decir, tiene complejidad de cómputo lineal sobre el tamaño de la entrada.*

Más concretamente, lo que buscamos es una **función hash criptográfica**, la cual debe satisfacer además las propiedades de *resistencia a colisiones*, *hiding* y *puzzle friendliness*, que se analizarán en detalle a continuación.

Definición 2 *Una función H es **resistente a colisiones** si es inviable encontrar dos valores x e y tales que $x \neq y$, $H(x) = H(y)$.*

Es importante notar que pueden existir colisiones [1], pero lo relevante es que a nivel práctico, no sea factible encontrarlas por ser muy costoso.

[1] De hecho, en general estas coliciones existirán, ya que la entrada tiene un tamaño más grande que la salida.

Definición 3 *Una función hash H se dice que es* ***hiding*** *si conocida la función H y un valor* $H(r||x)$ *es inviable encontrar* x*, donde el valor de* r *es desconocido y* $||$ *denota concatenación.*

La interpretación de esta propiedad implica que dada la salida de una función hash $y = H(x)$ es inviable saber cuál era la entrada x.

Definición 4 *Una función hash H se dice que es* ***puzzle friendly*** *si, conocida la función H, un valor de salida y de n bits y el valor k es inviable encontrar x tal que* $H(k||x) = y$ *en un tiempo significativamente menor que* 2^n.

2.2. Estructuras de datos

Antes de describir las estructuras que usa *Ethereum* para el manejo eficiente de sus datos, es necesario aclarar que estos necesitan un previo tratamiento para alcanzar una uniformidad.

Con este fin, *Ethereum* aplica un algoritmo de serialización consistente, sobre toda estructura definida, ya sea una cuenta, una transacción o un bloque. En este caso, ***Recursive Length Prefix***, en adelante *RLP*.

A lo largo de este proyecto se va hacer uso de otros como *XML*, *JSON* y *Java Serialization*. Por una parte, *XML* y *Java Serialization* ya embebidos en Android permiten la transmisión de datos entre la vista y el controlador y entre las distintas *activities*, respectivamente. Por otra parte, se han definido objetos *JSON* propios para facilitar y controlar la interacción con las APIs externas a la aplicación, concretamente para el envío de notificaciones a través de *Firebase* y para la carga y almacenamiento de datos en *IPFS*, tal y como se verá en detalle en el capítulo 4.

2.2.1. Estructuras basadas en árboles

Antes de explicar en profundidad los árboles usados para el almacenamiento de datos en *Ethereum* se necesita exponer los conceptos de puntero hash y cadena de bloques.

Definición 5 *Un* ***puntero hash*** *es un puntero que además de señalar a la ubicación de los datos, almacena también un hash de los mismos.*

Utilizando este tipo de punteros, pueden definirse estructuras de datos formadas por nodos enlazados, siempre y cuando no tengan ciclos.

Definición 6 *Una* ***cadena de bloques*** *es una lista enlazada usando punteros hash. Cada bloque almacena un puntero hash al bloque anterior.*

Las cadenas de bloques constituyen un buen sistema para construir un registro resistente a falsificaciones ya que si se modifican los datos de un bloque, el puntero hash del siguiente pasa a ser incorrecto, propagándose la modificación a los siguientes. De este modo, solo se necesita saber el hash del bloque raíz para comprobar si ha sido o no alterado. Ese bloque especial que almacena el hash se llama *bloque génesis*.

Radix Trees

Definición 7 *Un **r-Radix Tree** es un árbol que optimiza el acceso a sus datos compactando los nodos que tengan un solo hijo con su antecesor, de modo que los nodos internos tendrán a lo sumo r hijos, donde* $r > 0$ *[58].*

También conocidos como *Compact Prefix Trees* en el contexto de *Ethereum*, los bloques representarían las hojas y tendrán un hash asociado que representará el camino, de modo que el coste de buscar un valor es lineal en la profundidad del árbol.

Merkle Trees

Definición 8 *Un **Merkle Tree** es un árbol binario enlazado con punteros hash [58].*

También conocidos como *Hash trees*, cada hoja almacena el puntero hash de un bloque de datos. De modo que se agrupan de dos en dos construyendo un nuevo puntero hash en función de los punteros hash de sus hojas. Este proceso se repite hasta llegar a la raíz.

Por esta estructura, al igual que en las cadenas de bloques, cualquier modificación en alguno de los nodos u hojas se propaga hacia la raíz y altera el resto de los nodos del camino. Por tanto solo necesitamos recordar la raíz para asegurarnos de que no ha sido modificado.

Además, esta propiedad permite comprobar de manera eficiente si un determinado bloque pertenece o no al árbol.

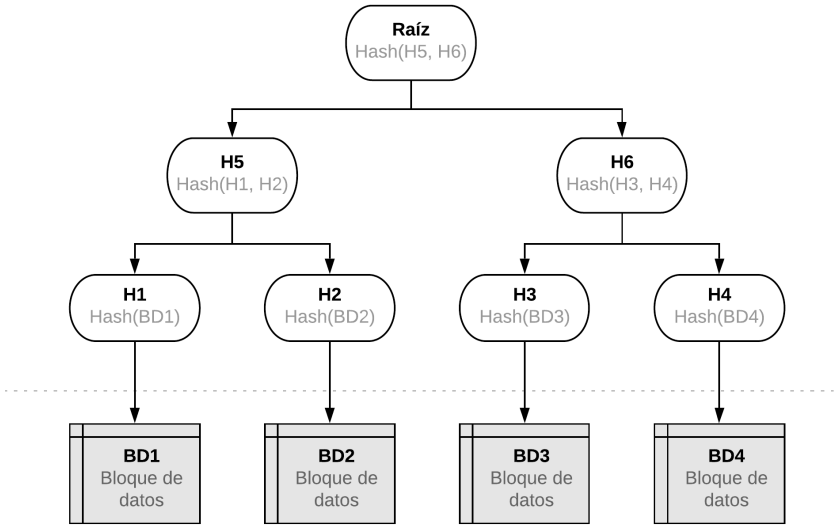


FIGURA 2.1: Merkle tree

Prueba de pertenencia. Dado un bloque y el camino hacia él, se puede comprobar que el bloque pertenece al árbol, con complejidad $\mathcal{O}(log\ n)$ sobre el número de nodos. Además, si el árbol está ordenado, podemos demostrar que un dato no está contenido en tiempo logarítmico viendo el camino de las dos hojas adyacentes según el criterio de ordenación.

Patricia Trees

De la combinación y optimización de los dos tipos de árboles anteriormente expuestos, *Ethereum* crea una nueva estructura denominada *Patricia Trees*, según [59].

Partimos de que todo dato de la *blockchain* se almacena como un par (a, b) donde

- *b = rlp(valor)* es el valor que se va a almacenar codificado en *RLP*.
- *a = sha3(b)* identifica el *path* del valor, siendo sha3 una función hash criptográfica con las características anteriormente explicadas.

Para evitar los valores nulos en el *path*, se aplica la lógica utilizada en los *Radix Trees* optimizando la búsqueda de datos, hasta llegar a la hoja donde el valor quedará codificado según RLP.

2.3. Ethereum

Ethereum se estructura como una cadena de bloques, como se ve en la figura 2.2. Cada bloque se compone de múltiples elementos entre los cuales nos vamos a centrar en el *hash* del bloque anterior, para asegurar la inmutabilidad y las raíces de tres *Patricia Trees*: uno global que actualiza el estado, uno de transacciones y uno con los recibos de las mismas.

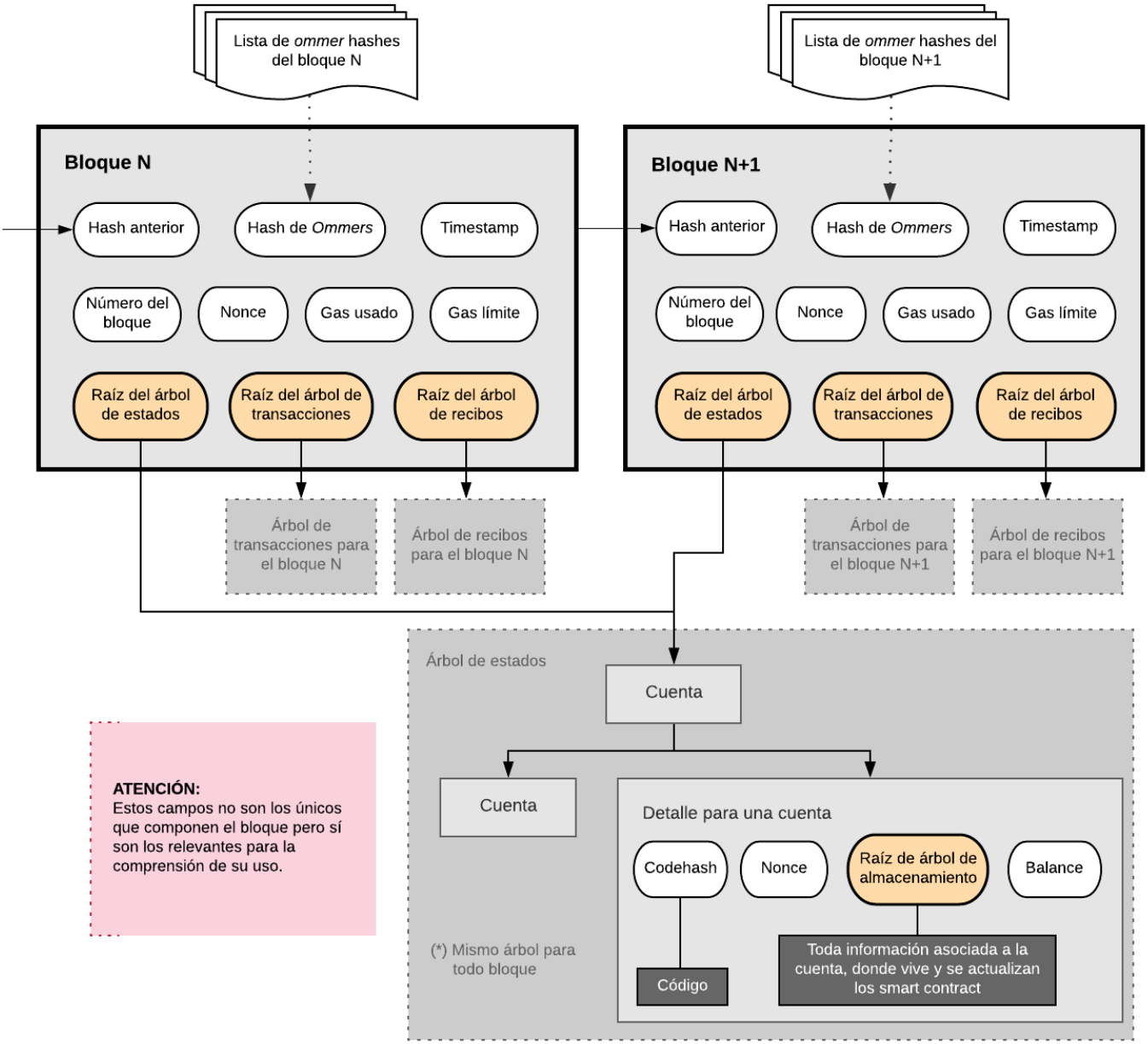


FIGURA 2.2: Estructura de los bloques de la red de Ethereum

2.3.1. Árbol de estado

Por una parte, el árbol del estado es global a todos los bloques y se va actualizando. En particular, almacena cada una de las cuentas con su información asociada, que se componen por 4 elementos:

```
[nonce, balance, storageRoot, codeHash]
```

- Nonce. Es un número natural que indica el número de transacciones hechas hasta el momento. No será posible confirmar una transacción desde una cuenta con $nonce = n$ si no se ha completado el envío de las anteriores con $nonce = 0, \ldots, n-1$.
- Balance. Es el saldo que tiene asociada la cuenta.
- StorageRoot. Es la raíz de un nuevo árbol de almacenamiento que contiene todos los datos de los contratos.
- CodeHash. Es el hash del código generado al crear la cuenta (en la máquina virtual de Ethereum). Es el único valor que permanece inmutable de estos cuatro.

Wallet

Es un programa software que guarda una clave pública y una privada e interactúa con distintas *blockchain* para permitir el envío y la recepción de transacciones.

Ligera o pesada. [18] Una cuenta pesada o *full node* tiene un acceso directo con la red de *Ethereum* y requiere descargar su contenido completo para operar. Este contenido pesa varios *gygabytes* y está creciendo constantemente.
Por el contrario, una cuenta ligera o *light client*, necesita de un nodo auxiliar para poder operar. En lugar de descargar todo el contenido, se conecta a un nodo que le proporcione la información necesaria y verifique sus transacciones. Se emplean cuando el dispositivo que las alberga no tiene o no se quiere utilizar una gran capacidad computacional, por ejemplo, para móviles.

Hot o cold [21] Una *hot wallet* se caracteriza por estar conectada a internet. Esto acelera el acceso a los fondos y las transacciones que se quieran hacer con ella, pero también implica un riesgo de seguridad y una mayor vulnerabilidad ante *hackeo* o *phising*. Mientras que una *cold wallet* se dice que se mantiene desconectada, por lo que la seguridad es mucho mayor, pero la accesibilidad a los fondos es menor.

Una vez explicados estos conceptos, describimos que tipos de *wallets* se han utilizado

para el desarrollo del proyecto. Por un lado, una *wallet* desde el navegador gestionada a través de *Metamask* que simplifica el despliegue de los contratos al conectarse con su compilador online, *Remix*. Por otro lado, una usada desde un dispositivo móvil, donde dicho nodo intermedio es generado mediante un *framework* denominado *Infura*. Siendo ambas *hot wallets* ligeras.

Sin entrar en detalles, lo más relevante es que sean compatibles con *Ethereum* y que al utilizarlas queden registradas en el árbol de estado junto con su actividad sobre la red.

2.3.2. Árbol de transacciones y árbol de recibos

Cada bloque tiene su propio árbol de transacciones y árbol de recibos. Ambos permanecerán inmutables en el momento en que bloque se complete. En particular, cada transacción está compuesta por 6 elementos:

```
[nonce, gasPrice, gasLimit, to, value, init/data]
```

Partiendo de una versión simplificada del concepto de transacción, cuyo objetivo es el envío con un cierto valor, *value*, a una dirección destino, *to*. Esta viene identificada por un número entero natural, *nonce*, que se va incrementando en 1 cada vez que una transacción se complete.

Ahora se extiende esta definición, viendo una transacción como cualquier interacción con contratos u otras cuentras de la red.

- Si es solo de lectura de datos, no tiene coste, ya que el contenido de la red no se modifica.
- Si implica una modificación de algún elemento de la red, tendrá un gasto mínimo asociado para cubrir el coste computacional del cálculo del hash de los datos enviados. Este gasto se conoce como tasa de transacción y su valor se obtiene mediante:

  ```
  transactionFee := (gasUnitUsed) x (gasPrice)
  ```

  donde el gas es la unidad que estima el esfuerzo necesario para enviar de una transacción y el precio del gas viene dado en *ether* o *ETH*. De aquí entendemos que el *ether* no solo es una criptomoneda sino que funciona como motor para mover todo el ecosistema de *Ethereum*. Ahora, como usuarios de *Ethereum* podemos fijar el precio del gas, pero no las unidades usadas ya que estas varían en función del peso de los datos enviados.

Cabe destacar que el concepto de contrato que estamos empleando se refiere a *smart contract* [4]. Es decir, un fragmento de código con funciones que permiten determinar unas reglas y acuerdos ejecutados al interactuar con él. Una vez habitan en la red de *Ethereum* pueden comunicarse con otros contratos o cuentas, tomar decisiones y almacenar datos, entre otras funcionalidades. Por estas características se les considera *smart*, ya que a pesar de haber sido definidos por desarrolladores, actúan de forma independiente. Su tiempo de existencia y ejecución están ligados a los de propia red, a menos que se hayan programado para autodestruirse. Además, al habitar en la red, adquiere su carácter descentralizado, inmutable y transparente.

En este proyecto los *smart contracts* tienen un papel fundamental para la accesibilidad de los datos generados por la aplicación, definendo métodos de almacenamiento y consulta de los mismos.

Una vez visto esto, surgieron una serie de dudas:

¿De dónde se extrae el *ether* empleado en las transacciones? Toda transacción viene asociada a una cuenta origen que pagará la tasa.

Si el precio de la transacción puede elegirse, ¿por qué no se toma un precio muy bajo o 0**?** Debido a que el precio del gas define la prioridad con que se quiere enviar una transacción, poner uno muy bajo puede implicar que no se envíe nunca, quedando en un estado bloqueado, *Pending*. Como se ha visto, se pueden lanzar varias transacciones desde la misma cuenta pero estas deben terminar en orden, para evitar conflictos con los *nonce*. Esto implica que quedarán en un estado bloqueado hasta que se termine el envío de las anteriores. Se propone una solución como futuro trabajo, en el capítulo 5.

Según la estación de gas [12], se recomienda fijar el precio a [2]:

- 2 GWEI $\Rightarrow$ Lenta ($<$ 30min).
- 2,1 GWEI $\Rightarrow$ Media ($<$ 5min).
- 3 GWEI $\Rightarrow$ Rápida ($<$ 2min).

Aunque hay que tener en cuenta que estos valores fluctúan a diario.

[2] Aunque oficialmente se utiliza ETH, el valor de la tasa de una transacción suele ser muy pequeño así que se utiliza en su lugar GWEI, siendo $1GWEI = 0{,}000000001ETH$.

Si no se puede controlar el número de unidades de gas usadas, ¿cómo se conoce cuánto se va a gastar en la transacción? Afortunadamente, al realizarla se puede fijar un valor máximo de pago por la tasa, conocido como *gas limit*. En caso de excederse, no se bloquea, sino que devuelve un recibo con estado fallido.
De nuevo siguiendo las recomendaciones de la estación de gas [12], el límite estándar por transacción es 21000. Además, aunque no hay un límite fijo directo para el tamaño de las transacciones, cada bloque tiene un límite en torno a 3000000. Esto equivale a 800*kb* de información transmitida, aproximadamente, lo cual sí restringe el tamaño de la transacción.

En particular en nuestro proyecto, tras realizar una serie de pruebas para estimar cuáles serían los valores para el precio del gas y límite del gas utilizado, se han fijado:

```
GAS PRICE := 1.5 GWEI = 0.0000000015 ETH
GAS LIMIT := 180000
```

Se puede destacar que el *gas limit* sugerido es bastante superior a la media, pues no se trata de una transacción estándar entre cuentas sino que contiene un alto volumen de datos. Por tanto, es más costosa.

¿Cómo se despliegan los *smart contract* en la red de Ethereum? La puesta en marcha de un *smart contract* se considera un caso particular de una transacción donde el código es inyectado en el campo *init* y se devuelve una dirección asociada junto con una interpretación del mismo que se ejecuta cada vez que se intente interactuar con él.

¿Cómo se puede interactuar con los *smart contract*? Una vez el contrato está en la red, conocida su dirección, los datos se transmiten a través del campo *data*.

Finalmente, hay que tener en cuenta que independientemente de si la transacción se ha realizado con éxito o no, la tasa se cobra y se devuelve un recibo que contiene información sobre el estado de la transacción, el gas usado y el bloque al cual ha quedado adherida.

Capítulo 3

Diseño de la aplicación

Este capítulo abarca todo lo relativo al diseño de la aplicación, describiendo la metodología del diseño guiado por objetivos y entrando en detalle en cada una de sus fases.

Diseño guiado por objetivos - *DGO*. Esta metodología se basa en el desarrollo del diseño de la interfaz centrándonos en el usuario, tal y como propone Alan Cooper [6].

- **Fase de investigación.** El proyecto comienza con la identificación de los objetivos de la aplicación en función de los comportamientos y motivaciones de las personas usuarias junto con un análisis de la competencia.
- **Fase de modelado y definición de requisitos.** Consistente en la modelización de perfiles que engloban las características y comportamientos principales del *target* de la aplicación, conocidos como *personas*.

  Dicho modelo se genera desde una perspectiva *top-down*, dando inicialmente una visión genérica del sistema que se quiere definir y a continuación, profundizar hasta alcanzar el nivel de detalle deseado o necesario para la validación del modelo.

  Finalmente conseguimos crear una *persona* que nos permite contextualizar la aplicación y definir los escenarios en los que usarse, así como los requisitos que ha de satisfacer.
- **Fase diseño del *framework*.** En esta fase se trabaja principalmente en el diseño de la interfaz en función de los requisitos. Se definen los elementos funcionales de la aplicación al igual que su distribución.
- **Fase de refinamiento.** Partiendo del diseño de la interfaz definido en la fase anterior, se evalúa y valida mediante entrevistas con personas expertas y usuarias

y se redefine. Este proceso itera tantas veces como sea necesario hasta conseguir el nivel de fidelidad deseado.

- **Fase de desarrollo.** Tras completar el diseño se inicia la implementación de la aplicación. En realidad, esta fase ha evolucionado en paralelo con el refinamiento y se describirá en el detalle en el siguiente capítulo 4.

3.1. Entrevistas

Con el fin de entender las necesidades de las personas usuarias de la aplicación, comenzamos la fase de investigación realizando entrevistas a un grupo de 10 mujeres de entre 18 y 30 años. Estas se llevaron a cabo entre el 30 de octubre y el 2 de noviembre, tanto de forma presencial como vía *Skype*, con una duración estimada de 30 minutos. Toda persona citada dio consentimiento previo para ser grabada y poder utilizar los datos extraídos con fines académicos. Aun así, debido al contenido sensible, se ha decidido exponer sólo lo necesario y más relevante para el desarrollo del proyecto a través de la extracción de factoides. Consultar tabla 3.1.

Aunque realmente no hay restricciones en el uso de la aplicación, esta franja se considera la más representativa a nivel de incidencia del problema.

1. **Creación de esqueletos y personas.** En función de la agrupación de factoides por su afinidad se procede a crear un esqueleto asociado, ver tabla 3.2, el cual proporciona información suficiente para identificar el target principal de nuestra aplicación y nos lleva al modelado de la *persona*. Finalmente, validando que cubre los factoides.

2. **Conclusiones.** Para finalizar la fase inicial de entrevistas, se han formalizado una serie de funcionalidades que va a cubrir la aplicación:

   • Petición de auxilio con múltiple alcance a:

   · Dispositivos que tengan la aplicación instalada y que se encuentren geográficamente cerca, agrupándose por código postal.

   · Grupos de Telegram predefinidos con envío mediante un bot.

   • Añadir puntos geográficos al mapa: título, descripción y opcionalmente foto, con gestión de la localización tanto automática como manual.

   • Consulta del mapa con los puntos que han compartido otros usuarios.

   • Creación de uno o varios canales para la transmisión de notificaciones.

   • Conectividad con otras aplicaciones del dispositivo: GPS, galería y cámara.

Agrupación de factoides Usuario Medio	Más Frecuente	Menos frecuente	Puntuales
Cuáles son las zonas peligrosas	· Zonas periféricas · Zonas humildes · Cuidades grandes · Sitios solitarios	Zonas con pubs o alcohol	· Transporte público, metro · Lugares de venta de droga
Por qué eligen esas zonas	· Vivencias personales · Rumores círculo cercano	· Noticias en los medios · Aspecto personas	Aspecto de calles y casas
Diferencia Madrid / ciudades más pequeñas	Similar	Peor Madrid	En pueblo pequeño, el peligro puede generalizarse a todo el pueblo
Si no conocen la ciudad	· Preguntar a la gente	Consulta foros o internet	Gente te avisa, antes de preguntar
El género influye en las zonas elegidas	Sí	Chicos también pueden encontrar peligro, pero menos	
Influye la franja horaria más peligrosa	Noche	-	· Indiferente · Influye época del año, más en verano
Han sentido inseguridad en la calle	Sí	-	No
Cómo actúan en situación de inseguridad	Huir de la situación	Ignorar	
Cómo actúan en situación de emergencia	· Huir de la situación · Búqueda de ayuda : gente cerca, gente confianza o servicios emergencia.	· Bloqueo ante miedo	· Defenderse
Cómo actúan si ven a alguien en situación de emergencia (en grupo)	Intervenir	Llamar para pedir ayuda	-
Cómo actúan si ven a alguien en situación de emergencia (solas)	Llamar para pedir ayuda	Intervenir	No hacer nada si desconoces el contexto
Se plantean evitar zonas peligrosas	Sí	No, suele ir en grupo	-
Forma de evitar zonas peligrosas	Ir por calles concurridas, aunque sea más largo	-	-
Denuncia de carteles			
Dónde encuentran carteles ofensivos	· Establecimientos · Paradas transporte publico	Anuncios de empresas	· Farolas · Carta bar
Actúan si un cartel les ofende	· Sí, deberían eliminarse · No había pensado en hacer nada	No, población demasiado sensible	
Cómo actúan si un cartel les ofende	Difusión circulo cercano	-	· Arrancarlo, si se puede · Hablar con dueño establecimiento
Utilidad de difusión de la existencia de los carteles	· Denuncia más eficaz · Visibilización	Más concienciación	· Habría que ir directamente al anunciante
Uso del sistema			
Compartir ubicación anónima	Sí	· Recalcan que sea anónima · No con desconocidos	No
Acudir a ayudar a un desconocido cerca	Sí	Sí, si puede intervenir	Sí, si no está ya policía
Cuándo no llevan el móvil	Lo lleva siempre encima	Cuando está cargando	· Cuando va acompañada · Para ir a un examen
Lugar para el móvil si no hay bolsillos	En la mano	· Cintura del pantalón · Mochila · Bolso	Funda colgada al cuello
Cuándo activan el GPS	· Puntualmente · Para aplicaciones concretas, mapas	Siempre	-
Por qué no lo activan	Problemas de privacidad	Problemas de batería	Problemas de rendimiento
Búsqueda de sitios nuevos	En casa antes de salir en ordenador y movil	· De camino en el movil · Preguntar a la gente	Mapas de metro y similares
Puntos violeta	· Conocen su existencia, pero no que son · Saben lo que son, pero no están familiarizadas	· Saben lo que son y como encontrarlos	No saben lo que son
Opinión puntos violeta	Buena idea	Deberían estar en todas partes	-
Difusión de existencia y localización de los puntos violeta	A favor	-	-

Usuaria básica de la App

Edad : $16 - 45$ años

Sexo : Mujer

Descripción	Persona joven que está estudiando o trabajando y ha vivido o visto algún tipo de acoso callejero. Está bastante familiarizada con el uso de dispositivos móviles y aplicaciones.
Formación	• Estudios básicos o de bachillerato completados. • No tiene formación específica en informática. • Ha oído hablar de los puntos violeta pero no tiene claro qué son, ni dónde aparecen.
Problemas	• Si surge o ve una situación de peligro en la calle, tiende al bloqueo sin saber a quién avisar. • No soporta los anuncios ofensivos que se encuentra por la calle. • Tiende a evitar zonas solitarias. • A veces le gustaría volver a casa acompañada.
Objetivos	• No quiere seguir sintiéndose insegura en la calle, ni que otras personas también. • Desaparición de carteles que le resulten insultantes.
Información adicional	• Si va por la calle y ve un conflicto, es más probable que actúe si va en grupo. • No suele llevar el GPS activado pero sí lo haría si con ello puede ayudar o ser ayudada.

FIGURA 3.2: Esquelteto de persona usuaria básica

• Soporte no restringido a un área geográfica, tanto para registro de alertas como carteles, es decir, con acceso global.

• Compatibilidad para distintas versiones de Android. Debe ser igual o superior a la versión *Marshmallow 6.0* (o $API > 23$) ya que la creación y gestión de las cuentas de Ethereum se realizan a través de una librería que requiere un mínimo SDK. Esto engloba en torno al 71 % de los dispositivos activos en el ecosistema Android.

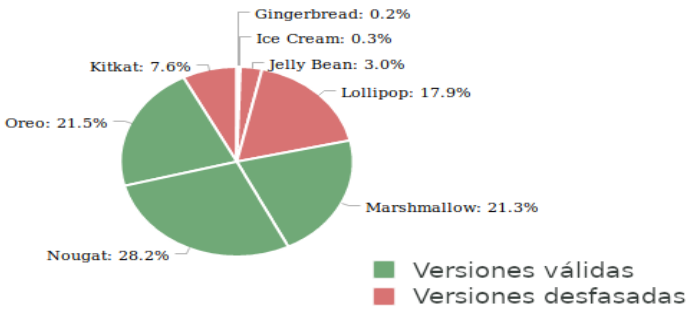


FIGURA 3.3: Versiones de Android de los dispositivos activos

Se pueden consultar más detalles sobre las entrevistas en el apéndice A.5.

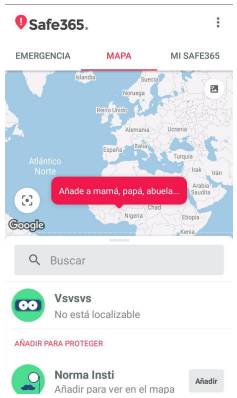
3.2. Análisis de la competencia

Tras haber fijado el núcleo del proyecto en función de las conclusiones extraídas de las entrevistas, se realizó un estudio de la competencia comparándolo con otras aplicaciones que ya están en el mercado.

Se procede con la búsqueda de aplicaciones para distintas plataformas móviles, tanto iOS como Android, que tengan competencias parciales en común. A continuación se analiza en detalle cada una, ordenándolas de mayor a menor según el impacto social que hayan tenido y la similitud con nuestro proyecto.

Safe365. [47] (anteriormente Alpify)

FIGURA 3.4: safe365



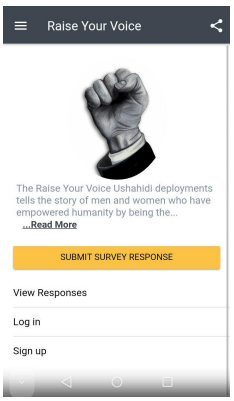
Esta aplicación permite enviar un aviso en caso de emergencia a otros usuarios de la misma. Está orientada a personas de la tercera edad o dentro de un ámbito familiar de confianza, ya que dichas alertas quedan restringidas a una lista predefinida de tus contactos, llamados *protectores*, de modo que la información que se transmite es más concisa y personal, entre ella: nombre, foto, número de teléfono, localización y batería restante. Esta información puede consultarse en cualquier momento, sin necesidad de realizar un aviso.

- Plataformas soportadas: Android, iOS y web
- Número de usuarios: +1000000
- Puntuación: 4,1 / 5
- Contra: Al estar compartiendo continuamente datos con el resto de usuarios, tiene un alto consumo de batería y datos móviles.

Ushahidi. [61]

Gestiona y organiza por categorías datos de encuestas o comunidades que quieren compartir sus experiencias. En concreto, una de las funcionalidades reporta espacios públicos donde se ha producido acoso sexual, junto con una descripción.

FIGURA 3.5: Ushahidi



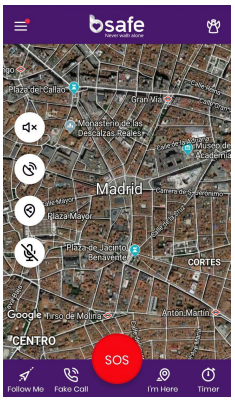
Estos datos quedan distribuidos en un mapa donde los usuarios pueden interactuar para corroborar la información.

- Plataformas soportadas: Android, iOS y web
- Número de usuarios: +1000
- Puntuación: 4,4 / 5
- Pro: Es open source
- Contra: Banco de datos muy grande, la búsqueda de información concreta se hace más compleja.

bSafe. [1]

Esta aplicación funciona como botón del pánico. Previamente configuras un conjunto de usuarios de confianza como *guardianes* y en caso de emergencia, puedes elegir entre solicitar ayuda presencial o una llamada telefónica.

FIGURA 3.6: bSafe



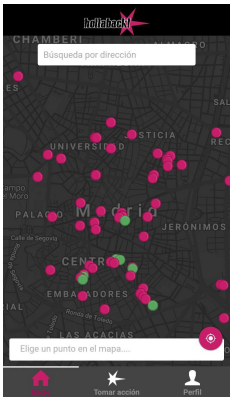
Adicionalmente, tiene funciones secundarias como la activación por voz o la reproducción en directo para monitorizar a la persona auxiliada hasta que esté a salvo. Sin embargo, estas funcionalidades adicionales son de pago.

- Plataformas soportadas : Android y iOS
- Número de usuarios: +500000
- Puntuación: 3,65 / 5
- Contras: Muchas funcionalidades son de pago y no incluye todas las regiones (solo USA y Noruega).

Hollaback. [16]

Esta aplicación registra en un mapa agresiones y acosos callejeros, creando un código de colores para distinguir las vivencias suministradas.

FIGURA 3.7: Hollaback



Además permite a los usuarios interactuar con el contenido del mapa, para mostrar apoyo a otras personas.

- Plataformas soportadas: Android y iOS
- Número de usuarios: +1000
- Puntuación: 4,1 / 5
- Contra: No incluye todas las regiones.

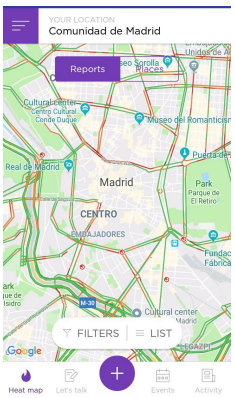
No more. [39]

Permite en caso de emergencia avisar a otros usuarios que estén geográficamente cerca de quien pida ayuda. No encontrada en las plataformas de distribución de aplicaciones oficiales ya que aún está en desarrollo.

Safecity. [48]

Esta aplicación facilita reportar acosos producidos en espacios públicos a través de un mapa. Permite configurar alertas sobre lugares frecuentados o nuevas localizaciones, así como leer y contar experiencias de personas que las valoran. Adicionalmente, funciona como foro y promueve la difusión de eventos relacionados con feminismo.

FIGURA 3.8: SafeCity



- Plataformas soportadas: Android y iOS
- Número de usuarios: +1000
- Puntuación: 4,7 / 5

Circleof6. [5]

Permite guardar a 6 contactos que elijas de tu agenda y avisarles en caso de emergencia, para una llamada o chat o para que te vayan a auxiliar.

- Plataformas soportadas : iOS
- Puntuación : 4,7 / 5
- Contras : No mantienen versión para Android.

Todas las aplicaciones seleccionadas tienen descarga gratuita aunque luego puedan ampliar funcionalidades de pago.

Además de las aplicaciones móviles, se han tenido en cuenta otros dispositivos que funcionan como botón de emergencia físico. Aunque estos suelen estar restringidos a municipios o pequeñas regiones, como en Durango [62] o San Juan [45] y contactan directamente con los cuerpos de seguridad de la zona.

3.2.1. Dimensiones del producto

Dado que no se han encontrado aplicaciones que unifiquen los dos conceptos principales (auxilio y mapa informativo), el análisis los trata por separado, esperando una respuesta binaria (✓/ ✗). Por una parte, la petición de ayuda:

Aplicaciones	Config	UAge	UGeo	Tlf	Presen	Nombre	Foto	Local
Safe365	✗	✓	✗	✗	✗	✓	✓	✓
Ushahidi	✗	✗	✗	✗	✗	✗	✗	✗
bSafe	✓	✓	✓	✓	✓	✓	✓	✓
Hollaback	✗	✗	✗	✗	✗	✗	✗	✗
Safecity	✗	✗	✗	✗	✗	✗	✗	✗
Circle of 6	✓	✓	✗	✓	✓	✓	✓	✓

CUADRO 3.1: Dimensiones referidas a las compentencias de la petición de ayuda

- Alcance.
  - · **Config**: Permite elegir a qué usuarios pides ayuda.
  - · **UAge**: Notifica a usuarios predefinidos de la agenda o de la app.
  - · **UGeo**: Notifica a usuarios de la app geográficamente cerca.
  - · **Tlf**: Ayuda telefónica.
  - · **Presen**: Ayuda presencial.
- Información proporcionada.
  - · **Nombre**: Nombre y descripción del problema.
  - · **Foto**: Foto del usuario.
  - · **Local**: Localización del usuario.

Por otra parte, la información que gestiona el mapa:

Aplicaciones	Descrip	Foto	Clasif	Manual	Auto	Unir	Inter
Safe365	✓	✓	✗	✗	✓	✗	✗
Ushahidi	✓	✗	✓	✓	✗	✓	✗
bSafe	✓	✓	✗	✗	✓	✗	✗
Hollaback	✓	✗	✓	✓	✗	✗	✓
Safecity	✓	✗	✓	✓	✗	✓	✓
Circle of 6	✗	✗	✗	✗	✗	✗	✗

CUADRO 3.2: Dimensiones referidas a las compentencias al mapa

- **Descrip**: Título y descripción.
- **Foto**: Fotografía.
- **Clasif**: Clasificación de puntos ya sea por colores u otros distintivos.
- Distintas formas de añadir contenido:
  - **Manual**: Manualmente.
  - **Auto**: Automáticamente.
- **Unir**: Unificación de puntos que estén juntos.
- **Inter**: Interactuar con el contenido del mapa, para corroborarlo o comentarlo, no solo lectura.

Tras ver los detalles del análisis, se concluye confirmando la existencia de productos que cubren nuestras necesidades, pero parcialmente. Por tanto, nuestro trabajo consiste en unificar estas funcionalidades y además hacerlo de forma descentralizada para mantener el pseudoanonimato de los usuarios.

3.3. Generalidades de diseño

Partiendo de que el desarrollo se presenta en torno al sistema operativo Android, se especifican unos estándares para realizar el diseño de la aplicación. Este proyecto aplica **Material Design** [31] como guía de estilo para la creación de los elementos funcionales por estar ampliamente extendido dentro del ecosistema Android, aunque no sea exclusivo al mismo.

De igual modo, los bocetos y el prototipado se han realizado sobre plantillas de dimensiones 720×1280 y orientación vertical por uniformidad. Aunque estas son las proporciones estándar usadas por los desarrolladores, la interfaz final se reajustará en la medida de lo posible a las dimensiones específicas del dispositivo. Esto es posible gracias al uso de *ConstraintLayout* [3] en cada una de las vistas, que consiste en un ajuste de los elementos y márgenes relativo al tamaño de los contenedores, sin necesidad de anidarlos.

Asimismo la paleta de colores e iconos seleccionados siguen las directrices de *Material Palette* [32], manteniendo la simplicidad, pero sujeta a ciertos cambios respecto a la profundidad de los elementos.

3.4. Navegación

Adicionalmente, para alcanzar una interfaz sin colisiones de flujo o la aparición de puntos sin retorno, se esquematizó en un diagrama 3.9 todas las posibles interacciones con la aplicación.

3.5. Planificación de las entrevistas y evolución de la interfaz

Previamente a la evaluación de la interfaz se redacta un conjunto de escenarios de contexto que cubran los casos de uso de la aplicación y sobre los que la persona entrevistada pueda sentirse identificada en una situación real. Este guión puede consultarse en la segunda sección del apéndice A.5.

Ahora, con el fin de alcanzar un diseño que cubra los objetivos propuestos y sea intuitivo para los usuarios y usuarias, hemos elaborado un plan de diseño consistente en tres ciclos de entrevistas.

3.5.1. Prototipo en papel

La primera iteración plantea un prototipo elemental realizado a papel. Tras la puesta en común de los bocetos y justificando los diseños en función de las generalidades

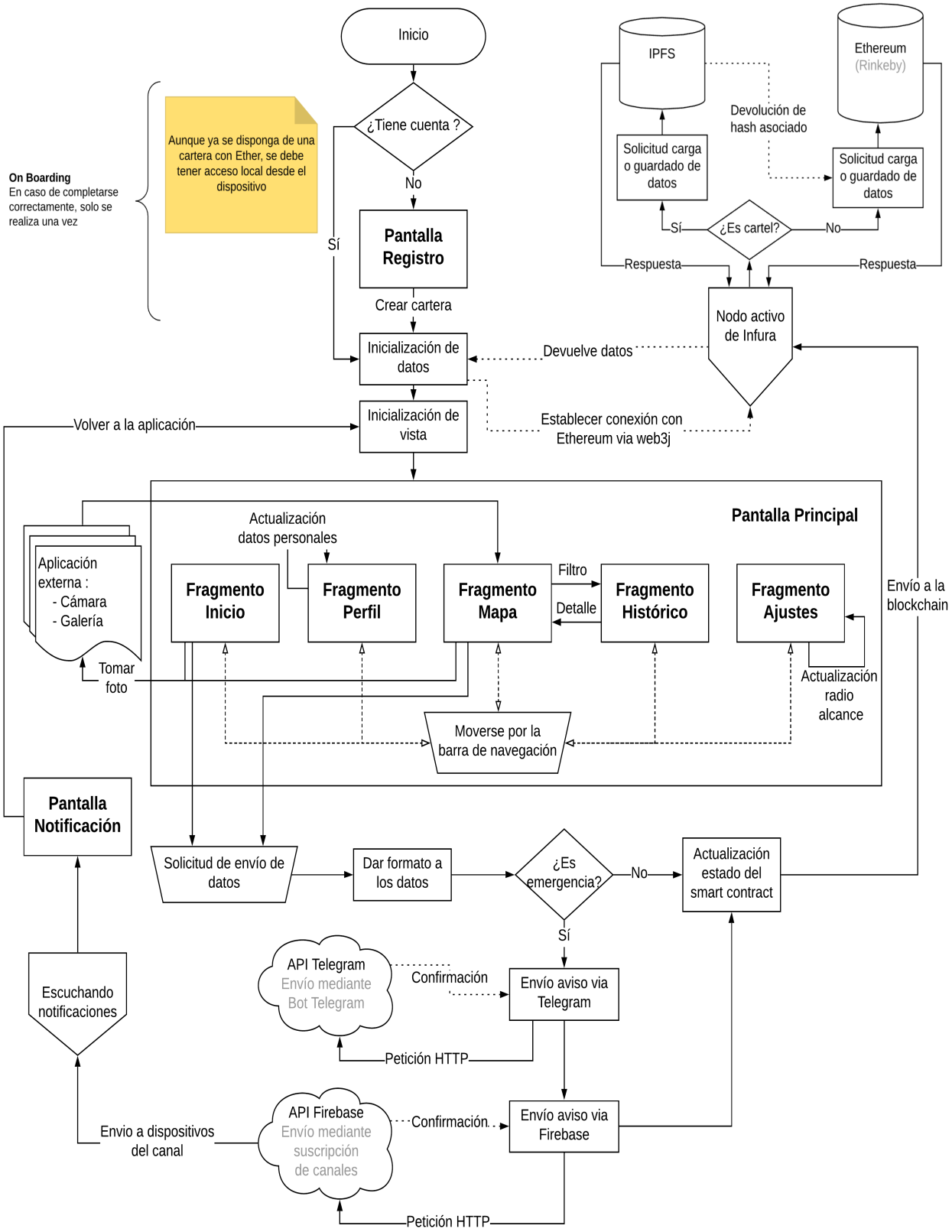
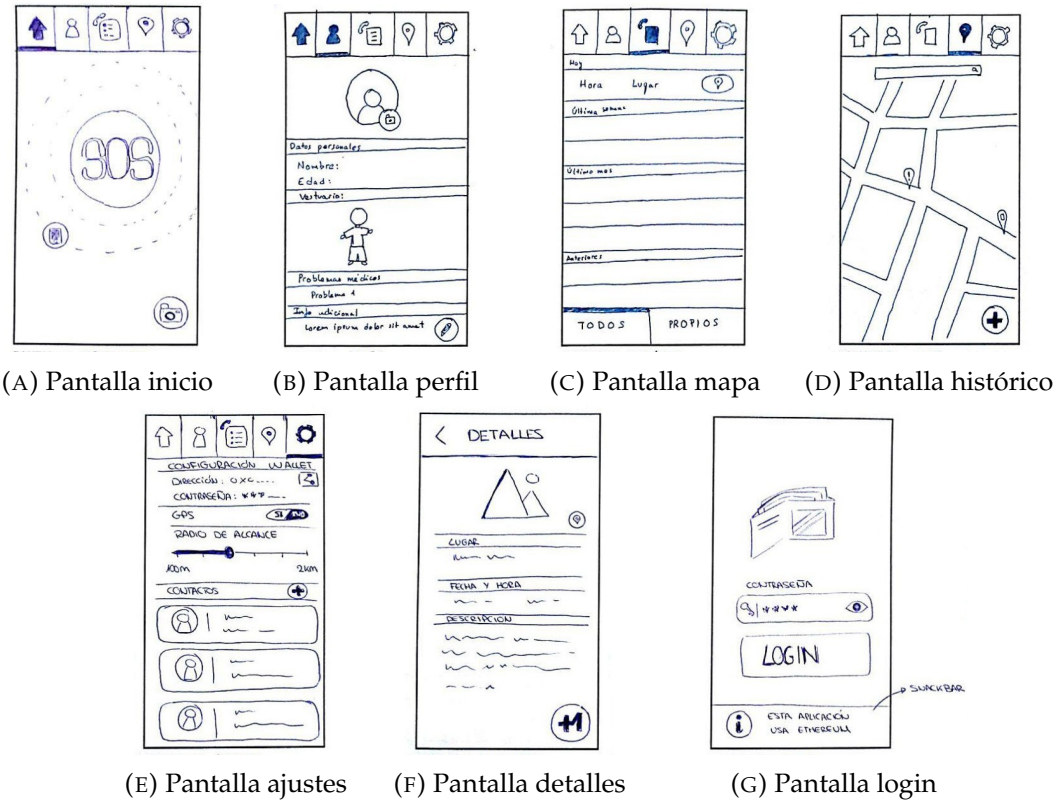


FIGURA 3.9: Diagrama de flujo de la aplicación

preestablecidas, combinamos las ideas llegando a la figura 4.6 con siete pantallas : registro, principal, perfil, mapa, histórico, ajustes y detalles.



(A) Pantalla inicio (B) Pantalla perfil (C) Pantalla mapa (D) Pantalla histórico

(E) Pantalla ajustes (F) Pantalla detalles (G) Pantalla login

FIGURA 3.10: Pantallas del prototipo en papel

Para poder testar esta interfaz cómodamente sin necesidad de acarrear papeles, se empleó la aplicación *Marvel*, anteriormente conocida como *Pop*. Esta digitaliza los bocetos permitiendo incluir zonas de interacción que establecen conexiones entre las distintas pantallas.

Tras completar la primera iteración con 7 personas entrevistadas, quedaron identificados dos problemas graves de la aplicación:

- La confusión entre la pantalla de ajustes y la de perfil, ya que en ambas se accede a la configuración de algún aspecto de la aplicación.

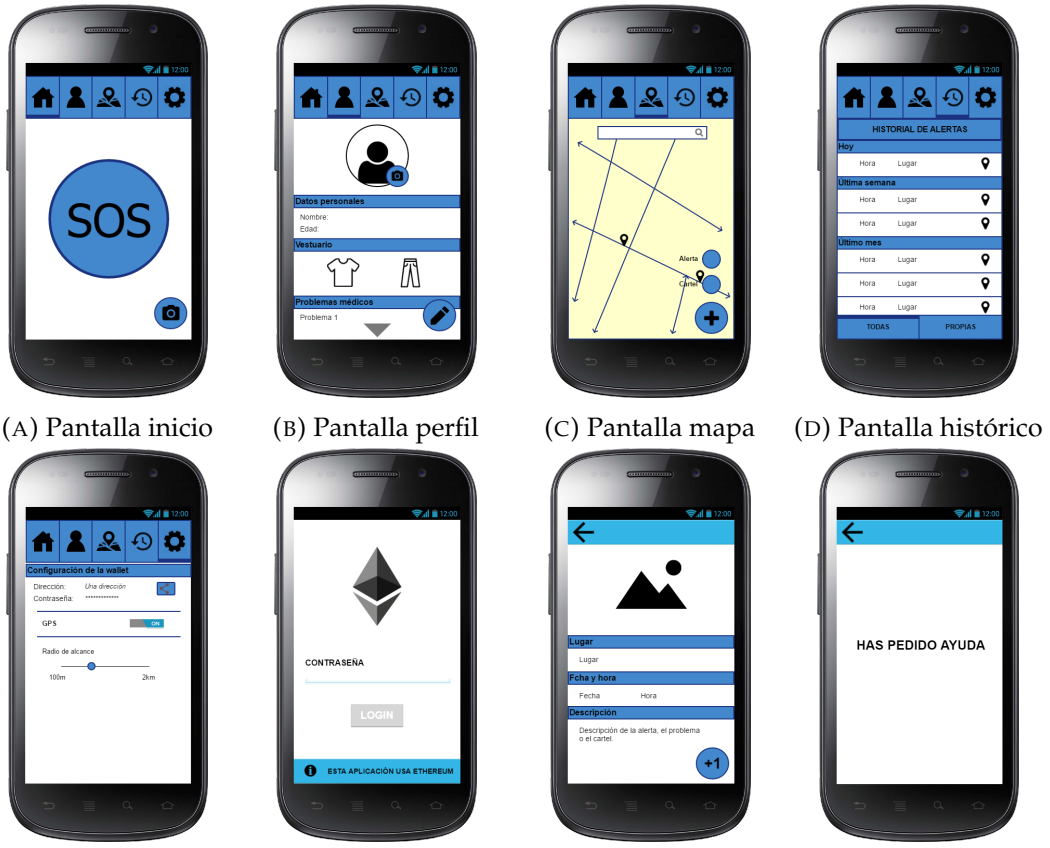
  Solución: Unificar los datos personales en la pantalla perfil y dejar los datos correspondientes a la cuenta de *Ethereum* en la pantalla de ajustes.

- La falta de feedback por parte de la aplicación, pues en muchos casos, costaba reconocer si un elemento de la vista era interactivo o simplemente pertenecía al fondo.

  Solución: Crear un prototipo digital, añadiendo colores y profundidad a los elementos y algunas pantallas auxiliares de reacción.

3.5.2. Prototipo digital

Una vez resueltos los problemas encontrados en función de los comentarios recibidos, se diseña una versión digital con *Pencil*, como muestra la figura 3.11.



(A) Pantalla inicio (B) Pantalla perfil (C) Pantalla mapa (D) Pantalla histórico

(E) Pantalla ajustes (F) Pantalla login (G) Pantalla detalles (H) Pantalla respuesta

FIGURA 3.11: Pantallas del prototipo digital

En esta segunda ronda de evaluación, se reconocieron:

- La falta de información transmitida para la identificación de una persona que envía la alerta.

  Solución: Incluir un *santo y seña*[1] y un campo abierto donde se puede introducir la información adicional que se desee.

- El desconocimiento de las criptomonedas. Aunque se intenta abstraer el uso de *ether* en la aplicación, la pantalla de ajustes resulta confusa al no saber en qué consiste la cuenta y el saldo que muestra.

  Solución: Incluir un punto de información desplegable que explica en detalle qué es y cómo usar la dirección de la cuenta asociada.

[1]Definimos la estructura de *santo y seña* como un sistema rápido de pregunta y respuesta para identificar a la persona que ha enviado la alerta de una forma discreta.

3.5.3. Prototipo final

Finalmente, para medir la fiabilidad del modelo se plantearon los mismos escenarios a dos profesionales en usabilidad, a quienes se les solicitó rellenar una plantilla de evaluación basadas en las heurísticas de Nielsen, con una escala de severidad prefijada del 1 al 10, de menor a mayor.

- N01: Visibilidad del estado del sistema.
- N02: Relación entre el sistema y el mundo real : lenguaje cercano.
- N03: Control y libertad de interacción.
- N04: Consistencia y estándares.
- N05: Prevención de errores.
- N06: Reconocimiento mejor que recuerdo.
- N07: Flexibilidad y eficiencia.
- N08: Estética y diseño minimalista.
- N09: Ayudar a reconocer, diagnosticar y recuperarse de errores.
- N10: Ayuda y documentación.

Posteriormente se unifican los comentarios recibidos en función de la violación de las heurísticas, destacando y resolviendo los errores de diseño que se ha considerado que tenían mayor prioridad.

N01	N02	N03	N04	N05
Separación en el histórico de los carteles y las alertas.	Falta de uniformidad para designar alertas, registros o avisos, en alertas.	Falta de feedback cuando se pulsa el botón de ayuda.	Registro de marcadores en localización distinta a la actual.	Botón de ayuda demasiado susceptible a ser pulsado.
	Botón de ayuda produce pánico al pulsar.			Botón de segmentar el mapa no se entiende.
N06	**N07**	**N08**	**N09**	**N10**
Separación innecesaria de ajustes entre la pantalla ajustes y perfil.	Mapa e histórico muestran información distinta.			Información sobre la cuenta de Ethereum ilegible para quien no conozca la tecnología.
	Falta de filtros en el histórico.			

Problemas solucionados

FIGURA 3.12: Resumen de heurísticas.

Tras terminar la fase de diseño, llegó el momento del desarrollo de la lógica. Se exponen en detalle en el siguiente capítulo 4 las pantallas de registro, mapa, histórico y un apartado del perfil.

Capítulo 4

Implementación de la aplicación

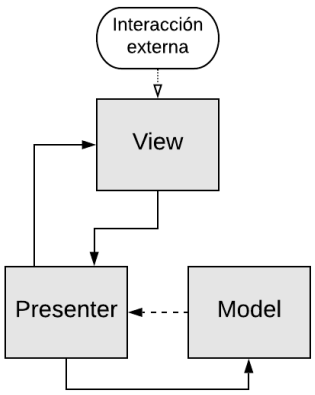
Este capítulo comienza con la introducción y justificación del empleo de los métodos elegidos para la implementación, continuando con la descripción de la estructura y lógica aplicada en cada pantalla de la aplicación.

4.1. Métodos aplicados

MVP: Model-View-Presenter [36].
Desde el punto de vista de la programación orientada a objetos, se puede definir como una derivación del patrón modelo-vista-controlador.

- El presenter se encarga de definir toda la lógica.
- La vista es la receptora de cualquier interacción con la persona usuaria y encargada de notificar al presenter. Normalmente, se corresponde con las activities y los fragments.
- El modelo es una interfaz implementada en la vista y con invocación desde el presenter. Su función es informar a la vista de que su trabajo ha terminado y trasmitirle los datos actualizados que se quieren mostrar.

FIGURA 4.1: Model View Presenter



Clean Architecture.
Es una arquitectura por capas donde se desacopla la lógica, de la vista y los datos. La motivación de su uso es facilitar el mantenimiento de la aplicación, volviéndose flexible a cambios tanto funcionales como de agentes externos. Esto permite crear test unitarios más sencillos y tener una mayor granularidad de los componentes.

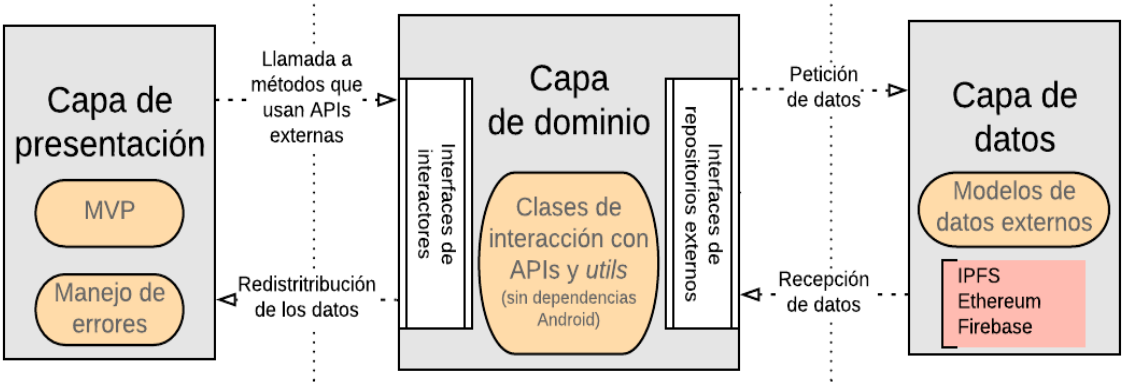


FIGURA 4.2: Capas del modelo *Clean Arquitecture*

Esta arquitectura se ha definido siguiendo los patrones descritos en *Software Architecture Patterns* [43] que esctructura el proyecto en las siguientes capas:

- **Capa de presentación.** Contiene todas las clases referidas al patrón MVP que quedarán clasificadas en tres paquetes tal y como se ha explicado en el apartado anterior.

- **Capa de dominio.** Sirve como conexión entre los datos y la interfaz, incluye la definición de los casos de uso que llevan a cabo los procesos de lógica requeridos por los *presenter* de la capa de presentación.
  El objetivo de esta capa es crear una independencia de la lógica respecto de los datos para simplificar el proceso de testeo de cada componente.

- **Capa de datos.** Se encarga de generar los modelos para los datos tanto guardados localmente como externos, en nuestro caso procedentes de la API de Firebase y el nodo de *Infura* conectado a *IPFS* y *Ethereum*. Estos modelos suelen ser objetos en Java conocidos como POJOs y contienen métodos de inicialización y consulta *(set/get)* y en caso de ser necesario, la serialización y deserialización de los datos. Particularmente, se ha elegido la notación JSON como formato para los modelos, véase el ejemplo 4.1 de un cartel en formato JSON, tal y como quedará guardado en IPFS.

LISTING 4.1: Ejemplo de cartel serializado como JSON

```
{
  "title": "Ejemplo poster ",
  "description": "...la descripcion...",
  "latitude": "40.215797",
  "longitude": "-3.770023",
  "datetime": "1556548361127",
  "image": "iVBORw0KGgoAAAANSUhEUgAABLAAAAAGQCAYAAAAC
      +tZleAAAABmJLR0QAAAAAAAD5Q7t/AAAACXBIWXMAAC4j
      AAAuIwF4pT92AAAAB3RJTUUH4wQIFQgUpN5yQAAABl0RV
      h0Q29tbWVudABDcmVhdGVkIHdpdGggR0lNUFeBDhcAACA
      ASURBV..." (imagen codificada)
}
```

Llamadas asíncronas.
Con el fin de agilizar la interacción, reduciendo tiempos de espera en la ejecución principal del programa, todas las llamadas a la *blockchain* y a cualquier API se realizan de forma asíncrona, usando programación reactiva con *RxJava* [46]. Esta librería funciona mediante la construcción de bloques observables y subscriptores:

- Los objetos observables están compuestos por uno o varios métodos que emiten información.
- Los objetos subscriptores generalmente crean una hebra alternativa a la ejecución principal para escuchar a los observables. Se encargan de consumir la información y redistribuirla en *Subscriber.onNext()* si hay más de un item, *Subscriber.onComplete()* si se ha consumido toda la información o *Subscriber.onError()* si algo ha ido mal.

Supongamos que al utilizar la aplicación se manda una alerta. Para completar este proceso hay que conectar con varias APIs, lanzar dicha alerta y esperar la respuesta, como se detallará más adelante. Esta información puede tardar bastante en adquirirse y depende de la señal de la conexión a Internet. Si se realizase en la hebra principal, cualquier interacción se añadiría a la cola de tareas de ejecución dando la sensación de parada. Por eso se ha decidido implantar esta metodología en el proyecto.

4.2. Tecnologías aplicadas

Para el desarrollo de este proyecto, se ha buscado en todo momento que los servicios, programas y librerías aplicadas funcionasen bajo licencias no privativas, fomentando el uso del software libre. Asimismo, para permitir la visibilidad, modificación y uso de nuestro proyecto, se ha implantado una licencia MIT.

Se detallan a continuación los programas utilizados así como una breve descripción de su uso y la licencia asociada. Véase la figura 4.3.

4.2.1. Documentación

Contenido de la memoria.
Desarrollo, feedback y control de versiones del documento gestionado mediante *Overleaf* [40], en *LaTex*.

Bibliografía.
Todo libro, *paper*, artículo u otro tipo de documentación se almacena a través de *Zotero* [65] y es exportado como *BibTex* para poder ser introducido cómodamente en la memoria.

Software	Licencia
Overleaf	MIT
Zotero	GNU GPLv3
Remix	MIT
IPFS	MIT
Metamask	MIT
Marvel	MIT
Pencil	GNU GPLv2
Gravit	Apache-2.0
Android	GNU GPLv2
Java	GNU GPL
Mapbox	MIT
API Firebase	Apache License 2.0
API Telegram	GNU GPLv2
Infura	Apache-2.0
Web3j	Apache-2.0

FIGURA 4.3: Software utilizado junto con sus licencias

4.2.2. Software

Github.
Servicio *web-hosting* para gestionar versiones de software con Git, donde se han establecido unas reglas de contribución al proyecto. Pueden consultarse los detalles del código en el repositorio:

https://github.com/patriciaTel/SpreadApp.

Blockchain

- **Remix.** [42] Compilador online de *Solidity* aplicado en la implementación, testeo y despliegue de los *smart contract*.
- **IPFS - InterPlanetary File System.**[25] Protocolo y red diseñados para compartir contenido en un sistema de archivos distribuidos sobre la metodología *P2P*, donde el contenido almacenado se direcciona con un hash. En este caso, los carteles.
- **Metamask.**[34] Aplicada como extensión en el navegador, es un gestor de cuentas que da soporte tanto a la red principal de *Ethereum* como a las de desarrollo.
- **Rinkeby.**[44] Testnet de Ethereum para realizar pruebas sin tener un coste real de *ether*.

Diseño de la aplicación

- **Marvel, anteriormente Pop.**[29] Aplicación que permite la interacción con un prototipo en papel digitalizado.
- **Pencil.**[20] Herramienta de creación de prototipos, en nuestro caso de baja fidelidad.
- **Gravit.**[17] Herramienta online usada para la creación de iconos e ilustraciones vectoriales del proyecto.

Implementación de la aplicación

- **Android Studio y herramientas SDK.**[33] Entorno de desarrollo para aplicaciones Android, construido sobre *IntelliJ IDEA*.
- **Mapbox.**[28] Proveedor y customizador de mapas.
- **API Firebase.**[15] Administrador y distribuidor de notificaciones entre dispositivos sin necesidad de servidor.
- **API Telegram Bot.**[53] Administrador y gestor del comportamiento de los bots de Telegram, necesario para enviar las alertas a los grupos predefinidos.

Conexión *blockchain* ⇔ Aplicación Android

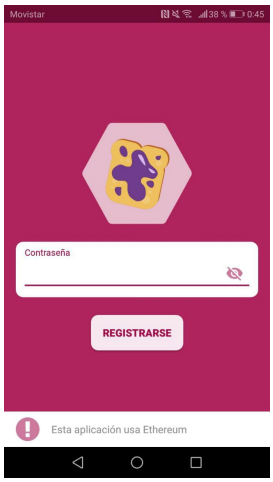
- **Infura.**[22] Como se ha visto anteriormente, la cuenta que vamos a crear y utilizar en la aplicación debe ser ligera y por tanto necesitamos un nodo para establecer la conexión con Ethereum. Infura proporciona dicho nodo tanto para la red principal como para desarrollo, en nuestro caso, *Rinkeby*.
- **Web3j.**[63] Librería de Android que permite la conexión al nodo de Infura sin sobrecargar la aplicación, dando acceso al último estado de la *blockchain* actualizado.

Una vez expuestas las tecnologías y métodos aplicados se profundiza en los detalles de la implementación de la aplicación. Para ello se va seguir el diagrama de flujo 3.9, donde se identifican las pantallas como los elementos visuales con los que se interactúa. En Android, se conocen como *activities* y generan la vista en función del *XML* asociado. A continuación se describe en detalle cuál es el contenido de cada una de ellas.

4.3. Pantalla registro

La pantalla de registro aparece al inicio del flujo. Es común a muchas aplicaciones y si se completa correctamente solo aparece la primera vez que se abre la aplicación tras la instalación. El objetivo principal es comprobar si el dispositivo posee una cuenta o *wallet* local asociada y en caso de no tenerla, crearla con la contraseña elegida.

FIGURA 4.4: Pantalla registro



Al realizar las pruebas de la aplicación, la mayoría de las personas entrevistadas no tenían nociones sobre criptomonedas, por lo que se decidió incluir una breve explicación. De igual modo, tras completar el registro, se conduce al usuario por un breve tutorial 4.5 para exponer las funcionalidades que va a encontrarse.

El registro no volverá a aparecer, pero tanto el tutorial como la información sobre criptomonedas quedarán accesibles en todo momento desde el fragmento de ajustes.

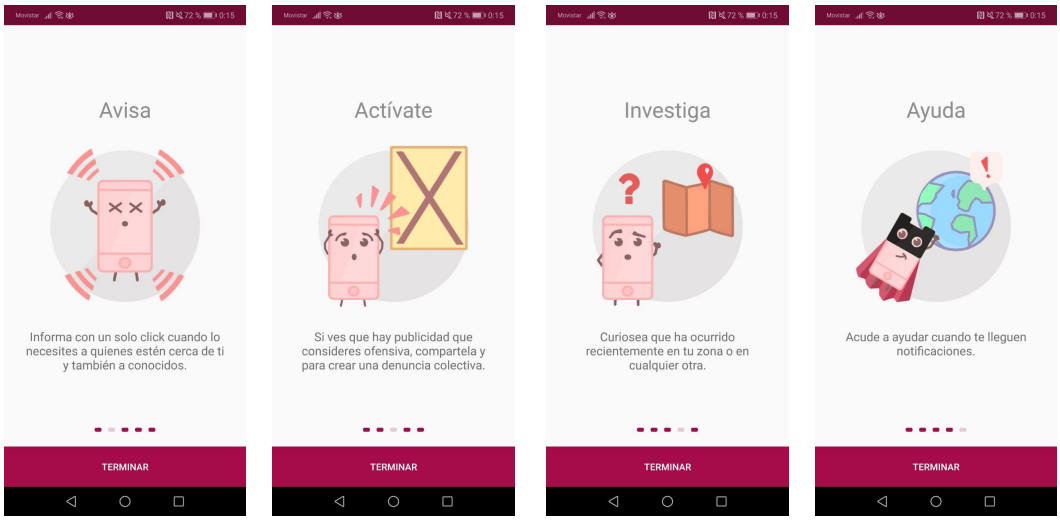


FIGURA 4.5: Pantallas del tutorial

4.4. Pantalla principal

Al iniciar la pantalla principal, el primer paso es sincronizar la *wallet* local con la red con el fin de recibir cualquier transacción nueva. Para ello, establecemos una conexión con *Ethereum* a través del nodo proporcionado por *Infura*. Una vez actualizado el estado de la *wallet*, se solicitan los datos guardados por la aplicación. Estos datos externos serán los marcadores del mapa para las alertas y los carteles que se encuentran en los *smart contract* y se van a distribuir a las distintas componentes de la pantalla.

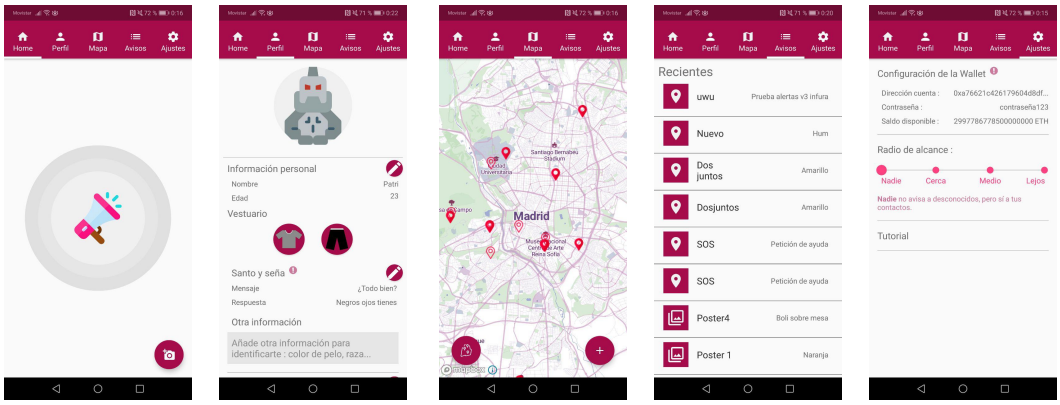
El segundo paso es la inicialización de un servicio de geolocalización del dispositivo. Primero, se debe informar y pedir permisos al usuario para poder recopilar esta información.

En caso de no aceptar los permisos solo podrá publicar contenido de forma manual y en ningún momento podrá recibir alertas del resto de usuarios y usuarias.

Suponiendo que sí se han aceptado, el servicio inicializado funcionará como un subproceso de la aplicación que se ejecutará cuando la aplicación esté abierta o en segundo plano. Su tiempo de vida termina únicamente cuando el usuario cierre explícitamente la aplicación o cuando se rechacen los permisos. Dicho servicio crea un objeto que escucha a los proveedores *NETWORK_PROVIDER* y *GPS_PROVIDER* del dispositivo, es decir, detecta los cambios de localización en función de la red *WiFi* y el *GPS*. Está configurado para que se actualice cada 5 segundos (como mínimo) y con una distancia de error de 10 metros. Cuando se detecta un cambio, primero se guarda localmente la nueva localización y acto seguido, se deben actualizar los canales de escucha de notificaciones para recibir las alertas.

Esta pantalla está compuesta por cinco fragmentos que van rotando y están interconectados por una barra de navegación situada en la parte superior. Se ha elegido utilizar esta estructura porque los fragmentos permiten mantener la autonomía de cada una de las funcionalidades y a su vez depender de unos mismos datos globales suministrados por la pantalla principal. De hecho, por estar en la misma *activity*, se pueden inicializar todos a la vez y que la interacción entre ellos requiera un tiempo de espera ínfimo.

- Fragmento inicio. Aparece por defecto cada vez que se abra la aplicación tras completar el resgistro y permite la emisión de las alertas.
- Fragmento perfil. Almacena localmente la información personal que se va a compartir en las alertas.
- Fragmento mapa. Muestra un mapa con las alertas y carteles almacenados en la blockchain.
- Fragmento histórico. Muestra un listado de las alertas y carteles recientes, acotados por las coordenadas del mapa.
- Fragmento ajustes. Muestra la información de la cuenta con *ether* y permite configurar el radio de alcance de las alertas.



(A) Fragmento inicio (B) Fragmento perfil (C) Fragmento mapa (D) Fragmento histórico (E) Fragmento ajustes

FIGURA 4.6: Fragmentos de la pantalla principal

Se va a profundizar a continuación en los fragmentos del mapa y el histórico.

4.5. Fragmento mapa

La publicación y consulta de las alertas y carteles generados es una de las funcionalidades principales de la aplicación. Esta pantalla en conjunto con la pantalla del histórico, explicada en la siguiente sección, cubren dicha funcionalidad.

4.5.1. Tipos de marcadores

Se muestran los distintos marcadores para diferenciar entre carteles y alertas a simple vista, como explicita la figura.

(A) Marcador de alertas

(B) Marcador de carteles

4.5.2. División por zonas

Además de consultar directamente la información en el mapa, la aplicación permite visualizar fácilmente mediante un código de colores qué zonas tienen registradas más alertas.

Para lograr este objetivo se ha elegido el cálculo del diagrama de Voronoi. Aunque este método solo se ha aplicado a la ciudad de Madrid, con los datos de entrada necesarios, queda generalizado a cualquier otra ciudad tal y como se describe a continuación.

Obtención de datos
En primer lugar se necesitan una serie de puntos estratégicos, de aquí en adelante se denominarán **sitios**. Los puntos elegidos son las coordenadas de las estaciones de metro de Madrid que facilita el Consorcio Regional de Transportes de Madrid, en adelante CRTM [27] en su página web, con formato UTM.

Por consistencia con el formato de coordenadas tomado en la librería de Mapbox, el siguiente paso ha sido convertirlas de las originales UTM a coordenadas geográficas. Para ello, debemos saber en que se diferencian. Por un lado, el sistema de coordenadas *UTM* secciona en cuadrículas un mapa como si fuese una matriz y proporciona un par de coordenadas (x, y) que determinan el punto buscado dentro de la cuadrícula, $30T$, en el caso de Madrid. Mientras que el sistema de *coordenadas geográficas* traza

arcos en dirección horizontal y vertical conocidos como paralelos y meridianos. Estas coordenadas esféricas quedan identificadas como latitud (horizontal) y longitud (vertical), obviando la altitud ya que nuestro modelo es en 2D.

Por tanto, buscaríamos la conversión :

$$(\text{zona}\ \ x\ \ y) \Rightarrow (\text{latitud}, \text{longitud})$$

Conversión de coordenadas

Se va a seguir un artículo de IBM para la conversión de coordenadas [49]. El primer paso es descargar el *.csv* de la página del CRTM y guardar los datos de las columnas x e y localmente en un archivo de texto. Su formato por coordenada será:

```
"30 T valorX valorY"
```

Ahora podemos invocar el método de conversión que nos proporciona la clase *CoordinateConversion* y quedarnos de nuevo con los pares de retorno (latitud, longitud) y guardarlos en *Coordinates.txt*.

	zone	l	north	east	longitude	latitude
1						
2	30	T	4475360	440100	-3.7061486381994078	40.4267192021832
3	30	T	4473846	443960	-3.6605134609254155	40.4133492333852
4	30	T	4477258	440632	-3.700054757719159	40.443855373765835
5	30	T	4472069	434921	-3.7668611901217464	40.39668370765227
6	30	T	4471170	440111	-3.7056246492507667	40.38897445848729
7	30	T	4475118	443392	-3.6673209581031325	40.42476965450273
					⋮	

FIGURA 4.8: Muestra del archivo de coordenadas

Cálculo de polígonos

Una vez obtenidos los sitios en el formato deseado, se procede al cálculo de los polígonos que delimitan su área. Dado que se va a operar sobre el mapa que es un plano en $\mathbb{R}^2$, podemos aplicar el método de **diagramas de Voronoi** y usar sus propiedades en el plano euclídeo.

Primero, dado un conjunto de n puntos p_i con $i = 1 \dots n$, definimos la distancia euclidea de cada par $p_i,\ p_j\ \forall i, j \in R$ tales que

$$||p_i - p_j|| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Ahora se calculan las distancias de todos los puntos entre sí y trazando las *mediatrices*[1].

[1]Se dice *mediatriz* de $p_i,\ p_j$ a la recta perpendicular al vector director $p_i p_j$ que pasa por el punto medio entre p_i y p_j.

Estas generan n hiperplanos H cuya intersección engendra un polígono de a lo sumo n lados, denominado, **región de Voronoi** y quedando descrito por :

$$V(p_i) = \bigcap_{j=1,\, i \neq j}^{n} H(p_i, p_j)$$

El conjunto de todos los polígonos disjuntos compone un teselado que se denomina ***diagrama de Voronoi*** plano.

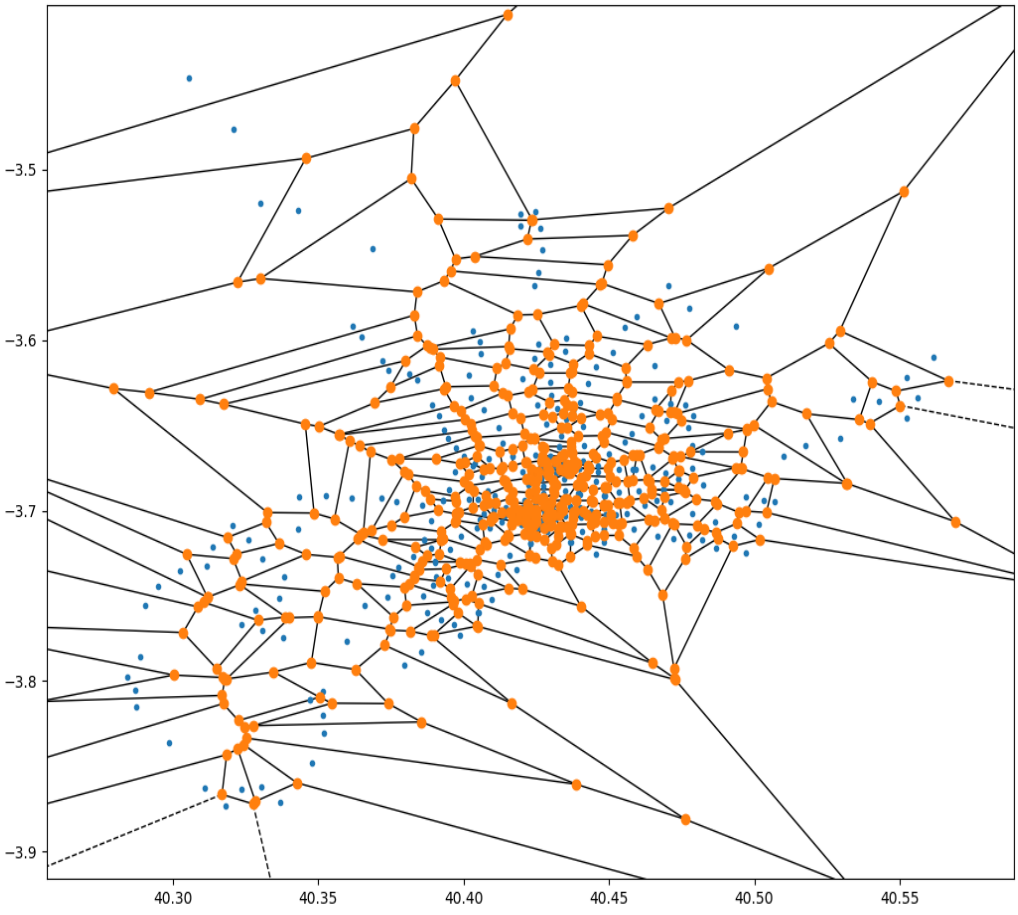


FIGURA 4.9: Representación del diagrama de Voronoi sin mapa

Como hasta el momento los datos van a mantenerse fijos, tanto la extracción y el formateo de coordenadas de puntos como el cálculo de los polígonos puede hacerse localmente e importar estos datos para evitar cálculos innecesarios cada vez que se inicie la aplicación.

Aplicándolo sobre las estaciones de metro se obtiene un gráfico como el representado en 4.9, donde los ejes se mueven en torno a $(-40{,}25, 40{,}60)$ y $(-3{,}41, -3{,}91)$ y se corresponden con los límites de la latitud y longitud de Madrid, respectivamente.

Implementación El teselado de Voronoi se genera mediante un *script* en Python usando el módulo *NumPy* [38]. El coste en tiempo del algoritmo es del orden $O(n \log(n))$, siendo n el número de sitios. Primero se leen las coordenadas de *Coordinates.txt* que representarán los sitios. A continuación se aplica el algoritmo sobre los mismos. Finalmente, tras su cálculo, se escriben en un archivo las coordenadas de los vértices de cada polígono en sentido horario, donde cada línea representa un polígono, como se ve en la figura 4.10.

```
1  40.4248313866831452401l -3.52422209302441746814 40.41502758493117397620 -3.40898974788930919644
2  40.32089327119717125925 -3.47617113572817615363 40.18077330727185625392 -3.52913861877475021345
3  40.33024838154158686621 -3.51932226900616385024 40.01439747032248561709 -3.58538335255585671746 •••
4  40.29512569875472394187 -3.74449817474094803771 39.87692837422624592136 -3.58058157736750715117
5  40.29062148557424194451 -3.75555489750654558634 38.45043489202413411476 -3.51176278184780166214
                                                    ⋮
```

FIGURA 4.10: Muestra del archivo de coordenadas de los polígonos

Representación de los polígonos

Ya se puede hacer uso de los polígonos ya que tenemos su definición a través de sus vértices ordenados.

Conocidos estos datos, se emplea la librería de *Mapbox* para Android para su representación sobre el mapa. Cada polígono genera una capa nueva debido a que estas no pueden mezclar distintos elementos geométricos y se asigna un identificador: *polygonLayerX*, para poderlas mostrar u ocultar según el interés.

Una vez dibujados, queremos saber qué color asignarle y para ello, hay que ver a qué polígono pertenece cada punto.

En primer lugar, se identifica cada sitio p_i con su polígono correspondiente, P_i. A continuación, usando la definición del diagrama, se afirma que las aristas de los polígonos definen la situación geométrica donde cada par de sitios más cercanos entre sí, son equidistantes. De este modo, dado un punto x y el conjunto de sitios $p_1 \dots p_n$, el polígono al que pertenece p viene determinado por la distancia mínima de entre las posibles. Puede verse gráficamente en la figura 4.11.

Además, dado que el interés no está en conocer la distancia en sí misma sino la mínima alcanzable, por proporcionalidad este cálculo se optimiza eliminando la raíz cuadrada.

Sea n es el número de sitios y m el número de marcadores, esta búsqueda tendrá coste $O(nm)$, por ser lineal respecto al número de sitios, pero con necesidad de realizarla para cada punto añadido.

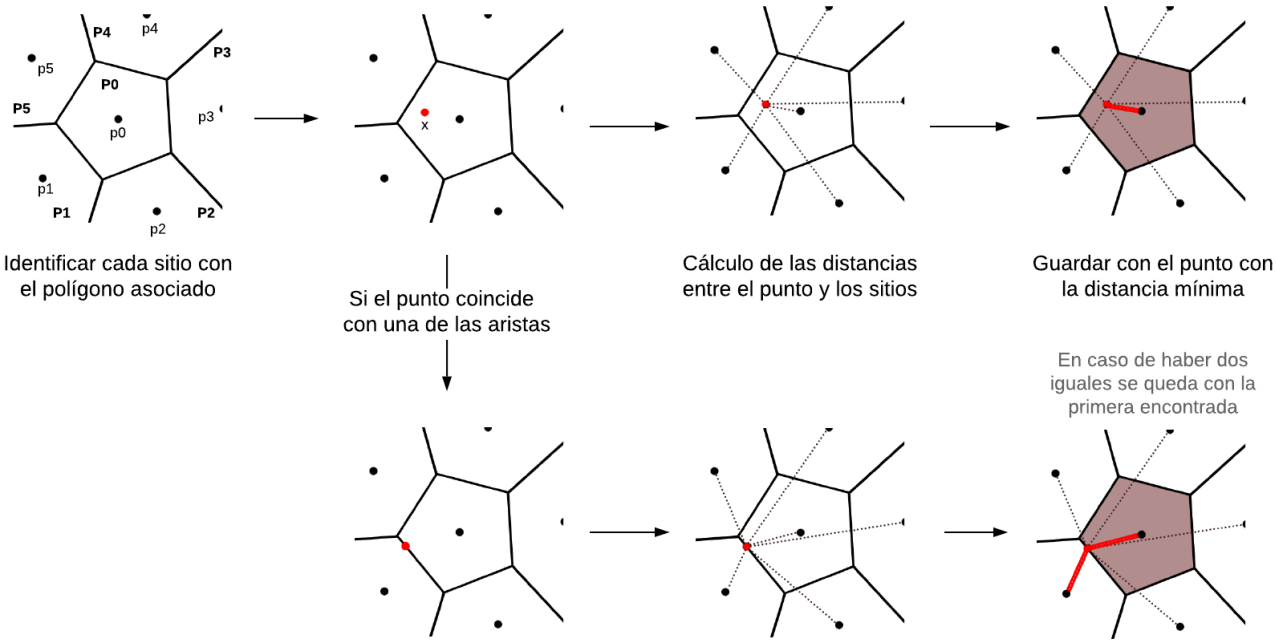


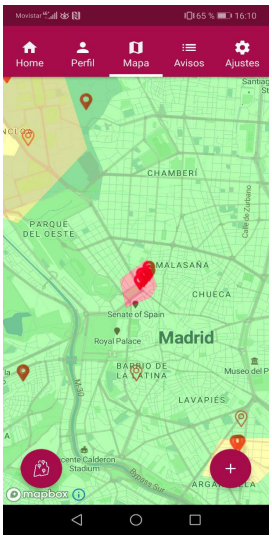
FIGURA 4.11: Búsqueda del polígono

Se concluye esta sección definiendo un código de colores para distinguir zonas identificadas como peligrosas:

- Rojo ⇒ Zona peligrosa, si el número de alertas es superior a 5.
- Amarillo ⇒ Zona peligrosa circunstancialmente, si el número de alertas está entre 2 y 5.
- Verde ⇒ Zona tranquila, si el número de alertas oscila entre 0 y 1.

Puede verse el resultado final en la figura 4.12.

FIGURA 4.12: Mapa por zonas



4.5.3. Añadir marcadores

Otra de las funcionalidades del mapa es añadir alertas o carteles manualmente. El problema reside en la imposibilidad de publicar algo en la aplicación de manera inmediata ya sea por la falta de conexión a internet o por el desconocimiento de su existencia, pudiéndose reportar *a posteriori*.

FIGURA 4.13: *Dialog* para añadir un cartel

En el proceso de añadir el marcador, se despliega una pantalla emergente donde se puede introducir un título, una descripción y en caso de ser un cartel, una imagen, que puede importarse desde la galería o tomarse en el momento. Véase en la figura 4.13. Además, por tener activo el servicio de rastreo anteriormente descrito, permite elegir entre la geolocalización automática o manual.

Visto como se gestionan los marcadores localmente dentro de la aplicación, véase a continuación como se almacenan en la red de *Ethereum*.

4.5.4. Integración con *blockchain*

En lugar de tomar una base de datos tradicional para almacenar los datos recopilados por la aplicación, se ha optado por el uso de la tecnología *blockchain*.

Por una parte, los datos compartidos pueden tener un contenido sensible por lo que se requiere mantener el *pseudo-anonimato*. Se habla de *pseudo-anonimato* o *anonimato relativo* porque toda información permanece pública, así como quien la emite. Sin embargo, la persona emisora viene identificada por la dirección de la cuenta con la que se ha realizado la transacción, no por sus datos personales. Por esta razón, se quiere evitar que una entidad los posea y pueda dar una mala gestión de los mismos.

Por otra parte, se desea que la información no se modifique en beneficio de ninguna de las partes. Por ejemplo, para crear una mala fama a un barrio en particular. Esto se evita gracias a la inmutabilidad que ofrece la *blockchain* como se ha repasado en el capítulo 2.

SmartContract - Solidity

El primer paso para almacenar datos es crear un modelo para esos datos y una forma de interactuar con ellos. En nuestro caso queremos generar dos estructuras: *Poster* y *Alert*. Ambas necesitan almacenar un título, una descripción, una localización dada por el par (latitud, longitud), una fecha y hora y adicionalmente, el cartel debe incluir una imagen.

```
struct Poster {
    string title;
    string description;
    string latitude;
    string longitude;
    string date;
    bitmap image;
}
```

```
struct Alert {
    string title;
    string description;
    string latitude;
    string longitude;
    string date;
}
```

Puesta en marcha

Una vez fijadas las estructuras de datos en la *blockchain* se desea poder manipularlos. Se ha optado por almacenar los datos en listas de *Poster* y *Alert* y definir métodos que añaden y leen los elementos de las mismas.

Tras la implementación, compilamos el contrato. Se puede hacer tanto a través de *Remix*, como de la propia terminal. Por comodidad se elige la opción de la terminal (instalando *Solidity Compiler* y *Web3j*) para tener localizados los archivos *.abi* y los *.bin* asociados al contrato. Para generarlos:

```
solc <smart-contract>.sol --bin --abi --optimize -o /<output-dir>/
```

Estos archivos son necesarios para su encapsulado como clase Java, como se verá en el siguiente apartado.

Con el fin de hacer uso del mismo es necesario desplegarlo en un entorno de desarrollo para evitar costes del proyecto, en nuestro caso en *Rinkeby*. Para ello, se necesita una cuenta previamente creada con *Metamask* donde se ha ingresado *ether*.

Encapsulado

Finalmente, teniendo el *smart contract* desplegado en la *blockchain*, se establece una conexión desde la aplicación para tener acceso a los métodos y conocer cuáles son los parámetros de entrada y salida. Para alcanzar este objetivo:

- Por una parte, se han de guardar las direcciones generadas en el despliegue.

```
AlertContract : 0x715b16b2e6d68ac3c4f270621e11e3a2798346a4
PosterContract : 0x5a1575f6b1577b5d0817f6f075ff441552fa4259
```

- Por otra parte, encapsular el contrato como una clase Java e invocarlo dentro de nuestro proyecto.

```
web3j solidity generate /<path-to>/<smart-contract>.bin
  /<path-to>/<smart-contract>.abi -o /<output-dir>/ -p <package-name>
```

IPFS - InterPlanetary File System

Viendo ahora la estructura de almacenamiento de información de carteles en nuestra aplicación, se entiende que para que un cartel tenga sentido, debe tener asociado al menos una imagen. Si esta se guardase directamente en la *blockchain*, cada transacción sería bastante pesada y consumiría una gran cantidad de *ether* o incluso podría no realizarse si excede el límite del tamaño del bloque, como se explicó en el capítulo 2. Para solventar este problema, se hace uso de IPFS.

- Primero se necesita establecer una conexión con IPFS desde nuestra aplicación, al igual que se hizo con Ethereum, Infura facilita un nodo para conectarnos.
- Se serializa el objeto convirtiendo la información relativa a cada cartel en un *JSON*.
- Se almacena esta información en un nodo de IPFS que nos devuelve un identificador *hash*.
- Este hash de tamaño prefijado será guardado en la *blockchain* para poder tener acceso a su contenido, reduciendo así el coste de la transacción.

Cabe decir que este protocolo no es realmente necesario para guardar las alertas de emergencia, ya que la información se reduce a la estructura *Alert*, correspondiente a cinco *strings*.

En el apéndice B queda a disposición del lector el fragmento de código correspondiente al *smart contract* tanto para los carteles como para las alertas en *Solidity*.

Ajustes necesarios

A lo largo del desarrollo, despliegue y el encapsulado de los *smart contracts* se han encontrado algunos problemas que se quieren destacar de cara a aportar una ayuda a futuros proyectos:

- Al comenzar la implementación del *smart contract* se necesitaba almacenar las coordenadas de los puntos que tienen una precisión de 14 decimales. Sin embargo, *Solidity* no permite guardar datos numéricos no enteros.

  Como solución se convierten a *strings*, aunque otra alternativa habría sido separar la parte entera y decimal y guardarlas como dos enteros.

- Hay muchas versiones de compiladores de código *Solidity* y no todas son compatibles entre sí, asi que por uniformidad se fija la versión *5.1-commit* para su desarrollo.

- El último problema se detectó al encapsular los contratos como clases Java, ya que los métodos se generan en una versión de Java distinta a la utilizada por la librería de Android. En respuesta a esto, hay que descartar manualmente aquellos que tienen como argumento *ContractGasProvider*, pues este tipo de datos no está definido y usar en su lugar tomar los desfasados que usan *BigInteger*.

4.6. Fragmento histórico

La aplicación cuenta con un apartado de consulta de información adicional con la intención de no sobrecargar la vista desde el mapa. Estos dos apartados van intrínsecamente ligados ya que en el histórico se muestran las alertas que se encuentran en la vista actual del mapa y viceversa, un elemento de la lista redirecciona al mapa.

4.6.1. Filtrado de datos

- Al inicializar el fragmento se pasan como argumentos:

  Las dos listas de datos para alertas y carteles.

  Las coordenadas de los extremos inferior izquierdo y superior derecho de la cámara del mapa. Partiendo de que la latitud y la longitud crece en dirección $\nearrow$, podemos definir el rectángulo acotado sobre el que queremos buscar.

$$\text{extremo1} := (lat_{init}, long_{init})$$

$$\text{extremo2} := (lat_{final}, long_{final})$$

- Ahora para filtrar los datos de la lista simplemente comprobamos si cada $(lat_j, long_j)$ punto dado está dentro del rectángulo:

$$lat_{init} \leq lat_j \leq lat_{final} \text{ y } long_{init} \leq long_j \leq long_{final}$$

Si es la primera vez que se abre el histórico sin haber abierto previamente el mapa, los extremos tomados son los límites de las coordenadas en el globo, es decir, $\text{extremo1} := (-90, -180)$ y $\text{extremo2} := (90, 180)$. Si no, el mapa escucha activamente cada vez que se hace un cambio en la vista, actualizando los valores de los extremos del rectángulo, que se guardan localmente.

4.6.2. Estructura de datos

Continuando con la implementación, Android tiene dos estructuras que permiten la visualización listada de elementos: *ListView* y *RecyclerView*.

Por una parte, *ListView* permite una declaración sencilla de datos pero, cuando el volumen de datos es muy grande la vista tarda mucho en cargarse o incluso puede llevar al cierre espontáneo de la aplicación por sobrecarga. Por eficiencia se ha elegido usar *RecyclerView* cuyas principales ventajas frente a *ListView* son la declaración de un *ViewHolder* para la reutilización de las celdas al hacer scroll sobre la lista y el desacoplamiento del contenido del contenedor mediante un *LayoutManager*, lo que

permite añadir nuevo contenido dinámicamente.

Finalmente queda por definir el contenido de cada item de la lista, que se implementa mediante dos niveles para no sobrecargar la vista con demasiada información para las personas usuarias. Se puede consultar en 4.14

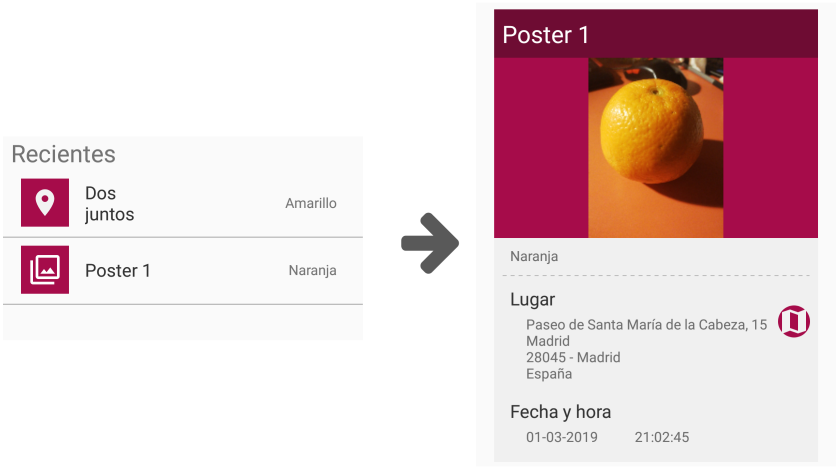


FIGURA 4.14: Información histórico

- **Primer nivel.** Consiste en un icono para diferenciar fácilmente entre alerta y cartel, un título y una breve descripción.
- **Segundo nivel.** Si se tiene más interés en el contenido de uno de los item, se despliega un panel *layout* añadiendo detalles de localización, fecha y hora y una imagen adicional en caso de ser un cartel.

4.6.3. Rotación en el mapa

Con la técnica del rectángulo anteriormente expuesta al hacer rotaciones en la cámara del mapa, pueden quedar excluidos algunos marcadores del mapa, ya que la superficie queda transformada.

Prosiguiendo con una visión geométrica de la solución 4.16. Se quiere ahora comprobar si un punto P pertenece o no a un rectángulo independientemente de su ángulo de rotación, en particular,razonando mediante el cálculo de las áreas. Por una parte, se calcula el área del rectángulo en cuestión usando la definición de distancia como se había hecho antes:

$$\mathcal{A}_{\text{rectángulo}} := ||B - A|| \cdot ||D - A||$$

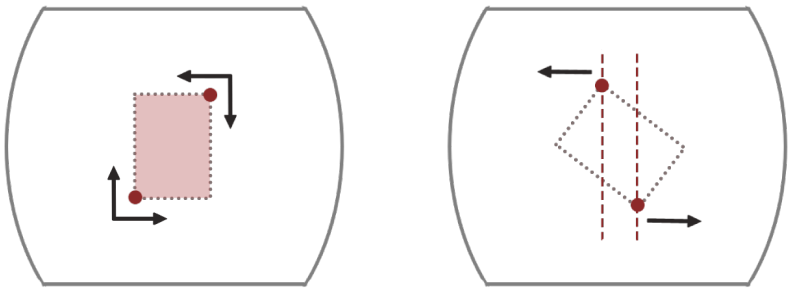


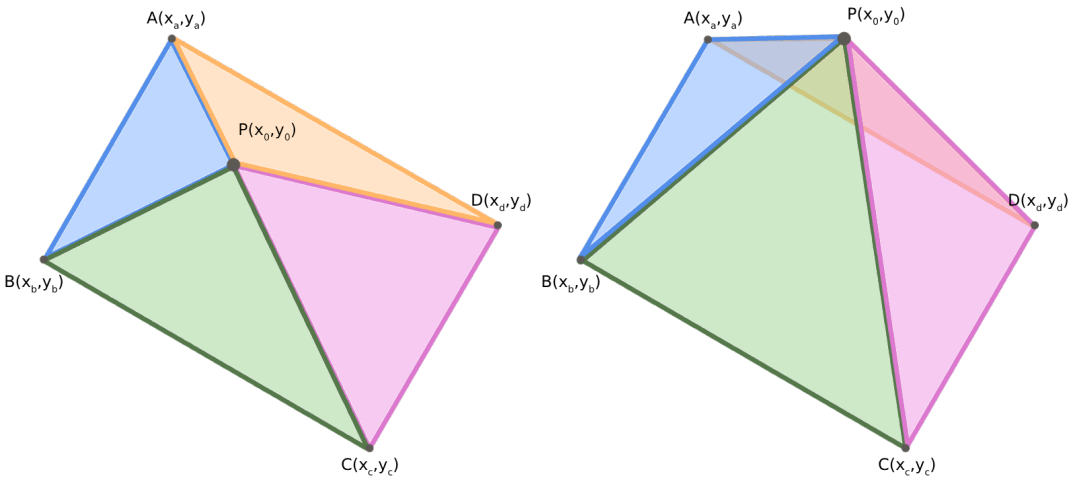
FIGURA 4.15: Problema de rotación

Por otra parte, el área de los triángulos formados por P y cada par de vértices, por ejemplo para $\widehat{ABP}$, tomamos los vectores $\overrightarrow{AB}$ y $\overrightarrow{AP}$ y construimos mediante el producto vectorial el área del romboide que es el doble del área del triángulo buscada:

$$\mathcal{A}_{\text{triángulo}_{\widehat{ABP}}} := \frac{1}{2}|\overrightarrow{AB} \times \overrightarrow{AP}|$$

En consecuencia, se tiene que

$$P \in \text{Rectángulo} \Leftrightarrow \mathcal{A}_{\text{rectángulo}} \leq \sum_i \mathcal{A}_{\text{triángulo}_i} \text{ con } i = \widehat{ABP}, \widehat{ADP}, \widehat{BCP}, \widehat{CDP}$$



(A) Si P no pertenece al rectángulo (B) Si P no pertenece al rectángulo

FIGURA 4.16: Cálculo de las áreas de los triángulos

Esta solución no está implementada, en su lugar se ha restringido la posibilidad de rotación en el mapa.

4.7. Fragmento perfil: Notificaciones por Telegram

La notificación y recepción de las alertas de emergencia cubren otra funcionalidad principal de la aplicación. Este apartado hace un inciso en las notificaciones trasmitidas a través de aplicaciones de mensajería, concretamente, Telegram.

En un primer momento, se estudió la idea de automatizar el envío de un mensaje directamente desde la cuenta del usuario a contactos prefijados. Sin embargo, esto no es posible ya que Telegram, por seguridad, tiene restringido el acceso de escritura en primera persona a otras aplicaciones.

Descartando este método, se opta por crear un bot de Telegram sencillo y a través del mismo enviar las notificaciones. Gracias a la API de Telegram [53], se ha podido desarrollar el bot en cuestión, en adelante *@spread_notifications_bot*, que evoluciona sobre un proyecto para la asignatura de ética, legislación y profesión que originalmente apareció con el fin de gestionar un calendario de eventos relacionados con la mujer en Madrid. Para consultar en detalle la documentación véase el artículo de la wiki [56].

Sin embargo, esto no es suficiente para poder establecer una conexión desde nuestra aplicación. La idea es notificar a grupos concretos de Telegram y para ello se necesita un identificador del mismo que debe introducirse manualmente. Lamentablemente, desde la aplicación no hay forma posible de saber cuál es ese identificador.

Se propone una forma alternativa de obtenerlo a través del bot:

1. Primero se ha de invitar al bot al grupo en cuestión, como se haría con cualquier otro usuario. Se asegura que el bot no está escuchando las conversaciones de los mismos, solo cuando se le invoca directamente mediante los comandos */<command-name>*.
2. El siguiente paso es preguntarle cuál es el identificador, mediante el comando */getchat*.
3. Finalmente, se introduce este identificador junto con el nombre que se desee para reconocerlo y se guarda.

FIGURA 4.17: Configurar un grupo de Telegram desde la aplicación

Este método no es intuitivo desde el punto de vista del usuario y se considera apropiado dejar explicados detalladamente estos pasos en la aplicación tal y como muestra la figura 4.17.

Estos identificadores quedarán guardados localmente en la aplicación y en el momento de solicitar ayuda se utilizarán para hacer una petición HTTP a la API de Telegram por cada uno de los grupos predefinidos:

```
https://api.telegram.org/bot<TOKEN>/sendMessage?chat_id=<CHAT_ID>&text=<TEXT>
```

Siendo **<TOKEN>** una constante que identifica el bot, **<CHAT_ID>** el identificador del grupo explicado previamente y **<TEXT>** el contenido del mensaje con el formato:

```
<NAME> necesita ayuda, está en <ADDRESS>,
<LINK_GOOGLE_MAPS>
```

FIGURA 4.18: Mensaje recibido en Telegram

Capítulo 5

Resultados y trabajo futuro

Tras la implementación de la aplicación, queremos destacar que se han cumplido con los objetivos fijados al inicio de la memoria, aunque algunos se han modificado debido a las limitaciones de la tecnología y al tiempo, y se dejan como trabajo futuro.

En la etapa final, se tomó la decisión de evaluar el modelo con usuarios y usuarias reales. Siguiendo el guión de tareas ya definido y tras haber probado el prototipo, se le envió un cuestionario de valoración anónimo para conseguir opiniones más abiertas. Este puede consultarse en el apéndice A.5.

Contando con un total de 10 personas entrevistadas, la evaluación general de la aplicación ha sido positiva, como se ve en la tabla 5.1. Sin embargo, aunque el envío de alertas a través de Telegram y entre dispositivos es gratuito, sí hemos notado cierto descontento al informar de la existencia de un coste para registrar información públicamente en la *blockchain*. Creemos que este problema podría resolverse en el futuro proponiendose una solución en el siguiente apartado.

Afirmación (1-Totalmente en desacuerdo ... 7-Totalmente de acuerdo)	Puntuación
La interfaz resulta intuitiva. Es fácil saber qué hacer.	6,11
La interfaz resulta agradable.	6,55
Los iconos son inesperados y cuesta comprenderlos.	2,22
La aplicación es útil.	6,55
Me fio de la información que publica la aplicación.	5,88
La aplicación cubre las tareas que esperaba que cumpliese.	6,66
La aplicación me hace sentir que pierdo privacidad.	1,88
Es fácil equivocarse al realizar tareas con la aplicación.	2,22

FIGURA 5.1: Puntuaciones de la aplicación sobre 7

Por otra parte, analizando los tiempos de respuesta, la aplicación resulta bastante intuitiva a la hora de realizar las tareas propuestas, pero se han detectado algunas funcionalidades concretas que parecen confusas. En particular, la configuración de los grupos de Telegram y la búsqueda de información relativa a la *wallet*, sería conveniente como trabajo en un futuro rediseñar la interfaz de estas dos tareas.

5.1. Trabajo futuro

En función de los resultados obtenidos y los problemas encontrados a lo largo del desarrollo del proyecto, se proponen un conjunto de ideas como líneas de trabajo.

5.1.1. Referidos a la tecnologías descentralizadas

- Permitir registrar una wallet ya existente. En lugar de obligar a crear una nueva al iniciar la aplicación por primera vez, estableciendo una conexión con una ya existente a través de su dirección y contraseña.

- Usar metatransacciones. Esta solución evita que las personas usuarias deban tener alguna noción de que es una criptomoneda ya que el pago de las transacciones se realizaría a través de una cuenta *raíz*. Esta cuenta intercepta las interacciones de las otras cuentas con el *smart contract* y paga las transacciones en su lugar. Habría que regular qué cuentas están haciendo estas transacciones y con qué frecuencia para que el saldo de la *raíz* no se agote sin necesidad.

  Esta cuenta raíz podría autosustentarse con publicidad.

- Incluir un método para poder mostrar el apoyo sobre alguno de los marcadores.

- Prevenir bloqueos de la cuenta. Si una transacción consume más *ether* del determinado como límite, la transacción nunca llegará a ejecutarse por completo, quedando en un estado *Pending* que bloquea tanto la transacción actual, como las posteriores.

  Primera propuesta de solución: abrir una nueva cuenta. Hay que tener en cuenta que en caso de cuentas de desarrollo no hay problema porque son *ether* que no tienen valor, pero si el dinero invertido es real, se pierde.

  Segunda propuesta de solución: lanzar una transacción con el mismo *nonce* pero cuyo *gas limit* sea 10 % superior al de la anterior transacción, lo que permite *sobreescribir* la última transacción.

- Permitir borrar carteles cuando ya no existan, modificando la interfaz para poder solicitar un borrado de datos. Primero habría que comprobar que la identidad de quien quiere borrarlo y de quien lo publicó son la misma. Si lo son, habría que crear un nuevo método que permita editar el contenido y

actualizarlo para todos los dispositivos. Si no, se debe enviar una notificación al dispositivo que publicó el contenido e informarle de que alguien quiere eliminarlo junto con el motivo y este deberá actuar bajo su criterio.

5.1.2. Referidos a la aplicación Android

- Convertir el código a *Kotlin* [26]. Cierto es que muchas de las aplicaciones que se encuentran activas están desarrolladas en Java. Sin embargo, *Kotlin* es un lenguaje bastante consolidado ya entre los desarrolladores de Android por tener una sintaxis más limpia.

- Inyectar dependencias. Para reducir el ruido de crear nuevos objetos de forma repetitiva al comienzo de cada clase, se puede abstraer un encabezado común y se inyecta mediante la librería *Dagger 2* [7]. Explicando brevemente cómo funciona, necesitaríamos añadir dos nuevas estructuras al proyecto:

  Módulo. Es una clase que se encarga de construir y proveer los objetos que se quieren inyectar.

  Componente. Es la interfaz usada para generar el inyector, es decir, se encarga de conectar los módulos con las clases que los requieren. En este proyecto, lo ideal sería inyectar los presenter en los fragments y las activities.

- Acotar el radio de alcance de las alertas. Como se ha explicado, actualmente las distancias entre dispositivos se calculan en función del código postal. Se propone calcular un nuevo radio estimado en función de la precisión decimal de la latitud-longitud. Por ejemplo, partimos de las coordenadas $(40{,}00000000001111, -3{,}70000000000111)$ digamos que los usuarios cercanos están en torno a los 12 primeros decimales de los 14 originales. Entonces, las alertas se enviarían a los dispositivos con las siguientes coordenadas:

```
(40.00000000000100, -3.70000000000100)
(40.00000000000101, -3.70000000000100)
(40.00000000000102, -3.70000000000100)
.
.
.
(40.00000000001111, -3.70000000001111)
```

- Permitir la búsqueda de una localización concreta y generar una ruta óptima por la que se pueda evitar el mayor número de zonas consideradas peligrosas.

- Respecto a la determinación de si un punto pertenece o no a un polígono, en lugar de comparar con todos los sitios (orden cuadrático), sería más eficiente subdividir el mapa en cuadrículas, usarla como un *hashmap<cuadrícula, lista_de_puntos>* y reducir los cálculos a los sitios pertenecientes a la misma cuadrícula que el punto dado y quizá, las adyacentes.
- Permitir más filtros en el mapa, por ejemplo por franjas horarias o mostrar solo los elementos de un tipo : *alertas*, *carteles*.

Apéndice A

Entrevistas en profundidad

A.1. Guión de las entrevistas

Preguntas orientadas a actitudes

- ¿Crees que hay sitios en Madrid por los que es peligroso ir si no vas en grupo? ¿Cuáles? ¿Por qué? ¿Por qué sabes que no es un sitio seguro (o cómo te has enterado)? ¿Es peligroso a cualquier hora del día? ¿En qué franja horaria?
- ¿Crees que hay sitios tu ciudad (en caso de no ser Madrid) por los que es peligroso ir si no vas en grupo? ¿Cuáles? ¿Por qué? ¿Por qué sabes que no es un sitio seguro (o cómo te has enterado)? ¿Es peligroso a cualquier hora del día? ¿En qué franja horaria?
- Si te mudas a una nueva ciudad que no conoces, ¿cómo te enterarías? (o te has enterado)
- ¿El género influye en las zonas elegidas? Si pertenecieses al opuesto, ¿cambiarían las zonas en las que pensabas? Explica en qué medida.

Preguntas orientadas a flujos de trabajo

- ¿Te has sentido alguna vez insegura en la calle?

  En caso afirmativo, ¿por qué? ¿cómo has reaccionado?

- Cuando sales de noche, ¿sueles plantearte si vas a pasar por algún sitio conflictivo? ¿cómo crees que podrías conseguir esa información? (o cómo la consigues)

  ¿Influye la franja horaria? ¿Cómo?

- Imagínate que sales y un chico te molesta: te dice cosas y no te deja en paz. ¿Qué harías?
- Sales y ves que una chica está teniendo problemas o que necesita ayuda. ¿Qué haces?

- ¿Has visto alguna vez algún anuncio o cartel que te haya parecido ofensivo respecto a tu género? ¿Has hecho algo al respecto?

  En caso afirmativo, ¿qué?

  En caso negativo, ¿por qué no? ¿quién crees podría hacerlo?

- ¿Has estado alguna vez en una situación de emergencia en la calle?

- ¿Sueles tener activado el GPS en el teléfono?

  En caso afirmativo, ¿por qué?

  En caso negativo, ¿en qué situaciones lo activas?

Preguntas orientadas a objetivos

- Tienes que irte o mudarte a un sitio nuevo que no conoces, ¿a qué sistema de búsqueda acudes para orientarte? [guía, internet, preguntas en persona...]

- En caso de ser un medio digital, ¿prefieres hacerlo desde el móvil o desde el ordenador?

- En caso de estar en una situación de emergencia en la calle, ¿qué es lo primero que harías?

  En caso de avisar a alguien, ¿quién sería? ¿cómo contactarías con ellos? ¿y si no están cerca?

Preguntas orientadas al sistema

- ¿Estarías cómoda compartiendo tu ubicación de forma anónima si con esto puedes ayudar y ser ayudada de forma más rápida?

- Suponiendo que puedes saber si alguien cerca de ti (geográficamente) necesita ayuda en el momento, ¿acudirías aunque no le conozcas?

- ¿Crees que recolectar muchos carteles de contenido ofensivo ayudaría a la eliminación de los mismos?

- ¿Cuando sales a la calle llevas el teléfono contigo siempre? ¿En qué ocasiones no lo llevas?

- Si la ropa que llevas no tiene bolsillos, ¿qué haces con el móvil?

- ¿Conoces los puntos violeta? (Si no saben se explica) ¿Qué opinas de ellos?

  En caso de conocerlos, ¿Sabes dónde se ubican? ¿Cómo lo sabes?

A.2. Guión de las pruebas de los prototipos

Introducción Esta app te permite pedir ayuda en caso de emergencia a gente que esté cerca por defecto pero también puede avisar a contactos que predefinas y crear quejas colectivas ante anuncios que consideres ofensivos.

Añadir registros

- Sales una tarde y conoces a alguien, estás a gusto pero oye, te cansas y te apetece irte, se lo dices y ya no es tan amable como antes, te dice que te quedes, pero no quieres y volverte sol@ a casa no parece muy seguro, ¿cómo pedirías ayuda?

  · Además de pedir ayuda a gente que esté cerca te gustaría avisar a tu herman@ que siempre os cuidáis, ¿qué cambio harías en la app para que le llegue el aviso también?

  · Parece que nadie aparece y sigues ahí, te gustaría ampliar el alcance de la llamada aunque la ayuda esté un poco más lejos, ¿cómo lo harías?

- Pasas junto a un bar y ves que en la puerta hay un cartel que no te parece apropiado y te gustaría compartirlo con más gente para hacer una denuncia colectiva, ¿cómo lo harías con la app?

  · Le hiciste una foto al cartel pero no tenías datos así que esperaste a subirlo cuando llegaste a casa, ¿qué pasos seguiste?

Consultar registros

- Algunas veces has comentado y compartido con tus amigas momentos incómodos que han pasado por la calle y te preguntas si también le ocurre a otras personas, con esta aplicación puedes consultarlo en tu zona, ¿cómo encontrarías esta información en la app?

  · Siempre has oído que hay zonas con peor fama que otras en tu ciudad, te gustaría ver si los rumores son ciertos, con la app puedes seccionar el mapa por zonas y tener una vista más genérica, ¿cómo lo harías?

- Supongamos ahora no tienes interés en la zona donde se producen las alertas si no en las más recientes, ¿cómo las consultarías?

  · Haciendo scroll en los registros, ves que uno es un cartel de una fiesta, igual al que has visto esta mañana en la marquesina del bus, te gustaría apoyar a la queja, ¿cómo lo harías?

Preguntas abiertas y otras

- ¿Qué información compartirías con quienes van a acudir en tu ayuda? ¿Cómo modificarías esta información?
- Esta aplicación funciona sobre criptomonedas, que se usan para la gestión de las consultas. Aunque en principio es gratuito (pero necesitas meterle ether) :
  - ¿Cuál es el identificador de tu cuenta?
  - ¿Cuánto saldo te queda?
- ¿Has echado en falta alguna funcionalidad en la app?

A.3. Modelado de la persona principal

A.4. Evaluaciones heurísticas

A.5. Encuesta de satisfacción

Liza Gómez

Persona media

Información personal

Descripción : **Liza Gómez**, *24 años*, Madrid, España.
Estudiante del último año de Biología en la Universidad Autónoma de Madrid, a la espera de que acepten la solicitud para cursar el máster en Genética y Biología Celular. Vive en la Latina, en un piso compartido con Andrea y Jorge, dos compañeros de clase y Miku, su gato.

Ocupación : **Estudiante**.
Aparte de la universidad, tiene un trabajo a tiempo parcial colaborando con la asociación ABRIGA, acogiendo animales abandonados.

Objetivos, deseos y motivaciones

Liza vive en un barrio multicultural y nunca se cansa de descubrir sitios nuevos, incluso dentro de su zona. Sin embargo, no se siente cómoda volviendo a casa sola, por lo que muchas veces depende de si sus compañeros de piso quieren y pueden salir.

Conocimientos y habilidades

Liza maneja con soltura aplicaciones de su móvil y ordenador sin conocer los detalles técnicos. Es muy activa en redes sociales, no obstante, hay cierta información que prefiere no compartir.

Entorno de la persona

Como trabajadora a tiempo parcial y pendiente de finalizar su carrera, sus ocupaciones entre semana están cubiertas. Ésto deja reducido su tiempo libre al fin de semana, que lo dedica a salir con sus amigos, leer manga y salir a correr por Madrid Río. Como esto se reduce a dos días, Liza no suele planificar sus salidas, sino que se apunta a los planes de sus amigos aunque no siempre sean cerca. Cuando no tiene con quien volver, prefiere coger un Uber, se gasta más dinero pero se evita pasar un mal rato.

Detalles relativos a reacciones ante el acoso

Ante las continuas noticias sobre acoso y violencia machista que aparecen en los medios a diario, Liza está preocupada cuando sale a correr por las tardes, aunque sea por ciudad. Nunca le ha pasado nada, pero si se ha llevado algún susto o comentario innecesario. Estaría más tranquila si supiese que alguien puede acudir a ayudarla rápido en caso de necesitarlo.

PLANTILLA DE EVALUACIÓN HEURÍSTICA

DATOS PERSONALES Y DE EVALUACIÓN
Nombre: David Llop Vila
Sexo: V
Fecha de evaluación: 12/02/19
Hora inicio evaluación: 13:20
Hora fin evaluación: 13:56

PROBLEMAS DETECTADOS
(Copiar y pegar si fuera necesario añadir más puntos)

- **Nombre:**
  - Breve descripción: SOS es demasiado asustadizo.
  - Heurística violada: 2
  - Dónde se ha encontrado: Pantalla principal
  - Grado de severidad: 9

- **Nombre:**
  - Breve descripción:Vestuario no se entiende.
  - Heurística violada: 2
  - Dónde se ha encontrado: Pantalla perfil
  - Grado de severidad: 9

- **Nombre:**
  - Breve descripción: Cuando das a ayuda, mantener el control.
  - Heurística violada: 3
  - Dónde se ha encontrado: Pantalla principal
  - Grado de severidad: 7

- **Nombre:**
  - Breve descripción: En avisos, falta el botón +.
  - Heurística violada: 3
  - Dónde se ha encontrado: Pantalla principal
  - Grado de severidad: 7

- **Nombre:**
  - Breve descripción: Al enviar aviso, pedir confirmación., como prevención de errores.
  - Heurística violada: 4
  - Dónde se ha encontrado: Añadir aviso
  - Grado de severidad: 1

- **Nombre:**
  - Breve descripción: El botón de zonas no da información de qué se trata.
  - Heurística violada: 2
  - Dónde se ha encontrado: Pantalla mapa
  - Grado de severidad: 5

PLANTILLA DE EVALUACIÓN HEURÍSTICA

DATOS PERSONALES Y DE EVALUACIÓN
Nombre: Raquel Hervás
Sexo: M
Fecha de evaluación: 12/02/2019
Hora inicio evaluación: 12:25
Hora fin evaluación:

IMPRESIONES POSITIVAS DE LA APLICACIÓN

- **Visualmente está muy bien y es muy intuitiva de usar**
- **Sigue bastante bien las directrices de Material Design tanto en colores como en controles**
- **La parte del mapa visualiza muy bien la información que se quiere mostrar**
- **La funcionalidad principal, la del SOS, está bien en la pantalla principal y con un botón grande**

PROBLEMAS DETECTADOS
(Copiar y pegar si fuera necesario añadir más puntos)

- **Nombre: Feedback SOS**
    - Breve descripción: Cuando se pulsa el botón de SOS no se muestra ningún feedback, y el usuario podría dudar de si ha ocurrido algo.
    - Heurística violada: N01
    - Dónde se ha encontrado: En la pestaña de Home
    - Grado de severidad: 10

- **Nombre: Activación del SOS**
    - Breve descripción: Aunque es la funcionalidad principal y se quiere que sea rápido de activar, si hay un error de pulsación las consecuencias son graves (se preocupa todo el mundo), así que se debería estudiar que la acción fuera menos propensa a errores.
    - Heurística violada: N05
    - Dónde se ha encontrado:En la pestaña de Home
    - Grado de severidad:5

- **Nombre: Ajustes divididos en dos partes (1)**
    - Breve descripción: Hay ajustes que se hacen en la pestaña de perfil, y otros que se hacen en los ajustes. No acabo de entender por qué no está todo junto, ya que al final son “ajustes” tanto del usuario como de aplicación.
    - Heurística violada: N06 (porque el usuario puede dudar al ir a cambiar la configuración sobre a dónde tiene que ir)
    - Dónde se ha encontrado: Pestañas de Perfil y Ajustes
    - Grado de severidad: 7

- **Nombre: Ajustes divididos en dos partes (2)**
    - Breve descripción: Mismo que el anterior pero viola también la heurística de eficiencia porque si el usuario tiene que estar buscando tardará más en activar las opciones.

    - Heurística violada: N07
    - Dónde se ha encontrado:Pestañas de Perfil y Ajustes
    - Grado de severidad:3 (por temas de eficiencia lo veo menos grave).

- **Nombre: Mapa para registrar evento de otro lugar**
    - Breve descripción: Cuando un usuario abre el mapa suele esperar que se muestre o bien la situación general en distintas zonas del mapa, o que se active alguna función en el punto en el que está. Me queda raro que también en el mapa se puedan dar de alta eventos que han ocurrido en otro lugar distinto del que estoy.
    - Heurística violada: N04
    - Dónde se ha encontrado:En la opción del Mapa
    - Grado de severidad:6

- **Nombre: Avisos vs. Mapa**
    - Breve descripción: La información de los avisos existentes está de alguna manera dividida en dos: en el mapa se ven las zonas más peligrosas, y en los avisos se ve como lista más especializada. ¿Se podría de alguna manera conectar las dos?
    - Heurística violada: N07
    - Dónde se ha encontrado: Pestañas de Avisos y Mapa
    - Grado de severidad: 7

- **Nombre: Filtrado y ordenación de los Avisos**
    - Breve descripción: En los avisos, aunque están ordenados por recientes, sería útil poder ordenarlos por otros campos o incluso filtrarlos por zonas.
    - Heurística violada: N07
    - Dónde se ha encontrado: Pestaña de Avisos
    - Grado de severidad: 8

- **Nombre: Información de los Avisos (1)**
    - Breve descripción:En los Avisos, para cosas como lo de los carteles o para saber si alguien ya los ha reportado, estaría bien que hubiera una foto o algo así que los identificara rápidamente.
    - Heurística violada:N01
    - Dónde se ha encontrado:Pestaña de Avisos
    - Grado de severidad:6

- **Nombre: Información de los Avisos (2)**
    - Breve descripción:En los Avisos, para poder mirar si alguien ya ha reportado algo, tal y como está se viola la heurística de eficiencia porque sería muy pesado ir mirándolo todo.
    - Heurística violada:N07
    - Dónde se ha encontrado:Pestaña de Avisos
    - Grado de severidad:6

- **Nombre: Info del Wallet**

    - Breve descripción:La información sobre el wallet que hay en los Ajustes es ilegible para cualquier usuario que no esté familiarizado con el tema.
    - Heurística violada:N10
    - Dónde se ha encontrado:Pestaña de Ajustes
    - Grado de severidad: 9

- **Nombre: Consulta de Avisos**
    - Breve descripción: Aunque en la aplicación se llaman Avisos, en la evaluación se han llamado registros. Ser consistentes, buscando una palabra que se use en el mundo normal. Avisos puede sugerir notificaciones de la aplicación, por ejemplo.
    - Heurística violada: N02
    - Dónde se ha encontrado:Pestaña de Avisos
    - Grado de severidad: 3

Encuesta sobre SpreadApp

Si has recibido esta encuesta porque has realizado el test de usabilidad de la aplicación, asi que en primer lugar, ¡muchas gracias! Queremos pedirte un último favor completando de forma anónima como de acuerdo estas con las siguientes afirmaciones.

1. **La interfaz resulta intuitiva. Es fácil saber qué hacer.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

2. **La interfaz resulta agradable.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

3. **Los iconos son inesperados y cuesta comprenderlos.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

4. **La aplicación es útil.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

5. **Me fio de la información que publica la aplicación**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

6. **La aplicación cubre las tareas que esperaba que cumpliese.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

7. **La aplicación me hace sentir que pierdo privacidad.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

8. **Es fácil equivocarse al realizar tareas con la aplicación.**
   Marca solo un óvalo.

	1	2	3	4	5	6	7	
Totalmente en desacuerdo	◯	◯	◯	◯	◯	◯	◯	Totalmente de acuerdo

9. **Si has tenido alguna dificultad al realizar alguna de las tareas, cuéntanos :**

_____

10. **¿Le añadirías algo más? En caso afirmativo, ¿qué?**

_____

_____

_____

_____

_____

Apéndice B

Código de los *smart contract*

B.1. Fragmento de código para alertas

```
pragma solidity ^0.5.1;
contract AlertContract {
    struct Alert {
        string title;
        string description;
        string latitude;
        string longitud;
        string date;
    }
    Alert[] public listAlerts;

    function addAlert(string memory t, string memory d, string memory lat,
                string memory longi, string memory date) public {
        listAlerts.push(Alert(t,d,lat,longi, date));
    }

    function getAlertsCount() public view returns(uint) {
        return listAlerts.length;
    }

    function getAlertByIndex(uint index) public view returns(string memory,
                string memory , string memory, string memory, string memory) {
        return (listAlerts[index].title, listAlerts[index].description,
                listAlerts[index].latitude, listAlerts[index].longitud,
                listAlerts[index].date);
    }
}
```

B.2. Fragmento de código para carteles

```
pragma solidity ^0.5.1;
contract PosterContract {
    string[] public listPosters;

    function addPoster(string memory hashPoster) public {
        listPosters.push(hashPoster);
    }

    function getPostersCount() public view returns(uint) {
        return listPosters.length;
    }

    function getPosterByIndex(uint index) public view
                                    returns(string memory) {
        return listPosters[index];
    }
}
```

Glosario de Abreviaturas

API	**A**pplication **P**rogramming **I**nterface
BSON	**B**inary J**SON**
DGO	**D**iseño **G**uiado por **O**bjetivos
INE	**I**nstituto **N**acional de **E**stadística
GPS	**G**lobal **P**ositioning **S**ystem
IPFS	**I**nter**P**lanetary **F**ile **S**ystem
JSON	**J**avaScript **O**bject **N**otation
MVP	**M**odel **V**iew **P**resenter
P2P	**P**eer-to-**P**eer
Patricia	**P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric
POJO	**P**lain **O**ld **J**ava **O**bject
RLP	**R**ecursive **L**ength **P**refix
UTM	**U**niversal **T**ransverse **M**ercator
XML	E**X**tensible **M**arkup **L**anguage

Bibliografía

[1] MobileSoftware AS. *BSafe – Never walk alone*. en-GB. URL: https://getbsafe.com/.

[2] Narayanan Arvind Bonneau Joseph. *Bitcoin and Cryptocurrency Technologies : A Comprehensive Introduction*. Princeton University Press.

[3] *Build a Responsive UI with ConstraintLayout*. URL: https://developer.android.com/training/constraint-layout/.

[4] *Building a smart contract on Ethereum*. URL: https://www.ethereum.org/greeter.

[5] *Circle of 6*. URL: https://www.circleof6app.com/.

[6] Alan Cooper. *About Face: The Essentials of User Interface Design*. en.

[7] *Dagger documentation*. URL: https://google.github.io/dagger/.

[8] Chris Dannen. *Introducing Ethereum and Solidity*. en. 2017.

[9] Primavera De Filippi y Samer Hassan. *Blockchain Technology as a Regulatory Technology: From Code is Law to Law is Code*. en. Inf. téc. Rochester, NY: Social Science Research Network. URL: https://papers.ssrn.com/abstract=3097430.

[10] *Decide Madrid : crea y apoya propuestas para mejorar Madrid*. es. URL: https://decide.madrid.es/.

[11] *Empower Women*. URL: https://www.empowerwomen.org/en.

[12] *ETH Gas Station*. URL: https://ethgasstation.info/index.php.

[13] *Ethereum Classic Technical Reference (BETA) — Ethereum Classic Technical Reference 0.1 documentation*. URL: https://ethereum-classic-guide.readthedocs.io/en/latest/.

[14] *European Institute for Gender Equality (EIGE)*. en. Jun. de 2016. URL: https://europa.eu/european-union/about-eu/agencies/eige_en.

[15] *Firebase API Reference*. URL: https://firebase.google.com/docs/reference/.

[16] *Get the App*. en-US. URL: https://www.ihollaback.org/take-action/get-app/.

[17] *Gravit Designer Gravit Cloud*. URL: https://gravit.io/.

[18] *Guide on Ethereum Wallets: Mobile, Web, Desktop, Hardware*. URL: https://cointelegraph.com/ethereum-for-beginners/ethereum-wallets#your-own-or-thirdparty-ethereum-wallet.

[19] *Guide to Ethereum: What is Gas, Gas Limit and Gas Price?* en-US. Feb. de 2018. URL: https://masterthecrypto.com/ethereum-what-is-gas-gas-limit-gas-price/.

[20] *Home - Pencil Project*. URL: https://pencil.evolus.vn/.

[21] *Hot wallet vs Cold wallet*. URL: https://blog.liquid.com/hot-wallet-vs-cold-wallet-how-should-you-store-crypto.

[22] *Infura - Scalable Blockchain Infrastructure*. en. URL: https://infura.io.

[23] *Instituto Nacional de Estadistica. (Spanish Statistical Office)*. URL: https://www.ine.es/.

[24] *International Knowledge Network of Women in Politics*. URL: http://iknowpolitics.org/en.

[25] *IPFS Documentation*. URL: https://docs.ipfs.io/.

[26] *Kotlin documentation for Android*. URL: https://kotlinlang.org/docs/reference/android-overview.html.

[27] *M4 Estaciones - Madrid*. URL: http://datos.crtm.es/.

[28] *Mapbox - Navigation SDK for Android*. URL: https://docs.mapbox.com/android/navigation/overview/.

[29] *Marvel - The design platform for digital products*. URL: https://www.marvelapp.com/.

[30] *Material Design*. URL: https://material.io/design/.

[31] *Material Design for Android*. URL: https://developer.android.com/guide/topics/ui/look-and-feel/.

[32] *Material Pallete for Android*. URL: https://www.materialpalette.com/.

[33] *Meet Android Studio | Android Developers*. URL: https://developer.android.com/studio/intro.

[34] *MetaMask*. URL: https://metamask.io/.

[35] *Model View Presenter(MVP) in Android with a simple demo project*. URL: https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42.

[36] *Model-view-presenter pattern in Android*. URL: https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42.

[37] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. en. URL: https://bitcoin.org/bitcoin.pdf.

[38] *Numpy documentation - Voronoi diagram*. URL: https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.spatial.Voronoi.html.

[39] David Olivares. *No More, una app contra la violencia de género, representará a España en el campeonato europeo de miniempresas*. es. Jun. de 2018. URL: https://www.muypymes.com/2018/06/25/no-more-una-app-contra-la-violencia-de-genero-representara-a-espana-en-el-campeonato-europeo-de-miniempresas.

[40] *Overleaf - Online LaTeX Editor*. URL: https://www.overleaf.com/.

[41] *Photopea - Online image editor*. URL: https://www.photopea.com/.

[42] *Remix - Solidity IDE*. URL: https://ethereum.github.io/browser-solidity/.

[43] Mark Richards. *Software Architecture Patterns*. en. Cap. 1. URL: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/.

[44] *Rinkeby - Ethereum testnet*. URL: https://www.rinkeby.io/.

[45] Yaritza Rivera Clemente. «Aplicación contra violencia de género en San Juan». En: *El Vocero* (). URL: https://www.elvocero.com/gobierno/aplicaci-n-contra-violencia-de-g-nero-en-san-juan/article_283a04cc-fdb2-11e8-a69b-d79ad4204f3d.html.

[46] *RxJava – Reactive Extensions for the JVM*. URL: https://github.com/ReactiveX/RxJava.

[47] *Safe365 - Localizador familiar para cuidar de los tuyos*. URL: https://safe365.com/es.

[48] Safecity. *Safecity*. en-US. URL: https://safecity.in/.

[49] Sami Salkosuo. *IBM: Coordinate conversions made easy*. en. URL: http://www.ibm.com/developerworks/library/j-coordconvert/index.html.

[50] Nicolas Schapeler. *An introduction to Ethereum Development on Android using Web3j and Infura*. Ene. de 2019. URL: https://medium.com/datadriveninvestor/an-introduction-to-ethereum-development-on-android-using-web3j-and-infura-763940719997.

[51] *Service*. en. URL: https://developer.android.com/reference/android/app/Service.

[52] *Services overview*. en. URL: https://developer.android.com/guide/components/services.

[53] *Telegram APIs*. URL: https://core.telegram.org/.

[54] *The Ethereum Wiki. Contribute to ethereum/wiki development by creating an account on GitHub*. Abr. de 2019. URL: https://github.com/ethereum/wiki.

[55] *The Reactive Manifesto 2.0*. URL: https://www.reactivemanifesto.org/pdf/the-reactive-manifesto-2.0.pdf.

[56] *Trabajo: Bot de Telegram para notificar eventos de carácter feminista - FdIwiki ELP*. URL: http://wikis.fdi.ucm.es/ELP/Trabajo:_Bot_de_Telegram_para_notificar_eventos_de_car%C3%A1cter_feminista.

[57] Phan Sn T. *Data structure in Ethereum | Episode 1: Recursive Length Prefix (RLP) Encoding/Decoding*. Ene. de 2018. URL: https://medium.com/coinmonks/data-structure-in-ethereum-episode-1-recursive-length-prefix-rlp-encoding-decoding-d1016832f919.

[58] Phan Sn T. *Data structure in Ethereum | Episode 2: Radix trie and Merkle trie*. URL: https://medium.com/coinmonks/data-structure-in-ethereum-episode-2-radix-trie-and-merkle-trie-d941d0bfd69a.

[59] Phan Sn T. *Data structure in Ethereum | Episode 3: Patricia trie*. Feb. de 2018. URL: https://medium.com/coinmonks/data-structure-in-ethereum-episode-3-patricia-trie-b7b0ccddd32f.

[60] *Unsafe in the city | Plan International*. URL: https://plan-international.org/unsafe-city.

[61] *Ushahidi*. en-US. URL: https://www.ushahidi.com/.

[62] Isabel Valdés. «Un botón y dos clics contra la violencia machista». En: *El País* (). URL: https://elpais.com/sociedad/2019/05/04/actualidad/1556995565_877742.html.

[63] *web3j — web3j 4.1.0 documentation*. URL: https://web3j.readthedocs.io/en/latest/.

[64] Dr Gavin Wood. «Ethereum: A secure decentrlised generalised transaction ledger». en. En: (). URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[65] *Zotero - Your personal research assistant*. URL: https://www.zotero.org/.