



Universidad Politécnica  
de Madrid

**Escuela Técnica Superior de  
Ingenieros Informáticos**



Grado en Ingeniería Informática

Trabajo Fin de Grado

**Diseño e Implementación de un SDK  
Android para Facilitar la Interacción de  
Aplicaciones Móviles con una Blockchain**

Autor: Jorge Sol González  
Tutor(a): Francisco Javier Soriano Camino

Madrid, 06-2021

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*  
*Grado en Ingeniería Informática*

**Título:** Diseño e Implementación de un SDK Android para Facilitar la Interacción de Aplicaciones Móviles con una Blockchain

**Autor:** Jorge Sol González  
**Tutor:** Francisco Javier Soriano Camino  
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software  
ETSI Informáticos  
Universidad Politécnica de Madrid

## Resumen

Durante los últimos dos años he tenido la oportunidad de trabajar en la Cátedra Inetum investigando y estudiando todo lo relacionado a la tecnología blockchain. Dentro de la iniciativa Campus Blockchain de esta Cátedra, surgió un proyecto llamado “Estublock” con el cual se pretende resolver un problema más común de lo esperable. Es habitual que, como parte de los procesos de evaluación, se consideren actividades que requieren de presencialidad (laboratorios, participación activa en el aula, etc.). La utilización de “listas de asistencia” presenta múltiples problemas, incluidos los de retraso en el comienzo de la actividad por el tiempo que requiere ese tipo de control, posible pérdida de los listados, sobre todo cuando se utiliza el papel como soporte, posibilidad de que se suplanen identidades, etc. El problema radica en que no hay una forma común, confiable y transparente de validar la asistencia a dichas pruebas. Esto es extrapolable a cualquier tipo de evento que requiera que se valide la asistencia (obtención de créditos de libre elección, etc).

Para solucionar este problema se propone usar blockchain. Esta es una tecnología permite registrar transacciones de manera distribuida. Como un libro contable distribuido. Transacciones monetarias, transacciones con información o datos como el nombre de una persona, la asistencia a un evento, un billete de avión, una entrada de cine, etc. La red blockchain elegida para el trabajo es la red de Ethereum, debido a que ofrece la posibilidad de ejecutar Smart Contracts en ella. Los smart contracts son básicamente código que se ejecuta de manera automática ante una llamada a una de sus funciones. En el presente existen varios proyectos que aprovechan la red de Ethereum, como *Guts*, *LifeID* y *Voatz*.

Para desarrollar “Estublock” se ha utilizado Android, uno de los sistemas operativos más utilizados en el presente y para facilitar la comunicación con la base de datos y algunos procesos de la comunicación con la red blockchain, se ha desarrollado una API RESTful. El desarrollo de la aplicación y del SDK han supuesto un gran reto, pues nunca antes había trabajado en este campo. Se ha diseñado con Marvelapp un diseño de pantallas con la intención de que sirva como plantilla para después realizar en la aplicación móvil, con código XML, el diseño final de la aplicación. Junto a Marvelapp, se han diseñado unos diagramas y unos casos de uso para visualizar el objetivo de la aplicación. Aunque los diseños de Marvelapp han sido de gran utilidad, muchas de las pantallas han sufrido cambios según se iban programando y según el proyecto iba creciendo. Con respecto al código, la aplicación se ha desarrollado utilizando Java y múltiples librerías para facilitar las llamadas a las APIs, así como llamadas a la red blockchain. Las llamadas a la red blockchain se han implementado en un SDK a parte. Así, se podrá compartir en el repositorio Maven Central para que otros desarrolladores de aplicaciones móvil puedan utilizarlo. Al trabajar con Blockchain hay un concepto importante que se ha estudiado a lo largo del desarrollo, los wallets y los keystores. Piezas claves en las que los usuarios guardan a buen recaudo sus credenciales para firmar transacciones en la red blockchain. Además se han realizado pruebas a la aplicación utilizando la técnica *White Box Testing*.

La tecnología blockchain crece sin cesar, cada día surgen nuevos proyectos y también mejoras en el sistema, y aunque puede estar algo inmadura, es sin duda alguna el futuro. A “Estublock” le queda un largo camino por recorrer, se ha logrado terminar una primera versión viable, la cual puede probarse actualmente para validar la asistencia de alumnos en alguna prueba académica o taller. Sin embargo, hay que seguir desarrollándola, añadiendo funcionalidades para el usuario, tanto visuales como técnicas. Mejorar la documentación para que futuros desarrolladores puedan seguir con el trabajo hecho y preparar algunos tests para desarrollar con más seguridad y evitar fallos. Sin duda alguna, este proyecto me ha enseñado mucho, y me ha hecho ver lo mucho que queda por aprender y descubrir.

**Palabras Clave:** Blockchain, Smart Contract, Ethereum, Android, SDK, Web3j, Quorum, Java, Móvil, Librerías, Wallet, Keystore, etc.



## Abstract

During the last two years, I have had the opportunity to work in the *Cátedra Inetum* researching and studying everything related to blockchain technology. Within the Campus Blockchain initiative of this *Cátedra*, a project called “Estublock” emerged, with which is intended to solve a problem more common than expected. It is common that, as part of the evaluation processes, activities that require attendance (laboratories, active participation in the classroom, etc.) are considered. The use of “attendance lists” presents multiple problems, including delays in the start of the activity due to the time required for this type of control, possible loss of the lists, especially when a paper is used as a support, the possibility of identity theft, etc. The problem lies in the fact that there is no common, reliable, and transparent way of validating attendance at such tests. This can be extrapolated to any type of event that requires validation of attendance (obtaining elective credits, etc.).

To solve this problem it is proposed to use blockchain. This is a technology allows recording transactions in a distributed manner. Like a distributed ledger. Monetary transactions, transactions with information or data such as a person’s name, attendance to an event, a plane ticket, a movie ticket, etc. The blockchain network chosen for the work is the Ethereum network because it offers the possibility of executing Smart Contracts on it. Smart contracts are code that is executed automatically upon a call to one of its functions. At present, several projects leverage the Ethereum network, such as *Guts*, *LifeID* and *Voatz*.

To develop “Estublock” Android has been used, one of the most used operating systems at present and to facilitate the communication with the database and some processes of the communication with the blockchain network, a RESTful API has been developed. The development of the application and the SDK has been a great challenge, as I had never worked in this field before. A screen design has been designed with Marvelapp to serve as a template to later make in the mobile application, with XML code, the final design of the application. Together with Marvelapp, some diagrams and use cases have been designed to visualize the objective of the application. Although the Marvelapp designs have been very useful, many of the screens have changed as they were programmed and as the project grew. Concerning the code, the application has been developed using Java and multiple libraries to facilitate API calls, as well as calls to the blockchain network. The calls to the blockchain network have been implemented in a separate SDK. Thus, it can be shared in the Maven Central repository so that other mobile application developers can use it. When working with Blockchain there is an important concept that has been studied throughout the development, wallets, and keystores. Key pieces in which users keep their credentials safe to sign transactions in the blockchain network. In addition, the application has been tested using the *White Box Testing* technique.

Blockchain technology is growing steadily, new projects and also improvements in the system are emerging every day, and although it may be somewhat immature, it is undoubtedly the future. “Estublock” still has a long way to go, a first viable version has been completed, which can currently be tested to validate the attendance of students in an academic test or workshop. However, it is necessary to continue developing it, adding functionalities for the user, both visual and technical. Improve the documentation so that future developer can continue with the work done and prepare some tests to develop with more security and avoid failures. Undoubtedly, this project has taught me a lot and has made me see how much there is to learn and discover.

**Keywords:** Blockchain, Smart Contract, Ethereum, Android, SDK, Web3j, Quorum, Java, Mobile, Libraries, Wallet, Keystore, etc.



# Agradecimientos

---

Mil gracias a todos mis amigos, especialmente a los de la universidad, con los que he convivido en las buenas y en las malas durante la carrera. Gracias a Ferrero, Anto, Younes, Paula, Carlos, mis dos Diegos, Gaspar y Alex, Kalili y Borja, porque aunque no se lo crean, todos me han apoyado de una u otra manera a lo largo de estos años y de esta aventura que llamamos vida.

Gracias a los profesores, especialmente a Angel Herranz, que me ha enseñado lo importante que es tener curiosidad por lo desconocido. Gracias también a Victor Ramperez, porque por muy ocupado que este, siempre está ahí cuando le necesitas. Y por último, gracias a Javier Soriano, por darme la oportunidad de hacer este increíble TFG.

Gracias a Roberto García, Antonio González y María Salgado, por enseñarme, ayudarme y hacerme crecer como profesional. Gracias a Paula Pousa, por el increíble equipo que hemos hecho juntos, por su paciencia, dedicación, y alegría.

Y más importante aún, gracias a mis padres y a mi hermana, por quererme, aceptarme y apoyarme. Especialmente, gracias a mi madre, por darme la oportunidad todos los días de crecer como persona.

*Jorge Sol Gonzalez*  
Madrid, 2021



# Índice general

---

<b>Resumen</b>	<b>III</b>
<b>Abstract</b>	<b>v</b>
<b>Agradecimientos</b>	<b>VII</b>
<b>Índice de figuras</b>	<b>xi</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Contexto . . . . .	1
1.2. Motivación . . . . .	1
1.3. Objetivos . . . . .	2
1.4. Estructura . . . . .	3
<b>2. Estado del Arte</b>	<b>5</b>
2.1. Blockchain . . . . .	5
2.1.1. Definición . . . . .	5
2.1.2. ¿Cómo funciona? . . . . .	5
2.1.3. Tipos de redes blockchain . . . . .	9
2.1.4. Tipos de algoritmo de consenso . . . . .	10
2.1.5. Smart contracts . . . . .	12
2.1.6. Usos en el presente de Blockchain . . . . .	13
2.2. Ethereum . . . . .	13
2.2.1. Ether (ETH) . . . . .	14
2.2.2. Carteras . . . . .	14
2.2.3. Ethereum Smart Contract . . . . .	14
2.2.4. Funcionamiento básico de Ethereum . . . . .	15
2.2.5. Nivel 1: Máquina Virtual de Ethereum . . . . .	15
2.2.6. Nivel 2: Smart Contract . . . . .	16
2.2.7. Nivel 3: Nodos de Ethereum . . . . .	16
2.2.8. Nivel 4: API para el cliente de ethereum . . . . .	16
2.2.9. Nivel 5: Aplicaciones para usuario . . . . .	17
2.3. Aplicaciones móviles que usan Blockchain . . . . .	17
2.3.1. GUTS Tickets . . . . .	17
2.3.2. LifeID . . . . .	17
2.3.3. Voatz . . . . .	18
2.4. Android . . . . .	18
2.4.1. Conceptos Básicos de Android . . . . .	18
2.5. Desarrollo de Microservicio Externo . . . . .	21

<b>3. Desarrollo</b>	<b>25</b>
3.1. Análisis . . . . .	25
3.1.1. Objetivos y público de Estublock . . . . .	25
3.1.2. Casos de Uso . . . . .	28
3.2. Arquitectura de la Aplicación . . . . .	32
3.3. Interfaz de Usuario . . . . .	32
3.3.1. Interfaces Gráficas en Android . . . . .	32
3.3.2. Accesibilidad de Estublock . . . . .	33
3.3.3. Diseño de la Interfaz de Estublock . . . . .	36
3.3.4. Implementación XML . . . . .	44
3.4. Implementación . . . . .	46
3.4.1. Implementación Java . . . . .	47
3.5. SDK . . . . .	50
3.5.1. Diseño del SDK . . . . .	50
3.5.2. Comunicación con la Red Blockchain . . . . .	51
3.5.3. Comunicación con el Dispositivo Móvil . . . . .	52
3.5.4. “Callback Listener” . . . . .	53
3.5.5. Cómo Incorporarlo en Otras Aplicaciones . . . . .	54
3.6. Pruebas . . . . .	54
3.6.1. Pruebas en Estublock . . . . .	55
3.7. Wallet . . . . .	57
3.7.1. Gestión del Wallet . . . . .	57
3.7.2. Perdida y Recuperación del Wallet . . . . .	58
3.8. Seguridad y Keystore . . . . .	58
3.8.1. Keystore . . . . .	59
3.8.2. Keystore en Estublock . . . . .	61
3.9. Documentación . . . . .	62
<b>4. Análisis de Impacto</b>	<b>65</b>
<b>5. Conclusiones</b>	<b>67</b>
<b>6. Trabajo Futuro</b>	<b>69</b>
<b>Bibliografía</b>	<b>71</b>
<b>7. Anexo</b>	<b>75</b>

# Índice de figuras

---

2.1. Ejemplo del hashing de una contraseña . . . . .	6
2.2. Hash dificultad . . . . .	7
2.3. Vida de una transacción en Bitcoin . . . . .	7
2.4. Cadena de bloques . . . . .	8
2.5. Algoritmos de Consenso . . . . .	10
2.6. Máquina de ventas en solidity . . . . .	13
2.7. Ethereum . . . . .	14
2.8. Código ejemplo de un smart contract . . . . .	15
2.9. Lenguajes de Smart Contract . . . . .	15
2.10. Diagrama EVM . . . . .	16
2.11. Nodos vista genérica . . . . .	16
2.12. Logos de Guts y GET . . . . .	17
2.13. Nodos vista genérica . . . . .	18
2.14. Swagger . . . . .	21
2.15. Swagger Usuario . . . . .	21
2.16. Comunicaciones de la API . . . . .	23
3.1. Caso de uso general del alumno . . . . .	26
3.2. Caso de uso general del profe . . . . .	27
3.3. Caso de uso de registrar usuario . . . . .	29
3.4. Caso de uso de login . . . . .	29
3.5. Caso de uso de suscripcion a temas . . . . .	30
3.6. Caso de uso de crear evento . . . . .	30
3.7. Caso de uso de crear QR . . . . .	31
3.8. Caso de uso de escanear QR . . . . .	31
3.9. Arquitectura . . . . .	32
3.10. Android Layout . . . . .	33
3.11. Clasificación de las principales formas de discapacidad . . . . .	34
3.12. Jerarquía de prioridad . . . . .	35
3.13. Contrastos de un ícono . . . . .	36
3.14. Feedback ante un error . . . . .	36
3.15. Flujo entre pantallas de “Estublock” . . . . .	37
3.16. Código regex de verificación de correo . . . . .	38
3.17. Pantalla de Registro . . . . .	38
3.18. Pantalla de Login . . . . .	39
3.19. Pantalla de Menú . . . . .	40
3.20. Pantalla de Asignaturas . . . . .	40
3.21. Pantalla de Suscribirse . . . . .	41
3.22. Pantalla de Crear Evento . . . . .	42
3.23. Pantalla de Hora . . . . .	42
3.24. Pantalla de QR . . . . .	43

3.25. Pantalla de Escaneo de QR . . . . .	44
3.26. XML Jerarquia . . . . .	45
3.27. XML Herramienta Android Studio . . . . .	45
3.28. Java vs Kotlin . . . . .	46
3.29. Facilitar el salto a Kotlin . . . . .	47
3.30. Jerarquia Estublock . . . . .	48
3.31. Llamada POST con Volley . . . . .	48
3.32. Llamada DELETE con Okhttp . . . . .	49
3.33. Ejemplo de cifrado de contraseñas . . . . .	49
3.34. Ejemplo básico de generar QR . . . . .	49
3.35. Ejemplo de Intent en Android . . . . .	50
3.36. Constructor de TransactionsHelper . . . . .	51
3.37. Firmado y enviado de una transacción . . . . .	52
3.38. Modificación de proveedor de seguridad . . . . .	52
3.39. Creación de un nuevo wallet . . . . .	53
3.40. Recuperación de las credenciales . . . . .	53
3.41. Guardado y recuperación de datos . . . . .	53
3.42. Función de ejemplo de un callback . . . . .	54
3.43. Ciclos asociados con el desarrollo iterativo . . . . .	55
3.44. Visión general de los wallets . . . . .	58
3.45. Contenido de un Keystore . . . . .	59
3.46. Descifrado de la clave privada con AES . . . . .	60
3.47. KDF de una contraseña . . . . .	61
3.48. Comprobación de contraseña correcta con MAC . . . . .	61
3.49. Sistema de archivos interno de la aplicación . . . . .	62
4.1. Consumo eléctrico de bitcoin . . . . .	66
7.1. Código de GlobalState . . . . .	75
7.2. Código del escaneado de QRs . . . . .	76
7.3. Firmado y enviado de una transacción . . . . .	77
7.4. Documentación de TransactionsHelper . . . . .	78
7.5. Documentación de WalletHelper . . . . .	79
7.6. Flujo completo del funcionamiento de un keystore . . . . .	80

---

# CAPÍTULO 1

# Introducción

---

En esta sección se presenta el contexto del TFG, la motivación detrás de este trabajo, los objetivos a cumplir y una breve explicación de la estructura del documento.

## 1.1. CONTEXTO

A lo largo de los dos últimos años, hemos tenido la suerte de estar trabajando en la *Cátedra Inetum* de la Escuela técnica superior de ingenieros informáticos, en colaboración de la empresa Inetum. La cátedra Inetum se constituye para estrechar la colaboración entre la universidad y la empresa Inetum. Inetum es una compañía multinacional de servicios ágil que proporciona servicios y soluciones digitales y un grupo global que ayuda a compañías e instituciones a aprovechar al máximo la corriente digital. Durante estos dos años, hemos estado investigando sobre la tecnología blockchain y el potencial que puede aportar a las personas, en concreto enfocado al mundo universitario.

Dentro de esta cátedra, y con los conocimientos e investigaciones realizadas, se ha ideado desde cero un proyecto muy innovador. El proyecto “Estublock”, tiene el objetivo de lanzar una red blockchain que todos los estudiantes puedan utilizar para desarrollar sus propias aplicaciones. Como primer producto se ha desarrollado la aplicación “Estublock”, que surge ante la necesidad de un sistema fiable, robusto y rápido, de registro de asistencias a exámenes. A partir de esta necesidad, se ha expandido la funcionalidad de la aplicación para cubrir otros tipos de eventos como prácticas, talleres, laboratorios, etc. Con la meta, de hacer de Estublock el sistema estandar para validar la asistencia a exámenes, prácticas, talleres, etc de todas las Escuelas (y facultad) de la Universidad Politécnica de Madrid, y ofrecer la posibilidad de extenderlo al resto de universidades públicas y así asegurar una correcta educación.

## 1.2. MOTIVACIÓN

Es habitual que, como parte de los procesos de evaluación, se consideren actividades que requieren de presencialidad (laboratorios, participación activa en el aula, etc.). La utilización de "listas de asistencia" presenta múltiples problemas, incluidos los de retraso en el comienzo de la actividad por el tiempo que requiere ese tipo de control, posible pérdida de los listados, sobre todo cuando se utiliza el papel como soporte, posibilidad de que se suplanten identidades, etc. El problema radica en que no hay una forma común, confiable y transparente de validar la asistencia a dichas pruebas. Esto es extrapolable a cualquier tipo de evento que requiera que se valide la asistencia (obtención de créditos de libre elección, etc.).

No existe un único método para gestionar las asistencias a exámenes, charlas, talleres, prácticas, etc. Ni existe un protocolo que todos los profesores u organizadores de eventos sigan al pie de la letra. De hecho, raramente se gestiona la asistencia a exámenes o prácticas, exceptuando alguna en la que se pide al alumno identificarse, pero sin llegar a registrarla en ninguna parte. Perfectamente la solución a este problema puede ser pedir que los alumnos firmen una hoja, pero es tedioso, lleva

tiempo, seguimos sin poder verificar la verdadera identidad del alumno. Y al igual que un examen, la hoja puede desaparecer. Y es por eso mismo, que la mejor solución es usar la tecnología a nuestro favor.

“Estublock” viene a resolver este problema. La aplicación permitirá registrar estas asistencias, escaneando un código QR y registrando la información en una red blockchain. Además, dará soporte para otros eventos como talleres, el congreso anual que se realiza en la escuela “TryIT”, prácticas, charlas, etc. Con el tiempo, la aplicación irá creciendo y mejorando, trayendo mejoras poco a poco y con la capacidad de extenderlo a otros campus de la Universidad Politécnica de Madrid y posteriormente crecer a otras universidades públicas para que todas puedan aprovechar el potencial de “Estublock”.

Tanto por el bien del alumno, como del profesor, este proyecto es muy motivador pues ante todo nos parece que la vida ha de ser justa, y es motivador saber que con esta aplicación se evitarán los fraudes con los que se tiene que lidiar en el presente. Tanto alumnos irresponsables que no quieren aceptar la realidad, como docentes despistados a los que se les ha extraviado un examen. Además, hacer crecer este proyecto y ser capaces de validar mucho más que las asistencias a exámenes hace de este proyecto una oportunidad de mostrar todo lo aprendido, y será un gran orgullo verlo en funcionamiento en un futuro.

También, parte de este TFG es el desarrollo de un “kit de desarrollo software” (SDK) el cual será de código abierto y disponible en repositorios públicos como GitHub o Maven para que otros desarrolladores en cualquier parte del planeta puedan utilizarlo. Esta contribución al mundo del software libre, es de gran interés personal pues las innovaciones en tecnología están para compartirlas y lograr que crezcan con la ayuda de toda la comunidad de desarrolladores interesados.

Además, disponemos de la oportunidad de trabajar en la Cátedra Inetum, en el Campus Blockchain, lo que facilita la comunicación con profesionales en la materia del mundo del blockchain y la tecnología punta. Haciendo de esta, una gran oportunidad para aprender y hacer crecer el círculo de relaciones profesionales. Así como la oportunidad de mejorar en el trabajo en equipo y aprender cómo es la vida en una empresa y con qué problemas hay que lidiar a la hora de hacer crecer un proyecto en la vida real.

### 1.3. OBJETIVOS

En base a las necesidades que debe solventar la aplicación, se centran todos los objetivos principales que han sido desarrollados en este Trabajo de Fin de Grado.

- Analizar profundamente la tecnología Blockchain, en concreto la red de Ethereum y sus smart contracts. Esto es importante pues la aplicación móvil desarrollada deberá ser capaz de comunicarse con una red *Quorum* que aprovecha la red de Ethereum.
- Desarrollo de una API que sirva como medio de comunicación entre la aplicación móvil que se va a desarrollar con el servidor de la base de datos que guarda información sobre los alumnos, detalles de las asignaturas, etc. Y con el servidor que ejecuta uno de los nodos de la red blockchain.
- Desarrollo de la aplicación móvil Android, la cual a parte de comunicarse con la API anteriormente mencionada, tendrá también que enviar a la red blockchain transacciones firmadas por el usuario, así como crear y guardar el wallet del usuario.
- Desarrollo de un SDK a partir de la aplicación anteriormente mencionada, este SDK contendrá la funcionalidad de comunicación con la red blockchain y de gestión del wallet del usuario. La aplicación móvil utilizará entonces este SDK una vez terminado.
- Desarrollo de un documentación para el correcto uso del SDK, y así facilitar a otros desarrolladores su uso, así como normas para el despliegue del mismo.

Siendo de los puntos anteriormente mencionados, la Aplicación Android, el corazón del trabajo realizado.

#### 1.4. ESTRUCTURA

Para lograr cumplir con los objetivos propuestos, se ha dividido la estructura del trabajo en las siguientes secciones.

El capítulo *Estado del Arte*<sup>2</sup> contiene una introducción a la tecnología Blockchain, haciendo especial hincapié en la red de *Ethereum*<sup>[44]</sup>. La razón de esta decisión es que la red que se ha utilizado para este proyecto es una red de Quorum<sup>[46]</sup> la cual permite aprovechar el potencial de *Ethereum* en aplicaciones blockchain. Además, se introducen al final del capítulo ejemplos de otras aplicaciones móviles existentes en el presente con funcionalidades diferentes pero que utilizan la red de *Ethereum* para sus transacciones. Luego se han añadido dos apartados de trabajo realizado durante el TFG. El primero es un apartado sobre la API que se ha desarrollado para la comunicación entre el móvil y la base de datos y blockchain. Este apartado se encuentra en el estado del arte puesto que se ha realizado en equipo. Y, finalmente, se presenta un apartado sobre conceptos básicos de Android que se han ido aprendiendo a lo largo del desarrollo del TFG.

El siguiente capítulo *Desarrollo*<sup>3</sup>, explica todo el diseño de la aplicación móvil tanto a nivel de interfaz de usuario como a nivel interno. Además, se hace hincapié en la comunicación con la red blockchain así como las librerías utilizadas en el proceso. También se profundiza en el desarrollo del SDK, su funcionamiento, el tratamiento del wallet, y su uso en otras aplicaciones. Terminando con la documentación del SDK. Y por último un estudio sobre la seguridad de la aplicación y la seguridad del keystore.

Para considerar los objetivos de desarrollo sostenible, se ha añadido un capítulo *Impacto Medioambiental*<sup>4</sup> en el que se expone el impacto del uso de la aplicación desarrollada.

Por último, la *Conclusión*<sup>5</sup> y los *Pasos a Futuro*<sup>6</sup>, recogen los resultados del trabajo, las conclusiones personales, y cuales son los pasos a seguir para hacer crecer al proyecto “Estublock”.



---

## CAPÍTULO 2

# Estado del Arte

---

En este capítulo se explicará qué es blockchain, cómo funciona, qué tipos hay, así como ver su uso en el presente y qué aplicaciones se están desarrollando alrededor de la misma para sacarle su máximo potencial. Este apartado es de crucial importancia pues servirá para ver el estado del arte de las aplicaciones (en concreto aplicaciones móviles) del presente. Además, se explicará el trabajo realizado para preparar la aplicación móvil, en concreto la APIRestful que se ha desarrollado para comunicar el móvil con la base de datos y parte de la red blockchain. También, se estudian conceptos básicos de Android que se han aprendido para preparar el proyecto.

### 2.1. BLOCKCHAIN

Blockchain[11, 81, 84, 85] es un término escuchado hoy en día por todas partes, más a nivel económico que tecnológico. Y aunque pueda parecer complejo, su funcionamiento es bastante sencillo.

#### 2.1.1. Definición

Blockchain es una tecnología **DLT** (Distributed Ledger Technology), un *ledger* es un libro contable o de contabilidad. Por lo que DLT viene a ser una tecnología de libros de contabilidad distribuidos. DLT se refiere a la infraestructura tecnológica y a los protocolos que permiten el **acceso, validación y actualización** de registros de forma **simultánea e inmutable** a través de una red que se extiende por **múltiples entidades** o ubicaciones (en el caso de blockchain, estas entidades son ordenadores y se les llama nodos).

La tecnología blockchain se sostiene en tres pilares principales:

1. **Transparencia:** Toda la información que se registra en la red blockchain es pública, todas las transacciones pueden ser visualizadas por todos los nodos que conforman la red blockchain. Aunque pueda parecer peligroso que todo el mundo pueda ver todas las transacciones, Blockchain utiliza complejos mecanismos criptográficos para mantener el anonimato en la red, de modo que nunca se puede saber quienes son los participantes involucrados en una transacción.
2. **Inmutabilidad:** Una vez almacenada la información en la red, esta pasa a regirse por las reglas de la red, no pudiendo ser modificadas, alteradas o eliminadas de ninguna manera.
3. **Descentralización:** La información que registra la red blockchain no queda guardada en un único lugar, no hay una empresa con toda la información centralizada en sus instalaciones. Todos los nodos (ordenadores) que forman la red blockchain tienen una copia exacta y actualizada de la información que hay en la red. Las redes distribuidas permiten además eliminar la necesidad de una autoridad central que controle el sistema para evitar manipulaciones o fraudes.

#### 2.1.2. ¿Cómo funciona?

Una Blockchain es un sistema de registro de transacciones en constante crecimiento, junta la información en grupos conocidos como **bloques**. Cada bloque contiene información sobre las transacciones

que se han realizado, así como información adicional en la cabecera. En la cabecera del bloque tenemos información sobre el número del bloque, el **hash** del bloque anterior, el **timestamp** y el **nonce** entre otros.

Entendamos estos conceptos:

- Hash: Un hash<sup>[83]</sup> es un algoritmo que mezcla la información que se le introduce para generar una salida única e irreversible que tiene siempre una longitud fija. Es importante que cada salida es única para la información introducida, el más mínimo cambio en la información introducida y la salida cambiará por completo. Es como una huella digital de la información, no hay dos huellas iguales. Las funciones hash se utilizan de forma habitual para guardar la contraseña de personas en una base de datos, así, la contraseña nunca queda guardada en claro y si la base de datos se ve comprometida, es casi imposible recuperar la contraseña original [2.1](#). Aplicado a una red Blockchain, se ejecuta una función **hash256** sobre toda la información de un bloque. El resultado se utiliza en el siguiente bloque que se vaya a crear. Es decir, que cada bloque tiene el hash del bloque anterior (y el siguiente bloque tendrá el hash del bloque actual).
- Timestamp: Identifica la hora exacta en segundos de la creación del bloque.
- Nonce: Un nonce<sup>[82]</sup> es un número que solo puede ser utilizado una única vez.

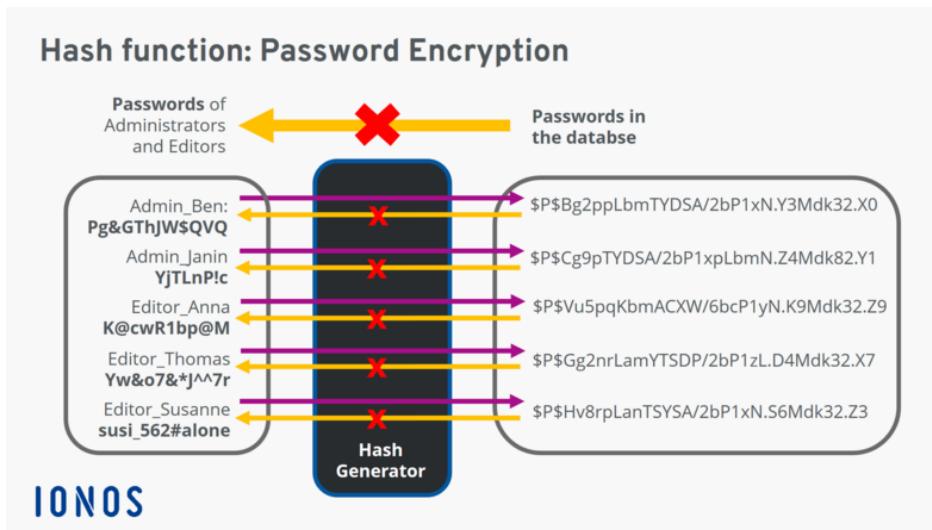
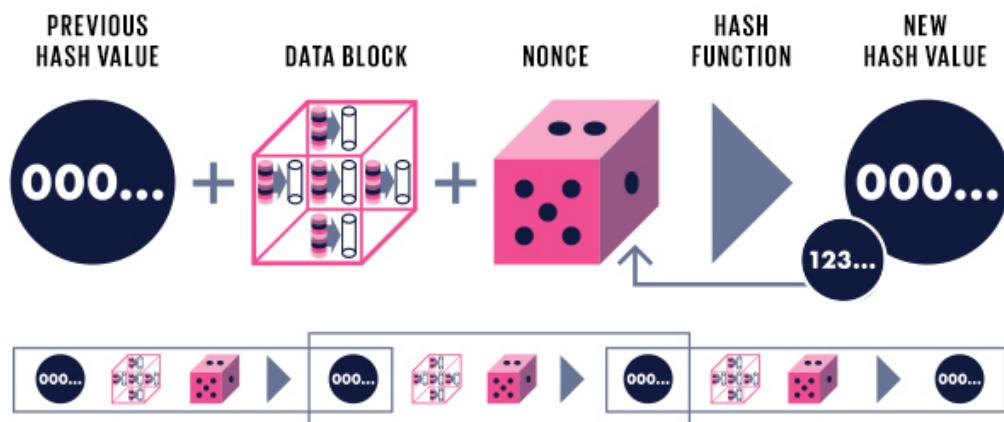


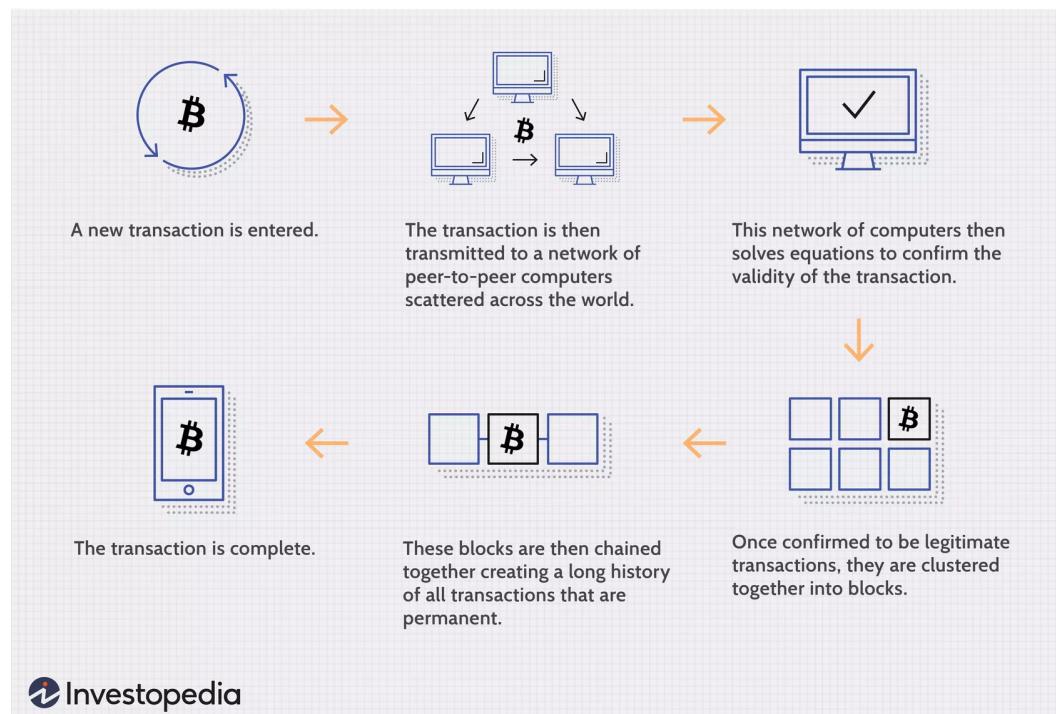
Figura 2.1: Ejemplo del hashing de una contraseña ([Ionos](#))

Para continuar con la explicación, utilizaremos redes blockchain que utilizan el algoritmo de consenso **Proof of Work**<sup>[29]</sup> como hace por ejemplo la red de **Bitcoin**<sup>[58]</sup>. La base es la misma para la inmensa mayoría de redes, sin embargo según el algoritmo de consenso el método difiere ligeramente. Los algoritmos se verán mas adelante [2.1.4](#). En el caso de Proof of Work, para añadir un nuevo bloque a la red, debe ser **minado**<sup>[40]</sup>. Para minar un bloque, todos los nodos de la red blockchain tratan de encontrar un hash con un requisito añadido. Ejecutar un hash sobre cualquier tipo de información, es un proceso muy fácil. Por ello, utilizando Proof of Work, se añade una dificultad al hash. La dificultad<sup>[2.2]</sup> consiste en obligar que el hash resultante tenga un número de **0's** al principio del resultado. Por lo tanto, el minado consiste en modificar el **nonce** (pues no se pueden modificar las transacciones que hay en el bloque o el hash del bloque anterior). Los nodos de la red modifican el nonce de forma aleatoria hasta que algún nodo logre dar con un nonce que cumpla con la dificultad del hash. Logrado esto, el resto de nodos de la red verifican que el nonce que ha encontrado resuelve en efecto la dificultad del hash y se procede a añadir el bloque a la red y vuelta a empezar. En el caso de bitcoin, se mina un bloque cada 10 minutos aproximadamente.



**Figura 2.2:** Dificultad del hash ([Spectrum](#))

La vida de una transacción en bitcoin a grandes rasgos puede verse en la figura 2.3. La razón de que blockchain sea una cadena de bloques, es que después de todo el proceso de hashing, de minado...los bloques quedan enlazados entre si gracias a estos hashes<sup>2.4</sup>.



**Figura 2.3:** Vida de una transacción en Bitcoin ([Investopedia](#))

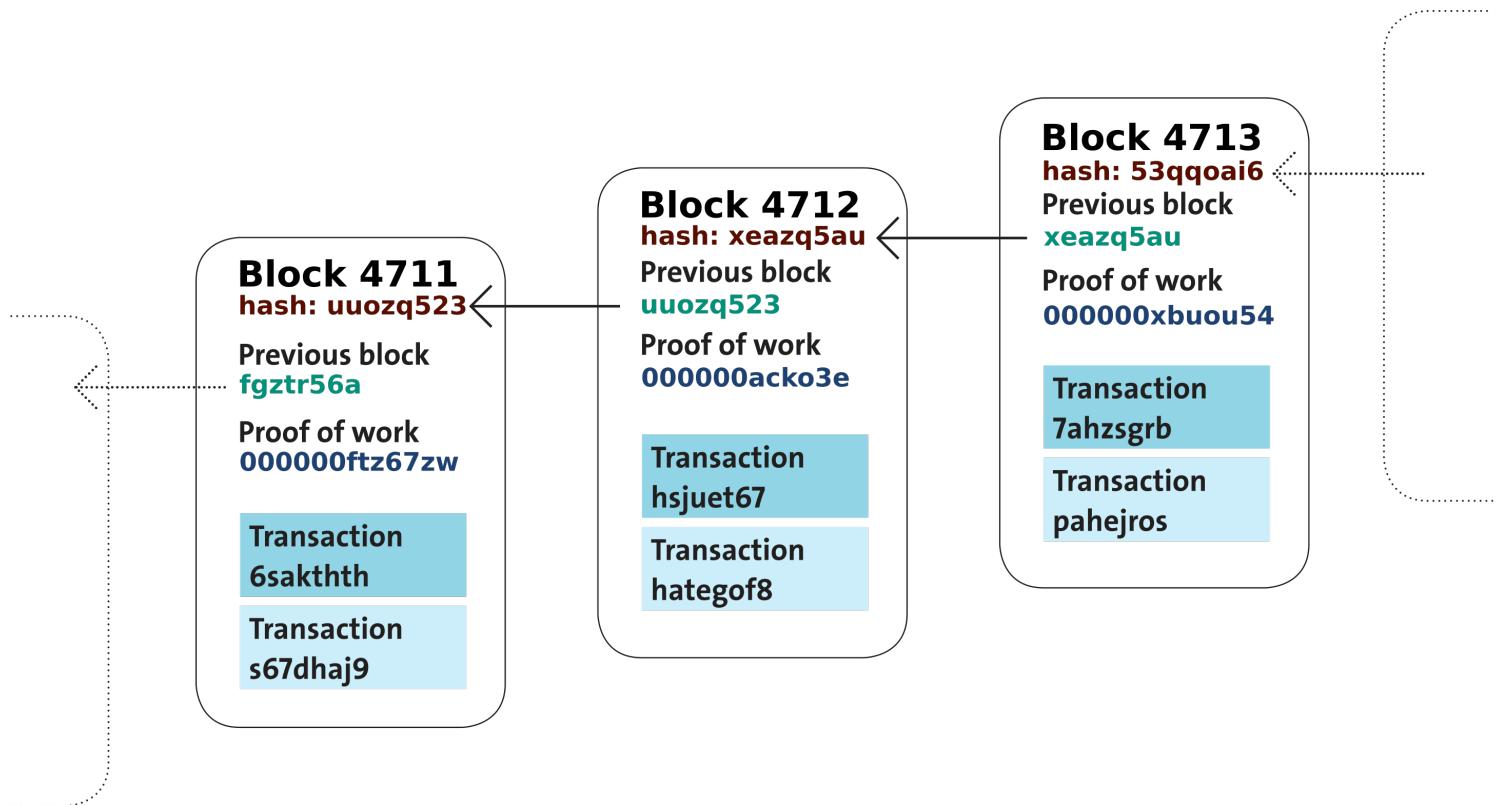


Figura 2.4: Cadena de bloques ([U.Zurich](#))

### 2.1.3. Tipos de redes blockchain

Cuando se habla de blockchain, parece dar la impresión de que solo existe un tipo. Sin embargo hay varias redes con sus ventajas y desventajas[8, 49]. Las principales diferencias entre ellas son las **funcionalidades, protocolos de consenso, administración y reglas para validar las transacciones**.

#### Blockchain pública

Las blockchain públicas no tienen permisos, cualquier usuario es bienvenido a unirse a la red, a enviar transacciones, a utilizar las funcionalidades que tiene la red, y a minar bloques. Las principales características de esta red son:

- Son **transparentes**: El código, el funcionamiento interno, los smart contracts si tiene (se verá este término mas adelante) son todos públicos y de código abierto.
- Sin **permisos**: Cualquier persona puede unirse a la red sin preguntar. Lo único que tiene que hacer es descargar la red y sincronizarse con los demás nodos.
- Usuarios **anónimos** y sin **administradores**: Nadie se conoce en la red, se trabaja siempre con lo que se conoce como **address**, que viene a ser un identificador único por miembro dentro de la red para identificarlo. Además, no existe administrador de la red, no hay una persona o grupo que tenga poder sobre la red para hacer cambios de ningún tipo.
- La información de la red puede ser **mantenida por todas las personas que lo deseen**. Y al minar nuevos bloques, dependiendo de la red, se ofrece un **incentivo**.

En resumidas cuentas, una blockchain pública es *descentralizada, distribuida, consensuada, abierta y segura*. Algunos ejemplos de redes públicas son bitcoin[42] y ethereum[44]

#### Blockchain privada

Las blockchains privadas son permisionadas, esto quiere decir que no cualquier persona puede añadirse como nodo libremente. Requieren de una **entidad** que ejerza de **administrador**. La mayoría de usuarios no consideran estas redes como blockchain a causa de esto mismo. El administrador de la red tiene que dar permiso a los usuarios para poder minar, enviar transacciones y participar en general en la red.

Además, es habitual que los datos estén almacenados en nodos centrales y no abiertos al público. Pudiendo acceder a los bloques de la red solo mediante invitación.

Algunos ejemplos de blockchains privadas son R3[47], Ripple[48] y Quorum[46]

#### Blockchain híbrida o federada

Estas redes son utilizadas por grandes empresas y gobiernos, no tienen porqué estar abiertas al público, teniendo la gestión varias entidades. Además no tienen una criptomoneda asociada y no recompensan por el minado de bloques. Sin embargo el software que utilizan es de código abierto, como puede ser **Hyperledger, Corda**[43, 45].

Como ejemplo tenemos la *Enterprise Ethereum Alliance*, en la que participan el Banco Santander y BBVA. Esta red utiliza la blockchain de Ethereum (pública), sin embargo tienen su propia plataforma privada.

#### Blockchain as a Service

Estas redes blockchain son controladas por un proveedor de servicios como puede ser *Amazon*. Estos proveedores permiten utilizar redes blockchain en la nube, permitiendo a los desarrolladores

aprovechar el potencial de las redes blockchain sin la necesidad de invertir en el cómputo que ello requiere.

#### 2.1.4. Tipos de algoritmo de consenso

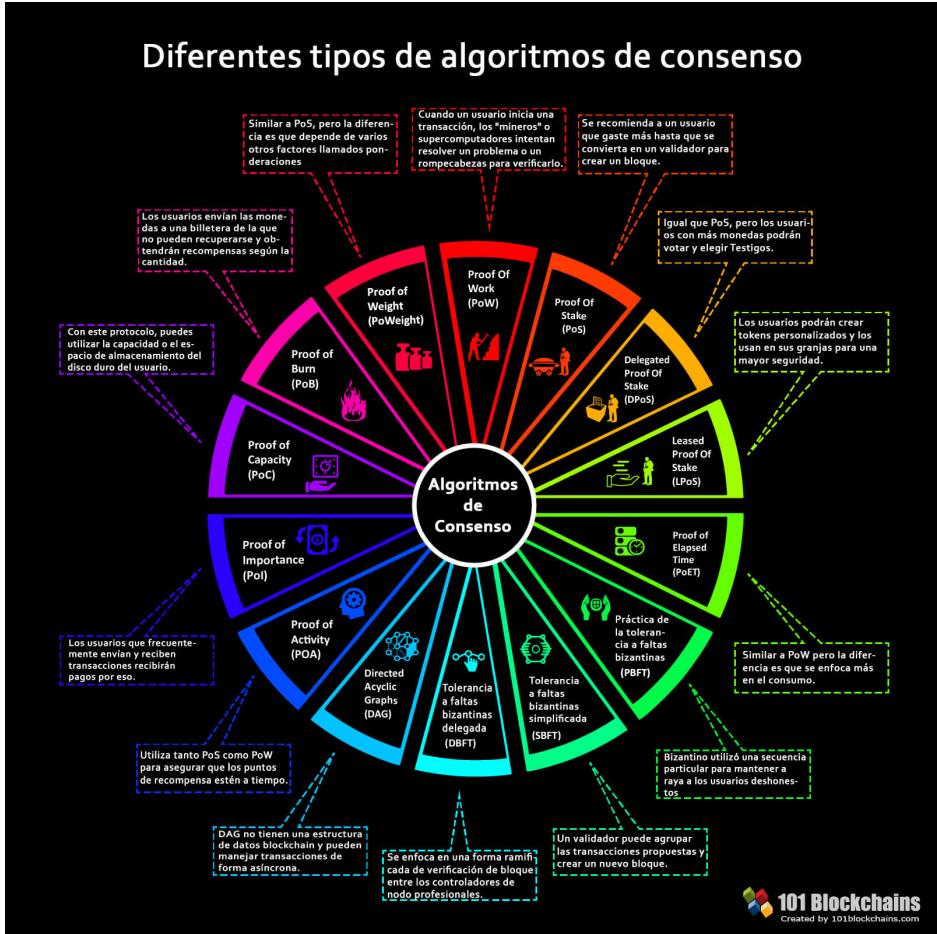


Figura 2.5: Diferentes tipos de algoritmos de consenso (101Block)

Existen múltiples algoritmos de consenso<sup>2.5</sup>, y estos evolucionan con el paso del tiempo. Los algoritmos de consenso son procesos o protocolos de toma de decisiones, dependiendo del algoritmo hay uno o varios nodos de la red con el poder de tomar la decisión sobre que bloque es el siguiente en añadirse a la red y si ha sido o no alterado. Los objetivos que busca blockchain con los algoritmos de consenso son:

- Llegar a un acuerdo
- Cooperación
- Colaboración
- Igualdad de derechos
- Participación
- Actividad

Puesto que hay una gran cantidad de algoritmos trataremos de explicar algunos a continuación<sup>[53]</sup>. Todos los algoritmos buscan el mismo objetivo, solucionar el problema de los **fallos bizantinos**(BFT)<sup>[61]</sup>. La tolerancia a fallos bizantinos es la resistencia de un sistema informático tolerante a fallas de componentes electrónicos. Si lo llevamos al mundo blockchain, cuando hablamos de fallas nos referimos a nodos de la blockchain defectuosos accidentalmente o provocado. Si una empresa tiene control de 200 nodos, puede tratar de crear transacciones falsas y minar ese bloque con los 200 nodos, el resto de nodos de la red tienen que ser capaces a través del algoritmo de consenso que se

utilice de descartar la información de estos 200 nodos.

### Proof of Work (PoW)

El algoritmo de prueba de trabajo es el primer algoritmo introducido en la red blockchain. Muchas blockchains utilizan este algoritmo para llevar a cabo el consenso y así confirmar todas las transacciones.

*¿Cómo funciona?* Cada nodo tiene descargada la red blockchain entera, cuando el nuevo bloque tiene las transacciones necesarias para ser minado, todos los nodos se ponen a buscar el resultado de un **hash** con la dificultad que tenga el mismo. A más poder de cómputo, más posibilidades de encontrar el resultado, validarla con los otros nodos y añadirlo a la red blockchain, además las redes que usan PoW suelen tener un sistema de premios por lo que cuando un nodo encuentra la solución al problema se le da criptomonedas a cambio.

Este sistema tiene dos principales desventajas, la primera es que el poder de cálculo que se necesita es muy grande, y estos últimos años han crecido lo que se conoce como granjas de minado las cuales se llevan el premio la gran mayoría de veces. Esto causa que el sistema empiece a centralizarse, rompiendo con la idea de descentralización que tiene blockchain. El problema es que si alguien logra tener el poder de cómputo de un 51 % de la red blockchain este puede añadir los bloques que quiera a partir de ese momento pues siempre serán validados por la mayoría de nodos (su 51 % de los nodos). La segunda gran pega va ligada a estas granjas de minado, consumen gran cantidad de energía, en 2020 bitcoin consumió 120 gigawatts por segundo<sup>[7]</sup> lo que a nivel medioambiental tiene un impacto negativo, por lo que PoW no es un algoritmo de consenso que se pueda mantener en el tiempo.

*Ventajas y Desventajas*

- PROS

- Evita ataques DDoS
- Es justo y transparente
- Fomenta el interés del público en mantener una red saludable

- CONS

- La adquisición de equipo para el minado es costosa
- La máquina destinada al minado no podrá utilizarse para otra tarea pues se necesita todo el poder de cómputo en la resolución de los problemas matemáticos (función hash).
- La red tiende a centralizarse a causa de las granjas de minado, dándole poder al dueño de la granja y rompiendo la descentralización.
- La minería podría desaparecer cuando no haya más incentivos, en el caso de Bitcoin, cuando se alcance el límite de Bitcoins (21 Millones) los mineros dejarán de recibir premios y se perderá la motivación del minado. No tiene porque desaparecer la red, pero perderá seguidores.

### Proof of Stake (PoS)

El concepto de participación establece que una persona puede minar o validar transacciones en bloque según el número de monedas que posea. Esto significa que cuanta más criptomoneda tengas, más poder de minado se te asigna.

Se creó como alternativa a PoW, para solventar algunos problemas que tiene (como el de la centralización a causa de las granjas de minado y el gasto energético derivado del poder de cómputo). PoS solventa este problema atribuyendo la potencia minera a la proporción de monedas que posee un minero. Por lo tanto, en vez de utilizar energía para responder al puzzle matemático como hace PoW, aquí el minero se limita a resolver un porcentaje de las transacciones. Si se tiene un 3 % de las criptomonedas disponibles, se puede minar un 3 % de los bloques, en esta ocasión no hay que resolver

ningún puzzle matemático, simplemente generar un **hash** con los datos, lo que es una tarea trivial.

PoS no es una solución definitiva, pues tiene problemas al igual que PoW. Al igual que antes mencionamos el ataque del 51 % (cuando una empresa o alguien tiene un 51 % de los nodos en su poder). En PoS, si tienes un 51 % de las criptomonedas, tienes un 51 % del poder de decisión sobre la red, pudiendo hacer los cambios que quieras en ella. Este ataque es frecuente en redes pequeñas con pocas criptomonedas en juego, puesto que de lo contrario, lograr tener un 51 % de las monedas supone tener muchísimo dinero. Para controlar estas fraudulencias, se penaliza económicamente a los nodos que tratan de saltarse las reglas (modificar bloques y transacciones), además, un ataque a la blockchain afecta al poder de la moneda y por lo tanto su valor en mercado disminuye, por lo que no compensa tratar de burlar las reglas de la blockchain. Los nodos de las redes que usan PoS, reciben una comisión al minar correctamente su parte del bloque.

Redes que utilizan este algoritmo son: Ethereum2.0[62] y NxT[71].

### **Proof of Elapsed Time (PoET)**

La prueba de tiempo transcurrido es un algoritmo que evita la alta utilización de recursos y el alto consumo de energía manteniendo el proceso más eficiente. El algoritmo utiliza un tiempo transcurrido generado aleatoriamente para decidir los derechos de minería y los ganadores de los bloques. Al ejecutar un código de confianza dentro de un entorno seguro, el algoritmo PoET también mejora la transparencia al garantizar que los resultados sean verificables por participantes externos. Este algoritmo se utiliza en redes blockchain permisionadas, por lo que se conoce al dueño de cada nodo.

Cada nodo participante en la red debe esperar durante un periodo de tiempo elegido al azar, y el primero en completar el tiempo de espera designado gana el nuevo bloque. Cada nodo de la red blockchain genera un tiempo de espera aleatorio y se pone a dormir durante esa duración especificada. El que se despierta primero, es decir, el que tiene el tiempo de espera más corto, se despierta y añade un nuevo bloque en la cadena de bloques, transmitiendo la información necesaria a toda la red de pares. El mismo proceso se repite para descubrir el siguiente bloque.

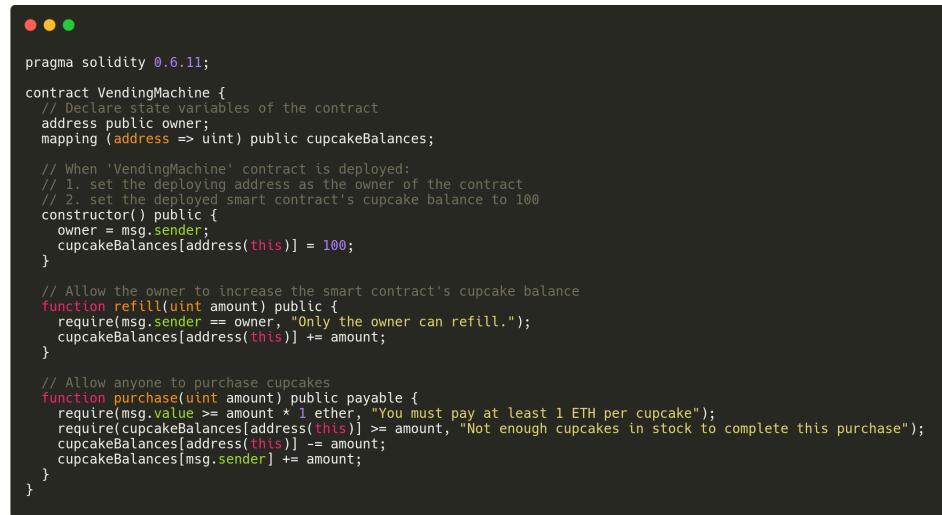
#### **2.1.5. Smart contracts**

Hasta ahora, hemos hablado de blockchain y criptomonedas, pero las criptomonedas son solo un muy pequeño uso del potencial de blockchain. Los **Smart Contracts**[31] son un programa que se ejecuta en la red blockchain dando así la capacidad a la red de ser más versátil, al poder ejecutar código (escrito en el smart contract). Básicamente, añade una lógica a la blockchain.

Una forma de entender los smart contracts es comparándolos a una máquina de ventas automática. Cuando quieres un snack introduces dinero en la máquina y pones el código del snack, la máquina tiene programada una rutina para verificar que has metido la cantidad adecuada y a cambio devuelve el snack seleccionado. Ese programa interno que tiene la máquina, es el equivalente a un **smart contract**.

Al estar programados en la red blockchain, no pueden ser modificados sin que todos los nodos se enteren. De hecho, cuando se quiere actualizar un smart contract, no se actualiza el existente en la red, sino que se realiza una nueva transacción la cual ha de ser validada por el resto de nodos. Los smart contracts permiten que se realicen transacciones y acuerdos de confianza entre partes dispares y anónimas sin necesidad de una autoridad central, un sistema legal o un mecanismo de aplicación externo. No se puede evadir o sortear al smart contract, cuando realizas una llamada a una de sus funciones no hay forma humana de lograr burlarlo, la rutina se ejecutará siempre sin fallos. El

equivalente se puede ver como tener a un notario robótico, el cual nunca comete errores.



```

pragma solidity 0.6.11;

contract VendingMachine {
    // Declare state variables of the contract
    address public owner;
    mapping (address => uint) public cupcakeBalances;

    // When 'VendingMachine' contract is deployed:
    // 1. set the deploying address as the owner of the contract
    // 2. set the deployed smart contract's cupcake balance to 100
    constructor() public {
        owner = msg.sender;
        cupcakeBalances[address(this)] = 100;
    }

    // Allow the owner to increase the smart contract's cupcake balance
    function refill(uint amount) public {
        require(msg.sender == owner, "Only the owner can refill.");
        cupcakeBalances[address(this)] += amount;
    }

    // Allow anyone to purchase cupcakes
    function purchase(uint amount) public payable {
        require(msg.value >= amount * 1 ether, "You must pay at least 1 ETH per cupcake");
        require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in stock to complete this purchase");
        cupcakeBalances[address(this)] -= amount;
        cupcakeBalances[msg.sender] += amount;
    }
}

```

**Figura 2.6:** Máquina de ventas en solidity

Los smart contracts pueden ser escritos en múltiples lenguajes de programación dependiendo de la red blockchain que se vaya a utilizar. Algunos ejemplos son:

- EOS Blockchain ->C++
- Ethereum ->[Solidity 2.6](#)
- NEO Blockchain ->JavaScript, Java
- Hyperldger ->Golang
- Cardano ->Haskell

La primera red blockchain en explotar el potencial de los smart contracts fue **Ethereum** y puesto que es la red blockchain sobre la cual se apoyará la aplicación Android que voy a desarrollar en este TFG, procederé a analizarla.

### 2.1.6. Usos en el presente de Blockchain

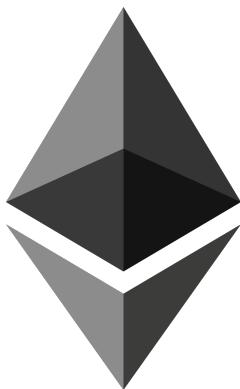
En el presente están en desarrollo múltiples aplicaciones basadas en Blockchain:

- Criptomonedas: *Bitcoin, Ethereum, Tether, Cardano, XRP, Litecoin, ChainLink, Dogecoin, TRON, VeChain, Monero, BitTorrent, Kusama, Neo, Dai, NEM, Dash, Maker...* [60].
- Firma digital y verificación de la identidad: Startups como *Civic o Niuron*[67, 70] buscan implementar firmas digitales para notarios y bancos y check-in telemáticos en hoteles y pisos turísticos.
- Trazabilidad alimentaria: Empresas como carrefour implementan trazabilidad de sus alimentos con la ayuda de IBM [9]
- Turismo y Hoteles: La empresa *TUI Group* cuenta con más de 300 hoteles y esta moviendo sus activos inmobiliarios y procesos internos a una blockchain [51].
- Votaciones y elecciones: El *Banco Santander* utilizó en 2018 con éxito una red blockchain para votar a su Junta General de Accionistas[54]

Y mucho más [3].

## 2.2. ETHEREUM

*Ethereum*[2.7](#) es una plataforma global de código abierto para aplicaciones descentralizadas, permite escribir código que controla el valor digital, funciona tal como se programó y al que puede accederse desde cualquier parte del mundo. Ethereum es un acceso abierto al dinero digital y a los servicios de



**Figura 2.7:** Logo de ethereum

*información para todos, sin importar su origen o ubicación. Es una tecnología creada por la comunidad tras la criptomoneda ether (ETH) y miles de aplicaciones que puedes usar hoy.*

### 2.2.1. Ether (ETH)

La moneda que utiliza la red de Ethereum es el **ether**, se utiliza para recompensar a los mineros que aseguran las transacciones. *Ethereum1.0* utiliza el algoritmo *PoW* y *Ethereum2.0* utiliza *PoS*[62]. Los ethers se utilizan como almacén de valor, prestamos de garantía, medio de intercambio, unidad de cuenta en mercados digitales ...

Cada transacción en ethereum lleva asociado un coste conocido como **gas**. El gas es la unidad que se utiliza para ver el coste de cómputo que tiene la transacción, para poder remunerar adecuadamente al minero. La unidad de gas se mide en **Gwei**[28] si por ejemplo queremos enviar una cantidad X de ether a otra persona esta transacción tiene un coste de 21.000Gwei lo que es equivalente a 0,000021 ETH (1ETH =  $10^9$ Gwei)

### 2.2.2. Carteras

Las carteras de ethereum son aplicaciones que permiten a los usuarios interactuar con sus cuentas de ethereum. Se las puede ver como una aplicación bancaria. Tu cartera te permite leer tu saldo, enviar transacciones y conectarte a aplicaciones. Se necesita una cartera para enviar fondos y gestionar tu ETH. Pero es únicamente una herramienta para gestionar tu cuenta, puedes cambiar de cartera sin problema pues no es quien custodia tus fondos, eres tú quien los custodia en todo momento.

La cartera tiene asociado un **address**, un address es el identificador que tiene tu cuenta en la red de ethereum, es único para ti. Si alguien quiere enviarte dinero lo hará desde su address a tú address, si quieres hacer una llamada a un Smart Contract, al smart contract le llegará como información tú address.

A grandes rasgos nos quedan estos tres términos importantes:

- **Cuenta** de ethereum.
- **Address** de la cuenta.
- **Cartera** para gestionar la cuenta.

### 2.2.3. Ethereum Smart Contract

Los smart contracts de ethereum están escritos en **Solidity**[57]2.8 o **Vyper**[64], sus logos pueden verse en 2.9. Cualquier persona es libre de programar un smart contract y desplegarlo en la red de ethereum, los smart contracts son una transacción más y tienen su propio address. Permitiendo que

cualquier persona haga llamadas al smart contract. Desplegar un smart contract cuesta *gas* al igual que cualquier transacción, sin embargo es bastante más caro.



```

// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.7.0;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}

```

**Figura 2.8:** Código ejemplo de un smart contract



(a) Logo de Solidity



(b) Logo de Vyper

**Figura 2.9:** Lenguajes de smart contract para ethereum.

## 2.2.4. Funcionamiento básico de Ethereum

Para entender el funcionamiento básico de Ethereum, vamos a exponerlo por niveles como hacen en la documentación oficial[30].

## 2.2.5. Nivel 1: Máquina Virtual de Ethereum

La **EVM**[2.10](#) es el entorno de ejecución de los smart contract. Esta, gestiona todo el procesamiento de transacciones en la red, creando un nivel de abstracción entre el código que se ejecuta y la máquina que lo hace. La EVM es *Turing-Completa* con 140 instrucciones únicas que la permiten ejecutar casi cualquier cosa.

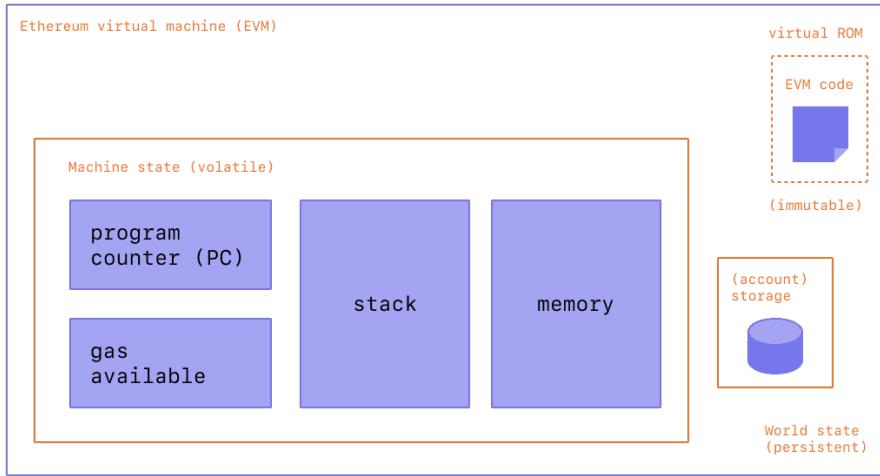


Figura 2.10: Diagrama de la máquina virtual de ethereum

### 2.2.6. Nivel 2: Smart Contract

Programas que se ejecutan en la red de ethereum.

### 2.2.7. Nivel 3: Nodos de Ethereum

Para que una app pueda interactuar con la red, necesita conectarse a un nodo de ethereum<sup>2.11</sup>. Los nodos son ordenadores que ejecutan el *software* cliente de ethereum, manteniendo el registro de bloques y validando las transacciones. Almacenan colectivamente el estado de la blockchain y llegan a un consenso sobre las transacciones para hacer crecer el número de bloques. Al conectar la app con un nodo, la app puede leer datos de la red, enviar nuevas transacciones...

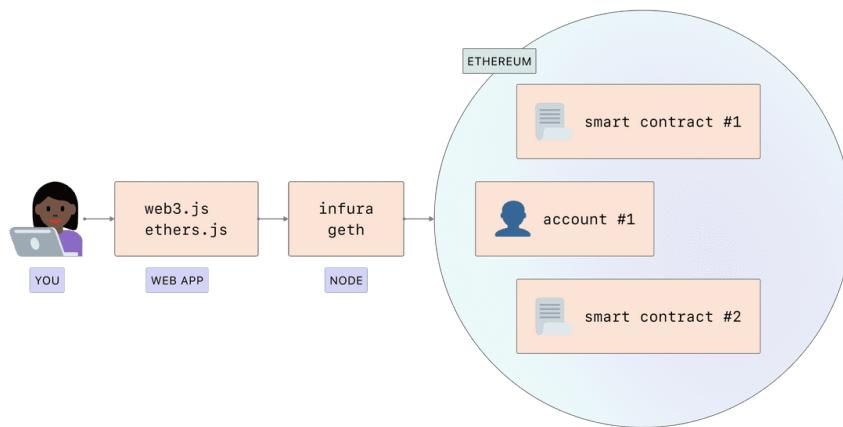


Figura 2.11: Nodo y vista genérica de aplicaciones.

### 2.2.8. Nivel 4: API para el cliente de ethereum

Las APIs permiten a los desarrolladores abstraer parte de la dificultad de interactuar directamente con los nodos de ethereum. Además proporcionan herramientas como para convertir datos de ETH a Gwei... Existen APIs en JavaScript, Java, Python por ejemplo, una de las más conocidas es la librería **Web3**[80], la cual evita al programador tener que implementar la comunicación con los nodos.

### 2.2.9. Nivel 5: Aplicaciones para usuario

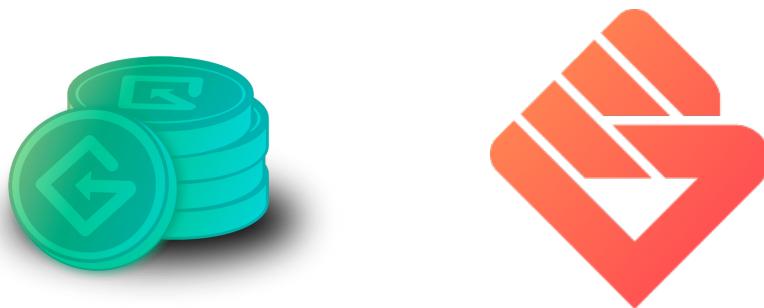
En el nivel superior están las apps orientadas al usuario. Principalmente son aplicaciones web y móvil. Las aplicaciones se construyen de la manera tradicional, y utilizan las APIs necesarias para comunicarse con la red y/o smart contracts. Las aplicaciones distribuidas o aplicaciones que funcionan gracias a una red blockchain, se las conoce como **Dapps**, básicamente son como una aplicación tradicional, pero ejecutadas en una red punto a punto (*P2P*) como blockchain.

## 2.3. APPLICACIONES MÓVILES QUE USAN BLOCKCHAIN

Ahora que entendemos que es Blockchain, visto múltiples ejemplos de redes, visto múltiples aplicaciones de la red Blockchain, y hemos indagado más sobre Ethereum y lo Smart Contract, vamos a pasar a ver los proyectos actuales que utilizan aplicaciones móviles y blockchain. Puesto que el objetivo de este TFG es el desarrollo de una aplicación Android que comunique con una blockchain para luego sacar esa funcionalidad en un SDK.

### 2.3.1. GUTS Tickets

Esta aplicación móvil usa blockchain para emitir entradas honestas que ponen fin a los vergonzosos precios del mercado de la “reventa” y al fraude en las entradas. **GUTS**[\[21\]](#)[\[2.12\]](#) es el primer sistema de ventas de entradas que hace uso del **Protocolo de Entrada Garantizada** (*Guaranteed Entrance Protocol (GET)*)[\[21\]](#)[\[2.12\]](#). Este protocolo permite crear entradas inteligentes y seguras permitiendo el seguimiento de las mismas así como el control de su precio original y secundario (reventa).



(a) Logo del token de GET

(b) Logo de la aplicación Guts

Figura 2.12: Logos

### Protocolo GET

El protocolo GET ofrece una solución de emisión de billetes inteligentes basadas en blockchain que puede ser utilizada por todos los que necesitan emitir billetes de forma transparente, segura y honesta. Evitando fraudes, y vergonzosas subidas del precio de las entradas en el mercado de la reventa. La funcionalidad de registro de entradas de GET funciona con un SmartContract escrito en solidity, su código fuente es de código abierto [\[10\]](#). La blockchain que utiliza para apoyarse es la de *Ethereum*, y disponen de **Token** propio para realizar las transacciones[\[63\]](#). GET guarda un historial para cada tickets utilizando **IPFS**[\[32\]](#) que luego guarda en la red blockchain con el smart contract, *IPFS, es un sistema distribuido que permite guardar y recuperar archivos, webs, aplicaciones... y pretende superar a HTTP para construir una web mejor para todos*.

### 2.3.2. Lifeld

Esta aplicación esta construyendo una plataforma de identidad digital segura y basada en blockchain. Con una sencilla aplicación móvil te ofrece el control sobre la gestión de tú identidad digital. Permite iniciar sesión en cualquier sitio, entrar en cualquier edificio en el que se requiera de autenticación o

participar en transacciones basadas en la identidad (como puede ser en un sistema de votación) todo con la tranquilidad, fiabilidad y control que da la blockchain.

Para ello utiliza la red blockchain de **ArcBlock**<sup>[4, 5]</sup>. ArcBlock es una plataforma para desarrollar aplicaciones en blockchains o **DApps**<sup>[38]</sup>.

### 2.3.3. Voatz

Voatz<sup>[2.13]</sup> es una plataforma electoral móvil que permite a los ciudadanos votar sin tener que acudir a su colegio electoral o presentar una papeleta por correo. Voatz aprovecha las características de seguridad integradas en las últimas versiones de la tecnología de los teléfonos como es la biometría. Y también aprovecha la seguridad, transparencia e inmutabilidad de la red blockchain para garantizar la seguridad de cada voto. Desde el 2016 han emitido más de 110000 votos en mas de 67 elecciones.



Figura 2.13: Logo de la aplicación Voatz.

## 2.4. ANDROID

Android<sup>[66]</sup> es un sistema operativo móvil basado en el *Linux Kernel*. Fue diseñado para dispositivos móviles inteligentes, con pantallas táctiles, disponible para tablets, relojes inteligentes, incluso hay versiones de Android para el software de automóviles<sup>[2]</sup>. Android es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado superior al 90 % en 2018. Su código fuente es libre para que cualquiera pueda consultarla o contribuir a él. Actualmente se encuentra en la versión **11** internamente conocido como “Red Velvet Cake” pero son pocos los móviles con opción a ella, a la hora de desarrollar una aplicación para dispositivos Android, lo ideal es elegir la versión **6** también conocido como “Oreo” la cual puede ejecutar una gran mayoría.

### 2.4.1. Conceptos Básicos de Android

A la hora de programar aplicaciones en Android, lo ideal es trabajar en paralelo con la documentación para desarrolladores<sup>[14]</sup>. En ella se detalla el completo funcionamiento y diferentes APIs disponibles para programar las aplicaciones. Para este apartado resumo algunos de los conceptos de la guía, siendo una gran recomendación si se quiere profundizar más ir directamente a la guía.

Las aplicaciones de Android se pueden escribir con **Kotlin, Java y C++**<sup>[15, 17, 18]</sup>. Una vez escrito el código fuente, las herramientas de Android SDK compilan el código generando un paquete o APK, el cual incluye todos los contenidos de la aplicación Android y permite ser ejecutado en un dispositivo móvil (el dispositivo requerirá de la versión Android mínima para poder ejecutar.).

Cada aplicación de Android reside en su propia “zona virtual”, un conjunto de factores permiten tener aisladas las aplicaciones Android del resto del móvil. Entre otras, puesto que Android reside en un sistema Linux, el cual es multiusuario, cada aplicación en el móvil es un usuario el cual puede acceder solo a sus archivos y documentos teniendo sus propios permisos como usuario dentro del dispositivo y creando los grupos que necesite. Cada proceso que se ejecuta tiene su propia máquina virtual, ejecutándose el código de forma independiente al resto de aplicaciones. Es el sistema operativo el

que se encarga de mantener los distintos procesos, así como arrancar procesos que sean requeridos por la aplicación.

Un ejemplo, si desde la aplicación de galería queremos compartir una foto por Whatsapp, la aplicación “galería” le pedirá al SO que ejecute una actividad de “Whatsapp” y será el SO quien se encargue de decidir si tiene permiso para eso o no, o si tiene recursos para ejecutarlo o no. En ningún momento la aplicación “galería” tiene libertad para ejecutar otros procesos.

De forma predeterminada las aplicaciones solo tiene derecho de usar sus propios archivos, pero puede pedir permiso al sistema operativo para guardar información en el dispositivo, en el caso de mi aplicación Android por ejemplo, el registro de claves para acceder a la blockchain se guardan de forma local en el dispositivo. También, la aplicaciones pueden pedir permiso para utilizar cámara, conexión bluetooth...

Por último, algo muy importante a tener en cuenta al programar aplicaciones Android es que no puedes hacer operaciones pesadas en el hilo de ejecución principal. Todas las ejecuciones pesadas deben hacerse en otro hilo y con un callback recuperar la información si es necesario. Por ejemplo, al hacer llamadas a una API estas llamadas deben hacerse en otro hilo de ejecución diferente. Android no permite que bloquee la interfaz de usuario con operaciones pesadas (como una llamada a una API).

## Componentes de la aplicación

Las aplicaciones Android tienen:

- Actividades
- Servicios
- Receptores de emisiones
- Proveedores de contenido

### Actividades

Las **Actividades** son el punto de entrada de interacción con el usuario. Se representan como una pantalla con una interfaz de usuario. Por ejemplo, Whatsapp tiene una actividad que es la pantalla en la que todos los grupos y gente con la que hablas, y al entrar en un grupo, Whatsapp ejecuta una actividad diferente. Las actividades trabajan juntas pero son independientes entre si, brindando la posibilidad de llamar a una actividad concreta desde otra aplicación. Si quieras compartir una foto con un grupo en Whatsapp, desde la app de galería seleccionas la foto y llamas a la actividad de Whatsapp del grupo de amigos concreto con quienes quieras compartir la foto. Las actividades permiten:

- Realizar un seguimiento de la pantalla que esta viendo el usuario.
- Permitir regresar a actividades anteriores (actividades detenidas) a las que puede volver el usuario si lo desea, priorizando entonces algunos procesos mas que otros.
- Permitir finalizar procesos, volviendo a la actividad anterior.
- Permitir ser llamadas desde otras aplicaciones (como el ejemplo de compartir una imagen)

### Servicios

Los **Servicios** son un punto de entrada general que permite mantener en ejecución una aplicación en segundo plano y no proporcionan una interfaz de usuario. Un servicio puede reproducir música, sincronizar datos... Además los servicios pueden dividirse en 2, **servicios iniciados** y **servicios enlazados**. Los iniciados son servicios que inicia el usuario como reproducir música dejando luego el proceso en segundo plano. Y un servicio enlazado sería cuando una aplicación hace uso de otra para alguna tarea. Esta segunda aplicación que esta siendo usada, se ejecuta en segundo plano como

servicio enlazado.

### Receptores de emisiones

Los **receptores** permiten que el sistema entregue eventos a una aplicación fuera del flujo habitual. El sistema puede entregar emisión de aplicaciones que no están en ejecución. Por ejemplo, una aplicación programa una alarma a una hora determinada, aunque la aplicación no esté en ejecución, la alarma sonará a la hora establecida. Muchas de las emisiones, vienen del propio sistema. Por ejemplo, pantalla apagada, batería baja, captura de pantalla... Los receptores de emisión no disponen de interfaz de usuario, pero a través de la API de Android pueden mostrar mensajes en la barra de tareas.

### Proveedores de contenido

Los **proveedores de contenido** administran conjuntos compartidos de datos de la aplicación que pueden ser almacenados en el sistema de archivos, en una BDD o en la web. A través del proveedor de contenido, otras aplicaciones pueden acceder a esos datos (siempre que tengan permiso). Por ejemplo, la aplicación de contactos del móvil, tiene un proveedor de contenido, lo que permite a otras aplicaciones pedir acceso a la agenda de datos (siendo el usuario el que acepta que la aplicación acceda a los contactos). También son útiles para leer y escribir datos privados que no quieras compartir con otras aplicaciones.

### Activación de componentes

Un aspecto exclusivo de Android es que cualquier aplicación puede iniciar un componente de otra aplicación. Si quieres que el usuario tome una foto, no hace falta desarrollar la comunicación con el hardware de la cámara, sino que simplemente puedes llamar a la aplicación de la cámara la cual te devolverá la foto que el usuario tome. Por ejemplo, si inicias la actividad de la cámara, la actividad se ejecuta en el proceso de la cámara no en tu aplicación. Por lo tanto, a diferencia de lo que sucede en las aplicaciones de otros sistemas, aquí no existe un método principal o `main()`. Android no tiene un único punto de entrada.

De los cuatro tipos de componente, actividades, servicios y receptores se activan mediante un mensaje asíncrono denominado `intent`, estos vinculan componentes entre sí durante el tiempo de ejecución. Son algo así como mensajeros. Sin embargo, los proveedores de contenido se activan con solicitudes de un `ContentResolver`. Además, existen para cada componente métodos independientes para activarlos según el objetivo que se busque.

### El archivo de manifiesto

Para que Android pueda iniciar un componente, debe reconocer la existencia de ese componente leyendo el archivo de manifiesto de la aplicación, `AndroidManifest.xml`. El manifiesto puede hacer lo siguiente:

1. Declarar los componentes de la aplicación
2. Identificar los permisos de usuario que requiere la aplicación (acceso a internet, o acceso a los contactos)
3. Declarar características hardware y software así como nivel de API mínimo.
4. Declarar APIs a las que la aplicación necesita estar vinculado (como la biblioteca de google maps)

### Recursos de la aplicación

Las aplicaciones Android se componen de mucho más que solo código. Disponen de imágenes, vídeos, fuentes, audios y otros elementos como las interfaces de usuario en XML. Los recursos facilitan la actualización de las características de la aplicación para permitir el cambio de idioma, de fuente, de

tamaño según el dispositivo...

Por cada recurso, el SDK define un ID con número entero único para poder hacerle referencia. Estos IDs se utilizan en el código, por ejemplo, para añadir una imagen puedes hacer referencia al ID de la imagen asignado por el SDK. Una de las ventajas de utilizar los recursos, es facilitar la traducción de la aplicación a otros idiomas, también puedes mostrar parte de la interfaz de usuario si el usuario tiene una suscripción concreta u otra. En el caso de mi aplicación móvil, utilizo esto para mostrar una aplicación diferente a los alumnos y profesores.

## 2.5. DESARROLLO DE MICROSERVICIO EXTERNO

Para poder desarrollar la aplicación móvil y el SDK de este TFG, se ha desarrollado previamente una API de microservicios la que centralizar las llamadas a una base de datos y a una red blockchain, básicamente es para agilizar las llamadas y tratamiento de datos por parte del dispositivo móvil, así como minimizar la cantidad de código, tamaño de la aplicación móvil, dependencias del proyecto... La API de microservicios con la que se comunica la aplicación móvil se ha desarrollado utilizando **NodeJS** y ha sido documentada utilizando **Swagger 2.14, 2.15**.

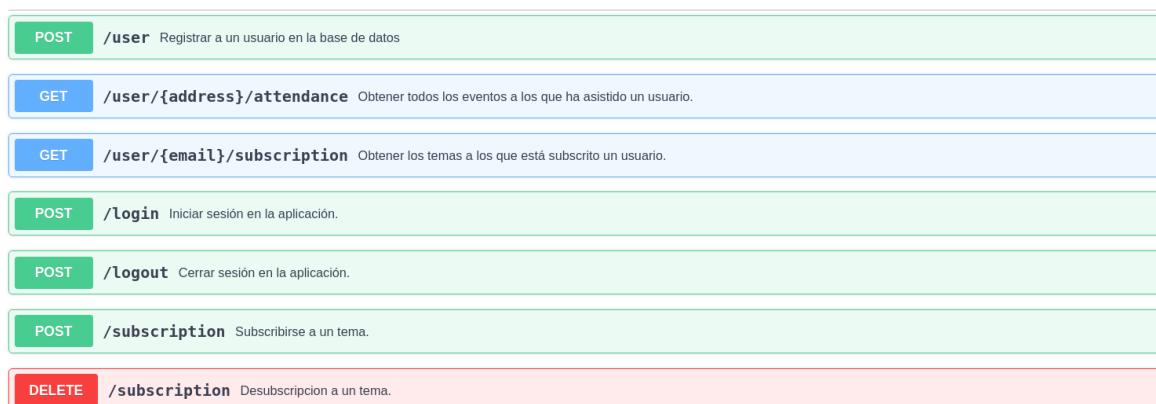


Figura 2.14: Fragmento de documentación en Swagger del Proyecto

The screenshot shows the detailed configuration for the **POST /user** endpoint:

- Parameters** (None listed)
- Try it out** button
- Name** **Description**
- user \* required** Datos del nuevo usuario.  
**object (body)** Example Value | Model
- JSON Input Example:**

```
{
  "nombre": "string",
  "password": "string",
  "apellido1": "string",
  "apellido2": "string",
  "correo": "string",
  "matricula": "string",
  "id_huella": "string",
  "wallet": "string"
}
```
- Parameter content type**: application/json

Figura 2.15: Fragmento de documentación de crear usuario

## NodeJS

Node.js[77] es un entorno de ejecución de **JavaScript** construido con el motor de *JavaScript V8 de Chrome*. Esta ideado como un entorno orientado a eventos asíncronos con el que se pueden diseñar aplicaciones escalables en la red. Permite tratar múltiples conexiones simultaneas sin que el programador tenga que preocuparse por la concurrencia. Hoy en día, los modelos de concurrencia usan hilos del sistema operativo y requieren de bloquear procesos para operaciones de lectura y escritura. Los usuarios de NodeJS están libres de preocuparse por el bloqueo del proceso, ya que no existe. De ahí que sea muy propicio desarrollar sistemas escalables con NodeJS, de todos modos aunque NodeJS trabaja sin hilos, se pueden aprovechar múltiples núcleos en su entorno, generando varios procesos.

Se ha decidido utilizar NodeJS para desarrollar la API de microservicios, pues nos da la seguridad de que responderá correctamente a miles de llamadas simultaneas (siempre y cuando el servidor en el que se este ejecutando pueda con el poder de cómputo que ello requiere). Por lo tanto, permite estar tranquilos y saber que el sistema no se caerá y podrá dar servicio un largo tiempo. Además, NodeJS junto con su gestor de paquetes *npm* permiten mantener el sistema actualizado con facilidad sin preocupeaciones de problemas de dependencias y paquetes desactualizados o con problemas de seguridad.

Algunos ejemplos de grandes proyectos que utilizan NodeJS son:

- Netflix
- Trello
- PayPal
- LinkedIn
- Uber

## Swagger

A la hora de programar una API, no es solo importante que el código sea correcto y funcione correcta y eficientemente. Sino que también es muy importante la documentación de la API, esto permite a otros programadores poder utilizarla sin preocuparse del código, y sin necesidad de conocer el proyecto. Para ello, se ha utilizado **Swagger** [79]. Swagger es un conjunto de herramientas profesionales y de código abierto las cuales ayudan a diseñar y documentar APIs de forma escalable. Swagger permite hacer varias cosas:

- Diseñar
- Desarrollar
- Documentar
- Testear
- Virtualizar
- Monitorizar

Para este proyecto, se han aprovechado principalmente las herramientas de *diseño, desarrollo y documentación*. Con la concentración sobre todo en la comodidad de documentación que tiene al utilizar archivos *YAML* (*yaml* es un formato de serialización de datos legibles por humanos) para generar una documentación ligera, fácil de entender y muy detallada.

## Funcionalidades de la API de Microservicios

Las comunicaciones de la API se pueden dividir en los siguientes 3 casos de uso generales 2.16:

Siendo más concretos, pasemos a listar todas las funcionalidades de las que dispone la API, lógicamente esta no es la documentación completa la cual trae consigo los objetos JSON que aceptan la URIs y los errores u objetos JSON que devuelve cada una de las URIs. En el presente la documentación

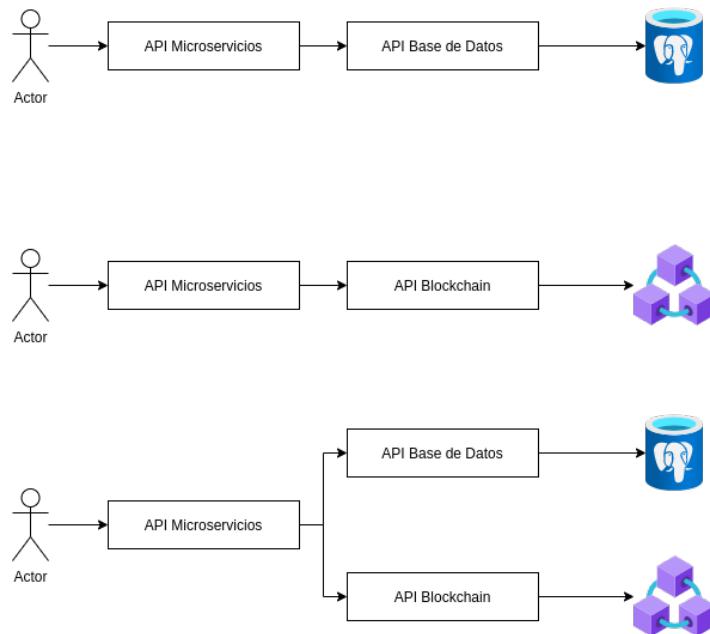


Figura 2.16: Comunicaciones de la API

se mantiene en un repositorio de *gitlab* privado.

<b>POST</b>	/user:	Permite registrar a un usuario en la base de datos.
<b>GET</b>	/user/address/attendance:	Permite obtener los eventos a los que ha asistido un usuario.
<b>GET</b>	/user/email/subscription:	Permite obtener los temas a los que está suscrito un usuario.
<b>POST</b>	/login:	Permite iniciar sesión (se comprueba que el usuario existe y se comprueba la contraseña hasheada).
<b>POST</b>	/logout:	Permite detectar que un usuario ha cerrado sesión.
<b>POST</b>	/subscription:	Permite a un usuario suscribirse a un tema.
<b>DELETE</b>	/subscription:	Permite a un usuario desabonar su suscripción de un tema.
<b>GET</b>	/topic:	Permite obtener la lista de temas disponibles.
<b>GET</b>	/topic/id/event:	Permite obtener un listado de los eventos de un tema concreto.
<b>GET</b>	/topic/id/subscription:	Permite obtener un listado con las suscripciones de un tema.
<b>POST</b>	/event:	Permite obtener los datos de la transacción de crear un evento, esta es firmada por el usuario a través de la aplicación Móvil y después enviada a la red blockchain.
<b>GET</b>	/event/id:	Permite obtener los datos de un evento.
<b>PUT</b>	/event/id:	Permite obtener los datos de la transacción para modificar un evento, también se firma y envía desde el móvil a la red blockchain.
<b>GET</b>	/event/id/delete/organizer:	Permite obtener los parámetros de la transacción para cerrar un evento, se firma y envía desde el móvil.
<b>GET</b>	/eventCatalog:	Permite obtener un listado con los tipos de eventos.
<b>POST</b>	/event/id/validator:	Permite añadir a un nuevo validador a un evento.
<b>GET</b>	/event/id/validator/organizer:	Permite obtener un listado con los validadores de un evento.

POST	/event/id/attendance:	Permite obtener los parámetros de la transacción para registrar la asistencia a un evento, igual que antes, se firma y envía desde el móvil.
GET	/event/id/attendance:	Permite obtener la lista de asistencias a un evento.

Gracias a usar swagger, la API sigue la documentación de forma estricta, es decir, no acepta llamadas en las que no se envíen los parámetro especificados en la documentación. Si para crear un usuario se requiere de un campo “email”, si este campo no existe al hacer la llamada a la API, swagger se encarga de rechazar la llamada con un error de formato. Así se logra evitar que el backend tenga errores por variables indefinidas, nulas, o se guarden datos vacíos...

---

## CAPÍTULO 3

# Desarrollo

---

En este capítulo se hablará del desarrollo de la aplicación (diseño, funcionalidades, librerías, gestión del wallet, documentación...) que se ha estado realizando a lo largo del TFG, así como el registro de claves para poder interactuar con la red blockchain que se ha levantado. También, se explicará en que consiste el SDK desarrollado, cuales son sus funcionalidades y como poder utilizarlo en otras aplicaciones Android.

El desarrollo de la aplicación móvil, se ha enfocado únicamente a dispositivos Android. Lógicamente, en un futuro, se tendrá que adaptar una aplicación para otros sistemas operativos como iOS. O por el contrario reescribir el código con lenguajes “cross platform” para permitir el correcto funcionamiento nativo tanto en Android como en iOS, una buena opción es utilizar [flutter\[20\]](#).

### 3.1. ANÁLISIS

Antes de arrancar con el desarrollo de la aplicación, hay que tener claros los objetivos que tiene que cumplir “Estublock”, y que casos de uso va a tener la aplicación.

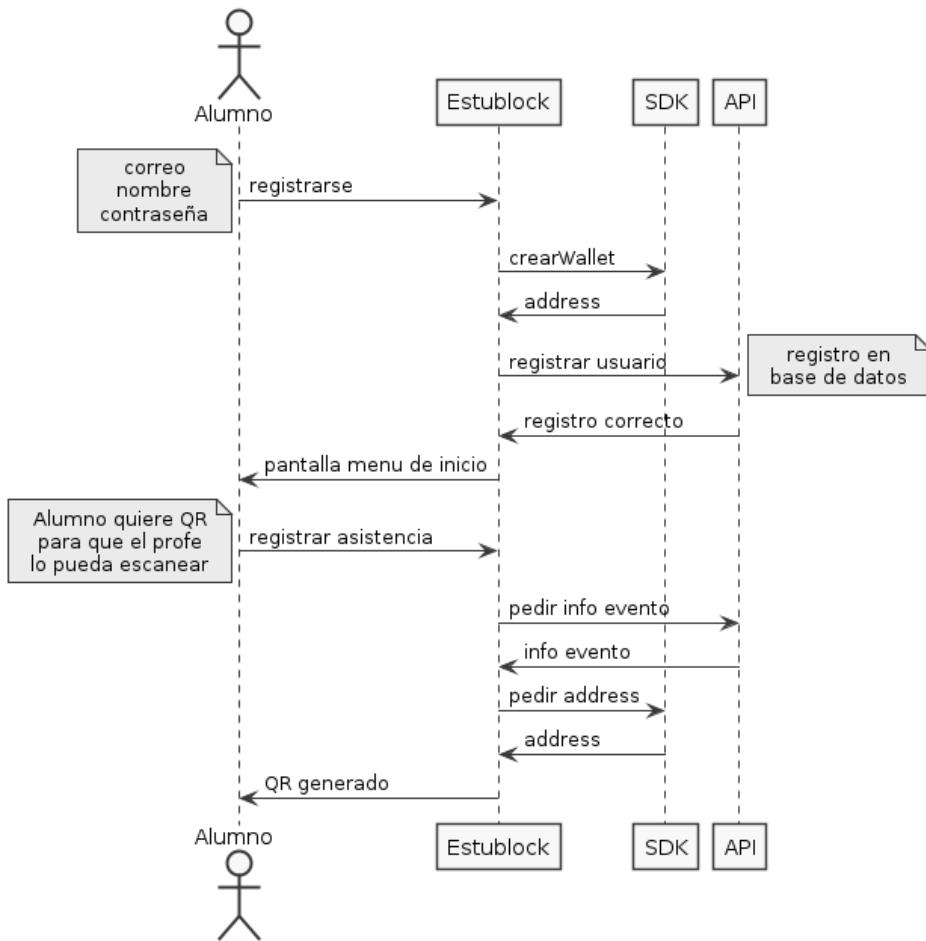
#### 3.1.1. Objetivos y público de Estublock

El **principal** objetivo de la aplicación *Estublock* es: *Permitir registrar asistencias a eventos*. Para cumplir con este objetivo, se va a utilizar la tecnología Blockchain, y aunque viene integrado con “Estublock”, la funcionalidad de comunicación con la red Blockchain se hace a través de un SDK. Dicho esto, para cumplir con el registro de asistencias, necesitamos que la aplicación permita registrar a usuarios, crear eventos y registrar a las personas en esos eventos. Por lo tanto, se requiere de una funcionalidad de login y registro, así como un lugar donde crear eventos y donde poder registrar a los usuarios en un evento dado.

La aplicación va principalmente dirigida a estudiantes, profesores y organizadores de eventos (dentro de este grupo, se ha pensado en las asociaciones de alumnos como *ACM* o *Delegación de Alumnos*). Como estos usuarios se mueven en el ámbito universitario, se requiere también de una funcionalidad que permita a los usuarios suscribirse a asignaturas. Con esto en mente, la decisión ha sido que “Estublock” funcione con un sistema de suscripciones. Es decir, alumnos, profesores y organizadores se suscribirán a temas (como asignaturas, congresos...) y una vez suscritos, los profesores y organizadores podrán crear eventos dentro del tema al que se han suscrito. Y todos los usuarios (profesores y organizadores incluidos) podrán asistir a los eventos del tema al que estén suscritos.

Aunque a continuación se van a analizar los casos de uso concretamente, veamos un caso de uso completo tanto desde el punto de vista de un alumno nuevo como de un profesor nuevo. En los siguientes diagramas se han obviado las suscripciones a temas, se profundizará más en el apartado siguiente, [Casos de uso](#). Por un lado tenemos el caso de uso del alumno [3.1](#) en el que un alumno se registra en la aplicación móvil y posteriormente genera un código QR que el profesor tendrá que

escanear para validar la asistencia de ese alumno al evento. Por otro lado, tenemos al profesor 3.2 el cual tras registrarse, crea un nuevo evento utilizando su wallet (pues tiene que firmar la transacción para enviarla a la red blockchain) y tras eso, escanea el QR del alumno y al igual que antes, firma y envía la transacción.



**Figura 3.1:** Caso de uso general del alumno

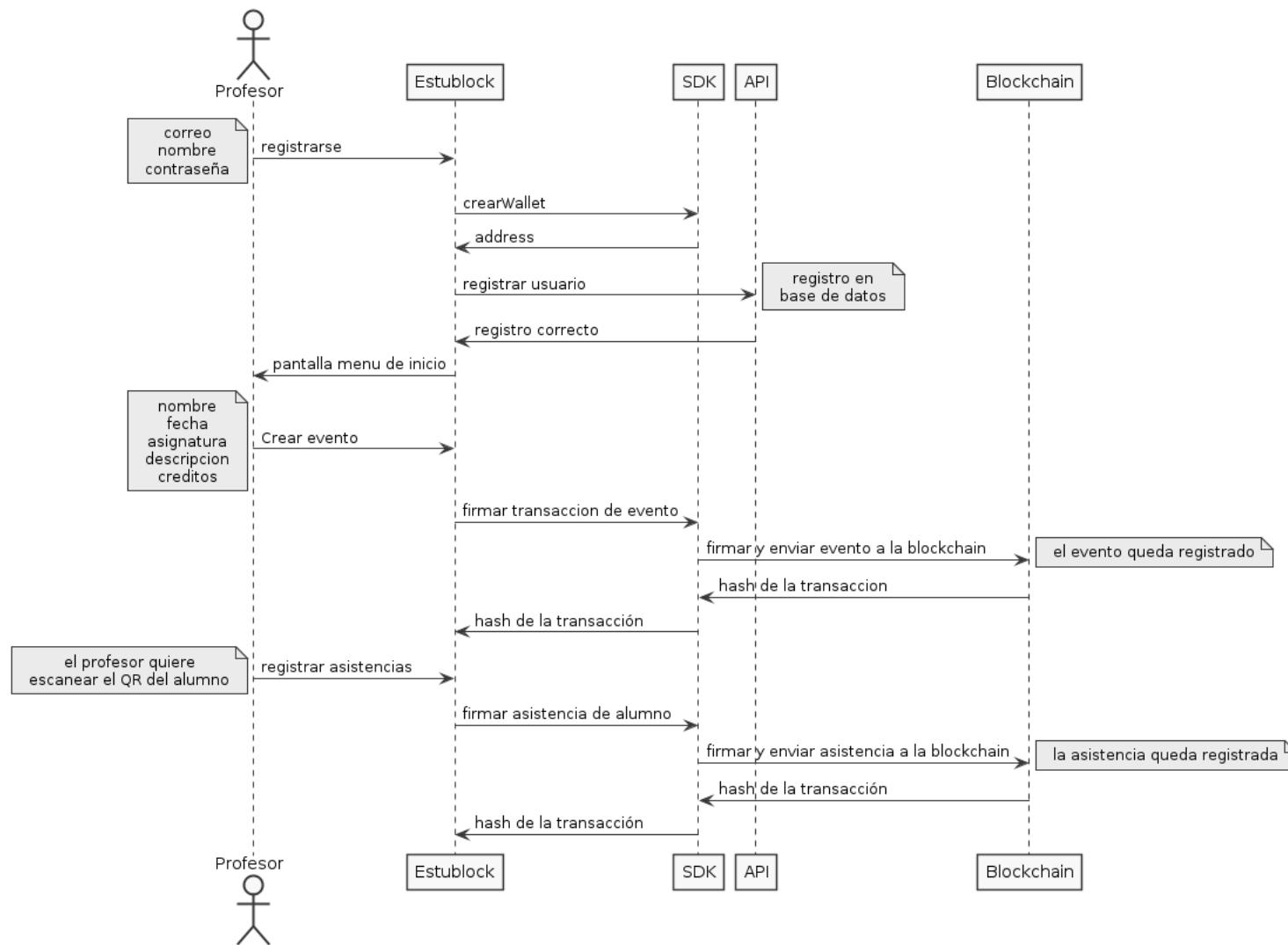


Figura 3.2: Caso de uso general del profe

### 3.1.2. Casos de Uso

Los diagramas con los casos de uso se encuentran al final de esta sección.

#### Registro

Para registrarse [3.3](#), el usuario introduce sus credenciales y la aplicación Estublock se encargará de crear el *wallet* y de llamar a la API que guarda los datos del usuario en la base de datos.

#### Login

Para hacer login [3.4](#) en la aplicación, se requiere del correo y la contraseña. Una vez la base de datos da el visto bueno de las credenciales, Estublock recupera con ayuda de las *SharedPreferences* (se verá más adelante) el directorio donde está guardado el wallet del usuario.

#### Suscripcion a Tema

Los usuarios deben suscribirse a temas [3.5](#) para poder asistir a los eventos de esos temas. Para ello, la aplicación Estublock le pide a la base de datos una lista de todos los temas existentes. De esos temas, el usuario podrá elegir los que le interesan y luego Estublock le pide a la base de datos que los guarde.

#### Creacion de un Evento

Profesores y organizadores tienen la opción de crear eventos [3.6](#), para ello, Estublock lista los temas a los que el usuario está suscrito. El usuario selecciona uno de los temas, introduce los datos del evento y cuando guarda esos datos, Estublock con ayuda del SDK firma (con la clave privada de Ethereum del usuario) los datos del evento y los envía a la red blockchain.

#### Asistir a un evento

Los usuarios que desean validar su asistencia a un evento [3.7](#), primero seleccionan el tema del evento, Estublock recupera la lista de eventos asociados a ese tema. Luego el alumno selecciona el evento concreto en el que se encuentran. Y Estublock se encarga de pedir al SDK el address del usuario y junto con los datos del evento genera un código QR que luego muestra al alumno. De este modo, en ese QR están los datos del evento y el address del alumno.

#### Registro de Asistencia

Para registrar una asistencia [3.8](#), Estublock utiliza la aplicación de la cámara de fotos para escanear los QR. Una vez se escanea un QR, se firman los datos de ese QR y se envían a la red blockchain con ayuda del SDK. Y se siguen escaneando QRs hasta que el usuario decida dejar de escanear QRs.

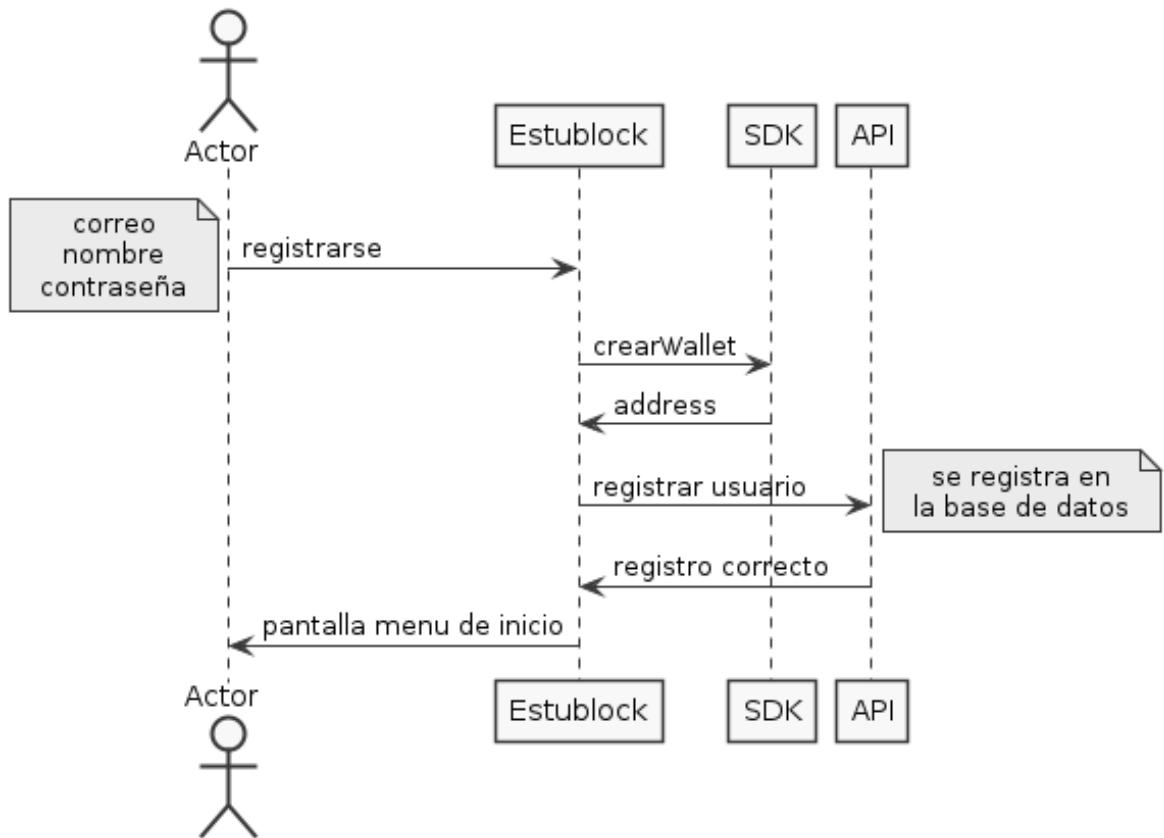


Figura 3.3: Caso de uso de registrar usuario

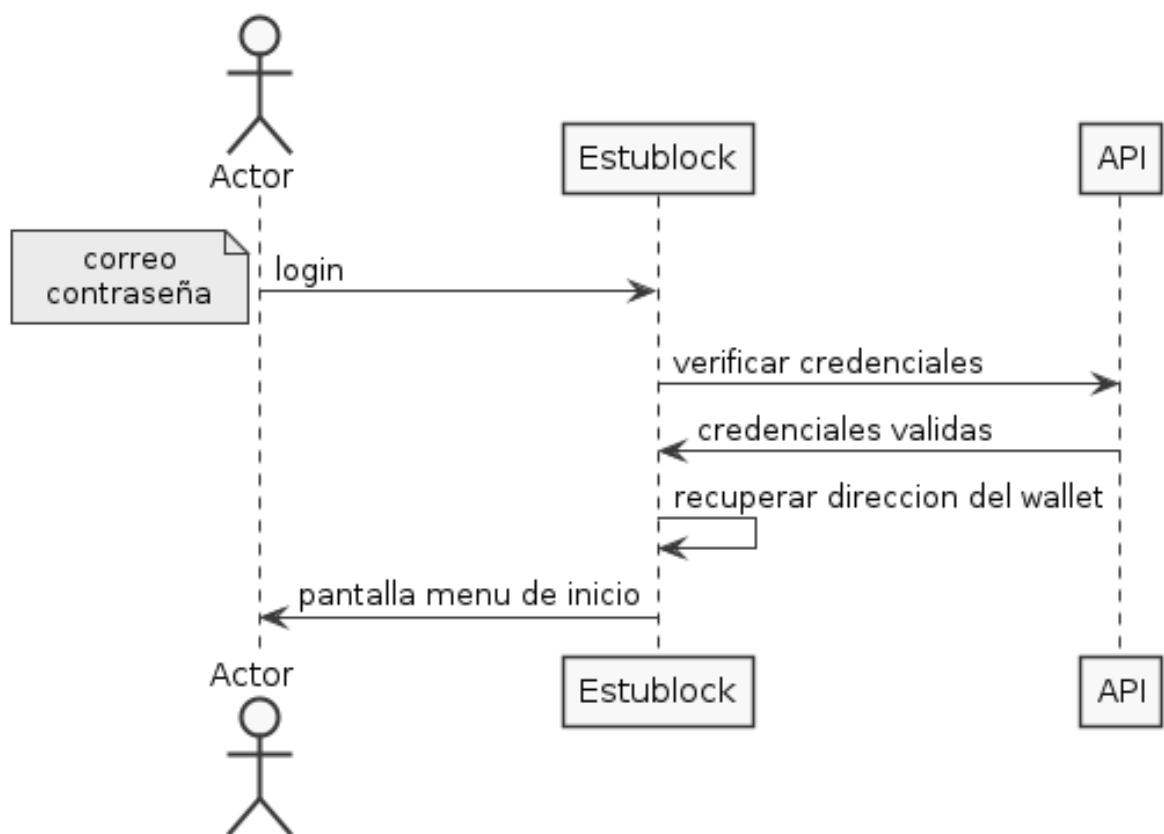


Figura 3.4: Caso de uso de login

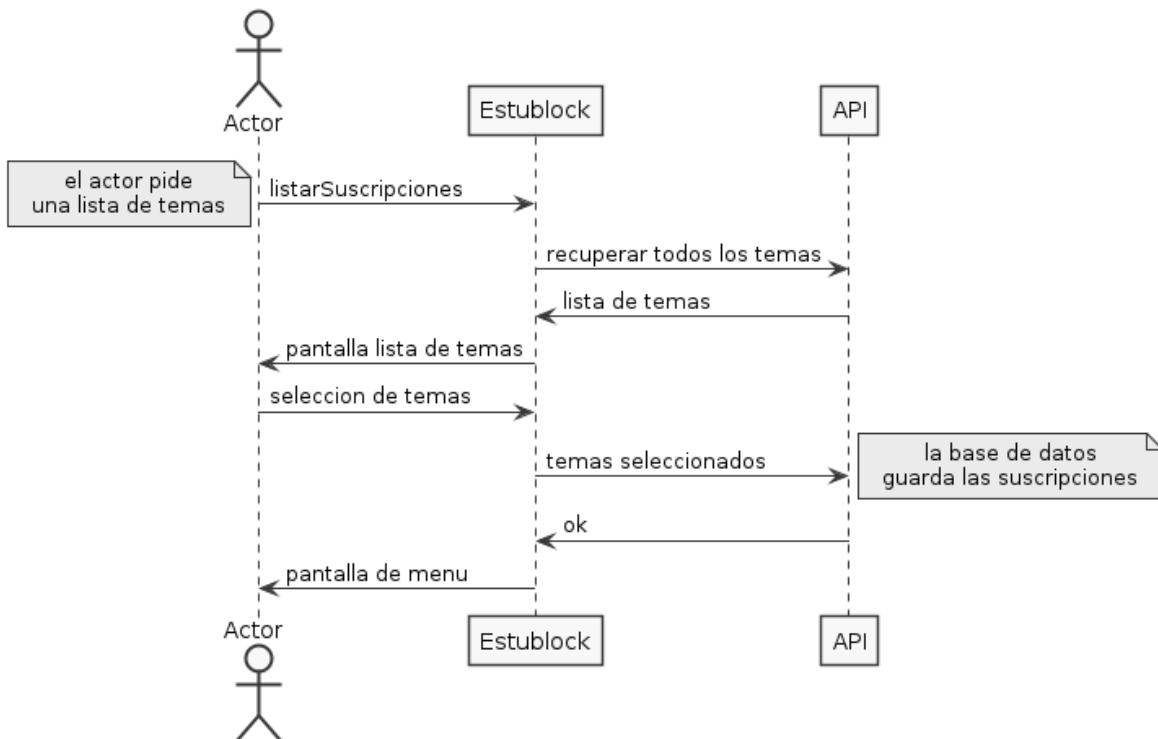


Figura 3.5: Caso de uso de suscripcion a temas

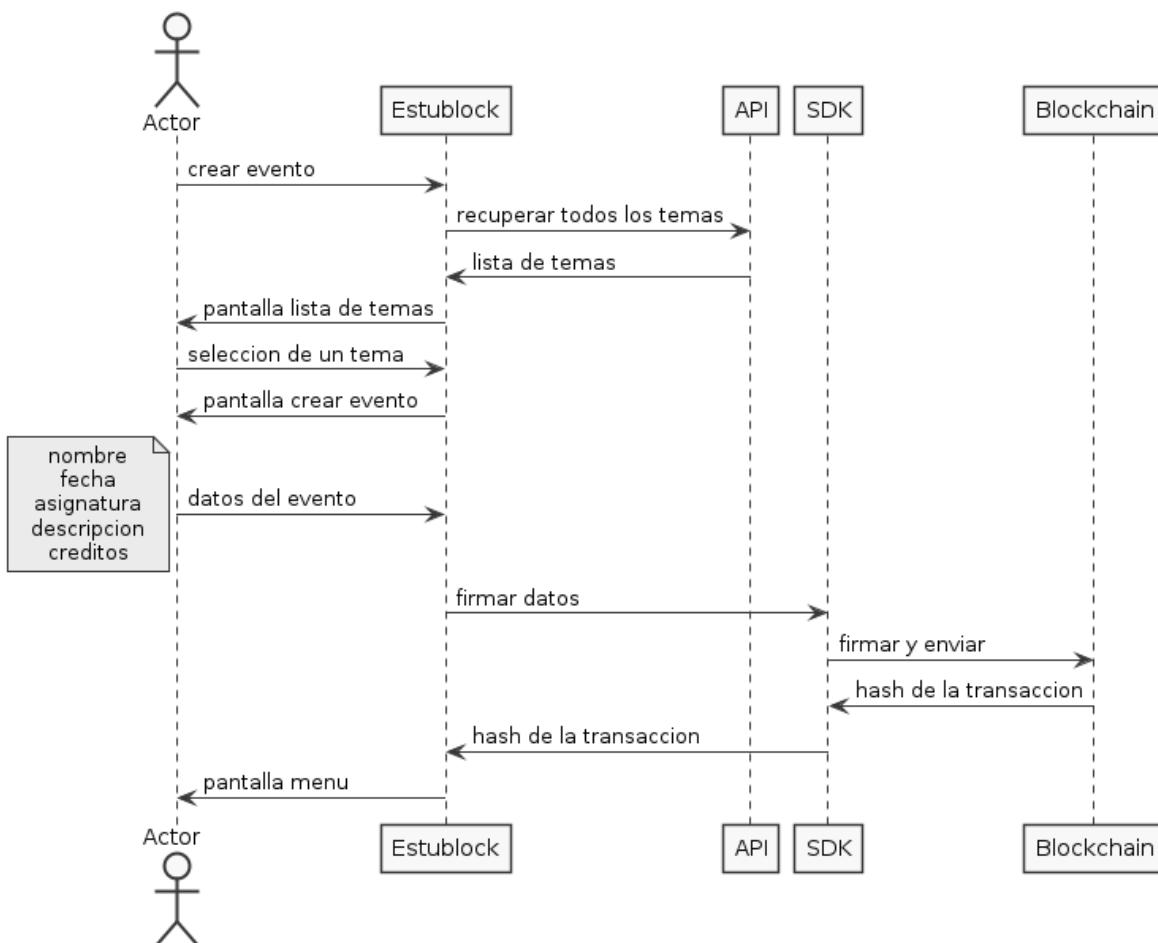


Figura 3.6: Caso de uso de crear evento

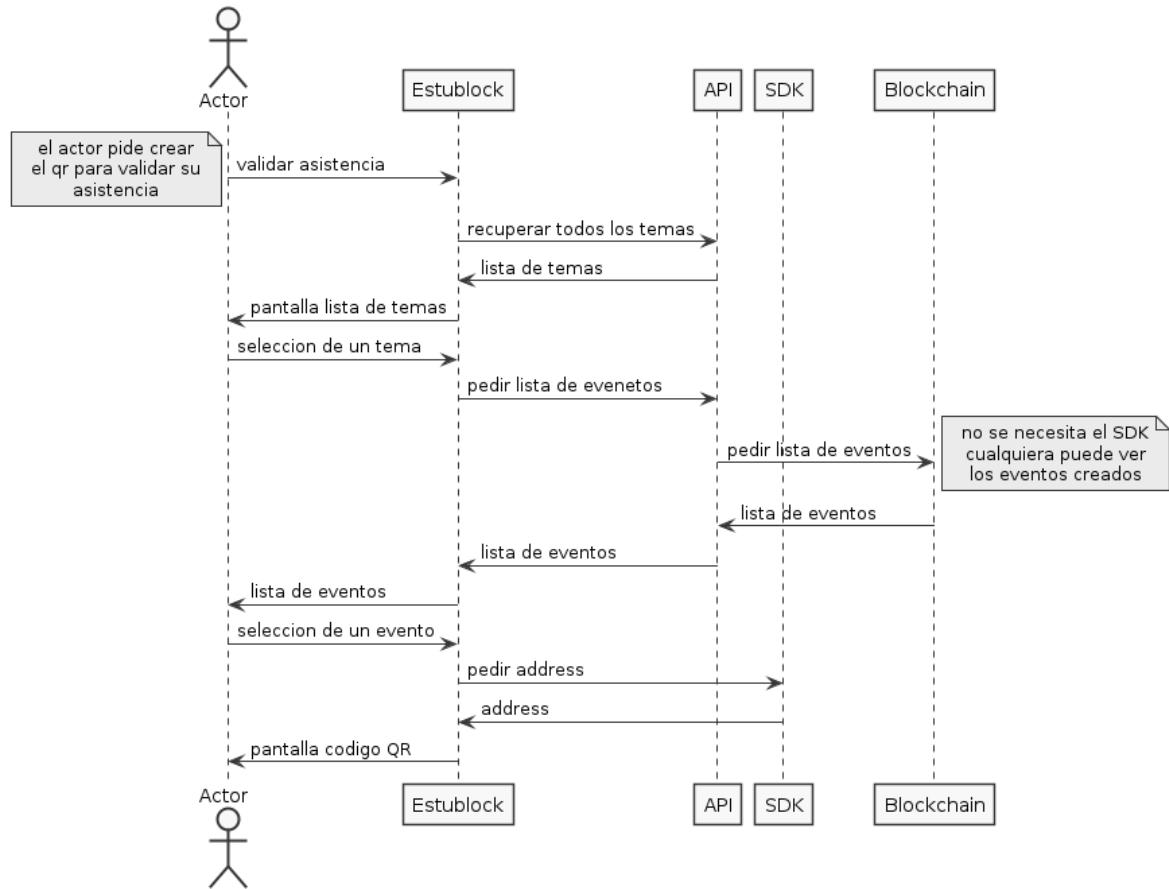


Figura 3.7: Caso de uso de crear QR

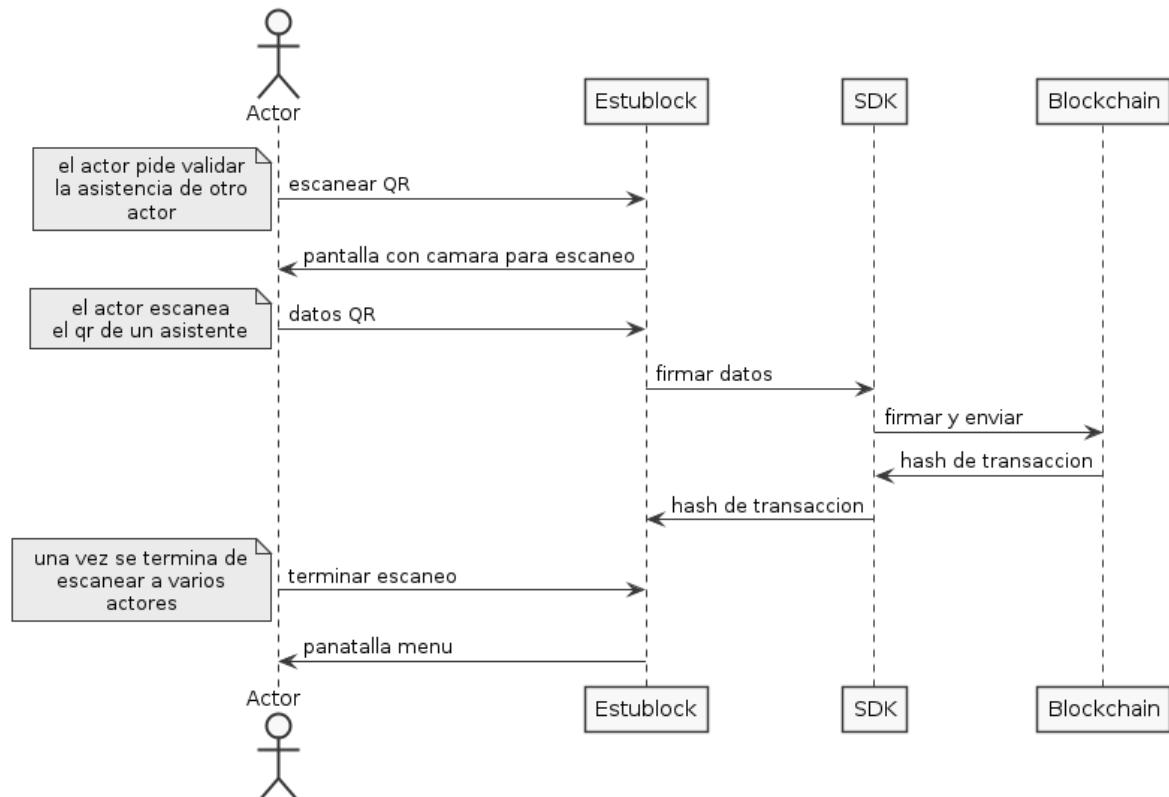
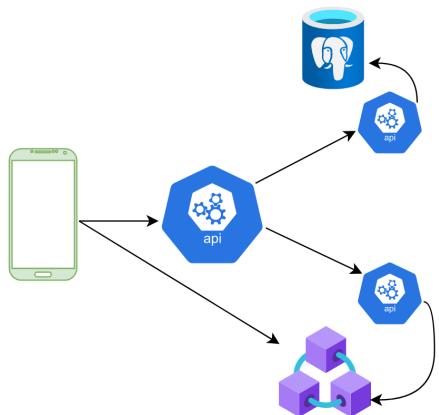


Figura 3.8: Caso de uso de escanear QR

### 3.2. ARQUITECTURA DE LA APLICACIÓN

Recordemos un poco el esqueleto del proyecto “Estublock”. Se ha decidido crear varios componentes modulares soportados por APIs, así cambios en el código de un componente no afectan al resto. El proyecto dispone de tres APIs diferentes. Tenemos primero la API que comunica con el servidor que tiene ejecutando la base de datos, luego la API que trata algunos datos con la red blockchain que se ha levantado. Y por último, una API intermedia que es con la que se comunica el dispositivo móvil para centralizar las llamadas, quitarle trabajo de cómputo al móvil y a la aplicación. Sin embargo, recordemos que el dispositivo móvil también hará llamadas directamente a la red blockchain a través del **SDK**. La arquitectura queda como se muestra en la figura 3.9.

El dispositivo móvil se comunica entonces con la api de microservicios con la ayuda de dos librerías que veremos más en profundidad en el apartado de [Código](#), esta API a su vez se comunica con la API de la base de datos o de la red blockchain según la operación que se haya especificado y estas se comunican con el servidor postgresql para la base de datos y la red de quorum para la blockchain.



**Figura 3.9:** Arquitectura Completa del proyecto

### 3.3. INTERFAZ DE USUARIO

La interfaz de usuario es donde los usuarios interactúan con la aplicación. Son las ventanas, botones, pantallas con las que el usuario interacciona. Deben tener un diseño intuitivo, fácil, que brinde una experiencia positiva. A más fácil de entender, mejor. Existen tres grandes tipos de interfaces de usuario, la interfaz de lenguaje natural, es la ideal y el sueño de todo usuario, pues permite comunicar humano y máquina con lenguaje natural. Un ejemplo de dispositivo que utiliza esta interfaz es *Alexa*, que cuenta con un software basado en modelos acústicos y del lenguaje. El segundo tipo de interfaz es la interfaz de preguntas y respuestas. En esta interfaz se muestra una pregunta al usuario y según su respuesta se actúa de una u otra manera. Un ejemplo de estas interfaces son los software de instalación, como puede ser el instalador de un sistema operativo. Y por último, la interfaz que más nos interesa para este proyecto, la **interfaz gráfica de usuario**, en inglés “Graphical User Interface” o **GUI**. Esta utiliza gráficos, imágenes, videos, iconos, menús... para permitir al usuario interactuar con la aplicación.

#### 3.3.1. Interfaces Gráficas en Android

En Android la interfaz de usuario se construye mediante una jerarquía de objetos, generalmente de tipo **View** y **ViewGroup**. Los **View** son componentes con los que el usuario puede interactuar, el usuario puede ver el componente en la pantalla. Sin embargo los **ViewGroup** son un contenedor

invisible que define la estructura de objetos *View* y otros *ViewGroup*, un árbol jerárquico puede verse en la figura 3.10.

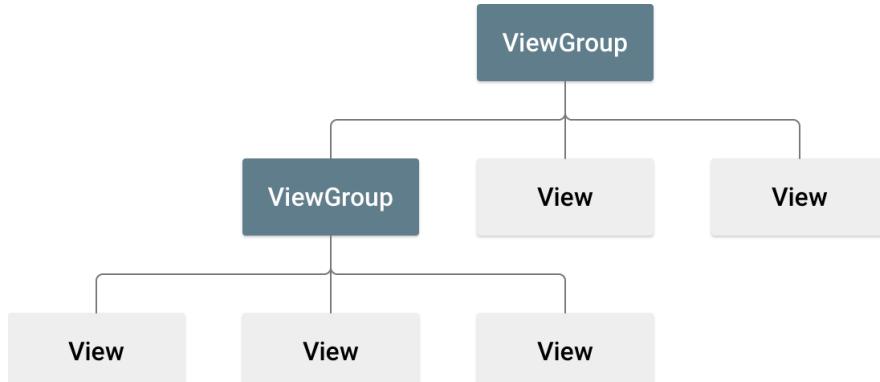


Figura 3.10: Jerarquía de una interfaz de Android

Los objetos *View* se denominan “widgets” y pueden ser Botones, Textos, “Switch”, “ScrollView”... Los objetos *ViewGroup* se denominan “diseño” y pueden ser “LinearLayout”, “ConstraintLayout”... Los diseños se pueden declarar de dos maneras, podemos declararlos con antelación, esto es programar en código XML como queremos que se vea una pantalla. Y también se pueden instanciar desde el código en tiempo de ejecución y modificar datos de la pantalla, o crear nuevos datos que no hay, (los datos pueden ser botones, cajas, menus, desplegables...).

Para la aplicación, se han usado ambas formas, pues por un lado hay pantallas estáticas, como pueden ser la pantalla de login o registro. En las que se sabe que botones tiene que haber, que textos tiene que mostrar y que diseño tiene con antelación. Pero también hay pantallas en las que se crean botones o texto de forma dinámica, por ejemplo, en pantallas que muestran próximos eventos, se muestra de forma dinámica (según el número de eventos que devuelva la API) más o menos botones.

Uno de los puntos fuertes de programar con antelación la interfaz con XML, es que se puede separar la presentación de la app del código que controla los componentes, además, facilita la creación de distintos diseños para diferentes tamaños de pantalla, orientación, colores... Es la mejor opción a la hora de crear una interfaz de usuario aunque en ocasiones se requiera de crear o modificar de forma dinámica en tiempo de ejecución algunos elementos. También es habitual crear con XML un diseño estático, y luego modificarlo en tiempo de ejecución según sea necesario accediendo a sus componentes.

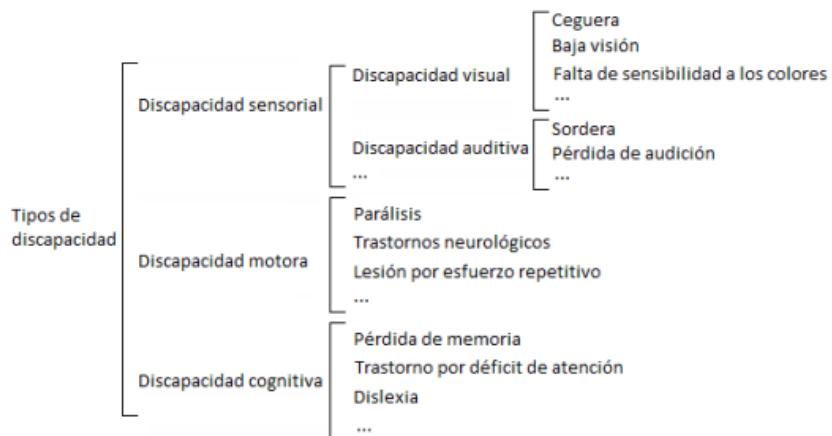
### 3.3.2. Accesibilidad de Estublock

Cuando se crea un sitio web o una aplicación ya sea de escritorio o de móvil, una de las principales preocupaciones que un desarrollador tiene (sobre todo a nivel de interfaz) es la **accesibilidad**. Accesibilidad es tratar a todo el mundo igual, sean cuales sean sus capacidades. Aunque la definición es muy abierta, lo que quiere es conseguir que todo el mundo pueda utilizar lo que sea sin importar cuales seas sus capacidades y dificultades. Con “lo que sea” nos referimos a que accesibilidad aunque aquí esté centrado en Software especialmente aplicaciones móviles, la accesibilidad es algo a tener en cuenta en todas partes, transporte, edificios, gimnasios, webs, videojuegos, aplicaciones móviles... Los desarrolladores de aplicaciones móviles, deben intentar ser más conscientes de las necesidades de todo el público potencial que puede querer utilizar la aplicación que se está desarrollando.

El *primer* paso para identificar como mejorar al máximo la accesibilidad de “Estublock” es definir quienes son los usuarios finales, que abanico de personas van a utilizar la aplicación, para así lograr en la medida de lo posible que todos los usuarios puedan utilizar y disfrutar de la aplicación. Dicho esto,

el público objetivo de la aplicación es toda persona entre los 18 años hasta personas muy mayores. Los jóvenes (de 18 para arriba) son los estudiantes, luego tenemos a ponentes de charlas que pueden ser jóvenes también o ser personas ya jubiladas. Por otro lado están los profesores, hay profesores jóvenes, y profesores más mayores. La edad no es un aspecto clave a tener en cuenta pero es cierto que no es lo mismo un público objetivo de 8 años que de 18 años.

Segundo paso, hemos mencionado que accesibilidad se refiere a que personas con distintas dificultades y capacidades puedan utilizar la aplicación. ¿Qué capacidades pueden limitar el uso de la aplicación “Estublock” por las personas? [3.11](#) Usuarios con **deficiencias visuales** que utilicen un lector de pantallas para moverse por la aplicación (es decir, pulsan un botón y el móvil lee el texto del botón, si vuelven a pulsar el botón se ejecuta la acción que haga ese botón). Luego hay usuarios con **miopía**, que tratarán de tener la fuente del texto más grande para facilitar su lectura, la **vista cansada** también entra aquí, pues es habitual agrandar el texto cuando uno tiene la vista cansada. Hay que tener en cuenta también a usuarios con **discapacidades motrices**, que utilizarán ayudas como dispositivos de señalización o punteros visuales para manipular la aplicación. Por ejemplo, un usuario que ha nacido sin extremidades superiores utilizará los pies para manipular un móvil. Aunque este usuario tendrá una habilidad con los pies que el resto no tenemos, agradecerá enormemente que las aplicaciones móviles tengan en cuenta la incomodidad que conlleva utilizar los pies para utilizar una aplicación móvil. Por otro lado, no todas las dificultades tienen un factor “físico”, hay que pensar también que hay usuarios **daltónicos**, y ante dos botones rojo y verde, el usuario daltónico vea ambos colores con la misma tonalidad.



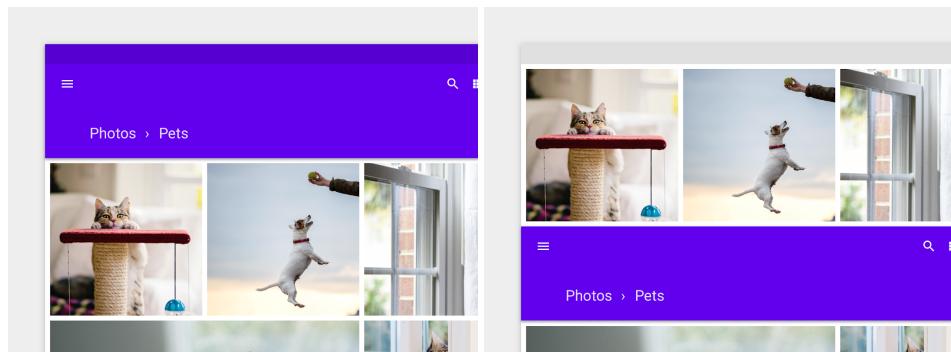
**Figura 3.11:** Clasificación de las principales formas de discapacidad

La clave para diseñar pensando en la accesibilidad, es pensar en un problema y decidir solucionarlo para más de un tipo de usuarios. Aunque la mejora manera de conseguir una solución bien hecha desde el principio es teniendo en cuenta todas las necesidades desde el primer momento. (Aunque siempre se pueden cambiar los planes y añadir algo que se pase haya pasado por alto). A la hora de desarrollar “Estublock” solo se ha tenido en cuenta la accesibilidad a nivel teórico, es decir, se tiene en mente la implementación de mejoras en accesibilidad, pero para lograr llegar al *deadline* del proyecto, se han obviado y se ha ido directo al funcionamiento de la aplicación sin tener en cuenta la accesibilidad.

Pasemos a ver, algunos puntos esenciales en los que hay que fijarse al añadir a “Estublock” una mejor accesibilidad.

## Jerarquía

Cuando la navegación es fácil, los usuarios entienden dónde están en la aplicación y qué es importante. Para enfatizar qué información es importante, se pueden utilizar múltiples señales visuales y textuales como el color, forma o movimiento. Cada botón, imagen y línea de texto añadidos aumentan la complejidad de una interfaz de usuario. Para mejorar la comprensión de la interfaz de usuario se pueden utilizar elementos claramente visibles, contrastes y tamaños adecuados, una clara jerarquía de importancia, información clave discernible de un vistazo... Así como colocar las acciones importantes en la parte superior o inferior de la pantalla, y colocando elementos relacionados con una jerarquía similar uno al lado del otro. Por ejemplo, en la siguiente figura 3.12 a la izquierda queda claro que en la parte superior están los elementos importantes que pueden permitirte navegar por la aplicación, pero a la derecha choca al usuario que los controles de la aplicación estén en medio de la pantalla, cortando la información.



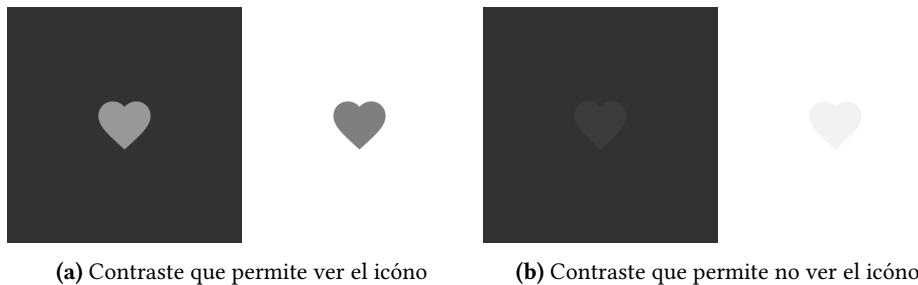
(a) Correcto: Acciones importantes en la parte superior  
 (b) Incorrecto: Acciones importantes en la parte de en medio

**Figura 3.12:** Jerarquía de prioridad

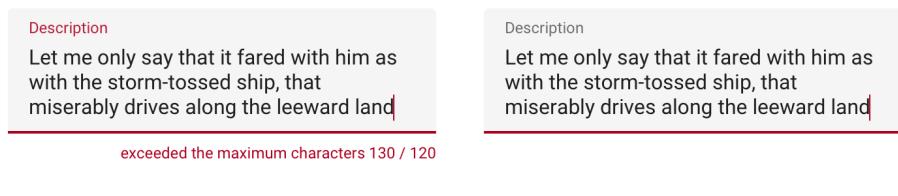
## Colores y Contraste

El color y el contraste pueden utilizarse para ayudar a los usuarios a ver e interpretar el contenido de su aplicación, interactuar con los elementos adecuados y comprender las acciones. Puede ayudar a comunicar el estado de ánimo, tono e información crítica. Colores primarios y secundarios pueden favorecer la accesibilidad. Al igual que el contraste de color entre elementos puede ayudar a los usuarios con baja visión a ver mucho mejor los elementos. Con el contraste de colores, los usuarios distinguen mejor los distintos elementos textuales y no textuales, puede hacer que imágenes se vean mejor, además, hay que tener en cuenta la luz externa, no es lo mismo utilizar la aplicación en un espacio cerrado que abierto, con sol o nublado, de día o de noche. Una de las principales ideas para “Estublock” es dar la posibilidad al usuario de personalizar el *tema* de la aplicación, para que pueda decidir entre un abanico de temas el que mejor se ajuste a sus necesidades. Si prefiere un tema oscuro por la noche y uno claro por el día, puede cambiar entre ambos temas, si le cuesta diferenciar los colores, podrá seleccionar un tema de colores que tenga mayor contraste o que utilice otro abanico de colores que sí pueda ver (muy útil para personas con *daltónicas*). La importancia del contraste puede verse en la figura 3.13.

Aunque el feedback de la aplicación no tenga tanto que ver con la accesibilidad sino con la usabilidad. Es cierto que una mejora en los colores del feedback puede hacer que un usuario vea con mejor claridad lo que está ocurriendo, o lo que la aplicación le está tratando de transmitir. Por ejemplo, ante un error, el móvil puede mostrar una pequeña línea roja que el usuario puede no ver. Pero, si está bien hecho, el móvil podría vibrar, a la vez que muestra en rojo el texto y la razón del error<sup>3.14</sup>. Para terminar una gran herramienta para elegir correctamente los colores de nuestra aplicación es [Color Tool](#)



**Figura 3.13:** Contrastos de un ícono



**Figura 3.14:** Feedback ante un error

### 3.3.3. Diseño de la Interfaz de Estublock

A la hora de diseñar la aplicación Estublock, se ha buscado un diseño ligero, intuitivo y rápido, para evitar que se pueda mal interpretar un botón, que el usuario se pierda al usar la aplicación o no sepa donde encontrar las cosas. Todo esto para evitar en la medida de lo posible que el usuario se lleve una mala experiencia con la aplicación. Para todo ello, se ha utilizado una herramienta de diseño para prototipos de pantallas y testeo de pantallas llamada *Marvelapp*[75]. Marvelapp permite diseñar con bastante detalle aplicaciones móviles y web, y además permite enlazar pantallas para probar la efectividad de las pantallas y el entendimiento de las mismas. Marvelapp tiene un gran potencial al permitir diseñar rápidamente pantallas y poder probar su efectividad rápidamente. El diagrama de flujo completo entre pantallas de la aplicación puede encontrarse en la figura 3.15

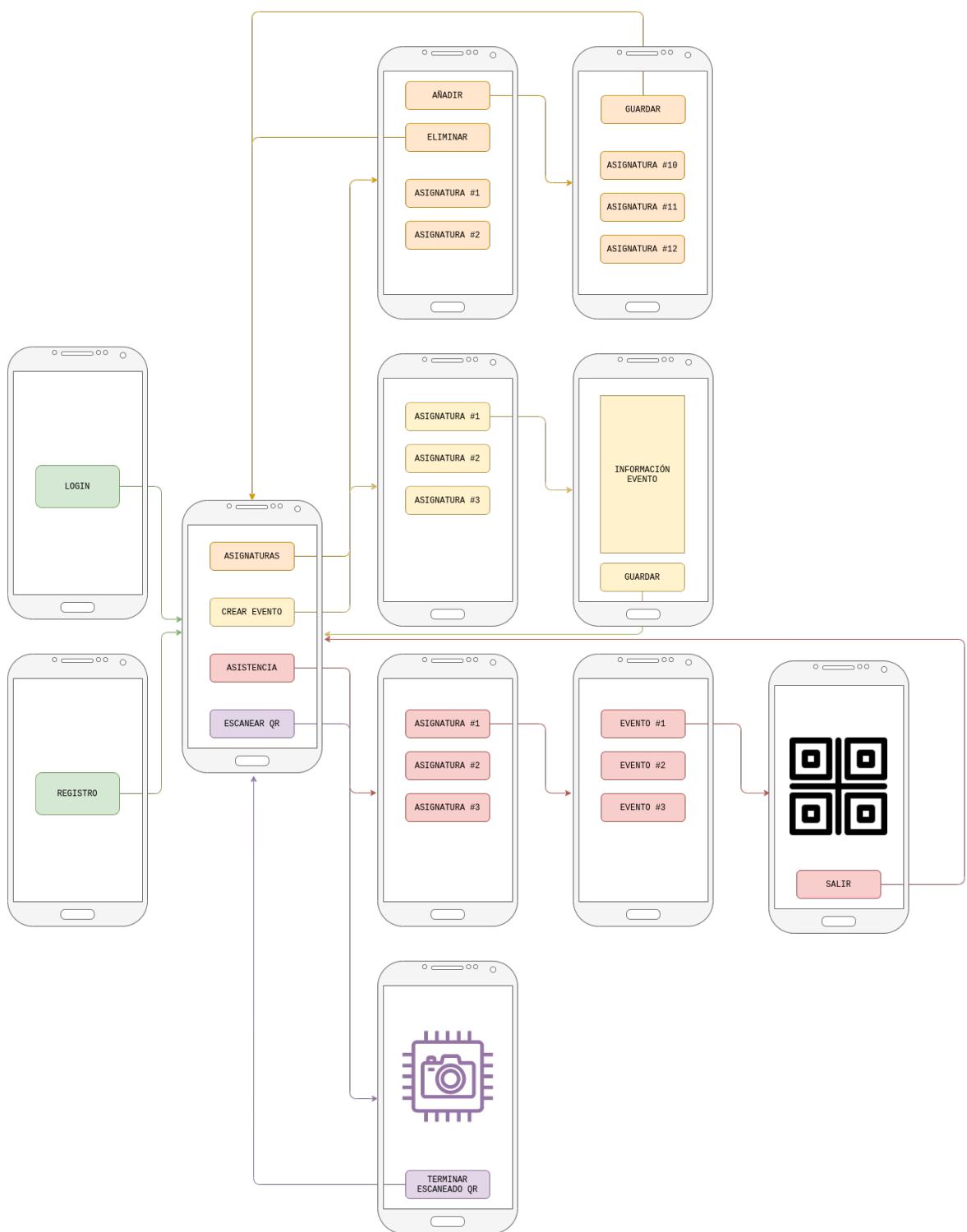
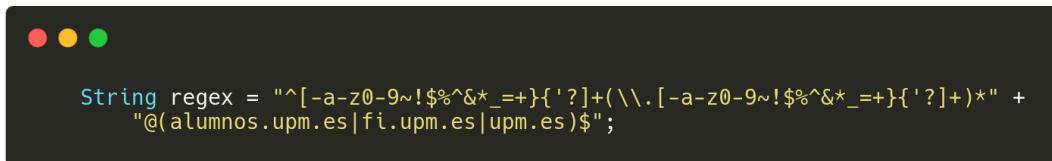


Figura 3.15: Flujo entre pantallas de “Estublock”

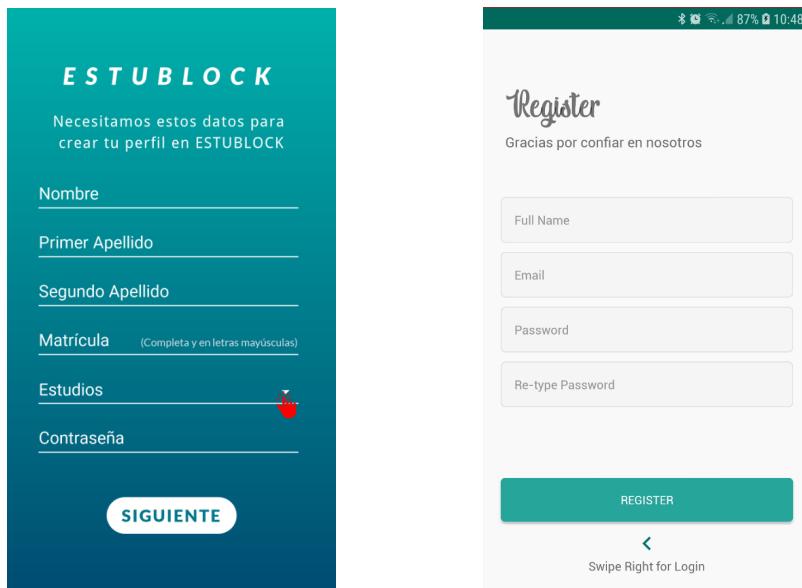
### Pantalla de Registro

En la pantalla de registro, se pide al usuario que introduzca sus datos personales. Se comprueba que los datos sean correctos, por ejemplo, al introducir el correo electrónico, un *regex* se encarga de verificar que el correo sea universitario. Los *regex* evalúan expresiones regulares y se puede buscar con él un patrón como puede ser `@alumnos.upm.es` para verificar que el usuario es por ejemplo un alumno. El *regex* que se ha utilizado para comprobar el correo se muestra en la figura 3.16.



**Figura 3.16:** Código regex de verificación de correo

También, se hace verificación de que la contraseña ha sido escrita correctamente dos veces. Esta es una **nueva** decisión de diseño, puesto que en un inicio como veremos más adelante, no se contemplaba en la pantalla de registro pedir la contraseña dos veces. Además, otros cambios que ha sufrido la pantalla de registro es en el número de datos que se le piden al usuario. Se ha eliminado la matrícula, esto es temporal, pues es probable que se añada en el futuro. La principal razón que nos llevó a tomar esta decisión, es que los profesores o ponentes de charlas que fuesen a utilizar esta aplicación, no disponen de matrícula. Solo los estudiantes tienen matrícula. Por otro lado, el nombre y apellidos se piden de forma conjunta, esto también es temporal pues hay que estudiar si van a ser o no datos fundamentales para poder acreditar al alumno la asistencia a un evento, o no es tan importante conocer el primer y segundo apellido del usuario, pues al final lo más importante es el correo electrónico. El diseño con *marvelapp* y el resultado final, pueden encontrarse en la figura 3.17



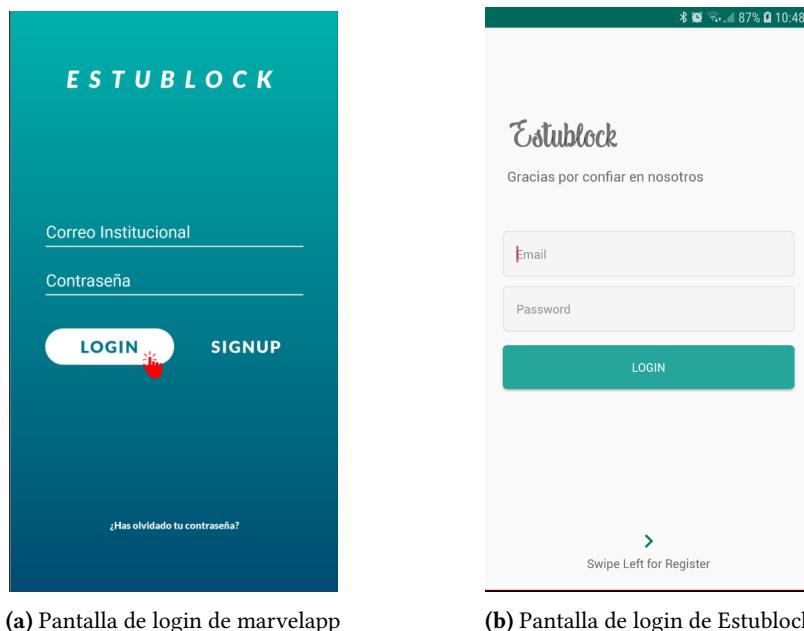
**Figura 3.17:** Pantalla de Registro

### Pantalla de Login

En la pantalla de login, los usuarios pueden iniciar sesión, desbloqueando además su wallet. Este término y sus detalles se verán en el apartado **Wallet**. Al hacer login, los usuarios solo necesitan introducir el correo electrónico con el que se dieron de alta y la contraseña. Como se puede observar

en la figura 3.18 en *Marvelapp* se añadió la opción de recuperar la contraseña del usuario. Sin embargo, no existe en la versión final.

Para poder recuperar la contraseña de un usuario, lo normal en un servicio es enviarle un correo electrónico al usuario (al correo que el usuario utilizó para darse de alta) y desde ese correo el usuario accede a un portal en el que puede modificar su contraseña. Y la contraseña modificada se pasa por una función *hash*[83] y se almacena en una base de datos. En el caso de Estublock, a pesar de poder implementar dicho servicio sin problema, y modificar en la base de datos el hash de la contraseña. No serviría de nada, pues la contraseña que se utiliza para desbloquear y utilizar el wallet no puede ser modificada, si el usuario pierde o se le olvida la contraseña, el wallet queda invalidado. Existen dos principales métodos para recuperar el acceso al wallet, estos se verán en el apartado *Wallet*. Por lo tanto, ante este problema, la pantalla de login final no tiene opción de recuperar las credenciales por ahora. Pues se puede implementar uno de los métodos de recuperación de wallets disponibles. Tanto la pantalla original como la final se pueden ver en la figura 3.18.



**Figura 3.18:** Pantalla de Login

### Menú Principal

El menú principal es el corazón de la aplicación. Pues desde ahí se abre camino al resto de funcionalidades de la aplicación. Por un lado es una pantalla importante, pues recoge gran parte de los componentes de la aplicación, pero a su vez, es muy básica y no tiene nada interesante, pues lo único que hace es redirigir al usuario a las funcionalidades disponibles. Estéticamente hablando, la pantalla que se diseño en *Marvelapp* ha perdido mucho valor visual, pues la pantalla resultante es más básica y menos colorida. Sin embargo, aunque la estética es importante, la funcionalidad principal del menú es redirigir al usuario y esto sí lo hace perfectamente bien. El diseño final así como los botones han sufrido muchos cambios. De hecho, excepto el botón de *asignaturas* el resto de botones es diferente. Esta decisión se ha tomado para agilizar el objetivo principal de la aplicación, es decir, queremos escanear QRs para validar la asistencia de usuarios a eventos. Por tanto, tenemos un botón *crear eventos*, un botón *asistencia* que genera el QR y un botón *Escanear QR* que permite escanear los QR. Desde el menú se puede hacer todo, los cambios se pueden ver en la figura 3.19.



(a) Pantalla de menú de marvelapp

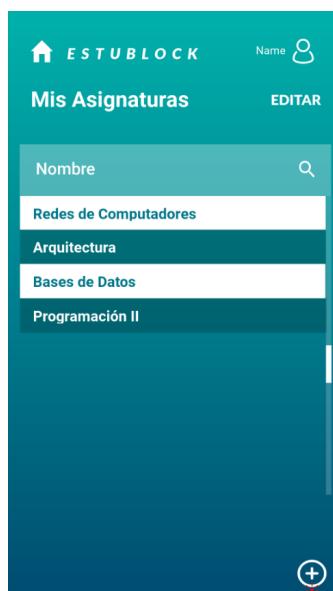


(b) Pantalla de menú de Estublock

**Figura 3.19:** Pantalla de Menú

### Pantalla de Suscripciones

En la pantalla de suscripciones el usuario puede ver la lista de asignaturas a las que está suscrito. También se le permite añadir y eliminar asignaturas, para que según avance el curso y vaya aprobando asignaturas pueda no verlas en la lista. Visualmente la pantalla de Marvelapp y la de Estublock han cambiado, pero las funcionalidades. Lo que sí se puede destacar, es que se ha puesto en la pantalla de Estublock un botón grande que pone *Añadir* y otro *Eliminar* con esto lo que conseguimos es que el usuario tenga claro donde puede ir a añadir asignaturas o eliminarlas. En la pantalla que se hizo con Marvelapp, el botón de añadir es pequeño y se encuentra en una esquina, y puede no verse con claridad. El diseño de la pantalla queda mostrado en la figura 3.20



(a) Pantalla de asignaturas de marvelapp

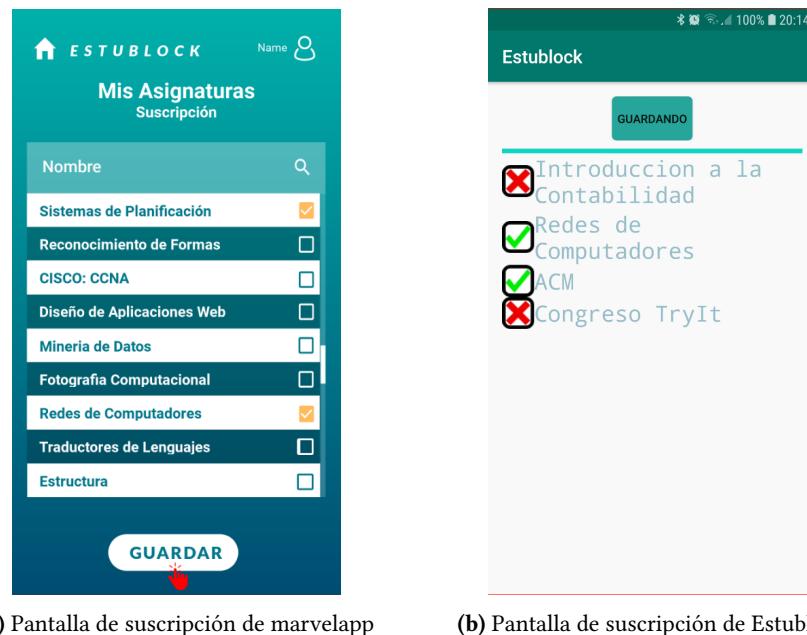


(b) Pantalla de asignaturas de Estublock

**Figura 3.20:** Pantalla de Asignaturas

Si el usuario decide añadir nuevas asignaturas a su lista, tendrá que darle al botón de *Añadir*. Esto le lleva a una nueva pantalla en la que se listan todas las asignaturas que hay disponibles para suscribirse.

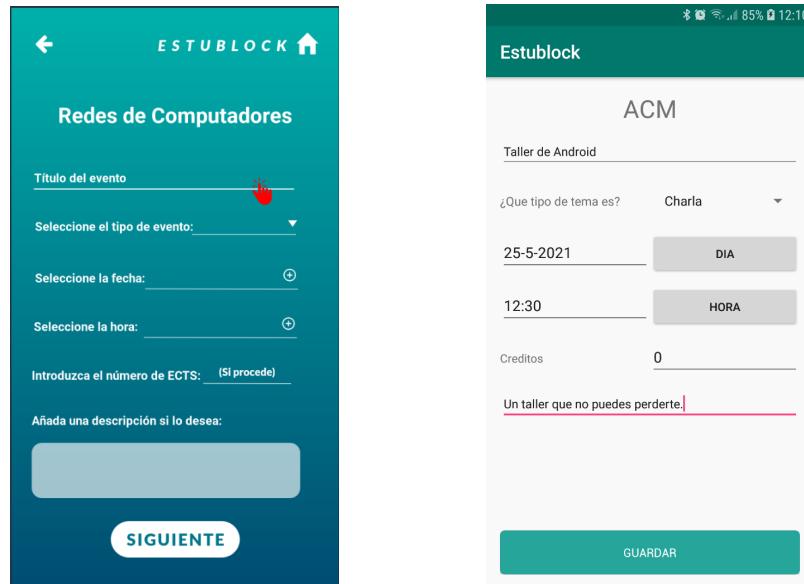
Una vez el usuario selecciona las asignaturas puede proceder a guardarlas. Con respecto al diseño original y al resultado final, caben destacar dos aspectos. El primero es que el botón *Guardar* se ha movido de sitio, no hay ninguna razón especial para esta decisión. Por otro lado, lo que falta es un buscador de asignaturas, este no se ha añadido por ahora, pues no era de crucial importancia al no tener una lista grande de asignaturas disponibles. Sin embargo, es importante que esto se añada en el futuro, pues son muchas las asignaturas que pueden acumularse. El resultado puede verse en 3.21



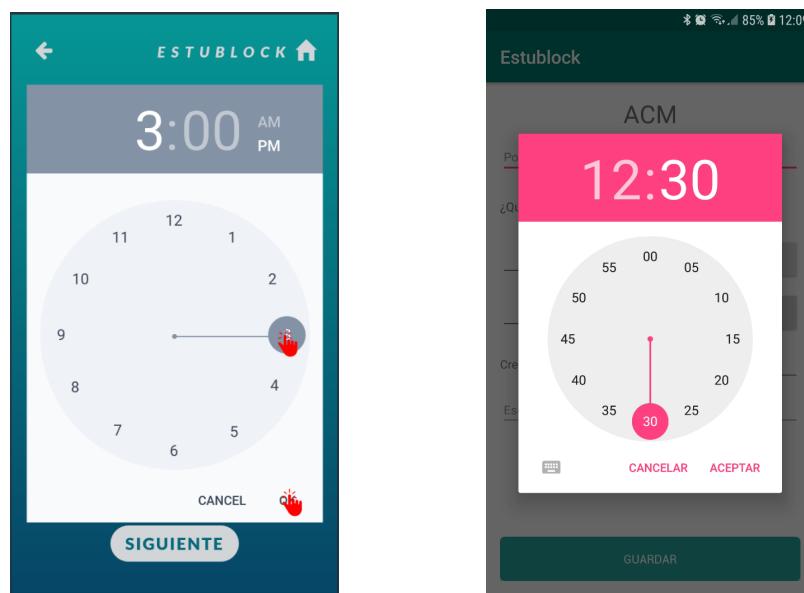
**Figura 3.21:** Pantalla de Suscribirse

### Pantalla de Creación de Eventos

Para llegar a esta pantalla, el usuario tiene que ir desde el menú, a *Crear Evento*, luego se le muestra una pantalla con la lista de asignaturas a las que está suscrito, y una vez seleccionada la asignatura de la que se quiere crear un evento, se le muestra la pantalla de creación de eventos. En esta pantalla el usuario debe introducir datos sobre el evento como el nombre, fecha, hora, descripción... Esta pantalla se ha mantenido muy parecida al diseño realizado con Marvelapp. La única diferencia es que en Marvellapp, el usuario pulsaba *Siguiente* y podía ver un resumen de los datos antes de guardarlos. Se ha decidido eliminar este paso extra, por no aportar nada, pues el usuario puede ver los datos que ha introducido de todos modos en la propia creación del evento, guardando una vez este seguro de los datos elegidos. A continuación se pueden ver unas capturas del resultado 3.22, 3.23



(a) Pantalla de creación de evento de marvelapp (b) Pantalla de creación de evento de Estublock

**Figura 3.22:** Pantalla de Crear Evento

(a) Pantalla de hora de marvelapp

(b) Pantalla de hora de Estublock

**Figura 3.23:** Pantalla de Hora

### Pantalla de generar un QR

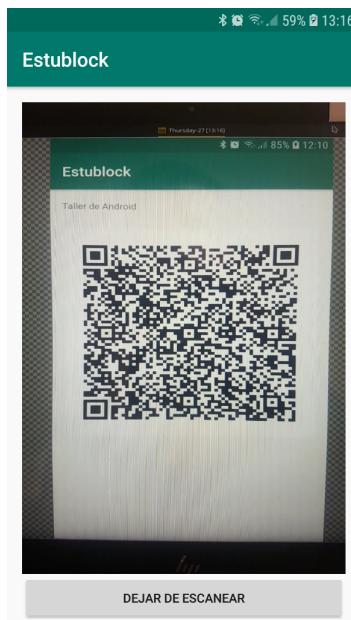
Para llegar a esta pantalla, el usuario ha elegido previamente el botón *Asistencia* así como seleccionar la asignatura y evento al que se quiere registrar. En la pantalla, aparece un gran QR que el profesor debe escanear. Esto es **muy importante**, es uno de los grandes cambios en la forma en la que se registra la asistencia. Al diseñar la aplicación “Estublock” en Marvelapp, se planteó que el profesor enseñaría el QR y los alumnos lo escanearían. Sin embargo ahora se hace al revés. Esta decisión se ha tomado, ya que quien tiene que firmar las transacciones a la red Blockchain es el profesor, para ello, tiene que ser él quien lea los datos de un alumno y firme con su wallet que ese alumno ha asistido al examen. Luego, la pantalla ha quedado más sencilla, pero con mucho más espacio para el QR y así facilitar su escaneo. La pantalla del QR se puede ver a continuación 3.24



**Figura 3.24:** Pantalla de QR

### Pantalla de Escanear un QR

Esta pantalla es especial, pues llama a la actividad de la **cámara de fotos**. Como hemos visto anteriormente en el estado del arte, una aplicación puede llamar sin problemas a otra aplicación (siempre y cuando esta lo permita). En esta ocasión, se requiere de la cámara del usuario para escanear el QR. No todas las cámaras de los móviles escanean QRs de forma automática, pero esto no es un problema ya que la librería que se ha utilizado para programar esta actividad, se encarga de ello. El usuario entonces verá como se abre su cámara y al enfocar un QR, este se escanea automáticamente enviando a la red blockchain la asistencia. Una representación visual puede verse en 3.25



**Figura 3.25:** Pantalla de escaneo de QR Estublock

### 3.3.4. Implementación XML

Como ya hemos mencionado, las interfaces de usuario en Android se programan utilizando código XML. XML, del inglés *Extensible Markup Language* es un metalenguaje que permite definir con un lenguaje de marcas información y datos. Es parecido a HTML, pero las etiquetas no están predefinidas sino que puedes inventarte las que quieras. Permite crear estructuras con parámetros y atributos los cuales pueden utilizarse por ejemplo en internet para enviar datos a una API. En el caso de las aplicaciones móviles, el código XML se utiliza para diseñar las pantallas o *layouts*, ya que XML es un lenguaje ligero y esto hace que las pantallas sean ligeras también. Es con el código XML con el que se definen las jerarquías que se mencionan en el apartado 3.2.1, por ejemplo, parte de la estructura de la pantalla *crear un evento* puede verse en 3.26 se han omitido los atributos para ahorrar espacio.

Programar pantallas para Android sin ningún tipo de feedback, es decir, sin saber como esta quedando la pantalla, es muy difícil. Por eso, Android Studio, trae consigo una herramienta de gran utilidad, la que cual permite visualizar y editar el código sin tener que escribirlo por completo. Se puede por ejemplo crear un botón, y la herramienta de forma automática escribirá el código XML correspondiente, luego se puede cambiar el color del botón, y la herramienta se encargará de escribir el código XML correspondiente. Sin embargo, tras una actualización de Android Studio (actualización a la versión 4.2.0.24-1) el sistema de atributos dejó de funcionar, y hubo que añadir los atributos manualmente, esto supuso un tiempo extra a la hora de realizar las pantallas, y es una de las razones por las que las pantallas (excepto login y registro) han quedado visualmente más básicas. La herramienta de Android Studio puede encontrarse en la figura 3.27.

XML no solo sirve para diseñar las pantallas, hasta ahora todo lo que hemos comentado se refería al esqueleto de una pantalla. Es decir, un botón arriba, un texto pegado abajo a la derecha... Pero con XML también se puede modificar la apariencia, color, forma, fuente de todos los elementos. Dentro de las carpetas de un proyecto Android, tenemos algunas carpetas en las que se guarda código XML que se utiliza posteriormente para darle un mejor diseño a un elemento. Por ejemplo, se puede crear un botón y asignarle un fondo con el atributo *android:background* a este atributo se le añade como parámetro la ruta al archivo que contiene la información en XML sobre como queremos que se vea este botón. Por otro lado, hemos mencionado también las fuentes, los archivos que modifican las fuentes son de tipo *TrueType Font* y definen la fuente de las letras. Por ejemplo, si creamos un



Figura 3.26: Fragmento de la jerarquía XML de la pantalla crear evento

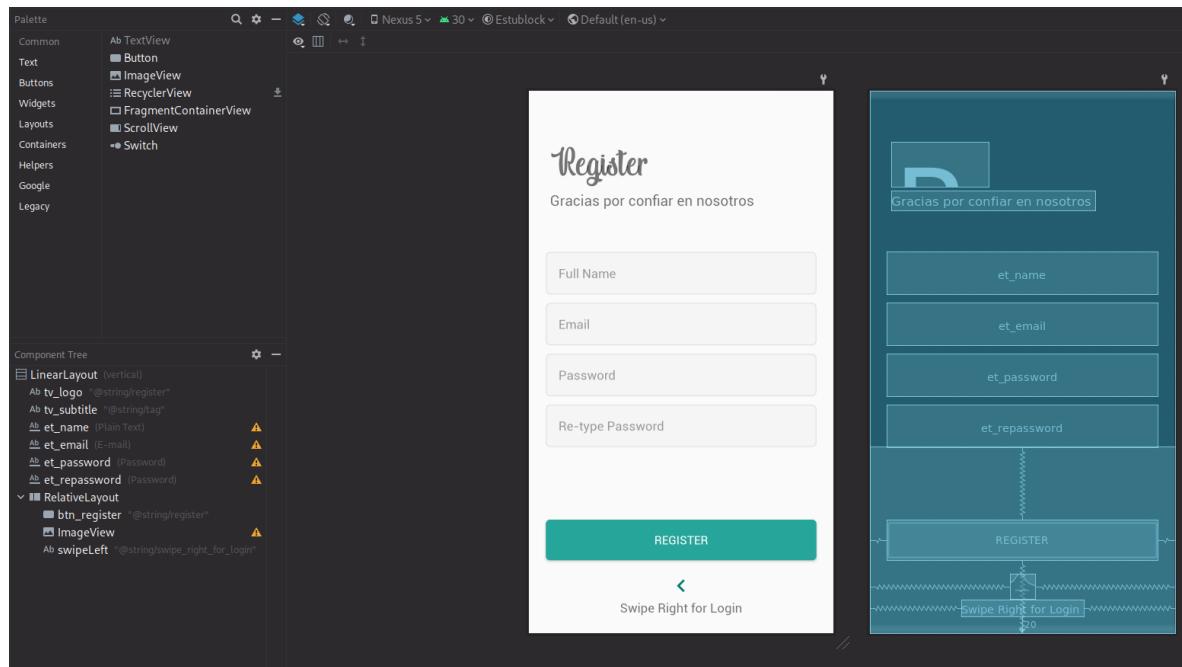


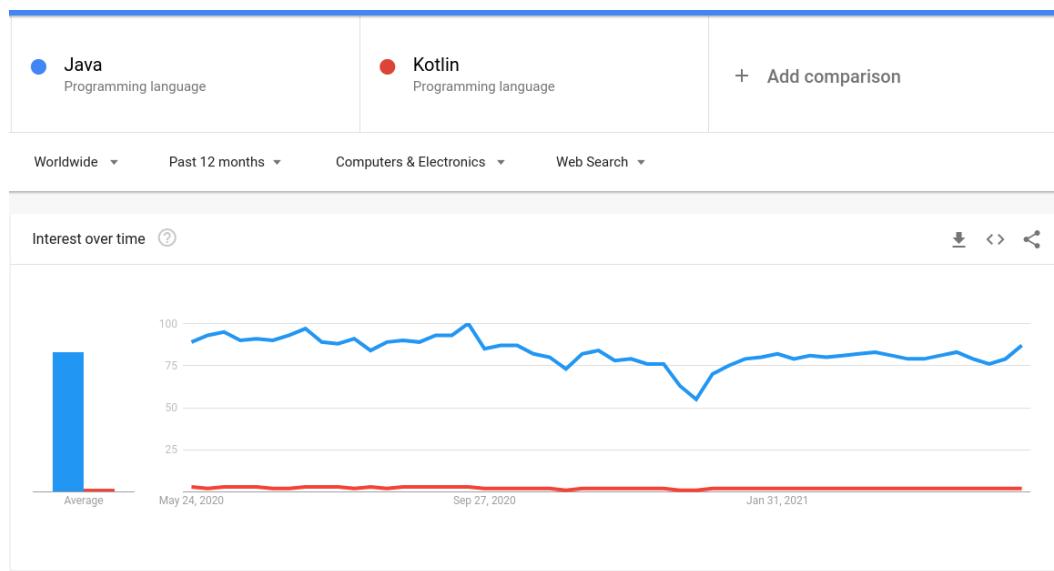
Figura 3.27: Herramienta de Android Studio para la edición de layouts (pantallas)

campo de texto *TextView* podemos ponerle una fuente específica con el atributo *android:fontFamily*. Básicamente, somos libres de modificar con total control cada aspecto que se quiera de las pantallas.

Por último, haremos mención de una carpeta muy importante llamada **values** en la que se guardan parámetros para definir los colores, las dimensiones de la pantalla, los distintos temas disponibles y un archivo llamado **strings** el cual guarda todo el texto estático que se quiere añadir a las pantallas. Este archivo existe, para facilitar el migrado de la aplicación a otros idiomas. Por ejemplo, en vez de escribir en un *TextView* una palabra en español, se le asignará un identificador dentro del archivo *strings*, de modo que si alguna vez tenemos varios idiomas disponibles, se puede tener con el mismo identificador la palabra en varios idiomas, y será tan sencillo como decirle a Android que utilice un archivo *strings* u otro, de modo que las pantallas no hay que modificarlas nunca si se quiere cambiar de idioma. Lo mismo aplica a los colores, temas (si queremos un tema oscuro, no hay que cambiar las pantallas sino que se cambian los colores del archivo de temas y el archivo de colores)...Básicamente, aportan flexibilidad y escalabilidad en la aplicación para poder modificarla con más seguridad y rapidez.

### 3.4. IMPLEMENTACIÓN

Las aplicaciones Android pueden ser programadas principalmente en dos lenguajes de programación, **Java** y **Kotlin**. En el presente, la inmensa mayoría de aplicaciones han sido desarrolladas con Java, sin embargo Kotlin es promete ser el futuro. Actualmente sigue siendo un lenguaje muy secundario (aunque se puede hacer todo lo que se puede hacer con Java y esta muy bien documentado). Según google trends<sup>3.28</sup>, Kotlin esta lejos de quitarle el puesto a Java aunque los nuevos desarrolladores de aplicaciones Android muestran más interés por Kotlin por su comodidad, falta de verbosidad y “limpieza” (es decir, con menos líneas de código haces lo mismo que Java).



**Figura 3.28:** Comparación de Google Trends entre Java y kotlin

Por lo tanto, aunque Kotlin promete mucho, la aplicación Estublock se ha desarrollado en el lenguaje Java para tener una mayor cantidad de documentación y de comunidad disponible. Con comunidad, nos referimos a las personas en el planeta que saben de programación Android con Java y que a lo largo de los años han respondido y contribuido en Internet, dando soluciones a problemas, documentación y contando sus trucos profesionales. De todos modos, se puede migrar con mucha facilidad a Kotlin puesto que AndroidStudio permite migrar automáticamente de Java a Kotlin. Lo único que se necesita para facilitar este proceso es añadir “anotaciones de java” que especifiquen las características de algunas variables o funciones para facilitar a AndroidStudio el reconocimiento de las variables. Un ejemplo puede encontrarse a continuación [3.29](#) donde se puede ver que se especifica que la variable no puede ser nula con `@NotNull`.



Figura 3.29: Ejemplo de anotación de java de variables para facilitar el salto a Kotlin

### 3.4.1. Implementación Java

Todo el código de la aplicación esta disponible en mi repositorio de github[25]. Vamos a tratar de resumir el trabajo realizado, pues gran parte de este TFG está reflejado en la aplicación móvil que se ha desarrollado.

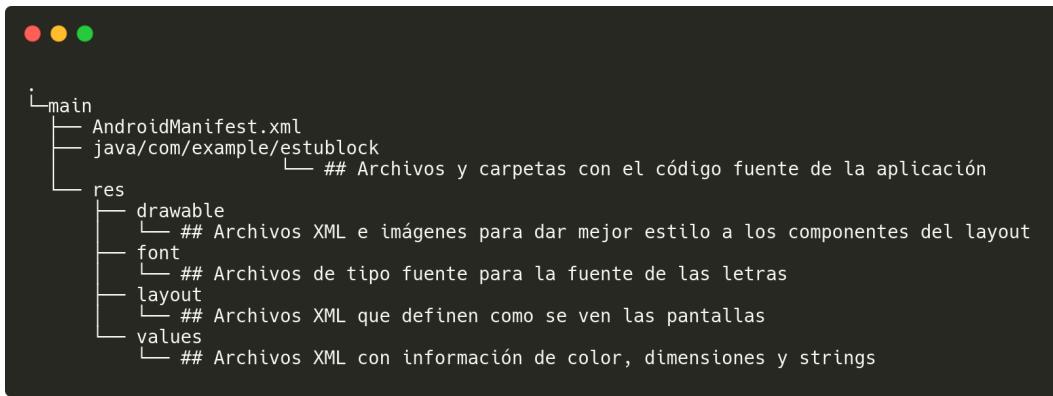
#### Archivos del Proyecto

Un proyecto en Android Studio dispone de un gran conjunto de carpetas y archivos que hacen funcionar a la aplicación entera. En el caso de “Estublock” vamos a mencionar los que son de gran importancia. Empezando por **AndroidManifest.xml**, todos los proyectos Android deben tener un archivo con este nombre en la raíz del proyecto. Este archivo, describe información esencial de la aplicación para las herramientas de creación de Android y para el sistema operativo. Entre otras cosas, el archivo declara el nombre del paquete que contiene la aplicación, los componentes de la aplicación (actividades, servicios...), los permisos de los que requiere la aplicación como permiso a la cámara o a internet como en el caso de “Estublock”. Si se usa Android Studio, este archivo se creará automáticamente pero para añadir permisos ha de hacerse manualmente. En el proyecto “Estublock” se han añadido a este archivo los permisos de acceso a **Internet**, escritura de **datos en memoria** y a la **cámara** del dispositivo.

Por otro lado, toda una carpeta de gran importancia en el proyecto es la carpeta de **Gradle**. Gradle es un paquete de herramientas de compilación avanzadas, que permite automatizar y administrar el proceso de compilación. Al ser una herramienta de compilación, trae consigo un conjunto de carpetas y archivos que utiliza para gestionar el proyecto, en concreto esta el archivo **build.gradle** en el que se escriben las dependencias del proyecto para poder utilizar librerías no incluidas por defecto en java o Android. A la hora de desarrollar la aplicación, se han añadido varias librería que se verán en mejor detalle en **Librerías** como **okhttp**, **bcrypt**, **web3j**...

La estructura de carpetas del SDK, es muy parecida a la de “Estublock” con la principal diferencia de que no tiene ninguna carpeta de interfaz (no tiene carpetas de colores, temas, strings, layouts...). Sin embargo tiene su propio archivo **build.gradle** y su propio archivo **AndroidManifest.xml**. Puesto que no tienen grandes diferencias, mostramos solo la estructura de la carpeta que contiene el proyecto “Estublock” pues tiene más contenido que el SDK [3.30](#)

Por último, al ejecutar la aplicación en el dispositivo móvil por primera vez y registrar a un usuario, se crea en el directorio de carpetas del usuario dos archivos nuevos. Uno de ellos es el **wallet** del cual hablaremos más adelante. Y el otro es una carpeta con un archivo en el que guardar datos en forma de un par {claves –> valor}. Este carpeta se llama **shared preferences**. En el caso de la aplicación, se utiliza para guardar el directorio en el que se ha guardado el wallet.



**Figura 3.30:** Jerarquía de carpetas del proyecto Estublock

### Librerías utilizadas

Se han utilizado 8 librerías extras (a parte de las que incluye por defecto android al iniciar el proyecto). *web3j* se mencionará en el apartado del [SDK](#) y *play-services-vision* es necesaria para *ZXing* pero no se ha usado directamente. Queda entonces:

1. **Volley:** Volley es una biblioteca HTTP que facilita y agiliza el uso de redes en apps para Android. Permite programación automática de solicitudes de res, varias conexiones de red simultáneas, almacenamiento de respuestas en cache... Se ha utilizado esta librería para todas las llamadas excepto la llamadas *DELETE* ya que daba problemas. Algo muy interesante de Volley es que las llamadas son asíncronas, es decir, se ejecutan separadas del hilo principal no bloqueándolo y una vez se recibe la respuesta de la llamada se puede recuperar la información en un callback. Un ejemplo de llamada POST puede verse en [3.31](#)

```
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,
        (URL), parametrosJSON,
        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                // Hacer algo con respuesta correcta.
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // Hacer algo con respuesta error.
            }
});
```

**Figura 3.31:** Ejemplo de llamada POST con Volley

Para solventar el problema con la llamada *DELETE* se ha usado la librería Okhttp.

2. **Okhttp:** Esta librería cumple la misma función que Volley, permitiendo conexiones HTTP... Sin embargo, Volley es asíncrono. Okhttp no lo es, “obligando” al programador a hacer las llamadas dentro de un hilo de ejecución nuevo. Un ejemplo se provee en la figura [3.32](#)
3. **Bcrypt:** Es una implementación del algoritmo de hash de contraseñas Blowfish. Básicamente permite cifrar contraseñas para poder guardarlas de forma segura en una base de datos. Con esta librería se cifra la contraseña del usuario que luego se manda a la API para que la guarde en la base de datos como se puede ver en [3.33](#)
4. **QRGenerator:** Librería para generar códigos QR tanto 1D (códigos de barra) como 2D (QR tradicional) con diferentes formatos. Vease [3.34](#)

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try{
            // Hacer cosas dentro del Thread

            okhttp3.RequestBody body = okhttp3.RequestBody.create(
                paramsJSON.toString(),
                MediaType.parse("application/json; charset=utf-8")
            );

            okhttp3.Request request = new okhttp3.Request.Builder()
                .url(URL)
                .delete(body)
                .build();
            okhttp3.Response response = client.newCall(request).execute();

        } catch(Exception e){
            // Hacer cosas en caso de error.
        }
    }
}).start();
```

Figura 3.32: Ejemplo de llamada DELETE con OkHttp

```
protected String hashPassword(String password){
    return BCrypt.withDefaults().hashToString(10, password.toCharArray());
}
```

Figura 3.33: Ejemplo de cifrado de la contraseña del usuario

```
WindowManager manager = (WindowManager)
getSystemService(WINDOW_SERVICE);
Display display = manager.getDefaultDisplay();
Point point = new Point();
display.getSize(point);

qrgEncoder = new QRGEcoder(evento.toString(), null,
QRGContents.Type.TEXT, Math.min(point.x, point.y));
bitmap = qrgEncoder.getBitmap();
// qr es el identificador de la interfaz de usuario para poner el QR
qr.setImageBitmap(bitmap);
```

Figura 3.34: Ejemplo básico de código de generar QR

5. **ZXing:** Esta librería permite generar códigos QR, pero más importante y la razón por la que se ha usado, permite escanear códigos QR. Para generar QR se utilizó la librería anterior puesto que su uso es más fácil de utilizar que ZXing para Android. El código en este caso es bastante más largo por lo que se ha incluido en el [Anexo](#) en concreto en [7.2](#).

### Archivo GlobalState

Todos los archivos de código fuente son muy importantes, pues engloban toda la funcionalidad de la aplicación. A la hora de desarrollar aplicaciones, en ocasiones hay que tener en cuenta la información que se quiere guardar sobre el usuario que ha hecho login temporalmente (en ejecución). Una forma de pasar información de una pantalla a otra para mantener por ejemplo el correo de un usuario y poder usarlo en otras pantallas sin tener que volver a pedírselo es utilizando los *Intents*. Un Intent es una descripción abstracta de una operación a realizar, puede utilizarse para lanzar otras actividades y junto al lanzamiento pasar como parámetros valores como se puede ver en [3.35](#)



```

public void startActivityForResult(View view) {
    Intent intent = new Intent(this, OneActivity.class);
    EditText editText = (EditText) findViewById(R.id.editText);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivityForResult(intent);
}

```

**Figura 3.35:** Ejemplo de Intent en Android

Sin embargo esto trae una desventaja, siempre que se pase de una a otra pantalla hay que añadir los datos que se quieren guardar para no perderlos, esto puede hacer que si quieras un dato en muchas pantallas, acabes acumulando datos, y hace que el código sea menos legible. Por lo tanto, se tomó la decisión de crear un archivo especial llamado **GlobalState.java**. Este archivo no tiene una actividad (pantalla) correspondiente, pues su único propósito es el de guardar el estado de algunos datos para evitar pedírselos nuevamente al usuario. Datos como el correo, nombre, directorio donde esta el wallet... Pero también se ha aprovechado para guardar las URL de las APIs, nodo de la blockchain... El código de este archivo esta disponible en el [Anexo](#) reffig:gs

## 3.5. SDK

Un *Kit de Desarrollo Software* o SDK (Software Development Kit) es generalmente un conjunto de herramientas de desarrollo de software que permiten a otros desarrolladores crear aplicaciones de forma más cómoda. Un buen ejemplo de SDK que se ha utilizado en este proyecto es la familia de SDKs de Android. Junto con AndroidStudio, se instalan múltiples SDKs que permiten al programador escribir código, recuperar componentes de las pantallas, hacer llamadas a paquetes, importar paquetes, detectar errores... Todo de forma automática haciendo que la programación sea mucho más fácil.

El SDK es entonces un conjunto de métodos los cuales pueden hacer llamadas a una API, llamadas al propio sistema operativo, llamadas a otras librería o SDKs... En el caso del SDK desarrollado para este TFG, su principal función es la de hacer llamadas a una red blockchain y la de hacer llamadas al sistema de archivos del dispositivo para guardar el wallet del usuario.

### 3.5.1. Diseño del SDK

El SDK dispone de 3 archivos separados del proyecto Estublock. Se han creado como librería desde AndroidStudio para poder compartir la después públicamente y que otros programadores puedan

utilizarla. Se ha dividido en 3 archivos para tratar de mantener una coherencia entre las funcionalidades que ofrece el SDK. Uno de los archivos se encarga únicamente de gestionar las transacciones, otro archivo se encarga únicamente de gestionar el wallet del usuario, y por último un archivo que se encarga de gestionar los callbacks de las llamadas a la red blockchain, pues estas se ejecutan de manera asíncrona.

Por otro lado, con respecto al diseño de los métodos, se ha optado por tratar de programarlos de la forma más genérica posible para que puedan aplicarse en una amplia variedad de situaciones, además se han sobrecargado los métodos (más adelante veremos que significa sobrecargar métodos) para que se puedan utilizar distintos tipos de variables en las funciones sin problema.

### 3.5.2. Comunicación con la Red Blockchain

En el archivo *TransactionsHelper.java* se encuentra importada la librería de *web3j* y se encuentra también el código que firma y envía las transacciones. Web3j es una biblioteca de Java y Android que permite trabajar con smart contracts e integrarse con clientes o nodos en la red de *Ethereum*. Esto permite trabajar con la red de Ethereum sin necesidad de implementar el código de integración para la plataforma. Implementa la API de cliente JSON-RPC de Ethereum sobre HTTP, soporta wallets, firmado de transacciones... JSON-RPC es un protocolo de llamada a procedimientos remotos que utiliza JSON para codificar los mensajes, es similar a XML-RPC.

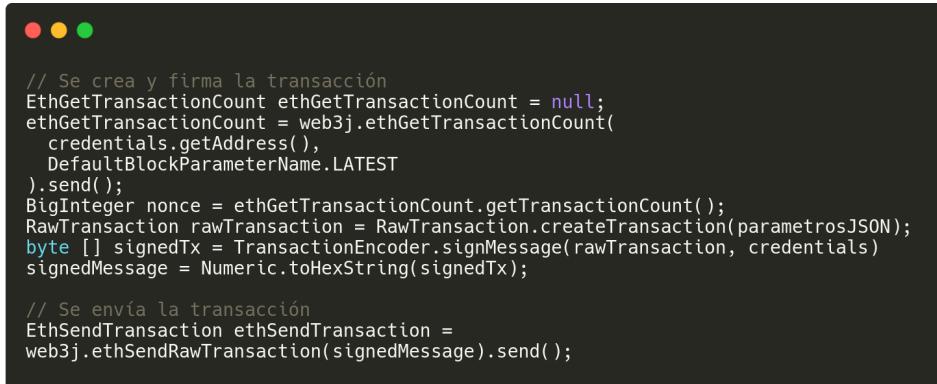
El uso de *TransactionsHelper* es sencillo, primero se crea el objeto pasándole como parámetro la URL del nodo de la red blockchain al que vamos a conectarnos [3.36](#)



```
// Constructor
public TransactionsHelper(@NotNull String blockchainURL){
    web3j = Web3j.build(new HttpService(blockchainURL));
}
// Llamada
TransactionsHelper txHelp = new TransactionsHelper(URL);
```

**Figura 3.36:** Constructor de TransactionsHelper

Una vez creado el objeto se puede llamar al resto de funciones, entre las que están *signTransaction(...)* que recibe seis argumentos entre otros las credenciales del wallet, el destino de la transacción (el smart contract) y un *listener* para recuperar la respuesta de la transacción. Recordemos, no se pueden hacer operaciones pesadas en el hilo principal de Android por lo que hay que hacerlo en un hilo a parte (es decir en un *Thread*). Otra función importante es *sendSignedTransaction(...)* la cual recibe dos parámetros, la transacción firmada y un *listener*. Ambas funciones están “sobrecargadas” (sobrecarga es la capacidad de un lenguaje de programación, que permite nombrar con el mismo identificador diferentes variables u operaciones), la razón de esta sobrecarga es permitir pasar los parámetro de múltiples maneras. Por ejemplo, los datos de la transacción se pueden pasar como JSON, como String, como HashMap... y gracias a la sobrecarga podemos ponerle el mismo nombre a la función. A grandes rasgos un pequeño ejemplo de código puede verse en [3.37](#) y un ejemplo completo de función puede verse en el Anexo [7.3](#)



```

// Se crea y firma la transacción
EthGetTransactionCount ethGetTransactionCount = null;
ethGetTransactionCount = web3j.ethGetTransactionCount(
    credentials.getAddress(),
    DefaultBlockParameterName.LATEST
).send();
BigInteger nonce = ethGetTransactionCount.getTransactionCount();
RawTransaction rawTransaction = RawTransaction.createTransaction(parametrosJSON);
byte [] signedTx = TransactionEncoder.signMessage(rawTransaction, credentials);
signedMessage = Numeric.toHexString(signedTx);

// Se envía la transacción
EthSendTransaction ethSendTransaction =
web3j.ethSendRawTransaction(signedMessage).send();

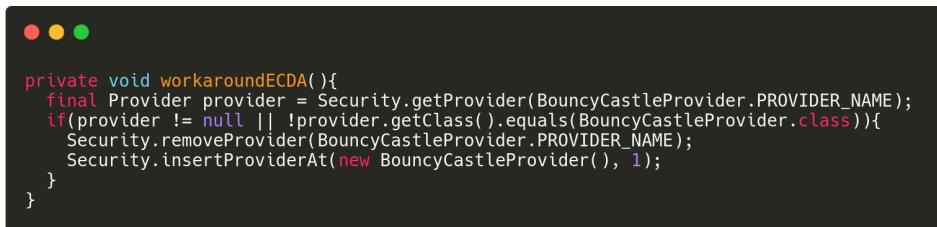
```

**Figura 3.37:** Firmado y enviado de una transacción

### 3.5.3. Comunicación con el Dispositivo Móvil

Para facilitar la generación de credenciales, wallets, guardar en la carpeta “Shared Preferences” de Android todos los datos necesarios para la comunicación con la red blockchain... Tenemos el archivo *WalletHelper.java*. En él además de la librería web3j, tenemos algunas funcionalidades de Seguridad de Java para modificar el proveedor de seguridad, y permitir que salten errores por algoritmos o parámetros erróneos.

En este archivo tenemos entonces un constructor, el cual modifica el proveedor de seguridad a causa de un error con el algoritmo “Elliptic Curve Digital Signature Algorithm”(ECDSA)[86] para el proveedor BC[12]. Más información sobre el error se puede encontrar en el repositorio de web3j en la [issue\\_915](#). Básicamente lo que hace el código es modificar el proveedor [3.38](#)



```

private void workaroundECDA(){
    final Provider provider = Security.getProvider(BouncyCastleProvider.PROVIDER_NAME);
    if(provider != null || !provider.getClass().equals(BouncyCastleProvider.class)){
        Security.removeProvider(BouncyCastleProvider.PROVIDER_NAME);
        Security.insertProviderAt(new BouncyCastleProvider(), 1);
    }
}

```

**Figura 3.38:** Modificación de proveedor de seguridad

Luego, el programador dispone de varias funciones para crear un nuevo wallet con *createNewWallet(...)*[3.39](#) que acepta dos parámetros, siendo estos una contraseña y luego el lugar en el que se quiere guardar el wallet creado. Al igual que antes, esta función esta sobrecargada permitiendo pasar la localización como objeto String o como objeto File. Y permite también recuperar el *address* del usuario o todas las credenciales[3.40](#).

```

    /**
     * Método que crea una nueva cartera virtual
     * @param password Contraseña de la cartera virtual
     * @param walletDirectory Dirección de la nueva cartera virtual
     * @return Devuelve el nombre del keystore
     */
    @NotNull
    public String createNewWallet(@NotNull String password, @NotNull File walletDirectory)
        throws CipherException, InvalidAlgorithmParameterException, NoSuchAlgorithmException,
               NoSuchProviderException, IOException {
        return WalletUtils.generateLightNewWalletFile(password, walletDirectory);
    }

```

Figura 3.39: Creación de un nuevo wallet

```

    /**
     * Devuelve las credenciales
     * @param password Contraseña de la cartera virtual
     * @param keyStoreDirectory Dirección de la cartera virtual
     * @return credenciales de la cartera virtual
     */
    @NotNull
    public Credentials getCredentials(@NotNull String password, @NotNull File keyStoreDirectory)
        throws IOException, CipherException {
        return WalletUtils.loadCredentials(password, keyStoreDirectory);
    }

```

Figura 3.40: Recuperación de las credenciales

Como hemos mencionado anteriormente en este archivo se incluye también el guardado de los datos en él dispositivo móvil, aunque la función de web3j `WalletUtils.generateLightNewWalletFile()` guarda los datos en el móvil. Para permitir que un usuario tenga varios wallets, se utiliza el `SharedPreferences` para enlazar al usuario con su wallet. Para ello tenemos la función `saveInPreferences(...)` la cual recibe cuatro parámetros entre ellos el lugar donde guardarlos, y los datos a guardar. Y luego tenemos otro método `getFromPreferences(...)` para recuperar los datos<sup>3.41</sup>.

```

    // Guardamos los datos
    SharedPreferences sharedpreferences = activity.getSharedPreferences(prefsName,
        Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedpreferences.edit();
    editor.putString(id, walletDirectory);
    editor.apply();

    // Devolvemos el dato con identificador id
    return sharedpreferences.getString(id, null);

```

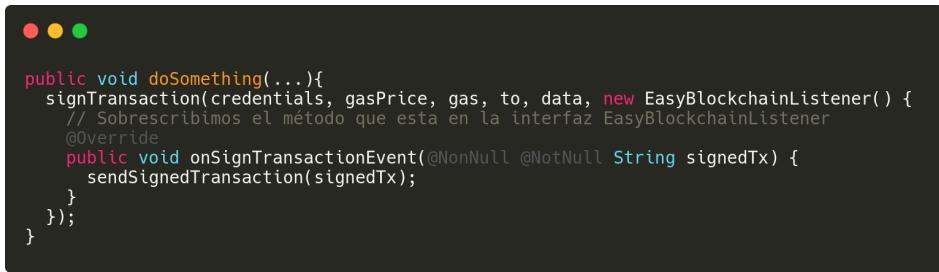
Figura 3.41: Guardado y recuperación de datos

Se ha decidido utilizar `SharedPreferences` para elegir que wallet desbloquear cuando un usuario hace login, ya que se ha visto la posibilidad de que un mismo usuario necesite más de un wallet. El caso de uso es el siguiente, el presidente de una asociación es alumno y a su vez presidente. Dicho de otro modo, por como esta diseñado “Estublock” el usuario tendrá dos wallets, uno para él como estudiante y otro para él como presidente de una asociación. Para ello utilizará por un wallet su correo como alumno y para el otro su correo de asociación, de ahí la necesidad de relacionar el correo con el directorio y nombre del keystore que se genera (más sobre keystore en [wallet](#)).

### 3.5.4. “Callback Listener”

Como las llamadas a la red blockchain son pesadas, han sido implementadas con threads: “`new Threads(new Runnable(){...})`”. Como el código no espera a la respuesta del thread, se han implementado

dos funciones que hacen de “escucha” para cuando termina la ejecución. Estas funciones están especificadas en *EasyBlockchainListener.java*, este archivo es una *interfaz de java* para poder ser sobrescritas por el programador para que las funciones hagan lo que el programador desee. Utilizando la anotación de java *@Override* con la que se sobrescribe un método para que el programador cambie el código y pueda hacer con la respuesta lo que considere necesario. Tenemos dos callbacks, uno para cuando se firma la transacción, y otro para cuando se envía a la red blockchain. Para utilizarlos no hay más que pasar como último parámetro de las funciones anteriormente mencionadas una función que será la que sobrescriba a los métodos como se puede ver en [3.42](#).



```

public void doSomething(...){
    signTransaction(credentials, gasPrice, gas, to, data, new EasyBlockchainListener() {
        // Sobrescribimos el método que está en la interfaz EasyBlockchainListener
        @Override
        public void onSignTransactionEvent(@NotNull @NotNull String signedTx) {
            sendSignedTransaction(signedTx);
        }
    });
}

```

**Figura 3.42:** Función de ejemplo de un callback

### 3.5.5. Cómo Incorporarlo en Otras Aplicaciones

Para que otro programador pueda utilizar el SDK que se ha desarrollado el primer paso es publicar el SDK en algún repositorio como en los repositorios de *Maven Central*[[76](#)]. Maven dispone de una inmensa cantidad de repositorios que usuarios pueden utilizar añadiendo la dependencia a sus proyectos, y gradle (el sistema de automatización de construcción de código que utiliza Android) se encarga de bajar la información automáticamente y así el usuario puede utilizar las funcionalidades que el usuario desea. Esta comodidad y facilidad en la gestión del SDK es la razón por la que se ha decidido utilizar *Maven*. Una alternativa era utilizar *jcenter()* sin embargo, no se ha hecho puesto que la empresa que mantiene el repositorio JCenter (empresa llamada *JFrog*[[19](#)]) lo convirtió en un repositorio de solo lectura el **31 de Marzo de 2021**.

Para publicar la librería se han de seguir a muy grandes rasgos estos pasos:

- Crear un “ticket” con *Sonatype*[[78](#)].
- Crear el proyecto en Sonatype.
- Verificación de propiedad mediante DNS.
- Instalación del plugin para gradle [gradle-maven-publish-plugin](#)
- Generar llaves GPG
- Configurar firmas
- Subir los artifacts a Sonatype
- Publicar la librería

Se pueden encontrar más detalles en el post de “Waseef Akhtar”[[50](#)].

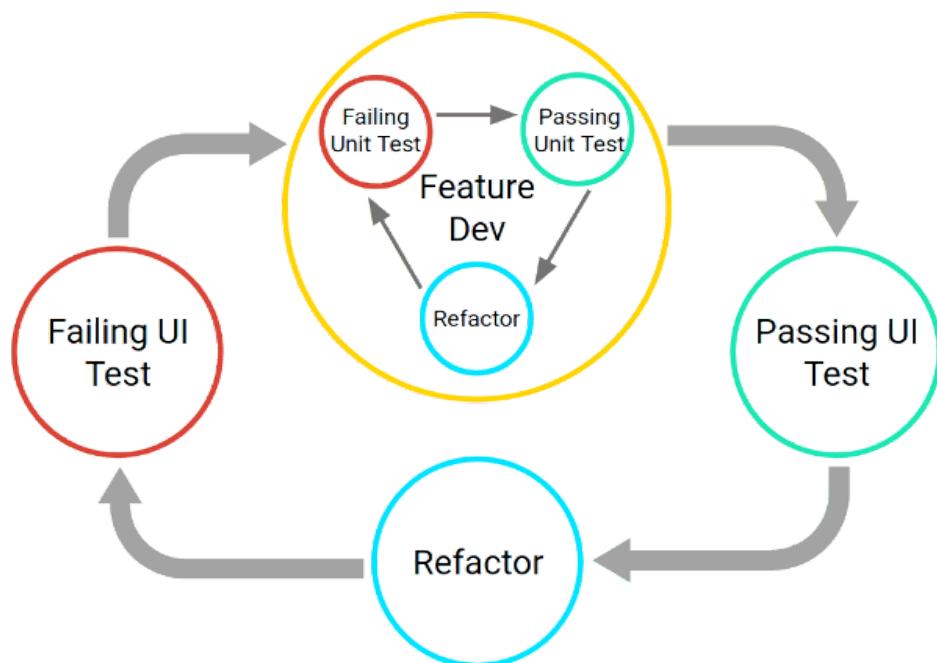
Una vez la librería es pública, cualquier programador puede añadirla a su proyecto añadiendo a sus dependencias la línea *implementation 'com.<entidad>.estublock:EasyBlockchain:1.0.0'*, el nombre de la entidad aún está por definir. Una vez añadida la línea y después de que gradle termine de sincronizar el repositorio, el programador puede utilizar todas las funciones que hayan en la versión 1.0.0 del SDK.

## 3.6. PRUEBAS

Probar la app es una parte importante dentro del proceso de desarrollo. Con pruebas coherentes, se puede verificar la corrección, comportamiento funcional y usabilidad de la aplicación antes de

lanzarla al público. Las pruebas brindan muchas ventajas, entre otras están: Los avisos rápidos de **fallos y detección** de fallos. **Refactorización** de código más segura, y proporcionan **velocidad de desarrollo** más estable al ayudar a minimizar el tiempo que se pierde arreglando errores más tarde en el proyecto. Además, permite programar con más seguridad, pues tras la implementación de un método o funcionalidad nueva esta se puede probar al momento, dando feedback al desarrollador sobre el correcto funcionamiento del código.

En Android, se pueden realizar pruebas unitarias con **JUnit**, o pruebas instrumentadas que se ejecutan en un dispositivo móvil. Además, Android facilita la integración de frameworks de prueba como *Mockito*[52], *Espresso*[26] o *UI Automator*[27]. Android recomienda crear y probar el código de forma iterativa 3.43. Por cada unidad, es decir, por cada método o un pequeño conjunto de métodos que trabajen juntos, se debe implementar una o varias pruebas que agoten todos los casos de interacción con esa unidad. Desde casos raros a triviales, pasando por entradas no validas, casos de acceso a recursos que no existen, interacciones estándar...



**Figura 3.43:** Ciclos asociados con el desarrollo iterativo

Existen pruebas de unidad local, estas se ejecutan en la maquina virtual de java en el ordenador del desarrollador. Son útiles cuando no existe dependencias con el framework de Android. Por otro lado, están las pruebas instrumentadas, que se ejecutan en un dispositivo móvil o emulador. Estas pruebas tienen acceso a las APIs y tienen su propio archivo *AndroidManifest.txt*, aunque Gradle se encarga de generarlo automáticamente durante la compilación.

### 3.6.1. Pruebas en Estublock

#### Tests Manuales

Las herramientas que proporciona Android son muy efectivas, pero requieren de una implementación y estrategia a la que no se le ha podido invertir el tiempo deseado. Por lo tanto, para Estublock, se ha optado por hacer *pruebas manuales*. Las pruebas manuales o tests manuales, son casos de prueba ejecutados manualmente por una persona sin herramientas que permitan automatizar las pruebas. El objetivo de las pruebas manuales es identificar los errores, problemas y defectos de la aplicación, y aunque sea la técnica más primitiva, ayuda a encontrar errores críticos en la aplicación. Existen

varios tipos de pruebas manuales, en concreto hay dos que nos interesan conocidas como **Black Box Testing** (test de caja negra) y **White Box testing** (test de caja blanca). En la primera, la persona que realiza el test desconoce el código que se ejecuta por debajo, solo le interesa ver la entrada de datos y que respuesta recibe. Este test es recomendable con usuarios finales o con personas que no han participado en el proyecto. El *white box testing* sin embargo, tiene en cuenta el código que se ejecuta.

Esta técnica es la que se ha utilizado en “Estublock” y en el SDK. Es una técnica de prueba de software en la que se comprueba la estructura interna, el diseño y la codificación del software para verificar el flujo de entrada-salida y mejorar el diseño, la usabilidad y la seguridad. También es conocida como *open box testing* o *transparent box testing* (test de caja abierta o test de caja transparente).

## Estublock y SDK

Para verificar el correcto funcionamiento de Estublock y del SDK, se han realizado entonces pruebas manuales con una estrategia *White Box*. En concreto:

- Se realiza el registro de un usuario que no existe en el sistema, se verifica que la aplicación entra al menú de inicio. Se verifica que el SDK ha creado un wallet para el usuario, y que este se encuentra registrado correctamente en la base de datos.
- Se hace login con el usuario creado. Se verifica que entra al menú de inicio y que se ha recuperado el directorio con el keystore del usuario.
- Se suscribe al usuario a un grupo de asignaturas. Con esto se comprueba que la base de datos devuelve correctamente la lista de asignaturas, que la aplicación es capaz de listarlas y que es capaz de seleccionar y guardar en la base de datos las asignaturas que el usuario ha seleccionado.
- Se crea un nuevo evento, con lo que se verifica que, se recupera correctamente la lista de asignaturas a las que el usuario está suscrito, y también se verifica que los datos que el usuario introduce para crear un evento nuevo son firmados y registrados con las credenciales del usuario en la red blockchain. Con lo que se comprueba que el SDK es capaz de preparar una transacción, firmarla y enviarla.
- Se procede a registrar la asistencia al evento creado. Se comprueba que el evento anteriormente añadido a la red blockchain puede recuperarse, y se comprueba que la aplicación puede generar un código QR con los datos del evento y del usuario.
- Por último, se comprueba el registro de la asistencia, es decir, el escaneo del código QR. Con ello se verifica que funciona correctamente la cámara junto con la librería que escanea códigos QR. Y, que con la información del código QR el SDK puede preparar la transacción, firmarla y enviarla.

Junto a todo esto, se han comprobado también manualmente, casos en los que un usuario no existe (intentar hacer login con credenciales inválidas) o tratar de añadir a un usuario que ya tiene una cuenta existente en el sistema (mismo correo electrónico). Así como intentar firmar una transacción con la contraseña que desbloquea el wallet incorrecta y tratar de suscribirse a un tema al que ya está suscrito el usuario.

Por otro lado, se han hecho pruebas específicas del SDK con ayuda de **ganache**<sup>[69]</sup>. Ganache permite levantar una red blockchain de pruebas creando además 10 cuentas aleatorias para poder “experimentar” con ellas. La prueba ha consistido en desplegar un smart contract en la red, y que el SDK firmase transacciones y las enviase a la red blockchain de ganache. La ventaja de utilizar ganache es poder utilizar el protocolo **Geth** desde la consola de comandos de linux para ver información sobre lo que está sucediendo en la red blockchain de forma local sin riesgos de comprometer la red blockchain original que se utiliza en “Estublock”. **Geth**<sup>[16]</sup> es una implementación en el lenguaje *Golang* del protocolo de Ethereum. Lógicamente, el objetivo de estas pruebas al igual que las anteriores es buscar errores, fallos y bugs en el SDK que no se hayan podido probar con las pruebas anteriores.

### 3.7. WALLET

El wallet es una aplicación que permite interactuar con la cuenta de Ethereum. Puede verse como una aplicación bancaria, que te permite ver tu saldo, enviar transacciones y conectar con aplicaciones. En el caso de “Estublock” no existe el concepto de saldo, pero sí se hacen transacciones y llamadas a una aplicación (el smart contract que tenemos ejecutando en la red blockchain a la que hacemos llamadas). El wallet guarda las credenciales del usuario, también conocidas como **keystore**.

Cuando en la aplicación se registra un nuevo usuario, se le crea un wallet y se guarda este wallet con las credenciales en el dispositivo móvil. En el keystore se guarda la clave privada del usuario. Una clave privada en Ethereum no es más que 64 caracteres hexadecimales elegidos al azar. Aunque cualquiera pudiera elegir la clave privada manualmente, no es recomendable por seguridad. La clave pública se deriva de la clave privada utilizando el algoritmo de curva elíptica ECDSA *Elliptic Curve Digital Signature Algorithm*. Una vez se tiene la clave pública y privada, se recupera el address del usuario, con este address se identifica al usuario en la red blockchain. En Ethereum se coge la clave pública y se ejecuta la función hash “SHA 3” sobre ella, y como resultado se obtienen 64 caracteres de los cuales los últimos 40 son el address del usuario. Lógicamente todo este proceso lo hace la aplicación “Estublock” con ayuda del SDK y de la librería *web3j*.

Para añadir una capa de seguridad al keystore, y la clave privada quede cifrada, se añade una contraseña de cifrado que el usuario debe introducir para desbloquear el wallet. En la aplicación se ha decidido que la contraseña que se va a utilizar para desbloquear el wallet sea la misma que la que se utiliza para hacer login. Por lo tanto, una vez hecho el login, el usuario no debe introducir la contraseña nuevamente para hacer las transacciones. La clave **privada** se utiliza para firmar las transacciones que se envían a la red blockchain, y la clave **pública** se utiliza para que cualquier persona pueda verificar la firma.

#### 3.7.1. Gestión del Wallet

Existen distintos tipos de wallet<sup>3.44</sup>. Cada uno con sus ventajas y desventajas. Los “web wallet”, son los más fáciles de utilizar, pero traen consigo un inconveniente. El usuario no tiene el control de sus credenciales. Las credenciales del usuario las mantiene una empresa externa. Por eso no se ha pensado en este tipo de wallets para la aplicación. Por otro lado tenemos las “Cold Wallets”, las cuales son muy seguras, y el usuario tiene total acceso y control de las credenciales almacenadas en un dispositivo externo sin conexión a internet. La falta de conexión a internet y la incomodidad al usar una cold wallets, fueron la razón por las que se descartó este tipo de wallets para el proyecto. Luego están las dos mejores, **Hardware y Software Wallets**. Ambas aportan gran seguridad y control de las credenciales, por su usabilidad, las hardware wallets son más incómodas pues se suelen utilizar dispositivos externos para guardar las credenciales, como un disco duro cifrado. Por lo que, la mejor decisión era utilizar las **Software Wallets** para el proyecto.

Las software wallets son fáciles de crear, el usuario tiene buen control de las credenciales pues están guardadas en su dispositivo móvil y tiene buena seguridad. Sin embargo hay que destacar las desventajas. Aunque no ocurre de forma habitual, el usuario puede perder el móvil, se lo pueden robar, romper o puede cambiarlo por uno mejor. Todas estas situaciones llevan al mismo lugar, el usuario no tiene acceso al wallet que tenía en el móvil original, y necesita acceso nuevamente al wallet. Otra situación en la que llegamos al mismo problema es **olvidar** la contraseña que utilizamos para desbloquear el Wallet.

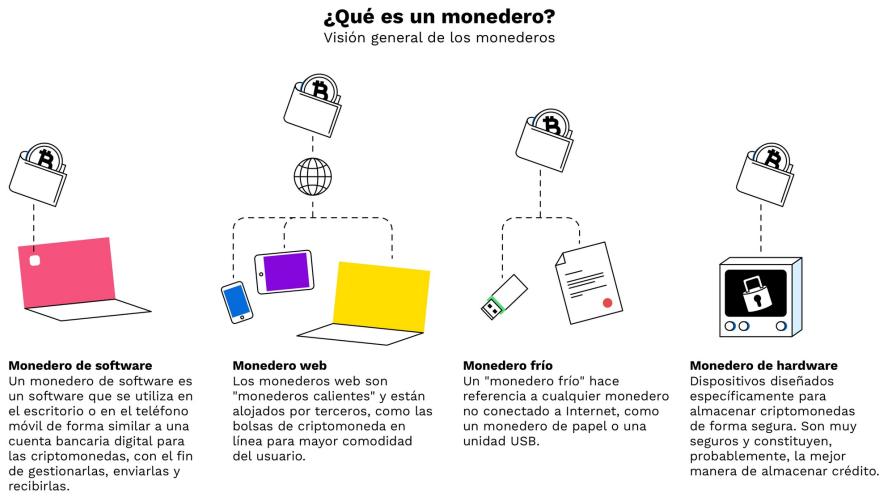


Figura 3.44: Visión general de los wallets (Bitpanda)

### 3.7.2. Perdida y Recuperación del Wallet

Un usuario puede perder acceso a su wallet a causa de múltiples razones como hemos mencionado antes. Aunque es muy difícil recuperar el acceso al wallet de un usuario, es posible. Existen compañías como el equipo de *walletrecovery*[73] los cuales se encargan de intentar recuperar en la medida de lo posible el acceso a tu wallet. Ya sea por causas hardware o que se ha olvidado la contraseña, probarán métodos de fuerza bruta, programas de recuperación de archivos dañados.... No es la única empresa que se encarga de esto, en *WRS*[74] también se encargan de recuperar wallets. Por último también existen programas de código abierto los cuales el usuario puede utilizar para recuperar las credenciales en la medida de lo posible, programas como **btcrecover**[22]. La herramienta permite generar un diccionario de fuerza bruta a partir de información dada por el usuario, así como probar varias combinaciones de palabras dada una "seed phrase".

En el caso de "Estublock" no hay implementado en el presente un sistema de recuperación del wallet. Sin embargo, se pueden implementar tres métodos para evitar que el usuario pierda acceso a dicho wallet. Por un lado tenemos las **seed phrases**, estas son un conjunto de palabras aleatorias que el usuario debe guardar a buen recaudo y "bajo llave" pues pueden servirle en caso de emergencia para recuperar el acceso al wallet. Si el usuario olvida la contraseña, las *seed phrases* servirán para recuperar dicho acceso. Por otro lado, existe la opción de compartir parte de la clave privada con varios usuarios de mucha confianza. En el caso de olvidar la contraseña de su wallet, el usuario tiene acceso a su clave privada reuniendo dicha clave con ayuda de esas personas de mucha confianza. Una vez reconstruida la clave privada, puede utilizar una nueva contraseña para cifrarla y seguir utilizando sin problemas el sistema. Por último, es ideal que el usuario haga una copia de seguridad del wallet para evitar que perder el móvil, o fallos en el móvil (archivos corruptos, móvil roto...) o un robo puedan hacerle perder el acceso al wallet.

## 3.8. SEGURIDAD Y KEYSTORE

En la sección anterior sobre el wallet hemos mencionado brevemente la palabra **keystore**. Aquí vamos a profundizar mucho más sobre su uso, funcionamiento, que seguridad se ha implementado en "Estublock" para tratar de mantener el archivo seguro y como evitamos que, aunque alguien tenga acceso al móvil, minimizar el riesgo de perder que el *keystore* se vea comprometido.

### 3.8.1. Keystore

Un archivo **keystore**[\[34\]](#) (conocido también como Ethereum UTC file) es una versión cifrada de la *clave privada de Ethereum*[\[34\]](#) del usuario. Esta clave privada, se utiliza para firmar las transacciones. Si se pierde el archivo, se pierde acceso a la clave privada y por tanto no se puede firmar ninguna transacción. Esto provoca que tu cuenta en Ethereum quede congelada para siempre sin que puedas utilizarla, ni recuperar el saldo. La clave privada se podría guardar en un archivo no cifrado sin problema, pero hay que atenerse a las consecuencias, pues esta queda completamente vulnerable a que cualquier persona la lea y se envíe a si mismo todo el dinero de la cuenta.

Es por esto mismo que existe el **keystore**, este permite guardar la clave privada cifrada con una contraseña. Si alguien quisiese robar la cuenta de otra persona, necesitaría la contraseña de la otra persona y el *keystore*, es obligatorio tener ambas cosas, aunque parezca un detalle pequeño, supone ya una barrera más de seguridad. Para entrar por ejemplo en la cuenta bancaria de una persona, solo se necesita tener la contraseña de esta persona. Aquí, hace falta acceso al *keystore* y a la contraseña. Cuando se utiliza un cliente de Ethereum para realizar transacciones, por ejemplo, la librería **web3j** que se ha utilizado en “Estublock”, el cliente pide la contraseña para poder descifrar el *keystore* y llevar a cabo la transacción.

¿Pero, qué información hay en el *keystore*? en la siguiente figura 3.45 se presenta un *keystore* creado por la aplicación “Estublock”. Este *keystore* esta guardado en el dispositivo móvil.



```
{
  "address": "585137d69931725c0a0e057b804c2be749ba4938",
  "id": "84da2f19-68b9-46c0-b6e1-99e505752865",
  "version": 3,
  "crypto": {
    "cipher": "aes-128-ctr",
    "cipherparams": {
      "iv": "3f5f3f82b06884cf84fbea18ebff945d",
      "ciphertext": "0a2fa6fe3d3669527a771cedbbddc57f8a1c261266cc0a2b5d5a616b24e6802",
      "kdf": "scrypt",
      "kdfparams": {
        "dklen": 32,
        "n": 4096,
        "p": 6,
        "r": 8,
        "salt": "8c5be4377b7edba163383517791b96d7bc494fefc03ba8fd0e34dc5d3de6e5c5"
      },
      "mac": "1572721da9f5307a38d7cc58a2d504f3235b9fb4246783db96873387f3aab822"
    }
  }
}
```

Figura 3.45: Contenido de un Keystore

Como se puede apreciar, el archivo es de tipo *json*, y la inmensa mayoría de la información se encuentra dentro de *crypto*. Pasemos a profundizar sobre los componentes:

- **address:** Address de la cuenta asociada a este keystore.
- **cipher:** Nombre del cifrado simétrico utilizado, en este caso **AES-128**[\[1\]](#). El cifrado AES, en inglés Advanced Encryption Standard, es un estándar utilizado en el cifrado de información para que se mantenga privada. Es muy popular y seguro (el gobierno de los estados unidos lo utiliza para mantener datos clasificados a salvo de hackers). AES cifra bloques de texto con la ayuda de una contraseña de X bits de longitud (en este caso 128 bits, de ahí que se llame AES-128).
- **cipherparams:** Parámetros requeridos para el algoritmo de cifrado.
- **ciphertext:** La clave privada de Ethereum del usuario, cifrada utilizando el algoritmo AES.
- **kdf:** Función de derivación de clave o *Key Derivation Function*(KDF)[\[13\]](#). Deriva una o más claves secretas a partir de una contraseña usando una función pseudoaleatoria. Los KDF se pueden utilizar para extender claves en otras más largas, aumentando la seguridad. Básicamente,

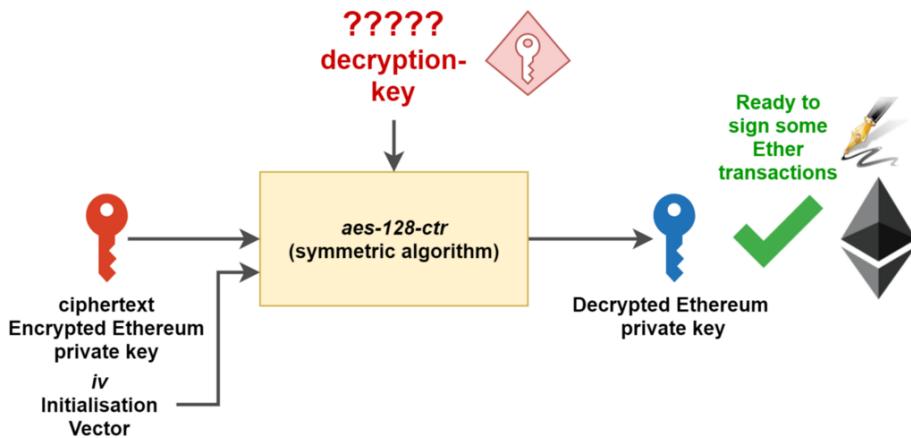
se utiliza en el *keystore* para que el usuario pueda cifrar el propio *keystore* con una contraseña.

- **kdfparams:** Parámetros necesarios para la función KDF.
- **mac:** Es un Código de Autenticación de Mensaje, o *Message Authentication Code*(MAC)[39]. Es una porción de información utilizada para autenticar un mensaje. En este caso, autentica la contraseña.

Pasemos a ver, los pasos y procedimientos que sigue un *keystore* para firmar transacciones.

### Cifrado de la clave privada

Como se ha mencionado en la sección anterior (**Wallet**), una cuenta de Ethereum tiene un par de claves pública-privada. Para mantener la clave privada a buen recaudo, es crucial utilizar un cifrado muy fuerte (AES). Este cifrado simétrico, permite cifrar información con una contraseña, y descifrarla con la misma contraseña (esta no es la contraseña del usuario, sino que es el resultado de aplicar a la contraseña del usuario la función de derivación KDF). Entonces, si juntamos la clave privada de Ethereum cifrada, con el algoritmo AES y nuestra contraseña, obtenemos la clave privada de Ethereum descifrada y lista para firmar una transacción en la red blockchain [3.46](#). Sin embargo, la contraseña que se utiliza para descifrar la clave privada, ha pasado antes por el *KDF*. Veamos como se protege nuestra contraseña.



**Figura 3.46:** Descifrado de la clave privada con AES ([Julian M](#))

### Protección de la contraseña

Como hemos mencionado antes, la contraseña del usuario pasa previamente por una función de derivación de claves. El resultado de esta función es lo que se utiliza (junto con AES) para descifrar la clave privada (como se ha mencionado antes). Para lograr esto, se le pasan a la función de derivación unos parámetros, en este caso se esta utilizando la función **scrypt**[\[55\]](#) a la que se le pasan varios parámetros.

- **dklen:** Longitud en octetos de la contraseña derivada (es decir, que tan largo tiene que ser el resultado de *scrypt*).
- **n:** Coste de CPU y Memoria.
- **r:** Tamaño del bloque.
- **p:** Parámetro de paralelización.
- **salt:** Número único generado aleatoriamente.

Todo lo anterior se ejecuta junto con la contraseña del usuario [3.47](#)

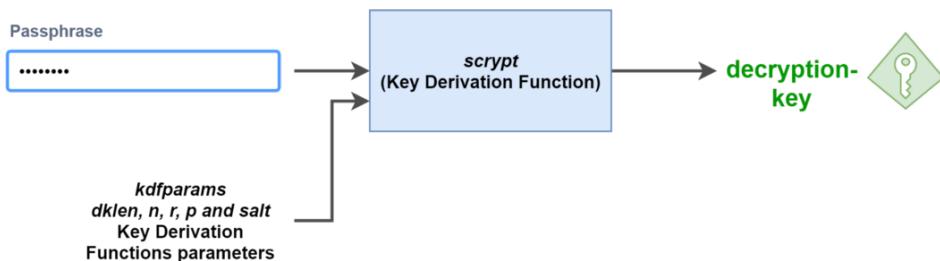


Figura 3.47: KDF de una contraseña (Julian M)

### Verificación de la contraseña

Para asegurarnos de que la contraseña que introduce un usuario es correcta, se utiliza el valor de MAC. Justo después de ejecutar el KDF sobre la contraseña, se cogen los 16 Bytes que están más a la izquierda del resultado y se concatenan con el parámetro **ciphertext**. Una vez hecho esto, se hashean con **SHA3-256**<sup>[56]</sup> y se compara con **MAC**. Si la comparación sale positiva, la contraseña ha sido introducida correctamente, y si sale negativa, no [3.48](#).

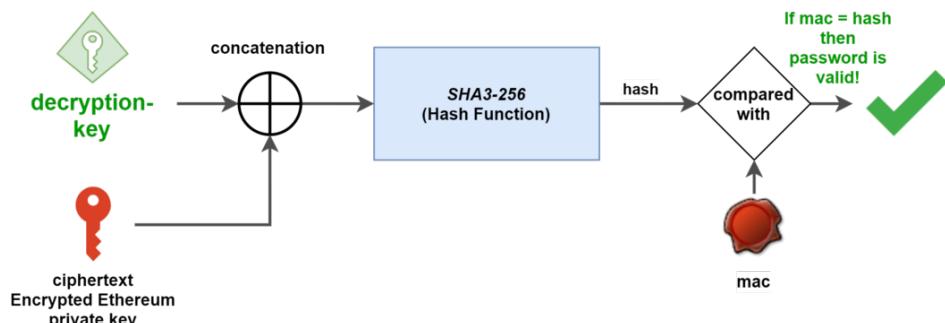


Figura 3.48: Comprobación de contraseña correcta con MAC (Julian M)

Un flujo completo con todas las partes anteriores juntas puede verse en el [Anexo](#), en concreto en [7.6](#).

#### 3.8.2. Keystore en Estublock

En “Estublock” el keystore se crea cuando el usuario se registra en la aplicación móvil por primera vez. Se le crea una **LightWallet**. Este wallet es de rápida creación comparado al wallet por defecto, muy útil en para aplicaciones móviles pues no disponen de la misma cantidad de recursos que un ordenador. (Si en vez de ser aplicación móvil, fuese aplicación de escritorio, sería conveniente utilizar las wallet “noramles” o por defecto). La **diferencia** radica en la función de derivación de claves. Como sabemos, la contraseña del usuario pasa por el KDF antes de cifrar el keystore con el algoritmo AES. 2 de los valores del KDF eran la **n** y la **p**. Estos son los dos valores que varian entre creación por defecto y creación ligera o **LightWallet**. En el caso de creación por defecto el valor de **n** es  $1^{18}$  frente a creación *light* que el valor es **n** igual a  $1^2$ . Con respecto a **p** tenemos 1 para creación por defecto y 6 para creación *light*. Con respecto a la clave privada, esta se crea también cuando el usuario hace login, utilizando el algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm) como se describió en el apartado del [Wallet](#)

El keystore se guarda en el dispositivo móvil del usuario. En concreto en una carpeta privada dentro del entorno de la aplicación llamada *files*, en la siguiente figura [3.49](#) se puede apreciar que hay 3 keystores creados (pues hay 3 cuentas en el dispositivo móvil). Este almacenamiento interno es específico de la app, es decir, no puede ser accedido por otras aplicaciones que pueda tener el usuario en el dispositivo móvil. Esto evita que una aplicación maliciosa pueda tener acceso al keystore. El

sistema Android evita que otras apps accedan a las ubicaciones internas (cifrando estas a partir de Android 10), lo que permite almacenar datos sensibles con seguridad y que solo la aplicación móvil en cuestión (“Estublock”) tenga acceso a dichos datos. La forma que tiene Android de evitar que otras aplicaciones accedan al keystore es utilizando los permisos de Linux. Recordemos que Android es un sistema operativo que funciona gracias al sistema operativo *GNU/Linux*[37]. En linux, los usuarios del sistema tienen diferentes permisos de lectura, escritura y ejecución según la carpeta o archivo al que quieren acceder[36]. En el caso del keystore, como se ve a la derecha en la figura 3.49, los permisos de acceso que tienen los archivos UTC (keystore) son de lectura y escritura **únicamente** para el usuario “Estublock” es decir, para la aplicación móvil que se ha desarrollado. Ninguna otra aplicación tiene acceso a esos archivos.

```

com.example.estublock
├── cache
│   ├── volley
│   ├── code_cache
│   └── databases
└── files
    ├── UTC--2021-05-20T11-01-44.461000000Z--8ad7fac86e11cd8af15a58b4729d686b1a71d9db.json -rw-----
    ├── UTC--2021-05-20T11-03-05.419000000Z--7521fe4ef2596b6701a8157795d5ef14531d041d.json -rw-----
    ├── UTC--2021-05-20T16-20-37.464000000Z--585137d69931724c0a0e057b804c2be749ba4938.json -rw-----
    ├── no_backup
    └── shared_prefs
        └── UserToWallet.xml -rw-rw-----

```

Figura 3.49: Sistema de archivos interno de la aplicación

*¿Cuándo desbloquea Estublock el keystore del usuario?*, al hacer login el usuario introduce su correo y contraseña. Al verificar la correcta identidad del usuario en la base de datos, se guarda la contraseña en una variable del archivo *GlobalState.java*. Esta contraseña se utilizará en caso necesario para desbloquear el keystore y poder firmar transacciones. Esta contraseña se guarda **únicamente** en tiempo de ejecución, una vez el usuario cierre la aplicación, se elimina, no quedando rastro de ella en ninguna parte.

### 3.9. DOCUMENTACIÓN

La documentación es tan importante como la implementación, hay debate sobre si la documentación es una obligación o una recomendación muy positiva. La experiencia desarrollando, nos ha demostrado que una buena documentación es clave para que un programa pueda seguir siendo desarrollado, entendido, mantenido, y sirva a otros desarrolladores. Una buena documentación garantiza un producto final de mayor calidad. Aunque no hay un método estricto en cuanto a “cuándo hay que poner comentarios”, se ha decidido poner comentarios al inicio de la clase, de los métodos y de las variables de clase. Así como en fragmentos de código no evidentes y en los que se hace algo raro.

Aunque la documentación dentro del código es un método correcto de documentar, un programador no tiende a mirar el código fuente, sino una documentación externa. Existen múltiples métodos para documentar y generar documentación, **Read the Docs**[72] por ejemplo simplifica el proceso de documentación automatizando la construcción, versionado y “hosting” de la documentación de tus programas. Es gratis, siempre actualizado y permite descargar la documentación en varios formatos, así como mostrar documentación de varias versiones del proyecto. Por otro lado tenemos **JavaDocs**[33] es el formato por defecto para todo proyecto Java, permite generar una página web con la documentación que hay en los comentarios que hay en el programa. También es gratis pero no tiene opción de “hosting”, es el programador el que debe dar el servicio online o que cada programador se genere la documentación en local. Por otro lado, para el futuro de la aplicación, en el caso de que se

migre el código a *Kotlin* tenemos **Dokka**[\[23\]](#), permite generar documentación de la misma forma que *javadoc*s pero para *Kotlin*. Por último otra opción es **Doxxygen**[\[68\]](#) es una herramienta de generación de documentación al igual que las anteriormente mencionadas, para lenguajes como *C*, *C++*, *Java*, *PHP*, *Python*...

Para documentar el SDK se ha decidido utilizar **javadoc**s pues es muy fácil de utilizar, es fácil documentar el código con la sintaxis de javadocs, y cualquier persona puede generar la documentación o mirar el código fuente para saber que hace cada función, que parámetros se esperan y que datos devuelve. La documentación del código se puede encontrar en el Anexo [7.4](#)



---

## CAPÍTULO 4

# Análisis de Impacto

---

En 2015, la ONU aprobó la *Agenda 2030 sobre el Desarrollo Sostenible*[35], una oportunidad para que los países y sus sociedades evolucionen y mejoren la vida de todos, sin dejar a nadie atrás. Constituye un llamamiento universal a la acción para poner fin a la pobreza, proteger el planeta y mejorar la vida de personas y animales. Se aprobaron entonces 17 objetivos[41] a alcanzar en 15 años.

Dentro de los 17 objetivos, hay 2 muy importantes en los que la aplicación Estublock juega un papel. La tecnología blockchain aporta sin duda una gran cantidad de ventajas, como la fiabilidad, transparencia de las operaciones (el código de los smart contracts esta a disposición de la gente), seguridad de los datos, inmutabilidad, anonimato... Pero, estas ventajas traen consigo una pequeña desventaja a nivel medioambiental. Como sabemos, una red blockchain es *descentralizada*, esto se debe a que hay varios nodos conectados entre si. Cada nodo es un ordenador en constante ejecución lo que repercute en el consumo de electricidad. Esto multiplicado por cada ordenador que hay en la red blockchain. Por ejemplo, la red de bitcoin dispone a día de hoy 2021-05-24 de 9786 nodos[24] los cuales consumen bastante energía<sup>4.1</sup>.

Por suerte, aunque parezca que blockchain va a consumir tanta energía como el *Centro de Datos de Todo el Mundo*, en 2019 Coinshares realizó una investigación[59] que reveló que el 74 % de las operaciones mineras se realizan con energía renovable, por lo que el impacto medioambiental es menos grave de lo que podría ser. Sin embargo, siempre se puede mejorar, y llegar al 100 % de consumo con energías renovables. Ethereum versión 2[62] es la alternativa verde a este problema, al cambiar el algoritmo de consenso de *Proof of Work* a *Proof of Stake*, ethereum disminuirá casi completamente el consumo de energía. Hasta tal punto, de que **Joseph Pallant**, fundador de la *blockchain for climate foundation*[6] (una fundación que busca utilizar la tecnología blockchain para lidiar con problemas relacionados con el cambio climático) utiliza Ethereum2.0. Aunque esta segunda versión de Ethereum ya está en funcionamiento, se requiere de un tiempo para migrar los datos de la versión 1 a la 2, pero con el tiempo la versión 2 de ethereum crecerá, disminuyendo enormemente el gasto energético que conlleva.

Aunque Estublock utiliza actualmente la red de Ethereum1.0, su gasto energético es de todos modos casi nulo (pues no hay suficiente tráfico como para que suponga un gasto energético grande), pero hay que tener en cuenta que aunque la red gaste poco, hay varios ordenadores encendidos consumiendo energía, por lo que habrá que migrar a Ethereum2.0 cuando sea posible. Aunque el gasto energético es muy bajo, no se debe olvidar que gastar energía, gasta y que sería recomendable lograr que los ordenadores estén lo más especializados posible para minimizar al máximo el gasto energético cumpliendo con el objetivo 9.4 “modernizar la infraestructura para que sean sostenibles”. “Estublock” tiene también un impacto en el punto 13 “Acción por el Clima” con respecto al objetivo de minimizar las emisiones de carbono al máximo, en las universidades en las que se utilice la aplicación “Estublock” y dispongan de nodos en la red blockchain, se puede tratar de utilizar siempre energías renovables como poner paneles solares en la universidad para que los nodos de la red utilicen esa energía renovable para funcionar.

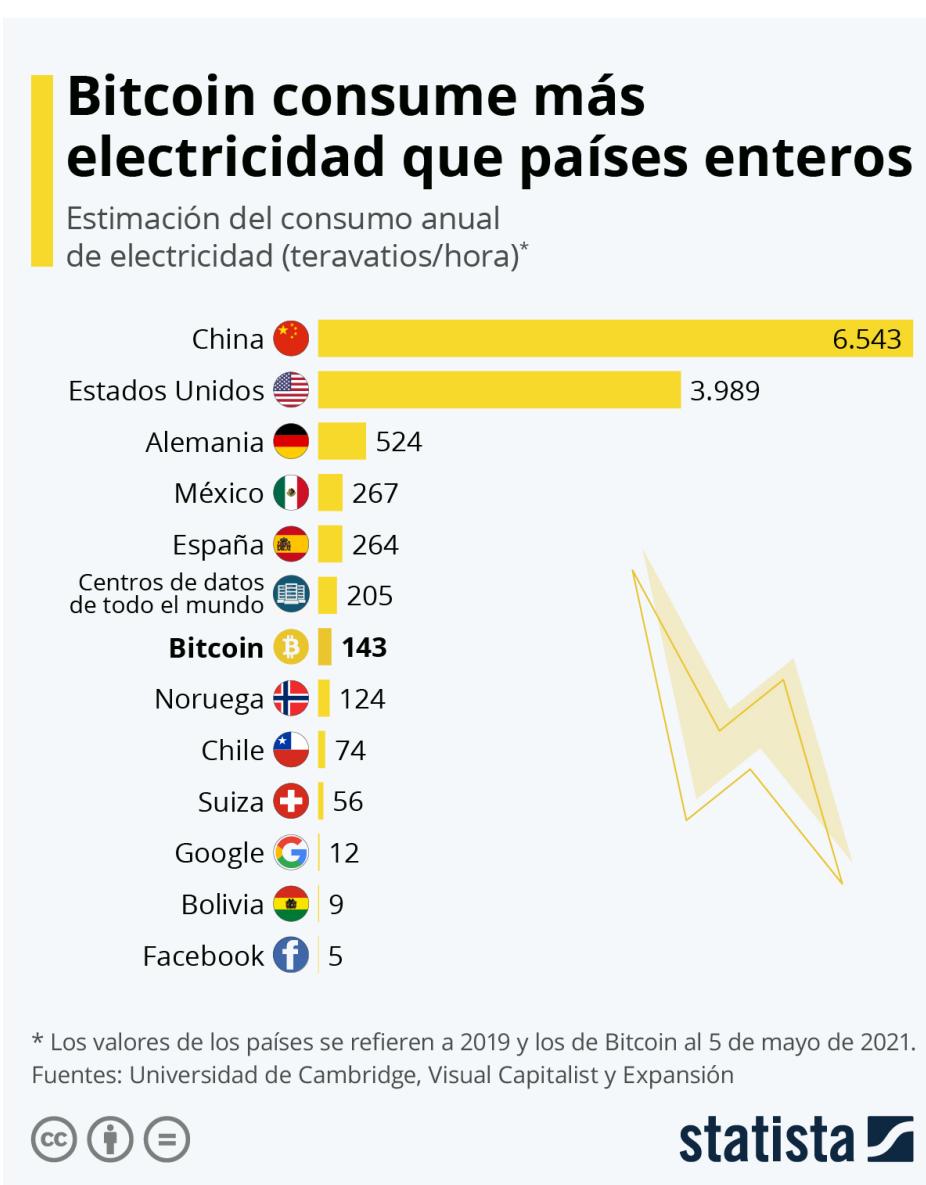


Figura 4.1: Consumo eléctrico de bitcoin (Statista)

---

## CAPÍTULO 5

# Conclusiones

---

La tecnología blockchain ha venido para quedarse. Cada día más empresas toman la decisión de utilizar su potencial y cada vez más, la población sabe de su existencia. La popularidad de blockchain se debe a que es un sistema seguro, fiable, transparente, descentralizado y flexible. Permite hacer transacciones bancarias, validar la identidad de un usuario, registrar la reserva de un billete de avión, etc. Para explotar esta tecnología y su creciente popularidad, han nacido aplicaciones web y móvil para que las personas puedan aprovechar las redes blockchain. Tenemos por ejemplo la aplicación móvil *Guts* para la compra y venta de entradas a eventos (cine, conciertos, teatro, etc). *LifeID*, plataforma desde la que gestionar tu identidad digital con total seguridad o *Voatz* que permite participar en votaciones a la gente desde sus dispositivos móviles, votaciones importantes como votar al presidente de un país.

Después de aprender desde cero a desarrollar mi primera aplicación Android, y aprender a hacer llamadas desde esta a la red blockchain, he echado en falta un mayor soporte de las librería que he utilizado y que he encontrado disponibles para dispositivos Android. Las librerías que he probado funcionan muy bien, pero no son específicas para aplicaciones Android. Esto hace que algunas funciones sean lentas, pesadas y que requieran de una potencia de cálculo que un móvil puede no tener. Por ejemplo, las llamadas a redes blockchain han de hacerse en un Thread diferente del principal para evitar que la pantalla de la aplicación se congele. El problema con Android, es que aunque este programado en Java, una librería de Java como *web3j* (la usada en este proyecto para comunicar con la red blockchain) no esta optimizada para Android, simplemente funciona bien con Java, pero hay que retocarla un poco para que funcione bien en Android. Por esta misma razón, el SDK que se ha implementado sí tiene en cuenta esta necesidad de enfocarse a aplicaciones Android.

Durante el desarrollo de este TFG se ha tenido la oportunidad de poner en práctica el funcionamiento de la red blockchain levantada para la aplicación móvil en el registro de asistencias del congreso anual TryIT que se celebra en nuestra facultad, así como en el taller de machine learning que se impartió en la Cátedra Inetum. En ambos casos, la red blockchain ha respondido bien, registrando perfectamente las asistencias. Esto es muy bueno, pues las mismas llamadas utilizadas para estos eventos, son las que el dispositivo móvil realiza. Esto junto con las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación móvil, da pie a confirmar que se ha cumplido con el objetivo principal de este trabajo. Este es, registrar las asistencias a eventos con una aplicación móvil guardando la información en la red blockchain. Y para esto último desarrollar un SDK que también se ha logrado terminar con éxito. Lógicamente a la aplicación le queda mucho por crecer y avanzar, se verá mas a fondo en el apartado [A Futuro](#).

Personalmente, el tiempo realizando este trabajo de fin de grado ha pasado volando. Esto solo puede significar una cosa, he disfrutado mucho, he aprendido mucho, y he mejorado mucho. Además, no solo me ha gustado, sino que espero con ansia que la aplicación Estublock y el SDK sigan siendo desarrollados y sigan creciendo, y deseo ver en un par de años este proyecto en uso por alumnos de varias universidades de Madrid. Con este TFG, a parte de ver mejoradas mis habilidades programando

en varios lenguajes, utilizando varias librerías y solventando muchos problemas, también he tenido que aprender a diseñar un proyecto, diseñar un SDK, estructurar y diseñar las pantallas poniéndome en la piel del usuario final. He redescubierto la metodología *SCRUM*, he descubierto herramientas de gestión de tareas como *taskwarrior* y gestión del tiempo con *timeboxing*, y muchas más cosas que he usado y aprendido de forma inconsciente. Dicho de otro modo, este TFG marca un antes y un después en mi vida como desarrollador, pues ahora me veo capaz de enfrentarme a cualquier proyecto, pues por duro que sea, sé que mis ganas de aprender y mi curiosidad me harán seguir adelante.

---

## CAPÍTULO 6

# Trabajo Futuro

---

Actualmente la aplicación “Estublock” puede utilizarse para registrar las asistencias a eventos en una universidad sin problema. Sin embargo es largo el camino que le queda para poder decir que esta terminada y que se pueda lanzar la primera versión de la aplicación. Ahora mismo es lo que se conoce como “una versión alpha”. Esta lista para ser probado pero no para que se utilice de forma masiva. ¿Qué avances y siguientes pasos necesita la aplicación para poder lanzar la primera versión *V.1.0.0?*

Actualmente el código es ligeramente caótico, durante los meses de desarrollo ha ido sufriendo muchos cambios, renombrado de archivos, creación y eliminación de código y archivos que en su momento tenían utilidad pero ya no. Y todo esto causa que tanto código como archivos, no estén tan ordenados y organizados como deberían. Lo ideal es entonces ordenarlo en carpetas con nombres significativos y eliminando toda repetición de código así como archivos que no hagan falta o crear archivos que pueden hacer que el código sea más mantenible. Esto es fundamental, pues para un futuro programador es más fácil coger un proyecto ordenado y estructurado siguiendo un “modus operandi”, a coger un proyecto caótico. Además, ordenar el código reduce errores al minimizar las líneas de código (por quitar código repetido) y permite tener mayor control de las funciones que causan problemas.

Como toda pieza de software, se necesitan tests para probar el correcto funcionamiento de la aplicación. Con una buena batería de tests, el programador puede probar el código que escribe, reduciendo los errores y reduciendo el tiempo que se tarda en corregirlos. Esto se debe a que los errores se detectan con más antelación y de forma más concreta. Dando lugar a un código mas estable, escalable y sostenible en el tiempo. Como no se disponía de experiencia en el desarrollo de aplicaciones móvil, ha sido difícil crear tests unitarios pues el código sufría muchos cambios diariamente con muchas modificaciones en el nombre de variables. Ahora que el código es más estable, y se tiene más experiencia, es un buen momento para desarrollar estos tests unitarios.

La documentación es también uno de los pilares de un buen software. Actualmente las funciones tienen pequeños comentarios, excepto las del SDK que están completamente comentadas. Sería ideal añadir comentarios completos en el resto de funciones siguiendo el estilo de comentarios de javadocs para poder generar posteriormente con esta herramienta una documentación más detallada. Muchas partes del código se explican por si mismas, pero la documentación es fundamental para que un proyecto sea mantenido en el tiempo, principalmente para que los futuros desarrolladores entiendan el código ya existente y no pierdan tiempo en averiguar que es lo que hace el código.

Con respecto a la interfaz de usuario, esta se ha mantenido muy sencilla. Pero una mejora en el diseño o un remodelado visual pueden hacer que la aplicación sea mucho más atractiva, haciendo que los usuarios disfruten mucho más con su uso. Así como, mejorar la experiencia de los usuarios al utilizarla. Para ello, se puede por ejemplo añadir una opción que permita al usuario personalizar a su gusto el tema de la aplicación, o alguna pantalla o botón que quieran poder mover de sitio.

Como se ha mencionado antes, la aplicación permite actualmente registrar las asistencias a eventos, así como crear nuevos eventos, aunque estas sean sus funcionalidades principales, se puede explotar la aplicación mucho más añadiendo nuevas funcionalidades como permitir modificar los datos personales a los usuarios, listar los eventos a los que se ha asistido a lo largo del tiempo o ver los usuarios que han asistido a un evento concreto. También, muy importante, ha de hacerse la diferencia entre alumno y profesor, mostrando distintas pantallas y funcionalidades según el usuario sea un profesor o un alumno.

Un punto muy interesante que puede tratarse también es la utilización de una identidad digital soberana como la que se está desarrollando en *Alastrá*[[65](#)] también con tecnología blockchain. Con esta identidad soberana se podría prescindir por completo de una base de datos que guarde información sobre los alumnos, dependiendo por completo del sistema de identidad soberana que se está desarrollando en Alastria. Y además, tener la seguridad y fiabilidad que aporta el uso de la tecnología blockchain.

Por último, si miramos más en el futuro. Uno de los objetivos estrella del proyecto “Estublock” es llegar a todos los campus de la Universidad Politécnica de Madrid, y posteriormente, al resto de universidades de Madrid y España. Aunque sea un objetivo muy lejano, no hay olvidarlo, este proyecto puede tener un impacto muy positivo en las universidades. Por ello, aunque sea lejano, hay que tener en mente que queremos llegar a todo el mundo.

# Bibliografía

---

- [1] *Advanced Encryption Standard*, TutorialsPoint. dirección: [https://www.tutorialspoint.com/cryptography/advanced\\_encryption\\_standard.htm](https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm) (visitado 02-06-2021).
- [2] *Android Auto Web*, Android. dirección: [https://www.android.com/intl/es\\_es/auto/](https://www.android.com/intl/es_es/auto/) (visitado 25-05-2021).
- [3] *Aplicaciones, utilidad y casos de uso del blockchain con ejemplos de 2021*, Criptomonedas.ninja. dirección: <https://criptomoneda.ninja/aplicaciones-blockchain/#criptomonedas> (visitado 25-03-2021).
- [4] *Arcblock and lifeID announce partnership to bring digital identity tools to blockchain3.0 developer ecosystem*, lifeID. dirección: <https://medium.com/lifeid/arcblock-and-lifeid-announce-partnership-to-bring-digital-identity-tools-to-blockchain-3-0-5296c60a0940> (visitado 26-03-2021).
- [5] *ArcBlock web page*, ArcBlock. dirección: <https://www.arcblock.io/en/> (visitado 26-03-2021).
- [6] *Blockchain for Climate Foundation*, BfCF. dirección: <https://www.blockchainforclimate.org/> (visitado 24-05-2021).
- [7] D. Bradburd, *How Much Power It Takes to Create a Bitcoin*, The Balance. dirección: <https://www.thebalance.com/how-much-power-does-the-bitcoin-network-use-391280> (visitado 23-03-2021).
- [8] M. Calvo, *Conoce los diferentes tipos de blockchain*, Blockchain Services. dirección: <https://www.blockchainservices.es/novedades/conoce-los-diferentes-tipos-de-blockchain/> (visitado 23-03-2021).
- [9] *Carrefour lanza el primer blockchain alimentario en España*, Carrefour. dirección: <https://www.carrefour.es/grupo-carrefour/sala-de-prensa/noticias2015.aspx?tcm=tcm:5-50248> (visitado 25-03-2021).
- [10] *Código fuente del Smart Contract de GET*, GET Protocol. dirección: <https://etherscan.io/address/0x4cd90231a36ba78a253527067f8a0a87a80d60e4#code> (visitado 26-03-2021).
- [11] L. Conway, *Blockchain Explained*, Investopedia. dirección: <https://www.investopedia.com/terms/b/blockchain.asp#storage-structure> (visitado 22-03-2021).
- [12] *Criptografia, Guias de Android*, Android. dirección: <https://developer.android.com/guide/topics/security/cryptography> (visitado 23-05-2021).
- [13] *Cryptographic Extraction and Key Derivation: The HKDF Scheme*, Hugo Krawczyk. dirección: <https://eprint.iacr.org/2010/264.pdf> (visitado 02-06-2021).
- [14] *Documentación de Android*, Google. dirección: <https://developer.android.com/guide> (visitado 23-04-2021).
- [15] *Documentación de C++*, Microsoft. dirección: <https://docs.microsoft.com/en-us/cpp/cpp/basic-concepts-cpp?view=msvc-160> (visitado 23-04-2021).
- [16] *Documentación de Geth*, Geth EthHub. dirección: <https://docs.ethhub.io/using-ethereum/ethereum-clients/geth/> (visitado 28-05-2021).

- [17] *Documentación de Java*, Oracle. dirección: <https://docs.oracle.com/en/java/> (visitado 23-04-2021).
- [18] *Documentación de Kotlin*, Kotlin. dirección: <https://kotlinlang.org/docs/home.html> (visitado 23-04-2021).
- [19] *Empresa JFrog*, JFrog. dirección: <https://jfrog.com/> (visitado 28-05-2021).
- [20] *Flutter web page*, Google. dirección: <https://flutter.dev/> (visitado 23-04-2021).
- [21] *GET Protocol web page*, GET Protocol. dirección: <https://get-protocol.io/> (visitado 26-03-2021).
- [22] *Github de btcrecover*, Gurnec. dirección: <https://github.com/gurnec/btcrecover> (visitado 28-05-2021).
- [23] *Github de Dokka*, Kotlin. dirección: <https://github.com/Kotlin/dokka> (visitado 28-05-2021).
- [24] *Global Bitcoin Nodes Distribution*, Bitnodes. dirección: <https://bitnodes.io/> (visitado 24-05-2021).
- [25] J. S. González, *Cuenta de Github*. dirección: <https://github.com/FORGIS98> (visitado 22-05-2021).
- [26] *Guía de Android sobre Espresso*, Android. dirección: <https://developer.android.com/training/testing/espresso> (visitado 02-06-2021).
- [27] *Guía de Android sobre UI Automator*, Android. dirección: <https://developer.android.com/training/testing/ui-automator> (visitado 02-06-2021).
- [28] *Gwei WebPage*, Gwei Ethereum. dirección: <https://gwei.io/es/> (visitado 24-03-2021).
- [29] A. Hertig, *What is Proof-of-Work*, Tech. dirección: <https://www.coindesk.com/what-is-proof-of-work> (visitado 22-03-2021).
- [30] *Intro to the ethereum stack*, Ethereum. dirección: <https://ethereum.org/en/developers/docs/ethereum-stack/> (visitado 25-03-2021).
- [31] *Introduction to smart contracts*, Ethereum. dirección: <https://ethereum.org/en/developers/docs/smart-contracts/> (visitado 24-03-2021).
- [32] *IPFS web page*, IPFS. dirección: <https://ipfs.io/> (visitado 26-03-2021).
- [33] *Javadoc*, Oracle. dirección: <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html> (visitado 28-05-2021).
- [34] *Keystore File*, Ethereum. dirección: <https://ethereum.org/en/glossary/#keystore-file> (visitado 02-06-2021).
- [35] *La Agenda para el Desarrollo Sostenible*, ONU. dirección: <https://www.un.org/sustainabledevelopment/es/development-agenda/> (visitado 24-05-2021).
- [36] *Linux User Permission*, Alokananda Ghoshal. dirección: <https://www.educba.com/linux-user-permission/> (visitado 02-06-2021).
- [37] *Linux y el sistema GNU*, Richard Stallman. dirección: <https://www.gnu.org/gnu/linux-and-gnu.html> (visitado 02-06-2021).
- [38] Matt Hussey, Scott Chipolina, *What are Dapps*, Decrypt. dirección: <https://decrypt.co/resources/dapps> (visitado 26-03-2021).
- [39] *Message Authentication*, TutorialsPoint. dirección: [https://www.tutorialspoint.com/cryptography/message\\_authentication.htm](https://www.tutorialspoint.com/cryptography/message_authentication.htm) (visitado 02-06-2021).
- [40] *Minar bitcoins, ¿en qué consiste y cómo funciona?*, bit2me Academy. dirección: <https://academy.bit2me.com/que-es-minar-bitcoins/> (visitado 22-03-2021).
- [41] *Objetivos de Desarrollo Sostenible*, ONU. dirección: <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/> (visitado 24-05-2021).
- [42] *Página de bitcoin*, Bitcoin. dirección: <https://bitcoin.org/es/> (visitado 23-03-2021).

- [43] *Página de Corda*, Corda. dirección: <https://www.corda.net/> (visitado 23-03-2021).
- [44] *Página de ethereum*, Ethereum. dirección: <https://ethereum.org/es/> (visitado 23-03-2021).
- [45] *Página de Hyperledger*, Hyperledger. dirección: <https://www.hyperledger.org/> (visitado 23-03-2021).
- [46] *Página de Quorum*, Quorum. dirección: <https://consensys.net/quorum/> (visitado 23-03-2021).
- [47] *Página de R3*, R3. dirección: <https://www.r3.com/> (visitado 23-03-2021).
- [48] *Página de Ripple*, Ripple. dirección: <https://ripple.com/> (visitado 23-03-2021).
- [49] A. Preukschat, *Los tipos de blockchain: públicas, privadas e híbridas*, INETUM. dirección: <https://inetum.com.es/es/blog/Post/Los-tipos-de-Blockchain-publicas-privadas-e-hibridas-y-II/> (visitado 23-03-2021).
- [50] *Publishing your first Android library to MavenCentral*, WaseefAkhtar. dirección: <https://www.waseefakhtar.com/android/publishing-your-first-android-library-to-mavencentral/> (visitado 23-05-2021).
- [51] F. Reporter, *TUI Group Uses Blockchain*, FT Reporter. dirección: <https://ftreporter.com/tui-group-uses-blockchain/> (visitado 25-03-2021).
- [52] *Repositorio de GitHub de Mockito*, Mockito. dirección: <https://github.com/mockito/mockito> (visitado 02-06-2021).
- [53] N. Rodriguez, *Algoritmos de Consenso: La raíz de la tecnología blockchain*, 101Blockchain. dirección: <https://101blockchains.com/es/algoritmos-de-consenso-blockchain/#1> (visitado 23-03-2021).
- [54] *Santander y Broadridge utilizan por primera vez tecnología blockchain para votar en una junta general de accionistas*, Santander y Broadridge. dirección: <https://www.santander.com/content/dam/santander-com/es/documentos/historico-notas-de-prensa/2018/05/NP-2018-05-17-Santander%20y%20Broadridge%20utilizan%20por%20primera%20vez%20tecnolog%C3%A1Da%20blockchain%20para%20votar%20en%20una%20ju-es.pdf> (visitado 25-03-2021).
- [55] *Scrypt manual page*, IETF. dirección: <https://datatracker.ietf.org/doc/html/rfc7914> (visitado 31-05-2021).
- [56] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, National Institute of Standards y Technology. dirección: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf> (visitado 02-06-2021).
- [57] *Solidity Welcome Page*, Ethereum/Solidity. dirección: <https://soliditylang.org/> (visitado 25-03-2021).
- [58] Tal Yellin, Dominic Aratari, Jose Pagliery, *What is bitcoin?*, CNN. dirección: <https://money.cnn.com/infographic/technology/what-is-bitcoin/index.html> (visitado 22-03-2021).
- [59] *The Bitcoin Mining Network*, CoinShares. dirección: <https://coinshares.com/research/bitcoin-mining-network-june-2019> (visitado 24-05-2021).
- [60] *Todas las criptodivisas*, CoinMarketCap. dirección: <https://coinmarketcap.com/es/all/views/all/> (visitado 25-03-2021).
- [61] *Tolerancia a faltas bizantinas*, Eugene Krasinsky. dirección: <https://freeton.house/es/tolerancia-a-faltas-bizantinas-el-metodo-del-consenso-de-free-ton/> (visitado 29-05-2021).
- [62] *Upgrading Ethereum to radical new heights*, Ethereum. dirección: <https://ethereum.org/en/eth2/> (visitado 24-03-2021).
- [63] *Visualización de las transacciones realizadas con el token GET*, GET. dirección: <https://etherscan.io/token/0x8a854288a5976036a725879164ca3e91d30c6a1b> (visitado 26-03-2021).

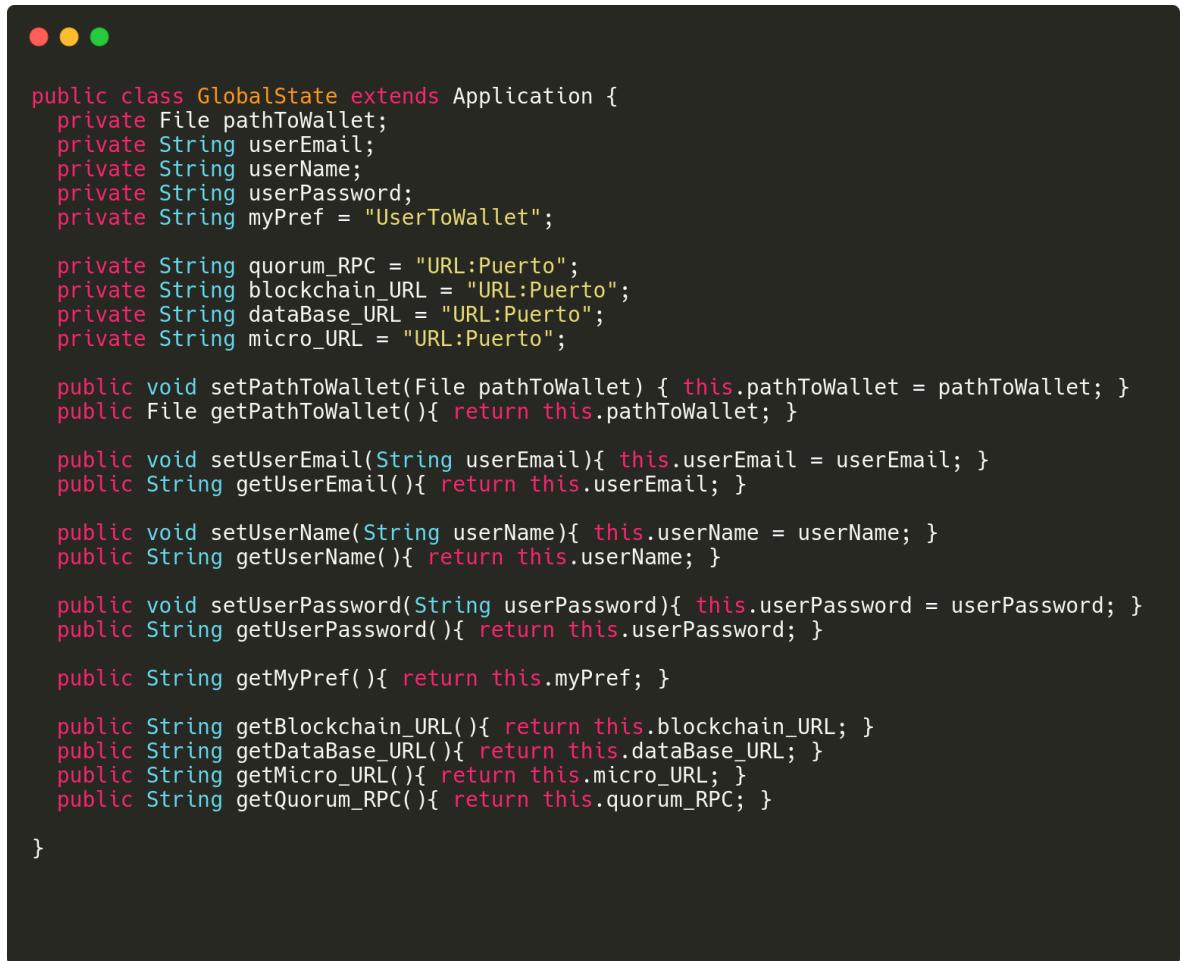
- [64] *Vyper Doc Page*, Ethereum/Vyper. dirección: <https://vyper.readthedocs.io/en/stable/> (visitado 25-03-2021).
- [65] *Web de Alastria*, Alastria. dirección: <https://alastria.io/home> (visitado 02-06-2021).
- [66] *Web de Android*, Android. dirección: [https://www.android.com/intl/es\\_es/](https://www.android.com/intl/es_es/) (visitado 25-05-2021).
- [67] *Web de Civic*, Civic. dirección: <https://www.civic.com/> (visitado 25-03-2021).
- [68] *Web de Doxygen*, Doxygen. dirección: <https://www.doxygen.nl/index.html> (visitado 28-05-2021).
- [69] *Web de ganache*, Truffle Suite. dirección: <https://www.trufflesuite.com/ganache> (visitado 28-05-2021).
- [70] *Web de Niuron*, Niuron. dirección: <http://niuron.io/> (visitado 25-03-2021).
- [71] *Web de NxT*, NxT. dirección: <http://nxtcrypto.org/> (visitado 24-03-2021).
- [72] *Web de read the docs*, Read the Docs. dirección: <https://readthedocs.org/> (visitado 28-05-2021).
- [73] *Web de Wallet Recovery*, Wallet Recovery. dirección: <https://walletrecovery.info/> (visitado 28-05-2021).
- [74] *Web de WRS*, Wallet Recovery Service. dirección: <https://www.walletrecoveryservices.com/> (visitado 28-05-2021).
- [75] *Web oficial de Marvelapp*, Marvel. dirección: <https://marvelapp.com/> (visitado 22-05-2021).
- [76] *Web oficial de Maven Central*, Maven. dirección: <https://mvnrepository.com/> (visitado 23-05-2021).
- [77] *Web oficial de NodeJS*, NodeJS. dirección: <https://nodejs.org/en/> (visitado 21-05-2021).
- [78] *Web oficial de Sonatype*, Sonatype. dirección: <https://www.sonatype.com/> (visitado 23-05-2021).
- [79] *Web oficial de Swagger*, Swagger Supported by SMARTBEAR. dirección: <https://swagger.io/> (visitado 21-05-2021).
- [80] *Web3 JavaScript API Page*, Web3. dirección: <https://web3js.readthedocs.io/en> (visitado 25-03-2021).
- [81] *What is a Blockchain*, Coinbase. dirección: <https://www.coinbase.com/es/learn/crypto-basics/what-is-a-blockchain> (visitado 29-05-2021).
- [82] *What is a golden nonce and what is its usage in blockchain?*, Toshendra Kumar Sharma. dirección: <https://www.blockchain-council.org/blockchain/what-is-a-golden-nonce-and-what-is-its-usage-in-blockchain/> (visitado 29-05-2021).
- [83] *What is a hash function*, IONOS. dirección: <https://www.ionos.com/digitalguide/server/security/hash-function/> (visitado 22-03-2021).
- [84] *What is Blockchain*, Oracle. dirección: <https://www.oracle.com/blockchain/what-is-blockchain/> (visitado 29-05-2021).
- [85] *What is blockchain technology*, IBM. dirección: <https://www.ibm.com/blockchain/what-is-blockchain> (visitado 22-03-2021).
- [86] *What is ECDSA Encryption? How does it work?*, EncryptionConsulting. dirección: <https://www.encryptionconsulting.com/education-center/what-is-ecdsa/> (visitado 23-05-2021).

---

## CAPÍTULO 7

# Anexo

---



```
public class GlobalState extends Application {
    private File pathToWallet;
    private String userEmail;
    private String userName;
    private String userPassword;
    private String myPref = "UserToWallet";

    private String quorum_RPC = "URL:Puerto";
    private String blockchain_URL = "URL:Puerto";
    private String DataBase_URL = "URL:Puerto";
    private String micro_URL = "URL:Puerto";

    public void setPathToWallet(File pathToWallet) { this.pathToWallet = pathToWallet; }
    public File getPathToWallet(){ return this.pathToWallet; }

    public void setUserEmail(String userEmail){ this.userEmail = userEmail; }
    public String getUserEmail(){ return this.userEmail; }

    public void setUserName(String userName){ this.userName = userName; }
    public String getUserName(){ return this.userName; }

    public void setUserPassword(String userPassword){ this.userPassword = userPassword; }
    public String getUserPassword(){ return this.userPassword; }

    public String getMyPref(){ return this.myPref; }

    public String getBlockchain_URL(){ return this.blockchain_URL; }
    public String get DataBase_URL(){ return this.DataBase_URL; }
    public String getMicro_URL(){ return this.micro_URL; }
    public String getQuorum_RPC(){ return this.quorum_RPC; }
}
```

Figura 7.1: Código de GlobalState

```

private void initialiseDetectorAndSources(){
    barcodeDetector = new BarcodeDetector.Builder(this)
        .setBarcodeFormats(Barcode.QR_CODE)
        .build();

    cameraSource = new CameraSource.Builder(this, barcodeDetector)
        .setRequestedPreviewSize(1920, 1080)
        .setAutoFocusEnabled(true)
        .build();

    surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() {
        @Override
        public void surfaceCreated(@NonNull SurfaceHolder surfaceHolder) {
            try{
                if(ActivityCompat.checkSelfPermission(
                    EscanearQR.this,
                    Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED
                )
                    cameraSource.start(surfaceView.getHolder());
                else{
                    ActivityCompat.requestPermissions(EscanearQR.this, new String[]
                    {Manifest.permission.CAMERA}, REQUEST_CAMERA_PERMISSION);
                }
            } catch (Exception e){
                e.printStackTrace();
            }
        }

        @Override
        public void surfaceDestroyed(@NonNull SurfaceHolder surfaceHolder) {
            cameraSource.stop();
        }
    });

    barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {
        @Override
        public void release() {
            Toast.makeText(getApplicationContext(), "Barcode has been stopped",
            Toast.LENGTH_SHORT).show();
        }

        @Override
        public void receiveDetections(@NonNull @NotNull Detector.Detections<Barcode> detections) {
            final SparseArray<Barcode> barcodes = detections.getDetectedItems();
            if(barcodes.size() != 0){
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        // Hacer cosas cuando se escanea un QR
                    }
                }).start();
            }
        }
    });
}

@Override
protected void onPause(){
    super.onPause();
    cameraSource.release();
}

@Override
protected void onResume(){
    super.onResume();
    initialiseDetectorAndSources();
}

```

**Figura 7.2:** Código del escaneado de QRs

```
● ● ●

/**
 * Método que envía una transacción ya firmada sin listener.
 * @param signedMessage Transacción firmada lista para ser enviada
 */
public void sendSignedTransaction(@NonNull String signedMessage){
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                web3j.ethSendRawTransaction(signedMessage).send();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }).start();
}

/**
 * Método que firma y envía una transacción.
 * @param credentials Las credenciales de un usuario, viene a ser su keystore
 * @param gasPrice Cantidad que quieras pagar por unidad de gas como tarifa al minero
 * @param gasLimit Límite máximo que estas dispuesto a pagar por la transacción
 * @param to Address del SMART CONTRACT de destino
 * @param data Información que se va a mandar al smart contract
 * @param listener Callback de la llamada a sobreescibir
 */
public void signAndSendTransaction(@NonNull Credentials credentials, @NonNull String gasPrice,
    @NonNull String gasLimit, @NonNull String to, @NonNull String data){
    this.signTransaction(credentials, gasPrice, gasLimit, to, data, new EasyBlockchainListener() {
        @Override
        public void onSignTransactionEvent(@NonNull @NotNull String signedTx) {
            sendSignedTransaction(signedTx);
        }

        @Override
        public void onSendSignedTransactionEvent(@NonNull @NotNull String transactionHash) {
    });
}
```

Figura 7.3: Firmado y enviado de una transacción

```


    /**
     * Constructor de la clase.
     * @param blockchainURL La URL y puerto al nodo de la red blockchain con la que se
     * quiere comunicar.
     */
    public TransactionsHelper(@NotNull String blockchainURL);

    /**
     * Método que firma una transacción.
     * @param credentials Las credenciales de un usuario, viene a ser su keystore
     * @param gasPrice Cantidad que quieras pagar por unidad de gas como tarifa al minero
     * @param gasLimit Límite máximo que estas dispuesto a pagar por la transacción
     * @param to Address del SMART CONTRACT de destino
     * @param data Información que se va a mandar al smart contract
     * @param listener Callback de la llamada a sobrescribir
     */
    public void signTransaction(@NotNull Credentials credentials, @NotNull String gasPrice, @NotNull String gasLimit,
                               @NotNull String to, @NotNull String data, @NotNull EasyBlockchainListener listener);

    /**
     * Método que envía una transacción ya firmada con listener.
     * @param signedMessage Transacción firmada lista para ser enviada
     * @param listener Callback de la llamada a sobrescribir
     */
    public void sendSignedTransaction(@NotNull String signedMessage, @NotNull EasyBlockchainListener listener);

    /**
     * Método que envía una transacción ya firmada sin listener.
     * @param signedMessage Transacción firmada lista para ser enviada
     */
    public void sendSignedTransaction(@NotNull String signedMessage);

    /**
     * Método que firma y envía una transacción.
     * @param credentials Las credenciales de un usuario, viene a ser su keystore
     * @param gasPrice Cantidad que quieras pagar por unidad de gas como tarifa al minero
     * @param gasLimit Límite máximo que estas dispuesto a pagar por la transacción
     * @param to Address del SMART CONTRACT de destino
     * @param data Información que se va a mandar al smart contract
     * @param listener Callback de la llamada a sobrescribir
     */
    public void signAndSendTransaction(@NotNull Credentials credentials, @NotNull String gasPrice, @NotNull String gasLimit,
                                      @NotNull String to, @NotNull String data);


```

**Figura 7.4:** Documentación del TransactionsHelper

```


    /**
     * Constructor de la clase, llama a la función workaroundECDA()
     */
    public WalletHelper();

    /**
     * Método que arregla el problema con el proveedor de seguridad
     * Para saber más ir a este link: {@link https://github.com/web3j/web3j/issues/915}
     */
    private void workaroundECDA();

    /**
     * Método que crea una nueva cartera virtual
     * @param password Contraseña de la cartera virtual
     * @param walletDirectory Dirección de la nueva cartera virtual
     */
    public String createNewWallet(@NotNull String password, @NotNull File walletDirectory);

    /**
     * Método que crea una nueva cartera virtual
     * ~ cat tmp.txt
     */
    /**
     * Constructor de la clase, llama a la función workaroundECDA()
     */
    public WalletHelper();

    /**
     * Método que arregla el problema con el proveedor de seguridad
     * Para saber más ir a este link: {@link https://github.com/web3j/web3j/issues/915}
     */
    private void workaroundECDA();

    /**
     * Método que crea una nueva cartera virtual
     * @param password Contraseña de la cartera virtual
     * @param walletDirectory Dirección de la nueva cartera virtual
     */
    public String createNewWallet(@NotNull String password, @NotNull File walletDirectory);

    /**
     * Método que crea una nueva cartera virtual
     * @param password Contraseña de la cartera virtual
     * @param walletDirectory Dirección de la nueva cartera virtual
     * @return @NotNull
     */
    public String createNewWallet(@NotNull String password, @NotNull String walletDirectory);

    /**
     * Método que devuelve el address de una cartera
     * @param password Contraseña de la cartera virtual
     * @param keyStoreDirectory Dirección de la cartera virtual
     */
    public String getAddress(@NotNull String password, @NotNull File keyStoreDirectory);

    /**
     * Devuelve el address de una cartera
     * @param password Contraseña de la cartera virtual
     * @param keyStoreDirectory Dirección de la cartera virtual
     * @return address de la cartera
     */
    public String getAddress(@NotNull String password, @NotNull String keyStoreDirectory);

    /**
     * Devuelve las credenciales
     * @param password Contraseña de la cartera virtual
     * @param keyStoreDirectory Dirección de la cartera virtual
     * @return credenciales de la cartera virtual
     */
    public Credentials getCredentials(@NotNull String password, @NotNull File keyStoreDirectory);

    /**
     * Guarda en el sharedpreferences de android un identificador y la dirección del wallet
     * @param id id para la dirección del wallet
     * @param walletDirectory Dirección de la cartera virtual
     * @param prefsName nombre de las shared preferences que se quiere usar
     * @param activity Actividad de android para poder cargar las shared preferences
     */
    public void saveInPreferences(@NotNull String id, @NotNull String walletDirectory,
                                 @NotNull String prefsName, @NotNull Activity activity);

    /**
     * Devuelve el valor asociado al id
     * @param id id del que se quiere su valor
     * @param prefsName nombre de las shared preferences que se quiere usar
     * @param activity Actividad de android para poder cargar las shared preferences
     * @return Devuelve la dirección donde esta guardar la cartera virtual de la persona
     */
    public String getFromPreferences(@NotNull String id, @NotNull String prefsName, @NotNull Activity activity);


```

**Figura 7.5:** Documentación del WalletHelper

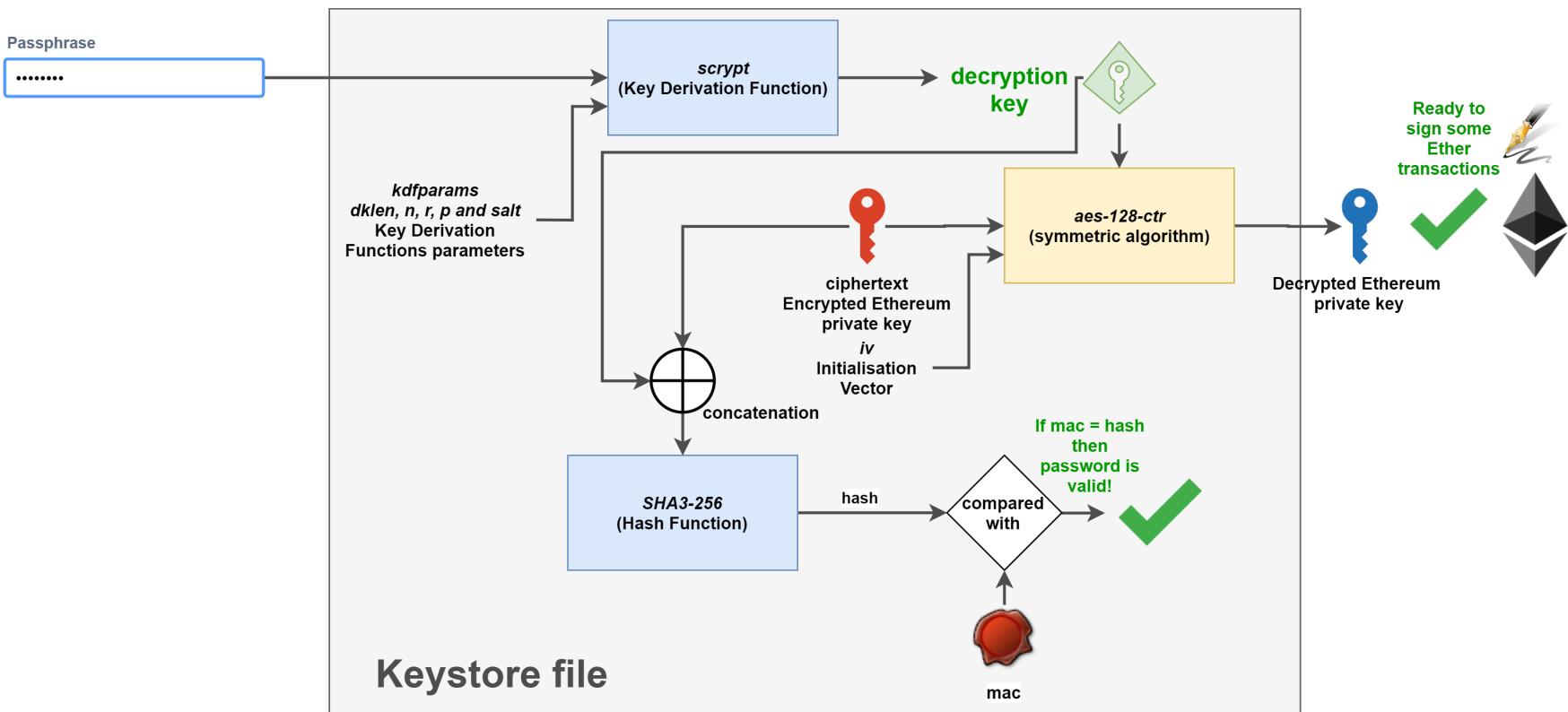


Figura 7.6: Flujo completo del funcionamiento de un keystore ([Julian M](#))