



**UNIVERSIDAD  
DE GRANADA**

TRABAJO FIN DE GRADO  
GRADO EN INGENIERÍA INFORMÁTICA

# **Diseño e implementación de un entorno de gestión forense basado en Blockchain**

---

**Autor**

Rubén Calvo Villazán

**Director**

Gabriel Maciá Fernández



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

Granada, Junio de 2019

## **Diseño e implementación de un entorno de gestión forense basado en Blockchain**

Rubén Calvo Villazán

**Palabras clave:** Blockchain, Ethereum, sistema, descentralizado, Smart Contract, huella, Oracle, gestión, forense, cifrado, cadena, custodia.

### **Resumen**

Actualmente vivimos en la era de la información, en la que, gracias a la revolución digital, es difícil encontrar a una persona que no posea un smartphone (o teléfono inteligente), que no tenga acceso a internet o cuyo trabajo se desarrolle en ausencia de un dispositivo tecnológico. La omnipresencia de la tecnología lleva inevitablemente a un incremento de los delitos informáticos producidos y a la búsqueda y aplicación de cada vez mas complejas y novedosas estrategias para llevarlos a cabo. Para poder solventar este problema surgió la informática forense, encargada de obtener, examinar, preservar y mostrar ante un tribunal judicial las evidencias recogidas sobre los delitos informáticos.

Los peritos informáticos o las Fuerzas de Seguridad del Estado son los encargados de recoger las evidencias del delito, que deberán ser correctamente almacenadas en un lugar seguro hasta el momento del juicio, en el que se entregarán al tribunal para su evaluación.

El periodo de tiempo transcurrido desde que un indicio es recogido, valorado y almacenado hasta que se muestra ante el tribunal responsable del caso es lo que se conoce como cadena de custodia, cuya finalidad es garantizar que lo analizado y expuesto en el juicio fue lo mismo que lo recogido.

Durante este periodo de tiempo debe asegurarse rigurosamente la viabilidad del indicio, pues cualquier alteración o modificación del mismo lo invalidaría como prueba frente al tribunal. Por ello, mientras dura el proceso de cadena de custodia, cualquier acto que implique el acceso, visualización, análisis, traslado, etc. del indicio, debe quedar oportunamente documentado, dejando constancia de cada procedimiento, asegurando con ello la total autenticidad, integridad e inalterabilidad de la prueba.

Así, en la cadena de custodia cada eslabón de la cadena hasta llegar al juicio queda recogido en lo que se denomina hoja de ruta de la prueba, en la que queda recogida cada responsable de la custodia y lo que ha hecho con la prueba, permitiendo la total trazabilidad del camino que ha seguido dicha evidencia. Para asegurar que las medidas utilizadas en la evaluación y análisis de la evidencia no la modifican, deben ser realizados siempre por expertos en la materia, confirmando mediante diversos análisis que siempre se va a acceder al indicio tal como se recogió, y obtener el mismo resultado tras la realización de dichos análisis.

En conclusión, es imprescindible llevar un registro exhaustivo de la hoja de ruta de la evidencia, que asegure que el indicio delictivo recogido en un primer momento es el mismo que se muestra finalmente ante el tribunal. Esto implica realizar copias de seguridad, recuperar ficheros, depositar las evidencias ante notario, realizar precintado de equipos informáticos, etc. y que cada uno de esos pasos quede debidamente documentado. Además, en los dispositivos volátiles (como memorias de almacenamiento o páginas web susceptibles a ser eliminadas) debe realizarse una copia del contenido incriminatorio. Todo esto asegurando siempre que no se cometa alteración, manipulación, contaminación, daños o destrucción de las pruebas. Para más información, ver la norma ISO 27037<sup>1</sup> para la recolección de evidencias.

El objetivo de este trabajo de fin de grado es mostrar cómo la emergente tecnología Blockchain puede simplificar el proceso de cadena de custodia anteriormente descrito, permitiendo almacenar, de manera segura y fiable, cualquier evidencia informática que deseemos. Esto es posible, tal y como se expondrá en este trabajo, puesto que los principios que se deben cumplir en la cadena de custodia coinciden con los principios básicos de la tecnología Blockchain.

Se comenzará describiendo qué es el Blockchain, en qué consiste y qué novedades aporta. Se realizará una introducción a los conceptos necesarios para entender cómo se puede utilizar dicha tecnología en el ámbito de la legalidad, además de las diferentes herramientas que se encuentran para desarrollar soluciones basadas en Blockchain.

Posteriormente, se procederá a analizar el diseño del sistema que permitirá almacenar evidencias de origen forense con la tecnología Blockchain, concluyendo con la creación de un prototipo que demostrará la funcionalidad de dicho diseño, almacenando evidencias de contenido en páginas web.

---

<sup>1</sup>ver: <https://peritoit.com/2012/10/23/isoiec-270372012-nueva-norma-para-la-recopilacion-de-evidencias/>

# Design and implementation of a forensic management environment based on Blockchain

Rubén Calvo Villazán

**Keywords:** Blockchain, Ethereum, decentralized, system, Smart Contract, footprint, Oracle, forensic, evidence, management, encryption, chain, custody.

## Abstract

We currently live in the information age. With the digital revolution, today it is difficult to find a single person who does not have a smartphone, easy access to the internet or whose work is carried out in the absence of a technological device.

The omnipresence of technology inevitably leads to an increase in computer crimes. Computer Forensics is responsible for obtaining, examining, preserving and showing before a court of law evidence collected on a computer crime. Computer experts or the National Security Agencies collect the evidence of the crime, which must be stored in a safe place until the moment of the trial, in which they are handed over to the court for evaluation. The period of time from when the evidence is collected and stored until it is presented to the court is what is known as Chain of Custody. During this time, maximum control must be ensured over the evidence.

While the chain of custody process lasts, it must have a total traceability on the evidence, saving information about where it is, who has it in possession, at what time the evidence has been accessed, who has accessed and for what, etc. In addition, the measures taken to evaluate and analyze the evidence should not modify it, ensuring that before several analyzes, the same result will always be obtained. Being carried out by experts in the field.

An exhaustive record of the roadmap of the evidence must be kept, which ensures that the evidence collected at the first moment is the same that is finally shown to the court. This involves making backup copies, recovering files, depositing evidence before a notary, sealing computer equipment, etc. Also in volatile devices (such as storage memories or web pages susceptible to being deleted) a copy of the incriminating content must be made. Everything ensuring that no alteration, manipulation, contamination, damage or destruction of the tests is committed. For more information, see the ISO 27037 standard for evidence collection.

The objective of this end-of-degree project is to show how the emerging Blockchain technology can simplify the aforementioned process, allowing us to store, in a truthful and reliable way, any evidence we want. This can be done (as will be explained throughout this work) since the principles that

must be met in the chain of custody coincide with the basic principles of Blockchain technology.

This project will begin by describing what the Blockchain is, what it consists of and what new features it brings. An introduction will be made to the necessary concepts in order to understand how this technology can be used, in addition to the different tools that are found to develop solutions based on Blockchain.

Subsequently, an explanation of the design of the system that will allow us to store forensic evidences with Blockchain technology, concluding with the creation of a prototype that will demonstrate its functionality, storing evidence of content on web pages.

---

Yo, **Rubén Calvo Villazán**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXXX autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Rubén Calvo Villazán

Granada a 5 de Junio de 2019.

---

D. **Gabriel Maciá Fernández**, Profesor del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado ***Diseño e implementación de un entorno de gestión forense basado en Blockchain***, ha sido realizado bajo su supervisión por **Rubén Calvo Villazán**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 5 de Junio de 2019.

**El director:**

**Gabriel Maciá Fernández**

# Agradecimientos

A mi familia por tener la paciencia necesaria en los peores momentos, a mis amigos por hacer este trayecto universitario mucho más ameno, a todas las personas que en algún momento me han servido de inspiración y a todos los informáticos dispuestos a ayudarse mutuamente con el fin de crear un mundo mejor.



# Contenido

<b>Lista de figuras</b>	<b>iii</b>
<b>1 Introducción</b>	<b>1</b>
1.1 Motivación y contexto del proyecto . . . . .	1
1.2 Objetivos del proyecto y logros conseguidos . . . . .	2
1.3 Estructura de la memoria . . . . .	4
1.4 Contenidos teóricos para la comprensión del proyecto . . . . .	5
1.4.1 Introducción a Blockchain . . . . .	5
1.4.2 Bloques . . . . .	8
1.4.3 Proof of Work y minado . . . . .	10
1.4.4 Ethereum . . . . .	12
1.4.5 Smart Contract . . . . .	12
1.4.6 Oráculo . . . . .	14
<b>2 Planificación y costes</b>	<b>16</b>
2.0.1 Asignación de tareas . . . . .	17
2.0.2 Costes . . . . .	18
<b>3 Análisis del problema</b>	<b>21</b>
3.1 Especificación de requisitos . . . . .	21
3.2 Análisis . . . . .	22
<b>4 Diseño</b>	<b>27</b>
4.1 Interfaz web . . . . .	29
4.2 Interacción con Metamask . . . . .	31
4.3 Interacción de Metamask con Ganache . . . . .	32
4.4 Desplegar el Smart Contract en la red Blockchain . . . . .	32
4.4.1 Desplegando en Rinkeby . . . . .	32
4.4.2 Desplegando en Ganache . . . . .	34
4.5 Interacción: Cliente - Smart Contract - Oráculo . . . . .	35
4.6 Comunicación entre Servidor Web y Blockchain . . . . .	36
4.7 Viendo los resultados con Ganache . . . . .	37

<b>5</b>	<b>Implementación</b>	<b>40</b>
5.1	Conceptos de JavaScript . . . . .	40
5.1.1	Promise . . . . .	40
5.1.2	Callback . . . . .	41
5.2	Directorio de ficheros . . . . .	41
5.2.1	build/ . . . . .	42
5.2.2	contracts/ . . . . .	43
5.2.3	migrations/ . . . . .	43
5.2.4	server/ . . . . .	43
5.2.5	Client.js . . . . .	43
5.2.6	ethereum.js . . . . .	43
5.2.7	Oracle.js . . . . .	44
5.2.8	index.js . . . . .	44
5.2.9	recompile.sh . . . . .	44
5.2.10	truffle-config.js . . . . .	44
5.3	Librerías necesarias . . . . .	44
5.4	Configuración . . . . .	45
5.4.1	2_deploy_contracts.js . . . . .	46
5.4.2	recompile.sh . . . . .	46
5.4.3	truffle-config.js . . . . .	46
5.5	Smart Contract . . . . .	47
5.5.1	Atributos . . . . .	47
5.5.2	Eventos . . . . .	48
5.5.3	Funciones . . . . .	48
5.6	Red Ethereum . . . . .	49
5.7	Cliente . . . . .	50
5.8	Oráculo . . . . .	51
5.9	Servidor . . . . .	54
5.10	Transacción de Metamask . . . . .	55
<b>6</b>	<b>Evaluación y pruebas</b>	<b>58</b>
6.1	Evaluación . . . . .	58
6.1.1	Costes reales de desarrollo . . . . .	58
6.1.2	Comparativa con otras soluciones . . . . .	58
6.2	Pruebas realizadas . . . . .	60
6.2.1	Probando el Smart Contract . . . . .	60
6.2.2	Probando el prototipo . . . . .	62
<b>7</b>	<b>Conclusiones</b>	<b>68</b>
7.1	Valoración personal . . . . .	69
<b>8</b>	<b>Bibliografía</b>	<b>71</b>
<b>A</b>	<b>Manual de usuario</b>	<b>74</b>

# Lista de figuras

1.1	Ejemplo de transacciones en una Blockchain . . . . .	7
1.2	Red Blockchain simplificada a cuatro nodos . . . . .	8
1.3	Estructura de un bloque . . . . .	9
1.4	Ejemplo del árbol de Merkle . . . . .	10
1.5	Cálculo del hash de un bloque . . . . .	11
1.6	Ejemplo de un Smart Contract . . . . .	14
2.1	Etapas en relación con el tiempo para llevar a cabo este trabajo	16
2.2	Tabla de costes estimados . . . . .	20
3.1	Alternativas seleccionadas para la implementación del prototipo	26
4.1	Esquema general de interacción . . . . .	27
4.2	Diagrama de interacción del prototipo implementado . . . . .	28
4.3	Diseño de la interfaz web, página principal . . . . .	29
4.4	Diseño de la interfaz web, página con el resultado . . . . .	30
4.5	Captura de pantalla de la interfaz web . . . . .	30
4.6	Diagrama de flujo de la interacción del servicio web con Metamask	31
4.7	Diagrama de interacción de Metamask con la red Blockchain Ganache . . . . .	33
4.8	Despliegue del Smart Contract en Rinkeby con Remix. . . . .	33
4.9	Despliegue del Smart Contract en Ganache. . . . .	34
4.10	Interacción entre Cliente, Smart Contract y Oráculo. . . . .	35
4.11	Interacción entre el Servidor Web y la red Blockchain de Ganache. . . . .	36
4.12	Ventana de contratos en la interfaz de Ganache. . . . .	37
4.13	Ventana de las cuentas disponibles en la interfaz de Ganache.	38
4.14	Ventana de transacciones realizadas en la interfaz de Ganache.	38
4.15	Ejemplo de evidencia almacenada . . . . .	38
4.16	Bloques en la Blockchain de Ganache . . . . .	39
5.1	Directorio de ficheros del prototipo implementado. . . . .	42
6.1	Ejecución de la función de prueba del Smart Contract. . . . .	61

---

6.2	Obtener la dirección del Smart Contract. . . . .	62
6.3	Obtener el ABI del Smart Contract. . . . .	62
6.4	Comprobación del algoritmo SHA-256 con el Oráculo . . . . .	63
6.5	Comrpobación del algoritmo SHA-256 desde una aplicación web . . . . .	63
6.6	El usuario no tiene Metamask instalado . . . . .	64
6.7	Conexión de la interfaz web con Metamask. . . . .	64
6.8	Metamask solicita permiso para realizar la transacción. . . . .	65
6.9	Historial de transacciones de Metamask. . . . .	65
6.10	Fondos en las direcciones de Ganache. . . . .	66
6.11	Transacciones realizadas en Ganache. . . . .	66
6.12	Transacciones en Ganache correspondiente al Cliente. . . . .	67
6.13	Transacciones en Ganache correspondiente al Oráculo. . . . .	67
A.1	Abrir el proyecto con Ganache . . . . .	75
A.2	Contratos desplegados en Ganache . . . . .	75
A.3	Aplicación web “Save My Evidence” . . . . .	76
A.4	Ejemplo de evidencia almacenada . . . . .	77

# Listados de código

5.1	Ejemplo de creación de Promise. . . . .	40
5.2	Ejecución de una Promise. . . . .	41
5.3	Crear directorio para el proyecto con Truffle. . . . .	41
5.4	2_deploy_contracts.js . . . . .	46
5.5	recompile.sh . . . . .	46
5.6	truffle-config.js . . . . .	46
5.7	Versión del compilador de Solidity . . . . .	47
5.8	Atributo del Smart Contract . . . . .	47
5.9	Evento para el Oráculo . . . . .	48
5.10	Evento para el Cliente . . . . .	48
5.11	Función a la que llama el Cliente . . . . .	48
5.12	Función a la que llama el Oráculo . . . . .	48
5.13	Indicar proveedor para la conexión . . . . .	49
5.14	Creación del objeto Contrato . . . . .	49
5.15	Función que devuelve la dirección a usar. . . . .	49
5.16	Extraer todas las direcciones disponibles. . . . .	50
5.17	Especificar dirección por defecto . . . . .	50
5.18	Guardar la fecha en formato Unix . . . . .	50
5.19	Ejecutar función del Smart Contract del Cliente . . . . .	50
5.20	El Cliente escucha su evento para suscribirse. . . . .	51
5.21	URL a evidenciar. . . . .	51
5.22	El Oráculo se suscribe al evento. . . . .	51
5.23	El Oráculo se suscribe al evento. . . . .	52
5.24	Funciones asíncronas que ejecuta el Oráculo. . . . .	52
5.25	El Oráculo guarda la solicitud. . . . .	53
5.26	Obtener la URL del formulario web. . . . .	54
5.27	Procedimiento del servidor web. . . . .	54
5.28	Validar la URL . . . . .	55
5.29	Comprobar que Metamask está instalado . . . . .	55
5.30	Definir dirección destino y cantidad de ETH. . . . .	55
5.31	Iniciar Ethereum con Metamask. . . . .	56
5.32	Definir la operación en bajo nivel de la API. . . . .	56
5.33	Definir parámetros de la transacción. . . . .	56
5.34	Juntar parámetros en una petición RPC. . . . .	56

---

5.35	Función que realiza la transacción con Metamask. . . . .	57
6.1	Función de prueba del Smart Contract . . . . .	61
6.2	Compilar el Smart Contract . . . . .	61
6.3	Desplegar el Smart Contract . . . . .	61
6.4	Obtener una instancia del Smart Contract . . . . .	61
6.5	Comprobar el algoritmo SHA-256 . . . . .	62

# Capítulo 1

## Introducción

### 1.1 Motivación y contexto del proyecto

Para entender mejor el objetivo de este proyecto, se plantean varios problemas:

1. Desde el momento en que cualquier individuo inicia una navegación por internet, se crea lo que se conoce como huella digital, es decir, el rastro que deja dicha persona de su actividad en la red, al registrarse en una red social o comentar en un foro, por ejemplo. En ocasiones, de manera intencionada o no, un usuario puede realizar comentarios ofensivos, subir contenido inadecuado a la red o incumplir leyes de protección de datos, copyright, etc. A pesar de la existencia de la huella digital, para la justicia es difícil identificar o evaluar la evidencia de dicho contenido si, por ejemplo, el usuario o la página realiza un borrado, o si el responsable utiliza una identidad anónima.
2. En la actualidad, existen numerosos métodos para borrar nuestro rastro “visible” en la red, aunque resulta bastante más complejo eliminar completamente tanto el contenido como la huella digital, es decir, realizar un borrado completo y eficaz.
3. Por otro lado, si terceras personas (como peritos informáticos, por ejemplo) requiriesen consultar información de cara a una investigación, es necesaria la implicación de entidades privadas, empresas o particulares que den acceso al contenido solicitado para que el procedimiento sea legal. Por ejemplo, de cara a la denuncia de un usuario por ofensas o suplantación de identidad, el perito deberá pedir acceso al propietario de la página web o la empresa responsable de la red social en la que hayan ocurrido los acontecimientos, entrando de este modo en juego más agentes a parte del denunciante y la justicia. La presencia de tantos eslabones supone, por un lado, una importante facilidad de manipulación de la información (por el usuario, el responsable de la web, informáticos ajenos capaces de modificar el contenido publicado...),

y por otro, que los datos recogidos no resulten del todo fiables y puedan ser cuestionados de cara a un juicio.

4. Los medios de comunicación nos bombardean constantemente con diferentes contenidos encontrados en la red que nos afectan directa o indirectamente como ciudadanos. Por ejemplo, noticias acerca de comentarios ofensivos en redes sociales y casos de acoso cibernético, noticias falsas que se hacen virales y engañan a miles de personas (conocidas como “Fake News”), políticos que han borrado o intentado borrar controvertidas publicaciones hechas en el pasado para poder optar a un cargo, numerosas estafas de compra-venta de artículos en diversos ámbitos (moda, vacacional), páginas web que han borrado contenido polémico antes de que pudiese dejar rastro, y un largo etcétera.
5. Con todo lo descrito, podemos hacernos a la idea de que vivimos en un mundo informatizado en el que los delitos informáticos cada vez se realizan con mayor inteligencia y complejidad. Para afrontarlos, los profesionales jurídicos requieren de numerosos conocimientos en el ámbito de la informática que en muchas ocasiones no poseen, enlenteciéndose los procedimientos hasta meses e incluso años. Esta situación plantea la necesidad de un nuevo método o sistema que permita evidenciar las pruebas de un delito de manera que los conocimientos sobre informática de los implicados en el juicio no sean relevantes para la exposición, comprensión o demostración de la veracidad de dichas pruebas ante el tribunal. En otras palabras, es necesario un método que permita, desde el primer momento en que el denunciante presenta cualquier tipo de demanda, el almacenamiento de la evidencia de forma transparente y verídica, sea cual fuere el resultado final del juicio.

A todos los problemas anteriormente descritos, se suma la gran variabilidad de tipos de delitos informáticos, no solamente los comentados hasta ahora, como delitos<sup>1</sup> de suplantación de identidad, ofensas o eliminación de información, sino también otros mas complejos como los fraudes en subastas online, no entrega de mercancías compradas en internet, o fraudes en el e-Commerce<sup>2</sup>.

## 1.2 Objetivos del proyecto y logros conseguidos

A la hora de realizar el análisis forense en cualquier delito, es imprescindible la correcta recogida y presentación de la cadena de custodia. La cadena de

---

<sup>1</sup>ver: Elena Pérez - “Los 10 delitos más frecuentes de internet 2018”  
[<https://www.giztab.com/los-10-delitos-informaticos-mas-frecuentes/>][05/06/2018]

<sup>2</sup>e-Commerce: Comercio electrónico, compra-venta realizada a través de internet



custodia “constituye un sistema formal de garantía que tiene por finalidad dejar constancia de todas las actividades llevadas a cabo por todas y cada una de las personas que se ponen en contacto con las evidencias, sirviendo como garantía de la autenticidad e indemnidad de la prueba. Con ello constituye una garantía de que las evidencias que se analizan y cuyos resultados se contienen en el dictamen pericial son las mismas que se recogieron durante la investigación criminal, de modo que no existan dudas sobre el objeto de la prueba pericial”<sup>3</sup>.

No cabe duda de la importancia de seguir protocolos de constancia, documentando todas las actividades relacionadas con el manejo de evidencias y, en definitiva, con la cadena de custodia. Lo que se pretende con ello es garantizar la trazabilidad de las evidencias, pues de otro modo, podrían ser declaradas inválidas a efectos de fundar una sentencia de condena.

El objetivo de este proyecto es utilizar la tecnología Blockchain para facilitar, agilizar y garantizar la fiabilidad de dicho proceso, usando una base de datos descentralizada que permite el almacenamiento de la evidencia forense de manera inalterable e inmutable, ya que cada dato registrado se guarda con una huella digital única. Este procedimiento permite, por tanto, que la evidencia guardada no pueda ser borrada, pese a la eliminación de la fuente de origen. La posibilidad de ejecución de este proyecto se basa en que los principios de la tecnología Blockchain (como se verá en el desarrollo de este documento) solventan las necesidades requeridas para la resolución de los problemas hasta ahora expuestos en los procedimientos legales, creándose gracias a esta tecnología un sistema de fácil uso para cualquier usuario, que permitiría almacenar evidencias forenses sin la intervención de un tercero experto en la materia (como un perito informático) y garantizando la fiabilidad e inalterabilidad de aquello que se ha decidido almacenar, para posteriormente mostrarlo como evidencia en un procedimiento judicial.

A continuación se expone una lista de los logros conseguidos en este proyecto:

- Se ha utilizado la tecnología Blockchain para diseñar un sistema que permite el almacenamiento de evidencias forenses en una base de datos totalmente fiable y no alterable por terceros.
- Se ha aplicado la tecnología de programación existente sobre Blockchain para implementar dicho sistema y poder aplicarlo al ámbito jurídico.
- Se ha diseñado una forma de interacción en la que el usuario, víctima de

---

<sup>3</sup>Manuel RICHARD GONZÁLEZ, Profesor Titular de Derecho Procesal (UPNA)., “La Cadena de Custodia en el proceso penal español”.

un delito informático, pueda almacenar la evidencia en la Blockchain sin la necesidad de recurrir a un experto informático.

- Se ha creado un prototipo que permite almacenar las evidencias forenses deseadas en delitos informáticos de contenido en páginas web.
- Se ha desarrollado una interfaz gráfica web que permite al usuario el fácil uso de dicho prototipo.

### 1.3 Estructura de la memoria

Continuando en este capítulo se explicarán todos los conceptos necesarios para entender en qué consiste la tecnología Blockchain. No se pretende entrar en detalle pero sí exponer las definiciones principales para un correcto entendimiento del trabajo. En el capítulo siguiente se mostrará la planificación llevada a cabo para el proyecto mediante diagramas, se presentará la asignación de tareas así como un resumen de los costes estimados para realizar el prototipo implementado.

Seguidamente se realizará un análisis del problema, especificando los requisitos necesarios para el correcto funcionamiento del prototipo llevado a cabo, a esto se le sumará un análisis de las diferentes alternativas evaluadas en cuanto al software, realizando una comparativa entre ellas y explicando cuál ha sido la característica decisiva para la selección de cada una.

En el capítulo de diseño se explicará la solución propuesta, entrando en detalle en todas las comunicaciones posibles que involucra la solución ya sean dentro o fuera de la red Blockchain. Se definirán mediante diagramas las comunicaciones del usuario con el servidor web, el servidor con la red Blockchain y las que aparecen dentro de la propia red. Mostrando en cada apartado los agentes implicados y su funcionalidad. Además, se encontrarán diagramas con el diseño de la interfaz web que facilitará al usuario el uso del programa.

Tras el capítulo de diseño, se encontrará el apartado con toda la implementación del prototipo en detalle. No se explicará todo el código pero sí las partes cruciales para entender a bajo nivel el funcionamiento del mismo, así como para facilitar la reproducción de la solución adoptada. En este apartado aparecerá la mayor parte del listado de código del proyecto.

Seguido de la implementación, aparecerá un capítulo con la evaluación de los costes reales de llevar a cabo el prototipo, así como una comparativa con otros prototipos o programas similares existentes actualmente. Concluyendo

este capítulo se hallará una batería de pruebas realizadas a la aplicación, con el fin de mostrar el correcto funcionamiento de la misma.

Este trabajo de fin de grado se terminará con un capítulo de conclusión, que además incluirá una valoración personal acerca de la tecnología utilizada y los conocimientos adquiridos.

## 1.4 Contenidos teóricos para la comprensión del proyecto

### 1.4.1 Introducción a Blockchain

El Blockchain es una base de datos arquitecturalmente descentralizada y por definición distribuida. Para comprender esto, es necesario hacer una serie de aclaraciones y definiciones algo complejas.

En primer lugar, se llama base de datos a un conjunto de información, perteneciente a un mismo contexto, ordenada de modo sistemático para su posterior recuperación, análisis y/o transmisión. Desde el punto de vista informático, una base de datos es aquella que almacena conjuntos de datos interrelacionados entre sí, y su fin es servir a uno o varios usuarios, sin redundancias perjudiciales e innecesarias, es independiente de la aplicación que la utilice y tiene operaciones específicas. Dentro de esta definición, cabe aclarar que existe gran variedad de tipos de bases de datos, cuyas características pueden variar y combinarse para ajustarse a las necesidades requeridas en función de la utilidad deseada para cada caso. Definamos por tanto las principales características de la blockchain: descentralizada y distribuida.

Que una base de datos informática sea descentralizada significa que los datos almacenados no se encuentran en una única computadora y ubicación, en cuyo caso para acceder a la información se debería ingresar a la computadora principal del sistema, conocida como “servidor” (mecanismo de funcionamiento de las bases de datos centralizadas). En el caso de las bases descentralizadas, no existe servidor que mantenga el almacenamiento central de la información, de manera que los datos no se almacenan en un solo lugar, sino en una serie de servidores distintos localizados en computadoras y lugares geográficos diferentes. Son estos múltiples servidores los que proveen de información a los clientes. Por definición, este tipo de bases de datos funciona como un grupo de bases de datos independientes; sin tener necesariamente conexiones lógicas entre ellas y que no están totalmente interconectadas mediante una red de comunicaciones <sup>4</sup>.

Por otro lado, las bases de datos distribuidas funcionan como una única base de datos lógica que está instalada en una serie de ordenadores (conocidos

---

<sup>4</sup>ver: <https://www.artechdigital.net/base-de-datos-blockchain/>

como nodos); ubicados en diferentes lugares geográficos y que no están conectados a una única unidad de procesamiento. Pero en este caso, los ordenadores sí están totalmente conectados entre sí a través de una red de comunicaciones. En este sistema todos los nodos contienen información y todos los clientes del sistema están en condición de igualdad. Es decir; las bases de datos distribuidas pueden realizar procesamientos autónomos <sup>5</sup>.

Por tanto, se llama sistema distribuido a un grupo de computadoras que están conectadas entre sí mediante un protocolo de comunicaciones estándar y que trabajan como una única supercomputadora, ofreciendo un servicio común. Tanto las bases de datos descentralizadas como las distribuidas, almacenan información en una serie de computadoras distintas, pero la diferencia clave es que las distribuidas funcionan como una única base de datos que está instalada en un grupo de computadoras interconectadas totalmente mediante una red de comunicaciones, mientras que las descentralizadas funcionan como un grupo de bases de datos independientes y no están totalmente interconectadas mediante una red de comunicaciones. Así, la Blockchain es una base de datos distribuida en la que la información está almacenada por todos los nodos que soportan esta red, pero también es una base de datos arquitectónicamente descentralizada, ya que no hay un punto central infraestructural.

De las numerosas e interesantes características de blockchain, cabe destacar su inalterabilidad, es decir, es una red permanente e inalterable. Cada nodo en el sistema tiene una copia del registro digital, de manera que para agregar una transacción, cada nodo necesita verificar su validez. Si la mayoría piensa que es válido, entonces se agrega al registro. Esto promueve la transparencia y la hace a prueba de corrupción, ya que sin el consentimiento de la mayoría de los nodos, nadie puede agregar bloques de transacciones al registro.

Otro hecho que respalda las beneficiosas características de la blockchain, es que una vez que los bloques de transacciones se agregan en el registro, nadie puede regresar y cambiarlo. Es un registro en el que se almacenan las diferentes transacciones realizadas de manera transparente, al que cualquier usuario tiene acceso, pero solamente para consultar, sin poder modificar esas transacciones. Se garantiza así la veracidad de la información porque ningún usuario de la red podrá editarlo, eliminarlo o actualizarlo.

Para entender mejor la red Blockchain se puede ejemplificar con su uso más conocido y extendido, el monetario. Las criptomonedas son un tipo de divisa o moneda digital, un medio digital de intercambio que utiliza criptografía para asegurar las transacciones financieras, controlar la creación de unidades adicionales y verificar la transferencia de activos. Existen

---

<sup>5</sup><https://www.artechdigital.net/base-de-datos-blockchain/>

diferentes tipos de criptomonedas <sup>6</sup> que se pueden utilizar como fondo para las transacciones (intercambio entre dos usuarios de la Blockchain), pero se utilizarán como ejemplo los conocidos Bitcoins (BTC), los primeros en aparecer de este tipo de tecnología, en una Blockchain reducida.

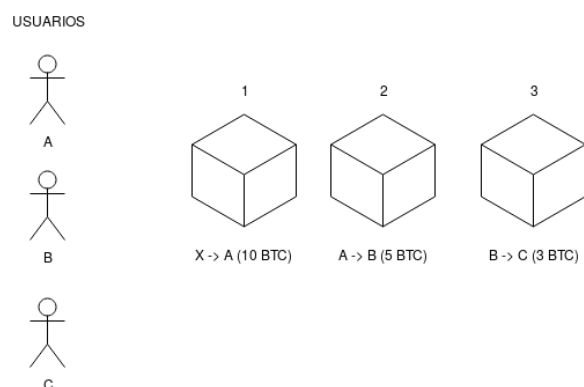


Figura 1.1: Ejemplo de transacciones en una Blockchain

La figura 1.1 muestra una versión simplificada de la Blockchain, en la que podemos observar tres usuarios (A, B y C) y tres bloques (1, 2 y 3). En lo respectivo a los bloques, su funcionamiento y utilidad se explicará en profundidad más adelante, siendo en este momento necesario únicamente comprender que cada bloque alberga el cómputo de un tipo de información determinada almacenada en la Blockchain, por ejemplo las transacciones realizadas, la fecha de creación del bloque, los hash, etc. Entre los supuestos usuarios se realizan las siguientes transacciones: A recibe 10 Bitcoins (BTC) en el bloque 1, A envía 5 BTC a B en el bloque 2 y B envía 3 BTC a C en el bloque 3. Estos bloques se almacenan en la Blockchain y tras realizarse la última transacción, se puede reconstruir el saldo de cada usuario: A se queda con 5 BTC (10 que recibe menos 5 que envía a B), B se queda con 2 BTC (5 que recibe menos 3 que envía a C), y C se queda con los 3 BTC que recibe de B.

Cada transacción realizada es válida ya que cada usuario está enviando una cantidad de BTC inferior o igual a la que dispone, consensuándose en la mayoría de nodos que esa operación puede ejecutarse. Si alguno de los usuarios no dispusiese de la cantidad que quiere enviar, la operación no se realizaría, pues todos los nodos pueden ver qué cantidad de BTC posee cada usuario, y por consenso afirmarían que la operación no puede ejecutarse. Así, para conocer el saldo actual de un usuario, basta con revisar todas las veces

<sup>6</sup>ver: Nuevo Financiero - “Criptomonedas, las 6 monedas digitales del momento y su tecnología Blockchain” [<https://nuevofinanciero.com/criptomonedas-momento-blockchain/>][26/11/2017]

que dicho usuario ha aparecido en el histórico de bloques de la red, que como hemos destacado anteriormente, es una información inalterable una vez ha sido registrada.

El cómputo de historiales de transacciones, almacenados en bloques, es lo que compone la base de datos, y se almacena de forma distribuida, siendo exactamente la misma en cada uno de los nodos<sup>7</sup> que participa en la Blockchain. En el ejemplo de la figura 1.2, se simplifica a cuatro nodos, conectados todos ellos entre sí, mas todos al mismo nivel, sin haber un nodo que predomine sobre los demás, ya que Blockchain es un sistema descentralizado, en el que no existe un servidor o computadora central más relevante que el resto.

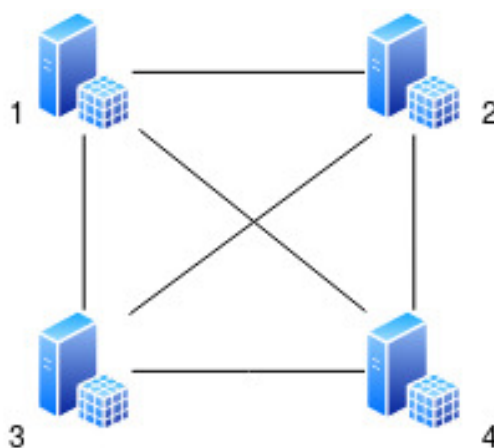


Figura 1.2: Red Blockchain simplificada a cuatro nodos

Al estar todos los nodos al mismo nivel jerárquico, ninguno puede dictaminar ordenes a los demás, cumpliéndose así uno de los principios más importantes de la red Blockchain, la no necesidad de un intermediario para realizar las transacciones, ya que son los nodos y la propia red los que gestionan su inherente seguridad e integridad.

### 1.4.2 Bloques

Cada bloque de los que componen la red Blockchain tiene esta estructura.

- El hash<sup>8</sup> del bloque previo es el resultado de aplicar el algoritmo SHA-256<sup>9</sup> al header o cabecera del bloque anterior. De esta forma se

<sup>7</sup>nodo: ordenador que tiene descargada la Blockchain y participa en ella

<sup>8</sup>Hash: valor resultante de aplicar una información de entrada a una función criptográfica

<sup>9</sup>Un algoritmo es una secuencia de instrucciones, en el algoritmo SHA-256 dada

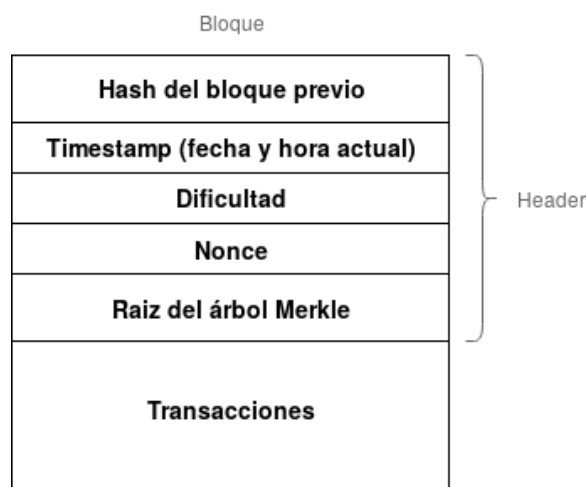


Figura 1.3: Estructura de un bloque

protege la integridad de la cadena, pues la alteración de un solo bit en un bloque anterior, conllevaría la modificación de todos los hash posteriores a dicho bloque hasta llegar al actual.

- El timestamp es la fecha y hora exactas en el momento de creación del bloque.
- La dificultad representa el valor que define cómo de difícil resulta el minado para cerrar ese bloque (este concepto será explicado mas adelante). A Mayo de 2019, toma un valor de 6704632680587.417.
- El nonce es un valor numérico aleatorio de un solo uso. En las Blockchain basadas en Proof of Work, explicadas más adelante, los nodos compiten por “averiguar” (ya que es un número aleatorio y la única forma de descubrirlo es a base de prueba y error) este número. Cuando se crea un nuevo bloque, los diferentes nodos compiten por descubrir qué nonce es el responsable de que al aplicar el algoritmo al bloque (con el nonce añadido), el resultado de dicha operación cumpla la condición predefinida. Por ejemplo, qué valor es necesario para que al aplicar el algoritmo sobre el bloque se cumpla la condición de que el resultado empiece por 15 ceros. El nodo que adivina este valor es el que cierra y escribe el bloque, escribiendo dentro del bloque una transacción en la que recibe Bitcoins a cambio (procedimiento al que se le denomina minado)
- El árbol de Merkle es una estructura de datos en forma de árbol que se

---

una información cualquiera (sin importar su extensión) usada como input o entrada al algoritmo, da lugar a un hash alfanumérico único de 256 bits

usa en Blockchain para simplificar la forma de almacenar los hash de los datos individuales en grandes conjuntos de datos. Es un mecanismo que se utiliza para garantizar que el gran conjunto de datos no se haya modificado y con ello la integridad de la información almacenada, además de que la verificación sea eficiente. Conforme se va ascendiendo por el árbol, se van calculando nuevos hash, combinación de los anteriores, finalizando en un único hash que permite medir la integridad completa del sistema (este hash cambia si cambia el hash de alguno de sus descendientes).

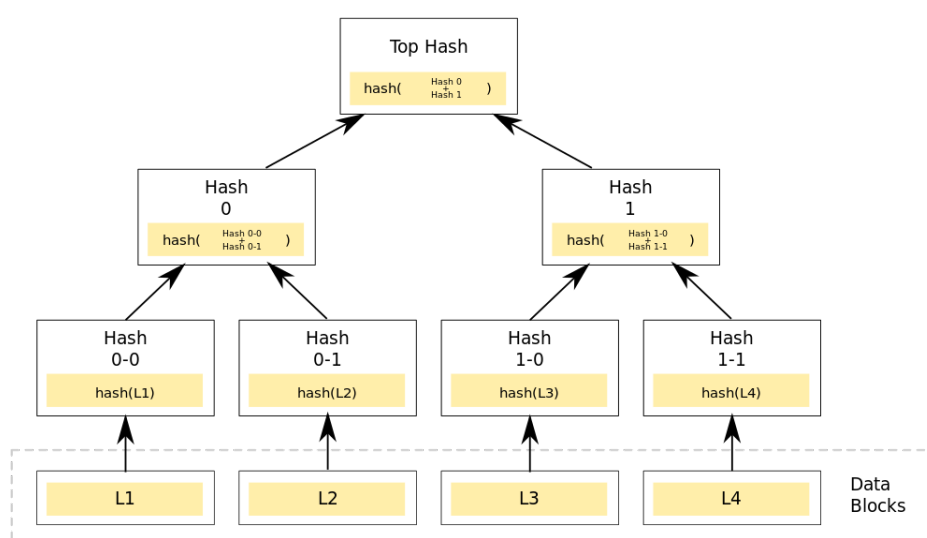


Figura 1.4: Ejemplo del árbol de Merkle

- Las transacciones realizadas. Se pueden almacenar hasta 3.500 en cada bloque en el caso de Bitcoin por ejemplo. Aquí aparecerá la transacción que contiene la recompensa para el minero como premio por adivinar el nonce antes mencionado (que añade BTC a su cuenta), seguida de las transacciones realizadas por los usuarios.

### 1.4.3 Proof of Work y minado

Proof of Work o prueba de trabajo es un tipo de consenso en una red de Blockchain. En la Blockchain, este método se usa para confirmar transacciones y producir nuevos bloques en la cadena.

Cuando se escribe y cierra un bloque, más de la mitad de los nodos existentes en la red deben validarlo, es decir, confirmar que el nonce descifrado



es el correcto. Un nodo resuelve la incógnita del número de ceros y la mayoría de los restantes deben estar de acuerdo en que dicha resolución es la correcta. Para evitar que una persona controle la cantidad suficiente de nodos (más de la mitad de los que conforman la red) y pueda generar las transacciones que desee, se implementa lo conocido como Proof of Work. Los principios de trabajo fundamentales son un complicado acertijo matemático, el nonce, y la posibilidad de probar fácilmente la solución. La prueba de trabajo es, por tanto, un cálculo que deben realizar los restantes nodos antes de poder cerrar y escribir un bloque.

El componente “dificultad” de cada bloque establece un determinado número de ceros por los que debe empezar el hash de la cabecera del bloque. Según la potencia existente en la red, cada 2016 bloques se reajusta dicha cantidad de ceros para que se tarde aproximadamente unos diez minutos en encontrar el valor del nonce.

El minado consiste en todos los nodos realizando cálculos aleatorios para encontrar el valor que, añadido a la cabecera del bloque, hace que su hash empiece por la cantidad de ceros definida en ese momento. El primer nodo que encuentre ese valor (nonce), escribe una transacción en el bloque para sí mismo, recibiendo Bitcoins como recompensa, y envía el bloque “completado” a los demás nodos para que verifiquen el valor del nonce. Si es correcto (cumple la condición), se añade el bloque a la Blockchain. Esa confirmación es la prueba de trabajo, ya que si el nonce es correcto, la operación matemática realizada por todos los demás nodos es sencilla y el resultado es inmediato e idéntico en todos ellos.

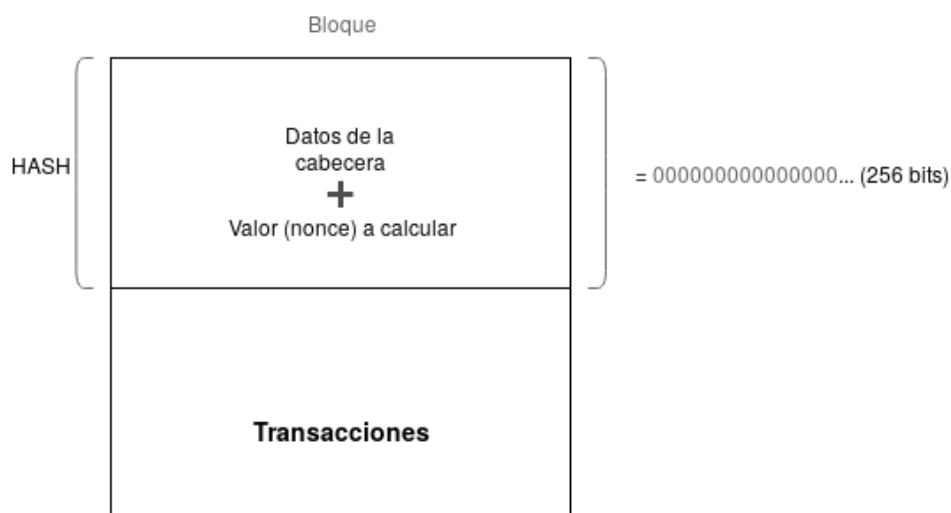


Figura 1.5: Cálculo del hash de un bloque

En la figura 1.5 se puede observar cómo la dificultad establecida en la red dictamina que el hash debe comenzar por 15 ceros. Los nodos deben

competir por calcular el valor del nonce para que añadido a los datos de la cabecera, el hash resultante cumpla dicha condición.

#### 1.4.4 Ethereum

Ethereum es una plataforma de desarrollo basada en la tecnología Blockchain, permite a los desarrolladores crear y publicar aplicaciones descentralizadas (ver el ejemplo de Smart Contract más adelante).

Ethereum es similar a Bitcoin en cuanto a que usa la tecnología Blockchain. Sin embargo, Bitcoin limita su funcionalidad únicamente al ámbito de intercambio monetario mientras que Ethereum está enfocada para poder ejecutar cualquier aplicación descentralizada en su red. En el caso de la plataforma Ethereum, existe una divisa denominada Ether que se puede utilizar para realizar transacciones o para comprar servicios dentro de la propia red. A diferencia de la red Bitcoin, los bloques se escriben en segundos en lugar de minutos y los nodos mineros reciben recompensas por validar los bloques, pero también por ejecutar programas, los contratos inteligentes, ampliado más adelante.

El objetivo de Ethereum es poder ejecutar aplicaciones dentro de la red, para ello se introduce el concepto de “Gas” que no es más que el coste de realizar las operaciones. No confundir el “Gas” con el Ether pues no es una divisa intercambiable. Estos conceptos pueden verse de forma clara con un ejemplo:

- El usuario A quiere enviar una transacción a B sólo si B tiene menos de 10 Ether (ETH) en su cuenta
- Suponiendo que el coste de cómputo de una transacción es de 10 Gas, y que el coste por la comprobación del saldo en la cartera de B es de 5 Gas, el coste computacional estimado es de 15 Gas en total.
- Suponiendo que el Gas tiene un precio de 0.1 ETH la unidad, el nodo que valide este bloque, de forma similar a la validación en la red Bitcoin, dicho nodo minero recibirá  $15 \times 0.1 = 1.5$  ETH de recompensa.

Los números empleados en este ejemplo son meramente ilustrativos, pero de esa forma se explica cómo hace la red Ethereum para protegerse ante la ejecución masiva de tareas, pues cada tarea tiene un coste, y cómo genera un incentivo para que los mineros sigan escribiendo bloques, al ofrecer ETH como recompensa tras ejecutar una tarea.

#### 1.4.5 Smart Contract

Hasta ahora solo se ha hecho hincapié en la función meramente transaccional de Blockchain para facilitar la explicación y comprensión de determinados

conceptos. Sin embargo, en el ejemplo desarrollado en la sección anterior se realizaba una transacción desde el usuario A hacia el usuario B solo si se cumplía la condición de que el saldo de la cuenta de B fuese inferior a 10 ETH. Dicha condición exige a los nodos de la red blockchain que puedan ser capaces no solo de realizar la transacción, sino también de validar los datos almacenados en los bloques, ya que por los propios principios de Blockchain, no podría intervenir un tercero para realizar tal validación. ¿Cómo pueden entonces realizarse este tipo de operaciones no transactivas, como comprobar el saldo de la cuenta?

La ejecución de operaciones diferentes y simultáneas a las transacciones puede realizarse gracias a lo que se denomina Smart Contracts o Contratos Inteligentes, cuya presencia en la red Ethereum aporta la mayúscula ventaja de poder condicionar, comprobar, autorizar, verificar, etc. las transacciones desde la propia red blockchain. Así, los Smart Contracts son programas informáticos que contienen una serie de órdenes y son asociados por el interesado a una transacción determinada. Dichas ordenes pueden ser comprobaciones del saldo, verificación de que las partes implicadas en el contrato son realmente quienes deben ser (comprobando que la dirección es correcta antes de realizar la transacción), computar las existencias de un producto en un almacén antes de solicitar un nuevo envío al distribuidor, etc. El contrato se establece entre los implicados y se sube a la Blockchain, de manera que no puede ser modificado por ninguna de las partes, y solo se realizará la transacción si se cumplen todas las condiciones establecidas y programadas en el contrato.

Las ventajas que esto proporciona son, entre otras:

- No cabe la posibilidad de controversia en la interpretación del contrato, pues es un programa informático el que ejecuta las instrucciones, consensuadas previamente por los interesados.
- Un Smart Contract es capaz de ejecutarse y hacerse cumplir de forma autónoma, automática, sin intermediarios ni mediadores que verifiquen o guarden constancia de lo acordado, como ocurre con los contratos escritos o verbales, en los que es necesario un interventor, notarios, testigos u otros terceros.
- Cada Smart Contract es un programa informático formado por sentencias y comandos que simulan las cláusulas y términos de los contratos tradicionales, de manera que la transacción sujeta al contrato se ejecutará solo en caso de cumplirse las condiciones establecidas. Además, con esto se consigue reducir los costes y tiempo empleados.
- Al estar desarrollados en una red blockchain, son códigos inmutables y visibles, transparencia que les aporta validez sin depender de las

autoridades que lo regulen.

- La transacción no esta sujeta a ninguna ley o jurisdicción territorial al realizarse en una red informática.

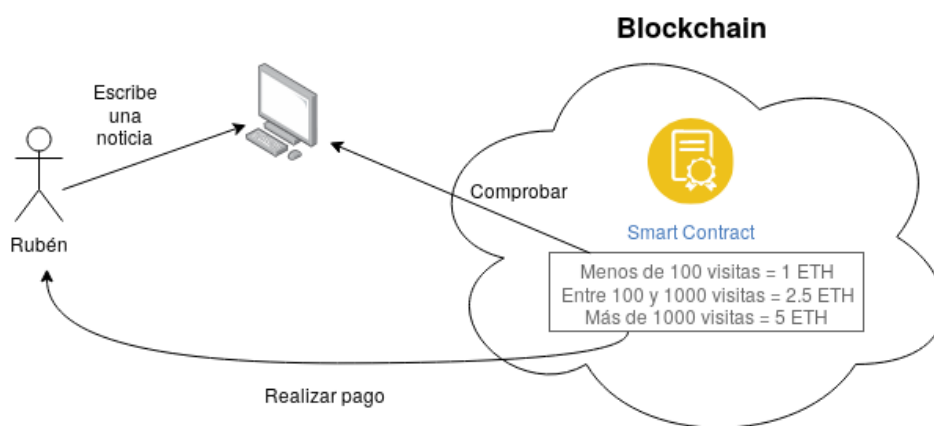


Figura 1.6: Ejemplo de un Smart Contract

En la figura 1.6 se ejemplifica de forma práctica cómo funciona un Smart Contract. La situación representada es la siguiente: El dueño de un periódico establece un contrato inteligente en la red Ethereum, en el que queda recogida como condición que, dependiendo del número de visitas que tengan las noticias escritas por sus redactores, se les realizará un determinado ingreso. El propietario sube este contrato a la red Blockchain de forma que queda inalterable y visible para todos sus empleados y el resto de integrantes. Cada vez que un redactor publica una noticia, el contrato comprobará el número de visitas en el lapso de tiempo establecido por el responsable, y se realizará el ingreso correspondiente en función al número de visitas, no siendo necesarios intermediarios tales como un responsable que compruebe las visitas, hacienda o notarios, realizándose la transacción de manera instantánea. De esta forma la empresa ahorra en costes y tiempo, y los empleados pueden confiar en que no existe sesgo en los cobros, pues no existe posibilidad de manipulación maliciosa de las condiciones establecidas, que son las mismas para todos los redactores.

#### 1.4.6 Oráculo

Ahora bien, en el ejemplo de la figura 1.6 se ha definido un Smart Contract cuyas cláusulas condicionan que se realice un ingreso si se alcanza un determinado

número de visitas. El contrato está almacenado en la Blockchain y por lo tanto, es transparente e inalterable, pero la página web del periódico se encuentra en los servidores de la redacción, externos a blockchain. En esta situación, ¿cómo es posible garantizar la veracidad de los datos del contador? Una página web externa a Blockchain es vulnerable a que un atacante o interesado pueda entrar en el sistema y aumentar manualmente el contador de visitas, engañando al sistema externo y con ello al contrato, para realizarse una transacción superior a la que debería. Por tanto, ¿cómo puede conseguirse que el contador de visitas, siendo un elemento externo a la red Blockchain, sea fiable dentro de la misma? Aquí es donde entran en juego los Oracles u oráculos.

Los contratos inteligentes ejecutan automáticamente una cláusula contractual cuando se cumple la condición preprogramada, y para comprobar si dicha condición se satisface o no, necesitan una conexión que brinde la información del exterior hacia la Blockchain. Los encargados de aportar de forma verídica esta información son los oráculos, plataformas externas a Blockchain que brindan la información fidedigna e inalterable que utilizan los contratos inteligentes. Sin ellos, las diferencias entre lo externo y lo propio de la red Blockchain hacen que sean dos mundos incompatibles por defecto, y solo la presencia de un oráculo puede hacer posible la comunicación entre ambos.

Los Oráculos son entidades externas al Blockchain, que le proporcionan información al mismo. Estas entidades son proyectos <sup>10</sup> o librerías<sup>11</sup> en las que los nodos del Blockchain han llegado al acuerdo de confiar. Ponen en contacto a los proveedores de la información (por ejemplo APIs <sup>12</sup>) con la Blockchain (Ethereum en este caso) de forma que la información externa obtenida sea fiable y pueda ser utilizada en el desarrollo de un Smart Contract.

En el ejemplo anterior, el Smart Contract recibirá la información acerca del número de visitas de parte de un Oráculo, para posteriormente realizar el ingreso programado.

---

<sup>10</sup>El proyecto más conocido es Oraclize, ahora conocido como Provable: <https://provable.xyz/>

<sup>11</sup>Conjuntos de código o colección de programas

<sup>12</sup>API: Conjunto de código (programa) que se puede emplear para que varias aplicaciones se comuniquen entre sí, ver: <https://neoattack.com/neowiki/api/>

## Capítulo 2

# Planificación y costes

Para llevar a cabo este proyecto, se ha dividido el desarrollo en una serie de objetivos o etapas de trabajo que se han ido completando a lo largo del curso académico. Para entender mejor estas etapas de desarrollo, se muestra a continuación un Diagrama de Gantt<sup>1</sup> que permite facilitar la visualización secuencial entre dichos objetivos, así como el comienzo y finalización de los mismos.

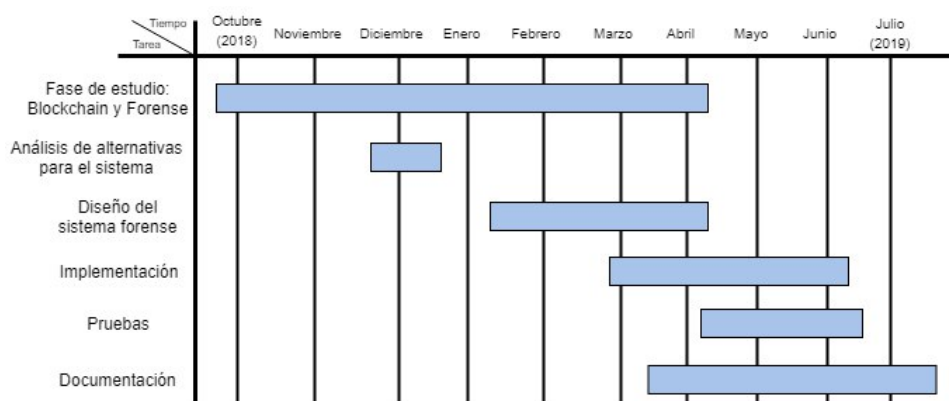


Figura 2.1: Etapas en relación con el tiempo para llevar a cabo este trabajo

Como se puede observar en la figura 2.1, la mayor parte del tiempo la ocupa la fase de estudio en Blockchain y Forense, dada la complejidad de ambos ámbitos, para poder vincular y aplicar los conocimientos adquiridos. Suponiendo un entorno de desarrollo real y en base a la experiencia que ha supuesto el desarrollo de este prototipo, implementarlo supondría la necesidad de invertir en la formación acerca de Blockchain, debido a lo novedoso y complejo de dicha tecnología. Previo a poner en funcionamiento Blockchain en el entorno jurídico, sería imprescindible formar no sólo al

<sup>1</sup>ver: [https://es.wikipedia.org/wiki/Diagrama\\_de\\_Gantt](https://es.wikipedia.org/wiki/Diagrama_de_Gantt)

personal judicial, sino también al policial e investigadores, así como a los ingenieros informáticos encargados de desarrollar eficazmente los programas implicados. Dando un paso más allá, y bajo el supuesto de que en un futuro se utilice esta tecnología para denunciar delitos informáticos, también debería educarse a la población en general, bien desde la escuela o desde centros de formación para adultos, de manera que, sin profundizar en conceptos técnicos, comprendiesen y supiesen utilizar los recursos que estos programas les aportan, salvaguardando así su seguridad e integridad en el peligroso mundo de internet.

Existe por tanto una dependencia directa entre la primera fase de formación y todas las demás, pues es necesario profundizar en los conocimientos mientras se esta llevando a cabo el desarrollo del prototipo. Además, cabe resaltar que, al estar manejando una tecnología en auge y que avanza cada día, es imprescindible mantenerse informado de las novedades desarrolladas por expertos en todo el mundo. Para ello, la lectura de artículos y el estudio de otros prototipos lanzados se convierte en una herramienta de uso imprescindible, pues no solo ayuda a completar y mejorar los proyectos de otros investigadores sino que evita el solapamiento y permite la colaboración entre desarrolladores del mismo campo.

Con respecto al resto de tareas representadas en la figura 2.1, todas ellas quedan reflejadas en apartados posteriores de este trabajo, por lo que no se hará más hincapié en ellas hasta su trato en los correspondientes apartados.

Volviendo al desarrollo de este prototipo funcional, en lo referente a infraestructura, el proyecto se ha desarrollado en un ordenador portátil personal, haciendo uso de programas que se pueden obtener de forma gratuita en internet, así como desarrollando desde cero gran parte del software del prototipo, reflejado, para más información, en el apartado de desarrollo. Por tanto, los costes monetarios que ha supuesto el avance de este proyecto son mínimos, en contraposición a los costes en tiempo de implicación. A pesar de ello, se realizará una estimación de costes en el apartado costes de este mismo capítulo, donde se evaluarán costes de licencias, infraestructura, gastos indirectos, etc.

### 2.0.1 Asignación de tareas

En el desarrollo de este trabajo de fin de grado se han realizado varias versiones borrador del sistema implementado, así como de la documentación. En las distintas reuniones con el tutor se han establecido diversos objetivos a alcanzar hasta finalmente llegar a la finalización del proyecto.

- Comenzando por objetivos a nivel teórico, se ha asignado inicialmente la tarea de recopilar documentación acerca de la tecnología Blockchain

y Forense.

- Se ha propuesto una búsqueda de alternativas a las diferentes tecnologías.
- Se ha propuesto la realización de pruebas con las alternativas escogidas para aproximar el conocimiento al prototipo objetivo a realizar.
- Se ha realizado la implementación del prototipo, siendo sujeto a diferentes pruebas y modificaciones.
- Se ha implementado el prototipo objetivo superando las pruebas previstas.
- Se ha creado una interfaz gráfica para interactuar con el prototipo.
- Se ha integrado todo el sistema superando las pruebas previstas.

### 2.0.2 Costes

Entrando más en detalle en el presupuesto u hoja de requerimientos de este proyecto, aparecerá un ordenador personal por cada trabajador dedicado al desarrollo del prototipo, acceso a internet y acceso a los programas antes mencionados que permitirán desarrollar y evaluar prototipos funcionales en la Blockchain, así como realizar pruebas y entender el correcto funcionamiento del ecosistema o entornos de desarrollo disponibles en la tecnología Blockchain.

Destacar que la mayor parte de licencias de software son gratuitas, aunque algunas son de pago como por ejemplo el programa “Provable”, mencionado anteriormente, que realiza las funciones del oráculo implementado en el software de este proyecto pero solicitando dinero real por los servicios. Es por ello que finalmente no se ha utilizado esta herramienta.

Para tener un punto de vista claro en cuanto al coste real de este proyecto y realizando por ello una estimación para una situación supuesta a media o gran escala, ya que los valores no son exactos, se han reflejado los costes en una tabla donde aparecen costes directos tales como el coste de los equipos, el coste de la infraestructura de desarrollo y el coste del personal, o gastos indirectos como la corriente eléctrica, la tarifa de internet contratada o el coste de alquiler de una oficina en caso de que fuese necesaria.

En caso de querer llevar el prototipo a un entorno de media escala, se debería estar en posesión de varios equipos que monitoricen la red Blockchain, además de una infraestructura de servidor web disponible para atender las peticiones de usuarios. Se podría optar por hosting web para albergar al servidor aunque sería necesario realizar una comparativa entre los beneficios y desventajas de ambas opciones.



Si por el contrario se deseara desplegar a gran escala el prototipo, los gastos de infraestructura serían aún mayores, necesitando mayor mantenimiento y más empleados encargados de la seguridad e integridad del sistema. Se podría optar por hacer legal el uso del programa e integrarlo en el sistema judicial, realizando cursos de formación a la ciudadanía y dejando de forma pública el código del programa. También llegando a colaboraciones con empresas para que integren sus páginas web con el prototipo de forma que resulte más sencillo para los usuarios el almacenar evidencias en determinadas páginas.

Cierto es que en ambos casos escasean los gastos en licencias de software, pues se ha conseguido desarrollar el prototipo haciendo uso de software disponible de forma gratuita en internet, pudiendo utilizar software de pago para incrementar las ventajas o desarrollar a escala más profesional el prototipo, pero pudiendo crear una versión de base de forma totalmente gratuita.

A continuación se muestra la figura 2.2 con la tabla de costes estimados básicos, pudiendo incrementarse o decrementarse en función de la escala deseada a la que llevar el proyecto.

Equipos	600€/unidad
Alquiler oficina	900€/Mes
Conexión internet	60€/Mes
Electricidad	100€/Mes
Servidor web	1.500€/unidad
Mantenimiento	50€/Mes
Seguridad	100€/Mes
Licencia Ganache	0€
Licencia Linux	0€
Licencia Truffle	0€
Hora Trabajo	25€/h
Director del proyecto	50€/h
Gastos extras	200€
Horas estimadas	1.320
<b>Coste estimado total</b>	<b>75.120 €</b>

Figura 2.2: Tabla de costes estimados

## Capítulo 3

# Análisis del problema

### 3.1 Especificación de requisitos

La solución adoptada comprende un sistema completo de gestión para las evidencias almacenadas, empezando por una interfaz web que permite al usuario cómodamente interactuar con la red Blockchain y con ello almacenar la evidencia.

Se parte de la base de la interfaz web. En primer lugar, el usuario escribe en dicha interfaz la URL sobre la que desea almacenar la evidencia. Posteriormente, el programa se comunicará con el servidor para enviar la URL, y el servidor establecerá la conexión con el Cliente, que es el que puede interactuar con el Oráculo a través de la Blockchain. Una vez establecida la comunicación entre el Cliente y el Oráculo, éste último accede a la web y envía la respuesta al cliente, quedando la evidencia almacenada dentro de la Blockchain. Con ello, el usuario obtiene su evidencia almacenada quedando exento de poder modificar la información una vez guardada, comunicándose con el servidor web y siendo éste el que realiza el almacenamiento en la Blockchain.

Mediante suscripciones a eventos (término que se explicará en profundidad en la parte de código), se realizará la comunicación entre el Oráculo y el Cliente. El Cliente registra una petición para almacenar la evidencia en el Smart Contract, que se encuentra lanzado en la red Blockchain. El Oráculo se suscribe a esa petición y obtiene la URL de la información del Smart Contract. Accede a la página con la URL solicitada y crea la evidencia forense guardando los datos según la versión establecida. En la versión primera por ejemplo almacena el título de la página web a evidenciar junto con un hash SHA-256 del código HTML de la misma, de manera que ante un cambio en el contenido de la web dicho hash será distinto (es importante que el usuario esté familiarizado con qué información se almacena en las distintas versiones del prototipo).

El usuario interactúa con el servidor (que ejecuta el Cliente antes mencionado) por medio de una interfaz web, luego es importante que dicha web sea clara y fluida. Además como utiliza tecnología poco conocida como es Blockchain, debe explicarse cómo obtener el resultado de la evidencia almacenada además de los requisitos necesarios para que el usuario haga uso del programa.

Realizar el almacenado de la evidencia supone un coste, es necesario realizar una transacción de ETH (Ether) entre el usuario y el sistema (el Oráculo). Para llevar esto a cabo el usuario debe tener instalada una cartera o monedero con fondos de Ether (la criptomoneda de la red Ethereum), para poder completar la transacción cada vez que lo solicite.

El usuario deberá tener la cartera virtual conectada al navegador, de forma que al acceder a la interfaz web ésta pueda ver y responder a la petición de la transacción hecha por el navegador. El servidor mostrará la interfaz web al usuario para que almacene la evidencia forense deseada, esta petición será procesada posteriormente por el servidor que la comunicará a la Blockchain mediante una interacción establecida entre ambos. Una vez en la Blockchain, se ejecutará el programa principal que consiste en el Cliente, el Smart Contract y el Oráculo, conectados todos a través de la red Blockchain. Estos elementos procesarán la solicitud y generarán una evidencia resultante, que será almacenada en la Blockchain y comunicado de nuevo al servidor para notificar al usuario de que su evidencia ha sido correctamente guardada.

Para establecer las comunicaciones entre los elementos núcleo del programa principal, el Cliente, el Smart Contract y el Oráculo, se hará uso de una API de JavaScript denominada **Web3**<sup>1</sup>. La red Blockchain utilizada será una red de Ethereum que permite realizar pruebas en ella denominada **Ganache**<sup>2</sup>, esta red crea varios usuarios e inicia una Blockchain en el ordenador para poder crear bloques y realizar transacciones. Si por el contrario se desea utilizar la red principal de Ethereum, cabe destacar la importancia de tener memoria suficiente y buena conexión (como se verá en el análisis de alternativas en el apartado siguiente).

## 3.2 Análisis

A continuación se mostrará un listado de las distintas alternativas escogidas para llevar a cabo cada apartado del prototipo implementado, comenzando por la parte más cercana al cliente (la interfaz web), y siguiendo el orden de la comunicación llegando finalmente hasta el Oráculo.

---

<sup>1</sup>Ver: <https://web3js.readthedocs.io/en/1.0/>

<sup>2</sup>Ver: <https://www.trufflesuite.com/ganache>

- **Interfaz web:** La interfaz web se ha desarrollado sobre una página web en código HTML, también ejecuta código JavaScript que realiza las comprobaciones necesarias, además de interactuar con el bridge o puente entre el usuario y la Blockchain, como se especificará en el apartado siguiente.
- **Conexión entre el usuario y la Blockchain:** Es necesario conectar la interfaz web con la red Blockchain si se desea realizar una transacción entre el usuario y el Oráculo por el servicio prestado de almacenamiento de evidencias, para ello se hará uso de una extensión para Google Chrome llamada **Metamask** (consultar bibliografía). Esta extensión permite ejecutar aplicaciones de Ethereum desde el navegador, sin necesidad de tener que descargar la red Blockchain en el ordenador convirtiendo el dispositivo en otro nodo. De esta forma, registrándose e iniciando sesión, pone a disposición un conjunto de carteras o monederos con fondos de Ether para poder realizar pruebas. Permitiendo también la conexión a diversas redes Ethereum tanto la principal (Main Ethereum Network) como las de testeo (Ropsten, Rinkeby), o incluso la red local para realizar pruebas provista por Ganache como se especificará más adelante.
- **Servidor web:** El servidor web se ha desarrollado con JavaScript, haciendo uso del framework web **Express**<sup>3</sup> Este servidor provee de una interfaz web al usuario, donde éste realiza la solicitud de almacenamiento que es recibida posteriormente por el servidor. Con dicha solicitud el servidor interactúa con la red Blockchain para guardar la evidencia necesaria. Como alternativa se podría haber utilizado cualquier otro lenguaje que permita desarrollar un servidor web, sin embargo se ha escogido JavaScript ya que es el mismo lenguaje en el que se encuentra implementado tanto el Cliente como el Oráculo, lo que proporciona facilidades en cuanto a compatibilidad.
- **Red Blockchain:** Como se ha mencionado previamente, para el uso de la red Blockchain se encuentran varias opciones disponibles.
  - En primer lugar, la red principal de Ethereum, que requiere de una cartera real con fondos para poder realizar las pruebas y transacciones necesarias (esta alternativa se ha descartado pues sería necesario invertir dinero real para el desarrollo del prototipo implementado), además sería necesario descargar la red entera en el ordenador personal para poder trabajar en ella.
  - En segundo lugar, las diferentes redes de testeo de Ethereum. Estas redes son por ejemplo Ropsten o Rinkeby. Para el uso de

---

<sup>3</sup>Ver: <https://expressjs.com/>

estas redes Blockchain no es necesario invertir dinero real pues las propias redes disponen de monederos con fondos para realizar las pruebas deseadas. Sin embargo estas redes dan numerosas dificultades a la hora de ejecutar un Oráculo (no así con el Smart Contract, que se ha podido subir a la red Rinkeby sin problemas). Esto se debe a que el Oráculo es un elemento externo a dichas redes y hay que afrontar numerosas dificultades de compatibilidad entre versiones y librerías para llevar a cabo el desarrollo.

- La opción finalmente escogida es la red local **Ganache**. Esta red permite ejecutar las aplicaciones deseadas de la red Ethereum realizando una fácil conexión y además proveyendo de varios monederos con los que realizar transacciones. Además ofrece una sencilla integración con el compilador del Smart Contract para realizar pruebas con el contrato inteligente. Como alternativa a esta red se encuentra TestRPC, que es la versión anterior. Se ha usado finalmente Ganache pues dispone de mejor soporte además de actualizaciones regulares (TestRPC está obsoleto).
- **Cliente y Oráculo:** El Cliente es el programa que se suscribe al evento del Smart Contract (solicita) con el fin de almacenar la evidencia deseada. El lenguaje elegido para desarrollar tanto el Cliente como el Oráculo es JavaScript, ya que proporciona la API **Web3** que contiene todo lo necesario para desarrollar aplicaciones sobre la red Ethereum, como la creación del contrato dentro del propio programa para manipularlo, la asignación de direcciones de la red Blockchain, etc. Además es el mismo lenguaje con el que ha sido implementado el servidor web. Como es necesario ejecutar JavaScript fuera de una página web, se emplea **NodeJS**<sup>4</sup> para construir la aplicación. Se ha optado por NodeJS ya que las comunicaciones entre Cliente y Oráculo no son síncronas, es decir, el Cliente no solicita almacenar una evidencia y el Oráculo responde sino que se realizan de forma asíncrona. En esto destaca NodeJS ya que da la posibilidad de manejar varias conexiones de forma asíncrona al mismo tiempo, además de ejecutar programas escritos en JavaScript en el lado del servidor.
- **Smart Contract:** Para desarrollar el Smart Contract se han evaluado varias alternativas, una de ellas es **Vyper**<sup>5</sup> que permite el desarrollo de contratos inteligentes en Python. La siguiente alternativa evaluada y la que finalmente se ha escogido es **Solidity**<sup>6</sup>. Se ha seleccionado esta alternativa ya que tiene una fácil integración con JavaScript, permitiendo utilizar el contrato como un objeto dentro del propio

---

<sup>4</sup>Ver: <https://nodejs.org/en/>

<sup>5</sup>Ver: <https://vyper.readthedocs.io/en/latest/index.html>

<sup>6</sup>Ver: <https://solidity.readthedocs.io/en/v0.5.3/>

programa.

Para trabajar con Solidity, el framework escogido es **Truffle**<sup>7</sup>. Truffle es una suite de herramientas para desarrollar aplicaciones con Blockchain. Provee de un compilador para el lenguaje Solidity, además de una integración con la red Ganache antes mencionada, permitiendo migrar (hacer deploy en la red) de los contratos implementados. A la hora de desarrollar aplicaciones sobre la red Ethereum, aparecen problemas de conexión entre todos los elementos de la aplicación. Para solventar esto Truffle organiza la estructura en una serie de ficheros similar a la estructura de ficheros de un servidor web, apareciendo también un fichero de configuración para editar las preferencias deseadas. De esta forma todos los elementos de la aplicación se encuentran en algún directorio significativo (esto se explicará en profundidad en el apartado de implementación).

Una alternativa utilizada para compilar y trabajar con el Smart Contract ha sido **Remix**, Remix es un compilador y entorno de desarrollo para el lenguaje Solidity que puede ser utilizado desde el navegador. Esta opción no ha sido escogida finalmente debido a las limitaciones que ello supone, pues no se tiene control sobre los archivos a nivel local, sin embargo Remix se ha utilizado para realizar pruebas y sobretodo en la fase de investigación y aprendizaje del lenguaje Solidity.

Remix ofrece la posibilidad de conectarse tanto a Metamask como a Ganache, según la red seleccionada en el entorno gráfico. También proporciona numerosas versiones de los compiladores de Solidity, así como un entorno de ejecución en tiempo real.

El uso principal que se le ha dado a Remix ha sido el de desplegar el Smart Contract a la red de testeo Rinkeby de Ethereum, para obtener así su dirección y poder utilizar dicho Smart Contract en el caso de querer desplegar el prototipo en ésta red. Aunque la red seleccionada finalmente es Ganache, si se quisiese utilizar el contrato en la red Rinkeby se podría hacer tan sólo utilizando la dirección del propio contrato en esa red. Cómo desplegar un Smart Contract a la red Rinkeby aparece explicado en el siguiente apartado de Diseño.

A continuación se especificará mediante un diagrama de interacción las alternativas seleccionadas en cada fase de la comunicación para llevar a cabo la implementación del prototipo, destacando que NodeJS ocupa la mayor parte del desarrollo.

---

<sup>7</sup>Ver: <https://www.trufflesuite.com/>

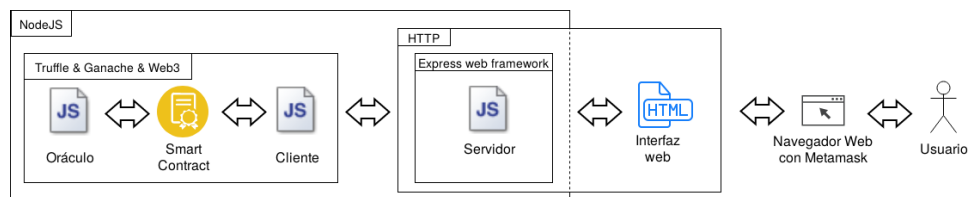


Figura 3.1: Alternativas seleccionadas para la implementación del prototipo



## Capítulo 4

# Diseño

En este capítulo se explicará claramente la solución propuesta, definiendo la arquitectura y desarrollando en profundidad los actores del sistema.

En la figura 4.1 se muestra un esquema general de interacción, de donde se puede obtener una idea general de las partes implicadas y dónde se encuentran localizadas cada una. Tanto el servidor web como la red Blockchain se encuentran en internet, sin embargo no están bajo la misma red pues es el servidor web el encargado de acceder y establecer todas las comunicaciones pertinentes con la Blockchain, donde se encuentran el Cliente, el Smart Contract y el Oráculo.

El servidor web provee de una página al usuario, a la que este accede con el navegador, es imprescindible que el usuario tenga Metamask instalado en su navegador para poder realizar las transacciones necesarias, motivo explicado más adelante. La solicitud de almacenamiento para la evidencia se enviará al servidor web, que será el encargado de establecer las comunicaciones con la red Blockchain para obtener posteriormente la respuesta y notificar al usuario.

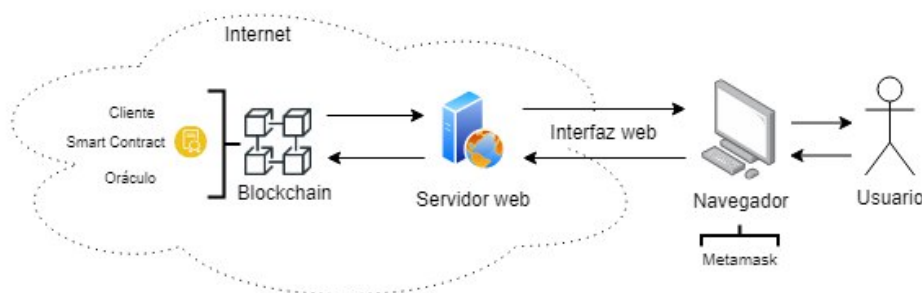


Figura 4.1: Esquema general de interacción

A continuación se procederá a describir un diagrama de interacción general (sin entrar en detalles de código) del prototipo implementado.

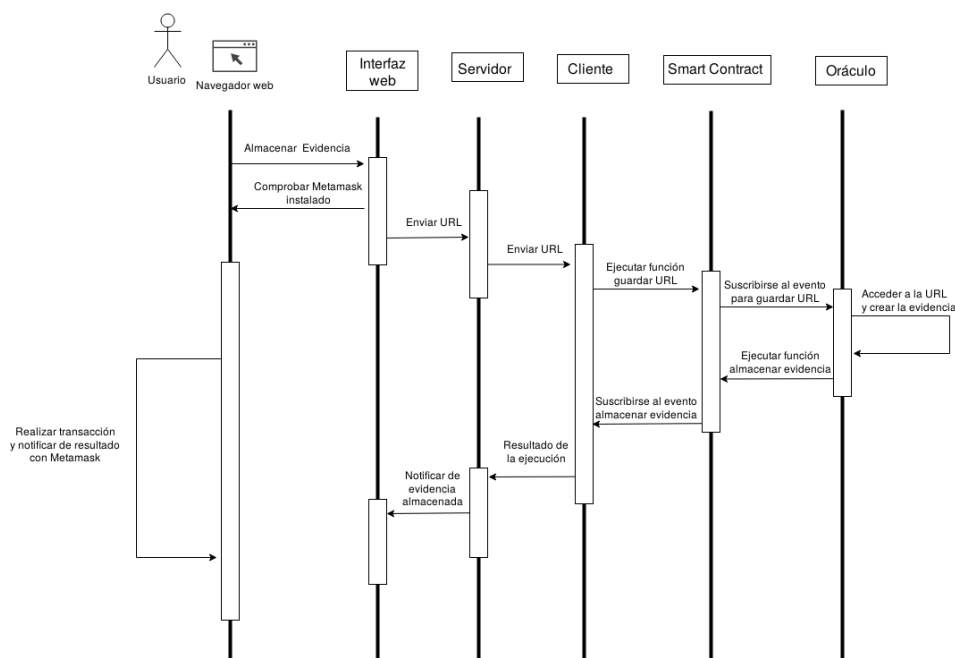


Figura 4.2: Diagrama de interacción del prototipo implementado

**Nota:** Aunque se profundizará en la parte de código, destacar que el término “evento” se refiere a una funcionalidad que permite ejecutar funciones asíncronas de JavaScript en la propia aplicación de la red Ethereum. Dicho de otra forma, es lo que permite comunicarse mediante “llamadas” a los actores participantes en la Blockchain (Cliente, Smart Contract y Oráculo).

Como se puede observar en la figura 4.2, el usuario inicia la interacción con el sistema por medio de un navegador web. El primer paso que se realiza es comprobar si tiene Metamask instalado, ya que es necesario para realizar la transacción correspondiente por el servicio de almacenar la evidencia forense en la Blockchain.

Una vez comprobado, la interfaz web envía la URL introducida por el usuario al servidor. El servidor realiza las comprobaciones pertinentes sobre la URL y la envía al Cliente, que es el encargado de ejecutar la función del Smart Contract para lanzar el evento de guardar evidencia.

Posteriormente, el Oráculo se suscribe a dicho evento recibiendo la URL solicitada. Accede a la web y crea la evidencia a almacenar. Según la versión del prototipo, el Oráculo añade una serie de campos específicos a la evidencia. En la versión primera almacena el título de la página junto con el

hash SHA-256 del código HTML de la misma. También almacena la fecha en formato Unix<sup>1</sup> y la versión actual del prototipo (para conocer los campos almacenados).

Una vez creada la evidencia, el Oráculo ejecuta la función del Smart Contract para almacenarla. El Cliente se suscribe a dicho evento quedando por tanto la evidencia almacenada dentro de la Blockchain. Para más detalles sobre esta interacción dirigirse a la parte de explicación de código.

Una vez haya finalizado la comunicación antes descrita, el servidor comprobará que no han habido errores. Si es así, notificará al usuario de que la evidencia se ha almacenado correctamente.

## 4.1 Interfaz web

Para la interfaz web se ha optado por un diseño sencillo, en el que la funcionalidad principal (que es introducir la URL con el contenido a evidenciar) aparece en el centro. En la parte de abajo de la web aparecen las instrucciones de uso junto con información relevante acerca de la página y forma de contacto.

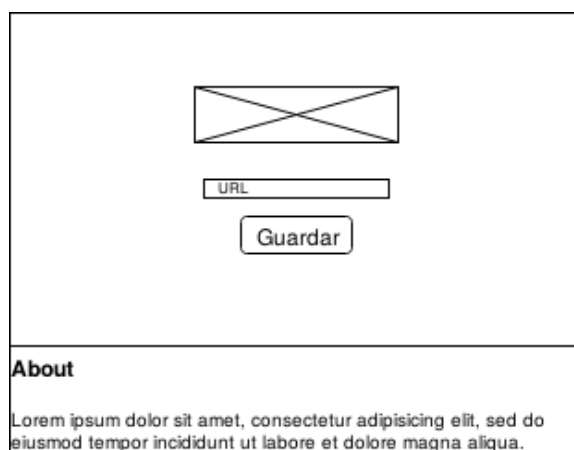


Figura 4.3: Diseño de la interfaz web, página principal

En la página web con el resultado, se muestra en la parte central la notificación de que la evidencia se ha almacenado correctamente. Además aparece un botón para volver a la página principal. El diseño es el siguiente.

La gama de colores escogida para la interfaz web es una variedad de tonos grises, para simular el estilo del icono de la red Ethereum (que aparece de fondo).

Se han utilizado algunos frameworks como **Bootstrap**<sup>2</sup> para añadir de

<sup>1</sup>Para conocer el formato de este timestamp, visitar: <https://www.unixtimestamp.com/>

<sup>2</sup>Ver: <https://getbootstrap.com/>

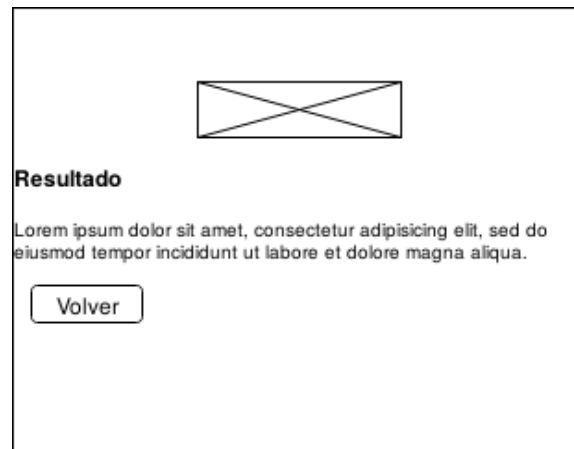


Figura 4.4: Diseño de la interfaz web, página con el resultado

forma sencilla un mejor estilo y diseño, y **JQuery**<sup>3</sup> para hacer más fácil la interacción con los elementos de la página. Para más información sobre estos frameworks dirigirse a la bibliografía.

A continuación se adjuntan una captura de pantalla de la interfaz web:

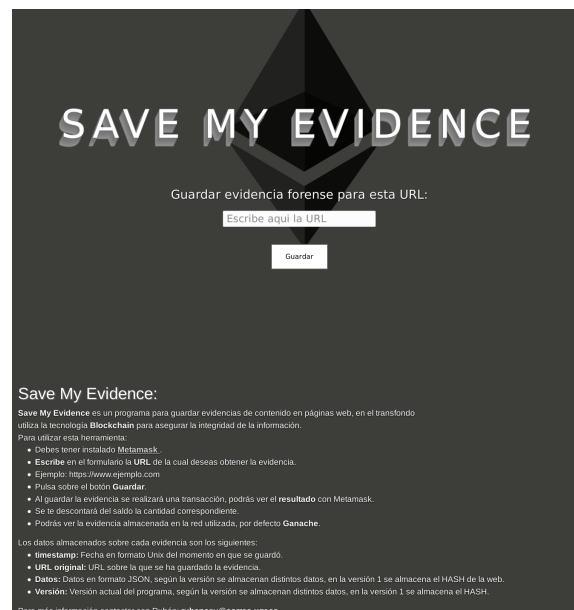


Figura 4.5: Captura de pantalla de la interfaz web

<sup>3</sup><https://jquery.com/>

## 4.2 Interacción con Metamask

A continuación se detallará la interacción que realiza la interfaz web con Metamask mediante un diagrama de flujo.

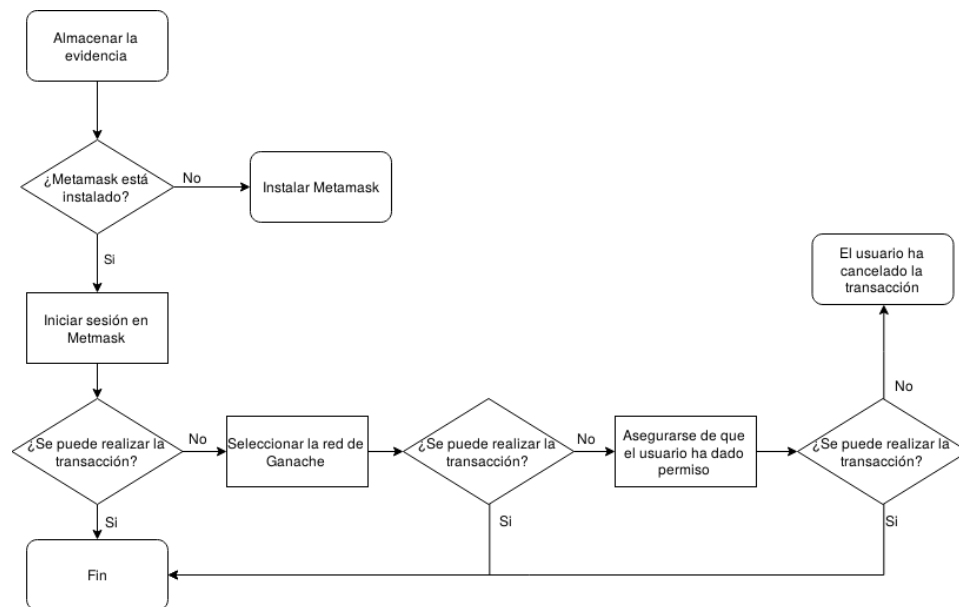


Figura 4.6: Diagrama de flujo de la interacción del servicio web con Metamask

La figura 4.6 muestra todas las comprobaciones y posibles casos a los que se puede enfrentar el usuario haciendo uso de la plataforma de manera conjunta con Metamask. Aparecen los posibles problemas y soluciones a ellos contemplados en la interfaz web del prototipo implementado, de manera que si no se llega a la situación de “Fin”, no se realizará la transacción.

### 4.3 Interacción de Metamask con Ganache

Como se ha explicado previamente, Metamask es el programa que posee la funcionalidad de puente entre el usuario y la Blockchain, permitiendo acceder a aplicaciones de la red Ethereum desde el navegador.

Ganache es la red Blockchain utilizada para desarrollar aplicaciones a nivel local, luego es importante que estos dos elementos estén conectados.

Esta conexión se consigue de la siguiente forma, Metamask ofrece dos posibilidades a la hora de iniciar sesión, crear una cartera nueva o bien importar una ya existente. Se deberá proceder por este segundo apartado. Al iniciar Ganache se crea un nuevo proyecto, para este proyecto creado se proporcionan diez direcciones de la Blockchain con 100.00 ETH de saldo cada uno, además de una frase de seguridad de doce palabras. Pues bien, esta frase será la utilizada para iniciar sesión posteriormente en Metamask. De esta forma, en Metamask, seleccionando la red de Ganache disponemos de las diez cuentas con 100.00 ETH cada una, provistas por Ganache, para realizar las pruebas deseadas.

El usuario dispondrá de las diez cuentas ofrecidas por Ganache para realizar las transacciones deseadas en Metamask, apareciendo una versión con toda la información sobre la transacción realizada disponible para su consulta en el programa de Ganache.

### 4.4 Desplegar el Smart Contract en la red Blockchain

En este apartado se comenzará explicando cómo desplegar el Smart Contract en la red Rinkeby (para el caso de querer realizar pruebas con el contrato en dicha red), y posteriormente desarrollando el despliegue del Smart Contract en la arquitectura elegida, la red Ganache y la suite de herramientas Truffle.

#### 4.4.1 Desplegando en Rinkeby

Como se especificó en el capítulo anterior, para desplegar el Smart Contract en la red Rinkeby es necesario el entorno de desarrollo Remix. Debe estar conectado a la red Rinkeby por medio de Metamask y en Metamask se debe poseer una cuenta con fondos de ETH en la misma red. Esto es así ya que para desplegar el contrato en Rinkeby se realiza una interacción con la red que requiere de una transacción.

Una vez se encuentren tanto Remix como Metamask conectados a la red Rinkeby, tan sólo será necesario abrir el Smart Contract con Remix y desplegarlo.

La figura anterior muestra una captura del proceso.

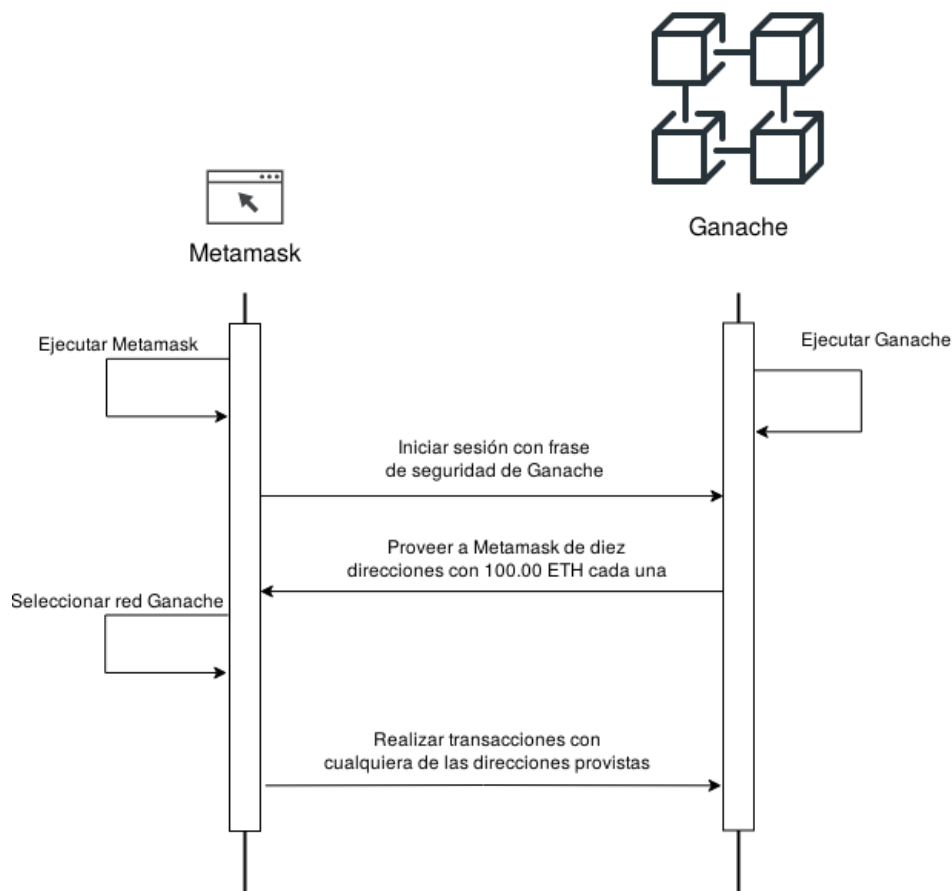


Figura 4.7: Diagrama de interacción de Metamask con la red Blockchain Ganache

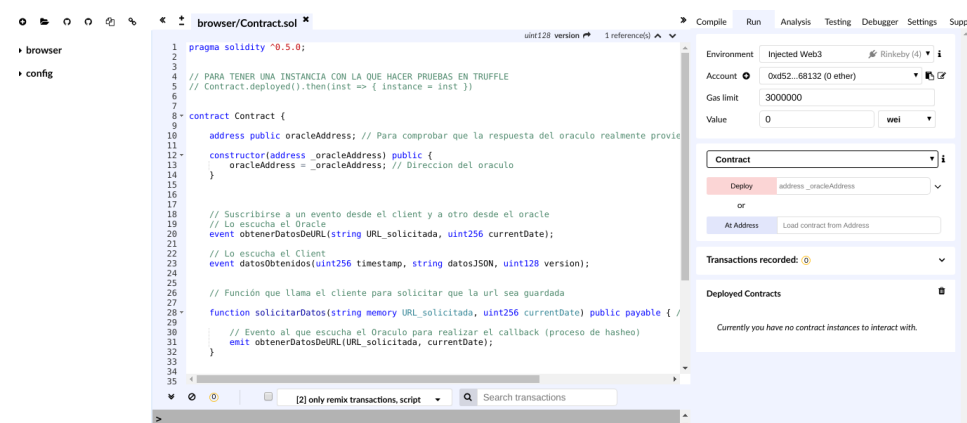


Figura 4.8: Despliegue del Smart Contract en Rinkeby con Remix.

#### 4.4.2 Desplegando en Ganache

Desplegar el Smart Contract en la Blockchain de Ganache se realiza de una forma algo distinta, si antes el contrato se encontraba en Remix, y desde Remix se lanzaba a Rinkeby por medio de Metamask, en este caso no aparecen ninguno de los implicados anteriores.

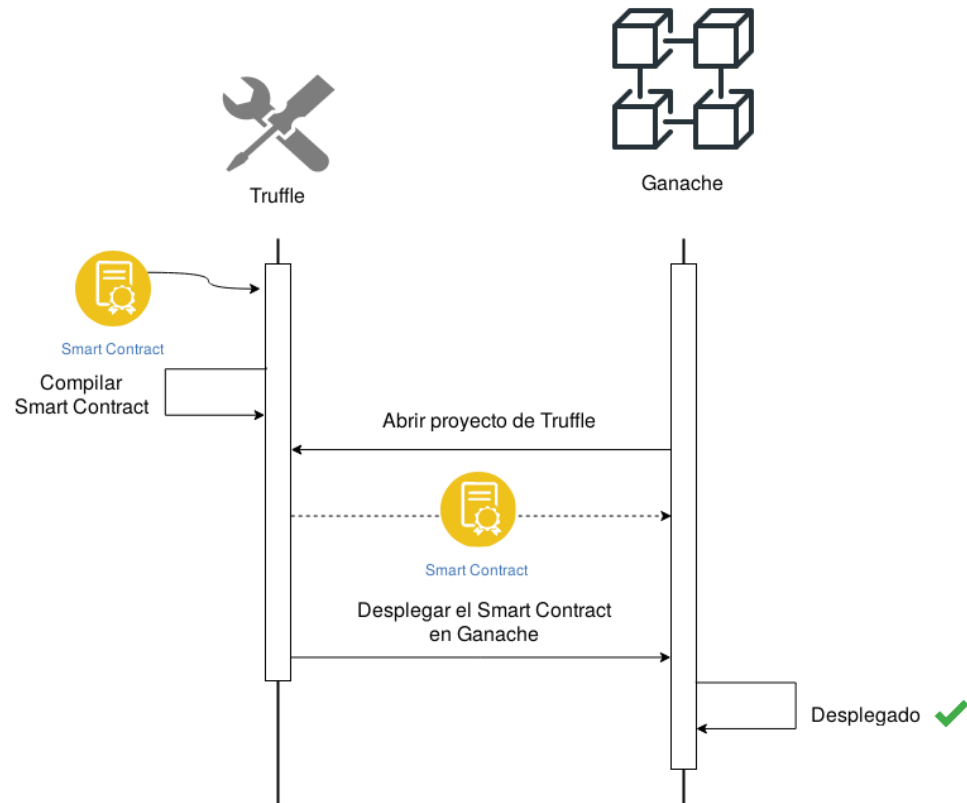


Figura 4.9: Despliegue del Smart Contract en Ganache.

Con el fin de realizar el despliegue en Ganache, se hace necesario un programa que compile el contrato y lo lance a la red Blockchain una vez esto haya ocurrido. Se puede ver de forma más clara en la figura 4.9



## 4.5 Interacción: Cliente - Smart Contract - Oráculo

Estos tres programas se encuentran dentro de la Blockchain (en este caso la red de Ganache). Siguen un proceso de comunicación como se puede ver en la siguiente figura.

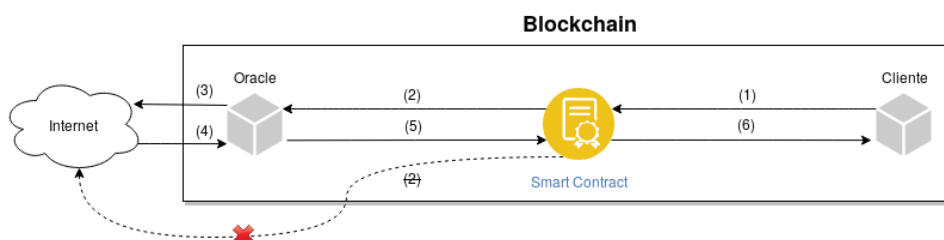


Figura 4.10: Interacción entre Cliente, Smart Contract y Oráculo.

- **(1):** El cliente inicia la comunicación solicitando almacenar la evidencia forense, por ejemplo en este caso de querer evidenciar el contenido de una página web, éste enviaría la URL en el primer paso. El cliente se suscribiría al evento del Smart Contract (ver en el capítulo de implementación) para realizar esta solicitud. Ejecutaría la función de almacenar evidencia que se encuentra implementada en el Smart Contract.
- **(2):** Como el Smart Contract no puede acceder fuera de la red Blockchain, no dispone de internet para acceder a la URL que ha recibido del cliente, luego entra en contacto con el Oráculo que será el encargado de procesar la URL solicitada. Esta comunicación se realizará pues el Oráculo se suscribirá al evento para almacenar la evidencia del Smart Contract.
- **(3):** El Oráculo accede a la página web con el contenido del delito a almacenar como evidencia. Obtiene, por medio de una petición desde NodeJS, el código HTML de la página a evidenciar.
- **(4):** Con la información obtenida, el Oráculo crea la evidencia forense. Según la versión del prototipo implementado, almacenará una serie de datos específicos. Por ejemplo, en la primera versión almacena el título de la página junto con un hash SHA-256 del código HTML de la misma. También guarda el timestamp (fecha y hora de ese momento) y el número de versión para saber interpretar los resultados antes descritos.
- **(5):** El Oráculo ejecuta la función del Smart Contract para guardar la evidencia creada, quedando de esta forma la transacción registrada en la Blockchain con todos los datos previamente evidenciados.

- (6): El Cliente se suscribe al evento del Smart Contract de almacenar evidencia, cerrando por tanto el círculo de comunicación entre el Cliente, el Smart Contract y el Oráculo. Cuando el cliente se suscribe a dicho evento queda guardada la evidencia en la Blockchain y aparece disponible para su futura consulta dependiendo de la red Blockchain utilizada. (En este caso a través del programa Ganache)

**Nota:** Destacar que siguiendo esta comunicación propuesta, resultaría imposible modificar la evidencia, estando totalmente sujeta a la Blockchain. Ganando en tiempo pues no es necesario que una tercera parte verifique la fiabilidad, y en seguridad pues se cuenta con toda la integridad en la información que ofrece la tecnología Blockchain.

Esto solventa los problemas acompañados a la cadena de custodia vistos en el prefacio de este proyecto.

## 4.6 Comunicación entre Servidor Web y Blockchain

Desde la interfaz web, el servidor es el encargado de procesar la URL introducida por el usuario para enviarla a la Blockchain con el fin de almacenar la evidencia. El servidor web no es un agente participativo en la Blockchain sino que facilita el uso del prototipo implementado de cara al usuario final. Proporcionando una interfaz gráfica cuyo diseño se ha descrito previamente.

La comunicación que se realiza entre el servidor web y la red Blockchain se especifica en el siguiente diagrama.

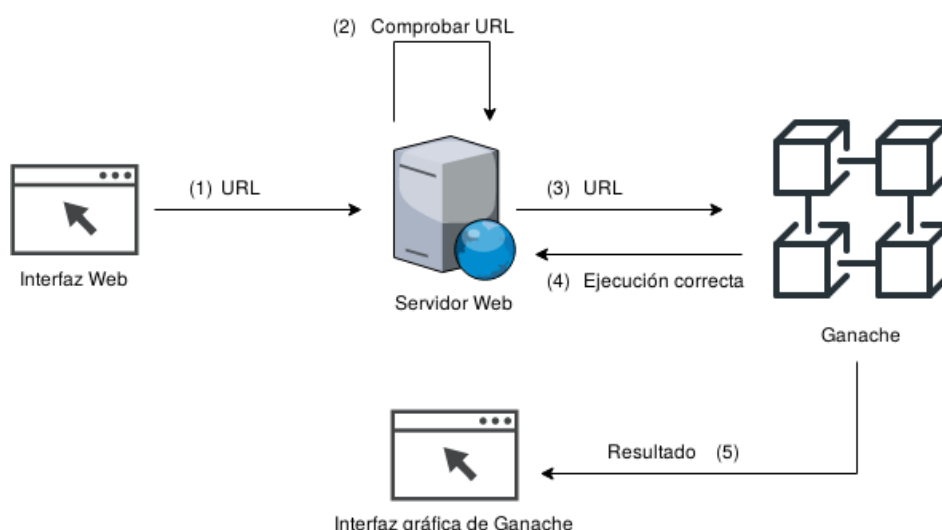


Figura 4.11: Interacción entre el Servidor Web y la red Blockchain de Ganache.

## 4.7 Viendo los resultados con Ganache

Las evidencias almacenadas se guardan en la Blockchain provista por Ganache. En caso de desear comprobar el resultado, se deberá hacer con la interfaz gráfica de Ganache.

Se ha escogido esta plataforma debido a su fácil uso y a su diseño simple de deducir para el usuario.

El primer paso sería realizar un despliegue desde Truffle del Smart Contract en la Blockchain. Una vez hecho el despliegue, se podrá ver la dirección del Smart Contract en la ventana **“Contracts”**. Con esta dirección se podrá proceder a utilizar el contrato dentro de los programas Oráculo y Cliente, para realizar transacciones o llamadas a las distintas funciones de almacenar evidencias. Esto se desarrollará en profundidad en la parte de código.

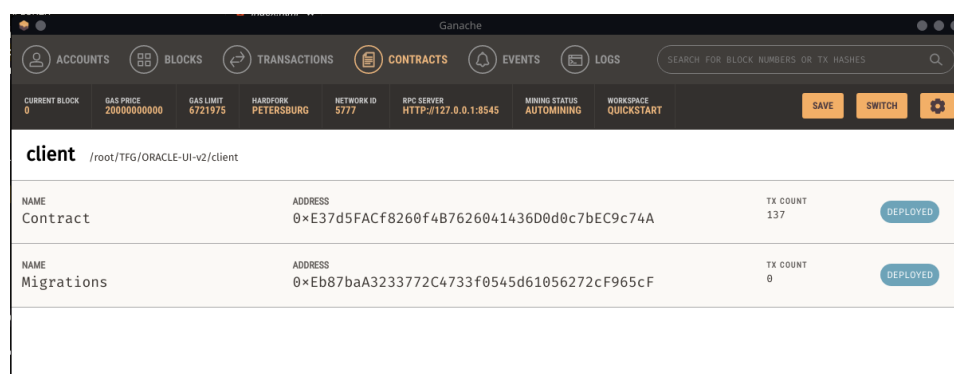


Figura 4.12: Ventana de contratos en la interfaz de Ganache.

En la ventana **“Accounts”** aparecen las diez direcciones que proporciona Ganache para realizar transacciones. Estas direcciones se pueden utilizar tanto para realizar pruebas como para el completo desarrollo de una aplicación basada en esta red.

En esta ventana también aparece la frase de doce palabras que el usuario utilizará para iniciar sesión en Metamask, de forma que éstas diez cuentas con 100.00 ETH cada una también estén disponibles en la extensión de Google Chrome.

Destacar los parámetros de red que aparecen bajo el menú superior. Aquí se especifica que Ganache está en la dirección 127.0.0.1 (localhost) en el puerto 8545. Estos mismos parámetros deberán indicarse en Metamask para poder utilizar las direcciones de Ganache en el navegador.

En la ventana **“Transactions”** se pueden encontrar todas las transacciones realizadas dentro de la red de Ganache. Aparece la dirección fuente, la dirección de destino y el hash de la transacción. Si se ha usado GAS, también aparece especificado, así como la cantidad de ETH en caso de que se haya enviado en la transacción.

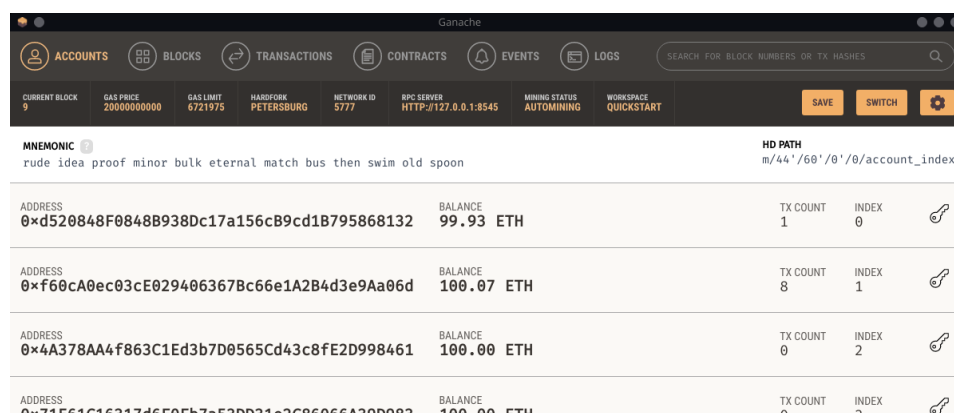


Figura 4.13: Ventana de las cuentas disponibles en la interfaz de Ganache.

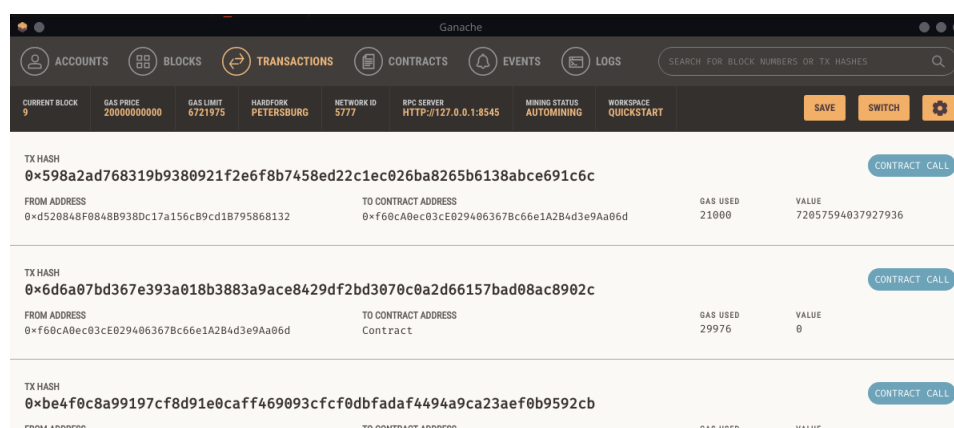


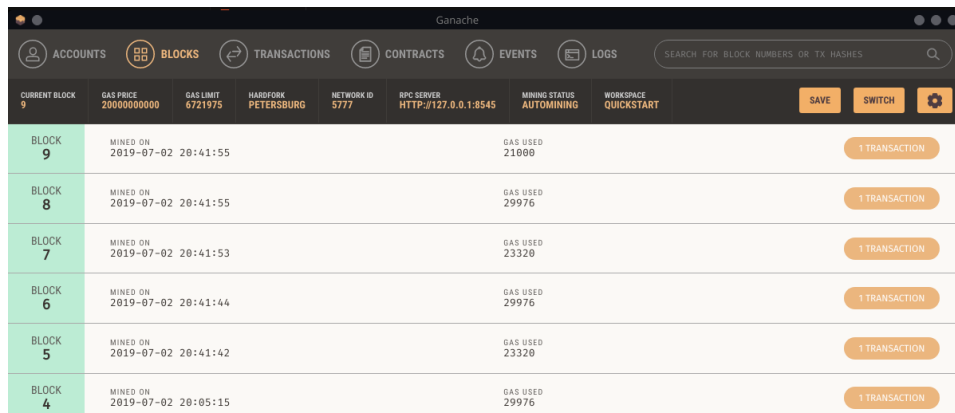
Figura 4.14: Ventana de transacciones realizadas en la interfaz de Ganache.

Viendo los detalles de cada transacción, se puede ver con detalle la evidencia almacenada y todos los datos creados en la operación. Un ejemplo de evidencia almacenada es la siguiente.



Figura 4.15: Ejemplo de evidencia almacenada

Como puede observarse en la figura 4.15, se almacena el timestamp (fecha y hora en formato Unix), un JSON con distintos datos dependiendo de



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE	
9	2000000000	6721975	PETERSBURG	5777	HTTP://127.0.0.1:8545	AUTOMINING	QUICKSTART	<span>SAVE</span> <span>SWITCH</span> <span>⚙️</span>
BLOCK 9	MINED ON 2019-07-02 20:41:55					GAS USED 21000		<span>1 TRANSACTION</span>
BLOCK 8	MINED ON 2019-07-02 20:41:55					GAS USED 29976		<span>1 TRANSACTION</span>
BLOCK 7	MINED ON 2019-07-02 20:41:53					GAS USED 29976		<span>1 TRANSACTION</span>
BLOCK 6	MINED ON 2019-07-02 20:41:44					GAS USED 29976		<span>1 TRANSACTION</span>
BLOCK 5	MINED ON 2019-07-02 20:41:42					GAS USED 23320		<span>1 TRANSACTION</span>
BLOCK 4	MINED ON 2019-07-02 20:05:15					GAS USED 29976		<span>1 TRANSACTION</span>

Figura 4.16: Bloques en la Blockchain de Ganache

la versión, y el número de versión para poder interpretar el JSON. En la primera versión, el JSON contiene el título de la página y el hash del código HTML de la misma.

En la ventana “**Blocks**” se contemplan todos los bloques que se han creado en la red Blockchain de Ganache. Cada vez que se realiza una nueva transacción, se crea un bloque por el proceso descrito en la introducción de este proyecto. En este apartado se pueden ver los bloques, la fecha en la que fueron creados y las transacciones que tienen asociadas.

## Capítulo 5

# Implementación

En este capítulo se expondrá todo lo necesario acerca de la implementación del prototipo, por ello se subdividirá en varias secciones que abarcarán diversos aspectos del sistema desarrollado.

### 5.1 Conceptos de JavaScript

En esta sección no se pretende crear un tutorial a fondo de JavaScript, sin embargo se explicarán algunos conceptos del lenguaje que son necesarios para entender el funcionamiento del prototipo, debido al carácter asíncrono de la solución implementada.

#### 5.1.1 Promise

En JavaScript, una Promise es un objeto que contiene el resultado de una función asíncrona, llama a dos posibles métodos, **resolve** y **reject**. Si la función asíncrona se ejecuta sin error, se llama a resolve, mientras que si hay algún error se llama a reject.

Como es un objeto que contiene el resultado de una función asíncrona, el objeto puede o no tener el valor de dicha función, ya que al ser asíncrona el programa sigue en ejecución sin esperar a que finalice y extraiga el resultado. Es por ello que si se intenta acceder al valor de la Promise y no está disponible, éste se encuentra en estado pendiente.

```
1 var myPromise = new Promise(function(resolve, reject) {  
2   ...  
3   if(error){  
4     reject(error);  
5   }else{  
6     resolve(...);  
7   }  
8  
9 });
```

Listado de código 5.1: Ejemplo de creación de Promise.

Las Promise se ejecutan con **then**, que recibe una función cuyo parámetro corresponde o bien con el `resolve` o bien con el `reject`, en función del resultado de la ejecución de la función asíncrona.

```
1 myPromise.then(function(result) {  
2   console.log(result);  
3 },  
4   function(error) {  
5     console.log(error);  
6   }  
7 );
```

Listado de código 5.2: Ejecución de una Promise.

### 5.1.2 Callback

Los callback son en JavaScript los equivalentes asíncronos a las funciones, se llaman al completar una tarea asíncrona.

Por ejemplo en JavaScript leer un fichero de texto es una función asíncrona, el programa continúa la ejecución mientras se sigue leyendo el fichero. Si se desea mostrar el contenido del fichero, se realiza mediante un callback que imprime dicho contenido por pantalla. En caso de que se desee mostrar un mensaje después de realizar la lectura, éste aparecerá por pantalla antes que el contenido del fichero pues al ser asíncrono, se mostrará el segundo mensaje mientras el fichero sigue siendo leído para, finalmente, mostrar después el mensaje del callback.

## 5.2 Directorio de ficheros

El directorio donde se encuentran los ficheros del prototipo implementado es destacable pues, sigue un orden necesario en el que cada elemento se debe encontrar en su carpeta correspondiente para el correcto funcionamiento del sistema.

Se comienza creando un directorio para el proyecto, dentro de dicho directorio se debe ejecutar la orden

```
1 $ truffle init
```

Listado de código 5.3: Crear directorio para el proyecto con Truffle.

Esto modificará el directorio creado habilitándolo para trabajar con proyectos haciendo uso de truffle, en dicho directorio aparecerán las carpetas **contracts**, **migrations** y **test**, además del archivo **truffle-config.js** para la configuración. El directorio **test** incluye todos los archivos para comprobar el funcionamiento de la aplicación, así como los contratos. No es necesario para el proyecto pues las comprobaciones de la aplicación se han realizado con Ganache y las pruebas del correcto funcionamiento del contrato se han

realizado con el compilador de Truffle desde la terminal, además de con Remix (el compilador de Solidity para navegador mencionado anteriormente).

Se deben crear los siguientes ficheros, quedando la estructura del directorio como se puede ver en la figura 5.1 .

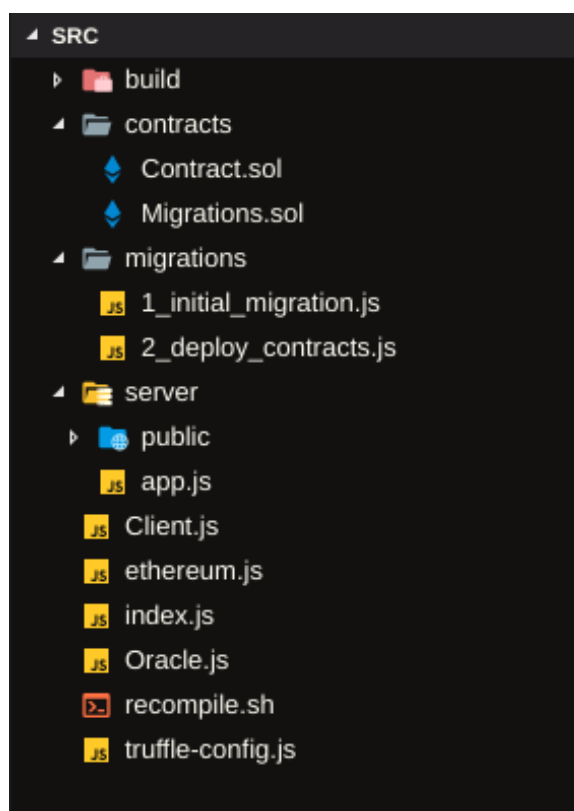


Figura 5.1: Directorio de ficheros del prototipo implementado.

A continuación se procederá a explicar cada uno de ellos.

### 5.2.1 build/

En este directorio se encuentra el subdirectorio **contracts**, en **build/contracts/** aparece un archivo JSON por cada contrato del directorio **contracts/** que haya sido compilado. Estos archivos en JSON contienen información que ayuda a describir la funcionalidad, arquitectura y diseño del software (se denominan artefactos o artifacts). Por ejemplo en **build/contracts/Contract.json** aparece toda la información relativa la contrato implementado. De aquí se extrae la ABI<sup>1</sup>, elemento importante para la creación del contrato en la red

<sup>1</sup>Application Binary Interface, objeto JSON que determina cómo se llaman las funciones del contrato, cómo se extrae la información, cómo se pasa la información de un elemento a otro, etc.



ethereum utilizada, como se verá más adelante. Destacar que cada vez que se compilan los contratos, este directorio se sobrescribe.

### 5.2.2 contracts/

Aquí se encuentran los contratos desarrollados en Solidity. Aparece un contrato predefinido llamado **Migrations.sol** que es requerido por el comando **truffle migrate** cada vez que se desea desplegar un contrato en la red haciendo uso de dicho comando. En este directorio se creará el contrato a utilizar por el prototipo, en este caso denominado **Contract.sol**, que definirá las funciones y eventos con los que trabajarán tanto el oráculo como el cliente.

### 5.2.3 migrations/

En este directorio se encuentran los scripts que despliegan los contratos en la red. Aparecen dos ficheros, **migrations/1\_initial\_migration.js** creado por defecto, que es el encargado de desplegar el contrato **Migrations.sol** (explicado anteriormente y que se encuentra bajo el directorio **contracts/**), y el fichero **migrations/2\_deploy\_contracts.js** que debe ser creado con las instrucciones necesarias en JavaScript para desplegar en la red Blockchain el contrato implementado (**Contract.sol**).

### 5.2.4 server/

Bajo este directorio se encuentra el servidor que muestra la página web al cliente para poder interactuar con el prototipo. Contiene un subdirectorio **public/** donde (junto con la hoja de estilo CSS) se encuentra un archivo **index.html** con la página web a mostrar. Además, bajo **server/** aparece el fichero **app.js** que lanza el servidor web a través de node para proveer al navegador de la página web, así como recibir y procesar la URL solicitada por el usuario para almacenar como evidencia en el formulario de la página.

En la web también se realiza toda la interacción con Metamask, como se verá posteriormente.

### 5.2.5 Client.js

Fichero en el que aparece implementado el cliente, éste inicia la comunicación con el Oráculo mediante la suscripción al evento de almacenar URL, con la URL recibida por el usuario desde la página web.

### 5.2.6 ethereum.js

En este fichero se crea la comunicación entre los distintos elementos que participan, es decir, el Cliente, el Oráculo y el Smart Contract. Aquí se

especifica que la interfaz para las comunicaciones que se va a utilizar es **web3**, se establecen el proveedor de la red (en este caso localhost), se crea el objeto contrato con la dirección y la ABI (antes mencionada), etc.

### 5.2.7 Oracle.js

Aquí se encuentra implementado el Oráculo. Este fichero contiene el acceso a la web externa, la creación de la evidencia, la comunicación con el Smart Contract, etc.

### 5.2.8 index.js

Este fichero ejecuta tanto el Oráculo como el Cliente, recibe como parámetro la URL introducida en la página web por el usuario, de forma que no es necesario ejecutar cada programa por separado, pudiendo hacer que el servidor web ejecute todo lo necesario de una vez.

### 5.2.9 recompile.sh

Este script de bash vuelve a compilar los contratos, reescribiendo el directorio **build/** visto anteriormente.

### 5.2.10 truffle-config.js

Archivo de configuración que une la red Ethereum utilizada (Ganache) con el entorno de Truffle, de forma que se puedan desplegar los contratos del directorio **contracts/** en la red Blockchain de Ganache

## 5.3 Librerías necesarias

Para la instalación del proyecto son necesarias ciertas librerías que se instalan a través de **npm**, npm es el gestor de paquetes de JavaScript para el entorno de **Node.js**. Además de los programas y software necesarios que se han especificado en el capítulo de herramientas utilizadas, los paquetes y librerías requeridas para hacer funcionar el prototipo son las siguientes.

Librerías necesarias para realizar la consulta a la página web desde el Oráculo a partir de la URL.

- `npm install --save request`
- `npm install --save request-promise`

Librería que contiene la operación para realizar un HASH de tipo SHA-256 al código de la página web.

- `npm install -g js-sha256`

Instalar **Ethereum TestRPC** (visto en el capítulo de alternativas) para poder realizar pruebas con el prototipo sin necesidad de una interfaz gráfica como la de **Ganache**.

- `npm install -g ethereumjs-testrpc`

Para realizar las pruebas con **Ganache**, que es el programa seleccionado finalmente.

- `npm install -g ganache-cli`

La conexión se realiza mediante el paquete **web3**, destacar que es necesario instalar una versión inferior a la 1 ya que estas versiones utilizan los proveedores para la red (ver capítulo de implementación), las versiones posteriores a la 1 hacen uso de **sockets** que aún son poco compatibles con los programas existentes.

- `npm install -g web3`
- `npm install -g web3-providers-http`

Finalmente, para la creación de la evidencia es necesario instalar un formateador de JSON.

- `npm install -g json-parser`

En cuanto al servidor web, es necesario instalar el framework sobre el que está implementado, además de un formateador para el cuerpo de la página web.

- `npm install -g express`
- `npm install -g body-parser`

Para instalar **Truffle** y la interfaz gráfica de **Ganache**, se ha seguido el ejemplo que aparece en su respectiva documentación. Consultar la bibliografía para obtener dichos enlaces.

## 5.4 Configuración

Este apartado se centrará en la configuración de los distintos componentes del prototipo implementado, centrándose en los archivos de configuración pero remarcando también cualquier modificación necesaria dentro del propio programa.

### 5.4.1 2\_deploy\_contracts.js

Como se mencionó anteriormente, este script despliega el contrato implementado (**Contract.sol**) en la red Blockchain utilizada. Para realizar esto, se debe crear un objeto contrato a partir del artefacto del contrato implementado, los artefactos son los elementos que ayudan a describir la funcionalidad del programa (descritos previamente en la explicación del directorio **build/**).

Una vez con el contrato, se despliega usando una función a la que se le pasa como parámetro la dirección del Smart Contract.

```
1 var Contract = artifacts.require("Contract");
2
3 module.exports = function(deployer) {
4
5     deployer.deploy(Contract, "0xE37d5FACf8260f4B7626041436D0d0c7bEC9c74A");
6 };
```

Listado de código 5.4: 2\_deploy\_contracts.js

### 5.4.2 recompile.sh

Este es un script en Bash que elimina el directorio **build/** y vuelve a compilar el proyecto. Al compilar de nuevo el proyecto, se crea el directorio **build/** con una versión actualizada. La razón de eliminar el directorio previamente es para evitar fallos de archivos no sobrescritos correctamente o utilizar archivos antiguos que no se han actualizado en la posterior creación del directorio.

```
1 rm -r -d build
2 truffle compile
```

Listado de código 5.5: recompile.sh

### 5.4.3 truffle-config.js

Aquí aparece toda la configuración relativa a la red. Este fichero conecta la suite de Truffle (compilación y despliegue) con la red Blockchain de Ganache. Se realiza mediante un JSON donde se especifica cuál es el directorio para las migraciones (**migrations/**), así como las características de la red a utilizar. En este caso la red es **localhost** en el puerto 8545.

Para utilizar la red de Ganache, se debe especificar su ID (5777), en caso de querer usar otra red, se deberá especificar el ID correspondiente.

También se debe indicar cuál es el Gas de la red Blockchain con un valor numérico (el valor utilizado es el genérico para los ficheros de configuración, revisar la bibliografía)

```
1 module.exports = {
2
3     migrations_directory: "./migrations",
4     networks: {
5         development: {
```

```
6     host: "localhost",
7     port: 8545,
8     network_id: "5777", // ID Ganache, para cualquier red poner "*"
9     gas: 4710000
10  },
11  }
12  };
```

Listado de código 5.6: truffle-config.js

## 5.5 Smart Contract

El lenguaje utilizado para la implementación del Smart Contract es Solidity (ver bibliografía). Es un lenguaje similar a Java y JavaScript, por ello no se va a entrar en detalle en cuanto al lenguaje en esta sección. Sin embargo, se va a proceder a explicar las características que posee dicho lenguaje, que constituyen un pilar fundamental en el funcionamiento del prototipo implementado.

Lo primero a especificar a la hora de implementar un Smart Contract con Solidity es qué versión de compilador se desea utilizar. Para ello, debe aparecer la siguiente línea al principio del fichero (la versión utilizada para el proyecto es la 0.5.0)

```
1 pragma solidity ^0.5.0;
```

Listado de código 5.7: Versión del compilador de Solidity

El Smart Contract se implementa de forma parecida a una clase en programación. En este caso, la clase se denomina “Contract” y tiene sus atributos y métodos.

### 5.5.1 Atributos

Como atributo de esta clase se encuentra “oracleAddress”, que es de tipo **address**. Se trata de una dirección que permitirá posteriormente comprobar que la respuesta del Oráculo realmente proviene del Oráculo y no de cualquier otra dirección. En el Contrato también hay un constructor que inicializa dicho valor.

```
1 address public oracleAddress;
2
3 constructor(address _oracleAddress) public {
4     oracleAddress = _oracleAddress;
5 }
```

Listado de código 5.8: Atributo del Smart Contract

### 5.5.2 Eventos

El Smart Contract dispone de dos eventos. El Cliente se suscribe a uno de ellos mientras que el Oráculo se suscribe al otro.

```
1 event obtenerDatosDeURL(string URL_solicitada, uint256 currentDate);
```

Listado de código 5.9: Evento para el Oráculo

El evento “obtenerDatosDeURL” es el que escucha el Oráculo. Cuando el Cliente realiza un **emit** para dicho evento, el Oráculo entra en ejecución.

```
1 event datosObtenidos(uint256 timestamp, string datosJSON, uint128 version)
    ;
```

Listado de código 5.10: Evento para el Cliente

El evento “datosObtenidos” es el que escucha el Cliente. Cuando el Oráculo realiza un **emit** para dicho evento, el Cliente confirma que la evidencia ha sido guardada.

### 5.5.3 Funciones

Cada función del Smart Contract es llamada por uno de los participantes (Cliente u Oráculo) para realizar el **emit** y las comprobaciones correspondientes. Es la forma de “avisar” al contrario de que realice su tarea.

La siguiente es la función a la que llama el Cliente para solicitar que la URL sea almacenada. Dicha función contiene un **emit** de “obtenerDatosDeURL” que es el evento al que escucha el Oráculo para, una vez suscrito, ejecutar todo el proceso de evidenciar.

```
1 function solicitarDatos(string memory URL_solicitada, uint256 currentDate)
    public payable { // payable -> Se realiza gasto de ETH
2
3     emit obtenerDatosDeURL(URL_solicitada, currentDate);
4 }
```

Listado de código 5.11: Función a la que llama el Cliente

La siguiente es la función a la que llama el Oráculo, se realiza una comprobación al principio para asegurarse de que la dirección del que envía la transacción (**sender**) corresponde con la dirección del Oráculo (aquí se utiliza el atributo de la clase antes descrito).

Posteriormente se realiza el **emit** del evento el cual escucha el Cliente para confirmar que la evidencia ha sido guardada correctamente.

```
1 function guardarSolicitud(uint256 timestamp, string memory datosJSON,
    uint128 version) public {
2     require(msg.sender == oracleAddress,
3         "Solicitud no realizada por el Oraculo"
4
5     emit datosObtenidos(timestamp, datosJSON, version);
6 }
```

Listado de código 5.12: Función a la que llama el Oráculo

Con los eventos y las funciones se consigue una ejecución entrelazada entre el Cliente y el Oráculo. Ambos están a la escucha de su respectivo evento pero sólo se suscriben (ejecutan su función) cuando el contrario realiza el **emit** del evento correspondiente.

## 5.6 Red Ethereum

Para realizar la comunicación entre Cliente y Oráculo es necesario establecer un proveedor de red, esto se realiza en el archivo **ethereum.js**.

El paquete de JavaScript que permite realizar la conexión es **web3**, es necesario especificar cuál es el proveedor, que debe coincidir con el especificado en el archivo de configuración de Truffle. La conexión se realiza en “localhost” en el puerto 8545.

```
1 var Web3 = require('web3');
2 var web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));
```

Listado de código 5.13: Indicar proveedor para la conexión

Posteriormente se procede con la creación del contrato, para ello se guarda el ABI (previamente descrito), junto con la dirección del contrato, creando a partir de ambos el propio objeto.

```
1 const abi = [ { "constant": true, "inputs": [], "name": "oracleAddress",
  "outputs": [ { "name": "", "type": "address" } ], "payable": false, "
  stateMutability": "view", "type": "function" }, { "inputs": [ { "name"
  : "_oracleAddress", "type": "address" } ], "payable": false, "
  stateMutability": "nonpayable", "type": "constructor" }, { "anonymous"
  : false, "inputs": [ { "indexed": false, "name": "URL_solicitada", "
  type": "string" }, { "indexed": false, "name": "currentDate", "type":
  "uint256" } ], "name": "obtenerDatosDeURL", "type": "event" }, { "
  anonymous": false, "inputs": [ { "indexed": false, "name": "timestamp"
  , "type": "uint256" }, { "indexed": false, "name": "datosJSON", "type"
  : "string" }, { "indexed": false, "name": "version", "type": "uint128"
  } ], "name": "datosObtenidos", "type": "event" }, { "constant": false
  , "inputs": [ { "name": "URL_solicitada", "type": "string" }, { "name"
  : "currentDate", "type": "uint256" } ], "name": "solicitarDatos", "
  outputs": [], "payable": true, "stateMutability": "payable", "type": "
  function" }, { "constant": false, "inputs": [ { "name": "timestamp", "
  type": "uint256" }, { "name": "datosJSON", "type": "string" }, { "name"
  : "version", "type": "uint128" } ], "name": "guardarSolicitud", "
  outputs": [], "payable": false, "stateMutability": "nonpayable", "type"
  : "function" } ]
2 const address = '0xE37d5FACf8260f4B7626041436D0d0c7bEC9c74A';
3 const contract = web3.eth.contract(abi).at(address);
```

Listado de código 5.14: Creación del objeto Contrato

Para ejecutar las funciones del Cliente y el Oráculo dentro de la red Ethereum, es necesario diferenciar qué dirección es la que realiza el envío de la transacción. Para ello se utiliza una función que recibe una variable entera con valor 0 para el Cliente y 1 para el Oráculo, y devuelve la dirección correspondiente en el vector de direcciones del cual dispone la red.

Visto desde Ganache, en las 10 direcciones provistas por el programa para realizar pruebas, esta función devolverá la primera dirección para el Cliente y la segunda para el Oráculo. La función que devuelve la dirección correspondiente a usar es la siguiente.

```
1 module.exports.account = function (account_to_use) {  
2  
3   return new Promise((resolve, reject) => {  
4     web3.eth.getAccounts((err, accounts) => {  
5       if (err === null) {  
6         resolve(accounts[account_to_use]);  
7       } else {  
8         reject(err);  
9       }  
10    });  
11  
12  }).catch(function(error) {  
13    console.log(error);  
14  });  
15 };
```

Listado de código 5.15: Función que devuelve la dirección a usar.

Como puede observarse, el vector que contiene todas las direcciones disponibles para utilizar en la red se extrae con la siguiente llamada.

```
1 web3.eth.getAccounts(...)
```

Listado de código 5.16: Extraer todas las direcciones disponibles.

## 5.7 Cliente

El Cliente es el encargado de solicitar el almacenamiento de la evidencia forense, como se ha especificado previamente, éste ocupa la posición 0 en el vector de direcciones de Ganache, es decir, le corresponde la primera dirección.

Al comienzo del Cliente se debe especificar el proveedor de la red al igual que se ha visto anteriormente, con la diferencia de que se debe especificar que la dirección de la cuenta que utilice por defecto sea la suya.

```
1 web3.eth.defaultAccount = web3.eth.accounts[account_to_use];
```

Listado de código 5.17: Especificar dirección por defecto

El Cliente realiza tres funciones principales cuando se ejecuta. La primera de ellas es guardar la fecha de ejecución en formato Unix<sup>2</sup>. Esto se realiza de la siguiente forma.

```
1 let fecha = (new Date()).getTime(); // Fecha en formato normal  
2 let fecha_unix = fecha / 1000; // Fecha en formato unix
```

Listado de código 5.18: Guardar la fecha en formato Unix

---

<sup>2</sup>La Blockchain guarda las fechas en este formato.



Posteriormente, el Cliente ejecuta la función del Smart Contract para guardar la URL. La función que ejecuta el Cliente es “solicitarDatos” y lo realiza mediante el objeto **ethereum** que se obtiene a partir del fichero **ethereum.js** visto anteriormente.

```
1 ethereum.contract.solicitarDatos(urlFormulario, fecha_unix);
```

Listado de código 5.19: Ejecutar función del Smart Contract del Cliente

Esta función recibe como parámetro la URL (que obtiene desde el servidor web) y la fecha antes obtenida.

Finalmente, el Cliente se mantiene escuchando al su evento correspondiente con la evidencia ya almacenada. Esto se realiza con un **return** de una función asíncrona que se mantiene a la escucha hasta que el Oráculo realice el **emit** el cual está escuchando el Cliente.

A partir del objeto **ethereum** y seleccionando la dirección a utilizar, se ejecuta una función que devuelve como resultado un callback de “datosObtenidos”. Esta es la forma con la que el Cliente se mantiene a la escucha de dicho evento, hasta que el Oráculo realice el **emit** correspondiente.

```
1 var datosObtenidos = function (callback) {  
2  
3   return new Promise(function(resolve, reject){  
4  
5     ethereum.account(account_to_use).then(function() {  
6       resolve(  
7         callback(ethereum.contract.datosObtenidos)  
8       );  
9     }).catch(error => reject(error));  
10  
11   }).catch(function(error) {  
12     console.log(error);  
13   });  
14 };
```

Listado de código 5.20: El Cliente escucha su evento para suscribirse.

## 5.8 Oráculo

El Oráculo es el encargado de acceder a la dirección web, extraer la información, crear la evidencia forense y almacenarla. Le corresponde la dirección segunda de la red de Ganache.

Tiene un comienzo igual que el del Cliente, se debe especificar la cuenta que debe usar (en este caso la 1) y el proveedor de red. Posteriormente sigue una ejecución estructurada de una serie de pasos que se explican a continuación.

Primero se guarda la URL a consultar, esto se realiza de la siguiente forma.

```
1 const options = {
2   uri: urlFormulario,
3   getURL : function(){
4     return this.uri;
5   }
6 };
```

Listado de código 5.21: URL a evidenciar.

Tras esto, el Oráculo se suscribe al evento “obtenerDatosDeURL” de forma que si no hay ningún error, se inicia la secuencia para crear y almacenar la evidencia.

```
1 obtenerDatosDeURL(function (res, error) {
2   if (error) {
3     console.log(error);
4   } else {
5     obtenerDatosCallback(options); // Se inicia el servicio
6   }
7 });
```

Listado de código 5.22: El Oráculo se suscribe al evento.

“obtenerDatosDeURL” es una función dentro del propio Oráculo cuyo objetivo es el de suscribirse al evento correspondiente al Oráculo, la forma en que lo realiza es mediante la llamada a un **callback** como se ha especificado previamente. Destacar que tiene el mismo nombre que el evento al que se suscribe para garantizar más claridad en el código. Dicha función es la siguiente.

```
1 var obtenerDatosDeURL = function (callback) {
2
3   return new Promise(function(resolve, reject){
4
5     ethereum.account(account_to_use).then(function(){
6       resolve(
7         callback(ethereum.contract.obtenerDatosDeURL) // Con el callback
          se suscribe al event
8       );
9     }).catch(error => reject(error));
10
11   }).catch(function(error) {
12     console.log(error);
13   });
14 };
```

Listado de código 5.23: El Oráculo se suscribe al evento.

Una vez explicados estos pasos, lo siguiente en ejecutarse es el núcleo del Oráculo, que reside en la función “obtenerDatosCallback” encargada de acceder a la URL y crear la evidencia a partir del contenido de la página web.

Esta función se compone de varias llamadas asíncronas, cuya ejecución se encadena por medio de **then** como se ha explicado en los conceptos necesarios de JavaScript.

```
1 function obtenerDatosCallback(options) {
2
```

```
3   return request(options)
4     .then(parseData)
5     .then(guardarSolicitud) // La funcion del Smart Contract que solo
                             puede ejecutar el Oracle
6
7     .catch(error);
8 };
```

Listado de código 5.24: Funciones asíncronas que ejecuta el Oráculo.

La función **request** accede a la URL establecida en “options”, como resultado envía el código HTML de la página accedida a la siguiente función, “parseData”, que es la encargada de crear el JSON con todos los datos a evidenciar.

Según el número de versión del programa, esta función puede crear determinados datos. Al encontrarse en la versión primera, se almacena el título de la web original junto con un HASH de algoritmo SHA-256 de todo el contenido, de forma que si cambia el contenido cambia el algoritmo. Esto está pensado para que se puedan almacenar distintos atributos de la página, en función de la necesidad de la evidencia, de forma que cambiando el número de versión se puede identificar fácilmente qué atributos son los almacenados en cada momento.

La función “parseData” únicamente crea el objeto JSON con los datos necesarios y lo devuelve mediante un **resolve**.

Finalmente el Oráculo ejecuta la función para guardar la solicitud, esta función (denominada “guardarSolicitud” e igualmente asíncrona) asigna los demás datos necesarios para almacenar, y los guarda junto con el JSON creado en la función previa de forma que se guarda toda la información de la evidencia completa.

Dicha función guarda la fecha en formato Unix de forma similar a como hacía el Cliente, posteriormente guarda los datos del JSON en una cadena de caracteres y asigna una versión al programa, por ejemplo la primera versión.

Seguidamente ejecuta la función “guardarSolicitud” del Smart Contract pasando toda esa información como parámetro, de forma que se realiza la transacción y la evidencia queda almacenada en la Blockchain.

Destacar que tanto la función del Oráculo como la del Smart Contract que guardan la solicitud se llaman igual para garantizar claridad en el código.

Además, recordar que la función del Smart Contract comprobaba que la dirección de quien enviaba la transacción correspondía con la del Oráculo, es en este momento de ejecución donde se realiza dicha comprobación antes de guardar la solicitud finalmente.

La función que guarda la solicitud es la siguiente.

```
1 var guardarSolicitud = function({ datosJSON }) {
2
3   return new Promise(function(resolve, reject) {
4
5     let fecha = (new Date()).getTime(); // Fecha en formato normal
```

```
6 let fecha_unix = fecha / 1000; // Fecha en formato unix
7
8 var datos = JSON.stringify(datosJSON);
9 var version = 1;
10
11 ethereum.contract.guardarSolicitud(fecha_unix, datos, version), (err,
12 res) => { // Solo lo puede ejecutar el Oraculo
13   resolve(res);
14 }
15 }).catch(function(error) {
16   console.log(error);
17 });
18 };
```

Listado de código 5.25: El Oráculo guarda la solicitud.

## 5.9 Servidor

El servidor web que provee de la interfaz al usuario se encuentra implementado en JavaScript haciendo uso del framework **express**. En esta sección no se va a profundizar en la configuración seguida en el servidor pues se trata de una configuración estándar, por el contrario, esta sección se centrará en la interacción relacionada con el usuario y el programa de la Blockchain.

El servidor está configurado para poder acceder mediante un navegador web introduciendo la ruta “localhost” con el puerto 8888. Una vez se ha accedido a dicha dirección, el usuario dispondrá de una página web con un formulario donde introducir la URL de la página que desea evidenciar.

Dicho formulario se dirige a una ruta que es captada por el servidor llamada “/saveURL”, esto se realiza mediante la función **post(...)** disponible en el framework.

Una vez procesada la ruta, se obtiene la URL del formulario .

```
1 const URL = req.body.url;
```

Listado de código 5.26: Obtener la URL del formulario web.

Se comprueba que la URL tenga un formato válido antes de enviarla al Cliente, esto se realiza mediante expresiones regulares para ver si cumple con el formato estándar de una URL, “http://ejemplo.com”.

Si la URL es correcta, se envía a los programas de la Blockchain para que trabajen con ella concluyendo con una redirección del servidor a la página web que notifica el correcto almacenamiento de la evidencia forense.

Todo esto se realiza con ayuda del framework web, visto en pseudocódigo resultaría de la forma siguiente.

```
1 var app = express();
2
3 app.post('/saveURL') {
4   const URL = req.body.url;
```

```
5
6  if(validURL(URL)){
7      index.run(URL); // Ejecuta Cliente y Oraculo con la URL del usuario
8
9      // Mostrar notificacion de resultado en el navegador
10
11  }else{
12      res.redirect('/'); // Redirige a la pagina inicial
13  }
14 }
```

Listado de código 5.27: Procedimiento del servidor web.

Al pasar la comprobación de que la URL es válida, el servidor llama a ejecutar al Cliente y al Oráculo enviando como parámetro la URL que ha introducido el usuario. Esa URL es una URL con formato válido, que puede pertenecer a una página a la que se tenga o no acceso, si por ejemplo se trata de una página inaccesible o el usuario no dispone de internet y al acceder aparece un mensaje de error, la página web con el mensaje de error será la que se almacene en la Blockchain. Se podrá saber si la página almacenada es la correcta pues, junto con los datos guardados, también aparece el título de la página, que marcaría la diferencia entre la web correcta o una página de error en caso de un intento fallido de acceder a dicha web.

Destacar que la función con la expresión regular que comprueba que la URL introducida por el usuario es válida es la siguiente.

```
1 function validURL(str) {
2   var expression = /[-a-zA-Z0-9@:%_\+.~#?&//=]{2,256}\.[a-z]{2,4}\b(\/[-a-
3     zA-Z0-9@:%_\+.~#?&//=]*)?/gi;
4   var regex = new RegExp(expression);
5
6   return str.match(regex);
7 }
```

Listado de código 5.28: Validar la URL

## 5.10 Transacción de Metamask

Durante el procedimiento de almacenar la evidencia forense, se realiza una transacción con origen Cliente y destino el Oráculo que ingresa una determinada cantidad de ETH. Esta transacción se realiza a través de **Metamask** (programa descrito en capítulos anteriores) por medio del navegador. Para implementar dicha transacción, es necesario seguir el procedimiento siguiente.

Primero es necesario comprobar que el usuario dispone de Metamask instalado en el navegador, esto se hace con una comprobación antes de realizar cualquier operación.

```
1 if (typeof web3 === 'undefined') {
```

```
2 // No tiene Metamask instalado
3 }else{
4 // Si tiene Metamask instalado
5 }
```

Listado de código 5.29: Comprobar que Metamask está instalado

En caso de no tener Metamask instalado, se devuelve un mensaje de error. En caso de sí tenerlo, se prosigue iniciando la comunicación con node, para ello es necesario definir la dirección destino de la transferencia y la cantidad total de ETH a enviar, esto se realiza de la siguiente forma.

```
1 const destination = '0xf60cA0ec03cE029406367Bc66e1A2B4d3e9Aa06d';
2 const value = web3.toWei('0.0001', 'ether');
```

Listado de código 5.30: Definir dirección destino y cantidad de ETH.

El siguiente paso es iniciar la red Ethereum, ésto se realiza con la siguiente instrucción.

```
1 ethereum.enable()
```

Listado de código 5.31: Iniciar Ethereum con Metamask.

A la instrucción anterior se le añade un **catch** para comprobar errores en el inicio de sesión, por ejemplo que el usuario le de al botón de cancelar a la hora de vincular el navegador con Metamask, que las credenciales no sean correctas, que no haya permiso por parte de la web para vincularse con Metamask, etc.

En caso de que se realice el inicio de sesión correctamente, se prosigue con un **then** en cuyo interior se ejecuta la función que realiza la transacción, obteniendo la primera dirección del vector “accounts” como dirección origen, y la dirección antes definida como destino.

La función que realiza la transacción comienza definiendo la operación a realizar en bajo nivel de la API, se trata de una transacción para enviar ETH.

```
1 const method = 'eth_sendTransaction'
```

Listado de código 5.32: Definir la operación en bajo nivel de la API.

Después define los parámetros necesarios para realizar la transacción, tales como la dirección origen, la dirección destino y la cantidad a enviar.

```
1 const parameters = [{
2   from: account,
3   to: destination,
4   value: value,
5 }]
```

Listado de código 5.33: Definir parámetros de la transacción.

Posteriormente se unen todos los parámetros definidos anteriormente en un objeto JSON final con el fin de realizar una petición RPC<sup>3</sup>.

---

<sup>3</sup>Remote Procedure Call, llamada a un procedimiento remoto o a una subrutina

```
1 const from = account;
2
3 const payload = {
4   method: method,
5   params: parameters,
6   from: from,
7 }
```

Listado de código 5.34: Juntar parámetros en una petición RPC.

El método que realiza la transacción se ejecuta a partir del objeto **ethereum** que provee Metamask. Al ser un método que requiere de autorización para realizar la transacción, este solicitará dicha autorización al usuario mediante una ventana. Otros métodos como leer de la Blockchain que no requieren de autorización no generarán una ventana de alerta para el usuario.

Visto en pseudocódigo (para simplificar mensajes de error) el método es el siguiente.

```
1 ethereum.sendAsync(payload, function (err, response) {
2
3   if (response.error && response.error.message.includes(rejected)) {
4     alertError('Transaccion cancelada, no se han retirado fondos')
5
6     return false;
7   }
8
9 }).then(function(response) {
10   if (response.result) {
11
12     alertExito();
13
14     return true; // Transaccion realizada
15   }
16 });
```

Listado de código 5.35: Función que realiza la transacción con Metamask.

## Capítulo 6

# Evaluación y pruebas

### 6.1 Evaluación

Como se ha ido explicando a lo largo de esta memoria, tan sólo haciendo uso de código y software adicional se puede crear un sistema completo y funcional encargado de almacenar las evidencias forenses en una Blockchain.

#### 6.1.1 Costes reales de desarrollo

Los gastos monetarios de creación del prototipo son inexistentes pues todo el software se encuentra de forma gratuita y como se ha utilizado una red Blockchain en local, tanto para implementar el prototipo como para crear la interfaz web no se ha realizado ninguna inversión. En caso de querer conocer los gastos estimados de llevar el prototipo a media o gran escala, remitirse al capítulo de planificación y costes donde aparece una tabla con los gastos estimados de infraestructura, mantenimiento, personal, etc.

Destacar que el ingreso monetario principal vendría por parte del usuario a la hora de utilizar la aplicación, pues se le extrae una determinada cantidad de ETH que se envía al Oráculo en el momento de realizar la transacción.

#### 6.1.2 Comparativa con otras soluciones

Al tratarse de una tecnología tan nueva y emergente como es el Blockchain, no existe ninguna otra solución implementada con la que pueda compararse.

Si bien es cierto que hay varios artículos publicados donde se realiza una investigación de cómo podría ser un prototipo, no hay ninguno que haya sido llevado finalmente a cabo.

Actualmente (Agosto de 2019) aparecen dos artículos que investigan cómo almacenar evidencias forenses haciendo uso de Blockchain. El primero de ellos es **“Blockchain Solutions for Forensic Evidence Preservation**



**in IoT Environments”** por Sotirios Brotsis, Nicholas Kolokotronis, Konstantinos Limniotis, Stavros Shiaeles, Dimitris Kavallieros, Emanuele Bellini y Clement Pavue. Este se aleja de la solución implementada en este trabajo pues se centra en almacenar evidencias forenses en un entorno IoT<sup>1</sup>, en el que aparecen nuevos agentes participativos tales como objetos cotidianos de casa que disponen de acceso a internet.

Por el contrario, el artículo que más se aproxima a la solución implementada es **“B-CoC: A Blockchain-based Chain of Custody for Evidences Management in Digital Forensics”** por Silvia Bonomi, Marco Casini y Claudio Ciccotelli. Destacar que aunque en el artículo aparece 26 de Julio de 2018 como fecha de publicación, en la web ha sido publicado con fecha 28 de Junio de 2019 luego no ha sido tomado como referencia para llevar a cabo este proyecto.

Uno de los artículos de investigación que sí ha sido utilizado como fuente de información para llevar a cabo el proyecto es **“Block4Forensic: An Integrated Lightweight Blockchain Framework for Forensics Applications of Connected Vehicles”** por Mumin Cebe, Enes Erdin, Kemal Akkaya, Hidayet Aksu y Selcuk Uluagac Department of Electrical and Computer Engineering, este artículo propone un modelo similar de Blockchain en el que almacenar pruebas forenses de delitos o accidentes ocurridos con los denominados vehículos sin conductor. En este modelo, el entorno está rodeado de sensores que guardan los datos relativos a la circulación mediante transacciones en la Blockchain.

Aparecerá la fuente de cada uno de estos artículos para acceder a todos ellos fácilmente en la bibliografía.

A continuación va a procederse a realizar una comparativa con la publicación que más se aproxima a la solución implementada, a pesar de que ésta haya sido consultada con posterioridad a cuando se ha realizado el prototipo.

La principal diferencia encontrada en dicha propuesta es la existencia de una base de datos para evidencias que se encuentra de forma separada a la Blockchain almacenando las evidencias en texto plano. Para ello guarda la evidencia junto con un número identificador obtenido a partir de un hash realizado a la evidencia en adición con el **nonce** (explicado en los conceptos teóricos de la introducción) lo que garantiza la exclusividad del ID para que no aparezca repetido.

En el prototipo implementado en este proyecto, no existe una base de datos externa a la Blockchain pues las consultas se realizan al Oráculo y

---

<sup>1</sup>Internet of Things

se almacenan en la propia Blockchain en el momento en que se realiza la transacción, luego en aspectos de seguridad gana el prototipo implementado pues en la alternativa propuesta en el artículo de investigación, sería necesario el trabajo de terceras partes para controlar el acceso a esa base de datos.

En el artículo de investigación se propone diferenciar entre dos tipos de nodo, un nodo validador que almacena la copia de la Blockchain, genera y valida transacciones, y nodos “ligeros” (lightweight) cuya función es la de ser clientes que dependen de los validadores para realizar sus funciones. Esto genera dificultad a la hora de llevar a cabo el prototipo pues es necesario realizar configuraciones en la Blockchain, ya que una de las premisas que tiene dicha tecnología es la igualdad de preferencia entre todos los nodos. Sin embargo, en el lado opuesto se encuentra el prototipo implementado en este proyecto, que no diferencia entre nodos ya que todos contienen la misma base de datos Blockchain con igual información y bloques, pudiendo de la misma forma consultar al Oráculo estando todos al mismo nivel.

En los demás aspectos la propuesta es similar, ya que sugiere la creación de una interfaz para facilitar la interacción con el usuario además de la creación de un Smart Contract, tal y como se ha realizado en este proyecto.

La mayor diferencia en cuanto a implementación recae en el uso de **Geth**<sup>2</sup> para llevar a cabo el prototipo. Geth es una interfaz de línea de comandos que ejecuta un nodo Ethereum completo implementado en Go, no se ha usado esta interfaz debido al gran tamaño que ocupa y la lentitud que conlleva tener un nodo Ethereum dispuesto en la computadora, en su lugar se ha usado **Truffle** y **Ganache** como se ha indicado en el capítulo de herramientas utilizadas.

## 6.2 Pruebas realizadas

### 6.2.1 Probando el Smart Contract

Para comprobar el correcto funcionamiento del Smart Contract puede hacerse uso de la suite de **Truffle**, esta suite ofrece un compilador y una herramienta de línea de comandos que puede ejecutar funciones del Smart Contract.

Se comienza implementando en el Smart Contract una función simple que permita obtener una respuesta por la línea de comandos, de forma que se pueda comprobar que el Smart Contract ha sido desplegado en la red correctamente.

---

<sup>2</sup>ver: <https://github.com/ethereum/go-ethereum/wiki/geth>

La función puede realizar una sencilla suma, como por ejemplo la siguiente.

```
1 function testSuma(uint128 numero) public pure returns (uint128){  
2   return (numero + 3);  
3 }
```

Listado de código 6.1: Función de prueba del Smart Contract

Si el Smart Contract se despliega correctamente, se debería mostrar por pantalla el resultado de sumar 3 al número introducido.

Primero se debe compilar el Smart Contract, para ello se debe ejecutar el siguiente comando dentro del directorio de trabajo de **Truffle**.

```
1 $ truffle compile
```

Listado de código 6.2: Compilar el Smart Contract

Posteriormente se debe desplegar el Smart Contract en la red Blockchain, para ello se ejecutan los siguientes comandos.

```
1 $ truffle developer  
2 $ truffle(develop)> migrate
```

Listado de código 6.3: Desplegar el Smart Contract

Una vez ha sido desplegado el Smart Contract, se podrá obtener una instancia del contrato para trabajar en local mediante la ejecución del comando siguiente dentro de la interfaz de Truffle developer.

```
1 $ truffle(develop)> Contract.deployed().then(inst => { instance = inst })
```

Listado de código 6.4: Obtener una instancia del Smart Contract

De esta forma se obtendrá acceso a las funciones del Smart Contract a través del objeto “instance”.

Para comprobar el correcto funcionamiento del Smart Contract, se puede proceder a ejecutar la función “testSuma” por ejemplo con el número 3.

```
truffle(develop)> Contract.deployed().then(inst => { instance = inst })  
undefined  
truffle(develop)> instance.testSuma(3)  
<BN: 6>  
truffle(develop)> □
```

Figura 6.1: Ejecución de la función de prueba del Smart Contract.

La suite de **Truffle** también permite obtener la dirección del Smart Contract, así como el ABI (figura 6.2 y 6.3 respectivamente) entre otras cosas.

```
truffle(develop)> instance.address  
'0xE37d5FACf8260f4B7626041436D0d0c7bEC9c74A'  
truffle(develop)> □
```

Figura 6.2: Obtener la dirección del Smart Contract.

```
truffle(develop)> instance.abi  
[ { constant: true,  
  inputs: [],  
  name: 'oracleAddress',  
  outputs: [ [Object] ],  
  payable: false,  
  stateMutability: 'view',  
  type: 'function',  
  signature: '0xa89ae4ba' },  
  { inputs: [ [Object] ],  
    payable: false,  
    stateMutability: 'nonpayable',  
    type: 'constructor',  
    constant: undefined },  
  { anonymous: false,  
    inputs: [ [Object] ],  
    name: 'obtenerHash',  
    type: 'event',  
    constant: undefined,  
    payable: undefined,  
    signature:  
      '0x92de516da46a85d860e29e22ddaf7ee934503fa99229e20b3f969d9840ce92b9' },  
  { constant: false,  
    inputs: [ [Object] ],  
    name: 'Guardar',  
    outputs: [],  
    payable: false,  
    stateMutability: 'nonpayable',
```

Figura 6.3: Obtener el ABI del Smart Contract.

### 6.2.2 Probando el prototipo

A continuación se exponen una serie de ejecuciones y pruebas realizadas en el prototipo, con el fin de demostrar que los resultados esperados coinciden con los obtenidos.

El primer paso es comprobar que el algoritmo SHA-256 se realiza correctamente, para ello definimos la siguiente secuencia de instrucciones en el Oráculo.

```
1 body = "hola";  
2 console.log("Cadena a guardar:");  
3 console.log(body);  
4  
5 console.log("Resultado tras realizar el HASH:");  
6 webHash = sha256(body);  
7 console.log(webHash);
```

Listado de código 6.5: Comprobar el algoritmo SHA-256

Esta secuencia imprime por pantalla la cadena de texto almacenada en “body” para posteriormente realizarle el hash SHA-256 con la función de

JavaScript, imprimiendo finalmente el resultado con el fin de comprobar que el hash se ha realizado.

```
Cadena a guardar:  
hola  
Resultado tras realizar el HASH:  
b221d9dbb083a7f33428d7c2a3c3198ae925614d70210e28716ccaa7cd4ddb79
```

Figura 6.4: Comprobación del algoritmo SHA-256 con el Oráculo

En la figura 6.4 puede observarse el resultado de realizar el hash a dicha cadena, que coincide efectivamente con el resultado mostrado en la figura 6.5 correspondiente a la ejecución del algoritmo SHA-256 desde una página web online, luego queda comprobado que el programa guarda correctamente el hash de la evidencia.

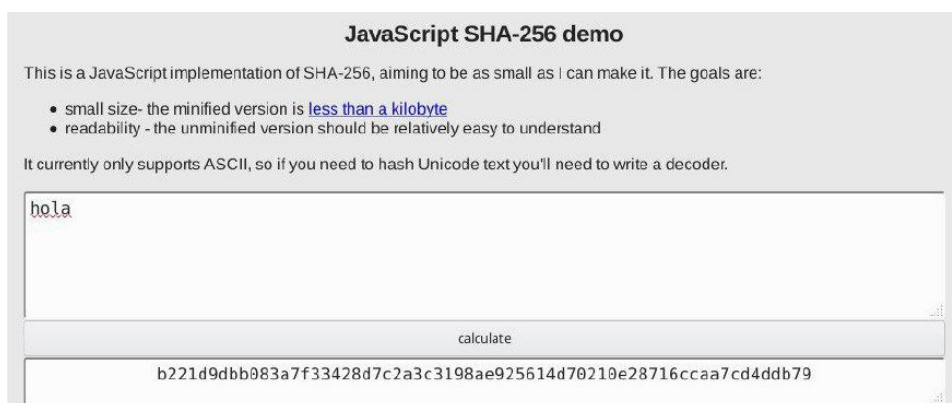


Figura 6.5: Comprobación del algoritmo SHA-256 desde una aplicación web

La siguiente prueba a realizar consiste en comprobar que el usuario tiene Metamask instalado, para ello se intenta acceder a la interfaz web desde un navegador distinto a Chrome y sin la extensión de Metamask, puede verse el mensaje de error en la figura 6.6 que impide al usuario hacer uso de la aplicación ya que no dispone de dicha extensión.

Finalmente queda comprobar que las evidencias se almacenan correctamente en la Blockchain, es necesario por lo tanto realizar una batería de pruebas en el programa con el objetivo de extraer una conclusión sobre el funcionamiento.

En primer lugar el usuario debe iniciar sesión en Metamask, de manera que cuando acceda a la interfaz web ésta se conecte con dicho programa. En caso de que se haya realizado la conexión, deberá aparecer el mensaje de la figura 6.7

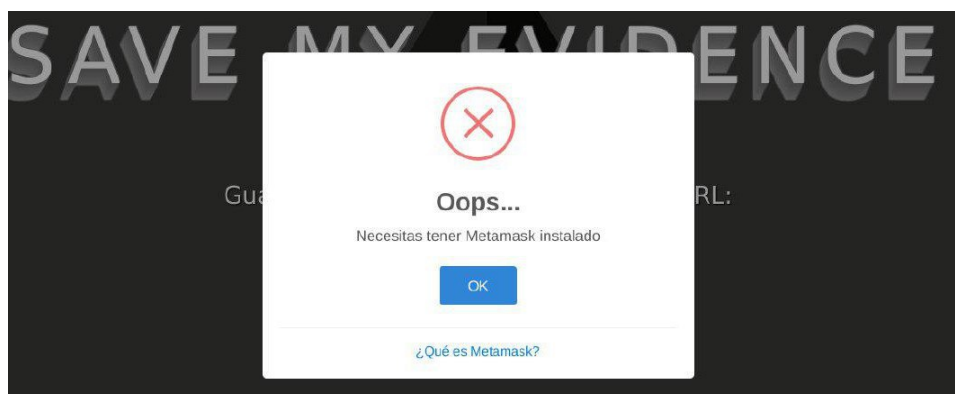


Figura 6.6: El usuario no tiene Metamask instalado

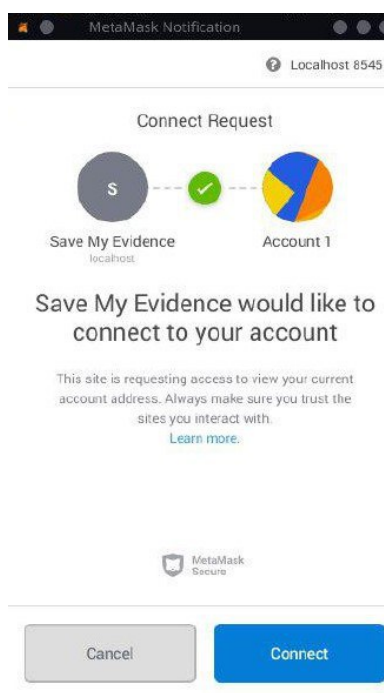


Figura 6.7: Conexión de la interfaz web con Metamask.

El siguiente paso es introducir la URL a evidenciar en el formulario de la página web, por ejemplo “https://www.ugr.es”, y hacer click en el botón “Guardar”. Si la conexión anterior se ha realizado correctamente, tras hacer esto Metamask solicitará al usuario permiso para enviar una transacción de origen Cliente y destino Oráculo, tal y como se ha visto en el apartado de implementación de Metamask, figura 6.8.

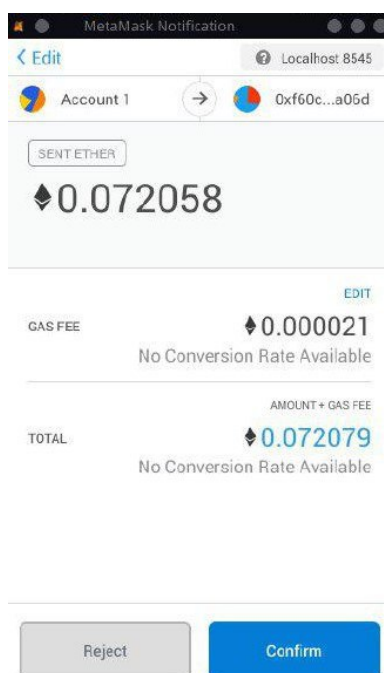


Figura 6.8: Metamask solicita permiso para realizar la transacción.

La red en la que se está ejecutando la aplicación es accesible desde “localhost” en el puerto 8545, tal y como se ve en la figura 6.7.

Metamask permite ver el historial de transacciones realizadas, para ello se debe acceder a la sección de la cuenta que está siendo utilizada, dentro de la página del programa. Tras realizar varias transacciones, el historial deberá tener una forma parecida a como se muestra en la figura 6.9.

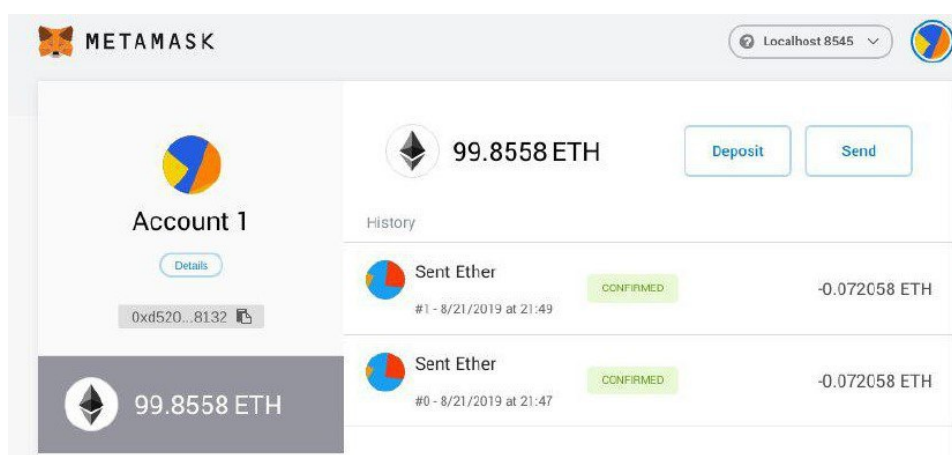
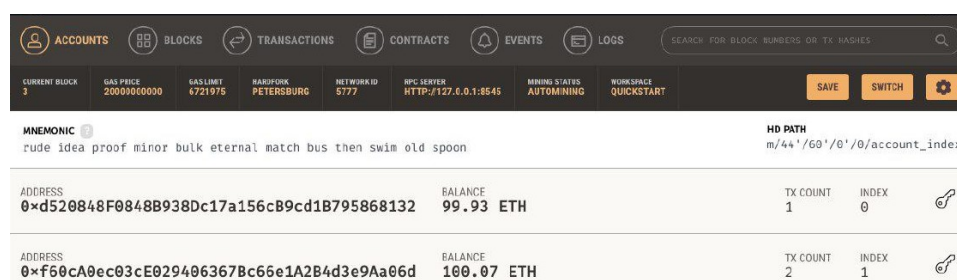


Figura 6.9: Historial de transacciones de Metamask.

A continuación, se evalúa el resultado con Ganache para ver la evidencia almacenada y las transacciones realizadas.

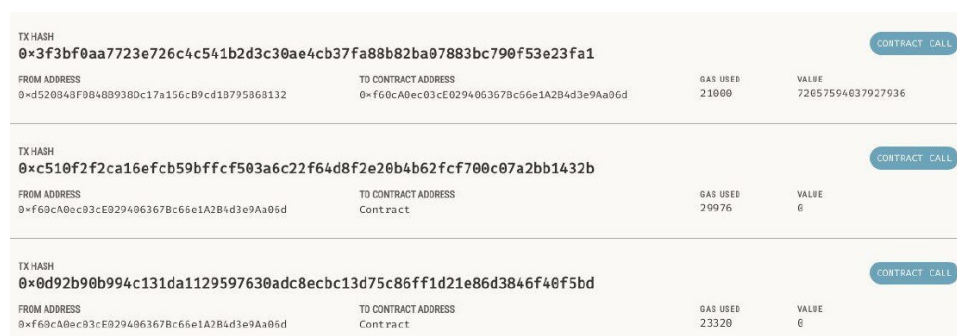
En primer lugar se comprueba que la cantidad de ETH de cada dirección ha cambiado, pues esto demuestra que la transacción realizada al almacenar la evidencia efectivamente se ha guardado en la red de Ganache, enviando los fondos solicitados desde la primera dirección que corresponde al Cliente, a la segunda dirección correspondiente al Oráculo.



ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS
CURRENT BLOCK 3	GAS PRICE 2000000000	GAS LIMIT 6721975	HARDFORK PETERSBURG	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:8545
MINING STATUS AUTOMINING					WORKSPACE QUICKSTART
MNEMONIC rude idea proof minor bulk eternal match bus then swim old spoon					HD PATH m/44'/60'/0'/0/account_index
ADDRESS 0xd520848f0848b9380c17a156c89cd1b795868132	BALANCE 99.93 ETH	TX COUNT 1	INDEX 0		
ADDRESS 0xf60cA0ec03cE029406367Bc66e1A2B4d3e9Aa06d	BALANCE 100.07 ETH	TX COUNT 2	INDEX 1		

Figura 6.10: Fondos en las direcciones de Ganache.

Una vez comprobado que los fondos de cada cuenta han cambiado, queda comprobar las transacciones realizadas. En el apartado de transacciones de Ganache puede verse como efectivamente aparece una transacción por cada operación efectuada, siendo la primera la correspondiente a la solicitud del Cliente, la segunda a la solicitud del Oráculo con la evidencia ya guardada y finalmente la tercera, correspondiente al pago monetario por el servicio prestado.



TX HASH	FROM ADDRESS	TO CONTRACT ADDRESS	GAS USED	VALUE
0x3f3bf0aa7723e726c4c541b2d3c30ae4cb37fa88b82ba07883bc790f53e23fa1	0xd520848f0848b9380c17a156c89cd1b795868132	0xf60cA0ec03cE029406367Bc66e1A2B4d3e9Aa06d	21000	72657594637927936
0xc510f2f2ca16efcb59bffc5f503a6c22f64d8f2e20b4b62fcf700c07a2bb1432b	0xf60cA0ec03cE029406367Bc66e1A2B4d3e9Aa06d	Contract	29976	0
0x0d92b90b994c131da1129597630adc8ecbc13d75c86ff1d21e86d3846f40f5bd	0xf60cA0ec03cE029406367Bc66e1A2B4d3e9Aa06d	Contract	23320	0

Figura 6.11: Transacciones realizadas en Ganache.

Entrando en detalle en cada transacción, al observar los parámetros del Smart Contract en la solicitud del Cliente puede apreciarse la URL solicitada junto con el timestamp.



CONTRACT	
CONTRACT Contract	ADDRESS 0xE37d5FACf8260f4B7626041436D0d0c7bEC9c74A
FUNCTION solicitarDatos(URL_solicitada: string, currentDate: uint256)	
INPUTS https://www.ugr.es, 1566475288	

Figura 6.12: Transacciones en Ganache correspondiente al Cliente.

Por otro lado en la transacción correspondiente al Oráculo, aparecen los datos del JSON almacenados tales como el título de la página web y el hash, así como el timestamp y el número de versión del programa.

CONTRACT	
CONTRACT Contract	ADDRESS 0xE37d5FACf8260f4B7626041436D0d0c7bEC9c74A
FUNCTION guardarSolicitud(timestamp: uint256, datosJSON: string, version: uint128)	
INPUTS 1562092913, {"Web":"Home   Universidad de Granada", "HASH":"8f93dbfb177e71ea51fe4c0ca6f80e8f580ab465b499c5334e90580dbaae6c27"}, 1	

Figura 6.13: Transacciones en Ganache correspondiente al Oráculo.

## Capítulo 7

# Conclusiones

Tras la finalización de este trabajo de fin de grado, se pueden extraer numerosas conclusiones.

Primero se comenzó exponiendo en qué consistía la cadena de custodia, qué problemas existían a la hora de custodiar una evidencia y los distintos agentes y procesos que se involucraban el hecho de guardar una evidencia forense sobre un delito. Se llegó a la conclusión de que había demasiados cabos sueltos por los que la custodia de una evidencia forense podía ser vulnerable, desde la recolección de la evidencia hasta la muestra de la misma ante un tribunal, pasando por la custodia de la misma, recayendo la confianza en terceras partes.

Después se explicó en qué consiste la tecnología Blockchain, así como sus principales características y contenidos teóricos. Definiendo los conceptos básicos de base de datos, descentralizada y distribuida, así como en qué consiste la red Blockchain con un modelo simplificado de la misma. Realizando transacciones con una escala pequeña de usuarios y explicando qué se almacena realmente en la red Blockchain, los bloques junto con las transacciones.

Se realizaron una serie de definiciones sobre los problemas que presenta el proceso de la cadena de custodia, junto con una comparativa de las características de la tecnología Blockchain que solventan dichos problemas. El que la red sea distribuida, no sean necesarias terceras partes y los nodos puedan evaluar la fiabilidad de la transacción a realizar son características cruciales para escoger finalmente este modelo frente a un almacenamiento clásico en una base de datos centralizada. Llegando a la conclusión de que la red Blockchain permite un ahorro en costes, tiempo de trabajo y personas en las que depositar la confianza, eslabones débiles de la cadena de custodia.

Posteriormente se definió el modelo de un sistema de almacenamiento

de evidencias forenses basado en la red Blockchain con el fin de permitir resolver los problemas antes definidos que conlleva la cadena de custodia, demostrando cómo la red Blockchain posee características que resuelven gran parte de dichos problemas. Tras esto, se concluyó en un prototipo que posibilita el almacenamiento de evidencias sobre el contenido de páginas web, buscando realizar el diseño e implementación al completo teniendo en mente la facilidad de uso para el usuario.

Previo a la implementación del prototipo, se determinaron los agentes que participan en dicho programa junto con los diagramas de todas las interacciones que pudiese haber entre ellos, incluyendo las comunicaciones dentro y fuera de la Blockchain, los posibles despliegues que pueden realizarse de los Smart Contracts, así como las comunicaciones del servidor web, responsable de proveer de la interfaz al usuario, con la red Blockchain.

Se desarrolló en profundidad el prototipo, mostrando con detenimiento la implementación en código, la instalación y los programas y conocimientos necesarios para hacerlo funcionar. Explicando en el capítulo correspondiente a la implementación la funcionalidad de cada línea de código importante en el programa, para luego demostrar el correcto funcionamiento con numerosos ejemplos y pruebas de ejecución. Concluyendo este trabajo de fin de grado con las pruebas realizadas para mostrar la operatividad del prototipo, junto con los pasos necesarios a seguir para repetir dicha batería de pruebas.

## 7.1 Valoración personal

En cuanto a lo personal, considero que la tecnología Blockchain tiene mucho potencial, no solo para este tipo de problemas sino para ser aplicada a muchos otros. Las características que posee dicha red y que han sido expuestas en este trabajo de fin de grado demuestran que ante numerosos problemas puede adoptarse una solución basada en la red Blockchain aunque el problema actualmente no se encuentre implementado en dicho modelo.

Un modelo de red Blockchain aporta diversas ventajas, pues se alcanza una disposición en la que la red se gestiona a sí misma, sin necesidad de intermediarios dentro de la propia red. Esto eliminaría por ejemplo los problemas burocráticos a la hora de realizar un contrato de trabajo, acotando únicamente el contrato a un programa informático encargado de producir una respuesta para una determinada situación, depositando la confianza en el código. Aplicando la red Blockchain, se eliminaría el tener que pagar a una tercera persona para validar cualquier tipo de información, así como impuestos y partes extra implicadas, pues serían los clientes, la transacción y la red los únicos que participarían en la ecuación.

Cabe destacar que resulta difícil llegar a conocer completamente esta tecnología, pues al ser un concepto relativamente nuevo tiene una curva de aprendizaje muy alta. Constantemente se publican modificaciones y vulnerabilidades nuevas, estando en un estado de cambio continuo. Además, los conceptos a entender son muy abstractos lo que da lugar a que para tener una idea general de en qué consiste la red Blockchain se necesiten numerosas fuentes de información y tiempo, sobretodo tiempo.

Blockchain es una tecnología que a pesar de las utilidades demostradas en este trabajo, no está lo suficientemente estandarizada, dando lugar a que no todo el mundo conozca sus características principales. Es difícil que esto ocurra pues requiere de tiempo y esfuerzo llevar estos conceptos a un público que carezca de conocimiento sobre informática. Sin embargo, considero que en el momento en el que se extienda y se empiecen a estandarizar las funcionalidades de esta tecnología, aparecerán más programas informáticos que utilicen la tecnología Blockchain y, por supuesto, ingenieros dispuestos a crearlos.

## Capítulo 8

# Bibliografía

- [1] Nakamoto, S. *Bitcoin: A peer-to-peer electronic cash system (2008)*. <http://www.bitcoin.org/bitcoin.pdf> [Online; accedido el 24-04-2019]
- [2] ANGULO GONZÁLES, Rubén Darío. “*Cadena de Custodia en Criminalística*”. Ediciones Doctrina y ley. Colombia, 2005.
- [3] Silvia Bonomi, Marco Casini y Claudio Ciccotelli. *B-CoC: A Blockchain-based Chain of Custody for Evidences Management in Digital Forensics*. <https://arxiv.org/pdf/1807.10359.pdf> [Online; accedido el 24-04-2019]
- [4] Imran Bashir. *Mastering blockchain*. Packt Publishing Ltd, 2017.
- [5] Imran Bashir. *Mastering blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd, 2018.
- [6] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the open blockchain*. ”O’Reilly Media, Inc.”, 2017.
- [7] Chris Dannen. *Introducing Ethereum and Solidity*. Springer, 2017.
- [8] Mayukh Mukhopadhyay. *Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity*. Packt Publishing Ltd, 2018.
- [9] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O’Reilly Media, 2018
- [10] Adam Gall. *Building your first Ethereum Oracle*. <https://medium.com/decentlabs/building-your-first-ethereum-oracle-1ab4cccf0b31> [Online; accedido el 24-04-2019]
- [11] Jesus Najera. *Blockchain Oracles: What They Are and Why They’re Necessary*. <https://medium.com/@setzeus/blockchain-oracles-af3b216bed6b> [Online; accedido el 05-06-2019]

- [12] Lovable Technology. *Building an “Oracle” for an Ethereum contract*. <https://medium.com/@mustwin/building-an-oracle-for-an-ethereum-contract-6096d3e39551> [Online; accedido el 05-06-2019]
- [13] Kendrick Tan. *Oracles in Ethereum - A Simple Guide (Nov 6, 2017)*. <https://kndrck.co/posts/ethereum/oracles/a/simple/guide/> [Online; accedido el 05-06-2019]
- [14] John R. Kosinski. *Ethereum Oracle Contracts: Setup and Orientation*. <https://www.toptal.com/ethereum/ethereum-oracle-contracts-tutorial-pt1> [Online; accedido el 30-May-2018]
- [15] Web3. <https://web3js.readthedocs.io/en/v1.2.1>, [Online; accedido el 05-06-2019]
- [16] Truffle Suite. <https://www.trufflesuite.com/>, [Online; accedido el 27-05-2019]
- [17] Geth. <https://github.com/ethereum/go-ethereum/wiki/geth>, [Online; accedido el 27-05-2019]
- [18] Solidity. <https://solidity.readthedocs.io>, [Online; accedido el 27-05-2019]
- [19] NodeJS. <https://nodejs.org/en/docs/> [Online; accedido el 27-05-2019]
- [20] Express Framework. <https://expressjs.com/> [Online; accedido el 15-06-2019]
- [21] Oraclize - Provable. <http://provable.xyz/> [Online; accedido el 15-06-2019]
- [22] Lukas Marx. *Storing Data on the Blockchain: The Developers Guide (5 Julio 2018)*. <https://malcoded.com/posts/storing-data-blockchain/> [Online; accedido el 15-06-2019]
- [23] Gary Simon. *Interacting with a Smart Contract through Web3.js (24 Oct 2017)*. <https://coursetro.com/posts/code/99/Interacting-with-a-Smart-Contract-through-Web3>, [Online; accedido el 15-06-2019]
- [24] Metamask. <https://metamask.io>, [Online; accedido el 15-06-2019]
- [25] Sotirios Brotsis, Nicholas Kolokotronis, Konstantinos Limniotis, Stavros Shiaeles, Dimitris Kavallieros, Emanuele Bellini y Clement Pavué. *Blockchain Solutions for Forensic Evidence Preservation in IoT Environments*. <https://www.researchgate.net>, [Online; accedido el 15-06-2019]

## Anexo A

# Manual de usuario

El prototipo ha sido desarrollado bajo la distribución Kali Linux<sup>1</sup> en su versión 2019.2. Es imprescindible que todos los paquetes descritos en el capítulo de librerías necesarias se encuentren instalados en el sistema, así como Google Chrome con la extensión de Metamask, Truffle y Ganache.

Para ejecutar el programa implementado se deben seguir los siguientes pasos.

- **(1):** Descargar el prototipo de mi GitHub, éste se encuentra en la dirección: <https://github.com/rubcv/TFG>.
- **(2):** Una vez descargado, dirigirse a la carpeta “SRC” con una terminal y ejecutar el script “recompile.sh” para compilar todos los contratos del directorio “contracts/” con Truffle.
- **(3):** Seguido de este paso se debe ejecutar Ganache. Se puede optar por la versión con o sin interfaz gráfica, en caso de ejecutar la interfaz gráfica es necesario abrir el proyecto dirigiéndose al directorio “SRC” desde Ganache y seleccionando el archivo de configuración “truffle-config.js”, esto cargará los contratos disponibles en el programa de Ganache.

---

<sup>1</sup><https://www.kali.org/>

## WORKSPACE

WORKSPACE NAME

TRUFFLE PROJECTS

```
/root/TFG/ORACLE-UI-v2/client/truffle-config.js
```

ADD PROJECT

REMOVE PROJECT

Figura A.1: Abrir el proyecto con Ganache

- (4): En una terminal a parte se debe ejecutar Truffle y acceder a la interfaz de “developer” para migrar los contratos en la red, de forma que se encuentren desplegados en la interfaz gráfica de Ganache. Para ello hay que ejecutar el comando “truffle developer” y, una vez dentro de en la interfaz “developer” ejecutar “migrate”.

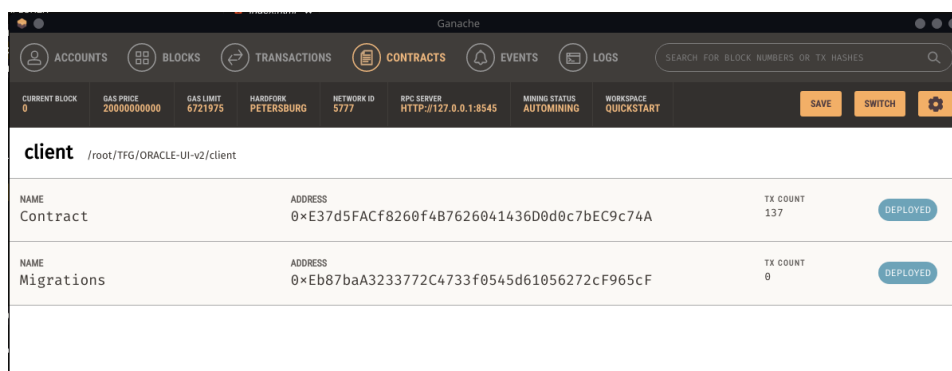


Figura A.2: Contratos desplegados en Ganache

- (5): Fuera de Truffle developer, es necesario lanzar el servidor web para poder acceder a la página con la interfaz gráfica desde un navegador. Esto se realiza iniciando el servidor con el comando “node app.js”



dentro del directorio “server”. El programa “Save My Evidence” estará accesible en el navegador ingresando en “localhost” mediante el puerto 8888.

**SAVE MY EVIDENCE**

Guardar evidencia forense para esta URL:

Escribe aquí la URL

Guardar

**Save My Evidence:**

Save My Evidence es un programa para guardar evidencias de contenido en páginas web, en el transcurso utiliza la tecnología **Blockchain** para asegurar la integridad de la información.

Para utilizar esta herramienta:

- Debes tener instalado **Metamask**.
- **Escribe** en el formulario la **URL** de la cual deseas obtener la evidencia.
- Ejemplo: <https://www.ejemplo.com>
- Pulsa sobre el botón **Guardar**.
- Al guardar la evidencia se realizará una transacción, podrás ver el **resultado** con Metamask.
- Se te descontará del saldo la cantidad correspondiente.
- Podrás ver la evidencia almacenada en la red utilizada, por defecto **Ganache**.

Los datos almacenados sobre cada evidencia son los siguientes:

- **timestamp:** Fecha en formato Unix del momento en que se guardó.
- **URL original:** URL sobre la que se ha guardado la evidencia.
- **Datos:** Datos en formato JSON, según la versión se almacenan distintos datos, en la versión 1 se almacena el HASH de la web.
- **Versión:** Versión actual del programa, según la versión se almacenan distintos datos, en la versión 1 se almacena el HASH.

Para más información contactar con Rubén: [rubencav@correo.ugr.es](mailto:rubencav@correo.ugr.es)

Figura A.3: Aplicación web “Save My Evidence”

- (6): En un navegador con Metamask instalado y vinculado a la aplicación “Save My Evidence” puede procederse a almacenar la evidencia de la página web deseada escribiendo la URL en el formulario de la página. Destacar que para realizar la transacción es necesario que Metamask haya sido iniciado con el usuario de Ganache y se encuentre con la red “localhost:8545” seleccionada. El programa de Ganache comenzará por defecto con la dirección del Cliente.
- (7): Una vez realizado el almacenamiento de la evidencia puede verse

el resultado en el apartado de transacciones de Ganache, así como la transferencia de ETH de una dirección a otra en la página que muestra la información del saldo junto con las respectivas direcciones disponibles.

CONTRACT	
CONTRACT	ADDRESS
Contract	0xE37d5FACf8260f4B7626041436D0d0c7bEC9c74A
FUNCTION	
guardarSolicitud(timestamp: uint256, datosJSON: string, version: uint128)	
INPUTS	
1562092913, {"Web":"Home   Universidad de Granada","HASH":"8f93dbfb177e71ea51fe4c0ca6f80e8f580ab465b499c5334e90580dbaae6c27"}, 1	

Figura A.4: Ejemplo de evidencia almacenada