Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Trabajo de Fin de Máster

Máster Universitario en Ciberseguridad e Inteligencia de los Datos

Aplicación de voto basada en cadena de bloques

Blockchain based voting application

Alien Embarec Riadi

La Laguna, 7 de septiembre de 2021

D. Cándido Caballero Gil, con N.I.F. 42.201.070-A, profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. Néstor Álvarez Díaz, con N.I.F. 54.115.839-J, doctorando en el programa de Doctorado de Ingeniería Industrial, Informática y Medioambiental.

C E R T I F I C A (N)

Que la presente memoria titulada:

“Aplicación de voto basada en cadena de bloques”

ha sido realizada bajo su dirección por D. **Alien Embarec Riadi**,

con N.I.F. 46.264.322-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de septiembre de 2021

Agradecimientos

A mis padres y seres queridos por el apoyo incondicional y continuo.

A mis tutores, Cándido y Néstor, por la excelente orientación durante el desarrollo del TFM.

Licencia

Resumen

El objetivo de este trabajo ha sido desarrollar una aplicación web de voto electrónico basada en la cadena de bloques.

De manera simultánea a la competencia en *Blockchain*, se persigue adquirir conocimientos en criptografía, en lo que concierne a asegurar la custodia del voto, y operar de manera confidencial sobre el mismo, sin revelar la intención del votante.

Estos requisitos se desarrollan a través del cifrado homomórfico, cuya característica principal es la capacidad de realizar operaciones sobre datos cifrados.

Los requerimientos de cada elección se especifican en un contrato electrónico que recoge los permisos y privilegios de todos los participantes de los comicios: votantes, candidatos y administradores.

Mediante los procesos de criptografía y cadena de bloques se pretende desarrollar una herramienta innovadora, y al mismo tiempo que se constituya como una alternativa a los procesos electorales tradicionales.

Palabras clave: contrato, votación, candidato, descentralización, transacción, bloque, algoritmo de consenso, red de pares, computación distribuida, *gas*, censo, circunscripción, *Ether*

Abstract

The objective of this work has been to develop an electronic voting web application based on the blockchain.

Simultaneously to the competence in Blockchain, it is pursued to acquire knowledge in cryptography, in what concerns to ensure the custody of the vote, and to operate confidentially on it, without revealing the voter's intention.

These requirements are developed through homomorphic encryption, whose main feature is the ability to perform operations on encrypted data.

The terms of each election are specified in an electronic contract that contains the permissions and privileges of all participants in the election: voters, candidates and administrators.

The cryptographic and blockchain processes are intended to develop an innovative tool, and at the same time an alternative to traditional electoral processes.

Keywords: contract, votation, candidate, decentralization, transaction, block, consensus algorithm, *peer-to-peer* network, distributed computing, *gas*, census, constituency, *Ether*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1 Motivación

El auge de la tecnología denominada cadena de bloques (*blockchain*) ha modificado la operativa de muchos sectores que hasta hace unos años se encontraban dominados por una tercera entidad intermediaria en las transacciones entre dos partes.

Figura 1.1: Logo de Bitcoin. Fuente: [24]

Su origen se remonta al año 1982, cuando el criptógrafo David Chaum propuso en una publicación [1] un protocolo similar a la *blockchain* titulado "*Sistemas informáticos establecidos, mantenidos y de confianza por grupos mutuamente sospechosos*". En 1991, Stuart Haber y W. Scott Stornetta publicaron un trabajo [2] que profundizó en la misma temática, una cadena de bloques criptográficos segura.

Haber y Stornetta introdujeron los árboles de *Merkle* en el diseño, querían implantar un sistema en el que las marcas de tiempo de los documentos no pudieran ser manipuladas.

La primera *blockchain* [3] referida con mayor nivel de detalle fue publicada en 2008 por un investigador bajo el seudónimo de Satoshi Nakamoto.

El diseño fue implementado al año siguiente por Nakamoto como un componente central de la criptomoneda *Bitcoin*, donde sirve como el libro de contabilidad público para todas las transacciones en la red.

El crecimiento de esta tecnología de la mano de las criptodivisas ha aflorado un

número importante de sectores que podrían hacer uso de ella para mejorar la gestión del negocio. El uso principal consiste en la capacidad de firmar acuerdos y transacciones entre dos entidades sin necesidad de una tercera parte de confianza.

Al ser la cadena de bloques pública, y utilizarse la criptografía para autorizar cambios en el estado de la red, se eliminan las inquietudes de los participantes, y se sustituye la autoridad intermediaria -que recauda una comisión a cambio de monitorizar las transacciones-, por la tecnología expuesta en este trabajo.

Uno de los avances más significativos que ha experimentado la *blockchain* recientemente consiste en la aparición de los contratos inteligentes (*smart contracts*). Estos consisten en piezas de código donde se pueden programar los términos de un acuerdo que rige un evento (ej: elección, banca, logística, compra-venta, etc).

Presentan múltiples ventajas, entre ellas la capacidad de delegar la firma y seguimiento de acuerdos a la tecnología.

Por otro lado, todos los términos del contrato quedan registrados en la cadena de bloques, lo que dota de mayor transparencia y trazabilidad los entendimientos entre las partes.

1.2 Objetivos

El objetivo principal de este Trabajo Fin de Máster consiste en llevar a cabo el desarrollo de una aplicación web en la que se emplee la *blockchain* al ámbito de las votaciones electrónicas.

En concreto se desea enfocar el uso de la aplicación a la capacidad de crear elecciones, registrar candidatos, votantes y administradores, y realizar el recuento y custodia de los votos mediante la tecnología señalada.

El uso de contratos inteligentes y la cadena de bloques en un sistema de votación se encuentra en una fase emergente, y por ello, se ha creído innovador y relevante orientar el TFM hacia la resolución del objetivo anteriormente mencionado.

Por otra parte, las nuevas formas de organizar elecciones extensivas buscan encontrar aceptación ante las autoridades, ya que suponen un compromiso para la transparencia, autenticidad e integridad de todo el proceso.

Por ello, de la mano de esta tecnología reciente se busca superar los retos de la votación electrónica, y reducir las dudas y sospechas de fraude que reciben las elecciones en general.

En primer lugar, la *blockchain* permitirá seguir el recuento de manera directa, y rastrear el destino de cada voto. Esto gracias a la estructura de árboles de *Merkle* enlazados criptográficamente, lo que otorga propiedades de inmutabilidad a todo el proceso, una cualidad exigible en toda votación.

1.3 Especificación del problema

Una elección es la designación, generalmente por votación, de una persona para ocupar un puesto en una comisión, consejo u organismo semejante.

La votación es la forma en la que una multitud elige a uno o varios individuos para desempeñar funciones de representación y gestión de los bienes comunes. Por ello, en todo proceso que implique ostentar cargos de poder ha de existir un proceso electoral transparente mediante el cual se seleccionan las personas que van a liderar la representación.

El derecho a votar a través de sufragio directo, transparencia y limpieza de un proceso electoral está claramente relacionado con los índices de democracia y participación en una institución u entidad [4].

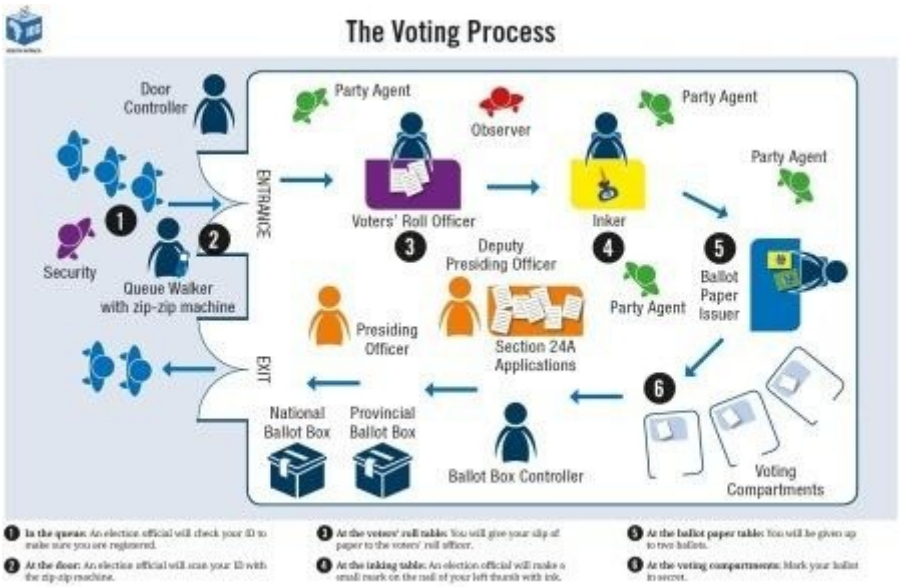

Figura 1.2: Fases del proceso de emisión del voto. Fuente: [25]

En la anterior figura es posible contemplar los controles que se llevan a cabo durante una votación presencial, afectando el desarrollo de la misma y desperdiciando una cantidad de tiempo importante para los votantes.

A menudo hay una aglomeración de votantes a la entrada del centro donde se encuentran las urnas, en el cual se efectúa un control de identidad de los ciudadanos confirmando que han sido convocados en el centro asignado por la comisión de administradores (*Paso 1*).

Seguidamente, en el *Paso 2*, un empleado (*Queue Walker*) escanea la identidad de los ciudadanos con derecho a voto en el centro, para que se pueda anular su derecho a votar por segunda vez cuando depositen la papeleta en la urna.

En el *Paso 3*, se entrega el documento de identidad a un funcionario para realizar comprobaciones adicionales relacionadas con el censo y la circunscripción.

En el *Paso 4*, *inking table* representa la mesa donde se pone una marca de tinta en el dedo pulgar izquierdo del votante. Esta marca, debido a la dificultad de borrado, impide el voto por segunda vez.

En los *Pasos 5 y 6*, se seleccionan dos papeletas, para el voto provincial y estatal, se anotan de manera secreta y se depositan en la urna.

Generalmente, en los procesos de votación tradicionales siempre hay un margen de sospecha de manipulación de los resultados, y el votante no tiene la posibilidad de ver cómo ha sido gestionado su voto.

En ocasiones no existe la certeza de que el voto haya sido válido, o haya sido destinado al candidato declarado en la papeleta, etc. En definitiva, hay un número importante de inquietudes en torno a los comicios presenciales.

La fiabilidad de los procesos plebiscitarios es una cualidad importante en todo país con estándares democráticos [26]. La tecnología expuesta en el presente trabajo intenta transformar la votación a un proceso telemático donde:

- Cada voto se apila en una cadena de bloques restringidos a modificaciones mediante un sistema de claves criptográficas. Gracias a esta característica se reduce la sospecha de amaño de manera considerable.
- Todos los votantes pueden observar los votos acumulados por cada candidato en tiempo real, sin ser revelado el sentido de voto de cada elector.
- Toda la información de usuarios autorizados a emitir su preferencia se encuentra almacenada de manera distribuida, y se verifica que el voto haya sido emitido de manera única para evitar falsificaciones.
- La Comisión de Administradores es pública, y tiene disponibles una serie de poderes para organizar el sufragio.
- El plazo de apertura y cierre de la elección se puede consultar de manera anticipada, y una vez finalizado, se bloquea el acceso a la aplicación para votar, y se transita a la fase de recuento.

1.4 Solución propuesta

Una vez presentada la naturaleza del problema que se quiere abordar y cómo la cadena de bloques podría ayudar en dicha tarea, a continuación se enumeran de manera más detallada los objetivos específicos del Trabajo Fin de Máster.

Se pretende construir una aplicación web para la plataforma *Ethereum* que a través de contratos inteligentes suministre los mecanismos necesarios para celebrar una elección con el fin de elegir una persona o asunto.

En la aplicación existen tres tipos de usuarios: Votante, Candidato, y Administrador. Los

permisos y limitaciones de cada rol son los siguientes:

- Votante (V): Solo puede enviar el voto una única vez, ha de estar inscrito por el Administrador en una o varias elecciones para disponer del derecho a votar.
- Candidato (C): Tiene fijada como incompatibilidad ejercer de Administrador de manera simultánea. Puede hacer uso de su derecho al voto. Ha de estar inscrito por el Administrador para presentar su candidatura.
- Administrador (A): Dispone de permisos para crear elecciones, e inscribir candidatos y votantes en las mismas.

La manera de funcionar de la aplicación consta de tres pasos. Primero se ha de crear la elección, seguidamente se declaran los candidatos y votantes autorizados, y finalmente, se da comienzo al proceso de votación, y una vez vencido el plazo, se publican los resultados finales.

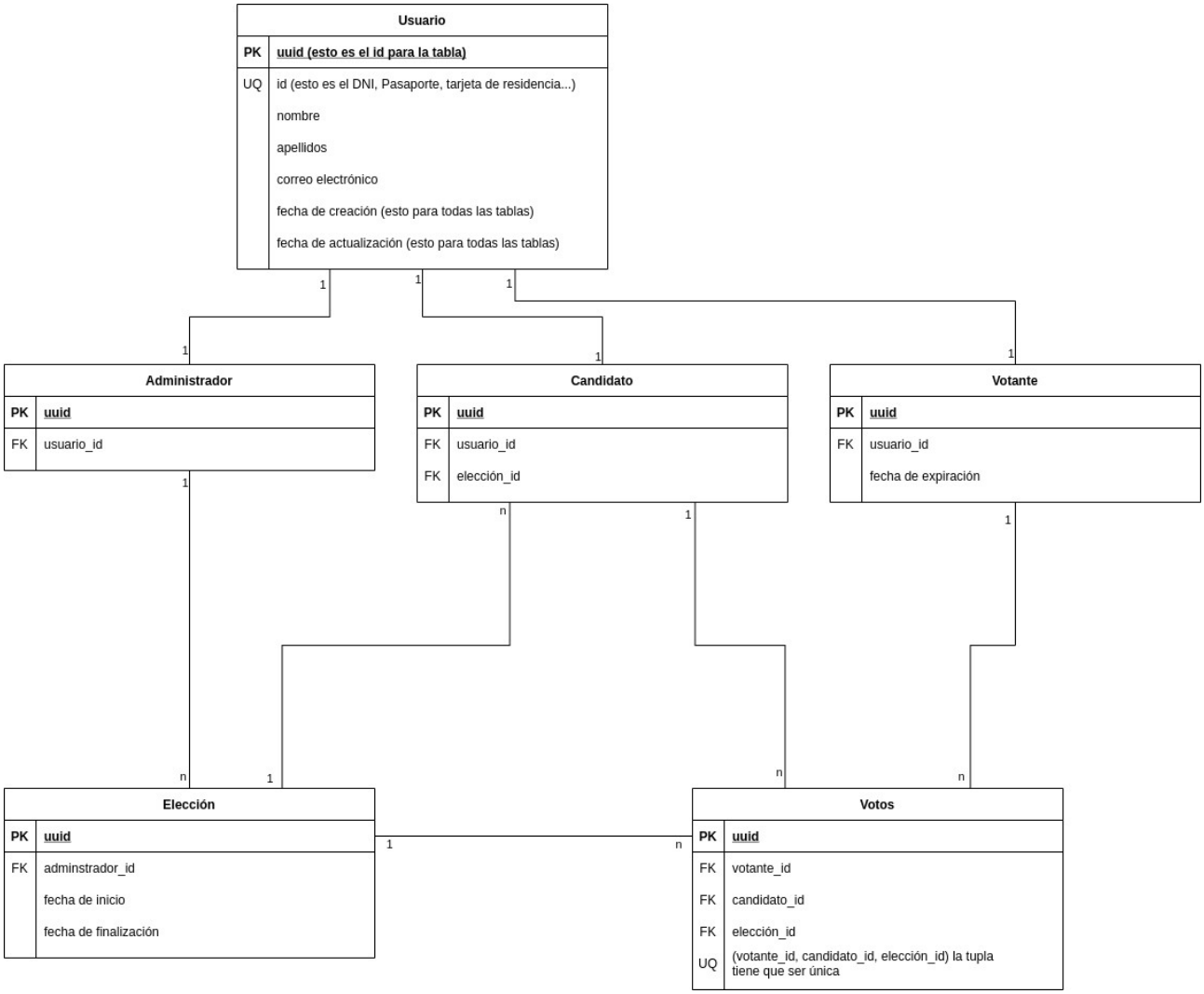

Figura 1.3: Modelo UML de la elección

En el modelo UML de la anterior figura se representan los atributos importantes de la elección, y las relaciones que se establecen entre ellos. Se puede observar la estructura lógica basada en claves ajenas.

Se parte de la base de que todos los participantes son usuarios con las mismas características, que pueden tener una de las tres categorías: Administrador, Candidato y Votante. Por ello, se crean tres tablas donde el identificador del usuario corresponde a una clave ajena que apunta al identificador original del usuario, es decir, la tabla *Usuario*.

Finalmente, los pormenores de los dos eventos principales, el modelo de elección, y los atributos necesarios para emitir el voto se almacenan en la tabla *Elección* y *Votos* respectivamente.

1.5 Fases de desarrollo

A continuación se presentan las fases que han constituido el desarrollo del Trabajo Fin de Máster.

1. Estudio e investigación iniciales: Adquisición de los conocimientos básicos sobre la cadena de bloques. El concepto de *nonce*, prueba de trabajo, bloque, criptografía homomórfica, *Bitcoin*, *hash*, árbol de *Merkle*, dirección de *Bitcoin*, transacción, *Ethereum*, *MetaMask*, *Ganache*, *Truffle*.

2. Definición de requisitos: Especificación de características y funcionalidad que ha de incorporar la aplicación.

3. Selección de tecnologías: Elección de las tecnologías a utilizar para el desarrollo de la aplicación tomando en consideración los requisitos fijados en la fase previa.

4. Desarrollo: Fase de desarrollo de la aplicación web. Ésta ha de incorporar las características enunciadas en la segunda fase y se ha de implementar haciendo uso de las tecnologías seleccionadas en la tercera fase.

5. Redacción de la memoria: Transcripción del trabajo realizado y los resultados obtenidos en la memoria del TFM.

El periodo de tiempo dedicado a la elaboración del Trabajo Fin de Máster abarca el intervalo comprendido entre el 15 de Diciembre de 2020 y el 6 de Septiembre de 2021.

1.6 Estructura de la memoria

Los contenidos de los siguientes apartados de esta memoria se presentan de acuerdo a la siguiente estructura.

En el Capítulo 2 se recoge la definición y explicación del funcionamiento de la tecnología *blockchain*, se profundiza en el concepto de los *smart contracts* y se expone la plataforma para la que se ha desarrollado la aplicación (*Ethereum*).

Las tecnologías y herramientas seleccionadas para la creación de la aplicación se presentan en el Capítulo 3. Junto a las tecnologías, se exponen los *smart contracts* implementados para abordar el objetivo del proyecto y el funcionamiento de la aplicación web construida.

El Capítulo 6 muestra un presupuesto estimado para cada una de las tareas que han conformado el desarrollo del proyecto. Por último, en los capítulos 4 y 5 se presentan las conclusiones alcanzadas tras el desarrollo del Trabajo Fin de Máster, y posibles mejoras a introducir en versiones futuras en los idiomas español e inglés respectivamente.

Capítulo 2
Antecedentes

En el transcurso de este capítulo se presentan de forma detallada las definiciones y el funcionamiento de los conceptos fundamentales en torno a los que se articula el presente Trabajo Fin de Máster. En primer lugar se explicarán algunas herramientas criptográficas sobre las que se construye la tecnología *blockchain*.

Seguidamente, se comentará el funcionamiento de esta tecnología y de qué manera se consigue que cuente con las características adecuadas para resolver el problema expuesto en este informe.

Además, se ahondará en el concepto de *smart contract* y en la importancia de los mismos en cuanto al gran crecimiento de las posibilidades de aplicación de la cadena de bloques.

Por último, se introducirá *Ethereum*, plataforma para la que se ha desarrollado la aplicación web objetivo del proyecto.

2.1 Criptografía

Antes de explicar el funcionamiento de la tecnología *blockchain* resulta indispensable introducir una serie de conceptos criptográficos en los que se basa.

2.1.1 Función hash

Una función *hash* [5] criptográfica es una función trampa que convierte un mensaje de entrada a un resumen o *hash* de tamaño fijo. Además, las funciones *hash* han de cumplir con una serie de propiedades:

- Debe ser capaz de calcular el resumen de manera eficaz independientemente del mensaje de entrada.
- Dado un resumen, ha de ser extremadamente complejo en términos de computación generar o calcular un mensaje de entrada que produzca dicho resumen.

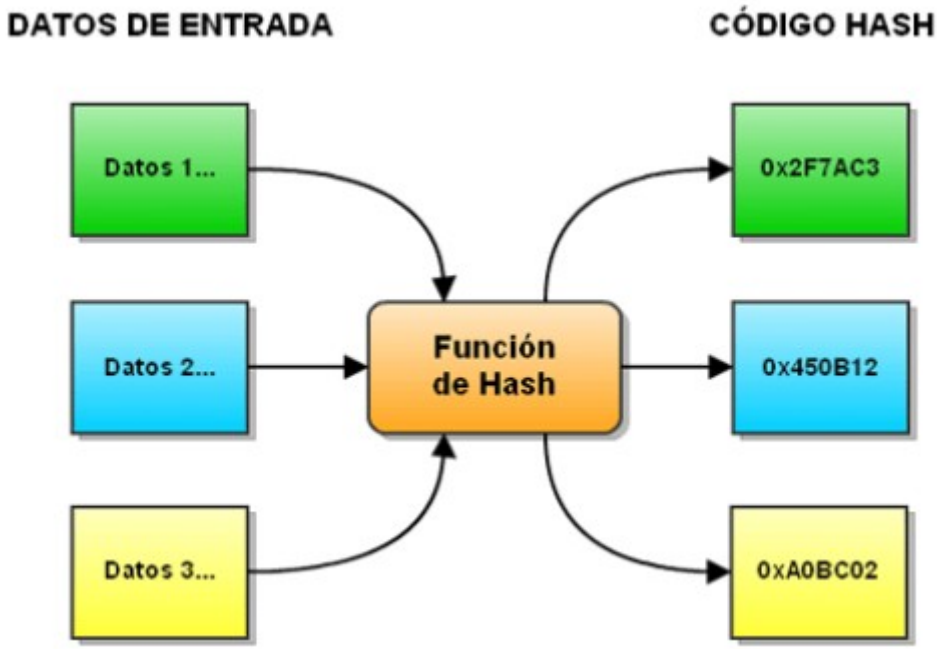

Figura 2.1: Ejemplo de una función has. Fuente: [27]

- La probabilidad de que dos textos de entrada ligeramente distintos colisionen produciendo el mismo resumen ha de ser extremadamente baja.
- Ha de ser una función determinista, esto es, para el mismo mensaje ha de devolver siempre el mismo resumen.

La *Figura 2.1* presenta una ilustración del ejemplo del funcionamiento de una función *hash* criptográfica. A partir de dos textos de entrada cualesquiera, se generan dos resúmenes del mismo tamaño.

2.1.2 Criptografía asimétrica

La criptografía asimétrica [6] o criptografía de clave pública es un sistema criptográfico en el que se utiliza una pareja de claves para el envío de mensajes. Cada persona o entidad dispone de una clave pública y una clave privada.

La clave pública puede ser compartida con las demás personas o entidades del sistema sin comprometer la seguridad del mismo, mientras que la clave privada solo debe ser accesible por su propietario.

El papel que juega en el presente trabajo está relacionado con que cada usuario dispone de una cuenta generada a partir de un par de claves pública y privada.

La clave pública se utiliza como dirección visible para los demás usuarios, de manera semejante a un número de cuenta (IBAN), mientras que la clave privada no debe ser revelada y se utiliza para validar la identidad del usuario en las transacciones.

2.2 Blockchain

Blockchain es una tecnología caracterizada por una estructura de datos formada por bloques que permiten el registro de transacciones entre partes de manera segura y permanente, puede entenderse como una base de datos descentralizada, consensuada y distribuida.

Las cadenas de bloques suelen ser gestionadas por una red de pares para su uso como libro de contabilidad distribuido públicamente, donde los nodos se adhieren de forma colectiva a un protocolo para comunicarse y validar nuevos bloques.

2.2.1 Historia

Como se ha señalado en el apartado 1.1, su origen se remonta a 1982, cuando David Chaum propuso [1] un protocolo de transacción entre partes mutuamente desconfiadas.

Más adelante, en 1991, Haber y Stornetta [2] plantean la idea de encadenar bloques de información asegurados criptográficamente. Sin embargo, no fue hasta el año 2008 cuando comenzó a ganar relevancia entre la comunidad científica.

La primera aplicación real de esta tecnología fue el sistema de pago de criptomonedas *Bitcoin*, como resultado de la difusión del informe de investigación por Nakamoto *[3]* en 2008, que supuso la primera propuesta de un sistema de pago electrónico basado totalmente en criptografía y en el que se elimina el problema del doble gasto.

2.2.2 Modelo tradicional

Para abordar dicho problema, los sistemas tradicionales implementan un modelo basado en confianza en el que una tercera parte vela por la fiabilidad de las transacciones.

Cabe citar como ejemplo de este sistema las comisiones que se abonan en el sector bancario por cada movimiento de efectivo, adquisición de hipotecas, compra-venta de bienes, etc, lo cual eleva el coste final de las transacciones, y las mantiene expensas a las fluctuaciones en las tarifas que cobran las entidades bancarias.

Una de las causas [23] de la Gran Crisis Económica Mundial en 2008 se atribuye a los elevados intereses y tasas que imputaban las corporaciones bancarias, lo cual provocó la quiebra económica de muchos consumidores, que veían inviable el abono de deudas, hipotecas, salarios, etc., por los elevados intereses fijados por el sistema financiero.

En esta nueva propuesta, cualquier par de interesados tiene la capacidad de llevar a cabo transacciones de manera directa sin necesidad de registrar los intercambios de capital ante una tercera parte.

La criptografía sustituye el modelo avalista asegurando que los contribuyentes solo pueden asumir transacciones de capital igual o superior al que disponen en su cartera de criptomonedas.

Y de manera adicional, todos los detalles de las adquisiciones, concesiones de préstamos, ventas, etc, son registradas en base a unos acuerdos inmutables gracias a la inalterabilidad de la cadena de bloques.

2.2.3 Problema del doble gasto

El doble gasto es un defecto del sistema monetario a través del cual un usuario puede utilizar más de una vez una misma cantidad de efectivo. Esto representa un fraude y genera desconfianza en el sector bancario, a parte de ocasionar problemas de inflación, pues una misma moneda se pone en circulación numerosas veces.

Los sistemas tradicionales construyen la solución al doble gasto haciendo uso de una autoridad central que observa todas las transacciones.

Por citar un ejemplo, si un usuario *A* desea enviar una cantidad de dinero *X* a *B*, la entidad central comprueba que *A* dispone de un importe ≥ *X*, aumenta el saldo de *B*, y reduce el de *A*.

Esto es visto como una amenaza a la privacidad de los usuarios, a parte del malestar que produce el cobro de comisiones por cada transacción. Por ello, la cadena de bloques implementa un protocolo descentralizado, y basado en criptografía, para asegurar la integridad de las transacciones y evitar el fraude.

El uso de tecnología en lugar de un organismo central motiva a los usuarios a confiar en el sistema.

Al mismo tiempo, la tecnología ofrece una serie de características muy oportunas para

el actual contexto socioeconómico: descentralización, inmutabilidad, seguridad y transparencia.

En los siguientes apartados se irán exponiendo los fundamentos de la cadena de bloques y cómo es capaz de mejorar el modo de funcionamiento de muchos sectores críticos ya existentes en la sociedad.

2.2.4 Descentralización

Almacenando los datos en una red de pares, la *blockchain* elimina el riesgo de ataque que apareja guardar de manera centralizada toda la contabilidad.

Adicionalmente, cada nodo tiene una copia del libro de contabilidad y de todo el estado de la cadena. No existe ningún nodo superior a otro como ocurre en los sistemas cliente-servidor.

La sincronización de los datos se mantiene gracias a un sistema de almacenamiento y replicación masivos. Los nodos mineros validan las transacciones, las añaden al bloque que están construyendo y luego difunden el bloque completo a otros nodos.

Cabe destacar que existen variantes a la *blockchain* inicialmente propuesta para el sistema *Bitcoin*.

Cada implementación puede hacer uso de distintas alternativas en cuanto a los distintos componentes como por ejemplo el algoritmo de consenso, la estructura de datos en la que se almacenan las transacciones, pero se mantiene la característica de descentralización.

2.2.5 Árbol de Merkle

Un árbol de *Merkle* [7] es una estructura de datos en forma de árbol binario en la que cada nodo que no es una hoja está etiquetado con el *hash* de la concatenación de las etiquetas o valores de sus nodos hijo.

Esta forma de almacenar los datos permite asociar un gran número de nodos a un único *hash*, el correspondiente al nodo raíz del árbol. Gracias a esto es posible verificar de forma eficiente y segura grandes cantidades de datos.

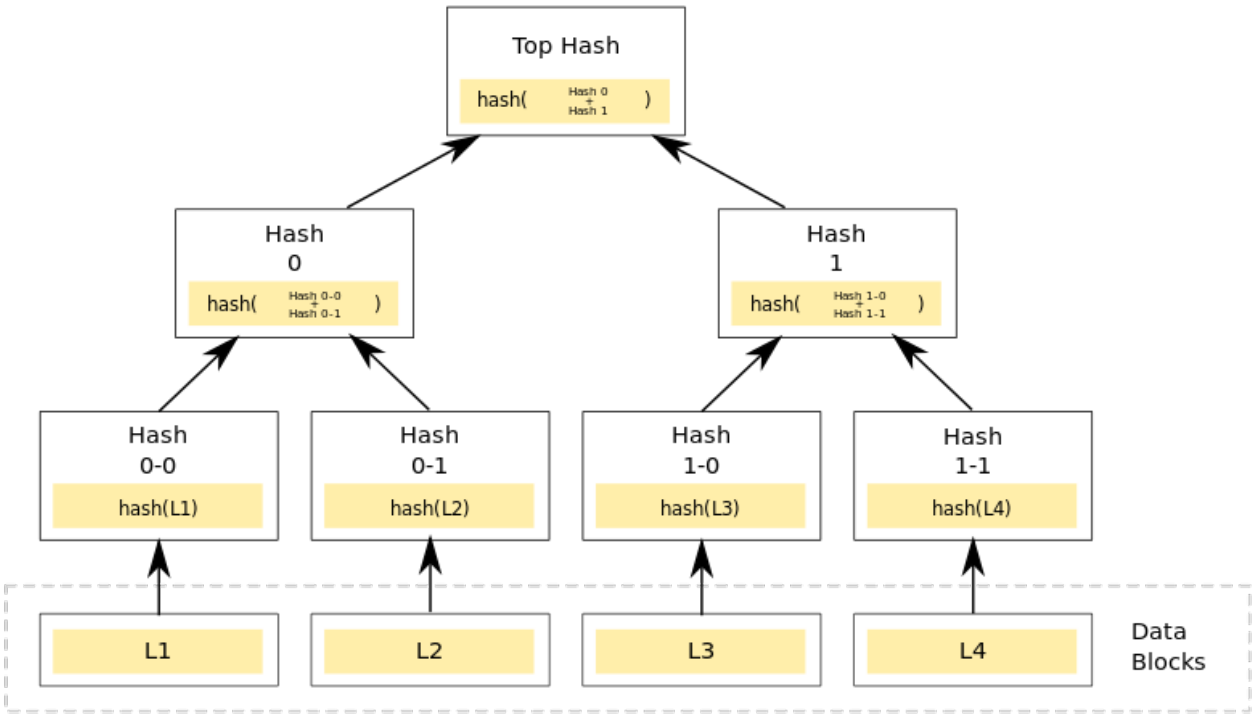

Figura 2.2: Validación de un Árbol de Merkle. Fuente: [25]

Se puede observar en la anterior figura la estructura de un *Árbol de Merkle*, *L1...L4* representa transacciones sin firmar, en el nivel superior los nodos corresponden a la firma digital de estas, y así sucesivamente, a medida que se asciende en el árbol se va concatenando el *hash* de la suma de los nodos inferiores.

2.2.6 Bloques

Un bloque es una estructura de datos en la que se almacenan las transacciones que se desea guardar en la cadena de bloques. Concretamente, cada bloque se compone de dos partes, una cabecera en la que se almacenan metadatos y una lista de transacciones.

Uno de los campos de la cabecera es el *hash* criptográfico del bloque anterior. Gracias a esto, cada bloque se une con el bloque previo, formando una lista enlazada hacia atrás de bloques de información (transacciones), esta característica da lugar al nombre de cadena de bloques.

A nivel más técnico, los bloques son archivos en los que se registran permanentemente los datos relativos a la red *Bitcoin*. Un bloque registra algunas o todas las transacciones más recientes de *Bitcoin* que aún no han entrado en ningún bloque anterior.

Los detalles de la información que se envía a la cadena de bloques y aquella que permanece privada es una decisión del equipo de desarrollo de la aplicación en cuestión.

Se trata de una cláusula que involucra utilizar una cadena de bloques pública, privada o

híbrida, en función de las características del producto y las necesidades de la empresa de publicar o mantener la privacidad de datos confidenciales.

Por lo tanto, un bloque guarda semejanza con una página de un libro de contabilidad o de registro. Cada vez que un bloque se termina, da paso al siguiente bloque en la cadena de bloques. Un bloque es, por tanto, un almacén permanente de registros que, una vez escrito, no puede ser alterado ni eliminado.

El *hash* de cada bloque se calcula sobre la cabecera del mismo. Dicho componente consta de los elementos que siguen:

- En la cabecera de un bloque se almacena la cadena resumen (*hash*) del bloque previo. Si el bloque *n − 1* sufre un cambio, su *hash* variará y no coincidirá con el *hash* almacenado en la cabecera del bloque *n*, esto implicará la actualización del bloque *n* para que los *hashes* coincidan.
    - A su vez, la cabecera del bloque *n* variará y el nuevo *hash* resultante no coincidirá con el *hash* almacenado en la cabecera del bloque *n + 1*, siendo impracticable manipular la cadena de información.
- De igual forma, la cabecera de cada bloque también dispone de un campo denominado raíz *Merkle*. El valor de este campo se establece al valor del nodo raíz del árbol de *Merkle* formado a partir de las transacciones almacenadas en el bloque.
- Incluyendo este valor en la cabecera y teniendo en cuenta el punto anterior, se deduce directamente que de esta forma se consigue la inmutabilidad e integridad de los datos almacenados. Dado que la única operación que se acepta sobre la cadena es la de añadir bloques, las transacciones realizadas son irreversibles.
- El primer bloque creado en cada *blockchain* no contiene en su cabecera el *hash* del bloque previo. Este bloque recibe una denominación específica: *genesis block*.

Otro aspecto que ha de comentarse es que cada bloque de la cadena puede identificarse de dos maneras, a partir de su hash y a partir del *height* (posición que ocupa en la cadena).

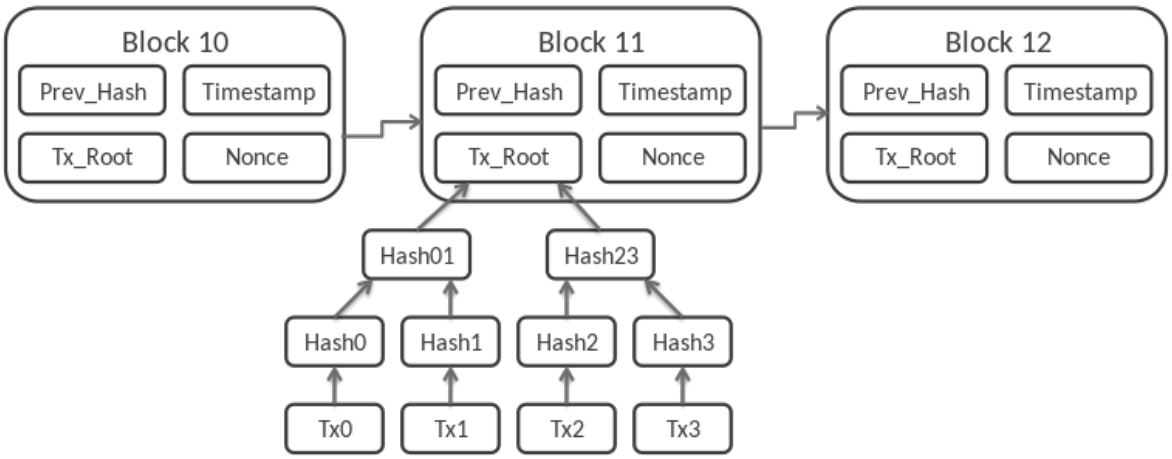

Figura 2.3: Estructura de una cadena de bloques. Fuente: [29]

Se puede observar en la figura anterior la estructura de bloques encadenados, cada bloque contiene un árbol de transacciones firmadas criptográficamente, el *nonce*, y el hash del bloque previo.

Esto permite dar cuenta de la robustez de la tecnología, siendo inmensamente complejo manipular la cadena a favor de un usuario determinado. A modo de analogía, es posible comparar las transacciones bancarias ordinarias con las transacciones en la red *Bitcoin*.

Una cadena de bloques es como un registro de transacciones bancarias, mientras que un bloque puede ser una confirmación de transacción única que un cajero automático imprime después de usar la máquina.

Dentro de la red *blockchain*, los bloques individuales construyen un libro de contabilidad muy parecido al que un cajero automático o un banco registraría sus transacciones. Sin embargo, la *blockchain* registra la cadena de todos sus usuarios en lugar de uno.

Cabecera

La cabecera de un bloque se compone de 6 campos:

- El número de versión del software de la cadena de bloques que se esté utilizando. No tiene una gran relevancia en el proceso de adición de un nuevo bloque, no obstante, un minero con un número de versión determinado puede señalar qué protocolo soporta.
- El *hash* del bloque previo
- El *hash* del nodo hoja del árbol de *Merkle*
- El tiempo en segundos desde el 1 de enero de 1970 a las 00:00 *UTC*
- El objetivo de la dificultad actual
- El campo *Nonce,* equivale al número de transacciones enviado desde una dirección *Bitcoin* particular, y se emplea para evitar el fraude en las transacciones.

Almacén de transacciones

A su vez, la parte principal del bloque contiene todas las transacciones que han sido validadas y agregadas a la cadena pública con éxito.

Cuando un minero construye un bloque, valida las transacciones. Es decir, comprueba que el remitente tiene suficiente saldo para enviar dinero. Gracias a la transparencia de la cadena de bloques, esta información es fácilmente accesible mediante una operación de lectura.

Tal y como se ha expuesto anteriormente, las transacciones que almacena el bloque se encuentran en una estructura de árbol de *Merkle*, el nodo raíz equivale al *hash* de la suma de todas las transacciones anteriores firmadas digitalmente.

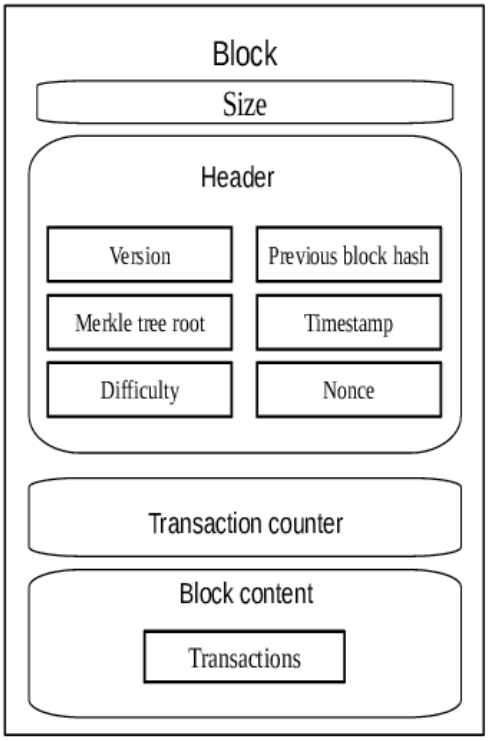

Figura 2.4: Componentes de un bloque de transacciones. Fuente: [30]

En la presente figura se observa con mayor profundidad la nomenclatura completa de un bloque, consta de cuatro componentes, el *Size* indica el número de transacciones que se podrán apilar, el resto de campos está relacionado con el árbol de *Merkle* (*Block content*), el contador de transacción, y la cabecera.

2.2.7 Nonce

El proceso de minería tiene como objetivo resolver el reto de adivinar el *nonce.* El *nonce* es usado para generar un *Block ID* o *hash* de bloque en *Bitcoin* por medio del algoritmo denominado Prueba de Trabajo (*Proof of Work – PoW*).

El *Block ID* representa un hash *SHA-256*. Cuando se da comienzo al proceso de minería para identificar el *nonce,* dicha tarea se lleva a cabo teniendo en consideración una serie de parámetros que son otorgados por la red *Bitcoin*. Uno de estos parámetros es el llamado *target* o *hash target.*

El *target* es un número muy grande (256 bits de ancho) que comparten todos los nodos de *Bitcoin*. Este *target* es el que indica el límite máximo del valor decimal de un *Block ID* para un determinado momento. Es decir, el *target* indica el valor que debe tener un *hash* de bloque para que este sea aceptado por la red.

Cuanto menor sea el *target*, más difícil será generar un *hash* de bloque válido. El trabajo de minería se reduce a generar un *hash* de bloque o *Block ID* que sea menor que el *target* que ha propuesto la red *Bitcoin*.

Para lograr esto, el nodo minero debe tomar los siguientes datos: versión, *hash* del bloque anterior, el *Merkle Root*, la marca de tiempo del bloque y la dificultad de la red. A todos estos datos se les suma el *nonce* generado por el minero.

El minero tomará todos estos datos y entonces dará inicio al proceso de generar el *hash* de bloque o *Block ID*.

Lo importante de este proceso es que el valor decimal del *hash* generado por el minero debe ser menor que el *target* que ha propuesto la red. En caso contrario, el minero debe reiniciar su trabajo.

Es decir, el ciclo de tomar todos los datos mencionados anteriormente, generar un nuevo *nonce* y calcular el *hash* del bloque, se repetirá hasta que el minero haya encontrado una solución al problema planteado.

Esto lo lleva a calcular en cada iteración un nuevo *nonce* y con ello un nuevo *hash* de bloque que deberá verificar. La dificultad de la tarea reside en que puede acarrear el cálculo de miles de millones de *nonce* y hashes distintos hasta dar con la combinación correcta.

Una vez localizado, es posible generar el bloque y ello permite acceder a la recompensa y a las comisiones asociadas a la minería de dicho bloque.

2.2.8 Problema del consenso

Tal y como se ha expuesto previamente, la tecnología de bloques se establece sobre redes distribuidas *peer-to-peer* [8]. En computación distribuida surge un concepto denominado problema del consenso.

Normalmente, resulta necesario que los numerosos procesos que se ejecutan de forma distribuida entre los nodos de la red lleguen a un acuerdo sobre algo en específico. En el caso de la *blockchain*, los nodos no dependen de una autoridad central que tome las decisiones por lo que han de convenir acuerdos de manera autónoma.

La capacidad de alcanzar el consenso entre los nodos de la red sin necesidad de un organismo regulador central que administre la toma de decisiones es una de las características más importantes de la tecnología.

Este es el motivo por el que se trata de una tecnología totalmente descentralizada. A continuación, se presentará cómo se consigue abordar este problema y para qué decisiones se ha de lograr el consenso entre los nodos de la red.

Para la resolución de este problema se aplican los algoritmos de consenso. Este tipo de mecanismos aseguran que los usuarios que participan en la *blockchain* (nodos de la red) alcancen un acuerdo sobre el estado de la cadena y la validez de los nuevos bloques que se vayan añadiendo.

Cada nodo de la red mantiene una copia local de la cadena de bloques, por lo que a

medida que aparecen nuevos bloques las copias han de ir actualizándose. Antes de extender la copia local con un bloque, cada nodo ha de estar seguro de que se trata de un bloque que no contiene una operación fraudulenta.

Específicamente, el algoritmo de consenso utilizado para la primera aplicación real de la *blockchain* fue el denominado *Proof-of-Work* [9] (*PoW*). En este algoritmo, los usuarios de la red *blockchain* compiten en el proceso de creación de bloques proporcionando los recursos computacionales dedicados de su máquina.

Una vez llevado a cabo el esfuerzo computacional requerido para la creación de un nuevo bloque, los contenidos del bloque no pueden alterarse sin antes volver a encontrar un valor adecuado para el campo *nonce*.

A medida que se unen bloques, el esfuerzo requerido para modificar un bloque aumenta de forma directamente proporcional al coste computacional necesario para volver a minar dicho bloque y ulteriores.

Dado que todos estos procesos tienen lugar en un entorno distribuido, puede darse la situación en la que dos nodos honestos minan un bloque de manera simultánea. Ambos bloques se transmitirían por la red y serían aceptados por distintos nodos.

Se daría la indeseable situación en la que existen en la red diferentes versiones válidas de la cadena de bloques. Sin embargo, el algoritmo *Proof-of-Work* también implementa una solución para este problema.

La única versión válida de la cadena de bloques es aquella de mayor longitud, o dicho de otro modo, aquella en la que se ha invertido más esfuerzo computacional (pruebas de trabajo).

Para la validación de un nuevo bloque, cada nodo de la red ha de realizar dos verificaciones:

1. El valor del campo *hash* previo de la cabecera del nuevo bloque es igual al *hash* del último bloque de su cadena. En caso contrario, la anomalía puede deberse a dos causas, en primer lugar, el nodo puede tener una copia desactualizada de la cadena, o el bloque nuevo puede no ser válido.

2. Aplicando la función *hash* sobre la cabecera del nuevo bloque se obtiene un *hash* que empieza con el número exigido de ceros. El trabajo computacional medio requerido para encontrar el valor del campo *nonce* aumenta exponencialmente con la cantidad de ceros necesarios al comienzo del *hash* resultante.

3. Por este motivo a esta suma de ceros se le denomina dificultad del algoritmo o dificultad de minado, y su valor varía en el tiempo con el propósito de regular la velocidad a la que se pueden crear nuevos bloques.

Existen numerosas alternativas al algoritmo de consenso *Proof of Work* (*PoW*):

- *Proof of Stake [10]* (*PoS*): Resuelve el problema de escalabilidad que sufren las redes *blockchain* que operan con *PoW*. El criterio introducido se basa en que la probabilidad de seleccionar un nodo como creador de un nuevo bloque está estrechamente relacionada a la riqueza que dicho nodo haya acumulado.
- *Proof of Authority [11]*: Se trata de una versión diseñada para redes privadas que se apoyan sobre contratos inteligentes. Su modo de operación consiste en el uso

de las identidades reales de los usuarios de cada nodo y sólo permite que un conjunto de estos actúen como validadores dentro de la red. Cada validador suministra su identidad real y méritos en la minería como prueba de transparencia.

2.2.9 Transacciones

En el presente epígrafe se introduce el tercer elemento vertebrador de la cadena de bloques, las transacciones.

Una transacción es la acción mediante la cual se lleva a cabo una modificación de la *blockchain* en virtud de un contrato inteligente, o una transferencia de criptomonedas entre clientes a más bajo nivel.

Es de señalar que en las operaciones de creación y verificación de las transacciones resultan necesarias las claves asociadas a cada usuario. Además, a continuación se presenta una lista con los pasos que componen el proceso:

1. El usuario *A* (emisor) quiere llevar a cabo una transacción con el usuario *B* (receptor), que puede consistir en el envío de 50 *Bitcoins* de *A* a *B*.

2. Los datos de la transacción, la clave pública del receptor y la clave privada del emisor se suministran como entrada a una función de firma. La salida de dicha función es la propia firma o prueba de autenticidad de la transacción.

3. La firma de la transacción, en adición de la clave pública del emisor y los datos de la transacción son transmitidos a través de la red. Todos los nodos responsables de crear nuevos bloques (mineros) y el receptor efectúan la comprobación de la validez de la nueva transacción, verifican que el emisor es quien dice ser y que cumple los requisitos necesarios para poder asumir la ejecución de la transacción (se elimina el problema del doble gasto).

4. Los requisitos contemplan verificar que el saldo de A es igual o superior a la cantidad transferida a *B*.

5. Para ello, utilizan los 3 elementos recibidos, es decir, la clave pública del emisor, la firma de la transacción y los datos de cabecera de la transacción. A través del reconocimiento de la firma son capaces de determinar si la transacción es válida.

6. En caso de que la transacción sea válida, los mineros la añaden al bloque en construcción. Cuando se termine de construir el bloque este es transmitido a la red.

7. Los distintos nodos de la red aceptan el bloque sí y solo sí todas las transacciones que contienen son válidas y pueden ejecutarse.

8. Los nodos que han comprobado la validez del bloque lo añaden a su copia local de la cadena y continúan trabajando en construir el siguiente, utilizando como *hash* previo el *hash* del nuevo bloque.

9. En este instante la transacción ya se ha ejecutado.

Mediante la utilización del par de claves de cada usuario es posible que los nodos comprueben que las transacciones se han iniciado por la persona correcta, de manera legal. El sistema de firma basado en criptografía asimétrica permite garantizar la

seguridad de las transacciones.

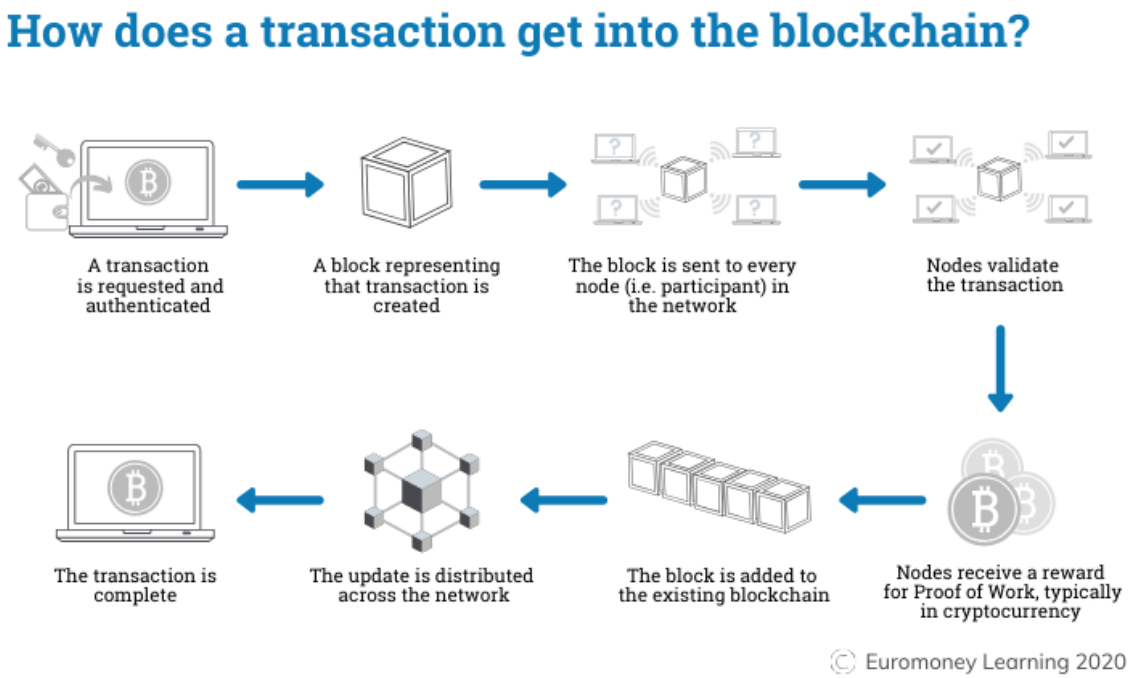

Figura 2.5: Proceso de Validación de una transacción. Fuente: [31]

En la anterior figura se puede observar el proceso de validación de una transacción, primero se valida y después se replica al resto de nodos para que actualicen su copia del libro de contabilidad de la *blockchain*.

2.2.10 Tipos de Blockchain

En la actualidad es posible definir tres categorías diferentes de la cadena de bloques. En la siguiente lista se describen sus particularidades:

- Públicas: las cadenas de bloques públicas son las más frecuentes y las más ampliamente utilizadas.
  - Son de código abierto y permiten que cualquier persona se una a la red (ordenadores conectados a Internet), ya sea como usuario básico o como minero, es decir, cualquier individuo es capaz de realizar transacciones o de ofrecer los recursos computacionales de su nodo para participar en el minado de nuevos bloques.
  - Habitualmente incluyen algún tipo de incentivo en forma de criptodivisa para

recompensar a los nodos mineros.

- Privadas o permisionadas: la aplicación de este tipo de *blockchains* es más común en ámbitos corporativos puesto que únicamente se permite la unión a la red a organizaciones conocidas, de manera que se forma una especie de red de negocios privada sólo para miembros.
    - Esto implica que la información almacenada en la cadena de bloques debe ser confidencial y accesible solo para usuarios autorizados.
    - Este tipo de cadenas, a diferencia de las públicas, se apoyan en la identidad de los miembros para confirmar los privilegios de acceso a la información.
    - Consecuentemente, los participantes en la red saben exactamente con quién están tratando. La verificación de las transacciones sólo puede llevarse a cabo por usuarios identificados previamente, por ello, en este tipo de redes los algoritmos de consenso como el *Proof-of-Authority* son los más adecuados.
- Híbridas: las cadenas de bloques híbridas combinan los beneficios de privacidad de los que disponen las cadenas permisionadas con las ventajas de seguridad y transparencia de las cadenas públicas.

2.2.11 Desafíos y limitaciones

Tal y como se ha comentado en el Capítulo 1 de la memoria, la tecnología de bloques está en una fase de desarrollo e implantación en la industria muy pausada. Por ello, existen desafíos y limitaciones que han de abordarse con el propósito de incrementar las posibilidades de aplicación de la misma. A continuación, se enumeran algunos de estos desafíos y limitaciones:

- Escalabilidad: Las cadenas que utilizan el algoritmo de consenso *Proof-of-Work* presentan problemas de ineficiencia por el gran esfuerzo computacional que se desperdicia.
    - A medida que se adhieren nuevos nodos a la red, la velocidad de transacción disminuye, resultando en una lentitud creciente a medida que se adhieren más equipos a la red. Se ha presentado una nueva versión del algoritmo *Proof of Authority*, que se encuentra aún en una fase de pruebas.
    - Almacenamiento: A medida que transcurre el tiempo, y debido a la popularidad que ha alcanzado la tecnología, el tamaño de la cadena de bloques crece.
        - Por ello, existe el peligro de que la cadena llegue a ocupar tanto espacio que los nodos convencionales (ordenadores personales) no sean capaces de almacenar la copia de la cadena, esto ocasionaría la pérdida de un gran número de nodos.

2.3 Ethereum

Ethereum es una plataforma que ofrece una *blockchain* en la que se incorpora *Solidity*, un lenguaje de programación orientado a objetos diseñado para la creación de contratos inteligentes.

El uso de este tipo de contratos permite a los desarrolladores implementar lo que se conoce como aplicaciones descentralizadas o *dapps*, en las que es posible definir funciones que ejecutan los términos de un acuerdo y formatos de transacción personalizados para el ámbito de aplicación específico.

Esto significa que la tecnología extiende el tipo de transacciones más allá del intercambio de criptomonedas entre usuarios, permitiendo programar la operativa de muchos sectores en líneas de código, e inmutabilizar todos los cambios en la cadena de bloques, ofreciendo más seguridad y certidumbre para el consumidor final frente a las manipulaciones que puedan hacer las partes intermediarias en los procesos físicos que requieren intermediación legal.

Ethereum mantiene la esencia de la tecnología de cadena de bloques citada en las secciones precendentes, con algunas diferenciaciones importantes en cuanto a funcionalidad nueva como los contratos inteligentes, algoritmo de consenso personalizado, el *Ether* como moneda de pago de las transacciones, y herramientas adicionales que se introducirán más adelante.

Extendiendo las ventajas que aporta la tecnología de cadena de bloques con la versatilidad y el amplio abanico de posibilidades que se abre con la invención de los contratos inteligentes se permitió que cualquier par de interesados pudieran firmar acuerdos sin necesidad de intermediarios, quedando registrados de manera inmutable los términos de los mismos y pasando a ser irreversibles y transparentes.

2.3.1 Historia

El origen de los *smart contracts* se remonta a 1997, año en el que Nick Szabo publica el artículo *Formalizing and Securing Relationships on Public Networks [12]*, en el que se presenta el concepto de *smart contract* como protocolo de transacción que ejecuta los términos de un contrato.

Szabo, influido por el informe de David Chaum [43], respaldaba la creación de contratos a través de una lógica transparente, y por medio de una verificación a través de protocolos criptográficos, resaltaba su capacidad para constituir una mejora importante sobre los contratos legales tradicionales.

Vitalik Buterin y Gavin Wood [13] crean *Ethereum* en 2014, aunando los conceptos de *smart contract* y *blockchain* en una única plataforma. El lanzamiento de esta tecnología supuso un momento transformador de la cadena de bloques.

Buterin argumentó al grupo desarrolladores de Bitcoin que "*Bitcoin y la tecnología blockchain podrían beneficiarse de otras aplicaciones y que necesitaba un lenguaje más robusto para el desarrollo de aplicaciones que pudiera llevar a adjuntar activos del mundo real, como acciones y propiedades, a la blockchain*".

2.3.2 Contratos inteligentes

Un contrato inteligente hace referencia a un contrato que se ejecuta por sí mismo sin intermediación de terceros y se escribe como un programa informático en lugar de utilizar un documento impreso con lenguaje legal.

Al crear un *smart contract*, el desarrollador define su comportamiento y los términos del acuerdo directamente en líneas de código.

Una vez que se incluye el contrato en la *blockchain* no puede ser modificado, por este motivo se puede asegurar que la ejecución del contrato se llevará a cabo tal y como fue programada.

Los contratos inteligentes pueden ser vistos como programas independientes almacenados en la *blockchain* que llevarán a cabo transacciones, operaciones u acciones de manera autónoma si ciertas condiciones se cumplen.

El desarrollador es el que especifica dichas acciones y condiciones dependiendo del ámbito y propósito de la aplicación que esté implementando.

Uno de los aspectos más relevantes de *Ethereum* reside en la facilidad con la que es posible crear aplicaciones descentralizadas que se aprovechen de todas las ventajas que ofrece la cadena de bloques. Al crear un *smart contract*, el desarrollador define su comportamiento y los términos del acuerdo directamente en líneas de código.

Antes de dar inicio a la explicación de la solución desarrollada en el siguiente capítulo conviene abordar de manera simplificada algunos de los componentes y particularidades más importantes de *Ethereum*.

2.3.3 Ether

El *Ether* (*ETH*) es la criptomoneda utilizada por *Ethereum* como recompensa a los mineros en un sistema de prueba de trabajo por añadir bloques a la cadena de bloques. Es la única moneda aceptada en el pago de las tasas de transacción, que también se destinan a los mineros.

La recompensa por bloque junto con las tasas de transacción proporcionan el incentivo a los mineros para mantener el crecimiento de la cadena de bloques, es decir, para seguir procesando nuevas transacciones.

Por lo tanto, el *Ether* es fundamental para el funcionamiento de la red. Cada cuenta de *Ethereum* tiene un saldo de *Ether* y puede enviar *Ether* a cualquier otra cuenta. La subunidad más pequeña de *Ether* se conoce como *Wei* y equivale a 10^{-18} *Ethers*.

2.3.4 Solidity

Solidity [44] es un lenguaje de programación orientado a objetos que se utiliza para programar contratos inteligentes, principalmente, en la plataforma Ethereum. Ha sido desarrollado por un equipo de ingenieros liderado por Christian Reitwiessner y Alex Beregszaszi.

Como característica principal, *Solidity* tiene un tipeado estático, se verifica la tipología de las variables en la fase de compilación, se ejecuta en la máquina virtual de *Ethereum*, y permite herencia múltiple.

También presenta una interfaz de aplicación binaria (*ABI*), un protocolo que especifica las llamadas a funciones en un contrato y recuperación de datos.

Un ejemplo sencillo de *Solidity* consiste en un contrato inteligente de telefonía móvil, entre una compañía, y un nuevo cliente.

```
pragma solidity >=0.4.22 <0.7.0;

contract CellSubscription {
        uint256 monthlyCost;

    constructor(uint256 cost) public {
        monthlyCost = cost;
    }
}
```

El contrato se crea con la palabra reservada *contract.* Se puede notar la semejanza entre *Solidity* y otros lenguajes de programación como *C* o *C++*, lo cual acerca mucho más la tecnología a los desarolladores de aplicaciones tradicionales.

El tipo *uint256* sirve para representar un número entero sin signo de 256 bits de longitud. Se utiliza para guardar el coste mensual de la suscripción.

En la primera línea se declara la versión de *solidity* que se está utilizando. Para abonar las cuotas del contrato, es necesario definir la lógica en una función aislada. La moneda de pago será el *Ether*.

```
function makePayment() payable public {

}
```

Toda la lógica necesaria se declara mediante la sentencia *payable.* El modificador de la función *payable* indica al contrato que almacene automáticamente cualquier *Ether* que se envíe internamente al llamar a la función *makePayment*, esta información se utilizará más adelante para determinar si el abonado ha pagado suficiente *Ether* a partir en un plazo de pago determinado.

A continuación, se implementa una segunda función que sirve para comprobar que el cliente ha abonado las cantidades pactadas en una fecha determinada. Esto es sencillo dado que únicamente requiere multiplicar la cuota mensual por el número de meses que han transcurrido, y comparar este dato con la cantidad real de dinero recibida.

```
function isBalanceCurrent(uint256 monthsElapsed) public view
returns (bool) {
        return monthlyCost * monthsElapsed >= address(this).balance;
    }
```

View indica que la función implementa un código que solo realiza operaciones de lectura en la *blockchain*, por lo que no consume *Ether*. La sentencia *address(this).balance* retorna el dinero abonado por el cliente, cuya dirección de *Ethereum* se recupera invocando la función *address* sobre la instancia actual del contrato.

Finalmente, la función retorna un booleano que indica si la balanza está equilibrada a favor de la empresa (*true*), o se encuentra desajustada debido a un impago del cliente (*false*).

En este punto de desarrollo del contrato, la pieza importante que resta explicar consiste en una función que implementa la lógica para retirar dinero de la cuenta del cliente y transferirlo a la cuenta de la compañía.

```
function withdrawBalance() public {
        msg.sender.transfer(address(this).balance);
    }
```

msg.sender hace referencia a la dirección que ha invocado la función, que corresponde al proveedor de telefonía. *transfer* lleva a cabo un movimiento de dinero a favor de *msg.sender.*

2.3.5 Cuentas

Hay dos tipos de cuentas en *Ethereum*: las cuentas de usuario y los contratos. Ambos tipos tienen un saldo de *Ether*, pueden enviar *Ethers* a cualquier cuenta, llamar a cualquier función pública de un contrato o crear uno nuevo, y se identifican en la *blockchain* por su dirección.

Las cuentas de usuario son el único tipo que pueden crear transacciones. Para que una transacción sea válida, debe estar firmada con la clave privada de la cuenta remitente, una clave hexadecimal de 64 caracteres que sólo debe conocer el propietario de la cuenta.

El algoritmo de firma utilizado es *ECDSA (Ellipitic Curve Digital Signature Algorithm)*. Es importante destacar que este algoritmo permite deducir la dirección del firmante a partir de la firma sin conocer la clave privada.

Los contratos son el único tipo de cuenta que lleva asociado un código (un conjunto de funciones y declaraciones de variables) y un almacenamiento de contratos (los valores de las variables en cada momento). Una función de contrato puede tomar argumentos y puede tener valores de retorno.

Dentro del cuerpo de una función, además de las declaraciones de flujo de control, el código de un contrato puede incluir instrucciones para: enviar *Ethers* como se ha observado en el ejemplo de contrato de telefonía, leer y escribir en su almacenamiento.

También permite crear un almacenamiento temporal (tipo *memory*) que se extingue al final de la función, realizar operaciones aritméticas y de *hashing*, llamar a sus propias funciones, llamar a funciones públicas de otros contratos, crear nuevos contratos y consultar información sobre la transacción actual o la *blockchain*.

Las direcciones de *Ethereum* están compuestas por el prefijo hexadecimal *0x*, concatenado con los 20 bytes más a la derecha del *hash Keccak-256* de la clave pública *ECDSA*.

En hexadecimal, 2 dígitos representan un byte, lo que significa que las direcciones contienen 40 dígitos hexadecimales, por ejemplo, *0xb794f5ea0ba39494ce839613fffba74279579268.*

Las direcciones de los contratos tienen el mismo formato, pero se determinan por el remitente y el *nonce* de la transacción de creación.

2.3.6 Ethereum Virtual Machine

La máquina virtual de *Ethereum* (*EVM*) es el entorno de ejecución de las transacciones en *Ethereum*.

Se trata de una pila de registros de 256 bits que está aislada de los demás archivos y procesos del nodo para garantizar que, para un estado y una transacción determinados, todos los nodos produzcan el mismo estado posterior a la transacción, lo que permite el consenso en la red.

2.3.7 Gas

El G*as* es una unidad de cuenta dentro del *EVM* que se utiliza en el cálculo de una tarifa de transacción, que es la cantidad de *Ether* que el remitente de una transacción debe pagar al minero que valida la transacción en la cadena de bloques.

Cada tipo de operación que puede ser realizada por el *EVM* está codificada con un determinado coste de gas, que pretende ser aproximadamente proporcional a la cantidad de recursos (computación y almacenamiento) que un nodo debe gastar para realizar esa operación.

Al crear una transacción, el remitente debe especificar un límite de gas y un precio de gas. El límite de gas es la cantidad máxima de gas que el emisor está dispuesto a utilizar en la transacción, y el precio del gas es la cantidad de *Ether* que el emisor desea pagar al minero por unidad de gas utilizada.

2.4 Criptografía homomórfica

El cifrado homomórfico [17] es una técnica que permite realizar operaciones sobre los datos cifrados y obtener resultados, también cifrados, equivalentes a las operaciones realizadas directamente sobre la información original.

La adopción extendida de este tipo de técnicas se debe al consumo elevado de recursos computacionales en las operaciones sobre los datos en claro, todo ello unido al riesgo de fuga de información que apareja el descifrado de la misma.

La utilidad de esta técnica en el problema expuesto radica en la necesidad de cifrar el voto, y asegurar la confidencialidad e integridad del mismo en todas las operaciones de recuento e identificación a lo largo del proceso electoral.

La idea básica del funcionamiento de estos sistemas se apoya en el concepto de homomorfismo matemático, el cual establece lo siguiente:

Dados dos grupos *(A, .)* y *(B, *)* y una función *F: A->B* que toma elementos del conjunto *A* y devuelve elementos del conjunto *B*. *F* es un homomorfismo si y solo si *F(x . y) = F(x)* F(y)* para cualquier par de elementos "*x*" e "*y*" pertenecientes al conjunto *A*.

Seguidamente, asociando este concepto con criptografía, el conjunto *A* puede interpretarse como un texto plano, el conjunto *B* como el texto cifrado, '.' como las operaciones que se pueden realizar sobre el texto plano, '*' como las operaciones posibles sobre el dato cifrado y finalmente *F* como la función de cifrado.

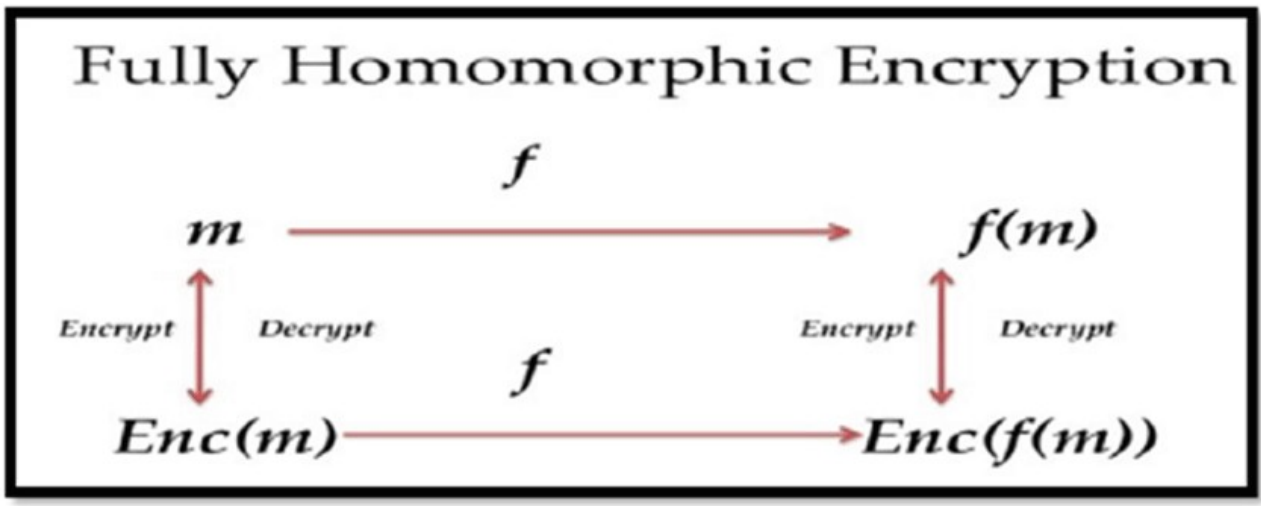

Figura 2.6: Esquema de un sistema de criptografía homomórfica completo. Fuente: [34]

Otro ejemplo sencillo que se puede enumerar se suele dar de manera frecuente en servicios de nube digital. Muchas corporaciones ven inviable almacenar sus datos localmente, por lo que contratan un proveedor de nube.

Habitualmente el almacenamiento en nube implica una operación de cifrado de la información. Por cuestiones de simplicidad del ejemplo para el lector, supóngase que el negocio *A* almacena todas sus ventas de manera remota, y para cifrarlas utiliza un esquema de cifrado homomórfico, que consiste en multiplicar los datos por dos.

Así, si en dos días consecutivos el establecimiento generó ingresos por valor de 1500 y 1700 euros respectivamente, la versión cifrada que se envía a la nube es 3000 y 3400.

Más tarde, el negocio necesita abonar tributos fiscales y solicita al proveedor digital los ingresos totales. Este realiza la suma, pero desconoce el importe real, porque la versión enviada por *A* fue cifrada.

Finalmente, el establecimiento descifra con su clave privada la suma encriptada, y obtiene los ingresos totales.

2.4.1 Tipos de criptografía homomórfica

Cifrado parcialmente homomórfico (PHE)

Es un sistema de cifrado que es homomórfico únicamente en la suma o en la multiplicación. Solo permite una de las dos operaciones sobre los datos cifrados.

Se trata de un esquema de cifrado que se utiliza en algoritmos de uso extendido como el *RSA*, que implementa cifrado homomórfico [39] mediante operaciones de multiplicación de los datos.

Un ejemplo de este tipo de criptosistemas es el de *Paillier* [41]. Se basa en el problema de que el cálculo de las clases de los *n* residuos es computacionalmente difícil. La

naturaleza del algoritmo permite que las operaciones de suma homomórfica produzcan la respuesta actual una vez descifrada.

Para cifrar un mensaje, se utiliza el mensaje como exponente de *g*, luego se eleva un valor aleatorio al otro valor de la clave pública *n*.

Cifrado algo homomórfico (SHE)

A diferencia del *PHE,* permite tanto la suma como la multiplicación en un conjunto de datos dado por un número limitado de veces.

Las operaciones que se realizan son representadas como circuitos aritméticos. *SHE* puede evaluar dos tipos de puertas lógicas, pero sólo para un subconjunto de circuitos.

Una de las ventajas del *PHE* y el S*HE* sobre el *FHE* consiste en la eficiencia. Al aplicar sólo una función aritmética sobre una parte de los datos, la computación se lleva a cabo más rápido.

Cifrado completamente homomórfico (FHE)

Se trata de un cifrado que soporta tanto la suma como la multiplicación por un número ilimitado de veces. La importancia de este tipo de cifrado reside en que con la suma y la multiplicación es posible construir los circuitos para realizar cualquier cómputo.

El *FHE* es un sistema de criptografía basado en el algoritmo de *Ring con Errores,* con una complejidad de tipo *NP-Hard,* lo cual confiere mayor robustez al *FHE* frente a los algoritmos basados en factorización como el *RSA.*

La desventaja de este tipo de cifrado radica en el rendimiento. Combinar [40] varias operaciones de cifrado sobre los datos implica un alto coste computacional, por lo que no es viable en sistemas de almacenamiento masivo.

Adicionalmente, en un sistema con múltiples usuarios, cifrar y descifrar datos con una rapidez suficiente sería inviable.

Capítulo 3
Desarrollo de la solución propuesta

En este capítulo de la Memoria se exponen las tecnologías seleccionadas para llevar a cabo el desarrollo de la aplicación web. A su vez, se explican y presentan los detalles de implementación y particularidades de dicha aplicación.

3.1 Selección de tecnologías

A lo largo de esta sección se expondrán de manera resumida las tecnologías seleccionadas, el porqué de dicha elección y una explicación del papel que juegan en el conjunto de la aplicación web desarrollada.

3.1.1 React y Material-UI

Se creyó conveniente utilizar *React* [18] como librería *JavaScript* de desarrollo de interfaces gráficas de usuario. Está basada en componentes encapsulados con estado propio y permite la creación de interfaces de usuario interactivas de forma sencilla.

El hecho de que la librería esté escrita en Javascript facilita la interacción entre las partes gráfica (apariencia visual) y lógica de la aplicación.

Mediante la declaración de componentes se favorece la reutilización de partes de la interfaz gráfica, permitiendo diseñar vistas simples para cada estado de la aplicación.

React se encarga de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos han sufrido cambios. Al crear las vistas de la aplicación de manera declarativa el código se vuelve más predecible y fácil de depurar.

Por último cabe destacar que gracias a la naturaleza basada en componentes de *React* existen numerosas librerías de componentes gráficos ya preparados.

Tal es el caso de *Material-UI [19]*, que ofrece numerosos componentes ya preparados para agilizar la construcción de interfaces gráficas siguiendo la normativa de diseño *Material Design*.

3.1.2 Truffle y Ganache

Truffle [20] y *Ganache* [21] forman parte de lo que se conoce como *Truffle Suite*, un conjunto de herramientas que facilitan el desarrollo de aplicaciones descentralizadas.

En concreto, *Truffle* proporciona un entorno de desarrollo y marco de pruebas para *blockchain* utilizando la *Ethereum Virtual Machine*. Permite la compilación, depuración y el despliegue de contratos inteligentes por línea de comandos o a través de una consola interactiva de manera sencilla.

Por otro lado, *Ganache* ofrece la posibilidad de desplegar una *blockchain* local de prueba para poder comprobar el funcionamiento de las aplicaciones descentralizadas en desarrollo.

3.1.3 Firebase

Firebase [36] es una plataforma para el desarrollo de aplicaciones web y móviles. Permite equipar servidores *backend* gratuitos para alojar las aplicaciones, además de disponer un conjunto de herramientas como analíticas, base de datos, procesamiento de datos en tiempo real, aprendizaje automático, almacenamiento, etc, para que los desarrolladores puedan analizar el flujo de tráfico que reciben.

En el presente proyecto se utiliza para crear un servidor web e integrar un servicio de base de datos, almacenar los usuarios de la aplicación, crear distintos perfiles de usuarios: votantes, candidatos y administradores y ejecutar peticiones para recuperar datos de los distintos perfiles en el proceso de votación.

3.1.4 Metamask

Metamask [22] es una extensión para el navegador que permite la ejecución de aplicaciones descentralizadas sin necesidad de un nodo *Ethereum*. En concreto se conecta a un nodo de la red externo denominado *INFURA* y ejecuta los *smart contracts* en dicho nodo.

Da la posibilidad de crear cuentas de usuario asociadas a la *blockchain* a la que se conecte, almacenando su clave privada de forma segura y permitiendo a dichas cuentas recibir y enviar transacciones y *Ether* a las aplicaciones aprobadas.

3.2 Aplicación desarrollada

La aplicación creada para el desarrollo de este Trabajo Fin de Máster tiene como nombre *BlockVote*. En esta sección se explican las particularidades de su implementación.

Figura 3.1: Logo de BlockVote.

3.2.1 Particularidades

La principal particularidad de *BlockVote* reside en el hecho de que es totalmente descentralizada, es decir, no se apoya en ningún tipo de servicio y es posible utilizarla únicamente con un navegador y la *blockchain* desplegada en local. En las siguientes secciones de este capítulo se entrará en detalle.

El deseo de mantener el carácter descentralizado junto a la necesidad de restringir el acceso a un sistema de votación supusieron los principales motivos por los que se decidió que el funcionamiento de *BlockVote* requiriera una *blockchain* privada.

Para desplegar la aplicación y poder llevar a cabo pruebas y observar los resultados del desarrollo fue necesario realizar los pasos que a continuación se enumeran:

1. Desplegar una *blockchain* de prueba con *Ganache*. Al desplegar la cadena de bloques, *Ganache* provee un número de cuentas de prueba con *Ether* ficticio. En el caso de la aplicación desarrollada para este trabajo, el *Ether* es necesario para poder abonar los gastos asociados a la ejecución de las transacciones de los contratos inteligentes (*Gas*).

2. Configurar un dominio en *Firebase*, y una base de datos *NoSQL* en tiempo real para gestionar el servicio de autenticación de usuarios.

3. Configurar *Metamask* desde el ordenador para hacer uso de la *blockchain* privada y poder acceder a las cuentas de usuario de prueba que provee *Ganache*.

4. Compilar y desplegar los contratos inteligentes haciendo uso de *Truffle*.

Una vez seguidos estos pasos, la aplicación es totalmente funcional. Es relevante aclarar que la gestión de cuentas de *BlockVote* se delega a *Metamask* que, como se ha expuesto con anterioridad, es capaz de almacenar de forma segura y distribuida las claves privadas de las cuentas de usuarios de la *blockchain*.

Al iniciar la aplicación por primera vez habrá que dar permisos a *Metamask* para que interactúe con ella. El usuario actual de la aplicación queda determinado por la cuenta seleccionada en *Metamask*.

Cada vez que el usuario realiza una acción que deriva en la invocación de una función de un *smart contract* por medio de una transacción (modificación de datos) *Metamask* abrirá una pequeña ventana para confirmar dicha transacción.

Al desplegar la aplicación se crea una cuenta especial única, denominada cuenta del administrador. Ésta cuenta es importante pues es la única cuenta capaz de habilitar otras cuentas.

Cuando se desea iniciar una elección a través de la aplicación, un usuario con cuenta de administrador inscribe los candidatos y votantes autorizados a participar, y define la fecha de inicio y fin de la votación.

Otra de las particularidades más importantes de *BlockVote* es su adaptabilidad, se ha procurado conseguir que cualquier acción del usuario que suponga un cambio significativo (cambio de cuenta en *Metamask*, creación de una nueva elección , ...) produzca un nuevo renderizado de los elementos de la interfaz gráfica correspondientes. Esto se ha conseguido en gran medida gracias a la facilidad que otorga React para ello.

Cuando se inicia la votación, los usuarios con rol de votante reciben invitaciones por correo electrónico con un código de seguridad que caduca después de unas horas. Acceden al evento, registran su voto, y pueden seguir todas las interacciones de la cadena de bloques relacionadas con la elección.

Una vez finalizado el plazo, se pasa a declarar los candidatos ganadores. Esta es una de las ventajas de la descentralización, los usuarios finales pueden ver todas las transacciones de la votación en directo, cuando se registra un nuevo voto, la fórmula para el recuento de votos, el número de votantes, etc.

3.2.2 Contratos desarrollados

En esta sección se exponen los distintos contratos inteligentes implementados para el desarrollo de *BlockVote*. Para cada uno se comentarán y mostrarán los métodos y atributos más relevantes, junto con su funcionamiento u objetivo.

A pesar de que no se profundice totalmente en los detalles de implementación de cada método por cuestiones de longitud del presente trabajo, se ha de destacar que se han utilizado cláusulas *require* de *Solidity* con el propósito de garantizar el correcto funcionamiento de los contratos inteligentes, cotejando previamente los parámetros recibidos.

Election

Es el contrato vertebrador de la aplicación, contiene los métodos para crear una elección, crear candidatos, registrar votantes, comenzar una elección, la lógica asociada al proceso de votación, y un método para cerrar la votación de manera automática una vez finaliza el plazo.

Finalmente, en él también se implementa la funcionalidad para el recuento de votos, agregación de administradores, almacenamiento de parámetros de la elección a fin de enviar información de transparencia a la cadena de bloques.

FirstPastThePost

El segundo contrato implementado calcula el ganador de la elección, y los resultados globales de todo el proceso, retornando el número de votos obtenidos por cada candidato.

La metodología de recuento utilizada se denomina *FirstPastThePost*, y consiste en declarar como ganador el candidato que recibe un mayor número de votos.

VotingSystem

Se trata de una clase abstracta que implementa los métodos del sistema de votación. De ella hereda la clase *FirstPastThePost,* que desarrolla el algoritmo *calculate* para obtener el candidato con la mayor suma de votos.

3.2.3 Vistas de la aplicación

La aplicación cuenta con cuatro vistas: Inicio de sesión, Registro, Panel para crear una elección y la Interfaz de Votación.

A continuación se describirán las acciones que se pueden realizar desde cada una de estas vistas y sus diferentes partes junto con una serie de capturas de pantalla de las partes más relevantes de *BlockVote* para que se pueda apreciar su aspecto general.

Además de las vistas principales, la aplicación incluye mecanismos auxiliares para notificación de errores (vistas secundarias, mensajes en pantalla, …).

A pesar de esto, siguiendo las instrucciones descritas en el repositorio de la aplicación es posible desplegar *BlockVote* localmente para poder realizar las pruebas y comprobaciones que se consideren oportunas.

Inicio de Sesión

Contiene la interfaz de autenticación de usuarios ya registrados, a través de usuario/email y contraseña.

Se trata de un punto de acceso único para las tres categorías de usuarios: Votantes, Candidatos y Administradores.

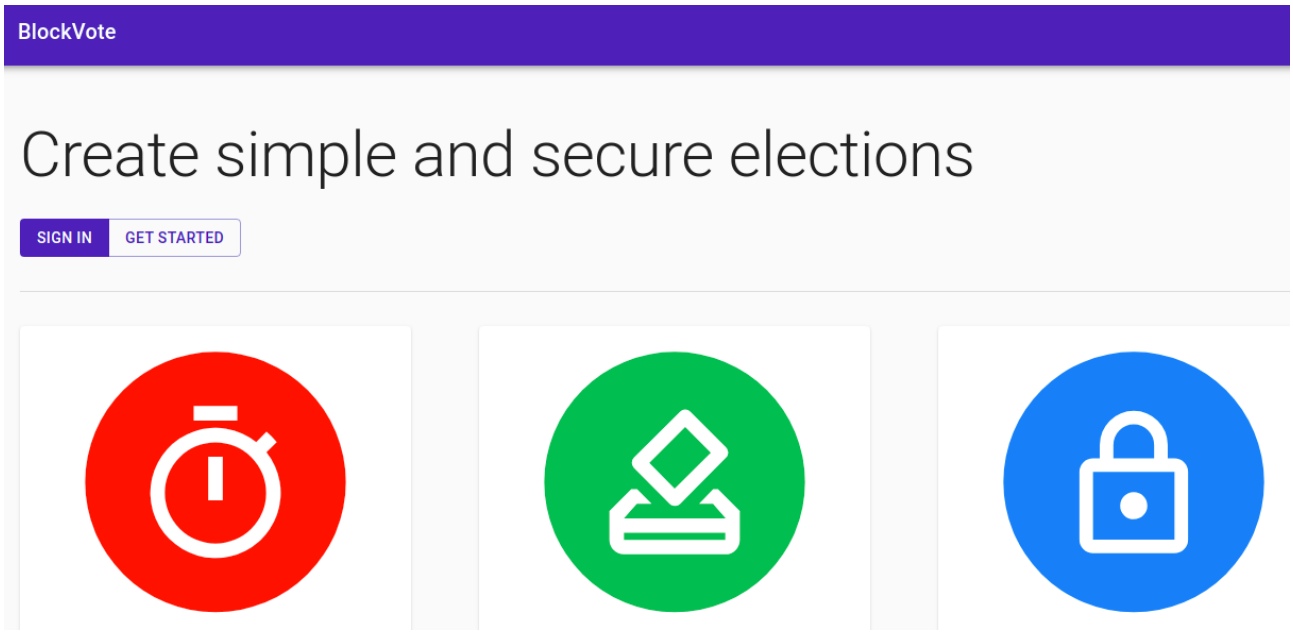

Figura 3.2: Interfaz principal de la aplicación

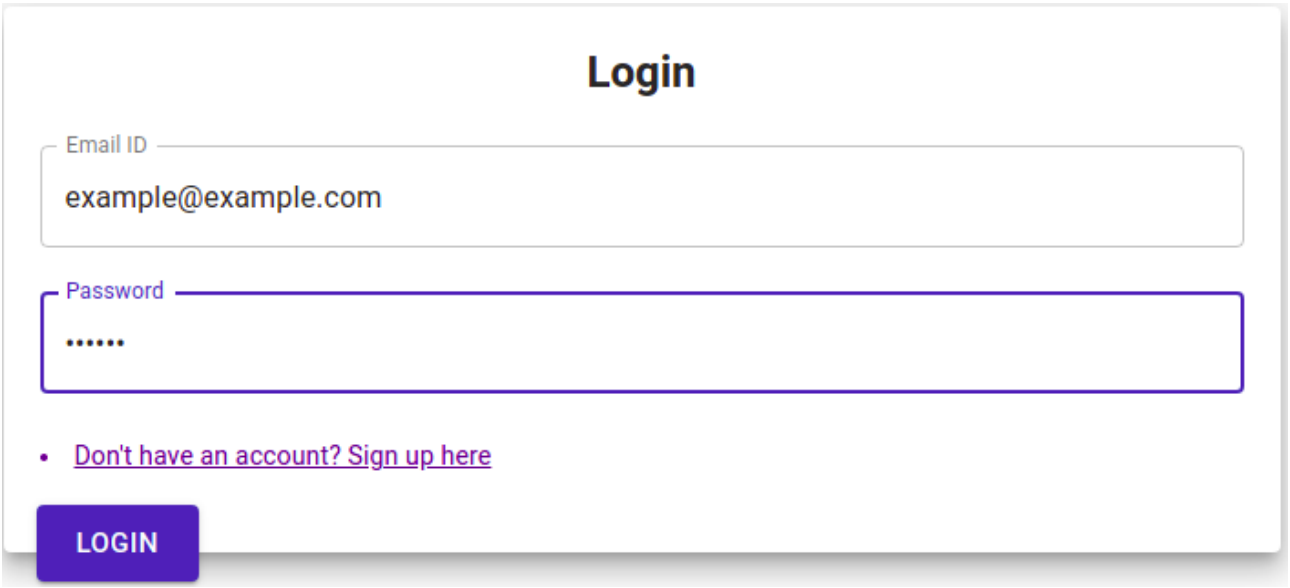

Figura 3.3: Panel de Inicio de Sesión

Registro

Es el medio de inscripción como usuario de la aplicación. Es posible crear una cuenta en las tres categorías de miembros citadas anteriormente.

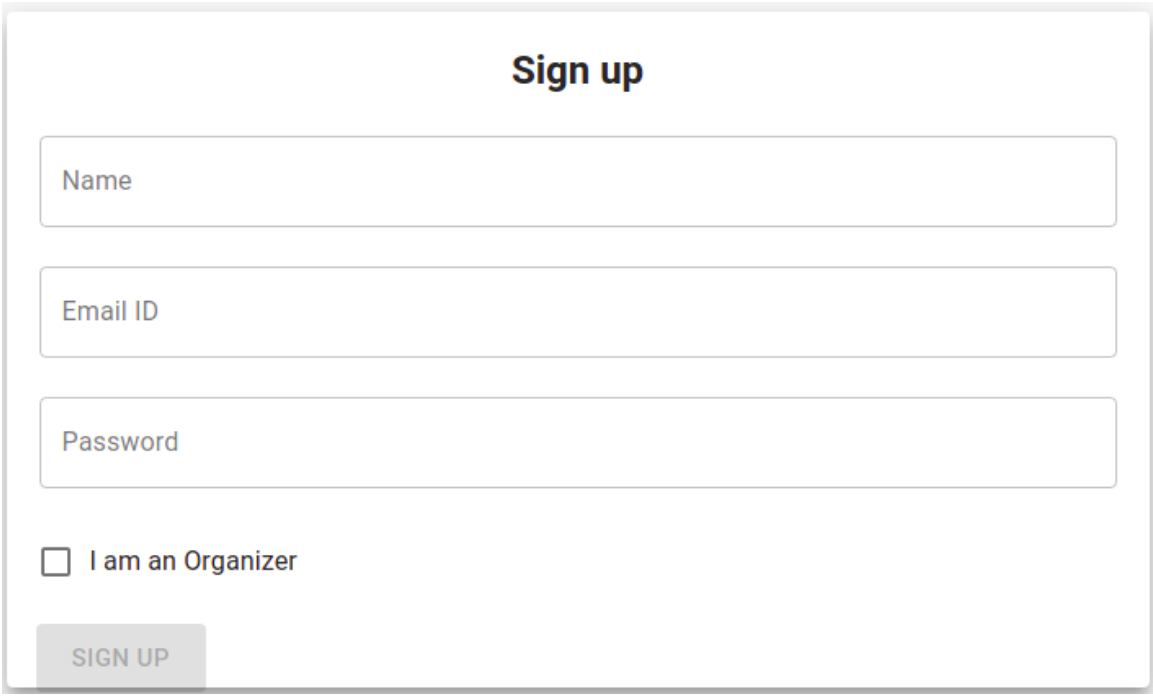

Figura 3.4: Panel de registro de la aplicación

Creación de Elecciones

Se trata de una vista que consta de funcionalidad para crear una elección con un identificador único, seleccionar los administradores de la misma, los candidatos, y finalmente, los votantes.

Incorpora una herramienta para enviar invitaciones a los usuarios seleccionados para participar de la elección.

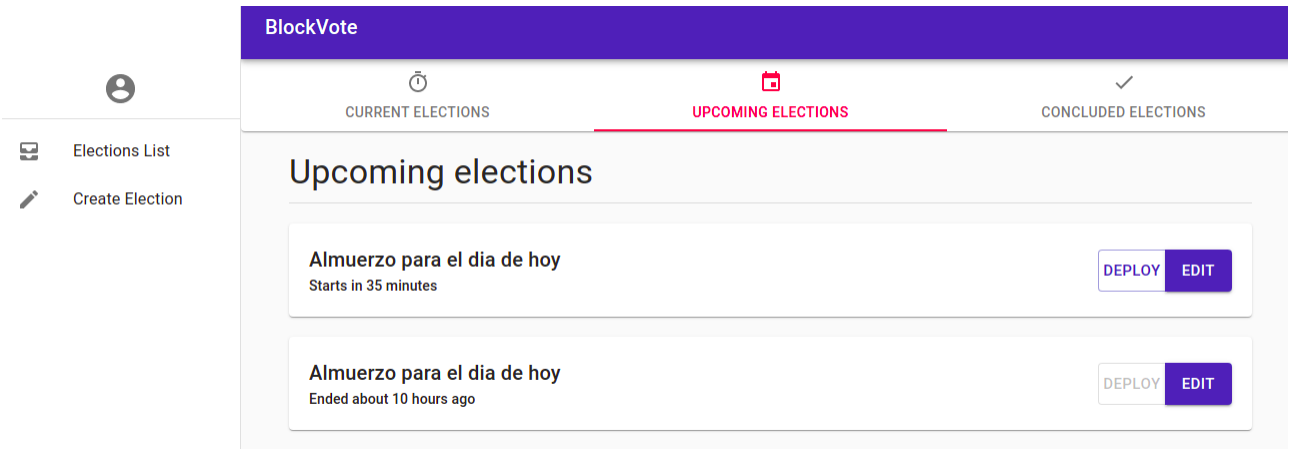

Figura 3.5: Interfaz para la creación de elecciones, con permisos de administrador

Panel de votación

Dispone de la interfaz para enviar el voto, y obtener la certificación de haber votado. Una vez el usuario emite su voto, se bloquea la interfaz en el navegador cliente para evitar la comisión de fraude.

Capítulo 4
Conclusiones y líneas futuras

En este Capítulo se presentan las conclusiones alcanzadas, así como las posibles líneas futuras.

4.1 Conclusiones

Partiendo de los sistemas de votación tradicionales donde es habitual el fraude, se ha realizado una investigación con el propósito de comprender el funcionamiento de una de las tecnologías más prometedoras de la actualidad: la cadena de bloques.

Gracias al estudio realizado, ha sido posible entender cómo al aplicar correctamente esta tecnología se consiguen resultados ventajosos.

La implementación de la aplicación web descentralizada para la plataforma *Ethereum* ha servido para asentar los conocimientos teóricos adquiridos durante las fases iniciales de investigación del proyecto.

A pesar del grado de simplicidad de las acciones que los usuarios de *BlockVote* pueden llevar a cabo haciendo uso de la aplicación, se ha demostrado que esta tecnología es capaz de alterar sustancialmente el modo en el que se realizan y controlan los procesos de numerosos sectores, logrando un estado de confianza máxima entre los interesados del mismo.

Sin embargo, es posible considerar que tal y como se ha concebido la aplicación web y su funcionamiento, no sería excesivamente complicado ampliar sus funcionalidades con aspectos de esta índole. A continuación, se presentan algunas de las posibles líneas futuras a considerar tras la elaboración del proyecto.

4.2 Líneas futuras

Estas son algunas de las posibles ampliaciones que se deberían contemplar como continuación del proyecto:

- Estudiar el uso de algún protocolo como *IPFS* para el almacenamiento distribuido

de datos, evitando su persistencia en la *blockchain* y mejorando el rendimiento a largo plazo

- Estudiar la posibilidad de desarrollar una versión para la plataforma Android e IOS, dado que los teléfonos inteligentes son los dispositivos de mayor uso actualmente.

- Añadir más vistas a la aplicación, ofrecer más opciones para los candidatos, como la posibilidad de hacer campañas enviando notificaciones a los votantes.

Capítulo 5
Summary and Conclusiones

This Chapter presents the conclusions, as well as some research and experimentation proposals to take into account in the future.

5.1 Conclusions

Starting from traditional voting systems where fraud is common, research has been carried out with the purpose of understanding the functioning of one of the most promising technologies of today: blockchain.

Thanks to the study carried out, it has been possible to understand how the correct application of this technology achieves advantageous results.The implementation of the decentralised web application for the Ethereum platform has served to consolidate the theoretical knowledge acquired during the initial research phases of the project.

Despite the degree of simplicity of the actions that *BlockVote* users can carry out using the application, it has been demonstrated that this technology is capable of substantially altering the way in which processes in numerous sectors are carried out and controlled, achieving a state of maximum trust among the stakeholders involved.

However, it is possible to consider that the way the web application has been conceived and functions, it would not be overly complicated to extend its functionalities with aspects of this nature. The following are some of the possible future lines to be considered after the development of the project.

5.2 Future work

These are some of the possible extensions that should be considered as a continuation of the project:

- Study the use of some protocol such as *IPFS* for distributed data storage, avoiding its storage in the blockchain and improving long-term performance.
- Study the possibility of developing a version for Android and IOS mobile devices, given that smartphones are currently the most widely used devices.
- Add more views to the application, offer more options for candidates, such as the possibility of campaigning by sending notifications to voters.

Capítulo 6
Presupuesto

En este Capítulo se expone una estimación del presupuesto del proyecto. El Cuadro 6.1 muestra el coste de cada una de las actividades de investigación e implementación llevadas a cabo.

6.1 Presupuesto

Actividad	**Precio / Hora**	**Horas invertidas**	**Coste**
Investigación *blockchain*	25 €/Hora	75	1875
Desarrollo Contratos Inteligentes	30 €/Hora	50	1500
Desarrollo Aplicación Web	28 €/Hora	35	980
Integración aplicación web - *Smart Contracts*	26 €/Hora	35	910
Redacción de la memoria del TFM	30 €/Hora	40	1200
Total	-	235	6465 €

Tabla 6.1: Desglose del coste de la aplicación dividido en tareas

6.2 Glosario

Bytecode Lenguaje intermedio al que se compilan los programas escritos en algunos lenguajes de alto nivel para posteriormente ser interpretados eficientemente por un intérprete.

Criptomoneda Medio digital de intercambio, se utiliza criptografía para asegurar las transacciones, controlar la creación de unidades adicionales y verificar la transferencia de activos usando tecnologías de registro distribuido (DLT)**.**

Decentralized Apps Aplicación descentralizada, tipo de aplicación cuyo funcionamiento se basa en una red descentralizada de nodos que actúan entre sí sin necesidad de un organismo regulador central.

Distributed Ledger Technology Tecnología de registro distribuído, mediante el uso de este tipo de tecnologías se permite la creación de registros inalterables gestionados de forma descentralizada.

Ether Criptomoneda nativa de la plataforma Ethereum.

Función trampa Función matemática cuyo cálculo directo es sencillo, pero en la que invertir los operandos a partir del resultado es muy complejo.

Gas Nombre de la cuota que ha de abonarse en la plataforma *Ethereum* al llevar a cabo transacciones.

Minero Nodo que participa en una red blockchain aportando su potencia de cómputo para la creación o “minado” de nuevos bloques para la cadena.

Peer-to-peer Red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, de manera que todos los nodos se comportan como iguales entre sí.

Censo Lista oficial de las personas que tienen derecho a votar en una población o un estado.

Circunscripción Una circunscripción electoral es una subdivisión territorial para elegir miembros a un cuerpo legislativo. Generalmente, solo los votantes (constituyentes) que residen dentro del distrito tienen permitido votar en una elección celebrada allí.

Backend Parte del desarrollo web que se encarga de que una página web funcione. Se trata de la parte no visible de la aplicación que se almacena en el Servidor.

6.3 Acrónimos

DLT *Distributed Ledger Technology*, Tecnología de Registro Distribuida.

EVM *Ethereum Virtual Machine*, máquina virtual de *Ethereum*.

P2P *Peer-to-Peer*, red de pares.

PoA *Proof-of-Authority*, prueba de autoridad

PoS *Proof-of-Stake*, prueba de participación.

PoW *Proof-of-Work*, prueba de trabajo.

Capítulo 7

7.1 Bibliografía

24: Bitboy, Bitcoin logo, 2014

1: David Chaum, Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups, 1982

2: Haber, Stuart; Stornetta, W. Scott, How to time-stamp a digital document, 1991

3: Satoshi Nakamoto (Seudonym), Bitcoin: A Peer-to-Peer Electronic Cash System, 2008

4: Rodríguez Burgos; Karla Eugenia, Democracia y tipos de democracia, 2015

25: Electoral Commision of South Africa, Voting: How it works?, 2021

26: Page, María, ¿Cómo medir la confiabilidad de las elecciones?, 2015

5: Hans Peter Luhn, The Automatic Creation of Literature Abstracts, 1958

27: Editor de la página, ¿QUÉ ES LA FUNCIÓN HASH?, 2016

6: Ronald Linn Rivest; Adi Shamir; Leonard Adleman , A Method for ObtainingDigital Signatures and PublicKey Cryptosystems , 1978

23: Redondo, Alba, Dís de conmoción en el sistema financiero mundial, 2008

7: Merkle Ralph, Method of Providing Digital Signatures, 1982

29: Wander, Mathaus, Bitcoin Block Data, 2013

30: Michelin, Regio, A Decentralised Approach to Task Allocation Using Blockchain, 2018

8: Steve Crocker, Host Software, 1969

9: Dwork, Cynthia; Naor, Moni, Pricing via Processing or Combatting Junk

Mail, 1993

10: King, Sunny; Nadal, Scott, PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake, 2012

11: Wood, Gavin, Proof of Authority Private Chains, 2015

31: Euromoney Editors, How does a transaction get into the blockchain?,

12: Szabo, Nick, Formalizing and Securing Relationships on Public Networks, 1997

43: Wikipedia, Historia de los contratos inteligentes, 2019

13: Buterin, Vitalik, A next-generation smart contract and decentralized application platform, 2014

44: Buterin, Vitalik, Solidity, 2014

17: Gentry, Craig, Fully homomorphic encryption using ideal lattices, 2009

34: El Yahyaoui, Ahmed, A New Encryption Scheme to Perform Smart Computations on Encrypted Cloud Big Data, 2019

39: Rivest, Ronald Linn; Shamir, Adi; Adleman, Leonard Max, A method for obtaining digital signatures and public-key cryptosystems, 1978

41: Paillier, Pascal, Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,

18: Walke, Jordan, React History, 2011

19: , Material Design Library,

20: Coulter, Tim; Feickert, Tyler; McVay, Wes; (+10), Truffle Suite, 2018

21: Coulter, Tim; Feickert, Tyler; McVay, Wes; (+10) , Ganache Suite, 2018

36: Google, Firebase,

22: Davis, Aaron; Finlay, Dan; Huang, Thomas; (+10), MetaMask: A Crypto Wallet, 2016

7.2 Apéndice

A continuación se anexa el código de los contratos desarrollados para organizar la elección.

Election

```
pragma solidity >=0.7.0 <0.8.0;


import "./VotingSystem.sol";
import "./FirstPastThePost.sol";
import "./DataTypes.sol";


/**
@title Election
@dev Main contract used to set up the election on the blockchain
 */
contract Election {
    DataTypes.Candidate[] public candidates;


    enum votingTypes {firstPastThePost}
    votingTypes votingType;


    uint256 public endTime;
    uint256 public startTime;


    address public organizer;
```

```
mapping(address => DataTypes.Voter) public voters;


modifier onlyHost() {
    require(msg.sender == organizer);
    _;
}


modifier onlyDuringElection() {
    require(block.timestamp < endTime, "Election has ended");
    require(block.timestamp >= startTime, "Election has started");
    _;
}


modifier onlyBeforeElection() {
    require(block.timestamp < startTime, "Election start time has passed");
    _;
}


modifier onlyAfterElection() {
    require(block.timestamp > endTime, "Election end time has not passed");
    _;
}
```

```
constructor(
    string[] memory candidateNames,
    uint256 _endTime,
    uint256 _startTime
) {
    votingType = votingTypes.firstPastThePost;
    organizer = msg.sender;
    endTime = _endTime;
    startTime = _startTime;
    for (uint256 i = 0; i < candidateNames.length; i++) {
        candidates.push(
            DataTypes.Candidate({
                name: candidateNames[i],
                voteCount: 0,
                id: i
            })
        );
    }
}



function giveRightToVote(address[] memory voterAddress) public onlyHost {
    for (uint256 i = 0; i < voterAddress.length; i++) {
        voters[voterAddress[i]].validVoter = true;
    }
}
```

```
function vote(uint256 candidateID) public onlyDuringElection {
    require(voters[msg.sender].validVoter, "Has no right to vote");
    require(!voters[msg.sender].voted, "Already voted.");
    voters[msg.sender].voted = true;
    voters[msg.sender].votedFor = candidateID;
    candidates[candidateID].voteCount += 1;
}


function getWinner()
    public
    onlyAfterElection
    returns (string memory winnerName)
{
    FirstPastThePost countMethod = new FirstPastThePost(candidates);


    uint256[10] memory winners = countMethod.calculate();
    winnerName = string(candidates[winners[0]].name);
    for (uint256 i = 1; i < winners.length; i++) {

        if (winners[i] == 0 && i != 0) {
            break;
        }
        winnerName = string(
```

```
                abi.encodePacked(winnerName, ",", candidates[winners[i]].name)
            );
        }
    }



    function numberOfCandidates() public view returns (uint256 num) {
        num = candidates.length;
        return num;
    }
}
```

FirstPastThePost

```
pragma solidity >=0.7.0 <0.8.0;
pragma experimental ABIEncoderV2;
import "./VotingSystem.sol";
import "./DataTypes.sol";

contract FirstPastThePost is VotingSystem {
    constructor(DataTypes.Candidate[] memory _candidate) {
        for (uint256 i = 0; i < _candidate.length; i++) {
            candidates.push(_candidate[i]);
        }
    }

    function calculate()
        external
```

```
        view
        override
        returns (uint256[10] memory winningCandidate)
    {
        uint256 winningVoteCount = 0;
        uint256 tempIndex = 0;
        for (uint256 i = 0; i < candidates.length; i++) {
            if (tempIndex > 10) {
                break;
            }
            if (candidates[i].voteCount > winningVoteCount) {
                winningVoteCount = candidates[i].voteCount;
                delete winningCandidate;
                tempIndex = 0;
                winningCandidate[tempIndex] = i;
            } else if (candidates[i].voteCount == winningVoteCount) {
                winningCandidate[tempIndex] = i;
                tempIndex++;
            }
        }
    }
}
```

VotingSystem

```
pragma solidity >=0.7.0 <0.8.0;

import "./DataTypes.sol";


abstract contract VotingSystem {
```

```
    DataTypes.Candidate[] candidates;

    function calculate() external view virtual returns (uint256[10] memory);
}
```

Datatypes

```
pragma solidity >=0.7.0 <0.8.0;

library DataTypes {

    struct Voter {
        bool voted;
        uint256 votedFor;
        bool validVoter;
    }

    struct Candidate {
        string name;
        uint256 voteCount;
        uint256 id;
    }
}
```