Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

SISTEMA DE TRANSFERENCIA DE SALDO MEDIANTE MONEDEROS VIRTUALES BASADO EN BLOCKCHAIN

Alumno: López Cano, David

Tutor: Prof D. Manuel José Lucena López
Dpto.: Departamento de informática

Septiembre, 2020

Índice de figuras

1. Introducción

1.1 Motivación

Desde la llegada en el año 2008 de Bitcoin, la tecnología blockchain ha generado un gran interés que ha ido creciendo de forma exponencial con el paso de los años.

Si bien el primer y más conocido campo de aplicación de blockchain ha sido el campo de las criptomonedas, actualmente la cadena de bloques es utilizada en áreas como la gestión de una cadena de suministros, servicios de notaría, internet de las cosas, voto electrónico, compañías aseguradoras, etc.

Además esta tecnología introduce un nuevo concepto conocido como internet del valor. El internet del valor es un paradigma que permite intercambiar cualquier tipo de activo en internet sin la necesidad de una autoridad central que gestione la confianza de las transacciones que se llevarán a cabo.

La posibilidad de convertir cualquier tipo de activo en *tokens,* y poder registrarlos, intercambiarlos y almacenarlos en una red blockchain ha generado un gran expectación en el uso de esta tecnología. Euros, dólares, una casa o incluso una bebida son activos que pueden ser tokenizables, y a la vez ser compartidos sin necesidad de depender de terceros o intermediarios que obtengan un porcentaje de la transacción, depositando la confianza de la transacción en el sistema. Este paradigma hace de la red un lugar más libre y democratico en el que es posible omitir la intermediación de grandes empresas o bancos.

Por ello en este trabajo fin de máster se estudiará el potencial de esta tecnología, mostrando especial atención en el intercambio de saldo. Se llevará a cabo el análisis, diseño y realización de una PoC (prueba de concepto) de un sistema que permita la gestión de monederos virtuales y la realización de transacciones de saldo seguras y fiables de entre los mismos quedando registradas en una blockchain privada.

1.2 Objetivos

Para el desarrollo del presente trabajo se han definido los siguientes objetivos.

- Hacer una revisión bibliográfica sobre la tecnología blockchain.
- Analizar la aplicación de la tecnología blockchain en el intercambio de activos.
- Diseñar e implementar una prueba de concepto de transferencias de saldos a través de monederos virtuales basada en blockchain.

1.3 Metodología

En este apartado se enumeran los conceptos básicos de metodología a desarrollar para la realización de este proyecto.

- Revisión del estado del arte sobre la tecnología blockchain.
- Recopilación de librerías de código abierto y análisis de sus puntos fuertes y débiles desde el punto de vista de la funcionalidad, eficiencia, facilidad de uso, etc.
- Elección justificada de una de las librerías evaluadas.
- Desarrollo de una aplicación simple que haciendo uso de la librería escogida, permita construir y gestionar una blockchain privada que permita el intercambio de activos.
- Discusión del resultado obtenido, y extracción de conclusiones.

1.4 Estructura del proyecto

Para facilitar la lectura de este documento se detallan a continuación los puntos por los que está formado.

En el apartado número dos, tecnología blockchain, se ha realizado una completa revisión bibliográfica de diversos ámbitos relacionados este paradigma. En primer lugar se han definido los fundamentos de la tecnología blockchain, comenzando por describir el contexto en el que nace esta tecnología, pasando a definir en qué consiste la cadena de bloques y finalizando por describir los tres principales pilares tecnológicos donde se apoya: las redes *Peer-to Peer*, la teoría de juegos y la criptografía. Otro de los conceptos que se han trabajado en este apartado ha sido el internet del valor y cuál es su relación con los tokens criptográficos, y en especial con el token ERC20. También se ha estudiado y definido cuales son las principales plataformas blockchain de código abierto principalmente diseñadas para un ámbito empresarial, así como en qué consisten los contratos inteligentes. Para finalizar este apartado se ha analizado el estado del arte actual de proyectos que se basan en el envío de saldo basadas en blockchain.

Las herramientas y tecnologías utilizadas para el desarrollo de este proyecto han sido descritas en el apartado número tres. Presentando más atención a tecnologías como Spring Boot sobre la cual se definirá la API Rest que dará forma este proyecto. Se destaca también el papel que ha tenido Web3J en este proyecto ya que es la librería que permite la comunicación entre el servidor y red blockchain.

En el apartado número cuatro se detalla el proceso de ingeniería del software llevado a cabo para el desarrollo de este proyecto. En la fase de análisis se han definido los requisitos funcionales y de calidad, los perfiles de usuario de la aplicación y se detallan los motivos de porque ha sido Quorum la plataforma blockchain elegida. A continuación en la fase de diseño se adjunta el diagrama de clases y los distintos diagramas de secuencias asociados a los distintos casos de uso del proyecto. Para finalizar, en la etapa de implementación se define la arquitectura del sistema, los IDEs utilizados para el desarrollo del proyecto y se comentan los detalles de implementación que se han considerado relevantes.

El quinto apartado aglutina las conclusiones obtenidas tras la realización de este proyecto y las propuestas futuras de mejora.

La bibliografía consultada durante la realización de este trabajo será definida en el apartado sexto.

Por último, este trabajo consta de tres anexos. El primer anexo recoge las pautas para crear desplegar la red blockchain Quorum. Por otro lado, el segundo anexo definirá cómo desplegar el servidor Spring Boot y utilizar el cliente Swagger. Para finalizar se detallan todos los archivos y ficheros que componen este proyecto.

1.5 Planificación temporal

A continuación se muestran las tareas que se han realizado para el desarrollo de este trabajo fin de máster, así como su planificación temporal durante el curso 2019/2020.

Work Breakdown Structure	Start	End
REVISIÓN BIBLIOGRÁFICA	02/09/2019	17/02/2020
Blockchain	02/09/2019	07/10/2019
Plataformas blockchain	07/10/2019	04/11/2019
Ethereum	04/11/2019	02/12/2019
Solidity	02/12/2019	06/01/2020
Quorum	06/01/2020	03/02/2020
Web3J	03/02/2020	17/02/2020
ANALISIS	17/02/2020	09/03/2020
Definición de requisitos	17/02/2020	02/03/2020
Elección de tecnología	02/03/2020	09/03/2020
DISEÑO	30/03/2020	04/05/2020
Diseño preeliminar	30/03/2020	13/04/2020
Diseño de la arquitectura	13/04/2020	27/04/2020
Elaboración de diagramas	27/04/2020	04/05/2020
IMPLEMENTACIÓN	04/05/2020	20/07/2020
Despligue red Quorum	04/05/2020	08/06/2020
API Rest	25/05/2020	29/06/2020
Configuración Swagger	29/06/2020	13/07/2020
Correción de errores	13/07/2020	20/07/2020
DOCUMENTACION	06/07/2020	24/08/2020
Elaboración de la memoria	06/07/2020	24/08/2020

Figura 1.1: Desglose de tareas

Se han considerado cinco grupos principales. El primero, y el más extenso en tiempo, ha sido la revisión bibliográfica de todo lo relacionado con la tecnología blockchain y sus aplicaciones, así como el lenguaje utilizado para el desarrollo de contratos inteligentes.

El segundo grupo se enmarca dentro del proceso de ingeniería del software. Se trata del análisis de la aplicación, en el se llevan a cabo todo lo relacionado con la definición de los requisitos, así como las deliberaciones que se han llevado a cabo para la elección de Quorum como tecnología blockchain.

El tercer bloque aglutina las tareas de diseño, tales como la elaboración de un diseño preliminar, pasando por un diseño final de la arquitectura a implementar y la elaboración del diagrama de clases y los diagramas de secuencia asociados a los casos de uso.

Las tareas relacionadas con la implementación se agrupan en el cuarto bloque. El despliegue de la red Quorum, el desarrollo de la API Rest en Spring Boot, la configuración de Swagger y la corrección de errores pertenecen a este bloque.

Y para finalizar, el desarrollo de la memoria, donde se recogerán todo lo realizado durante este proceso de desarrollo.

Se adjunta el diagrama de Gantt asociado a esta planificación de tareas.

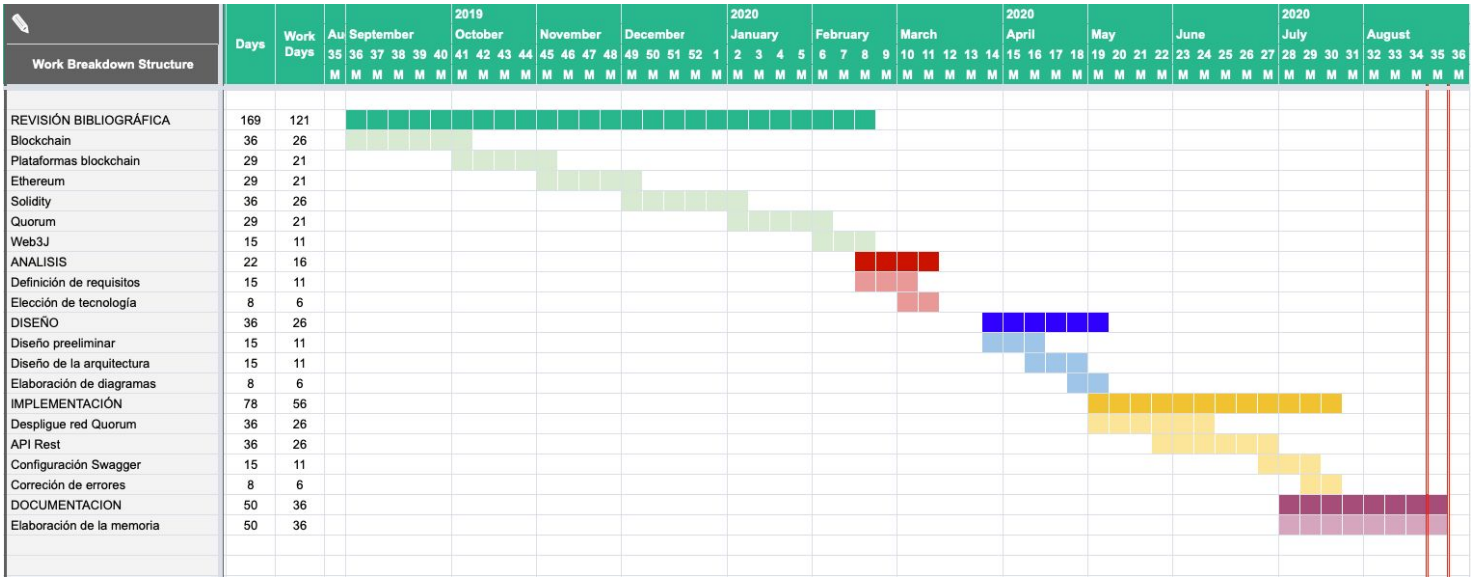

Figura 1.2: Diagrama de Gantt.

2. Tecnología blockchain

2.1 Fundamentos de la tecnología blockchain

2.1.1 Antecedentes

Si nos remontamos a los años 40 del siglo pasado, el dinero se basaba en el patrón “oro” y este estaba definido en función de unidades de oro. Este tipo de dinero, llamado fiduciario, establece una promesa de pago en función de las reservas de oro que cada país almacena.

Fue en 1971 cuando el gobierno de Estados Unidos debido a escasez de oro decidió que el dinero fuese respaldado por un Banco Central y no por las reservas de oro de cada país. Por lo tanto, no existe elemento tangible que dé valor a un billete o a una moneda , si no que el valor viene dado por decreto, por la confianza que los ciudadanos tienen en su Gobierno, a este dinero sin respaldo material lo conocemos como dinero fiat. Bajo esta nueva forma el Banco Central es el encargado de imprimir dinero y distribuirlo entre las entidades bancarias que pertenezcan a él y estas se encargarán de hacérselo llegar a los ciudadanos.

En el año 2007 se desencadenó un momento de inestabilidad dada la gran dinámica especulativa del sector inmobiliario en los primeros años del siglo XXI. Fueron desencadenantes de esta crisis tanto la concesión por parte de las entidades bancarias de préstamos hipotecarios sin las suficientes garantías, como la oferta de diversos productos financieros que prometían grandes beneficios al cliente sin que este conociera el riesgo del mismo y en qué estaba invirtiendo. Estos factores dieron lugar a una de las crisis económicas más importantes de la historia.

Los ciudadanos confiaban en que, el sistema financiero gozaba de buena salud para saldar sus deudas pero comprobaron que los bancos no lograban parar las órdenes de desahucio ni hacerse cargo de muchos fraudes cometidos al ofertar productos financieros en años de bonanza. Por lo tanto, en la población se generó un clima de desconfianza respecto a los bancos centrales por lo que, distintos grupos sociales demandaban modelos financieros alternativos que disminuyeran la dependencia con el dinero fiat.

Bajo este escenario de crisis económica ve la luz Bitcoin, mediante la publicación en el año 2008 de un artículo de forma anónima, bajo el seudónimo de Satoshi Nakamoto [2]. Bitcoin dio lugar a un nuevo sistema financiero alternativo al dinero fiat, que sustituye la autoridad del Banco Central por un protocolo de consenso entre las entidades participantes en el sistema, de forma que, se elimina

el rol de autoridad central y por tanto, la dependencia del sistema en una única entidad [3].

2.1.2 Concepto de blockchain

Con la llegada de Bitcoin aparece también un nuevo concepto, que estaba destinado a ser revolucionario, la cadena de bloques conocida en inglés como blockchain.

Definimos blockchain como una tecnología que funciona como un libro de contabilidad digital, llamado ledger, que es descentralizado y es actualizado por todos los participantes que forman la red de manera que se garantiza la confiabilidad y seguridad del sistema sin necesidad de que exista una autoridad central.

Las transacciones entre las cuentas quedan registradas en la cadena de bloques. Pueden ser transacciones monetarias, una transferencia de criptomonedas de la blockchain o también se puede transmitir otra información como por ejemplo una transacción de tokens. Las transacciones son almacenadas en bloques organizados según un árbol de Merkle, a su vez cada bloque forma entre sí una cadena que hace referencia al bloque anterior mediante un hash criptográfico. El uso de la criptografía es esencial para garantizar la privacidad y la transparencia.

Los datos se almacenan a través de una red *peer-to-peer*, en esta red todos los nodos funcionan tanto de cliente como de servidor almacenando una copia completa o parcial del estado de la red, permitiendo la descentralización del sistema. A su vez, los nodos validan las transacciones mediante algoritmos de consenso con el apoyo de la mayoría y en algunos casos ofreciendo incentivos económicos para la motivación y el éxito de la red [4].

En una blockchain pública cualquier persona ajena o propia de la red puede rápidamente verificar la red, no es necesario que un órgano central verifique la red. Esto no ocurre en el mercado financiero, ya que, no podemos verificar las transacciones que se realizan en este sistema a excepción, de los propietarios de cada transacción.

2.1.3 Redes peer to peer

Como hemos indicado con anterioridad, blockchain opera sobre internet en una red *peer-to-peer* (P2P) o red de pares [5]. En esta red todos los dispositivos se comportan como iguales actuando de manera simultánea, como clientes y servidores respecto a los demás nodos. Por lo tanto, cada nodo es independiente de la red, pero a la vez interdependiente del resto de nodos, dado que, sin la ayuda del resto de nodos la red no existiría.

La principal característica de esta arquitectura es la ausencia de un servidor central encargado de procesar el intercambio de información, por lo tanto, no existe ninguna entidad que tenga el control absoluto de la red, evitando así la congestión y los cuellos de botella, dotando a la red de una alta tolerancia a fallos y una gran robustez. Este tipo de redes son apropiadas para sistemas exigentes que necesitan el máximo aprovechamiento de los recursos sin tener que delegar en un servidor central.

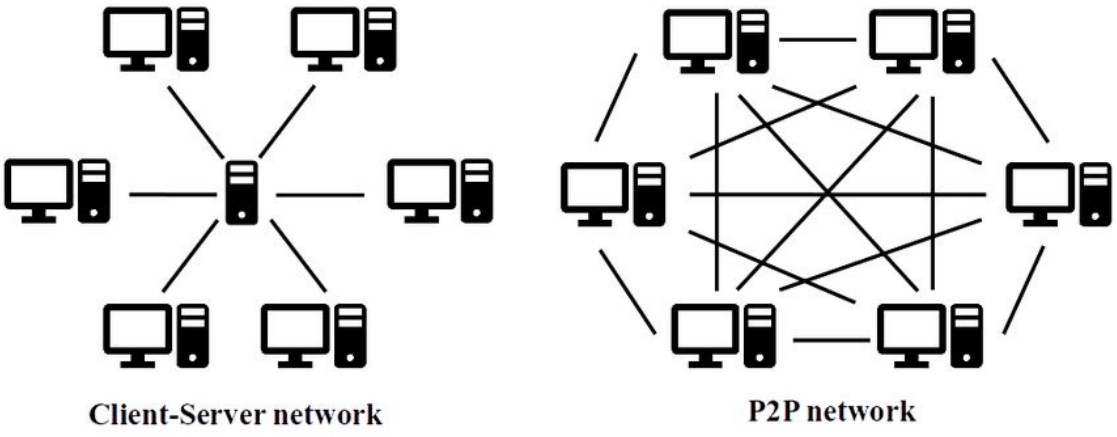

Figura 2.1: Diferencia entre arquitectura cliente-servidor y P2P

En blockchain, aunque todos los nodos son iguales, pueden tomar ciertos roles dentro de la red. Uno de los roles a destacar por su importancia es el de *nodo completo*, estos nodos son los encargados de almacenar una copia completa del estado de la red. Por lo tanto, toda la información almacenada en la blockchain no se podrá destruir siempre que exista al menos un nodo completo inalterado, y en este caso, siempre se podrá reconstruir la red.

2.1.4 Teoría de juegos

La teoría de juegos [6] es un área de la matemática aplicada que utiliza modelos para estudiar la toma de decisiones estratégicas. La teoría de juegos se ha convertido en una herramienta de gran importancia para la teoría económica y ha ayudado a comprender la conducta humana frente a la toma de decisiones, buscando estrategias óptimas, el comportamiento previsto y observando a los individuos en juegos. Cuando nos referimos a juego, en este ámbito, hacemos referencia a cualquier interacción entre dos o más personas en la que la recompensa que obtiene un participante se ve afectada por las decisiones de los demás.

La teoría de juegos tiene gran importancia en el protocolo blockchain, dado que, en esta red interdependiente existen múltiples actores en un ámbito distribuidor. La cadena de bloques está formada por un conjunto de bloques que están unidos entre sí por orden cronológico y estos contienen transacciones, el hash del bloque anterior vincula los bloques unos con otros formando una cadena. Cada bloque en la cadena tiene una puntuación, empezando por el bloque génesis, que es el primer bloque de la cadena, el cual tiene una puntuación nula, el resto de bloques puntuará como la suma de las puntuaciones de su bloque anterior más la suma de la prueba de trabajo. El estado que se considerará verdadero será el bloque con una puntuación más alta.

Los mineros pueden llegar a engañar al sistema, por ello, la cadena de bloques usa modelos de teoría de juegos que premian la honestidad del sistema. En el caso de que se introduzca un bloque no válido en el sistema, la cadena actúa marcando como invalido ese bloque y todos los demás que le preceden. De esa forma, se evitará que el resto de mineros continúen esa cadena, y seguirán minando la cadena que tiene una puntuación más alta y que se considera verdadera.

2.1.4.1 Algoritmos de consenso

Los algoritmos de consenso [7] son utilizados para llegar a un acuerdo dentro de un grupo, son procesos de toma de decisiones grupales donde cada individuo construye y apoya una decisión que funcione mejor para él mismo, aunque finalmente todos los miembros deberán de apoyar la decisión mayoritaria, sea o no la elegida por él. Estos algoritmos no solo están de acuerdo con la mayoría de los votos, sino que también con aquello que beneficie a todos por lo tanto estos

protocolos de consenso garantizan la igualdad y equidad del sistema. Los objetivos de estos algoritmos son los siguientes:

- Igualdad de derechos: la votación de cada participante tiene el mismo valor en la red, cada voto tiene la misma importancia.

- Cooperación: los participantes deben de buscar el beneficio de la red y dejar de lado los intereses propios.

- Participación: todos los que estén dentro de la red deben participar en la votación.

- Acuerdo: el objetivo del algoritmo de consenso es llegar a un acuerdo a partir de la votación

En una plataforma blockchain un algoritmo de consenso es utilizado para confirmar transacciones y añadir nuevos bloques a la cadena, y debe asegurar que los nuevos bloques sean la única versión de la verdad y muestran una representación fidedigna del sistema. También debe de evitar que participantes de la red cometan irregularidades en la red y consigan bifurcar la cadena hacia un estado erróneo, incentivando a los miembros que realicen un uso correcto del sistema.

Existen muchos algoritmos para lograr el consenso en una cadena de bloques, ya que, a medida que va pasando el tiempo, se van desarrollando nuevos algoritmos que permiten que las transacciones se confirmen con más rapidez y consuman menos recursos.

A continuación se describen los grupos de protocolos de consenso más utilizados.

2.1.4.2 Proof-of-work

El protocolo *Proof-of-work* [8] (PoW) o prueba de trabajo fue introducido por Bitcoin en el año 2008. Este protocolo se basa en obtener una respuesta a un problema matemático para llegar a obtener la solución se requiere del empleo de trabajo. El trabajo consiste en emplear capacidad computacional para resolver el problema planteado. Al proceso de encontrar esta solución se le conoce como minado, y los nodos encargados de llevarlo a cabo son llamados mineros.

En la práctica, para encontrar la solución, los nodos que forman la red deberán de ir aportando respuestas mediante fuerza bruta. Estas respuestas son el resultado de aplicar una función *hash* al bloque que se necesita validar. Este bloque está formado por el *hash* del bloque anterior, timestamp del momento en el que se aplica el *hash*, conjunto de transacciones sin validar formando un *árbol de merkle* y el *nonce.* El *nonce* es un número obtenido mediante ensayo y error con el que el resultado de la función hash va variando.

El problema estará resuelto cuando un hash alcance el valor del hash objetivo y por tanto, el bloque será añadido a la red, las transacciones validadas, y el minero será premiado con una recompensa.

Estamos ante un protocolo optimizado para grandes redes distribuidas que será más seguro ante un mayor número de nodos en la red, ya que es vulnerable ante *ataques del 51%*. Estos ataques son provocados cuando mineros deshonestos secuestrasen el 51% del poder de cómputo de la red ya que podrían manipular la red para su propio interés.

Este protocolo es utilizado en redes públicas como Bitcoin o Ethereum, en las que la confianza entre los participantes es nula, por lo tanto, la capacidad de cómputo requerida y la electricidad consumida es muy alta. PoW es un protocolo muy costoso que consume mucha energía, de forma inútil. Esta energía consumida garantiza la seguridad de la red pero no puede aplicarse a ninguna otra actividad productiva.

2.1.4.3 Proof-of-Stake

Proof-of-Stake [9] (PoS) o *prueba de participación* es un protocolo que fue diseñado para evitar las desventajas de PoW en redes blockchain públicas. En este protocolo la figura de los mineros que gastan energía para obtener una recompensa es sustituida por nodos validadores que deben de invertir una cantidad de criptomoneda para minar un bloque, por lo que la probabilidad de validar un bloque es directamente proporcional a la cantidad de monedas que se tienen acumuladas, y solo podrá participar en el porcentaje de recursos que tenga en su poder.

El minero que consiga validar el bloque suele ser elegido en función del valor invertido en la red en comparación con el valor total del sistema, con esto se consigue evitar un mal comportamiento de los mineros debido a que la verificación será realizada por los que tienen más valor en esta y por lo tanto, están interesados en que las verificación se realicen de manera correcta.

La principal ventaja de este protocolo frente a PoW es que su consumo de energía es mucho menor, lo que implica una mayor rentabilidad para los usuarios y un gran favor para el medio ambiente.

También aporta grandes beneficios con respecto al problema de un ataque del 51% dado que para realizar un ataque de este tipo con el protocolo PoS es necesario poseer un 51% de la criptomoneda o tokens asociados a la red, lo cual es mucho más difícil y costoso.

2.1.4.4 Crash fault tolerance (CFT)

El protocolo *Crash fault tolerance* [10] (CFT) o *protocolo de tolerancia a fallos* es el encargado de no dejar caer una red ante fallos hardware del sistema. Una conexión de red, un dispositivo o un nodo pueden dejar de funcionar y CFT garantiza que el sistema pueda alcanzar el consenso, si los componentes fallan.

Este protocolo es adecuado para redes en las que todos los participantes tienen confianza entre sí, ya que, no es tolerante a la presencia de actores maliciosos en el sistema, por lo que debe ser aplicado en redes privadas que se beneficiarán de su gran rapidez y rendimiento.

2.1.4.5 Byzantine fault tolerance (BFT)

Como hemos visto en el protocolo anterior, los sistemas distribuidos se enfrentan a ciertas amenazas, si bien los algoritmos basados en CFT solucionan el problema de fallos hardware en el sistema también se ha de tener en cuenta la presencia de actores maliciosos en la red, hablamos del protocolo de consenso *Byzantine fault tolerance* (BFT) o *protocolo tolerante al problemas de faltas bizantinas.*

Para poder comprender este protocolo de consenso es necesario conocer el problema de los generales bizantinos, que Leslie Lamport, Robert Shostak, y Marshall Pease plantearon en un paper en 1982 [11].

"Imaginamos que varias divisiones del ejército bizantino están acampadas fuera de una ciudad enemiga, cada división dirigida por su propio general. Los generales pueden comunicarse entre sí sólo por mensajería. Después de observar al enemigo, deben decidir sobre un plan de acción común, atacar o retirarse".

Este problema puede darse en cualquier paradigma de computación distribuida, como por ejemplo en blockchain, ya que, existe un riesgo de actores maliciosos. Cada miembro del sistema tiene el rol de general y recibe una transacción, que se asemeja a una orden del general, y emite esta orden a los demás. La orden de un traidor que sea contradictoria quedará en minoría y se dará por correcta la orden mayoritaria, de esta forma los traidores quedan anulados.

En términos de blockchain se permite que los nodos validadores gestionen el estado de la cadena de bloques e intercambien mensajes para acordar el histórico de transacciones correcto y garantizar la honestidad.

Entre las ventajas de este tipo de algoritmos destacamos la escalabilidad y el bajo consumo de energía.

2.1.5 Criptografía

Definimos criptografía [12] como el conjunto de técnicas, que tratan sobre la protección, ocultando la información frente a observadores no autorizados.

La criptografía tiene un papel muy importante en la tecnología blockchain pues dota al sistema de la seguridad necesaria, para que, se puedan llevar a cabo el correcto funcionamiento de la cadena de bloques. Es necesario generar un sistema fiable y robusto que aporte la confianza necesaria, sin que existan intermediarios o una entidad central reguladora.

A continuación, se describirán los conceptos criptográficos más importantes relacionados con la tecnología blockchain.

2.1.5.1 Criptografía asímetrica

Los algoritmos asimétricos o de clave pública han demostrado su interés para ser utilizados en redes donde la confianza entre los participantes es baja, como puede ser blockchain. Para ello, utiliza un par de claves que se encargan del cifrado y descifrado de los mensajes. El par de claves pertenece a la misma persona, y la clave pública será distribuida, mientras que la clave privada será mantenida en privado por el usuario.

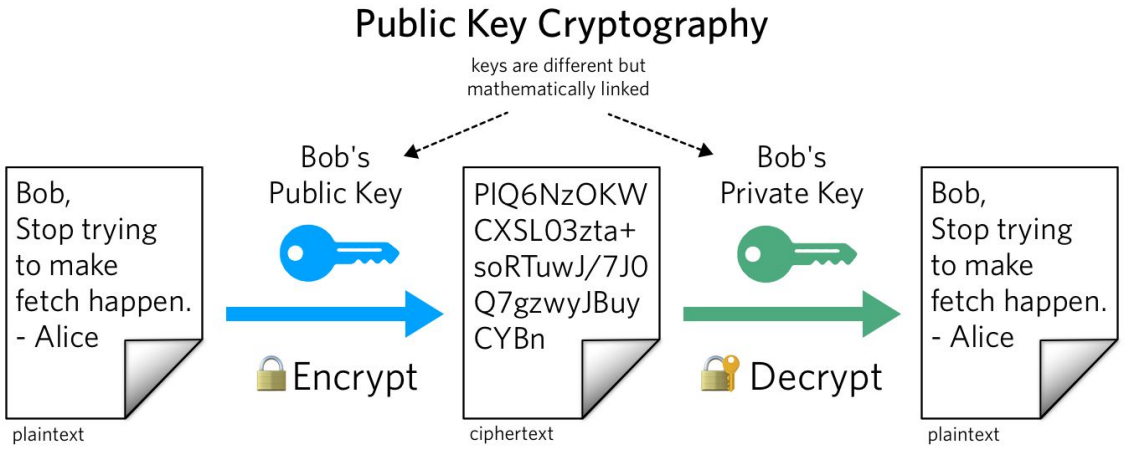

Figura 2.2: Ejemplo de cifrado y descifrado de un mensaje mediante criptografía asimétrica

Utilizando la clave pública, una persona puede encriptar un mensaje que únicamente podrá ser desencriptado utilizando la clave privada de esa persona. Mientras que mediante la clave privada puede encriptar un mensaje, que puede desencriptar con la clave pública asociada, a esto llamamos firma digital. La firma digital asegura que cualquier persona que posea la clave pública puede verificar que el mensaje fue creado por el propietario de la clave privada y no ha sido modificado.

En general, en blockchain la dirección del usuario ejerce como clave pública y la clave privada debe de quedar a buen recaudo en el wallet del propietario. La dirección se asemeja a un número de cuenta en un banco, debe de ser conocida por todas las personas que quieran realizarnos una transferencia, y quedará registrada como cuenta emisora cuando se realice una transferencia desde dicha cuenta. Por otro lado, la clave privada podemos compararla con la contraseña de la cuenta bancaria, por lo tanto, solo la persona que posea la clave privada podrá acceder a los fondos de la cuenta.

Los algoritmos de clave pública más utilizados son:

- RSA: se trata de un algoritmo desarrollado en 1977. Dada su sencillez para comprenderlo y utilizarlo se trata del algoritmo más utilizado de este tipo. La seguridad del mismo radica en el problema de factorización de grandes números, el par de claves se calculan a partir de un número que se obtiene como producto de dos números primos grandes. Se trata de un algoritmo bastante seguro conceptualmente, aunque existen algunas vulnerabilidades que pueden ser aprovechadas por un atacante.

- Diffie-Hellman: este algoritmo está basado en el problema homónimo, que se basa en acordar una clave común para dos participantes en un canal de comunicación no seguro [13]. En este sistema no son necesarias claves públicas propiamente dichas, sino, una información compartida por ambas partes. La seguridad de este algoritmo se halla en la dificultad de calcular logaritmos discretos en un cuerpo finito.

- Curva elíptica: los algoritmos de curva elíptica fueron propuestos como un nuevo método de criptografía asimétrica en los años 80 [14]. Apoyándose en las matemáticas de curva elípticas se consiguen claves más pequeñas con un nivel de seguridad equiparable a los algoritmos anteriores.

2.1.5.2 Funciones hash

Una función *hash* o función resumen es una función que asigna a un mensaje de longitud variable 'x', una clave de longitud fija 'y' independientemente de la longitud de la variable de entrada, de manera que, f(x)= y. La salida de la función proporciona una secuencia de bits de longitud fija que va asociada al propio mensaje introducido. Para que la función sea realmente útil debe de cumplir las siguientes propiedades:

- Resistencia a colisiones: un buen algoritmo no debe producir dos resultados iguales de forma que f(x) ≠ f(z).

- Unidireccionalidad: calcular el resultado de una función hash debe de ser un proceso fácil, pero debe de ser tarea imposible calcular el mensaje original a partir del resultado de la función hash.

Figura 2.3: Ejemplo hashing

Existen multitud de funciones hash disponibles en la red y pueden elegirse en función de los requisitos de entrada o salida de la misma. Uno de los más conocidos en el ámbito de blockchain por su equilibrio entre seguridad y coste computacional es el algoritmo SHA-256, que pertenece a la familia de funciones hash SHA-2. Este algoritmo independientemente de la longitud del mensaje de entrada, el resultado obtenido siempre es una cadena de 256 bits.

El hash es utilizado por los nodos para acordar que la red no ha sido modificada voluntariamente tras los cambios naturales que se van produciendo. Este cálculo es llevado a cabo aplicando mediante una función hash utilizando como entrada el estado completo de la red con todas las transacciones que se han realizado hasta ese momento y la cadena de 256 bits obtenida representa el estado de la misma.

2.1.5.3 Árboles de Merkle

Un árbol de merkle [15] también llamado árbol hash es una estructura de datos en árbol, en el que cada nodo que no es una hoja está etiquetado con el hash de la concatenación de etiquetas o valores de sus nodos hijo. Esto permite que gran cantidad de datos puedan estar ligados a un único valor de hash, el hash del nodo raíz del árbol, de manera que puedan ser verificados de una forma más eficiente.

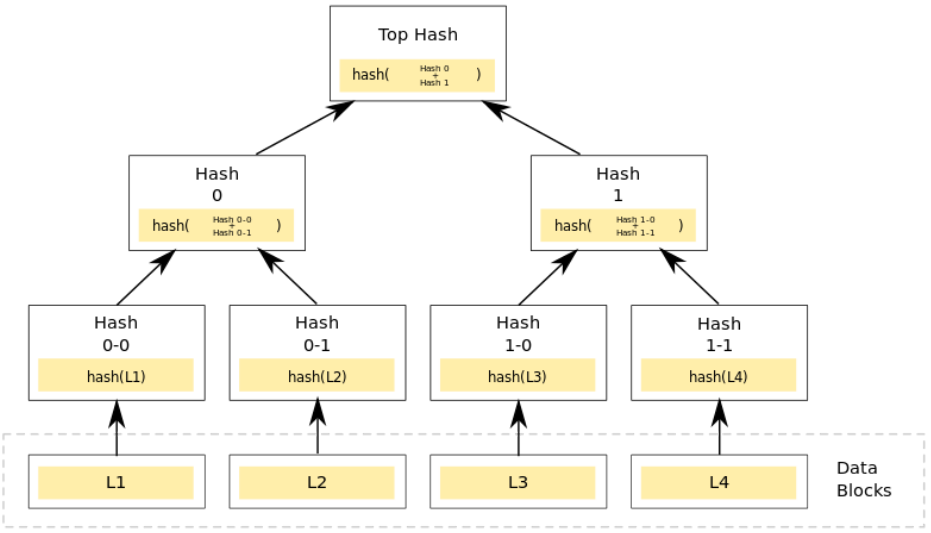

Figura 2.4: Ejemplo de árbol de Merkle

En la red Ethereum, todos los bloques son representados como un árbol de Merkle para poder agrupar todas las transacciones en un bloque, por lo tanto todas las transacciones van a estar relacionadas con un valor único de hash, el del nodo raíz, que irá firmado para asegurar que no ha sido modificado. Por lo que si se quisiera modificar una transacción en cualquiera de las hojas del árbol, se modificaría también el hash raíz dado que el hash sería diferente, y el protocolo detectaría que ese bloque es diferente y ha sido modificado y se invalidará.

2.1.6 Tipos de blockchain

Con la llegada de Bitcoin, descubrimos la blockchain pública y al alcance de todos. Con el paso del tiempo, tanto empresas como gobiernos han puesto el punto de mira en esta tecnología para poder usarlas en sus propios proyectos, pero como podemos imaginar los intereses de estos son distintos a los de las comunidades abiertas. Fue así como nacieron las blockchain privadas e híbridas.

A continuación se describirán las principales características de las redes blockchain en función del nivel de confianza en la misma.

Redes públicas

Las redes blockchain públicas permiten que cualquier persona pueda formar parte del sistema sin restricción alguna. Además el funcionamiento de la red es totalmente transparente y abierto, los datos desde el inicio de la red están abiertos para todos los participantes. Esto permite que la red pueda ser revisada y auditada en cualquier momento por cualquier participante.

En las redes públicas no existe una entidad central que regule su funcionamiento, son completamente distribuidas, por lo tanto, ningún nodo del sistema se considera de confianza. Estas redes usan algoritmos de consenso como PoW o PoS que basan su funcionamiento en la minería y el cobro de comisiones por cada transacción que se realice dentro de la red.

Un ejemplo de redes públicas son Bitcoin, Ethereum, Dash o Zcash.

Redes privadas

Este tipo de redes también conocidas como permisionadas, cuentan con los mismos elementos que una red blockchain pública pero a diferencia de estas, dependen de una unidad central que controla las acciones que se realizan en la red y restringen el acceso a personas con autorización. El acceso al libro de transacciones es privado y únicamente puede consultarlo la entidad propietaria de la red.

El rol de autoridad central propietaria de la red en la mayoría de los casos es tomado por empresas o gobiernos que confían en esta tecnología para sus proyectos, por lo tanto, el mantenimiento económico de la blockchain depende también de las empresas que sostienen el proyecto.

Al ser redes donde no todo el mundo tiene acceso, la confianza entre los participantes es mayor que en las redes públicas, por lo tanto, los algoritmos de consenso empleados son más rápidos y eficientes que los que no utilizan minería.

En el ámbito de las redes permisionadas destacan: Hyperledger, perteneciente a la fundación Linux, Corda propiedad de R3 o Quorum de JPMorgan.

Redes híbridas

Este tipo de redes aúnan características de las blockchains públicas y privadas en un intento de aprovechar lo mejor de cada una. Para acceder a la red es necesario la autorización de una autoridad central. Pero una vez dentro de la red es posible realizar tanto transacciones públicas como restringidas únicamente a ciertos participantes de la red. Se trata de una solución que sirve para que intercambien información y establezcan redes business-to-business (B2B) entre ellas.

Son ejemplos de redes híbridas: BigChainDB o Alastria.

2.2 Blockchain y el intercambio de valor

2.2.1 Internet del valor

Internet tal y como lo conocemos hoy en día tuvo su origen en los años 60 en Estados Unidos, y en las últimas dos décadas ha sufrido un crecimiento exponencial gracias a las nuevas infraestructuras, este internet es conocido como i*nternet de la información*. Se trata del medio de información y comunicación donde la información es libre, accesible y replicable para todos. Sin embargo, con el paso de los años se ha convertido en un internet centralizado, donde grandes compañías como Google, Amazon o Facebook han acaparado la mayor parte de la actividad que se genera en internet.

En el marco de internet de la información es necesaria la presencia de una autoridad central y de confianza, como puede ser un banco o un mercado, para comprar, vender o intercambiar activos. Los activos representan los derechos que una entidad posee sobre bienes y servicios que pueden ser objeto de comercio.

Con la llegada del blockchain surge también un novedoso y complejo concepto conocido como *internet del valor* [16], que se diferencia del paradigma de internet de la información en que se permite intercambiar valor sin la necesidad de terceros de confianza que administren los niveles de fiabilidad de las operaciones. El internet del valor es sostenido en la red por los usuarios y la tecnología, de forma consensuada y distribuida, y permite la transferencia de cualquier tipo de activo.

El internet del valor que nos trae blockchain se define a través de tokens o activos digitales, de tal forma que, el valor del activo se distribuirá por la red tal y como lo hace la información.

2.2.2 ¿Qué es un token?

Según William Mougayar en su libro *The business blockchain*, un token es "*una unidad de valor que una organización crea para gobernar su modelo de negocio y dar más poder a sus usuarios para interactuar con sus productos, al tiempo que facilita la distribución y reparto de beneficios entre todos sus accionistas*" [17].

Un token como se ha indicado en la definición, es una unidad de valor, que normalmente es emitido por proyectos de emprendimiento basados en blockchain, para los que puede representar un activo o tener otra utilidad en particular. El valor de un token puede representar a cualquier activo, y es aceptado por todos los miembros de la comunidad blockchain a la que pertenece. Llamaremos tokenizar a la acción de representar un activo en una red blockchain.

Los tokens son creados sobre el protocolo de una criptomoneda existente en una blockchain particular. Las criptomonedas se diferencia de los tokens en que estas son monedas digitales nativas y propias de su blockchain, como por ejemplo, Bitcoin (BTC) o Ether (ETH) [18]. Todas las criptomonedas existen en sus propios libros de contabilidad independiente: BTC opera en la blockchain de Bitcoin o ETH en la red de Ethereum. Por lo tanto, un token no es una criptomoneda, el token corre por encima de la blockchain a nivel de aplicación, y es representado a partir de un contrato inteligente [19].

Los tokens pueden dividirse en tres categorías:

Security token

Se trata de un tipo de token digital que está vinculado a los valores financieros tradicionales, entendiendo por tradicional cualquier activo intercambiable como los bonos, los swaps o los futuros.

Este tipo de tokens reflejan valores del mundo real, por lo tanto, los países aplican sus leyes sobre este tipo de cripto activos, esta situación ha elevado mucho el interés por ellos debido a la seguridad que ofrecen tanto a nivel tecnológico como legal.

Utility token

Estos tokens permiten a todo aquel que los posea acceder a productos y servicios prestados por una entidad privada, pero a diferencia de los security token, estos no están regulados legalmente.

Estos tokens fueron muy utilizados por diversos proyectos criptográficos durante el año 2017 en el denominado *boom de las ICOs.* Una *initial coin offering* (ICO) u *oferta inicial de monedas* es una “preventa de tokens” en la que las personas compran tokens a una organización con el objetivo de financiar nuevos proyectos de la compañía. Estos utility tokens se podrán usar cuando la organización ponga en marcha su proyecto.

Equity token

Los equity token están muy relacionados con los security token, ya que, funcionan como una representación digital de un activo de acciones tradicional. Además son también tokens que se encuentran regulados por las leyes de valores de muchos países pero a diferencia de los security tokens las leyes son más flexibles y eso ha despertado el interés de empresas que prefieren utilizarlos en lugar de los security tokens.

2.2.3 Token ERC20

Debido al gran número de tokens que es posible crear en el ecosistema Ethereum, y ante la necesidad de poder interactuar con otros tokens, importarlos a una billetera o realizar operaciones con ellos. Surgió la idea de crear un estándar para que los desarrolladores pudieran crear tokens compatibles de forma fácil y auditada.

Es por ello que, el token ERC20 [20] fue creado por la comunidad de Ethereum en noviembre de 2015. Los tokens ERC20 son utilizados para representar y gestionar tokens fungibles que pueden ser divisibles, y como todo token de Ethereum son contratos inteligentes que se ejecutan en la red blockchain. Este estándar aporta un lenguaje universal que utilizan todos los tokens Ethereum y permite que token sea negociado con otro. El valor de cada token dependerá de la cantidad, de la red y de la confianza que es generada por el token.

Para ello ERC20 define una lista común de reglas que deberán de seguir todos los tokens que se amparen en este estándar. En la práctica, estas reglas se traducen en funciones que deberán ser implementadas en el contrato inteligente. Son los siguientes:

Métodos

- *Name* (opcional) – Nombre del token.
- *Symbol* (opcional) – Símbolo del token.
- *Decimals* (opcional) – Número de decimales que utiliza el token.
- *TotalSupply* – Suministro total de tokens que existirán.
- *BalanceOf* – Saldo de la cuenta del propietario.
- *Transfer* – Transferencia a una cuenta.
- *TransferFrom* – Transferencia desde una cuenta a otra.
- *Approve* – Permite la retirada de fondos de una cuenta.
- *Allowance* – Devuelve la cantidad de fondos que se pueden retirar.

Eventos

- *Transfer* – Activado cuando se realiza una transferencia de tokens.
- *Approval* – Activado siempre que se aprueba una transferencia.

Además de los métodos estándar que se acaban de enumerar, los tokens ERC20 permiten la creación de métodos propios para una mayor personalización del mismo.

2.3 Plataformas blockchain

2.3.1 Contratos inteligentes

Un contrato inteligente es un software, que a diferencia del software tradicional este es inscrito en una blockchain, de manera que no puede ser modificado ni eliminado. Este software ejecuta de forma automática acuerdos que hayan sido determinados por dos o más partes. Las cláusulas del contrato son programadas con anterioridad con la capacidad de autoejecutarse en función del cumplimiento de lo estipulado.

A diferencia de los contratos convencionales que están escritos en lenguaje natural, los contratos inteligentes están escritos en un lenguaje de programación, de manera que se evitan las posibles malinterpretaciones que cada participante pueda realizar.

Sus principales características son:

- Autonomía: facilitan el consenso entre varias entidades sin intermediarios que validen el acuerdo, reduciéndose así el número de agentes externos que se implican en el contrato.

- Coste: los costes de este tipo de contratos son más reducidos que los del paradigma tradicional, ya que se requiere una menor intervención humana.

- Confianza: el uso de la tecnología blockchain permite que los contratos inteligentes sean encriptados y solo puedan ser descifrados por los usuarios que formen parte del acuerdo. Además al quedar registrado en la cadena de bloques será inmutable y deberá de ser consensuado por los nodos.

- Velocidad: el uso de software automatiza tareas que, si se realizaran de forma tradicional serían menos eficientes y se correría el riesgo de que se produjeran errores en el proceso.

Además, los contratos inteligentes también pueden obtener información exterior a su red utilizando oráculos. Los oráculos son programas que actúan como un tercero, intermediando entre las partes involucradas en el contrato.

Para desplegar un contrato las diversas plataformas blockchain utilizan un lenguaje distinto. Ethereum y Quorum llaman a sus contratos *Smart Contracts* y son desarrollados con el lenguaje *Solidity*. Por otro lado, en la plataforma Hyperledger son conocidos como Chaincode y pueden ser desarrollados con JavaScript, Java o Go. Sin embargo, Corda utiliza Kotlin como lenguaje de programación para sus contratos.

2.3.2 Ethereum

Ethereum [21] es una plataforma de código abierto y descentralizada que provee una infraestructura para la ejecución de Smart Contracts y permite a los desarrolladores la creación de aplicaciones descentralizadas, también conocidas como *dapps*.

El objetivo inicial de este proyecto era el de descentralizar la web introduciendo cuatro componentes como para de su hoja de ruta hacia la red 3.0: mensajes dinámicos, transacciones confiables, publicación de contenido estático y una interfaz de usuario integrada y funcional. Estos componentes estaban pensados para sustituir la experiencia web actual, pero haciéndolo de manera descentralizada y anónima [22].

Figura 2.5: Logo de Ethereum

En el núcleo de Ethereum se encuentra la *Ethereum Virtual Machine* (EVM) que permite ejecutar código sin importar la complejidad. Ethereum se considera un sistema Turing completo, y se refiere a la capacidad que tiene el lenguaje de programación de Ethereum, Solidity, para poder resolver cualquier tipo de problema computacional y posibilitar el uso de estructuras complejas como los bucles.

Como se ha indicado con anterioridad, Ethereum es código abierto pero esto no implica que su uso sea gratuito. Cada operación en la red de Ethereum requiere una cantidad de *gas* que será calculada en función del coste computacional y tiempo de ejecución asociado. El gas no debe entenderse como un medio de pago o una moneda sino como el "combustible" para las aplicaciones descentralizadas. El gas se paga en Ether, pero un usuario no posee gas, ya que este se calcula al realizarse la operación.

El Ether [23] es la criptomoneda nativa de la plataforma Ethereum y se trata de un activo digital empleado para que puedan ejecutarse tanto aplicaciones descentralizadas como contratos inteligentes y para compensar a los mineros que aseguran las transacciones.

Ethereum consigue procesar una transacción en un tiempo aproximado de 16 segundos, calculando la dificultad de minado en cada bloque, de manera que se va ajustando a las necesidades y requerimientos de la red para intentar alcanzar siempre la misma velocidad. Actualmente Ethereum valida las transacciones utilizando el algoritmo de consenso Proof of Work [24] pero está previsto que, con el lanzamiento de Ethereum 2.0 el protocolo de consenso empleado sea Proof of Stake.

2.3.3 Quorum

Quorum [25] es un protocolo blockchain de código abierto basado en Ethereum y orientado a entornos empresariales, que incluye un pequeño *fork* del cliente *Go Ethereum*, también conocido como Geth y permite la privacidad en transacciones y contratos inteligentes. Quorum es mantenido y desarrollado por la empresa J.P. Morgan. Esta plataforma aprovecha el trabajo que ya han realizado los desarrolladores de Ethereum, de manera que se garantiza el uso de un entorno que ha sido probado desde su nacimiento en el año 2015.

Figura 2.6: Logo de Quorum

Las principales características de Quorum, y en las que difiere de la red pública de Ethereum son:

- Privacidad de transacciones y contratos inteligentes.
- Múltiples mecanismos de consenso basados en votaciones, que no producen bifurcaciones en la cadena.
- Gestión de permisionado en la red y nodos.
- Mejora en el rendimiento.

Estas características hacen de Quorum una plataforma que incluye todas las innovaciones de la comunidad pública de Ethereum con mejoras que la hacen perfecta para su uso empresarial.

También cabe destacar el potencial de la comunidad de desarrolladores de Quorum. Una comunidad formada por personas de todo el mundo que utilizan la plataforma Slack como punto de encuentro donde se informa de todas las novedades relacionadas con Quorum y se resuelven dudas técnicas que los usuarios exponen en los distintos canales de la plataforma. Para la realización de este trabajo me ha sido de gran ayuda la información que me han ofrecido en esta comunidad.

A continuación, detallaremos la arquitectura, los algoritmos de consenso y el ciclo de vida de las transacciones en Quorum.

2.3.3.1 Arquitectura

Quorum está formado por el nodo Quorum y el gestor de privacidad [26].

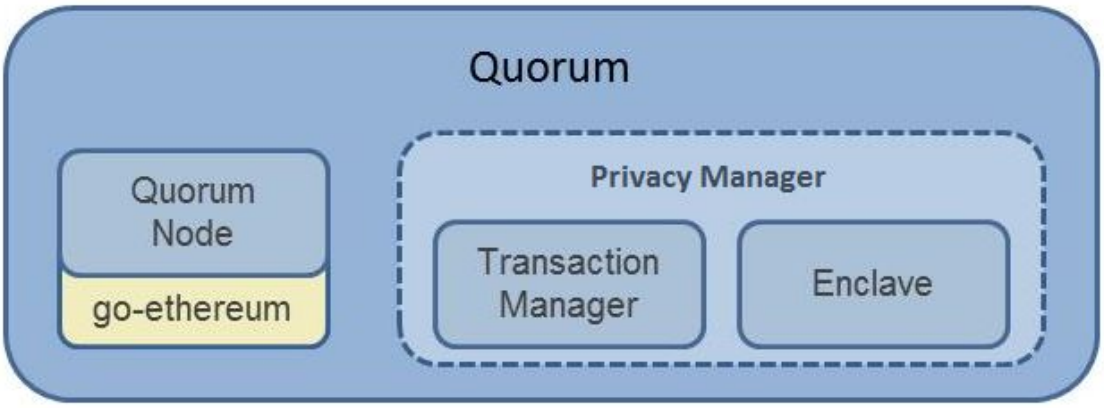

Figura 2.7: Arquitectura de Quorum

Nodo Quorum

Este nodo está diseñado mediante una pequeña modificación de Go-Ethereum para poder aprovechar las modificaciones en I+D realizadas por la comunidad de Ethereum y que está en constante crecimiento. El nodo Quorum incluye las siguientes modificaciones con respecto a Go-Ethereum.

- El consenso se alcanza con los algoritmos Raft e IBFT en lugar de utilizar Proof-of-Work.
- La capa P2P ha sido modificada y permite conexiones entre nodos autorizados.
- La lógica de generación y validación de bloques ha sido modificada para poder manejar transacciones privadas.
- El árbol que almacena los estados ha sido dividido en dos, uno público y otro privado.
- La creación de transacciones se ha modificado para permitir que los datos de las transacciones sean reemplazados por hash cifrados y así preservar los datos privados cuando sean necesarios.
- Se ha eliminado el precio del gas, aunque el concepto de gas sigue existiendo a precio 0.

Privacy manager

Constellation y Tessera son sistemas implementados por Haskell y Java de propósito general, utilizados para el envío de información segura. No son específicos de blockchain y son aplicables en multitud de aplicaciones, donde se necesita un intercambio de mensajes cifrados individualmente dentro de una red.

Los módulos Constellation y Tessera constan de dos submódulos, el Transaction Manager y el Enclave.

El Transaction Manager es el responsable de la privacidad de las transacciones, ya que almacena y permite el acceso a datos de transacciones cifradas e intercambia información cifrada con los Transaction Managers de otros participantes, pero no tiene acceso a ninguna clave privada confidencial.

Para mejorar el rendimiento y especialmente la seguridad, Enclave se encarga de la mayor parte de los procesos criptográficos. Trabaja estrechamente relacionado con el Transaction Manager y se encarga por ejemplo, de la generación de claves simétricas. Es el bloque encargado de reforzar la seguridad realizando la encriptación/desencriptación de una manera aislada del resto del proceso.

Actualmente, se recomienda el uso de Tessera debido a que es más estable , provoca menos errores y Constellation apenas recibe nuevas actualizaciones.

2.3.3.2 Algoritmos de consenso

Actualmente, Quorum trabaja con dos algoritmos de consenso distintos, Raft e Istanbul-BFT, que suponen una gran alternativa al algoritmo Proof-of-Work utilizado por Ethereum.

Raft

El algoritmo de consenso Raf [27] es útil para redes en las que no sea necesaria la tolerancia a fallos bizantinos y que prime la velocidad ya que se confirman las transacciones con velocidades muy cercanas a los milisegundos.

Los nodos que participan en este algoritmo, pueden estar en tres estados:

- Líder: Todos los cambios que se realicen en el clúster de nodos pasan por él primero. Como máximo puede haber uno en cada momento. Si se cae el líder actual, se abre un proceso para elegir al siguiente.

- Candidato: Nodo que no ha encontrado líder y solicita su elección como tal. El líder será seleccionado entre los distintos candidatos. Una vez que un nuevo líder ha sido elegido se dice que comienza un término.

- Seguidor: Nodo cuya responsabilidad es actuar frente a las peticiones del nodo líder.

Para explicar la elección de un nuevo líder supongamos que actualmente, todos los nodos son seguidores y esperan recibir la comunicación de un nodo líder. Cada nodo tiene un tiempo de espera después del cual, si un líder no se comunica con él, pasa a ser candidato y pide ser elegido líder.

Para ello, el algoritmo divide el tiempo en términos que son plazos de tiempo definidos por el algoritmo. Un nodo que quiere ser líder debe recibir la mayoría de votos de los seguidores en un término. Si existen N nodos la mayoría deben ser al menos (N/2) + 1 nodos.

Si un nodo candidato no recibe los votos necesarios y agota su tiempo de espera, espera el siguiente término y se repite el proceso de votación. Una vez elegido el líder cada cierto tiempo debe enviar mensaje, ya sea con información de actualización o no, a fin de que los nodos seguidores no agoten su tiempo de espera.

El ciclo de vida de una transacción mediante Raft es el siguiente:

1. La transacción se envía mediante una llamada RPC a Geth a través de la capa de transporte de Ethereum P2P.

2. La transacción será recibida por el nodo líder. El bloque con todas las transacciones recibidas será minado y se disparará un evento que anuncie a Raft que hay un nuevo bloque y debe de ser propuesto.

3. El bloque será enviado a través de la capa HTTP del protocolo Raft y se convertirá en el principio de la blockchain de todos los nodos.

4. Si existe un consenso, será enviado al resto de nodos.

5. La transacción ha llegado a todos los nodos y Raft garantiza que no se producirán bifurcaciones en la cadena.

Por defecto, un nodo es minado cada 50ms. Cuando llega una nueva transacción se mina un nuevo bloque inmediatamente si han pasado más de 50 ms desde el último bloque para no saturar y conseguir equilibrio entre rendimiento y latencia.

Istanbul BFT

Istanbul Byzantine Fault Tolerant (IBFT) [28] es una modificación del algoritmo original PBFT para poder trabajar con las cadenas de bloques. Istanbul BFT consta de tres fases para el consenso: *pre-prepare, prepare y commit.* El sistema puede tolerar, como mucho, un número F de fallos en los nodos de manera que la red debería estar formada por N=3F+1.

Antes de cada ronda los nodos validadores elegirán uno de ellos como *proposer*, por defecto, siguiendo el método round-robin por defecto. El nodo elegido hará una propuesta de bloque y mandará esta con el mensaje “pre-prepare”. Después se mandará el mensaje “prepare”.

Este paso sirve para verificar que todos los nodos validadores están en funcionamiento y trabajando en la misma ronda. Si se reciben 2F+1 mensajes de preparado, las validadores entran en la fase de preparados y lanza a toda la red el mensaje de “commit”. Este paso sirve para informar a los peers de que se acepta el bloque propuesto por el *proposer* y que va a introducir el bloque en la cadena. Por último, los validadores esperan a que les lleguen 2F+1 de mensajes “commit” para entrar en el estado ‘committed’ y añadir el bloque a la cadena.

En los bloques generados con el protocolo IBFT son finales, lo que significa que no pueden existir bifurcaciones y todos deben estar en la cadena principal. Para evitar que un nodo cree una cadena diferente, cada validador adjunta 2F+1 mensajes “commit” firmados en la cabecera del bloque antes de insertarlo en la cadena. Sin embargo, esto puede generar un problema en el cálculo del *hash,* ya que cada bloque puede añadir un número diferente de mensajes. Por eso, el *hash* se realiza sin contar los mensajes “commit” para asegurar la consistencia del bloque.

Los cambios de ronda son debidos a tres situaciones diferentes:

- El contador de cambio de ronda expira.
- Si se produce un mensaje no válido de “pre-prepare”.
- Una inserción de un bloque fallida.

Si un nodo validador detecta alguna situación de las anteriormente mencionadas lanzará un mensaje a toda la red pidiendo un cambio de ronda incluyendo el número de ronda propuesto.

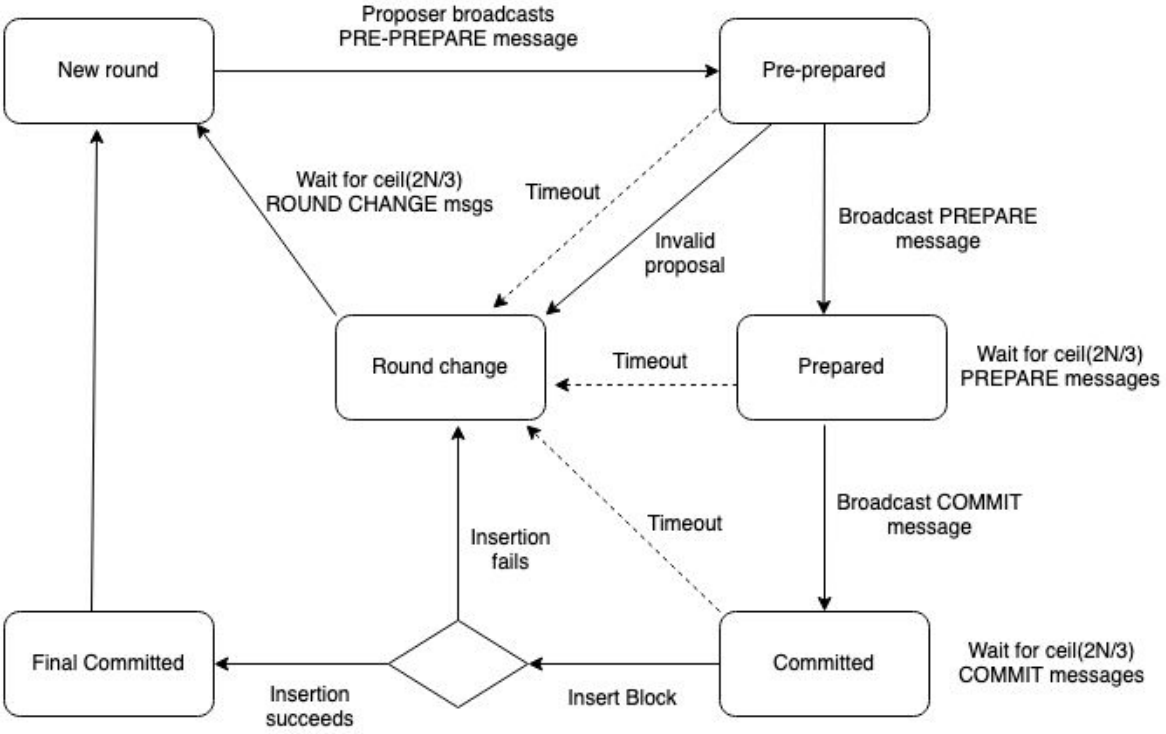

Figura 2.8: Máquina de estados. Algoritmo IBFT.

2.3.3.3 Transacciones

En la red de Quorum pueden realizarse transacciones que serán públicas o privadas [29].

Las transacciones públicas serán visibles para todos los miembros de la red, ya que son creadas a imagen y semejanza de las transacciones estándar de Ethereum. Las transacciones públicas se ejecutan de la forma estándar de Ethereum, por lo que si una transacción pública se envía a una cuenta que contiene un contrato inteligente, cada participante ejecutará el mismo código y sus bases de datos subyacentes se actualizarán en consecuencia.

Sin embargo, las transacciones privadas solo serán visibles para los participantes de la red cuyas claves públicas se especifican en el parámetro *privateFor* de la transacción. Cuando un nodo Quorum encuentra una transacción con un *privateFor* nulo, establece el valor de la firma de la transacción en 37 o 38, a diferencia de 27 o 28 que son los valores utilizados para indicar que una transacción es pública según el estándar Ethereum. Sin embargo, una transacción privada no cumple con el estándar Ethereum. Antes de que se propague la transacción al resto de nodos, se reemplaza la carga de la transacción original, es decir, su información con un hash del contenido encriptado, que proporciona el módulo de Constellation. Aquellos nodos que sean parte de la transacción serán capaces de recuperar la información original a través de Constellation, mientras que los nodos no participantes solo podrán visualizar el hash. Esto resulta en que, si la transacción se envía a una cuenta que contenga código, esos participantes de la red que no sean parte de la transacción simplemente no ejecutarán el código, lo obviaron. Los participantes de la transacción reemplazarán el hash con la información original antes de llamar a la máquina virtual de Ethereum (EMV) para la ejecución de su código. En este momento se podría pensar que esto provocaría un error en la cadena, puesto que algunos nodos tendrían una información diferente y no serían capaces de llegar a un consenso. Para evitar este problema, Quorum guarda el estado de los contratos en un *Public State Trie* que está sincronizado de manera global, y un *Private State Trie* que no se sincroniza de forma global.

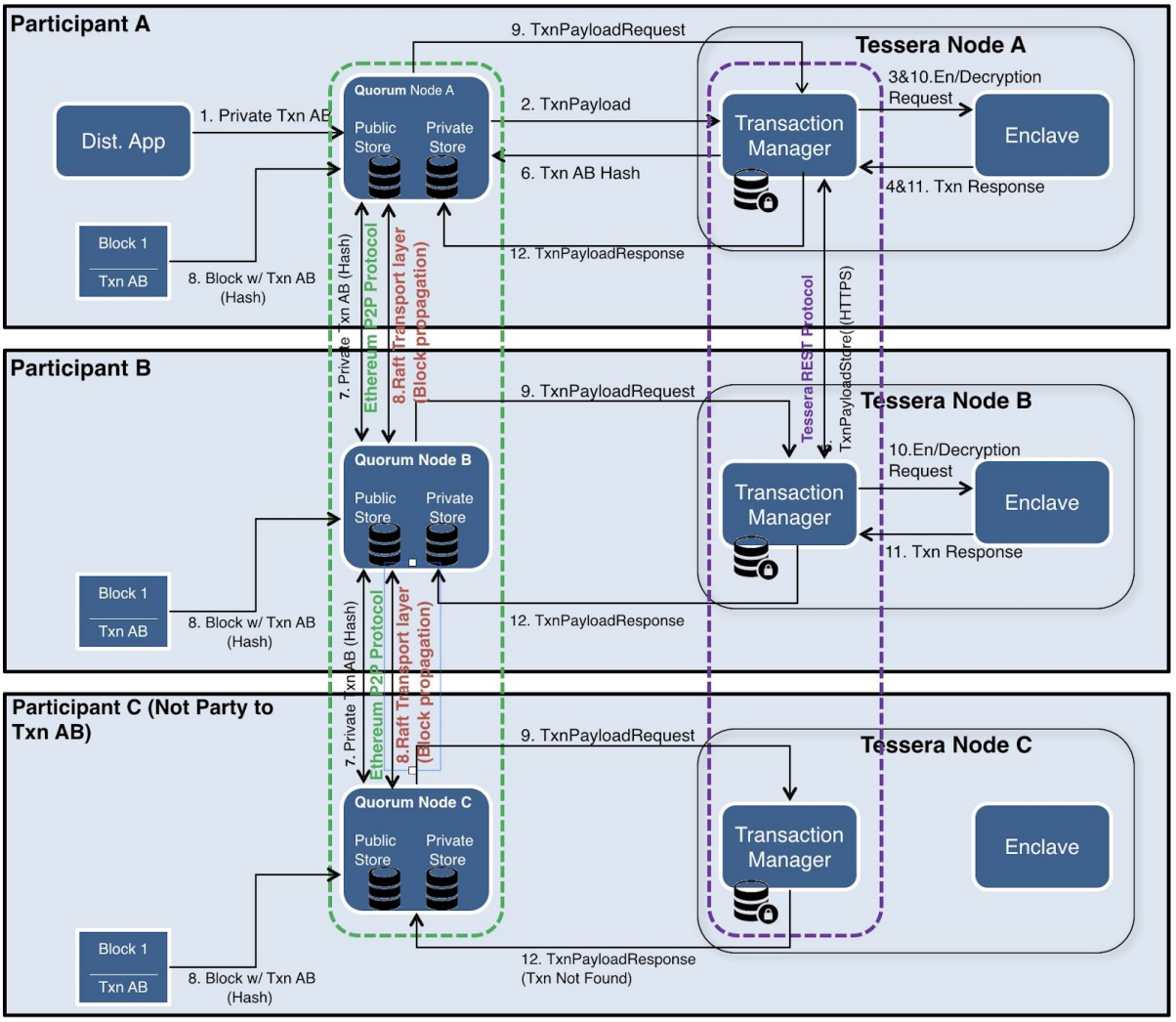

Figura 2.9: Ciclo de vida de una transacción privada en Quorum

A continuación, se describe el ciclo de vida de una transacción privada en Quorum [30]. En la imagen anterior, el participante A y el participante B son parte de la transacción AB, mientras que el participante C queda excluido.

1. El participante A envía la transacción a su nodo Quorum, indicando la información de la transacción y configurando el *privateFor* con la clave pública de los participantes A y B. La clave pública del emisor es opcional.

2. El nodo de Quorum del participante A enviará la transacción a su *Transaction Manager* para que este la cifre y almacene antes de enviarla a los destinatarios.

3. El Transaction Manager de A realizará una llamada a su enclave asociado para cifrar la información de los destinatarios.

4. El enclave de A cifra la información de la transacción privada generando una clave simétrica (llamada tx-key) y dos *nonces*. La información de la transacción será cifrada con la clave tx-key y un nonce. La tx-key será cifrada por separado para cada destinatario utilizando el segundo *nonce* mediante una clave compartida formada por la clave privada del emisor A, y la clave pública del destinatario B.

5. El Transaction Manager de A almacenará la respuesta del enclave y la reenviará a los destinatarios de la transacción.

6. Después el Transaction Manager de A devuelve el hash de la información cifrada al nodo Quorum. El nodo reemplaza el parámetro *data* del campo de la transacción por este *hash*, y cambia el valor *V* a 37 o 38. De esta forma la transacción quedará marcada como privada.

7. La transacción será propagada al resto de la red utilizando el protocolo geth P2P.

8. Se crea un bloque que contiene la transacción AB y se distribuye a cada nodo Quorum de la red.

9. Al procesar el bloque, todos los participantes intentarán procesar la transacción. Comprobando que el campo *data* es un *hash* y *V* tiene el valor de 37 o 38, cada nodo hará una llamada a su Transaction Manager para determinar si es parte de la transacción (es decir, hay una entrada para el hash dado en su base de datos).

10. Los Transaction Manager de los participantes A y B hacen una llamada a sus enclaves asociados para descifrar la información.

11. El Transaction Manager de A y B devolverá a sus nodos Quorum los datos de la transacción privada descifrada para poder ejecutarla con normalidad, actualizando así sus respectivas bases de datos. En el caso de C, su Transaction Manager devolverá un error 404 NOT FOUND, ya que no es destinatario de la transacción, por lo que el nodo Quorum omitirá la ejecución de la transacción y no se realizarán cambios en su base de datos

2.3.4 Hyperledger Fabric

Hyperledger [31] es una tecnología de código abierto, colaborativa que promueve proyectos bajo la tecnología blockchain y nace en el año 2015 bajo la tutela de la Linux Fundation. El objetivo de Hyperledger es desarrollar estándares y protocolos abiertos, en los que se incluirán variedad de redes blockchain con diferentes consensos, sistemas de almacenamiento, etc. Dentro del marco Hyperledger existen los siguientes frameworks: Fabric, Burrow, Indy, Iroha, Sawtooth o Grid. Por otra parte, existen herramientas que intentan abstraerse del framework utilizado e intentan ser independientes de la red que se utiliza debajo, destacan: Caliper, Cello, Composer, etc.

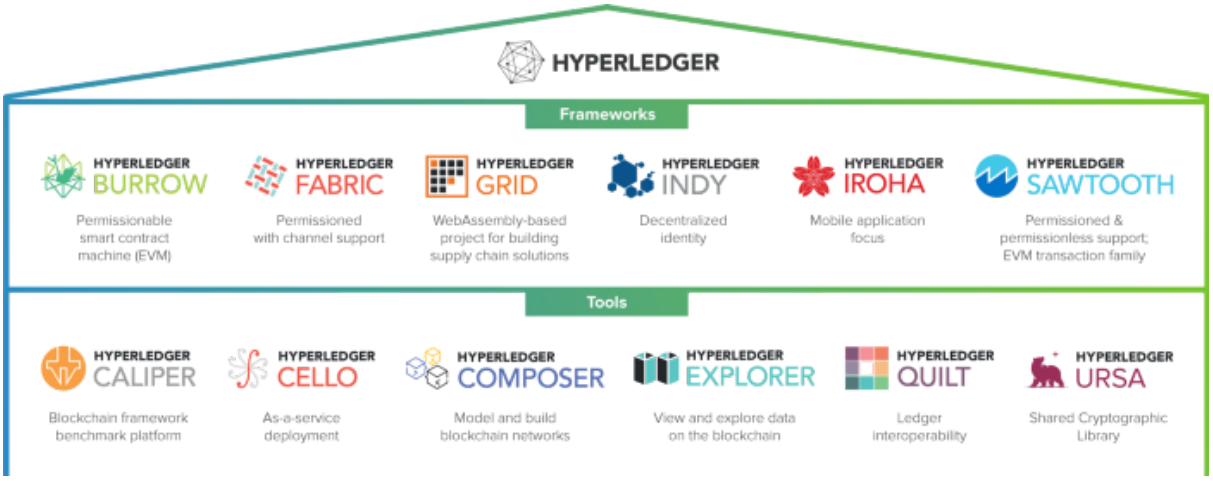

Figura 2.10: Ecosistema Hyperledger

Dentro del ecosistema Hyperledger nos centraremos en el framework Hyperledger Fabric. Fabric, como todas las demás plataformas de Hyperledger, es una plataforma de código abierto diseñada para su uso en un contexto empresarial [31].

Se trata de una red permisionada, donde todos los participantes deben ser identificables, dejando de lado el anonimato de las redes blockchain públicas. La arquitectura de Fabric es modular y configurable, de modo que permite adaptarse a casos de uso de banca, finanzas, seguros o logística. Los contratos inteligentes en esta plataforma son conocidos como *ChainCodes* y están escritos en lenguajes de programación más generales como Go, Java o Javascript. Fabric no hace uso de una criptomoneda de forma nativa de manera que se reduce significativamente el riesgo de un ataque al no tener que pagar por realizar una transacción. Tampoco existe el concepto de minado.

Mediante el uso de canales Fabric permite que coexistan en la misma red información pública y privada. Los canales privados proporcionan privacidad y confidencialidad a usuarios específicos de la red. Para ello, los participantes pueden establecer canales entre sí y garantizar que las transacciones realizadas en estos canales solo son conocidas por los usuarios del canal [32].

El consenso en Hyperledger Fabric está compuesto de tres fases: ejecución, orden y validación.

1. Ejecución: ejecuta una transacción y corrobora que es correcta, después la añade.
2. Orden: ordena las transacciones a través de un protocolo de consenso.
3. Validación: valida las transacciones contra una política de aprobación (endorsement) específica de la aplicación antes de escribirla en la blockchain.

En cada fase pueden usarse distintos algoritmos de consenso, entre los que destacan: Kafka, RBFT o Sumeragi.

2.3.5 Corda

Corda [33] es una plataforma de libro de contabilidad distribuida, diseñada específicamente para administrar los acuerdos financieros entre las instituciones financieras reguladas, que aunque bien pudiera ser considerado una blockchain, los creadores y ejecutivos del consorcio R3 han dejado en claro que no lo es, debido a que Corda no alcanza un consenso general entre todos los nodos.

Figura 2.11: Logo de Corda

Esta plataforma posee una estructura de registros distribuidos, pero el acceso a los mismos no es público, un tercero solo podrá acceder cuando las partes implicadas lo acepten .

Corda está inspirada en gran medida en las redes blockchain tradicionales pero añade beneficios que hacen que esta plataforma sea adecuada para muchos escenarios bancarios [34].

Entre las principales características de Corda se encuentran:

- Coordinación del flujo de trabajo entre empresas sin una entidad central.

- El consenso es logrado a nivel de acuerdos individuales entre empresas, no a nivel de sistema. Utiliza los algoritmos RBFT, Conectable y Raft.

- Las transacciones son validadas por las partes afectadas en dicha transacción, en lugar de un conjunto más amplio de validadores no relacionados. La red es capaz de validar hasta 170 transacciones por segundo.

- Los contratos inteligentes de Corda son desarrollados en Java y Kotlin.

- Corda dispone de una red de nodos autenticada, donde cada uno de ellos es un entorno de tiempo de ejecución de JVM que aloja los servicios de Corda y ejecuta aplicaciones conocidas como CorDapp´s.

- No dispone de una criptomoneda ni token asociado.

2.4 Estado del arte

2.4.1 Ripple

Ripple [35] es una tecnología que actúa como red de pago digital para transacciones financieras y como criptomoneda. Se trata de un protocolo de código abierto que permite la realización de transacciones rápidas y baratas. La criptomoneda de Ripple es conocida como XRP, pero también permite utilizar su plataforma RippleNet para que los usuarios puedan crear su propia moneda.

Ripple utiliza XRP como token para realizar transferencias de valor a través de su red. Su propósito es ser un mediador en el cambio de monedas, tanto para criptomonedas como para fiat.

Figura 2.12: Logo de Ripple

A diferencia de Bitcoin o Ethereum, Ripple se basa en blockchain y existe un registro público de transacciones diferenciadas, pero usa la tecnología blockchain como tal, si no que dispone de su propia tecnología llamada RPCA.

Por otro lado, RippleNet es una red de proveedores de pagos institucionales como bancos y empresas de servicios monetarios que utilizan soluciones desarrolladas por Ripple para proporcionar una experiencia sin complicaciones para enviar dinero a nivel mundial. La plataforma permite realizar pagos en cualquier divisa incluyendo Bitcoin y tener una comisión mínima de transacción interna de 0.00001$.

Ripple está especialmente pensado para realizar transacciones internacionales rápidas, ya que el tiempo promedio de una transacción en Ripple es de 4 segundos, mientras que una transacción bancaria internacional puede demorarse un par de días. Esto ha impulsado que los bancos muestren interés por esta tecnología y la han comenzado a integrar en sus ecosistemas. Bancos como Santander, Axis Bank, Yes Bank o Union Credit han depositado su confianza en Ripple.

Esta tecnología también es utilizada como cambio de divisas de baja comisión que no pueden ser convertidas directamente entre sí y para ello los bancos utilizan el dólar estadounidense como mediador. Ripple hace las veces de mediador pero con un coste mucho menor que el USD.

2.4.2 One Pay FX

De la unión de Ripple y Banco Santander surge One Pay FX [36], para facilitar las transacciones transfronterizas rápidas y transparentes. Esta aplicación es una plataforma internacional basada en Ripple y respaldada por Santander.

One Pay permite a los clientes conocer el importe exacto que llegará en la moneda del destinatario antes de confirmar la transacción, estableciendo el límite de las transferencias internacionales desde los 10€ hasta los 30.000€. Esta transacción no tiene comisiones para los clientes de banco Santander y puede realizarse desde la app móvil o la web del banco.

Figura 2.13: Logo de One Pay

Recientemente Santander ha expandido la lista de países entre los que pueden realizarse transferencias de dinero hasta un total de 19, entre los que se encuentran: Estados Unidos, Chile, Portugal, Reino Unido, Polonia España o Brasil entre otros.

La presidenta del banco Santander, Ana Botín, afirma que: "*One Pay FX puede llegar a manejar la mitad de las transferencias internacionales anuales del banco*" y añadió que "*este mecanismo debería aumentar a medida que el servicio se extienda a más territorios y clientes*" [37].

2.4.3 Iberpay

Los bancos españoles están apostando por la tecnología blockchain para el desarrollo de soluciones de pago. Iberpay es una compañía española encargada de gestionar el sistema español de pagos al por menor, conocido como Sistema Nacional de Compensación Electrónica (SNCE) [38]. El Banco de España es el responsable de aprobar las normas del sistema y llevar a cabo su vigilancia.

Iberpay ha sido la encargada de coordinar una iniciativa llevada a cabo por las entidades Banco Sabadell, Banco Santander, Bankia, BBVA y CaixaBank [39]. Esta iniciativa consiste en la realización de una prueba de concepto sectorial para desplegar una plataforma interbancaria blockchain de pagos.

El objetivo de la prueba es la realización de transferencias inmediatas de smart contracts desplegados en una red blockchain para ejecutar y programar todo tipo de pagos automáticos: pagos a la firma de un contrato, a la entrega de mercancías o cuando se cumplan diversas condiciones. De forma que se garantice la seguridad, eficiencia, trazabilidad e integridad de las transacciones, cumpliendo estrictamente toda la normativa vigente en materia de pagos.

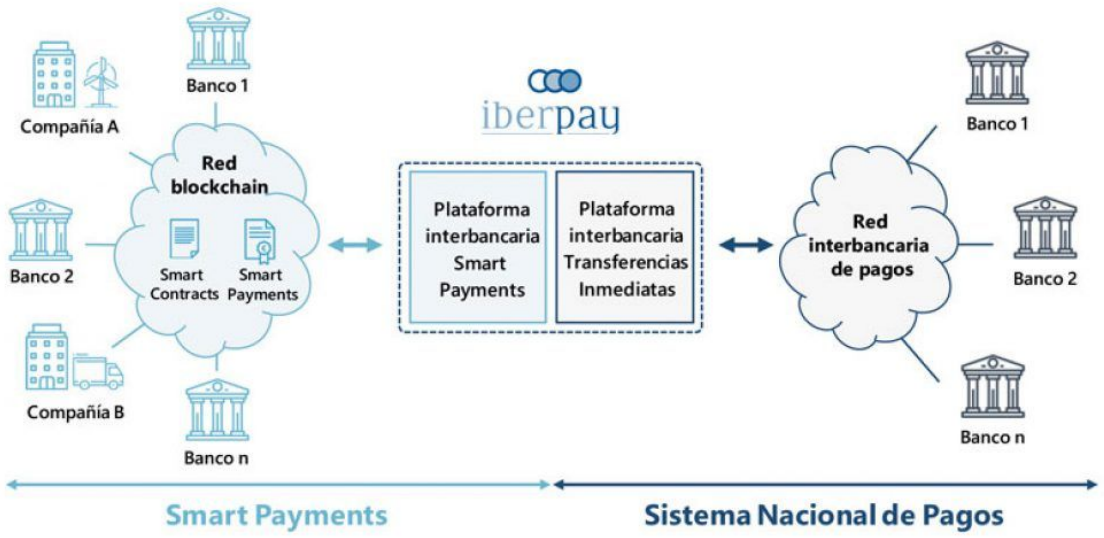

Figura 2.14: Prueba de concepto liderada por Iberpay

La prueba de concepto del sistema comenzó el pasado mes de octubre, con el respaldo técnico de Grant Thornton. La prueba incluye un piloto basado en un caso de negocio real desplegado en la red blockchain. La red ha sido implantada con seis nodos distribuidos que son gestionados por cada participante. Estaba previsto que la prueba debería de haber finalizado en el mes de abril del año 2020, pero debido a la situación derivada por la pandemia de la COVID-19, los resultados no han trascendido.

2.4.4 Stablecoins

Las *stablecoins* [40] son criptomonedas que han surgido con el objetivo de reducir la fuerte volatilidad que tienen monedas como el *bitcoin* o el *ether*. Las criptomonedas tradicionales siguen envueltas en una gran volatilidad, por lo que, tanto su uso como su inversión siguen estando relegados a un ámbito de gran riesgo.

Las *stablecoins* son tokens que están respaldados por distintos tipos de activos. En función del respaldo podemos clasificarlas en colateralizadas o no colateralizadas.

Stablecoins colateralizadas

Se trata de criptomonedas respaldadas por distintos tipos de activos.

En primer lugar, pueden estar avaladas por monedas fiat. Destacan Tether y TrueCoin, ambas respaldadas por el dólar estadounidense y respaldadas por empresas que ejercen de entidad central. No obstante, este sistema tiene sus contradicciones con el concepto fundamental de una criptomoneda. Pues las criptomonedas están orientadas a no depender de un ente centralizado como lo hacen las monedas fiduciarias.

Las *stablecoins* también pueden ser respaldadas por otras criptomonedas. Un ejemplo es DAI, una criptomoneda que se apoya en Ethereum y utiliza el valor del Ether para mantener su criptomoneda. En este caso los usuarios no compran directamente la criptomoneda DAI, sino que la "generan" a cambio de 'ethers' que intercambian en la plataforma a modo de depósito.

En último lugar, encontramos las criptomonedas respaldadas con bienes materiales como pueden ser el oro o inmuebles. Es el caso de G-Coin, una plataforma de tokens que equivalen a un gramo de oro físico cada uno. La compañía asegura que el oro está almacenado de forma segura y que emplean blockchain para garantizar que el material procede de zonas libres de conflicto [41].

Stablecoins no colateralizadas

Por otro lado, existe el grupo de *stablecoins* no respaldadas, que no están asociadas a ningún valor externo sino que únicamente utilizan algoritmos para evitar las fluctuaciones de su precio. Un ejemplo de este grupo es USDX en el que el sistema opera de manera descentralizada gracias a *smart contracts* que regulan su funcionamiento.

2.4.5 CBDC

Las monedas digitales emitidas por bancos centrales o Central Bank Digital Currency (CBDC), son un nuevo tipo de moneda legal, que ampliará el acceso digital del público a las cuentas de un banco central, que hoy en día solo está limitado a los bancos comerciales. Las CBDC combinan la naturaleza digital de los depósitos bancarios con las ventajas de las transacciones tradicionales [42]. El interés de los bancos por proyectos como las CBDC es cuidar el buen funcionamiento del sector bancario a través del cual se genera la intermediación para transformar depósitos en fuente estable de recursos para la inversión de la economía moderna.

Actualmente el banco central de China se encuentra desarrollando su propia moneda digital respaldada por el gobierno, el yuan digital. Esta moneda estará respaldada al 100% por las reservas que las instituciones comerciales paguen a la institución [43].

3. Herramientas y tecnologías utilizadas

3.1 Spring

Spring Boot [44] es una solución para el framework Spring de Java que sigue el principio *convención sobre configuración* y reduce la complejidad para desarrollar nuevos proyectos con el framework Spring. Desarrollado por la empresa Pivotal Software fue publicado en el año 2012.

Spring Boot ofrece la estructura básica del proyecto y las bibliotecas de terceros relevantes para la aplicación, lo que facilita el camino para comenzar a desarrollar de forma rápida simplificando la creación de aplicaciones con Spring. A día de hoy, la mayoría de aplicaciones basadas en este framework son desarrolladas con Spring Boot.

Las principales características de Spring Boot son:

- Incorporación directa de aplicaciones de servidores web o contenedores como son Apache Tomcat o Jetty, sin incorporar archivos WAR (Web Application Archive).
- Se simplifica la configuración de Maven con la incorporación de archivos POM (Project Object Models).
- Configuración automática de bibliotecas de Spring y de terceros siempre que sea posible.
- Sin ningún requisito de configuración XML, intentando evitarla por completo.

3.1.1 Fichero pom.xml

El archivo pom.xml contiene toda la información del proyecto y los detalles de configuración utilizados por Maven para construir la aplicación [45].

Las etiquetas *parent* contienen los elementos imprescindibles del proyecto, que son tres: *groupId*, *artifactId* y *version*.

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
</parent>
```

Figura 3.1: Etiqueta parent

Las dependencias del proyecto han sido añadidas utilizando los starters de Maven.

```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>org.web3j</groupId>
    <artifactId>quorum</artifactId>
    <version>4.5.16</version>
</dependency>

<dependency>
    <groupId>org.web3j</groupId>
    <artifactId>web3j-spring-boot-starter</artifactId>
    <version>1.6.0</version>
</dependency>
```

Figura 3.2: Dependencias de Spring Boot.

3.1.2 Clase principal

Se trata de la clase anotada con la etiqueta @SpringBootApplication donde se define el método *main()* que inicializará la aplicación. La anotación @SpringBootAplicattion realiza la combinación de tres anotaciones utilizadas en Spring tradicional: @Configuration, @EnableAutoConfiguration y @ ComponentScan. Estas anotaciones marcan la clase principal como fuente de definiciones de *beans* para el contexto de la aplicación, indica a Spring que debe configurarse en función de las dependencias añadidas e informa del paquete que debe escanear para registrar los componentes de la aplicación respectivamente.

```
@SpringBootApplication
public class BlockchainWalletApplication {

    public static void main(String[] args) {
        SpringApplication.run(BlockchainWalletApplication.class, args);
    }

    @Autowired
    MainAccount mainAccount;

    @Bean
    Quorum quorum() {
        String nodeEndpoint = mainAccount.getNodeEndpoint();
        Web3jService web3jService;

        web3jService = new HttpService(nodeEndpoint);

        return Quorum.build(web3jService);
    }
}
```

Figura 3.3: Clase BlockchainWalletApplication.java

3.1.3 Fichero application.yml

Se trata de un fichero que es usado por Spring Boot y sus dependencias como fichero de configuración, ya que sus propiedades son accesibles desde todo el código. Pueden utilizarse dos extensiones: .properties y .yml. La única diferencia entre estas es el formato en el que se declaran las propiedades.

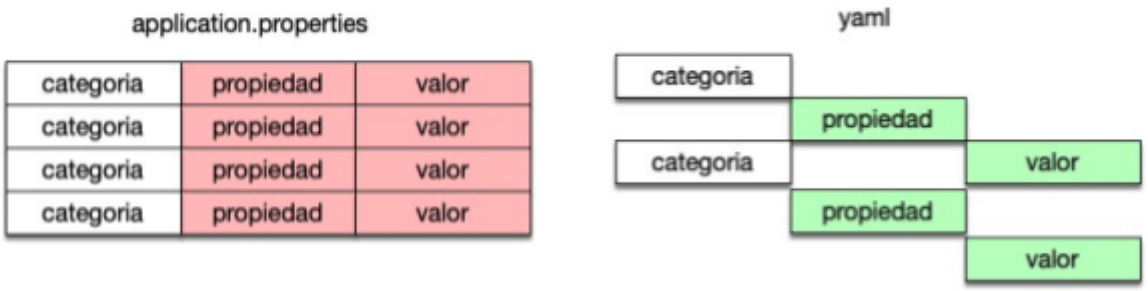

Figura 3.4: Diferencia entre fichero .properties y .yml

En este proyecto se ha utilizado el .yml porque se ha considerado que aporta una mejor lectura del mismo. En él se han definido el puerto en el que debe de correr la aplicación y los parámetros necesarios para realizar la conexión con la red blockchain. Estos parámetros son la ruta del nodo al que nos conectamos, la dirección del wallet, la ruta donde se encuentran las claves del wallet, la contraseña del wallet y la dirección donde se ha desplegado el Smart Contract

```yaml
server:
  port: 8080

mainaccount:
  nodeEndpoint: http://localhost:22000
  accountAddress: "0x868a54f8d625a3c86f84cba09c77736389078272"
  walletPath: "/Users/davidlopezcano/network/2-nodes/qdata/dd1/keystore/"
  walletName: "UTC--2020-07-08T21-34-57.225512000Z--868a54f8d625a3c86f84cba09c77736389078272"
  passWallet: ""
  smartContractAddress: "0x95797d8637D4B70BC3794483A4df0b919688F217"

loggingaccount:
  nodeEndpoint: "http://localhost:22000"
  accountAddress: "0x77d7d0856159c9190fff11d3c7e4d72387fa7f7b"
  walletPath: "/Users/davidlopezcano/network/2-nodes/qdata/dd1/keystore/"
  walletName: "UTC--2020-07-30T21-18-55.886000000Z--77d7d0856159c9190fff11d3c7e4d72387fa7f7b.json"
  passWallet: ""
```

Figura 3.5: Fichero properties.yml de la aplicación

Como se ha indicado con anterioridad, las propiedades definidas pueden recuperarse desde cualquier parte del código mediante anotaciones.

- @Value: para leer propiedades de forma individual y específica, normalmente para ficheros de configuración pequeños y sencillos.

- @ConfigurationProperties: mapea las propiedades del objeto Java marcado con esta anotación.

En este caso se ha utilizado la segunda opción para mapear las propiedades a un objeto que contiene todos los datos del nodo.

```
@Configuration
@ConfigurationProperties(prefix="mainaccount")
public class MainAccount {
```

Figura 3.6: Utilizando la anotación @ConfigurationProperties

3.1.3 API Rest

En este proyecto se ha creado una clase controlador llamada MainController.java. Spring Boot permite el desarrollo de microservicios mediante las clases anotadas con la etiqueta @RestController. Esta notación introducida en la versión 4 de Spring simplifica la creación APIs Rest. La notación @RequestMapping se está utilizando a nivel de clase y mapeará todas las peticiones que se realicen con la base */wallet*. La anotación @Api añade información en Swagger.

```
@Api("Controlador principal de la aplicación")
@RestController
@RequestMapping("/wallet")
public class MainController {
```

Figura 3.7: Controlador principal de la aplicación

Las principales características de una API Rest:

- Protocolo cliente servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, por lo que ni el cliente ni el servidor necesitarán recordar ningún estado previo para llevarla a cabo.

- Las operaciones HTTP son cuatro: POST (crear), GET (leer), PUT (editar) y DELETE (eliminar).

- Sistema de capas: existe una jerarquía entre los componentes y cada una de las capas lleva a cabo una funcionalidad dentro del sistema REST.

- Los objetos REST siempre se manipulan a partir de una URI. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

Si nos centramos en un método del controlador, podemos observar características similares a las vistas anteriormente. La anotación @ApiOperation aporta una información más precisa a Swagger sobre el método en cuestión. Y @RequestMapping, en este caso, recibe dos atributos, *value* indica la ruta que debe ser llamada para que el método sea ejecutado, se concatena con el @RequestMapping a nivel de clase quedando de la siguiente manera */wallet/recharge.* Con el atributo *method* indicaremos la operación HTTP con la que se debe de llamar al recurso.

```
@ApiOperation(value = "2-Recargar un wallet", notes = "Se simula la recarga de un wallet, tokenizando dinero fiat")
@RequestMapping(value = "/recharge", method = RequestMethod.POST)
public ResponseEntity<TransactionReceipt> recharge(HttpServletRequest request,
        @RequestBody TransactionModelNew transactionModel) throws Exception {

    LOGGER.info("Recargando wallet");
    LOGGER.info("Cuenta origen: " + mainAccount.getAccountAddress());
    LOGGER.info("Cuenta destino: " + logginAccount.getAccountAddress());
    LOGGER.info("Cantidad: " + transactionModel.getValue());

    try {

        TransactionReceipt result = smartContractService.recharge(transactionModel.getValue());
        return new ResponseEntity<TransactionReceipt>(result, HttpStatus.OK);

    } catch (Exception e) {

        throw e;

    }

}
```

Figura 3.8: Método del controlador principal

3.2 Swagger

Swagger es un framework utilizado para documentar APIs REST que hace referencia a un conjunto de reglas, especificaciones y herramientas. Swagger permite describir la estructura de las APIs para que las máquinas puedan leerlas. Para esto la API debe generar un fichero YAML o JSON que contenga una descripción detallada de la misma. Este archivo es esencialmente una lista de recursos de la API.

Swagger UI es una de las herramientas más atractivas de la plataforma. Para que la documentación sea útil, es necesario que esté todo perfectamente organizado para que se pueda navegar con facilidad por su estructura y se tenga un acceso intuitivo a los métodos de la API expuestos en él.

Swagger UI utiliza el documento JSON o YAML generado por la API y lo hace interactivo. Crea una plataforma que ordena cada uno de nuestros métodos (get, put, post, delete) y categoriza nuestras operaciones. Cada uno de los métodos es expansible, y en ellos podemos encontrar un listado completo de los parámetros con sus respectivos ejemplos e incluso ejecutar la llamada al *endpoint*.

Figura 3.9: Endpoints del controlador de la aplicación en Swagger UI.

3.3 Web3J

Web3J [46] es una librería válida para trabajar con Smart Contracts e integrarse con nodos de una red Ethereum. Se trata de una librería altamente modular y segura, válida para los lenguajes de programación Java y Android.

Web3J será capaz de encapsular desde la aplicación Java toda la interacción con la red blockchain sin necesidad de implementar toda la funcionalidad de una aplicación web en el lenguaje Solidity.

Si queremos trabajar con una red Quorum será necesario instalar una extensión de Web3J llamada Web3J-Quorum [47]. Esta extensión añade soporte para transacciones privadas de Quorum así como los medios necesarios para la correcta gestión de las demás funcionalidades propias de la red.

Para añadir esta librería a nuestro proyecto Spring Boot únicamente será necesario añadir dos librerías, la librería Web3J original y la extensión para Quorum.

```
<dependency>
    <groupId>org.web3j</groupId>
    <artifactId>web3j-spring-boot-starter</artifactId>
    <version>1.6.0</version>
</dependency>

<dependency>
    <groupId>org.web3j</groupId>
    <artifactId>quorum</artifactId>
    <version>4.5.16</version>
</dependency>
```

Figura 3.10. Dependencias de Web3J en Spring Boot

La interacción entre Java y Quorum se realiza a través del protocolo JSON-RPC, utilizando la conexión RPC para establecer la comunicación.

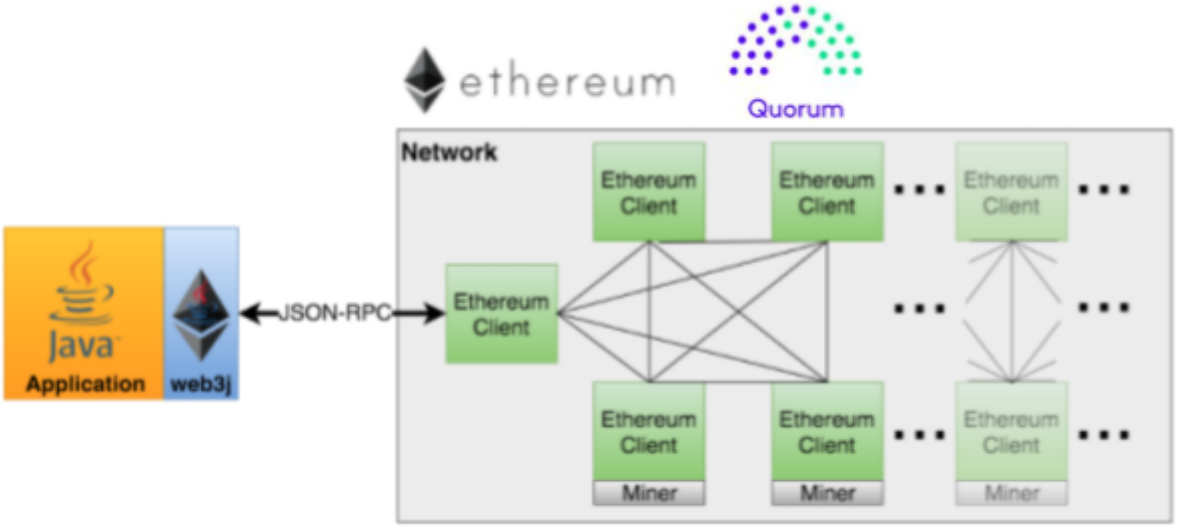

Figura 3.11: Arquitectura de Web3J

Web3J es capaz de generar automáticamente wrappers de contratos inteligentes de Java. Un wrapper es una clase Java generada a partir de los ficheros .bin y .abi de un contrato inteligente, y es utilizada para interactuar con el Smart Contract desplegado en la blockchain. Tras instalar Web3J en nuestro ordenador y añadir el .bin a las variables de entorno es posible ejecutar el comando y generar la clase wrapper.

```
davidlopezcano@MacBook-Pro-de-David 2-nodes % sudo web3j solidity generate --binFile=erc20.bin --abiFile=erc20.abi
 -o /Users/davidlopezcano/Desktop -p com.wallet.blockchainWallet.smartcontract
```

Figura 3.12: Comando para generar wrapper

Es necesario indicar al comando donde se encuentran los ficheros .bin y .abi, así como el directorio donde será generado y el paquete Java donde será añadido. La clase generada es Erc20.java.

```
/**
 * <p>Auto generated code.
 * <p><strong>Do not modify!</strong>
 * <p>Please use the <a href="https://docs.web3j.io/command_line.html">web3j command line tools</a>,
 * or the org.web3j.codegen.SolidityFunctionWrapperGenerator in the
 * <a href="https://github.com/web3j/web3j/tree/master/codegen">codegen module</a> to update.
 *
 * <p>Generated with web3j version 4.5.16.
 */
@SuppressWarnings("rawtypes")
public class Erc20 extends Contract {
    public static final String BINARY = "{\n"
            + "\t\"linkReferences\": {},\n"
            + "\t\"object\": \"608060405234801561001057600080fd5b50604051610e30380380610e3083398101806040528101908080519060200190929190805182019291906020018051906020019092919080
            + "\t\"opcodes\": \"PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x40 MLOAD PUSH2 0xE30 CODESIZE SUB
            + "\t\"sourceMap\": \"2357:2716:0:-;;;3068:638;8:9:-1;5:2;;;30:1;27;20:12;5:2;3068:638:0;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;3246:14;3223:8;:20;32
            + "}";
```

Figura 3.13: Clase autogenerada por Web3J

3.4 Metamask

Metamask [48] es una extensión de Google Chrome que funciona como puente entre las aplicaciones descentralizadas y el navegador de una forma segura y posibilita usar varias cuentas. Metamask nos permite ejecutar transacciones, administrar tokens y monederos a través de una interfaz fácil que agrupa todos los servicios de Ethereum y Quorum. Esto es posible gracias a la integración de la API Web3 en el código JavaScript de cada sitio web.

Metamask, además proporciona una capa extra de seguridad puesto que posee su propia llave privada, de manera que no maneja las claves de las cuentas y maneja los tokens propios, de esa manera las transacciones nunca tocan la configuración del servidor, reduciendo riesgos.

3.5 Cakeshop

Cakeshop [49] es un conjunto de herramientas y APIs para el desarrollo de entornos Ethereum. Se despliega como una aplicación Java e incluye los paquetes de Geth, Quorum, Tessera y Constellation, y un compilador de Solidity para los contratos inteligentes.

Cakeshop permite al usuario:

- Gestión de los nodos.
- Explorar la blockchain: ver transacciones, bloques y contratos, así como histórico de un contrato en un momento determinado.
- Consola de administración: iniciar y detener nodos, crear cluster y ver el estado general de la red.
- Creación de contratos: permite el desarrollo y compilación de Smart Contracts en Solidity.

4. Proceso de ingeniería del software

4.1 Análisis

4.1.1 Requisitos del sistema

Definimos requisito como: una condición o capacidad exigida por el usuario para solucionar un problema o alcanzar un objetivo. Según la clasificación realizada por H. Pohl [50] podemos clasificar los requisitos en requisitos funcionales y de calidad.

Requisitos funcionales

En ellos se especifica la funcionalidad que el sistema debe proporcionar a los usuarios. Los requisitos funcionales de nuestra aplicación son:

1. **Añadir un nuevo wallet:** un usuario debe poder crear nuevos wallet a un usuario.

2. **Recargar un wallet:** un usuario propietario del wallet debe poder recargar su wallet intercambiando euros por tokens del sistema.

3. **Consultar el saldo:** un usuario debe poder consultar los tokens de su wallet.

4. **Enviar tokens:** un usuario podrá enviar tokens de su wallet a otro wallet del sistema.

5. **Consultar las transacciones:** un usuario debe poder consultar las transacciones que ha enviado y recibido en su wallet.

Requisitos de calidad

Se trata de requisitos no funcionales relacionados con la calidad del sistema en desarrollo.

1. **Disponibilidad:** el sistema deberá de tener un alto porcentaje de tiempo durante el cual estará plenamente operativo y disponible para usarlo.
2. **Confiabilidad:** La probabilidad de que el sistema funcione sin fallos debe ser alta.
3. **Tiempo de escritura:** el sistema debe escribir en una red blockchain, por lo que el tiempo de escritura y validación en la red debe ser muy rápido.

4.1.2 Perfiles de usuario

Para comenzar esta fase de análisis será necesario definir los distintos tipos de usuarios que van a interactuar con el sistema. Se puede definir tipos de usuarios perfectamente diferenciados, un usuario administrador y un usuario cliente. Se tratan de dos usuarios, que no requieren de conocimientos previos o expertos en tecnologías blockchain, puesto que únicamente van a interactuar con una API común muy similar a cualquiera de las que se podrían utilizar con otras tecnologías.

La única distinción entre ambos usuarios es a priori la distinción de roles y la funcionalidad que pueden hacer cada uno de ellos. Mientras que todos los usuarios son capaces de consultar su saldo y realizar y consultar transacciones , es el administrador únicamente el que podrá crear nuevos wallets.

El usuario administrador será también, el que despliegue el contrato inteligente del token ERC20 en la red blockchain. Al desplegar el contrato todos los tokens creados serán almacenados en su wallet, por lo que al realizar una recarga de wallet, se realizará una transacción desde la cuenta del administrador a la cuenta del cliente que desee recargar su wallet.

Al tratarse este proyecto de una prueba de concepto, en un entorno local con una blockchain privada se unificarán los dos usuarios y se interactúa con el sistema como si de un único usuario se tratara.

4.1.3 Elección de tecnología blockchain

Tras realizar un minucioso estudio de las tecnologías blockchain que permiten la creación de redes privadas se ha optado por desarrollar este proyecto con Quorum.

El principal motivo por el que se ha elegido la red Quorum es por su excelente rendimiento. Quorum introduce dos algoritmos de consenso que son ideales para redes privadas, ya que son capaces de confirmar un bloque cada 50 ms y esto hace de Quorum una plataforma ideal para entornos empresariales donde la velocidad de ejecución de una transacción es muy importante.

También como ya se explicó en el apartado 3.2 Quorum se construye sobre Ethereum, por lo tanto nos hemos podido aprovechar de la robustez de una plataforma con una gran comunidad muy estable, así como, de todas las herramientas de terceros que nos permiten interactuar con la red.

Quorum además permite la realización de transacciones, que solo son visibles para las partes implicadas en la transacción. Si bien, en este trabajo todas las transacciones son públicas y se realizan entre nodos Quorum, esta funcionalidad puede ser muy útil en trabajos futuros relacionados.

Por último, estamos ante una plataforma de código abierto que hace que los desarrolladores sean libres de experimentar y mantener la integridad de la red mientras que se va renovando y actualizando.

4.2 Diseño

4.2.1 Diagrama de clases

El diagrama de clases muestra la estructura global del sistema, ya que se pueden observar todas las clases que lo implementan junto con sus métodos y atributos, y la relación existente entre ellas. Por lo tanto, este diagrama ofrecerá una visión más detallada de la API REST que contiene toda la lógica de negocio de la aplicación.

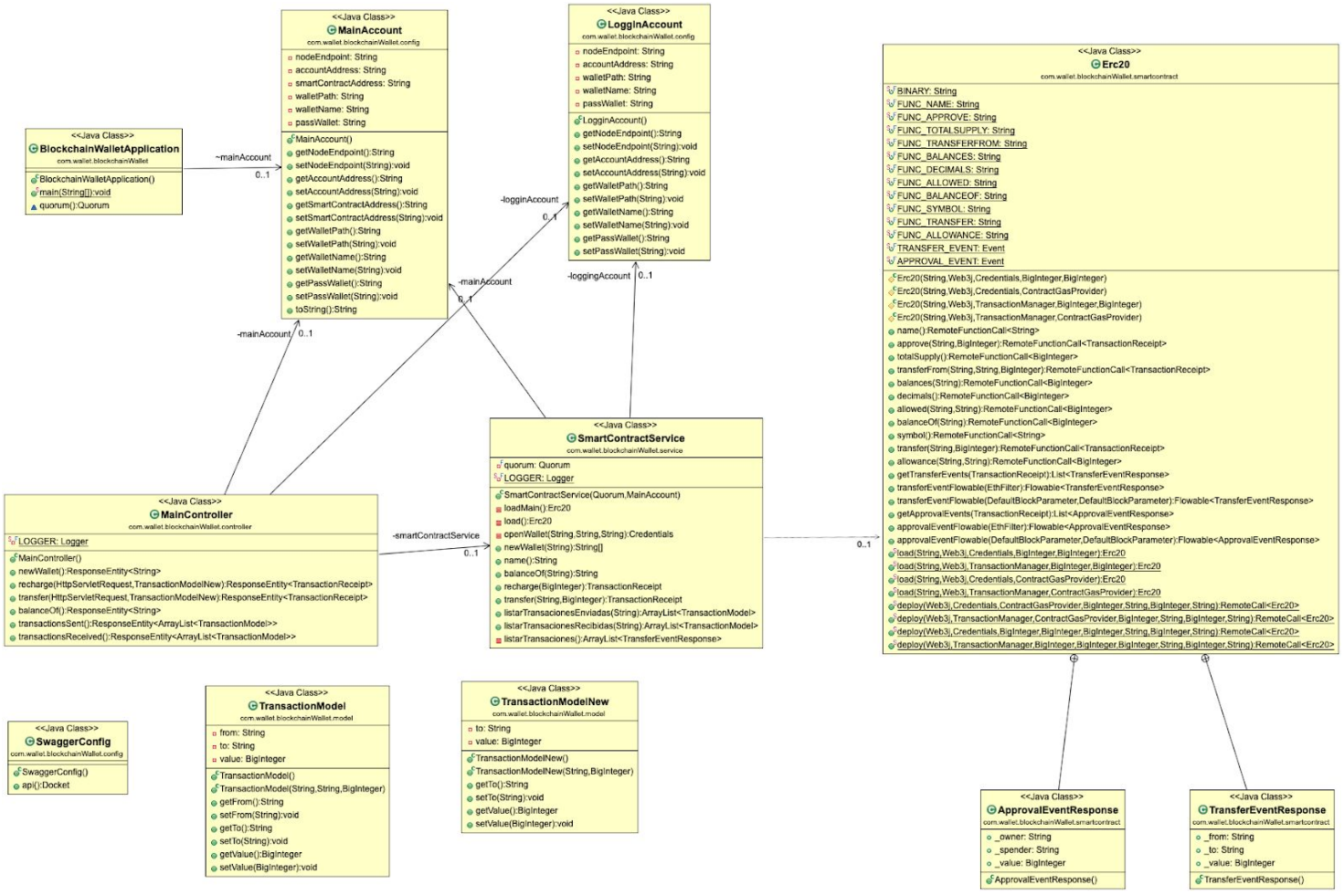

Figura 4.1: Diagrama de clases

Las clases están organizadas siguiendo el modelo vista controlador, entra las que destacan.

- MainController.java: se trata del controlador principal de la aplicación actuando como capa de presentación de la API REST.

- SmartContractService.java: clase encarga de gestionar la capa de negocio.

- Erc20.java: esta clase ha sido generada por Web3j, y permite ejecutar las funciones publicas del Smart Contract, que se encuentra desplegado en la red blockchain. Actúa como fuente de datos de la API REST dado que esta aplicación carece de base de datos, siendo la red blockchain la única estructura encargada de la persistencia de los datos.

- BlockchainWalletApplication.java: clase principal de la aplicación

- MainAccount.java: contiene los datos del wallet principal de la blockchain. Se trata de la cuenta que ha desplegado el Smart Contract en la red y por lo tanto, almacenará todos los tokens iniciales del sistema. Esta cuenta incluye la dirección del Smart Contract del estándar Erc20, piedra angular de nuestro sistema. Los parámetros se encuentran inicialmente definidos en el fichero application.yml.

- LogginAccount.java: contiene los datos del wallet que realizará las operaciones en la API REST. Se simula que un usuario se ha logueado en el sistema y se le permite operar en él. Los parámetros se encuentran inicialmente definidos en el fichero application.yml.

- SwaggerConfig.java: esta clase almacena los parámetros de personalización de Swagger y permite que se genere el fichero JSON que contiene toda la información de la API.

4.2.2 Diagrama de secuencia

El diagrama de secuencia UML representa los eventos en orden cronológico, razón por la que a veces se le llama diagrama de eventos o escenario de eventos. El diagrama de secuencia describe básicamente cómo los objetos intercambian mensajes en un orden determinado. Los objetos son los bloques de construcción básicos de los diagramas UML y representan ciertas características de un elemento del sistema, que varían dependiendo del diagrama. En las interacciones, los objetos son las conocidas como líneas de vida. Los mensajes intercambiados entre los objetos son representados por flechas horizontales que unen la línea de vida del objeto emisor con la línea de vida del objeto destinatario. Existen tres tipos de mensajes:

Figura 4.2: Tipos de mensajes en un diagrama de secuencia UML.

A continuación, se muestran los principales diagramas de secuencia que representan los casos de uso de nuestro proyecto y que ayudarán a comprender la interacción de los diferentes componentes del diagrama de clases.

Crear un wallet

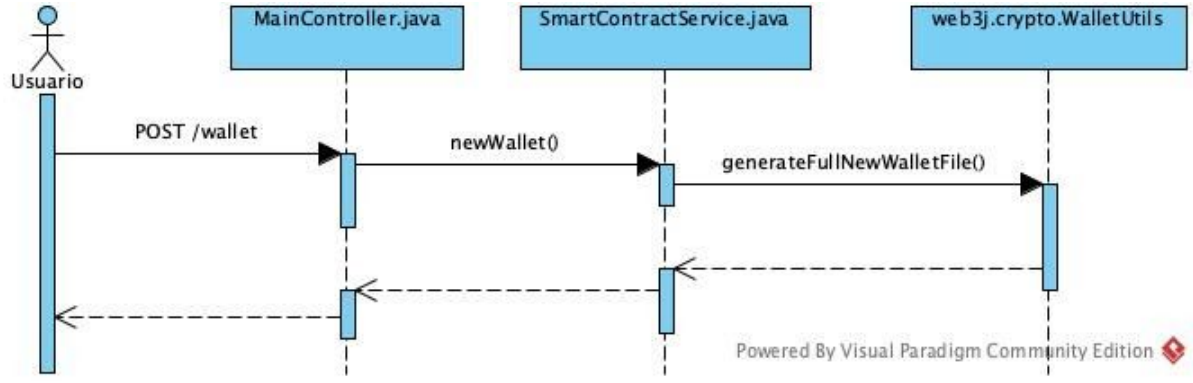

Figura 4.3: Diagrama de secuencia. Creación de Wallet.

Recargar un wallet

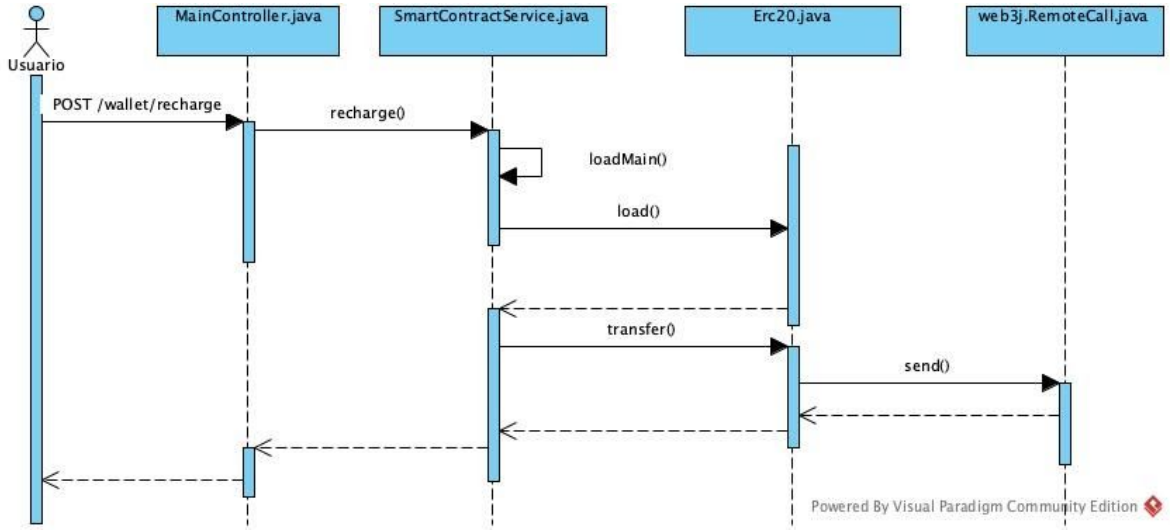

Figura 4.4. Diagrama de secuencia. Recargar un wallet

Realizar una transacción

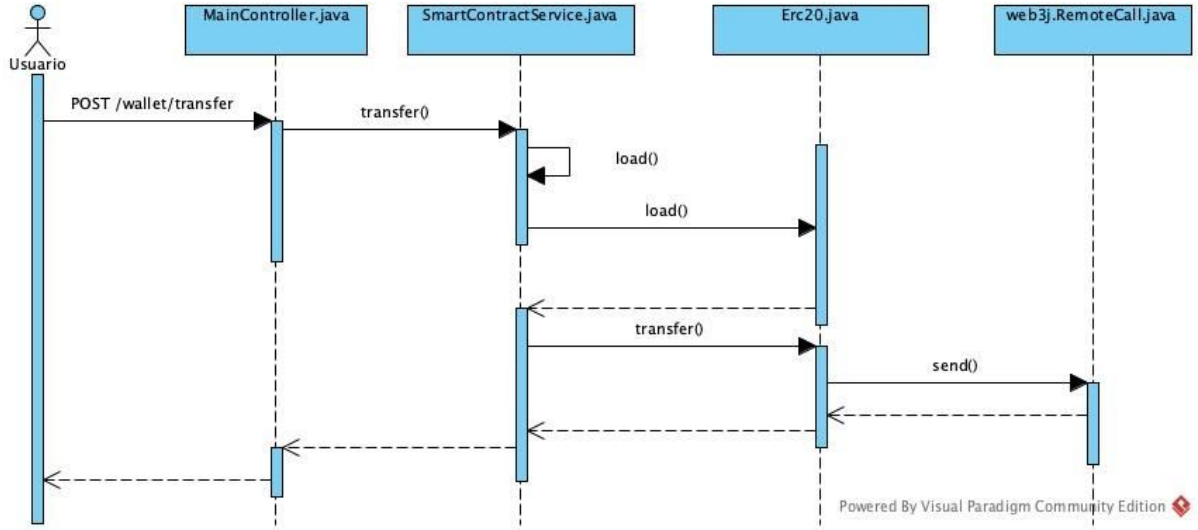

Figura 4.5. Diagrama de secuencia. Realizar una transacción.

Consultar el saldo de un wallet

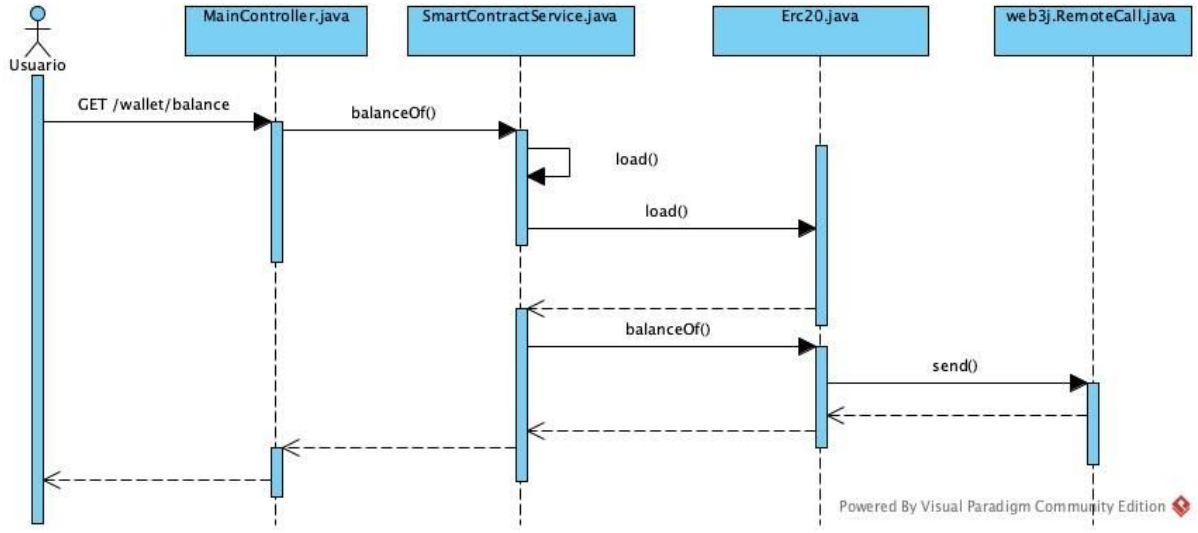

Figura 4.6: Diagrama de secuencia. Consultar saldo de wallet

Consultar las transacciones realizadas por un wallet

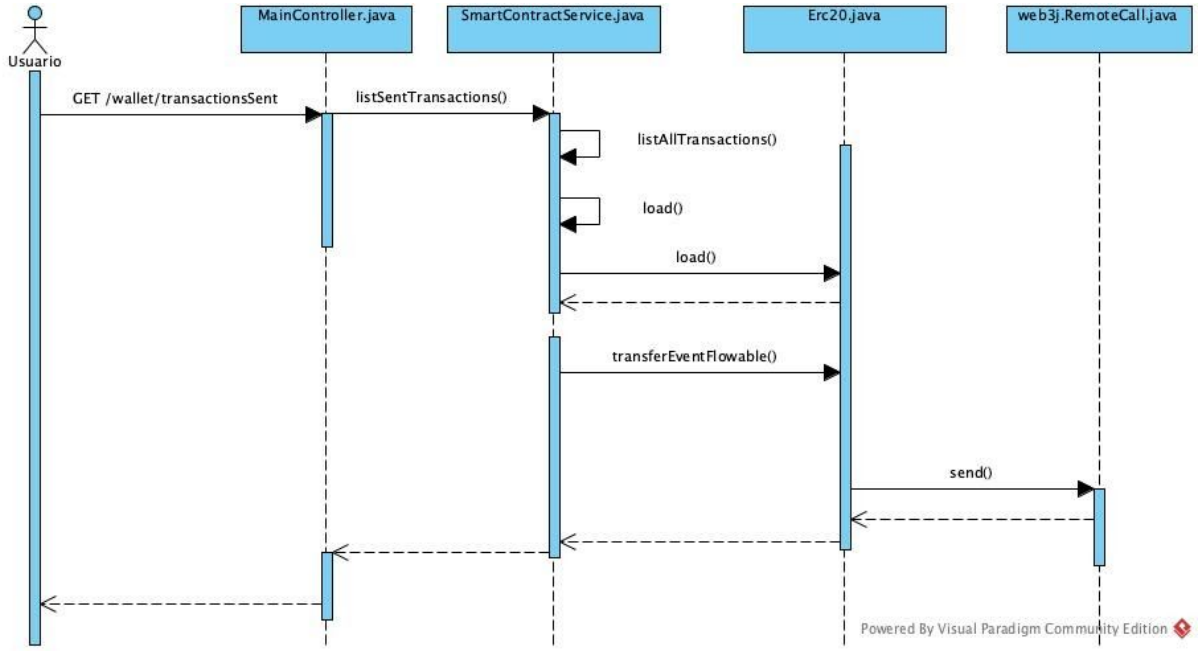

Figura 4.7: Diagrama de secuencia. Consultar las transacciones realizadas.

Consultar las transacciones recibidas en un wallet

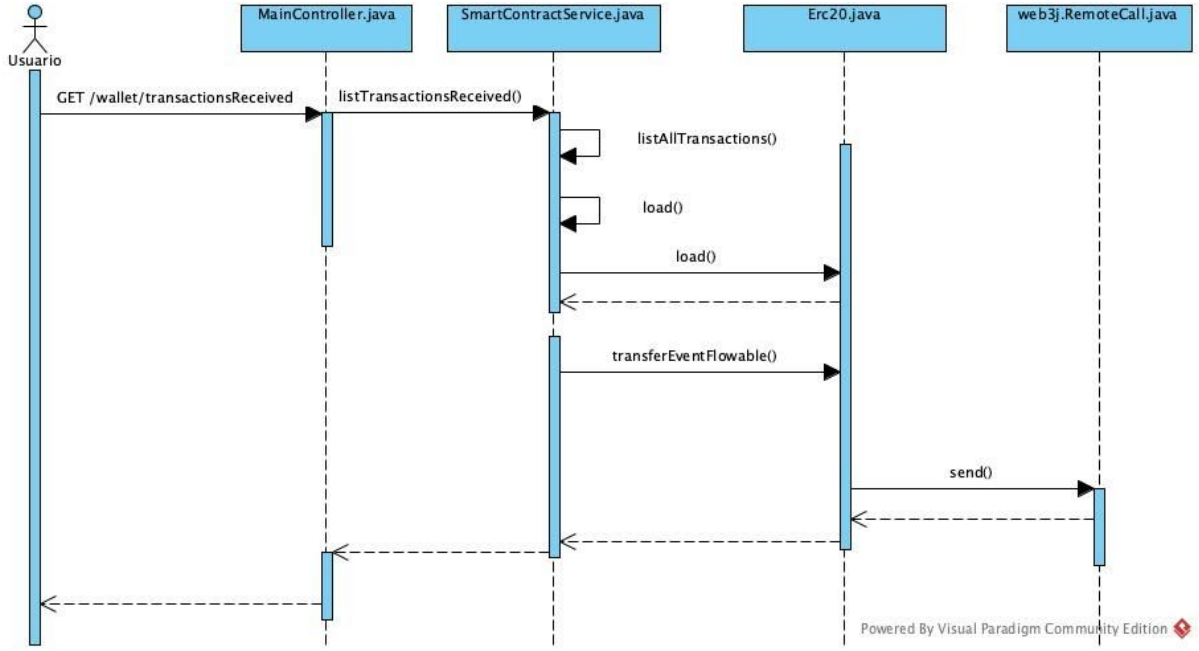

Figura 4.8: Diagrama de secuencia. Consultar las transacciones recibidas.

4.3 Implementación

En este apartado mostraremos una visión general sobre la arquitectura de la aplicación, en las que se exponen las partes que las componen y las relaciones que hay entre ellas.

Posteriormente se analizarán algunos detalles importantes sobre la implementación de nuestra aplicación, para comprender su funcionamiento y la interacción de los elementos constituyentes.

4.3.1 Arquitectura del sistema

En este apartado vamos a tratar la arquitectura de nuestra aplicación, esta incluye dos arquitecturas diferentes.

En primer lugar nos encontramos con una arquitectura cliente servidor. La comunicación entre ambos se llevará a cabo mediante una API REST. El protocolo REST separa totalmente la interfaz de usuario y del servidor, esto mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los componentes de la aplicación puedan evolucionar de forma diferente. Al tratarse de una prueba de concepto el cliente utilizado ha sido el framework Swagger, pero como hemos comentado con anterioridad este tipo de arquitectura permite en un futuro conectar un cliente personalizado y más intuitivo.

Por otro lado, la red blockchain Quorum sigue una arquitectura P2P. Al tratarse de una prueba de concepto los tres nodos de la red blockchain están desplegados en el mismo servidor local. El Smart Contract desarrollado en lenguaje Solidity, ha sido desplegado mediante el IDE Remix en la plataforma Quorum.

La comunicación entre el servidor y la red Quorum se lleva a cabo mediante la librería Web3J. A continuación, se muestra el diagrama de arquitectura con todas las tecnologías que han intervenido en este proyecto.

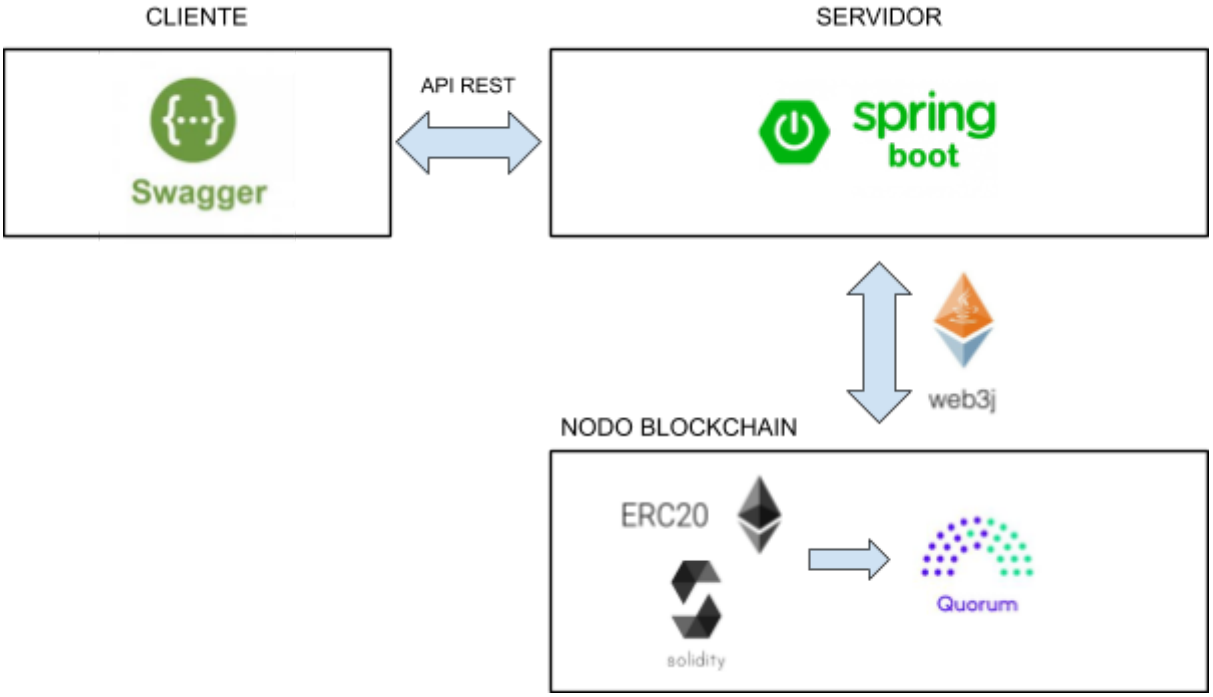

Figura 4.9: Diagrama de arquitectura

4.3.2 IDEs de desarrollo

Para la realización de este trabajo se han utilizado dos herramientas de desarrollo: Eclipse para el desarrollo de la API Rest en Java y Remix para la implementación de los Smart Contracts en Solidity.

Eclipse es un IDE genérico popularmente conocido que proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones. Se ha necesitado instalar el plugin de Spring Boot para realizar este proyecto.

Remix, anteriormente conocido como Browser Solidity, permite escribir, depurar y desplegar contratos inteligentes basados en Solidity. Se trata de una herramienta online accesible desde la web remix.ethereum.org. Además es posible tener acceso al estado y las propiedades de los smart contracts que ya hemos creado previamente, por lo que reduce el riesgo de cometer errores de codificación y aplicar mejoras previo análisis del código.

Solidity es el lenguaje de programación utilizado por Ethereum y Quorum para el desarrollo de sus Smart Contracts. Se trata de un lenguaje de alto nivel con una sintaxis similar a la de JavaScript, preparado para ejecutarse principalmente en la máquina virtual de Ethereum (EVM). Es necesario compilar el código Solidity para convertir el programa a *bytecode* de EVM.

4.3.2 Detalles sobre la implementación

Estructura del proyecto

La estructura de paquetes de nuestro proyecto sigue el esquema tradicional de los proyectos de Spring. Dentro del directorio Java encontramos los siguientes paquetes:

- blockchainWallet: contiene la clase principal de la aplicación.
- blockchainWallet.config: este paquete almacena la clase de configuración de Swagger y las clases que contienen los datos de las cuentas que interactúan con el sistema. Posteriormente se dará información detallada de estas clases
- blockchainWallet.controller: contiene el controlador principal de la aplicación.
- blockchain.Model: guarda dos modelos para mapear los datos necesarios de una transacción.
- blockchain.smartcontract: almacena las clase wrapper generada por Web3J, que realiza la comunicación con la red blokchain.
- blockchain.Service: contiene la clase que se comunica con el wrapper.

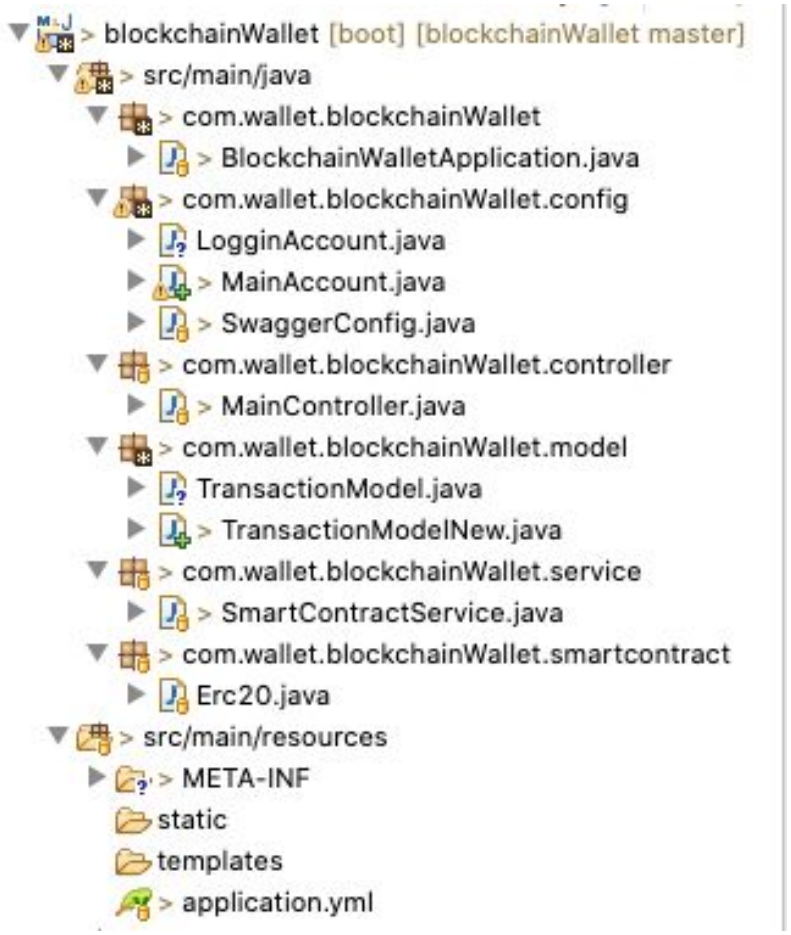

Figura 4.10: Estructura del proyecto

MainAccount y LoginAccount

Si realizáramos el despliegue de nuestra aplicación en un entorno de producción su funcionamiento y sobre todo la autenticación de un usuario funciona de manera distinta.

Al tratarse de una red blockchain privada un nuevo usuario debería de solicitar al administrador del sistema un wallet. Posteriormente recibiría los datos del nodo donde debe conectarse, la dirección de su wallet y su clave privada. Tras introducir estos datos en el sistema correspondiente sólo le quedaría definir una contraseña.

Puesto que en este proyecto se ha desarrollado una prueba de concepto el proceso de autenticación se ha simplificado ya que los datos que un usuario necesita para acceder al sistema se encuentran definidos en el fichero *application.yml*. Estos datos se mapean en la clase de configuración *LogginAccount.java* al arrancar la aplicación y sirven todas las peticiones de la API se realizan en nombre de este wallet puesto que es el que ha hecho *login* en el sistema. Por ejemplo, al consultar el saldo de un wallet en el endpoint correspondiente no será necesario indicar de qué cuenta se trata puesto que siempre se consultarán los datos para el wallet indicado en la *LogginAccount.java.*

```
loggingaccount:
  nodeEndpoint: "http://localhost:22000"
  accountAddress: "0xc8f1b33527d5fe2ab33fa1cba703c18ba5515147"
  walletPath: "/Users/davidlopezcano/network/3-nodes-raft-bash/qdata/dd1/keystore/"
  walletName: "UTC--2020-08-30T07-06-14.709760000Z--c8f1b33527d5fe2ab33fa1cba703c18ba5515147"
  passWallet: ""
```

Figura 4.11: Datos proporcionados para la logginAccount en application.yml

```
@Configuration
@ConfigurationProperties(prefix="loggingaccount")
public class LogginAccount {

    private String nodeEndpoint;

    private String accountAddress;

    private String walletPath;

    private String walletName;

    private String passWallet;
```

Figura 4.12: Clase LogginAccount.java

La clase *MainAccount* está formada por una estructura muy similar a *LogginAccount*. Esta clase almacenará los datos de la account que ha desplegado el Smart Contract Erc20 en la red blockchain. Esta cuenta almacena todos los tokens creados inicialmente, por lo que será una cuenta gestionada por la administración del sistema. Cuando se realice una petición para recargar un wallet, el emisor de la transacción será el wallet definido en la *MainAccount* y el receptor el wallet del *LogginAccount*. La *MainAccount* también almacenará la dirección del contrato inteligente desplegado en la red.

```yaml
mainaccount:
  nodeEndpoint: http://localhost:22000
  accountAddress: "0xc077bf171427abe5c3c62527cac4f8e29ba5823e"
  walletPath: "/Users/davidlopezcano/network/3-nodes-raft-bash/qdata/dd1/keystore/"
  walletName: "UTC--2020-08-29T16-31-58.798939000Z--c077bf171427abe5c3c62527cac4f8e29ba5823e"
  passWallet: ""
  smartContractAddress: "0x015917bBDAF81B97998A23D7a9FAb5033C35eF95"
```

Figura 4.13: Datos proporcionados para la mainAccount en application.yml

```java
@ConfigurationProperties(prefix="mainaccount")
public class MainAccount {

    private String nodeEndpoint;

    private String accountAddress;

    private String smartContractAddress;

    private String walletPath;

    private String walletName;
```

Figura 4.13: Clase MainAccount.java

Ejemplo de petición

Por último mostraremos el flujo que realiza una petición desde que se realiza la petición en la API hasta que se obtienen los datos en la blockchain, para ello nos centraremos en el caso de uso de realizar una transacción.

Cuando un usuario realiza una petición **POST** al endpoint /wallet/transfer Spring Boot intercepta la petición y realiza la llamada al método *transfer()* de la clase MainController. El JSON del cuerpo de la petición se mapea en un objeto de la clase *TransactionModelNew* que contiene los datos de la misma.

```
@ApiOperation(value = "3-Realizar transferencia", notes = "")
@RequestMapping(value = "/transfer", method = RequestMethod.POST)
public ResponseEntity<TransactionReceipt> transfer(HttpServletRequest request,
        @RequestBody TransactionModelNew transactionModel) throws Exception {

    LOGGER.info("Realizando nueva transferencia");

    LOGGER.info("Cuenta origen: " + logginAccount.getAccountAddress());
    LOGGER.info("Cuenta destino: " + transactionModel.getTo());
    LOGGER.info("Cantidad: " + transactionModel.getValue());

    try {

        TransactionReceipt result = smartContractService.transfer(transactionModel.getTo(),
                transactionModel.getValue());
        return new ResponseEntity<TransactionReceipt>(result, HttpStatus.CREATED);

    } catch (Exception e) {

        throw e;

    }

}
```

Figura 4.15: Función transfer en el controlador MainController

El controlador realiza una llamada a la función *transfer* de *SmartContractService.* Esta función en primer lugar obtiene una instancia de la clase *Erc20* relacionada con la dirección del wallet que va a enviar la transacción. Para ello hay que llamar a la función *load()* de esta clase que utilizará las credenciales del wallet de la clase *LogginAccount* para realizar la autenticación. También será necesario indicar el precio del gas, que en el caso de Quorum siempre es 0, y la cantidad máxima de gas a consumir en una llamada

```
private Erc20 load() throws Exception {
    LOGGER.info("load -- Cargando cuenta logeada");

    Credentials credentials = openWallet(loggingAccount.getWalletPath(),  loggingAccount.getWalletName(), loggingAccount.getPassWallet() );

    BigInteger gasPrice = BigInteger.valueOf(0);
    BigInteger gasLimit = BigInteger.valueOf(999999);
    StaticGasProvider gas = new StaticGasProvider(gasPrice, gasLimit);

    return Erc20.load(mainAccount.getSmartContractAddress(), quorum, credentials, gas);

}
```

Figura 4.16: Función load en la clase SmartContractService

Una vez obtenida la instancia de la clase wrapper con el método load, se realizará la llamada a la función transfer de la clase Erc20.

```
public TransactionReceipt transfer(String to, BigInteger value) throws Exception {
    LOGGER.info("transfer -- Realizando transferencia");

    Erc20 erc20 = load();

    return erc20.transfer(to, value).send();

}
```

Figura 4.17: Función transfer en la clase SmartContractService

Esta clase ha sido generada automaticamente por Web3J y es la que establece la comunicación con el Smart Contract desplegado en Quorum.

```
public RemoteFunctionCall<TransactionReceipt> transfer(String _to, BigInteger _value) {
    final Function function = new Function(
            FUNC_TRANSFER,
            Arrays.<Type>asList(new org.web3j.abi.datatypes.Address(160, _to),
            new org.web3j.abi.datatypes.generated.Uint256(_value)),
            Collections.<TypeReference<?>>emptyList());
    return executeRemoteCallTransaction(function);
}
```

Figura 4.18: Función transfer en la clase Erc20

Finalmente la función *transfer* definida en el Smart Contract es la que contiene la lógica de la transacción obteniendo el saldo de la cuenta que realiza la transacción, decrementando el saldo del emisor y aumentando la cuenta del receptor.

ERC20.sol

```
function transfer(address _to, uint256 _value) public returns (bool success) {
    require(balances[msg.sender] >= _value);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    emit Transfer(msg.sender, _to, _value);
    return true;
}
```

Figura 4.19: Función transfer en el Smart Contract

5. Conclusiones y trabajos futuros

5.1 Conclusiones

La finalidad de este proyecto era adentrarse en el conocimiento de la tecnología blockchain y conseguir una visión más específica de las posibilidades que aporta al momento actual en el que nos encontramos en general, y en particular a un ámbito financiero.

Con el desarrollo de este proyecto se han conseguido llevar a cabo los objetivos que se definieron en la propuesta de este TFM y que están documentados en el punto 1.2 de la presente memoria.

Se ha realizado una revisión bibliográfica sobre la tecnología blockchain, analizando sus antecedentes, su arquitectura y las distintas plataformas que permiten la implementación de esta tecnología en un ámbito empresarial, siendo Quorum la tecnología elegida para el desarrollo de este proyecto.

Además se ha analizado cómo se está aplicando esta tecnología actualmente en un ámbito financiero, prestando especial interés en los casos de uso relacionados con bancos que incorporan la blockchain a sus productos.

Para finalizar se ha desarrollado una prueba de concepto en la que nuestra API actúa de entidad *tokenizadora* de dinero fiat y lo transforma en tokens de nuestro sistema. Esta aplicación se ha desarrollado usando código auditado y estandarizado para los tokens ERC20, y para el contrato de la gestión de los tokens se ha intentado utilizar código simple sin redundancias ni bucles que consumen más recursos de la red.

Para mi ha supuesto un reto la realización de este trabajo debido a que inicialmente mi conocimiento sobre la tecnología era nulo y he podido observar que la curva de aprendizaje es bastante alta. El principal problema que he encontrado en estos meses ha sido asimilar correctamente todos los conceptos que rodean a la cadena de bloques y conseguir comprender como llegar a poner en práctica los conceptos teóricos aprendidos. Una vez que se llevó a cabo el proceso de análisis y diseño de la aplicación, el desarrollo del proyecto ha sido una labor relativamente sencilla debido a que Spring Boot es una tecnología que utilizo cada día en mi puesto de trabajo.

Para concluir podemos afirmar que la tecnología blockchain ha conseguido adaptarse a un ámbito empresarial. Si bien en sus orígenes proliferaron las redes blockchain públicas, que utilizaban algoritmos de consenso poco eficientes y con un tiempo de validación de transacción bastante alto, a día de hoy se ha solucionado problemas de escalabilidad, tiempos de respuesta y existen multitud de soluciones que aportan privacidad y eficiencia cumpliendo así las necesidades de una empresa.

5.2 Propuestas de mejora

Debido a que en este trabajo se ha desarrollado una prueba de concepto, se ha implementado la funcionalidad básica para poder comunicarse con la red blockchain.

Como hemos podido ver en la memoria, Quorum permite el envío de transacciones privadas entre pares. Crear una red blockchain que permita la ejecución de este tipo de transacciones abriría un abanico de casos de uso que permitirían conversaciones privadas en un entorno previamente permisionado.

Añadir una interfaz de usuario más intuitiva que Swagger facilitaría el uso de sistema a usuarios con conocimientos informáticos menores.

6. Bibliografía

[1] L. H. Encinas, J. D. Vico, and D. A. Guardeño, Blockchain. CSIC, 2019.

[2] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008.

[3] L. H. Encinas, J. D. Vico, and D. A. Guardeño, Blockchain. CSIC, 2019.

[4] "Blockchain Explained - Intro - Beginners Guide to Blockchain", BlockchainHub, 2020. [Online]. Available: https://blockchainhub.net/blockchain-intro/

[5] Steinmetz, R.; Wehrle, K (2005). 2. What Is This “Peer-to-Peer” About?. Springer Berlin Heidelberg. pp. 9-16.

[6] "Blockchain y la teoría de juegos, parte VI — Steemit", Steemit.com, 2020. [Online]. Available: https://steemit.com/cryptocurrency/@juanfb/blockchain-y-la-teoria-de-juegos-parte-vi

[7] 101blockchains.com, 2020. [Online]. Available: https://101blockchains.com/es/algoritmos-de-consenso-blockchain/

[8] A. Tar, "Proof-of-Work, Explained", Cointelegraph, 2020. [Online]. Available: https://cointelegraph.com/explained/proof-of-work-explained.

[9] U. Chohan, "Proof-of-Stake Algorithmic Methods: A Comparative Summary", SSRN Electronic Journal, 2018. Available: 10.2139/ssrn.3131897.

[10] P. Denning, "Fault Tolerant Operating Systems", ACM Computing Surveys, vol. 8, no. 4, pp. 359-389, 1976. Available: 10.1145/356678.356680.

[11] L. Lamport, "The Weak Byzantine Generals Problem", Journal of the ACM (JACM), vol. 30, no. 3, pp. 668-676, 1983. Available: 10.1145/2402.322398.

[12] M. Lucena López, Criptografía y seguridad en computadores, 5th ed. Jaén, 2019.

[13] Diffie, W. y M.E.Hellman. "New directions in cryptography", IEEE Transactions on Information Theory 22 (1976), pp. 644-654.

[14] Hankerson, Menezes, Vanstone: "Guide to Elliptic Curve Cryptography", Springer-Verlag, 2004.

[15] "¿Qué es un Árbol de Merkle y en qué influye dentro de la minería?", Ethereum, 2020. [Online]. Available: https://miethereum.com/mineria/#toggle-id-5.

[16] P. Orden and W. González, "Apropósito del valor de la Internet Del Valor. Aportes metodológicos y referencias empíricas para conceptualizar las tecnologías blockchain en la actualidad.", 2019.

[17] W. Mougayar and V. Buterin, The Business Blockchain: a primer on the Promise, Practice and Application. John Wiley & Sons, 2016.

[18] "Qué son los tokens y cómo se diferencian de las criptomonedas", CriptoNoticias - Bitcoin, blockchains y criptomonedas. [Online]. Available: https://www.criptonoticias.com/mercados/que-son-tokens-como-diferencian-criptomonedas/

[19]"¿Cuál es la diferencia entre criptomonedas, tokens, monedas virtuales y digitales?", Cripto247. [Online]. Available: https://www.cripto247.com/educacion/cual-es-la-diferencia-entre-criptomonedas-token-y-monedas-digitales-184274#:~:text=%C2%BFQu%C3%A9%20es%20un%20token%3F,(no%20propia)%20para%20funcionar.

[20] E. Proposals, "EIP-20: ERC-20 Token Standard", Ethereum Improvement Proposals. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20.

[21] "Ethereum White Paper", GitHub. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[22] "What is Ethereum? - EthHub", Docs.ethhub.io. [Online]. Available: https://docs.ethhub.io/ethereum-basics/what-is-ethereum/

[23] "What is Ether (ETH)? - EthHub", Docs.ethhub.io. [Online]. Available: https://docs.ethhub.io/ethereum-basics/what-is-ether/.

[24] "Proof of work and mining", ethereum.org. [Online]. Available: https://ethereum.org/en/learn/#proof-of-work-and-mining.

[25] GoQuorum, "GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/

[26] GoQuorum, "Architecture - GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/en/latest/Concepts/Architecture/.

[27] "Raft - GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/en/latest/Concepts/Consensus/Raft/.

[28] "IBFT - GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/en/latest/Concepts/Consensus/IBFT/

[29]"Public and private transactions - GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/en/latest/Concepts/Privacy/PrivateAndPublic/

[30] "Private transaction lifecycle - GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/en/latest/Concepts/Privacy/PrivateTransactionLifecycle/.

[31] V. Muñoz Reyes, "Hyperledger Fabric - ¿Qué es Hyperledger?", Medium. [Online]. Available: https://medium.com/babel-go2chain/hyperledger-fabric-qu%C3%A9-es-hyperledger-en-espa%C3%B1ol-d13f8d144455

[31] V. Muñoz Reyes, "Introducción a Hyperledger Fabric", Medium. [Online]. Available: https://medium.com/babel-go2chain/introducci%C3%B3n-a-hyperledger-fabric-en-espa%C3%B1ol-9cba1c0cf73b

[32] "Arquitectura de Hyperledger y consenso". [Online]. Available: https://andresarayafalcone.cl/2018/04/06/arquitectura-de-hyperledger-y-consenso/

[33] "Enterprise Blockchain Platform | Corda Platform and Services by R3", R3. [Online]. Available: https://www.r3.com/corda-platform/

[34] "Corda, la plataforma de R3 que se asemeja muy poco a la blockchain", CriptoNoticias - Bitcoin, blockchains y criptomonedas. [Online]. Available: https://www.criptonoticias.com/tecnologia/corda-la-plataforma-de-r3-que-se-asemeja-muy-poco-a-la-blockchain/

[35] "Instantly Move Money to All Corners of the World | Ripple", Ripple. [Online]. Available: https://ripple.com/

[36] "One Pay: transferencias internacionales al instante - Banco Santander", Bancosantander.es. [Online]. Available: https://www.bancosantander.es/es/particulares/banca-online/one-pay

[37] "Santander expande One Pay FX a 19 países en asociación con Ripple, Chile entre ellos - BeInCrypto", BeInCrypto. [Online]. Available: https://es.beincrypto.com/santander-expande-one-pay-fx-19-paises-asociacion-con-ripple-xrp/.

[38] "Banco de España - Sistemas de pago - Sistemas de pago al por menor - El SNCE", Bde.es. [Online]. Available: https://www.bde.es/bde/es/areas/sispago/Sistemas_de_pago/El_SNCE/El_SNCE.html

[39] “Los principales bancos españoles desarrollan una prueba de concepto coordinada por Iberpay para habilitar pagos en redes blockchain”. Available: https://www.iberpay.es/Documentos/NOVEDADES_NOTAS_INFORMATIVAS/IBERPAY_LOS_BANCOS_ESPANOLES_DESARROLLAN_UNA_POC_PARA_HABILITAR_PAGOS_EN_REDES_BLOCKCHAIN.PDF.

[40] "Qué es una Stablecoin", Bit2me, 2020. [Online]. Available: https://academy.bit2me.com/que-es-stablecoin/

[41] "¿Qué son las ‘stablecoins’ y para qué sirven?", BBVA NOTICIAS. [Online]. Available: https://www.bbva.com/es/que-son-las-stablecoins-y-para-que-sirven/.

[42] "El sistema financiero del futuro - ¿Quiénes se benefician de las CBDC?", Cointelegraph. [Online]. Available: https://es.cointelegraph.com/news/the-financial-system-of-the-future-who-benefits-from-cbdcs.

[43]"El Banco Popular de China “avanza sin problemas” con el yuan digital", Cointelegraph. [Online]. Available: https://es.cointelegraph.com/news/peoples-bank-of-china-progressing-smoothly-with-digital-yuan.

[44]S. Phillip Webb, "Spring Boot Reference Documentation", Docs.spring.io. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.

[45] J. Porter, "Maven – Introduction to the POM", Maven.apache.org. [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html.

[46] "web3j/homebrew-web3j", GitHub. [Online]. Available: https://github.com/web3j/homebrew-web3j.

[47] "web3j/web3j-quorum", GitHub. [Online]. Available: https://github.com/web3j/web3j-quorum.

[48] "MetaMask: El puente entre Ethereum y tu navegador", CriptoNoticias. [Online]. Available: https://www.criptonoticias.com/aplicaciones/metamask-puente-ethereum-navegador/

[49] "Cakeshop - GoQuorum", Docs.goquorum.consensys.net. [Online]. Available: https://docs.goquorum.consensys.net/en/latest/Concepts/Cakeshop/

[50] K. Pohl, Requirements engineering: Fundamentals, principles, and techniques. Heidelberg: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2010.

Anexo I. Instalación

Despliegue de la red Quorum

Para poder crear una red Quorum en nuestro ordenador personal utilizaremos una herramienta de línea de comandos llamada Quorum Wizard y que es parte del ecosistema Quorum.

Quorum Wizard ha sido desarrollado en Javascript y diseñado para poder ejecutarse como un módulo NPM. Una vez instalado Node.js, ejecutamos el siguiente comando en la shell.

```
npm install -g quorum-wizard
```

Si la instalación se lleva a cabo con éxito, todo está listo para ejecutar Quorum Wizard en nuestro ordenador. Únicamente será necesario ejecutar el comando:

```
quorum-wizard
```

Y aparecerá en pantalla el siguiente menú:

```
davidlopezcano@MacBook-Pro-de-David ~ % quorum-wizard
?
Welcome to Quorum Wizard!

This tool allows you to easily create bash, docker, and kubernetes files to start up a quorum network.
You can control consensus, privacy, network details and more for a customized setup.
Additionally you can choose to deploy our chain explorer, Cakeshop, to easily view and monitor your network.

We have 3 options to help you start exploring Quorum:

  1.  Quickstart - our 1 click option to create a 3 node raft network with tessera and cakeshop

  2.  Simple Network - using pregenerated keys from quorum 7nodes example,
      this option allows you to choose the number of nodes (7 max), consensus mechanism, transaction manager, and the opti
on to deploy cakeshop

  3.  Custom Network - In addition to the options available in #2, this selection allows for further customization of your
 network.
      Choose to generate keys, customize ports for both bash and docker, or change the network id

Quorum Wizard will generate your startup files and everything required to bring up your network.
All you need to do is go to the specified location and run ./start.sh

 (Use arrow keys)
❯ Quickstart (3-node raft network with tessera and cakeshop)
  Simple Network
  Custom Network
  Exit
```

Figura A1.1: Menú tras ejecutar quorum-wizard

Este menú ofrece al usuario tres opciones de inicialización de la red, dos opciones predeterminadas y una opción personalizable, en nuestro caso seleccionaremos esta última.

Esta opción permite al usuario seleccionar los siguientes parámetros:

- Algoritmo de consenso: Raft o IBFT.
- Seleccionar el número de nodos: se recomienda seleccionar un mínimo de 3 nodos. Una red de N nodos seguirá funcionando correctamente siempre que se cumpla N=2F+1, siendo F el número de nodos que han dejado de funcionar.
- Versión de Quorum: actualmente permite elegir entre la versión 2.5.0 y 2.6.0
- Versión de Tessera: es posible seleccionar distintas versiones de Tessera. En el caso de que no se requiera realizar transacciones privadas, se omitirá este paso.
- Instalación de Cakeshop: la instalación de este componente para explorar la red blockchain es voluntaria.
- Generación de claves para la red: este paso puede obviarse y generar las claves de forma manual, pero advierten que no se recomienda para un uso en producción de la red.
- Nombre de la red.
- Personalización de los puertos de los nodos.

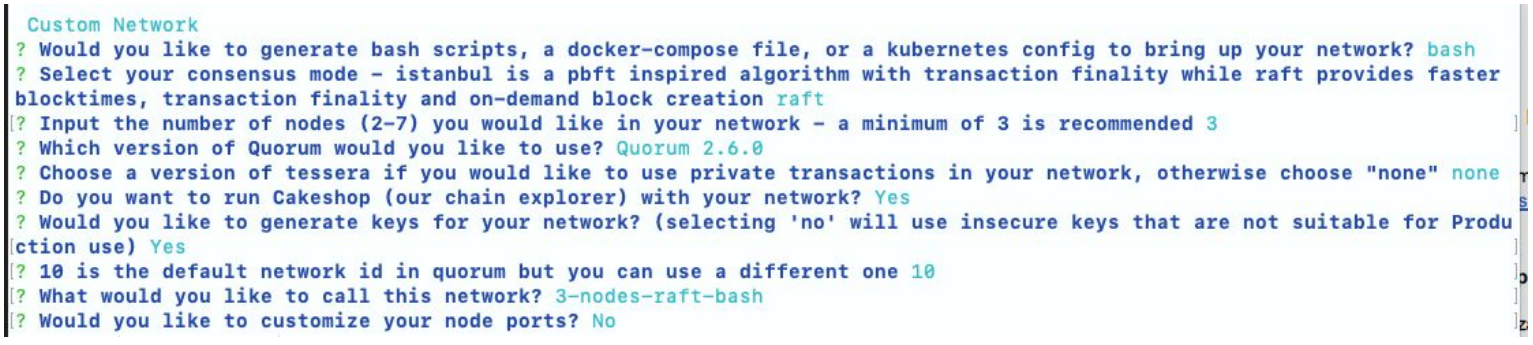

Figura A1.2: Quorum wizard tras seleccionar todos los parámetros

Tras realizar estos sencillos pasos la red Quorum estará creada. Por defecto Quorum Wizard crea los ficheros de la red en el directorio */network* de nuestro equipo. El nombre de la carpeta será el mismo que el nombre de la red que hemos indicado en el proceso de creación.

Figura A1.3: Directorios y ficheros generados

Para desplegar los nodos Quorum, junto a los Tessera y los Cakeshop es necesario acceder al directorio de la red y ejecutar el script *start.sh*.

```
cd network/3-nodes-raft-tessera-bash
./start.sh
```

```
davidlopezcano@MacBook-Pro-de-David ~ % cd network
davidlopezcano@MacBook-Pro-de-David network % cd 3-nodes-raft-bash
davidlopezcano@MacBook-Pro-de-David 3-nodes-raft-bash % ./start.sh

Starting Quorum network...

Starting Quorum nodes
Starting Cakeshop
Waiting until Cakeshop is running...
Cakeshop is not yet listening on http
Waiting until Cakeshop is running...
Cakeshop is not yet listening on http
Waiting until Cakeshop is running...
Cakeshop is not yet listening on http
Waiting until Cakeshop is running...
Cakeshop is not yet listening on http
Waiting until Cakeshop is running...
Cakeshop started at http://localhost:8999
Successfully started Quorum network.
```

Figura A1.4: Ejecución de start.sh

Despliegue de Smart Contract

Una vez que la red está montada y todos los elementos configurados podemos proceder a desplegar el Smart Contract que contiene el código de un token ERC20.

Aunque una blockchain esté formada por varios nodos, un contrato inteligente solo podrá ser desplegado desde una cuenta que esté asociada a un único nodo. En el fondo desplegar un contrato inteligente en la red blockchain no es más que una transacción que es emitida por una cuenta de un nodo, y que como cualquier transacción realizada en blockchain deberá de ser validada por los demás nodos y almacenada en un bloque.

Para poder iniciar este proceso debemos de acceder a la web de Remix (https://remix.ethereum.org). En primer lugar es necesario activar el plugin de Quorum, para ello en la barra lateral seleccionamos *Plugin Manager,* escribimos Quorum en el buscador y una vez que aparezca el módulo *Quorum Network* pulsamos en *Activate*. Una vez activado el logo de Quorum aparecerá en la barra lateral.

Figura A1.5: Plugin Quorum Network

Después será necesario importar el código del Smart Contract a Remix. Para ello debemos acceder al explorador de archivos y pulsar sobre el botón para importar un archivo.

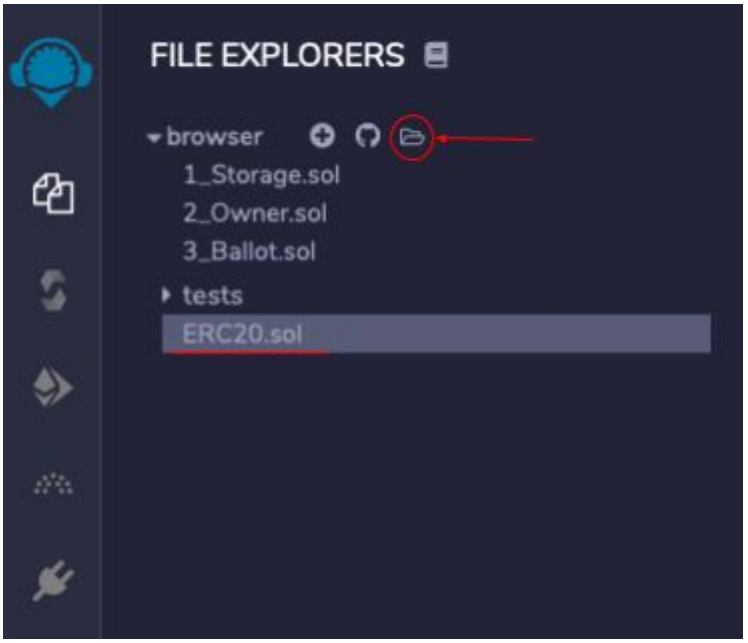

Figura A1:6: Importar Smart Contract

Este Smart Contract implementa una interfaz y su aspecto es el siguiente:

```
pragma solidity ^0.4.26;

contract EIP20Interface {
    /* This is a slight change to the ERC20 base standard.
    function totalSupply() constant returns (uint256 supply);
    is replaced with:
    uint256 public totalSupply;
    This automatically creates a getter function for the totalSupply.
    This is moved to the base contract since public getter functions are not
    currently recognised as an implementation of the matching abstract
    function by the compiler.
    */
    /// total amount of tokens
    uint256 public totalSupply;

    /// @param _owner The address from which the balance will be retrieved
    /// @return The balance
    function balanceOf(address _owner) public view returns (uint256 balance);

    /// @notice send `_value` token to `_to` from `msg.sender`
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transfer(address _to, uint256 _value) public returns (bool success);

    /// @notice send `_value` token to `_to` from `_from` on the condition it is approved by `_from`
    /// @param _from The address of the sender
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);

    /// @notice `msg.sender` approves `_spender` to spend `_value` tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @param _value The amount of tokens to be approved for transfer
    /// @return Whether the approval was successful or not
    function approve(address _spender, uint256 _value) public returns (bool success);

    /// @param _owner The address of the account owning tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @return Amount of remaining tokens allowed to spent
    function allowance(address _owner, address _spender) public view returns (uint256 remaining);

    // solhint-disable-next-line no-simple-event-func-name
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

Figura A1.6: Interfaz implementada por el Smart Contract

Más tarde debemos de compilar el Smart Contract, con la misma versión de compilador que la que se indica en la cabecera del contrato inteligente, en este caso utilizaremos la versión 0.4.26 de Solidity.

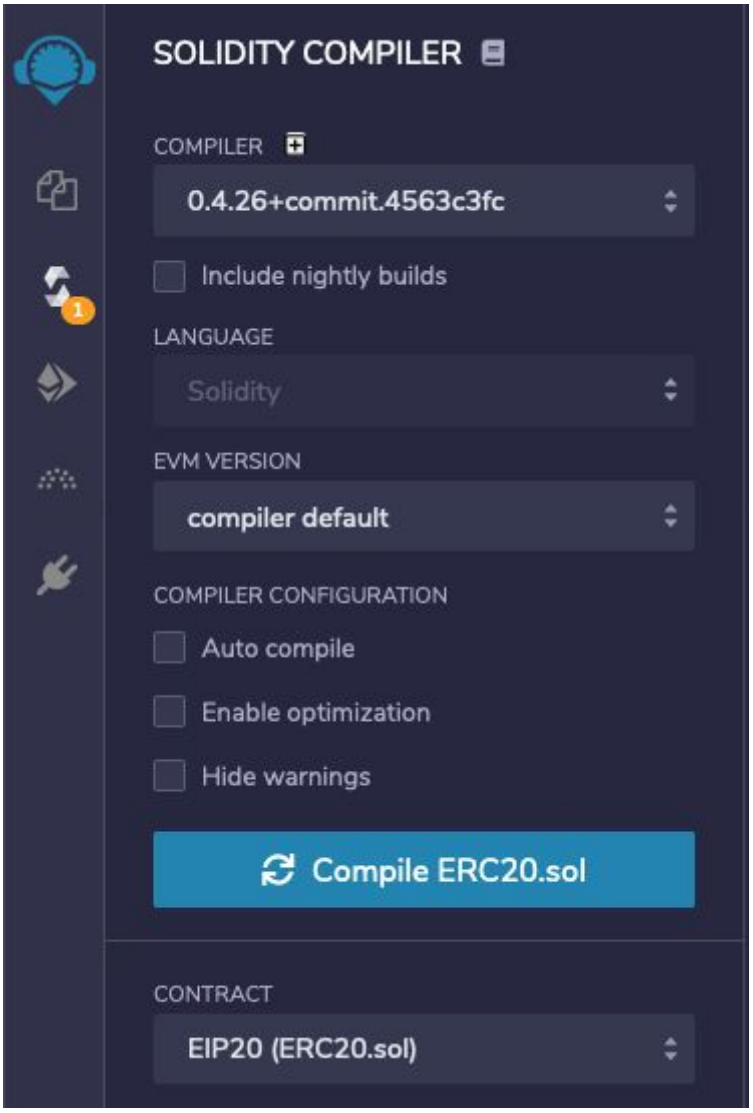

Figura A1.7: Compilador de Smart Contract

El último paso será desplegar el contrato compilado en la red. Para ello accedemos al plugin de Quorum que previamente hemos activado, e introducimos la ruta y el puerto de nuestro nodo, en este caso http://localhost:22000/ y la cuenta propietaria del contrato, 0xc077bf171427abe5c3c62527cac4f8e29ba5823e. También debemos seleccionar los valores iniciales del token que son:

- La cantidad de tokens inicial en el sistema
- El nombre del token
- Los decimales del token
- El símbolo del token

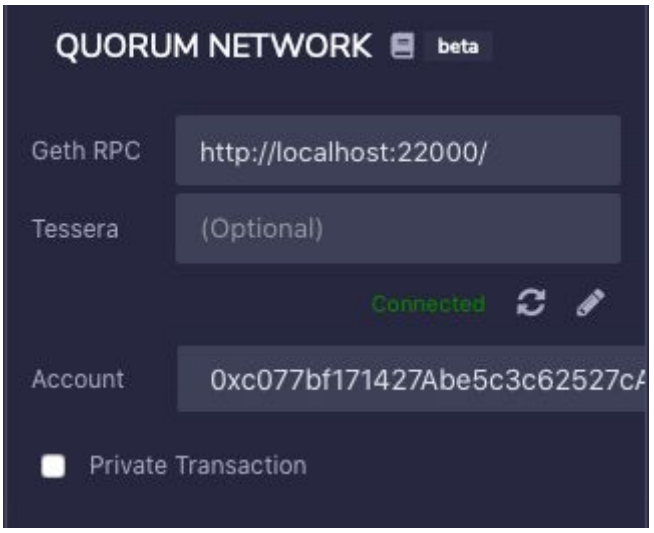

Figura A1.8: Datos de la red para el despliegue del contrato

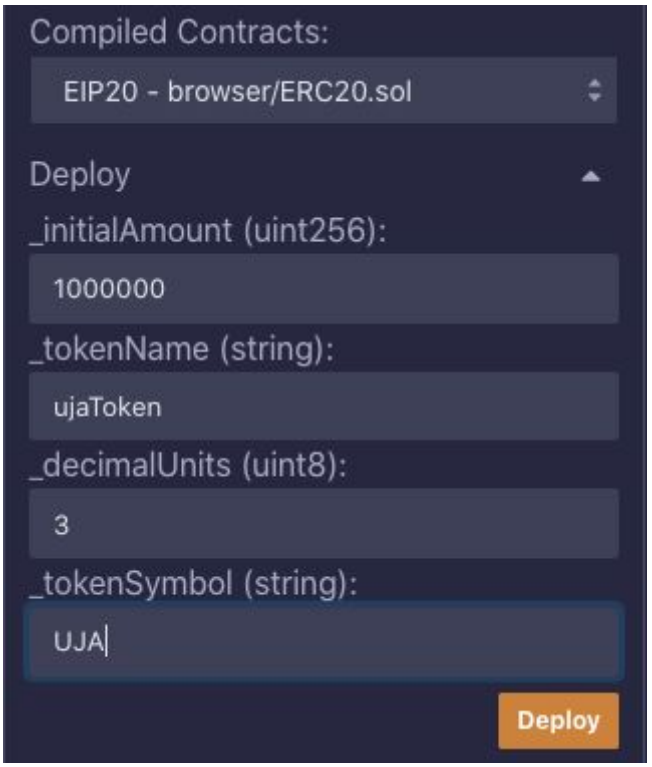

Figura A1.9: Datos necesarios para inicializar el token

Tras pulsar el botón *deploy,* el contrato se desplegará en la red y quedará almacenado en la blockchain. Remix nos devolverá la *address* donde ha quedado registrado y nos proporciona una interfaz simple para poder ejecutar las funciones de este contrato.

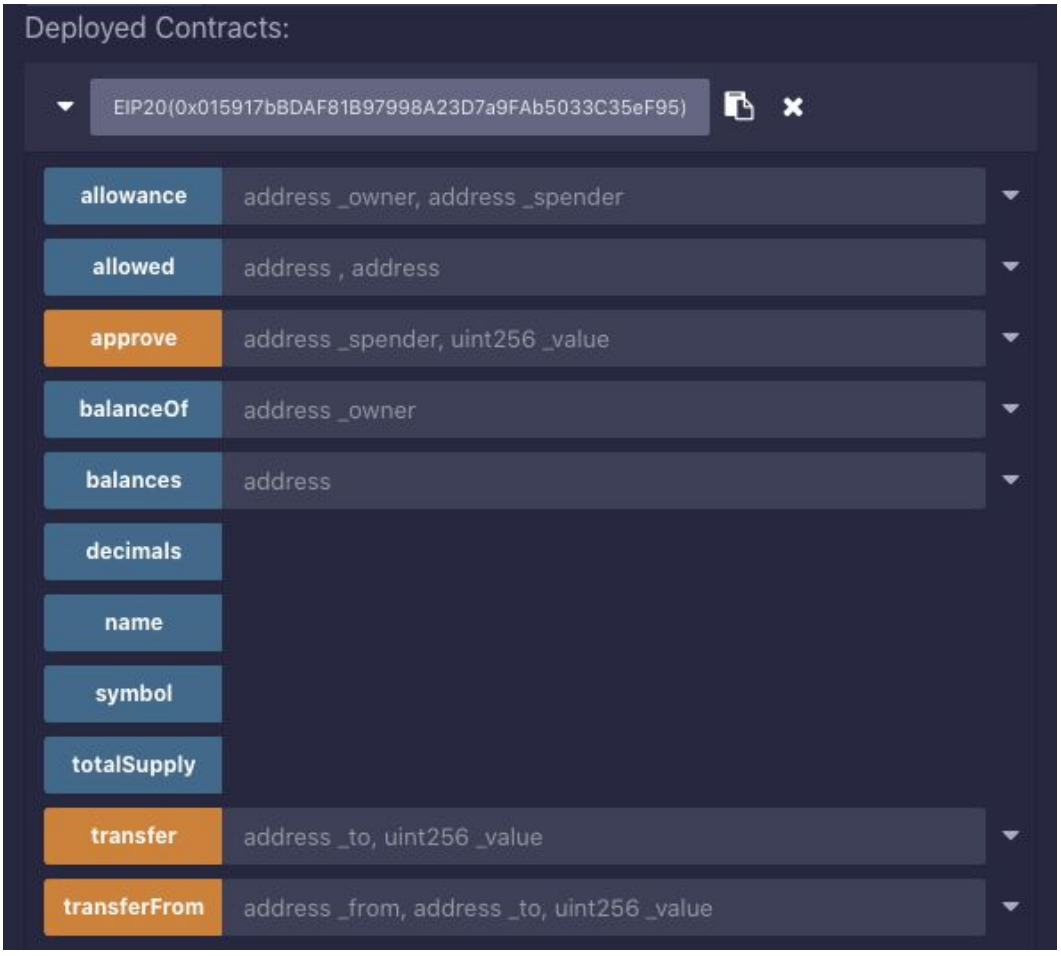

Figura A1.9: Smart Contract desplegado con Remix

Configuración de Spring Boot

Una vez que la red está operativa y el Smart Contract desplegado en ella ha llegado el momento de suministrar a Spring Boot la información necesaria para poder conectarse mediante Web3J a la blockchain. Esta información debemos de indicarle en el fichero application.yml del proyecto Spring.

Debemos de indicar los datos de *wallets* de la blockchain, por un lado los datos de la cuenta que ha desplegado el Smart Contract en la red y que actuará de administrador (*mainAccount*), dado que todos los tokens de la red han sido almacenados en esta cuenta y será la que envíe estos tokens a las distintas cuentas cuando estas realicen una recarga. Por otro lado, simularemos que una cuenta (*logginAccount*) se ha registrado en el sistema indicando todos sus datos en el archivo application.yml.

Los datos que hay que suministrar en ambos casos son: el nodo donde se encuentra la cuenta, la dirección propia de la cuenta, la ruta local donde está almacenado el fichero de claves de la cuenta, el nombre del fichero de claves, y la contraseña de la cuenta. Además en el caso de la *mainAccount* debemos de aportar la dirección donde se ha desplegado el contrato inteligente.

```
mainaccount:
  nodeEndpoint: http://localhost:22000
  accountAddress: "0xc077bf171427abe5c3c62527cac4f8e29ba5823e"
  walletPath: "/Users/davidlopezcano/network/3-nodes-raft-bash/qdata/dd1/keystore/"
  walletName: "UTC--2020-08-29T16-31-58.798939000Z--c077bf171427abe5c3c62527cac4f8e29ba5823e"
  passWallet: ""
  smartContractAddress: "0x015917bBDAF81B97998A23D7a9FAb5033C35eF95"

loggingaccount:
  nodeEndpoint: "http://localhost:22000"
  accountAddress: "0xc8f1b33527d5fe2ab33fa1cba703c18ba5515147"
  walletPath: "/Users/davidlopezcano/network/3-nodes-raft-bash/qdata/dd1/keystore/"
  walletName: "UTC--2020-08-30T07-06-14.709760000Z--c8f1b33527d5fe2ab33fa1cba703c18ba5515147"
  passWallet: ""
```

Figura A1.10: Datos de la red Quorum en Spring Boot

Anexo II. Manual de usuario

Ejecución de la aplicación

En el primer anexo se han explicado los pasos necesarios para crear una red Quorum y desplegar un Smart Contract en la blockchain utilizando Remix. En primer lugar es necesario tener levantados los nodos de la blockchain. Como se indicó en el apartado anterior, debemos de ejecutar en una terminal el comando:

```
cd network/3-nodes-raft-tessera-bash
./start.sh
```

Seguidamente debemos de desplegar la API, para ello debemos de importar el proyecto en el IDE Eclipse y ejecutar el proyecto.

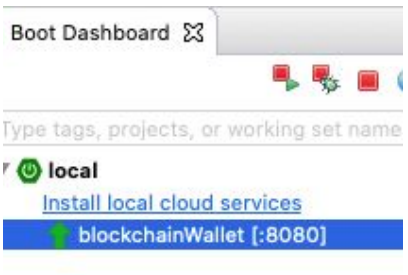

Figura A2.1: Ejecución del proyecto Spring

Para poder utilizar la API utilizaremos Swagger, que se ha desplegado junto con la aplicación en la url: http://localhost:8080/swagger-ui.html

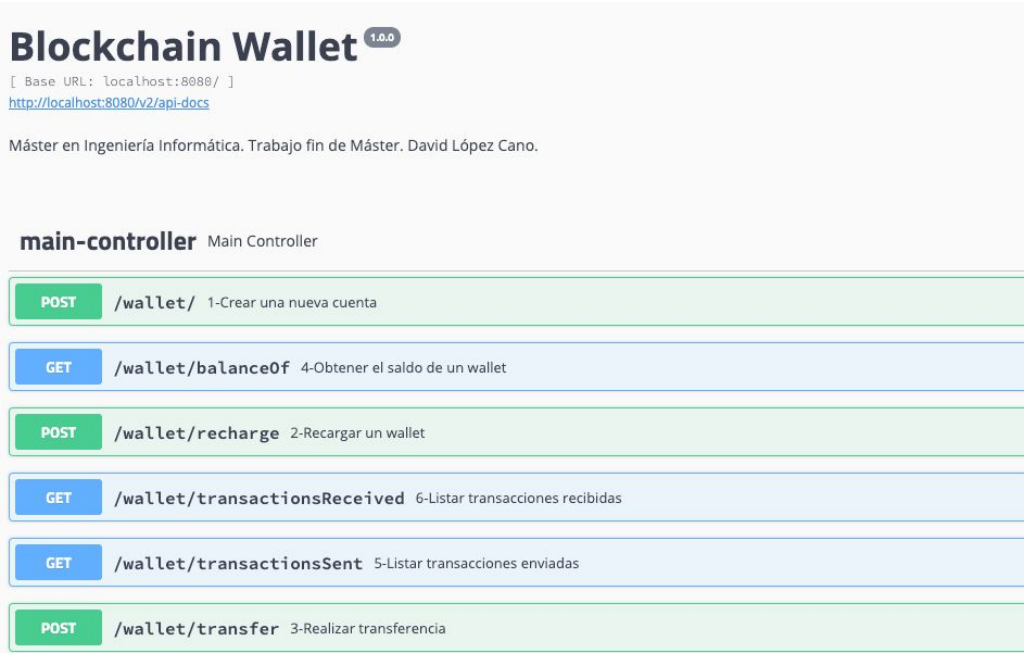

Figura A2.2: Swagger UI

Tras pulsar en main-controller aparece un desplegable con todos los *endpoints* del controlador. Para poder realizar la llamada a cada *endpoint* debemos de pulsar sobre el método en cuestión y aparecerá un botón con el título *Try it out* que debemos de pulsar y rellenar los parámetros de entrada de la función.

Crear wallet

El primero de los casos de uso de nuestra aplicación es la creación de un wallet, para ello realizaremos una petición **POST** al endpoint **/wallet/.** En este caso no es necesario indicar ningún parámetro de entrada. Si el wallet se ha creado con éxito devolverá toda la información del mismo, en caso contrario se lanzará una excepción y la API devolverá un error 500.

Si queremos utilizar un wallet creado como wallet logueado en la aplicación debemos introducir los datos suministrados por Swagger en el fichero application.yml de Spring Boot y volver a lanzar la aplicación Java.

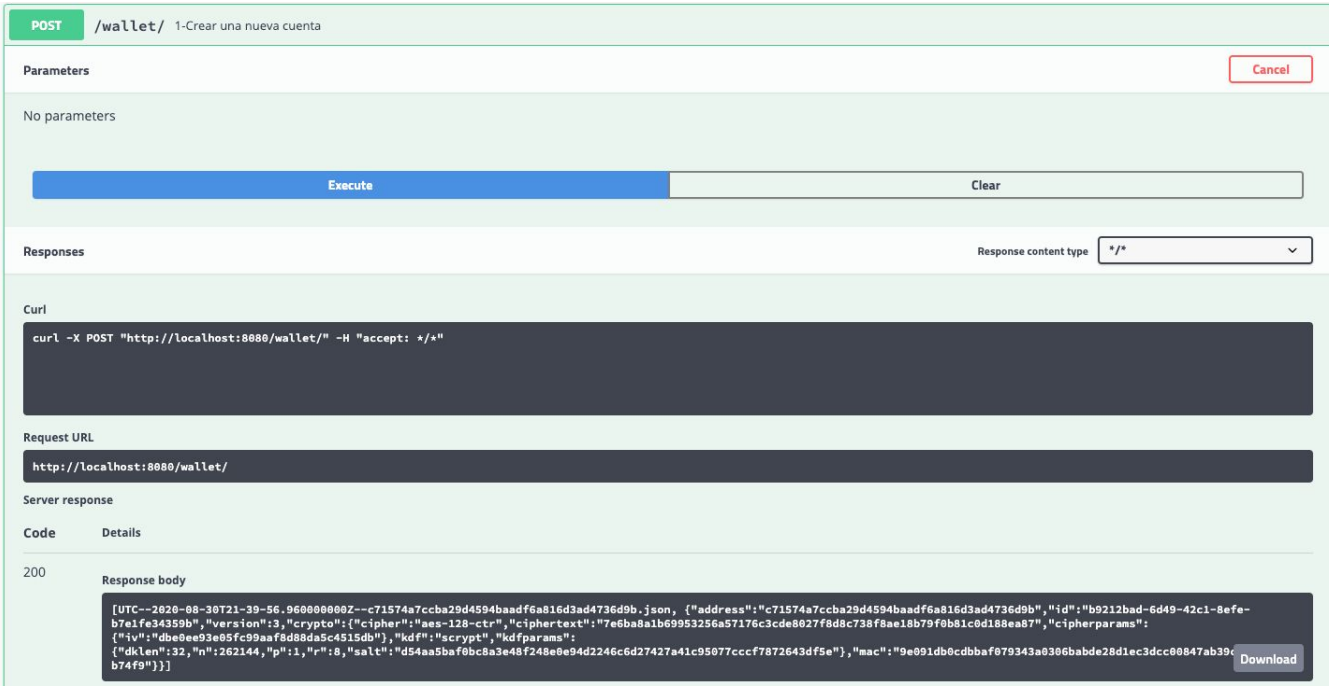

Figura A2.3: Crear wallet

Recargar un wallet

Con este endpoint simularemos la tokenización de dinero fiat a tokens de nuestro sistema. Únicamente será necesario introducir la cantidad de euros que se va a recargar. El wallet recargado será el indicado en el *logginAccount* del fichero application.yml. En caso de éxito la API devolverá la traza completa de la transacción en Quorum, en caso contrario se lanzará una excepción y la API devolverá un error 500.

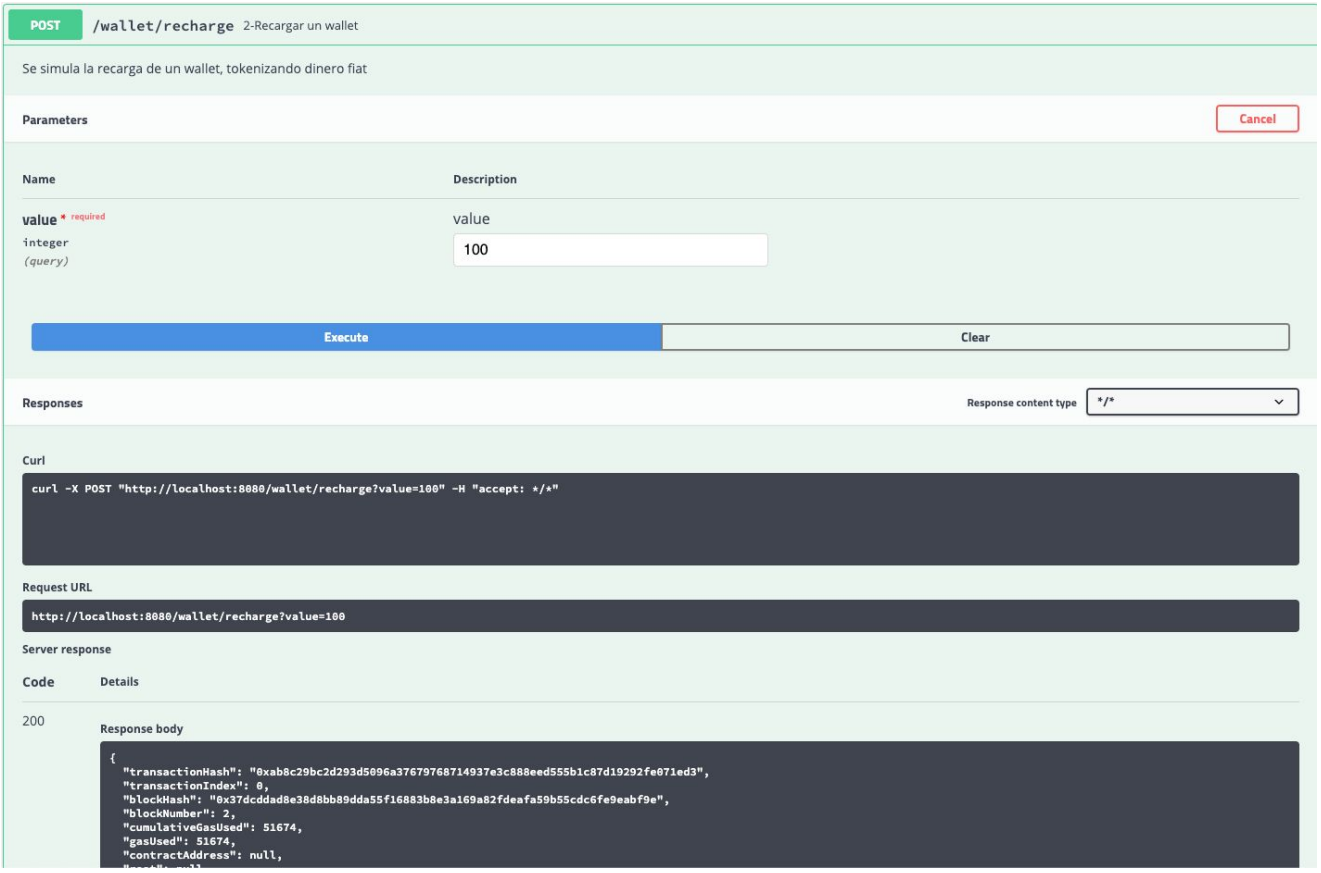

Figura A2.4: Recargar wallet

Consultar saldo de wallet

Para consultar el saldo de un wallet se realizará una petición **GET** a la ruta **/wallet/balanceOf.** Esta petición no necesita parámetros de entrada debido a que se consultará el saldo del wallet definido en la logginAccount. En caso de éxito la API devolverá el saldo del wallet, en caso contrario se lanzará una excepción y la API devolverá un error 500.

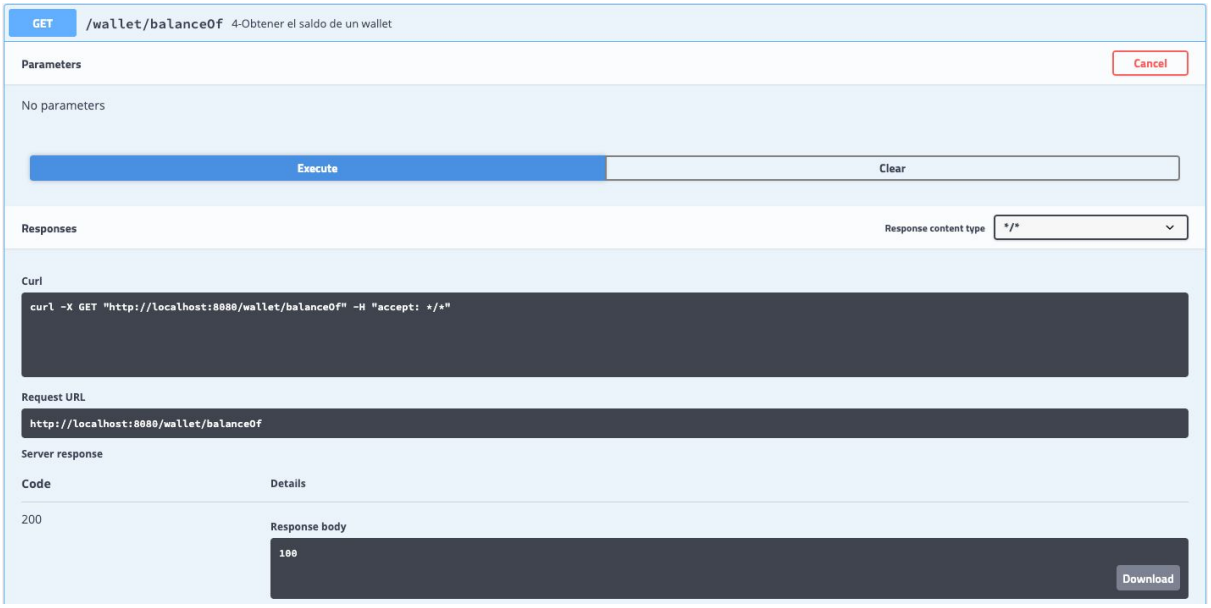

Figura A2.5: Consultar saldo de un wallet

Realizar transacción

Para enviar saldo de un wallet a otro wallet se realizará una petición **POST** al endpoint **/wallet/transfer**. Los parámetros de entrada que debemos de añadir a la petición son el wallet receptor y la cantidad de saldo a enviar. Como en los endpoints anteriores, el wallet emisor será el indicado en el *logginAccount* del fichero application.yml. En caso de éxito la API devolverá la traza completa de la transacción en Quorum, en caso contrario se lanzará una excepción y la API devolverá un error 500.

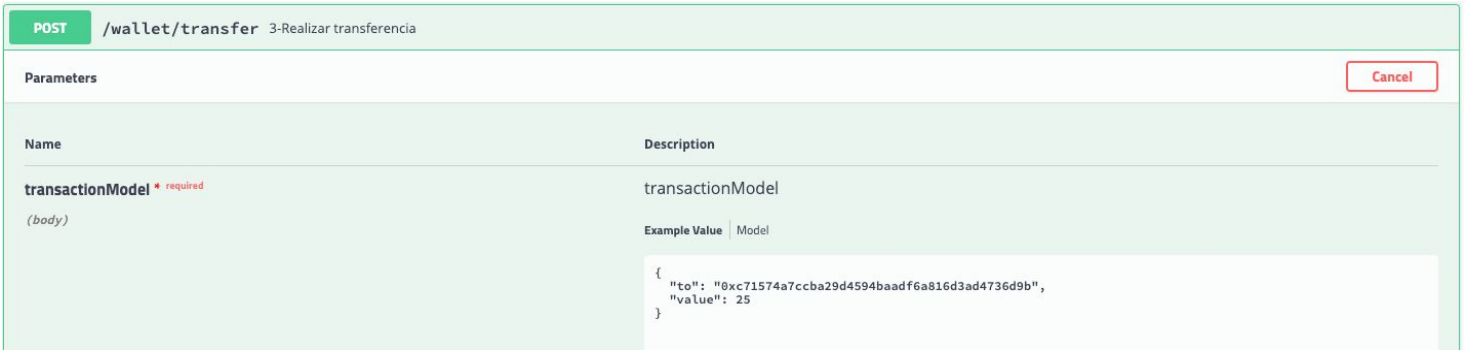

Figura A2.6: Realizar transacción. Parámetros de entrada.

Figura A2.7: Realizar transacción. Respuesta con éxito

Consultar transacciones enviadas

Si realizamos una petición **GET** al *endpoint* **/wallet/transactionsSent** consultaremos todas las transacciones que han sido enviadas por el wallet indicado en el *logginAccount* del fichero application.yml. Esta petición no requiere de parámetros de entrada. En caso de éxito la API devolverá una lista que contiene el emisor, receptor y la cantidad enviada en cada transacción, en caso contrario se lanzará una excepción y la API devolverá un error 500.

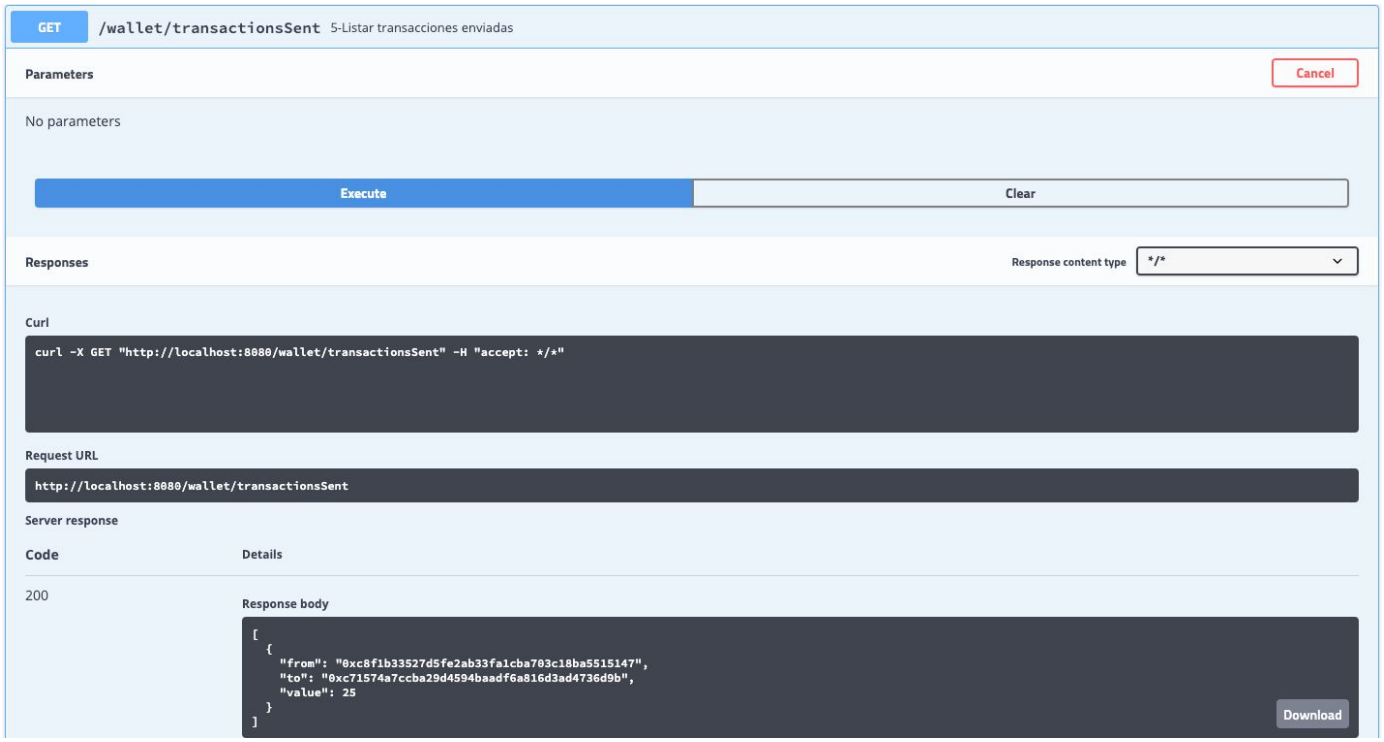

Figura A2.7: Listado de transacciones enviados

Consultar transacciones recibidas

Si realizamos una petición **GET** al *endpoint* **/wallet/transactionsReceived** consultaremos todas las transacciones que han sido recibidas en el wallet indicado en el *logginAccount* del fichero application.yml. Esta petición no requiere de parámetros de entrada. En caso de éxito la API devolverá una lista que contiene el emisor, receptor y la cantidad recibida en cada transacción, en caso contrario se lanzará una excepción y la API devolverá un error 500.

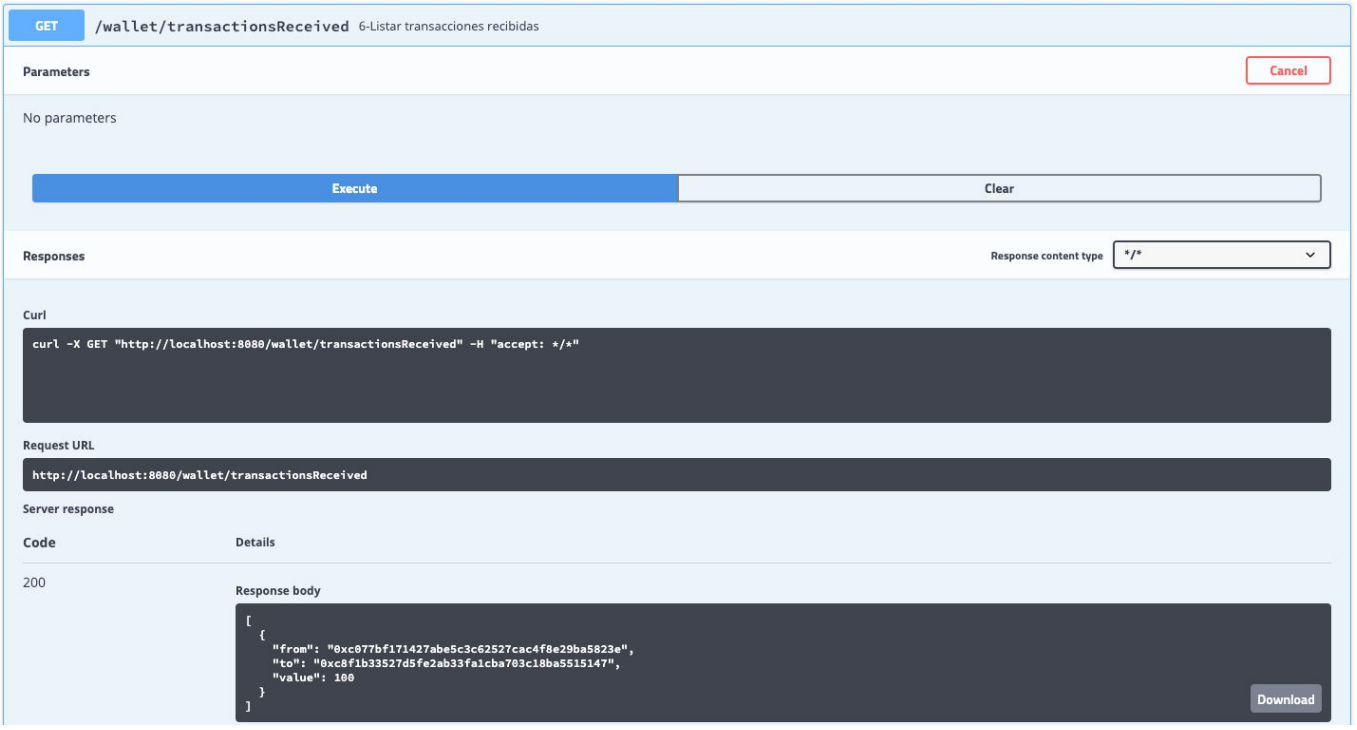

Figura A2.8: Listado de transacciones recibidas

Anexo III. Descripción contenidos suministrados

El archivo que se entrega contiene:

- **codigo_DavidLopezCano**: contiene el código fuente del proyecto.
- **memoria_DavidLopezCano**: memoria del TFG en formato PDF.
- **diagramas_DavidLopezCano**: diagramas de secuencia y de clases en formato jpg para una mejor consulta.